L4Re – L4 Runtime Environment

Generated by Doxygen 1.8.8

Contents

1 Overview **1**

2 Introduction **3**

2.1 Fiasco.OC . . . 3

2.1.1 Communication . . . 3

2.1.2 Kernel Objects . . . 3

2.2 L4Re System Structure . . . 4

2.3 L4 Runtime Environment (L4Re) . . . 5

3 Tutorial **7**

3.1 Customizations . . . 8

4 Programming for L4Re **9**

4.1 Capabilities and Naming . . . 9

4.2 Initial Environment and Application Bootstrapping . . . 10

4.2.1 Configuring an application before startup . . . 11

4.2.2 Connecting clients and servers . . . 11

4.3 Memory management - Data Spaces and the Region Map . . . 12

4.3.1 User-level paging . . . 12

4.3.1.1 Data spaces . . . 12

4.3.1.2 Virtual Memory Handling . . . 12

4.3.1.3 Memory Allocation . . . 12

4.4 Program Input and Output . . . 12

4.5 Initial Memory Allocator and Factory . . . 13

4.6 Application and Server Building Blocks . . . 13

4.6.1 Creating Additional Application Threads . . . 13

4.6.2 Providing a Service . . . 13

4.7 Pthread Support . . . 14

4.8 Interface Definition Language . . . 14

4.8.1 Parameter types for RPC . . . 15

4.8.2 RPC Return Types . . . 16

4.8.3 RPC Method Declaration . . . 16

4.9 L4Re Servers . . . 17
4.9.1 Moe, the Root-Task . . . 18
4.9.1.1 Memory Allocator, Generic Factory . . . 18
4.9.1.2 Name-Space Provider . . . 19
4.9.1.3 Log Subsystem . . . 19
4.9.1.4 Scheduler Subsystem . . . 19
4.9.1.5 Command-Line Options . . . 19
4.9.2 Ned, the Init Process . . . 20
4.9.2.1 Lua Bindings for L4Re . . . 20
4.9.3 lo, the lo Server . . . 22
4.9.4 Mag, the GUI Multiplexer . . . 24
4.10 L4Re Build System . . . 26
4.10.1 Building L4Re . . . 26
4.10.2 Writing BID Make Files . . . 27
4.10.3 prog.mk - Application Role . . . 28
4.10.4 include.mk - Header File Role . . . 29
4.10.5 test.mk - Test Application Role . . . 30

5 Deprecated List **33**

6 Module Index **35**
6.1 Modules . . . 35

7 Namespace Index **39**
7.1 Namespace List . . . 39

8 Hierarchical Index **41**
8.1 Class Hierarchy . . . 41

9 Data Structure Index **51**
9.1 Data Structures . . . 51

10 Module Documentation **63**
10.1 ARM Virtual Registers (UTCB) . . . 63
10.1.1 Detailed Description . . . 63
10.2 Atomic Instructions . . . 64
10.2.1 Detailed Description . . . 65
10.2.2 Function Documentation . . . 65
10.2.2.1 l4util_add8 . . . 65
10.2.2.2 l4util_add8_res . . . 66
10.2.2.3 l4util_atomic_add . . . 66
10.2.2.4 l4util_atomic_inc . . . 66

10.2.2.5 l4util_cmpxchg . . . 66
10.2.2.6 l4util_cmpxchg16 . . . 67
10.2.2.7 l4util_cmpxchg32 . . . 67
10.2.2.8 l4util_cmpxchg64 . . . 68
10.2.2.9 l4util_cmpxchg8 . . . 69
10.2.2.10 l4util_inc8 . . . 69
10.2.2.11 l4util_inc8_res . . . 69
10.2.2.12 l4util_xchg . . . 69
10.2.2.13 l4util_xchg16 . . . 70
10.2.2.14 l4util_xchg32 . . . 70
10.2.2.15 l4util_xchg8 . . . 70
10.3 Auxiliary data . . . 71
10.3.1 Detailed Description . . . 71
10.4 Base API . . . 72
10.4.1 Detailed Description . . . 74
10.5 Basic Macros . . . 75
10.5.1 Detailed Description . . . 76
10.5.2 Macro Definition Documentation . . . 76
10.5.2.1 L4_ALWAYS_INLINE . . . 76
10.5.2.2 L4_HIDDEN . . . 76
10.5.2.3 L4_NOTHROW . . . 76
10.6 Bit Manipulation . . . 77
10.6.1 Detailed Description . . . 77
10.6.2 Function Documentation . . . 77
10.6.2.1 l4util_bsf . . . 77
10.6.2.2 l4util_bsr . . . 78
10.6.2.3 l4util_btc . . . 78
10.6.2.4 l4util_btr . . . 79
10.6.2.5 l4util_bts . . . 79
10.6.2.6 l4util_clear_bit . . . 80
10.6.2.7 l4util_complement_bit . . . 80
10.6.2.8 l4util_find_first_set_bit . . . 80
10.6.2.9 l4util_find_first_zero_bit . . . 81
10.6.2.10 l4util_next_power2 . . . 81
10.6.2.11 l4util_set_bit . . . 81
10.6.2.12 l4util_test_bit . . . 82
10.7 Bitmap graphics and fonts . . . 83
10.7.1 Detailed Description . . . 83
10.8 Buffer Registers (BRs) . . . 84
10.8.1 Detailed Description . . . 84

10.8.2 Enumeration Type Documentation . . . 84
10.8.2.1 l4_buffer_desc_consts_t . . . 84
10.9 C++ Exceptions . . . 85
10.9.1 Detailed Description . . . 85
10.10 C++ IPC Interface Definition. . . . 86
10.10.1 Detailed Description . . . 86
10.11 CPU related functions . . . 87
10.11.1 Detailed Description . . . 87
10.11.2 Function Documentation . . . 87
10.11.2.1 l4util_cpu_capabilities . . . 87
10.11.2.2 l4util_cpu_capabilities_nocheck . . . 88
10.11.2.3 l4util_cpu_has_cpuid . . . 88
10.12 Cache Consistency . . . 90
10.12.1 Detailed Description . . . 90
10.12.2 Function Documentation . . . 90
10.12.2.1 l4_cache_clean_data . . . 90
10.12.2.2 l4_cache_coherent . . . 91
10.12.2.3 l4_cache_dma_coherent . . . 92
10.12.2.4 l4_cache_flush_data . . . 92
10.12.2.5 l4_cache_inv_data . . . 92
10.13 Capabilities . . . 93
10.13.1 Detailed Description . . . 93
10.13.2 Enumeration Type Documentation . . . 94
10.13.2.1 l4_cap_consts_t . . . 94
10.13.2.2 l4_default_caps_t . . . 94
10.13.3 Function Documentation . . . 94
10.13.3.1 l4_capability_equal . . . 94
10.13.3.2 l4_is_invalid_cap . . . 95
10.13.3.3 l4_is_valid_cap . . . 95
10.14 Capability allocator . . . 97
10.14.1 Detailed Description . . . 97
10.14.2 Function Documentation . . . 97
10.14.2.1 l4re_util_cap_last . . . 97
10.15 Chunks . . . 98
10.15.1 Detailed Description . . . 98
10.15.2 Function Documentation . . . 98
10.15.2.1 l4shmc_add_chunk . . . 98
10.15.2.2 l4shmc_chunk_capacity . . . 99
10.15.2.3 l4shmc_chunk_ptr . . . 99
10.15.2.4 l4shmc_chunk_signal . . . 99

10.15.2.5 l4shmc_get_chunk . . . 100
10.15.2.6 l4shmc_get_chunk_to . . . 100
10.15.2.7 l4shmc_iterate_chunk . . . 100
10.16 Client/Server IPC Framework . . . 102
10.16.1 Detailed Description . . . 102
10.17 Comfortable Command Line Parsing . . . 103
10.17.1 Detailed Description . . . 103
10.17.2 Function Documentation . . . 103
10.17.2.1 parse_cmdline . . . 103
10.18 Console API . . . 105
10.18.1 Detailed Description . . . 105
10.19 Consumer . . . 106
10.19.1 Detailed Description . . . 106
10.19.2 Function Documentation . . . 106
10.19.2.1 l4shmc_chunk_consumed . . . 106
10.19.2.2 l4shmc_chunk_size . . . 106
10.19.2.3 l4shmc_enable_chunk . . . 107
10.19.2.4 l4shmc_is_chunk_ready . . . 107
10.19.2.5 l4shmc_wait_chunk . . . 107
10.19.2.6 l4shmc_wait_chunk_to . . . 108
10.19.2.7 l4shmc_wait_chunk_try . . . 109
10.20 Consumer . . . 110
10.20.1 Detailed Description . . . 110
10.20.2 Function Documentation . . . 110
10.20.2.1 l4shmc_enable_signal . . . 110
10.20.2.2 l4shmc_wait_any . . . 110
10.20.2.3 l4shmc_wait_any_to . . . 111
10.20.2.4 l4shmc_wait_any_try . . . 111
10.20.2.5 l4shmc_wait_signal . . . 111
10.20.2.6 l4shmc_wait_signal_to . . . 112
10.20.2.7 l4shmc_wait_signal_try . . . 113
10.21 DMA API for L4Re . . . 114
10.22 DMA Space Interface . . . 115
10.22.1 Detailed Description . . . 115
10.22.2 Typedef Documentation . . . 115
10.22.2.1 l4re_dma_space_t . . . 115
10.22.3 Function Documentation . . . 116
10.22.3.1 l4re_dma_space_map . . . 116
10.22.3.2 l4re_dma_space_unmap . . . 116
10.23 Dataspace interface . . . 117

10.23.1 Detailed Description . . . 117
10.23.2 Enumeration Type Documentation . . . 118
10.23.2.1 l4re_ds_map_flags . . . 118
10.23.3 Function Documentation . . . 118
10.23.3.1 l4re_ds_allocate . . . 118
10.23.3.2 l4re_ds_clear . . . 118
10.23.3.3 l4re_ds_copy_in . . . 118
10.23.3.4 l4re_ds_flags . . . 118
10.23.3.5 l4re_ds_info . . . 119
10.23.3.6 l4re_ds_phys . . . 119
10.23.3.7 l4re_ds_size . . . 119
10.24 Debug interface . . . 120
10.24.1 Detailed Description . . . 120
10.24.2 Function Documentation . . . 120
10.24.2.1 l4re_debug_obj_debug . . . 120
10.25 Debugging API . . . 121
10.25.1 Detailed Description . . . 121
10.26 EDID parsing functionality . . . 122
10.26.1 Detailed Description . . . 122
10.26.2 Enumeration Type Documentation . . . 122
10.26.2.1 Libedid_consts . . . 122
10.26.3 Function Documentation . . . 122
10.26.3.1 libedid_check_header . . . 122
10.26.3.2 libedid_checksum . . . 123
10.26.3.3 libedid_dump . . . 123
10.26.3.4 libedid_dump_standard_timings . . . 123
10.26.3.5 libedid_num_ext_blocks . . . 123
10.26.3.6 libedid_pnp_id . . . 123
10.26.3.7 libedid_prefered_resolution . . . 124
10.26.3.8 libedid_revision . . . 124
10.26.3.9 libedid_version . . . 124
10.27 ELF binary format . . . 125
10.27.1 Detailed Description . . . 136
10.27.2 Macro Definition Documentation . . . 136
10.27.2.1 DF_1_CONFALT . . . 136
10.27.2.2 DF_1_DIRECT . . . 136
10.27.2.3 DF_1_DISPRELDNE . . . 136
10.27.2.4 DF_1_DISPRELPND . . . 136
10.27.2.5 DF_1_ENDFILTEE . . . 136
10.27.2.6 DF_1_GLOBAL . . . 136

10.27.2.7 DF_1_GROUP . . . 136
10.27.2.8 DF_1_INTERPOSE . . . 136
10.27.2.9 DF_1_LOADFLTR . . . 137
10.27.2.10 DF_1_NODEFLIB . . . 137
10.27.2.11 DF_1_NODELETE . . . 137
10.27.2.12 DF_1_NODUMP . . . 137
10.27.2.13 DF_1_NOOPEN . . . 137
10.27.2.14 DF_1_NOW . . . 137
10.27.2.15 DF_1_ORIGIN . . . 137
10.27.2.16 DF_P1_GROUPPERM . . . 137
10.27.2.17 DF_P1_LAZYLOAD . . . 137
10.27.2.18 DT_HIPROC . . . 138
10.27.2.19 DT_LOPROC . . . 138
10.27.2.20 DT_NULL . . . 138
10.27.2.21 EI_CLASS . . . 138
10.27.2.22 EI_CLASS . . . 138
10.27.2.23 EI_DATA . . . 138
10.27.2.24 EI_DATA . . . 138
10.27.2.25 EI_OSABI . . . 138
10.27.2.26 EI_OSABI . . . 139
10.27.2.27 EI_PAD . . . 139
10.27.2.28 EI_PAD . . . 139
10.27.2.29 EI_VERSION . . . 139
10.27.2.30 EI_VERSION . . . 139
10.27.2.31 ELFCLASSNONE . . . 139
10.27.2.32 ELFCLASSNONE . . . 139
10.27.2.33 ELFDATA2LSB . . . 140
10.27.2.34 ELFDATA2LSB . . . 140
10.27.2.35 ELFDATA2MSB . . . 140
10.27.2.36 ELFDATA2MSB . . . 140
10.27.2.37 ELFDATANONE . . . 140
10.27.2.38 ELFDATANONE . . . 140
10.27.2.39 ELFOSABI_AIX . . . 140
10.27.2.40 ELFOSABI_FREEBSD . . . 140
10.27.2.41 ELFOSABI_HPUX . . . 141
10.27.2.42 ELFOSABI_HPUX . . . 141
10.27.2.43 ELFOSABI_IRIX . . . 141
10.27.2.44 ELFOSABI_LINUX . . . 141
10.27.2.45 ELFOSABI_MODESTO . . . 141
10.27.2.46 ELFOSABI_NETBSD . . . 141

10.27.2.47 ELFOSABI_OPENBSD . . . 141
10.27.2.48 ELFOSABI_SOLARIS . . . 141
10.27.2.49 ELFOSABI_SYSV . . . 141
10.27.2.50 ELFOSABI_SYSV . . . 142
10.27.2.51 ELFOSABI_TRU64 . . . 142
10.27.2.52 EM_ARC . . . 142
10.27.2.53 NT_VERSION . . . 142
10.27.2.54 PT_GNU_EH_FRAME . . . 142
10.27.2.55 PT_GNU_RELRO . . . 142
10.27.2.56 PT_GNU_STACK . . . 142
10.27.2.57 PT_HIOS . . . 142
10.27.2.58 PT_HIPROC . . . 142
10.27.2.59 PT_L4_AUX . . . 143
10.27.2.60 PT_L4_KIP . . . 143
10.27.2.61 PT_L4_STACK . . . 143
10.27.2.62 PT_LOOS . . . 143
10.27.2.63 PT_LOPROC . . . 143
10.27.2.64 SHF_GROUP . . . 143
10.27.2.65 SHF_MASKOS . . . 143
10.27.2.66 SHF_TLS . . . 143
10.27.2.67 SHT_NUM . . . 143
10.28 Error Handling . . . 144
10.28.1 Detailed Description . . . 144
10.28.2 Enumeration Type Documentation . . . 144
10.28.2.1 l4_ipc_tcr_error_t . . . 144
10.28.3 Function Documentation . . . 145
10.28.3.1 l4_error . . . 145
10.28.3.2 l4_ipc_error . . . 146
10.28.3.3 l4_ipc_error_code . . . 147
10.28.3.4 l4_ipc_is_rcv_error . . . 147
10.28.3.5 l4_ipc_is_snd_error . . . 148
10.29 Error codes . . . 149
10.29.1 Detailed Description . . . 149
10.29.2 Enumeration Type Documentation . . . 149
10.29.2.1 l4_error_code_t . . . 149
10.30 Event API . . . 151
10.30.1 Detailed Description . . . 151
10.31 Event interface . . . 152
10.31.1 Detailed Description . . . 152
10.31.2 Function Documentation . . . 152

10.31.2.1 l4re_event_get_axis_info . . . 152
10.31.2.2 l4re_event_get_buffer . . . 153
10.31.2.3 l4re_event_get_num_streams . . . 153
10.31.2.4 l4re_event_get_stream_info . . . 153
10.31.2.5 l4re_event_get_stream_info_for_id . . . 154
10.32 Exception registers . . . 155
10.32.1 Detailed Description . . . 155
10.32.2 Function Documentation . . . 155
10.32.2.1 l4_utcb_exc . . . 155
10.32.2.2 l4_utcb_exc_is_pf . . . 156
10.32.2.3 l4_utcb_exc_pc . . . 156
10.32.2.4 l4_utcb_exc_pc_set . . . 156
10.33 Extended vCPU support . . . 157
10.33.1 Detailed Description . . . 157
10.33.2 Function Documentation . . . 157
10.33.2.1 l4vcpu_ext_alloc . . . 157
10.34 Factory . . . 158
10.34.1 Detailed Description . . . 159
10.34.2 Function Documentation . . . 159
10.34.2.1 l4_factory_create_factory . . . 159
10.34.2.2 l4_factory_create_factory_u . . . 160
10.34.2.3 l4_factory_create_gate . . . 161
10.34.2.4 l4_factory_create_gate_u . . . 162
10.34.2.5 l4_factory_create_irq . . . 163
10.34.2.6 l4_factory_create_irq_u . . . 164
10.34.2.7 l4_factory_create_task . . . 165
10.34.2.8 l4_factory_create_task_u . . . 166
10.34.2.9 l4_factory_create_thread . . . 167
10.34.2.10 l4_factory_create_thread_u . . . 168
10.34.2.11 l4_factory_create_vm . . . 169
10.34.2.12 l4_factory_create_vm_u . . . 170
10.35 Fiasco extensions . . . 172
10.35.1 Detailed Description . . . 173
10.35.2 Function Documentation . . . 173
10.35.2.1 fiasco_gdt_get_entry_offset . . . 173
10.35.2.2 fiasco_gdt_set . . . 174
10.35.2.3 fiasco_ldt_set . . . 175
10.35.2.4 fiasco_tbuf_get_status . . . 176
10.35.2.5 fiasco_tbuf_get_status_phys . . . 176
10.35.2.6 fiasco_tbuf_log . . . 176

10.35.2.7 fiasco_tbuf_log_3val . . . 177
10.35.2.8 fiasco_tbuf_log_binary . . . 177
10.35.2.9 fiasco_watchdog_takeover . . . 177
10.35.2.10 fiasco_watchdog_touch . . . 177
10.36 Fiasco real time scheduling extensions . . . 178
10.37 Fiasco-UX Virtual devices . . . 179
10.37.1 Detailed Description . . . 179
10.37.2 Enumeration Type Documentation . . . 179
10.37.2.1 l4_vhw_entry_type . . . 179
10.38 Flex pages . . . 180
10.38.1 Detailed Description . . . 181
10.38.2 Enumeration Type Documentation . . . 182
10.38.2.1 anonymous enum . . . 182
10.38.2.2 anonymous enum . . . 182
10.38.2.3 L4_cap_fpage_rights . . . 182
10.38.2.4 l4_fpage_cacheability_opt_t . . . 183
10.38.2.5 l4_fpage_consts . . . 183
10.38.2.6 L4_fpage_rights . . . 184
10.38.2.7 L4_obj_fpage_ctl . . . 184
10.38.3 Function Documentation . . . 184
10.38.3.1 l4_fpage . . . 184
10.38.3.2 l4_fpage_all . . . 184
10.38.3.3 l4_fpage_contains . . . 184
10.38.3.4 l4_fpage_invalid . . . 185
10.38.3.5 l4_fpage_max_order . . . 185
10.38.3.6 l4_fpage_page . . . 186
10.38.3.7 l4_fpage_rights . . . 187
10.38.3.8 l4_fpage_set_rights . . . 188
10.38.3.9 l4_fpage_size . . . 188
10.38.3.10 l4_fpage_type . . . 189
10.38.3.11 l4_iofpage . . . 190
10.38.3.12 l4_is_fpage_writable . . . 190
10.38.3.13 l4_obj_fpage . . . 190
10.39 Functions for rendering bitmap data in frame buffers . . . 192
10.39.1 Detailed Description . . . 192
10.39.2 Typedef Documentation . . . 192
10.39.2.1 gfxbitmap_color_pix_t . . . 193
10.39.2.2 gfxbitmap_color_t . . . 193
10.39.3 Function Documentation . . . 193
10.39.3.1 gfxbitmap_bmap . . . 193

10.39.3.2 gfxbitmap_convert_color . . . 193
10.39.3.3 gfxbitmap_copy . . . 193
10.39.3.4 gfxbitmap_fill . . . 194
10.39.3.5 gfxbitmap_set . . . 194
10.40 Functions for rendering bitmap fonts to frame buffers . . . 195
10.40.1 Detailed Description . . . 196
10.40.2 Enumeration Type Documentation . . . 196
10.40.2.1 anonymous enum . . . 196
10.40.3 Function Documentation . . . 196
10.40.3.1 gfxbitmap_font_data . . . 196
10.40.3.2 gfxbitmap_font_get . . . 196
10.40.3.3 gfxbitmap_font_height . . . 196
10.40.3.4 gfxbitmap_font_init . . . 196
10.40.3.5 gfxbitmap_font_text . . . 197
10.40.3.6 gfxbitmap_font_text_scale . . . 197
10.40.3.7 gfxbitmap_font_width . . . 197
10.41 Functions to manipulate the local IDT . . . 198
10.41.1 Detailed Description . . . 198
10.42 IA32 Port I/O API . . . 199
10.42.1 Detailed Description . . . 199
10.42.2 Function Documentation . . . 199
10.42.2.1 l4util_in16 . . . 199
10.42.2.2 l4util_in32 . . . 200
10.42.2.3 l4util_in8 . . . 200
10.42.2.4 l4util_ins16 . . . 200
10.42.2.5 l4util_ins32 . . . 200
10.42.2.6 l4util_ins8 . . . 201
10.42.2.7 l4util_out16 . . . 201
10.42.2.8 l4util_out32 . . . 201
10.42.2.9 l4util_out8 . . . 201
10.42.2.10 l4util_outs16 . . . 201
10.42.2.11 l4util_outs32 . . . 202
10.42.2.12 l4util_outs8 . . . 202
10.43 IO interface . . . 203
10.43.1 Detailed Description . . . 204
10.43.2 Typedef Documentation . . . 204
10.43.2.1 l4io_resource_t . . . 204
10.43.3 Enumeration Type Documentation . . . 204
10.43.3.1 l4io_device_types_t . . . 204
10.43.3.2 l4io_iomem_flags_t . . . 204

10.43.3.3 l4io_resource_types_t . . . 204
10.43.4 Function Documentation . . . 205
10.43.4.1 l4io_has_resource . . . 205
10.43.4.2 l4io_lookup_device . . . 205
10.43.4.3 l4io_lookup_resource . . . 205
10.43.4.4 l4io_release_iomem . . . 205
10.43.4.5 l4io_release_ioport . . . 206
10.43.4.6 l4io_request_iomem . . . 206
10.43.4.7 l4io_request_iomem_region . . . 206
10.43.4.8 l4io_request_ioport . . . 207
10.43.4.9 l4io_request_resource_iomem . . . 207
10.43.4.10 l4io_search_iomem_region . . . 207
10.44 IPC Messaging Framework . . . 209
10.45 IPC Streams . . . 210
10.46 IPC-Gate API . . . 211
10.46.1 Detailed Description . . . 211
10.46.2 Function Documentation . . . 211
10.46.2.1 l4_ipc_gate_bind_thread . . . 211
10.46.2.2 l4_ipc_gate_get_infos . . . 212
10.47 IRQ handling library . . . 213
10.47.1 Detailed Description . . . 213
10.48 IRQs . . . 214
10.48.1 Detailed Description . . . 215
10.48.2 Enumeration Type Documentation . . . 215
10.48.2.1 L4_irq_mode . . . 215
10.48.3 Function Documentation . . . 216
10.48.3.1 l4_irq_attach . . . 216
10.48.3.2 l4_irq_attach_u . . . 217
10.48.3.3 l4_irq_detach . . . 218
10.48.3.4 l4_irq_detach_u . . . 219
10.48.3.5 l4_irq_mux_chain . . . 220
10.48.3.6 l4_irq_mux_chain_u . . . 220
10.48.3.7 l4_irq_receive . . . 221
10.48.3.8 l4_irq_receive_u . . . 222
10.48.3.9 l4_irq_trigger . . . 223
10.48.3.10 l4_irq_trigger_u . . . 224
10.48.3.11 l4_irq_unmask . . . 225
10.48.3.12 l4_irq_unmask_u . . . 226
10.48.3.13 l4_irq_wait . . . 227
10.48.3.14 l4_irq_wait_u . . . 227

10.49 Initial Environment . . . 229
10.49.1 Detailed Description . . . 229
10.49.2 Typedef Documentation . . . 230
10.49.2.1 l4re_env_t . . . 230
10.49.3 Function Documentation . . . 230
10.49.3.1 l4re_env . . . 230
10.49.3.2 l4re_env_get_cap . . . 231
10.49.3.3 l4re_env_get_cap_e . . . 231
10.49.3.4 l4re_env_get_cap_l . . . 232
10.49.3.5 l4re_kip . . . 233
10.50 Integer Types . . . 234
10.50.1 Detailed Description . . . 235
10.50.2 Typedef Documentation . . . 235
10.50.2.1 l4_int16_t . . . 235
10.50.2.2 l4_int32_t . . . 235
10.50.2.3 l4_int64_t . . . 235
10.50.2.4 l4_int8_t . . . 235
10.50.2.5 l4_uint16_t . . . 236
10.50.2.6 l4_uint32_t . . . 236
10.50.2.7 l4_uint64_t . . . 236
10.50.2.8 l4_uint8_t . . . 236
10.51 Interface for asynchronous ISR handlers with a given IRQ capability. . . . 237
10.51.1 Detailed Description . . . 237
10.51.2 Function Documentation . . . 237
10.51.2.1 l4irq_request_cap . . . 237
10.52 Interface for asynchronous ISR handlers. . . . 238
10.52.1 Detailed Description . . . 238
10.52.2 Function Documentation . . . 238
10.52.2.1 l4irq_release . . . 238
10.52.2.2 l4irq_request . . . 238
10.53 Interface using direct functionality. . . . 240
10.53.1 Detailed Description . . . 240
10.53.2 Function Documentation . . . 240
10.53.2.1 l4irq_attach . . . 240
10.53.2.2 l4irq_attach_ft . . . 241
10.53.2.3 l4irq_attach_thread . . . 241
10.53.2.4 l4irq_attach_thread_ft . . . 241
10.53.2.5 l4irq_detach . . . 242
10.53.2.6 l4irq_unmask . . . 242
10.53.2.7 l4irq_unmask_and_wait_any . . . 242

10.53.2.8 l4irq_wait . . . 242
10.53.2.9 l4irq_wait_any . . . 243
10.54 Interface using direct functionality. . . . 244
10.54.1 Detailed Description . . . 244
10.54.2 Function Documentation . . . 244
10.54.2.1 l4irq_attach_cap . . . 244
10.54.2.2 l4irq_attach_cap_ft . . . 244
10.54.2.3 l4irq_attach_thread_cap . . . 244
10.54.2.4 l4irq_attach_thread_cap_ft . . . 245
10.55 Internal Helpers . . . 246
10.55.1 Detailed Description . . . 246
10.56 Internal constants . . . 247
10.56.1 Detailed Description . . . 248
10.57 Internal functions . . . 249
10.57.1 Detailed Description . . . 249
10.58 Interrupt controller . . . 250
10.58.1 Detailed Description . . . 251
10.58.2 Typedef Documentation . . . 251
10.58.2.1 l4_icu_info_t . . . 251
10.58.3 Enumeration Type Documentation . . . 251
10.58.3.1 L4_icu_flags . . . 251
10.58.4 Function Documentation . . . 252
10.58.4.1 l4_icu_bind . . . 252
10.58.4.2 l4_icu_bind_u . . . 252
10.58.4.3 l4_icu_info . . . 253
10.58.4.4 l4_icu_info_u . . . 254
10.58.4.5 l4_icu_mask . . . 255
10.58.4.6 l4_icu_mask_u . . . 256
10.58.4.7 l4_icu_msi_info . . . 256
10.58.4.8 l4_icu_msi_info_u . . . 257
10.58.4.9 l4_icu_set_mode . . . 258
10.58.4.10 l4_icu_set_mode_u . . . 259
10.58.4.11 l4_icu_unbind . . . 260
10.58.4.12 l4_icu_unbind_u . . . 260
10.58.4.13 l4_icu_unmask . . . 261
10.58.4.14 l4_icu_unmask_u . . . 262
10.59 Kernel Debugger . . . 263
10.59.1 Detailed Description . . . 264
10.59.2 Macro Definition Documentation . . . 264
10.59.2.1 asm_enter_kdebug . . . 264

10.59.2.2 asm_enter_kdebug . . . 264
10.59.2.3 enter_kdebug . . . 265
10.59.2.4 enter_kdebug . . . 265
10.59.2.5 kd_display . . . 265
10.59.2.6 kd_display . . . 266
10.59.2.7 ko . . . 266
10.59.2.8 ko . . . 266
10.59.3 Function Documentation . . . 267
10.59.3.1 l4_debugger_global_id . . . 267
10.59.3.2 l4_debugger_kobj_to_id . . . 267
10.59.3.3 l4_debugger_set_object_name . . . 268
10.59.3.4 l4kd_inchar . . . 268
10.59.3.5 outchar . . . 268
10.59.3.6 outdec . . . 269
10.59.3.7 outhex12 . . . 269
10.59.3.8 outhex16 . . . 269
10.59.3.9 outhex20 . . . 269
10.59.3.10 outhex32 . . . 269
10.59.3.11 outhex8 . . . 269
10.59.3.12 outnstring . . . 270
10.59.3.13 outstring . . . 270
10.60 Kernel Interface Page . . . 271
10.60.1 Detailed Description . . . 272
10.60.2 Function Documentation . . . 272
10.60.2.1 l4_kernel_info_version_offset . . . 272
10.60.2.2 l4_kip_clock . . . 272
10.60.2.3 l4_kip_clock_lw . . . 273
10.60.2.4 l4_kip_version . . . 273
10.60.2.5 l4_kip_version_string . . . 274
10.61 Kernel Interface Page API . . . 275
10.61.1 Detailed Description . . . 275
10.61.2 Macro Definition Documentation . . . 275
10.61.2.1 l4util_kip_for_each_feature . . . 275
10.61.3 Function Documentation . . . 275
10.61.3.1 l4util_kip_kernel_abi_version . . . 275
10.61.3.2 l4util_kip_kernel_has_feature . . . 276
10.61.3.3 l4util_kip_kernel_is_ux . . . 276
10.61.3.4 l4util_memdesc_vm_high . . . 276
10.62 Kernel Objects . . . 277
10.62.1 Detailed Description . . . 278

10.63 Kumem allocator utility . . . 279
10.63.1 Detailed Description . . . 279
10.63.2 Function Documentation . . . 279
10.63.2.1 l4re_util_kumem_alloc . . . 279
10.64 Kumem utiltities . . . 280
10.64.1 Detailed Description . . . 280
10.64.2 Function Documentation . . . 280
10.64.2.1 kumem_alloc . . . 280
10.65 L4 IPC Opcodes . . . 281
10.65.1 Detailed Description . . . 281
10.65.2 Enumeration Type Documentation . . . 281
10.65.2.1 L4_icu_opcode . . . 281
10.65.2.2 L4_ipc_gate_ops . . . 282
10.65.2.3 L4_platform_ctl_ops . . . 282
10.65.2.4 L4_task_ops . . . 282
10.65.2.5 L4_thread_ops . . . 283
10.65.2.6 L4_vcon_ops . . . 283
10.66 L4 V-BUS functions . . . 284
10.66.1 Detailed Description . . . 285
10.66.2 Enumeration Type Documentation . . . 285
10.66.2.1 L4vbus_dma_domain_assign_flags . . . 285
10.66.3 Function Documentation . . . 285
10.66.3.1 l4vbus_assign_dma_domain . . . 285
10.66.3.2 l4vbus_get_device_by_hid . . . 286
10.66.3.3 l4vbus_get_hid . . . 286
10.66.3.4 l4vbus_get_next_device . . . 287
10.66.3.5 l4vbus_get_resource . . . 287
10.66.3.6 l4vbus_is_compatible . . . 288
10.66.3.7 l4vbus_release_resource . . . 288
10.66.3.8 l4vbus_request_resource . . . 289
10.66.3.9 l4vbus_vicu_get_cap . . . 290
10.67 L4 VIRTIO Block Device . . . 292
10.67.1 Detailed Description . . . 292
10.67.2 Enumeration Type Documentation . . . 292
10.67.2.1 L4virtio_block_operations . . . 292
10.67.2.2 L4virtio_block_status . . . 293
10.68 L4 VIRTIO Interface . . . 294
10.68.1 Detailed Description . . . 294
10.69 L4 VIRTIO Transport Layer . . . 295
10.69.1 Detailed Description . . . 297

10.69.2 Typedef Documentation . . . 297
10.69.2.1 l4virtio_config_queue_t . . . 297
10.69.3 Enumeration Type Documentation . . . 297
10.69.3.1 Error . . . 297
10.69.3.2 L4_virtio_irq_status . . . 297
10.69.3.3 L4_virtio_opcodes . . . 297
10.69.3.4 L4virtio_device_ids . . . 298
10.69.3.5 L4virtio_device_status . . . 298
10.69.4 Function Documentation . . . 298
10.69.4.1 l4virtio_config_queue . . . 298
10.69.4.2 l4virtio_config_queues . . . 299
10.69.4.3 l4virtio_device_config . . . 299
10.69.4.4 l4virtio_register_ds . . . 299
10.69.4.5 l4virtio_register_iface . . . 300
10.69.4.6 l4virtio_set_status . . . 300
10.70 L4 kernel object type information . . . 301
10.70.1 Detailed Description . . . 302
10.70.2 Enumeration Type Documentation . . . 302
10.70.2.1 anonymous enum . . . 302
10.70.3 Function Documentation . . . 302
10.70.3.1 kobject_typeid . . . 302
10.71 L4Re C Interface . . . 303
10.71.1 Detailed Description . . . 304
10.71.2 Function Documentation . . . 304
10.71.2.1 l4re_inhibitor_acquire . . . 304
10.72 L4Re C++ Interface . . . 306
10.72.1 Detailed Description . . . 307
10.73 L4Re Capability API . . . 308
10.73.1 Detailed Description . . . 309
10.73.2 Function Documentation . . . 309
10.73.2.1 make_auto_cap . . . 309
10.73.2.2 make_auto_del_cap . . . 310
10.73.2.3 make_ref_cap . . . 310
10.73.2.4 make_ref_del_cap . . . 310
10.73.3 Variable Documentation . . . 310
10.73.3.1 cap_alloc . . . 310
10.74 L4Re ELF Auxiliary Information . . . 311
10.74.1 Detailed Description . . . 312
10.74.2 Macro Definition Documentation . . . 312
10.74.2.1 L4RE_ELF_AUX_ELEM . . . 312

10.74.2.2 L4RE_ELF_AUX_ELEM_T . . . 312
10.74.3 Enumeration Type Documentation . . . 312
10.74.3.1 anonymous enum . . . 312
10.75 L4Re Protocol identifiers . . . 313
10.75.1 Detailed Description . . . 313
10.76 L4Re Util C Interface . . . 314
10.77 L4Re Util C++ Interface . . . 315
10.77.1 Detailed Description . . . 315
10.78 L4vbus GPIO functions . . . 316
10.78.1 Detailed Description . . . 317
10.78.2 Enumeration Type Documentation . . . 317
10.78.2.1 L4vbus_gpio_generic_func . . . 317
10.78.2.2 L4vbus_gpio_pull_modes . . . 317
10.78.3 Function Documentation . . . 317
10.78.3.1 l4vbus_gpio_config_get . . . 317
10.78.3.2 l4vbus_gpio_config_pad . . . 318
10.78.3.3 l4vbus_gpio_config_pull . . . 318
10.78.3.4 l4vbus_gpio_get . . . 319
10.78.3.5 l4vbus_gpio_multi_config_pad . . . 319
10.78.3.6 l4vbus_gpio_multi_get . . . 320
10.78.3.7 l4vbus_gpio_multi_set . . . 321
10.78.3.8 l4vbus_gpio_multi_setup . . . 321
10.78.3.9 l4vbus_gpio_set . . . 322
10.78.3.10 l4vbus_gpio_setup . . . 323
10.78.3.11 l4vbus_gpio_to_irq . . . 323
10.79 L4vbus PCI functions . . . 325
10.79.1 Detailed Description . . . 325
10.79.2 Function Documentation . . . 325
10.79.2.1 l4vbus_pci_cfg_read . . . 325
10.79.2.2 l4vbus_pci_cfg_write . . . 326
10.79.2.3 l4vbus_pci_irq_enable . . . 327
10.79.2.4 l4vbus_pcidev_cfg_read . . . 328
10.79.2.5 l4vbus_pcidev_cfg_write . . . 329
10.79.2.6 l4vbus_pcidev_irq_enable . . . 329
10.80 Log interface . . . 331
10.80.1 Detailed Description . . . 331
10.80.2 Function Documentation . . . 331
10.80.2.1 l4re_log_print . . . 331
10.80.2.2 l4re_log_print_srv . . . 332
10.80.2.3 l4re_log_printn . . . 332

10.80.2.4 l4re_log_printn_srv . . . 333
10.81 Logging interface . . . 334
10.81.1 Detailed Description . . . 334
10.82 Low-Level Thread Functions . . . 335
10.83 Machine Restarting Function . . . 336
10.83.1 Detailed Description . . . 336
10.84 Memory allocator . . . 337
10.84.1 Detailed Description . . . 337
10.84.2 Enumeration Type Documentation . . . 337
10.84.2.1 l4re_ma_flags . . . 337
10.84.3 Function Documentation . . . 338
10.84.3.1 l4re_ma_alloc . . . 338
10.84.3.2 l4re_ma_alloc_align . . . 339
10.84.3.3 l4re_ma_alloc_align_srv . . . 340
10.84.3.4 l4re_ma_free . . . 341
10.84.3.5 l4re_ma_free_srv . . . 342
10.85 Memory descriptors (C version) . . . 343
10.85.1 Detailed Description . . . 344
10.85.2 Typedef Documentation . . . 344
10.85.2.1 l4_kernel_info_mem_desc_t . . . 344
10.85.3 Enumeration Type Documentation . . . 344
10.85.3.1 l4_mem_type_t . . . 344
10.85.4 Function Documentation . . . 344
10.85.4.1 l4_kernel_info_get_mem_desc_end . . . 344
10.85.4.2 l4_kernel_info_get_mem_desc_is_virtual . . . 345
10.85.4.3 l4_kernel_info_get_mem_desc_start . . . 345
10.85.4.4 l4_kernel_info_get_mem_desc_subtype . . . 345
10.85.4.5 l4_kernel_info_get_mem_desc_type . . . 345
10.85.4.6 l4_kernel_info_get_num_mem_descs . . . 345
10.85.4.7 l4_kernel_info_set_mem_desc . . . 345
10.86 Memory operations. . . . 347
10.86.1 Detailed Description . . . 347
10.86.2 Enumeration Type Documentation . . . 347
10.86.2.1 L4_mem_op_widths . . . 347
10.86.3 Function Documentation . . . 348
10.86.3.1 l4_mem_read . . . 348
10.86.3.2 l4_mem_write . . . 349
10.87 Memory related . . . 350
10.87.1 Detailed Description . . . 351
10.87.2 Macro Definition Documentation . . . 351

10.87.2.1 L4_LOG2_PAGESIZE . . . 351
10.87.2.2 L4_LOG2_SUPERPAGESIZE . . . 351
10.87.2.3 L4_PAGEMASK . . . 351
10.87.2.4 L4_SUPERPAGEMASK . . . 351
10.87.2.5 L4_SUPERPAGESIZE . . . 352
10.87.3 Enumeration Type Documentation . . . 352
10.87.3.1 l4_addr_consts_t . . . 352
10.87.4 Function Documentation . . . 352
10.87.4.1 l4_round_page . . . 352
10.87.4.2 l4_round_size . . . 352
10.87.4.3 l4_trunc_page . . . 353
10.87.4.4 l4_trunc_size . . . 354
10.88 Message Items . . . 355
10.88.1 Detailed Description . . . 355
10.88.2 Enumeration Type Documentation . . . 355
10.88.2.1 l4_msg_item_consts_t . . . 355
10.88.3 Function Documentation . . . 356
10.88.3.1 l4_map_control . . . 356
10.88.3.2 l4_map_obj_control . . . 356
10.89 Message Registers (MRs) . . . 358
10.89.1 Detailed Description . . . 358
10.90 Message Tag . . . 359
10.90.1 Detailed Description . . . 360
10.90.2 Typedef Documentation . . . 360
10.90.2.1 l4_msgtag_t . . . 360
10.90.3 Enumeration Type Documentation . . . 360
10.90.3.1 l4_msgtag_flags . . . 360
10.90.3.2 l4_msgtag_protocol . . . 361
10.90.3.3 L4_platform_ctl_proto . . . 362
10.90.4 Function Documentation . . . 362
10.90.4.1 l4_msgtag . . . 362
10.90.4.2 l4_msgtag_flags . . . 363
10.90.4.3 l4_msgtag_has_error . . . 364
10.90.4.4 l4_msgtag_is_exception . . . 364
10.90.4.5 l4_msgtag_is_io_page_fault . . . 365
10.90.4.6 l4_msgtag_is_page_fault . . . 365
10.90.4.7 l4_msgtag_is_preemption . . . 366
10.90.4.8 l4_msgtag_is_sigma0 . . . 366
10.90.4.9 l4_msgtag_is_sys_exception . . . 367
10.90.4.10 l4_msgtag_items . . . 367

10.90.4.11 l4_msgtag_label . . . 368
10.90.4.12 l4_msgtag_words . . . 368
10.91 Name-space API . . . 370
10.91.1 Detailed Description . . . 370
10.91.2 Enumeration Type Documentation . . . 370
10.91.2.1 Query_result_flags . . . 370
10.91.2.2 Register_flags . . . 371
10.92 Namespace interface . . . 372
10.92.1 Detailed Description . . . 372
10.92.2 Enumeration Type Documentation . . . 372
10.92.2.1 l4re_ns_register_flags . . . 372
10.92.3 Function Documentation . . . 372
10.92.3.1 l4re_ns_query_srv . . . 372
10.92.3.2 l4re_ns_query_to_srv . . . 373
10.92.3.3 l4re_ns_register_obj_srv . . . 374
10.93 Object Invocation . . . 375
10.93.1 Detailed Description . . . 376
10.93.2 Enumeration Type Documentation . . . 376
10.93.2.1 l4_syscall_flags_t . . . 376
10.93.3 Function Documentation . . . 377
10.93.3.1 l4_ipc . . . 377
10.93.3.2 l4_ipc_call . . . 378
10.93.3.3 l4_ipc_receive . . . 380
10.93.3.4 l4_ipc_reply_and_wait . . . 381
10.93.3.5 l4_ipc_send . . . 383
10.93.3.6 l4_ipc_send_and_wait . . . 384
10.93.3.7 l4_ipc_sleep . . . 385
10.93.3.8 l4_ipc_wait . . . 386
10.93.3.9 l4_sndfpage_add . . . 387
10.94 Parent API . . . 389
10.94.1 Detailed Description . . . 389
10.95 Platform Control C API . . . 390
10.95.1 Detailed Description . . . 390
10.95.2 Function Documentation . . . 390
10.95.2.1 l4_platform_ctl_cpu_disable . . . 390
10.95.2.2 l4_platform_ctl_cpu_enable . . . 391
10.95.2.3 l4_platform_ctl_system_shutdown . . . 391
10.95.2.4 l4_platform_ctl_system_suspend . . . 392
10.96 Producer . . . 393
10.96.1 Detailed Description . . . 393

10.96.2 Function Documentation . . . 393
10.96.2.1 l4shmc_chunk_ready . . . 393
10.96.2.2 l4shmc_chunk_ready_sig . . . 393
10.96.2.3 l4shmc_chunk_try_to_take . . . 394
10.96.2.4 l4shmc_is_chunk_clear . . . 394
10.97 Producer . . . 395
10.97.1 Detailed Description . . . 395
10.97.2 Function Documentation . . . 395
10.97.2.1 l4shmc_trigger . . . 395
10.98 Random number support . . . 396
10.98.1 Detailed Description . . . 396
10.98.2 Function Documentation . . . 396
10.98.2.1 l4util_rand . . . 396
10.98.2.2 l4util_srand . . . 396
10.99 Realtime API . . . 397
10.100 Region map API . . . 398
10.100.1 Detailed Description . . . 398
10.100.2 Enumeration Type Documentation . . . 399
10.100.2.1 Attach_flags . . . 399
10.100.2.2 Detach_flags . . . 399
10.100.2.3 Detach_result . . . 399
10.100.2.4 Region_flags . . . 400
10.101 Region map interface . . . 401
10.101.1 Detailed Description . . . 402
10.101.2 Enumeration Type Documentation . . . 402
10.101.2.1 l4re_rm_flags_t . . . 402
10.101.3 Function Documentation . . . 402
10.101.3.1 l4re_rm_attach . . . 402
10.101.3.2 l4re_rm_attach_srv . . . 403
10.101.3.3 l4re_rm_detach . . . 404
10.101.3.4 l4re_rm_detach_ds . . . 404
10.101.3.5 l4re_rm_detach_ds_unmap . . . 405
10.101.3.6 l4re_rm_detach_srv . . . 406
10.101.3.7 l4re_rm_detach_unmap . . . 406
10.101.3.8 l4re_rm_find . . . 407
10.101.3.9 l4re_rm_find_srv . . . 408
10.101.3.10 l4re_rm_free_area . . . 408
10.101.3.11 l4re_rm_free_area_srv . . . 408
10.101.3.12 l4re_rm_reserve_area . . . 409
10.101.3.13 l4re_rm_reserve_area_srv . . . 409

10.101.3.14 re_rm_show_lists . . . 410
10.102 Scheduler . . . 411
10.102.1 Detailed Description . . . 412
10.102.2 Enumeration Type Documentation . . . 412
10.102.2.1 L4_scheduler_ops . . . 412
10.102.3 Function Documentation . . . 412
10.102.3.1 idle_time . . . 412
10.102.3.2 info . . . 412
10.102.3.3 is_online . . . 413
10.102.3.4 l4_sched_cpu_set . . . 413
10.102.3.5 run_thread . . . 414
10.103 Server-Side IPC framework . . . 415
10.103.1 Detailed Description . . . 416
10.103.2 Enumeration Type Documentation . . . 416
10.103.2.1 Reply_mode . . . 416
10.104 Shared Memory Library . . . 417
10.104.1 Detailed Description . . . 417
10.104.2 Function Documentation . . . 418
10.104.2.1 l4shmc_area_overhead . . . 418
10.104.2.2 l4shmc_area_size . . . 418
10.104.2.3 l4shmc_area_size_free . . . 418
10.104.2.4 l4shmc_attach . . . 418
10.104.2.5 l4shmc_attach_to . . . 419
10.104.2.6 l4shmc_chunk_overhead . . . 419
10.104.2.7 l4shmc_connect_chunk_signal . . . 419
10.104.2.8 l4shmc_create . . . 420
10.105 Sigma0 API . . . 422
10.105.1 Detailed Description . . . 423
10.105.2 Enumeration Type Documentation . . . 423
10.105.2.1 l4sigma0_return_flags_t . . . 423
10.105.3 Function Documentation . . . 423
10.105.3.1 l4sigma0_debug_dump . . . 423
10.105.3.2 l4sigma0_map_anypage . . . 423
10.105.3.3 l4sigma0_map_errstr . . . 424
10.105.3.4 l4sigma0_map_iomem . . . 424
10.105.3.5 l4sigma0_map_kip . . . 424
10.105.3.6 l4sigma0_map_mem . . . 424
10.105.3.7 l4sigma0_map_tbuf . . . 425
10.105.3.8 l4sigma0_new_client . . . 425
10.106 Signals . . . 426

10.106.1 Detailed Description . . . 426
10.106.2 Function Documentation . . . 426
10.106.2.1 l4shmc_add_signal . . . 426
10.106.2.2 l4shmc_attach_signal . . . 427
10.106.2.3 l4shmc_attach_signal_to . . . 427
10.106.2.4 l4shmc_check_magic . . . 427
10.106.2.5 l4shmc_get_signal_to . . . 428
10.106.2.6 l4shmc_signal_cap . . . 428
10.107 Small C++ Template Library . . . 429
10.107.1 Detailed Description . . . 430
10.107.2 Enumeration Type Documentation . . . 430
10.107.2.1 anonymous enum . . . 430
10.107.2.2 anonymous enum . . . 430
10.107.2.3 anonymous enum . . . 431
10.107.3 Function Documentation . . . 431
10.107.3.1 max . . . 431
10.107.3.2 min . . . 431
10.107.3.3 operator new . . . 432
10.108 Task . . . 433
10.108.1 Detailed Description . . . 433
10.108.2 Enumeration Type Documentation . . . 434
10.108.2.1 l4_unmap_flags_t . . . 434
10.108.3 Function Documentation . . . 434
10.108.3.1 l4_task_add_ku_mem . . . 434
10.108.3.2 l4_task_cap_equal . . . 435
10.108.3.3 l4_task_cap_has_child . . . 435
10.108.3.4 l4_task_cap_valid . . . 436
10.108.3.5 l4_task_delete_obj . . . 436
10.108.3.6 l4_task_map . . . 437
10.108.3.7 l4_task_release_cap . . . 438
10.108.3.8 l4_task_unmap . . . 438
10.108.3.9 l4_task_unmap_batch . . . 440
10.109 Thread . . . 442
10.109.1 Detailed Description . . . 443
10.109.2 Enumeration Type Documentation . . . 444
10.109.2.1 L4_thread_control_flags . . . 444
10.109.2.2 L4_thread_control_mr_indices . . . 444
10.109.2.3 L4_thread_ex_regs_flags . . . 444
10.109.3 Function Documentation . . . 445
10.109.3.1 l4_thread_arm_set_tpidruro . . . 445

10.109.3.2 l4_thread_ex_regs . . . 445
10.109.3.3 l4_thread_ex_regs_ret . . . 446
10.109.3.4 l4_thread_ex_regs_ret_u . . . 447
10.109.3.5 l4_thread_ex_regs_u . . . 448
10.109.3.6 l4_thread_modify_sender_add . . . 449
10.109.3.7 l4_thread_modify_sender_commit . . . 450
10.109.3.8 l4_thread_modify_sender_start . . . 451
10.109.3.9 l4_thread_register_del_irq . . . 451
10.109.3.10 l4_thread_stats_time . . . 452
10.109.3.11 l4_thread_switch . . . 452
10.109.3.12 l4_thread_vcpu_control . . . 453
10.109.3.13 l4_thread_vcpu_control_ext . . . 453
10.109.3.14 l4_thread_vcpu_control_ext_u . . . 454
10.109.3.15 l4_thread_vcpu_control_u . . . 455
10.109.3.16 l4_thread_vcpu_resume_commit . . . 456
10.109.3.17 l4_thread_vcpu_resume_start . . . 457
10.109.3.18 l4_thread_yield . . . 457
10.110 Thread Control Registers (TCRs) . . . 459
10.110.1 Detailed Description . . . 459
10.111 Thread control . . . 460
10.111.1 Detailed Description . . . 460
10.111.2 Function Documentation . . . 461
10.111.2.1 l4_thread_control_alien . . . 461
10.111.2.2 l4_thread_control_bind . . . 462
10.111.2.3 l4_thread_control_commit . . . 463
10.111.2.4 l4_thread_control_exc_handler . . . 464
10.111.2.5 l4_thread_control_pager . . . 465
10.111.2.6 l4_thread_control_start . . . 466
10.111.2.7 l4_thread_control_ux_host_syscall . . . 466
10.112 Timeouts . . . 468
10.112.1 Detailed Description . . . 469
10.112.2 Macro Definition Documentation . . . 469
10.112.2.1 L4_IPC_TIMEOUT_0 . . . 469
10.112.3 Typedef Documentation . . . 469
10.112.3.1 l4_timeout_s . . . 469
10.112.3.2 l4_timeout_t . . . 469
10.112.4 Enumeration Type Documentation . . . 470
10.112.4.1 l4_timeout_abs_validity . . . 470
10.112.5 Function Documentation . . . 470
10.112.5.1 l4_ipc_timeout . . . 470

10.112.5.2 l4_rcv_timeout . . . 470
10.112.5.3 l4_snd_timeout . . . 470
10.112.5.4 l4_timeout . . . 470
10.112.5.5 l4_timeout_abs . . . 471
10.112.5.6 l4_timeout_get . . . 472
10.112.5.7 l4_timeout_is_absolute . . . 473
10.112.5.8 l4_timeout_rel . . . 473
10.112.5.9 l4_timeout_rel_get . . . 474
10.112.5.10 l4_utcb_mr64_idx . . . 474
10.113 Timestamp Counter . . . 476
10.113.1 Detailed Description . . . 477
10.113.2 Function Documentation . . . 477
10.113.2.1 l4_busy_wait_ns . . . 477
10.113.2.2 l4_busy_wait_us . . . 477
10.113.2.3 l4_calibrate_tsc . . . 478
10.113.2.4 l4_get_hz . . . 478
10.113.2.5 l4_ns_to_tsc . . . 479
10.113.2.6 l4_rdpmc . . . 479
10.113.2.7 l4_rdpmc_32 . . . 479
10.113.2.8 l4_rdtsc . . . 480
10.113.2.9 l4_rdtsc_32 . . . 480
10.113.2.10 l4_tsc_init . . . 480
10.113.2.11 l4_tsc_to_ns . . . 481
10.113.2.12 l4_tsc_to_s_and_ns . . . 481
10.113.2.13 l4_tsc_to_us . . . 481
10.114 Utility Functions . . . 483
10.114.1 Detailed Description . . . 484
10.114.2 Function Documentation . . . 484
10.114.2.1 l4_sleep . . . 484
10.114.2.2 l4_touch_ro . . . 485
10.114.2.3 l4_touch_rw . . . 485
10.114.2.4 l4util_micros2l4to . . . 486
10.114.2.5 l4util_splitlog2_hdl . . . 486
10.114.2.6 l4util_splitlog2_size . . . 487
10.115 VM API for SVM . . . 489
10.115.1 Detailed Description . . . 489
10.116 VM API for TZ . . . 490
10.116.1 Detailed Description . . . 490
10.117 VM API for VMX . . . 491
10.117.1 Detailed Description . . . 492

10.117.2 Enumeration Type Documentation . . . 492
10.117.2.1 anonymous enum . . . 492
10.117.2.2 L4_vm_vmx_caps_regs . . . 493
10.117.2.3 L4_vm_vmx_dfl1_regs . . . 493
10.117.3 Function Documentation . . . 493
10.117.3.1 l4_vm_vmx_clear . . . 493
10.117.3.2 l4_vm_vmx_field_len . . . 494
10.117.3.3 l4_vm_vmx_field_order . . . 494
10.117.3.4 l4_vm_vmx_get_caps . . . 495
10.117.3.5 l4_vm_vmx_get_caps_default1 . . . 495
10.117.3.6 l4_vm_vmx_get_cr2_index . . . 495
10.117.3.7 l4_vm_vmx_ptr_load . . . 496
10.117.3.8 l4_vm_vmx_read . . . 496
10.117.3.9 l4_vm_vmx_read_16 . . . 497
10.117.3.10 l4_vm_vmx_read_32 . . . 497
10.117.3.11 l4_vm_vmx_read_64 . . . 498
10.117.3.12 l4_vm_vmx_read_nat . . . 498
10.117.3.13 l4_vm_vmx_write . . . 499
10.117.3.14 l4_vm_vmx_write_16 . . . 500
10.117.3.15 l4_vm_vmx_write_32 . . . 500
10.117.3.16 l4_vm_vmx_write_64 . . . 500
10.117.3.17 l4_vm_vmx_write_nat . . . 501
10.118 Video API . . . 502
10.118.1 Detailed Description . . . 503
10.118.2 Typedef Documentation . . . 503
10.118.2.1 l4re_video_view_t . . . 503
10.118.3 Enumeration Type Documentation . . . 503
10.118.3.1 l4re_video_goos_info_flags_t . . . 503
10.118.3.2 l4re_video_view_info_flags_t . . . 504
10.118.4 Function Documentation . . . 504
10.118.4.1 l4re_video_goos_create_buffer . . . 504
10.118.4.2 l4re_video_goos_create_view . . . 504
10.118.4.3 l4re_video_goos_delete_buffer . . . 505
10.118.4.4 l4re_video_goos_delete_view . . . 505
10.118.4.5 l4re_video_goos_get_static_buffer . . . 505
10.118.4.6 l4re_video_goos_get_view . . . 505
10.118.4.7 l4re_video_goos_info . . . 505
10.118.4.8 l4re_video_goos_refresh . . . 506
10.118.4.9 l4re_video_view_get_info . . . 506
10.118.4.10 l4re_video_view_refresh . . . 506

10.118.4.14 l4re_video_view_set_info . . . 506
10.118.4.15 l4re_video_view_set_viewport . . . 507
10.118.4.16 l4re_video_view_stack . . . 507
10.119 Virtual Console . . . 508
10.119.1 Detailed Description . . . 509
10.119.2 Enumeration Type Documentation . . . 509
10.119.2.1 L4_vcon_i_flags . . . 509
10.119.2.2 L4_vcon_l_flags . . . 509
10.119.2.3 L4_vcon_o_flags . . . 510
10.119.2.4 L4_vcon_size_consts . . . 510
10.119.3 Function Documentation . . . 510
10.119.3.1 l4_vcon_get_attr . . . 510
10.119.3.2 l4_vcon_get_attr_u . . . 510
10.119.3.3 l4_vcon_read . . . 511
10.119.3.4 l4_vcon_read_u . . . 512
10.119.3.5 l4_vcon_read_with_flags . . . 513
10.119.3.6 l4_vcon_send . . . 514
10.119.3.7 l4_vcon_send_u . . . 515
10.119.3.8 l4_vcon_set_attr . . . 516
10.119.3.9 l4_vcon_set_attr_u . . . 517
10.119.3.10 l4_vcon_write . . . 518
10.119.3.11 l4_vcon_write_u . . . 518
10.120 Virtual Machines . . . 520
10.120.1 Detailed Description . . . 520
10.121 Virtual Registers (UTCBs) . . . 521
10.121.1 Detailed Description . . . 522
10.121.2 Typedef Documentation . . . 522
10.121.2.1 l4_utcb_t . . . 522
10.121.3 Function Documentation . . . 522
10.121.3.1 l4_utcb_br . . . 522
10.121.3.2 l4_utcb_mr . . . 523
10.121.3.3 l4_utcb_tcr . . . 523
10.122 amd64 Virtual Registers (UTCB) . . . 525
10.122.1 Detailed Description . . . 525
10.123 vCPU API . . . 526
10.123.1 Detailed Description . . . 527
10.123.2 Enumeration Type Documentation . . . 527
10.123.2.1 L4_vcpu_state_flags . . . 527
10.123.2.2 L4_vcpu_state_offset . . . 527
10.123.2.3 L4_vcpu_sticky_flags . . . 527

10.124 CPU Support Library . . . 528
10.124.1 Detailed Description . . . 529
10.124.2 Function Documentation . . . 529
10.124.2.1 l4vcpu_irq_disable . . . 529
10.124.2.2 l4vcpu_irq_disable_save . . . 529
10.124.2.3 l4vcpu_irq_enable . . . 530
10.124.2.4 l4vcpu_irq_restore . . . 531
10.124.2.5 l4vcpu_is_irq_entry . . . 532
10.124.2.6 l4vcpu_is_page_fault_entry . . . 532
10.124.2.7 l4vcpu_print_state . . . 533
10.124.2.8 l4vcpu_wait_for_event . . . 533
10.125 x86 Virtual Registers (UTCB) . . . 535
10.125.1 Detailed Description . . . 535
10.125.2 Enumeration Type Documentation . . . 535
10.125.2.1 L4_utcb_consts_x86 . . . 535

11 Namespace Documentation **537**
11.1 cxx Namespace Reference . . . 537
11.1.1 Detailed Description . . . 538
11.2 cxx::Bits Namespace Reference . . . 539
11.2.1 Detailed Description . . . 539
11.3 L4 Namespace Reference . . . 539
11.3.1 Detailed Description . . . 542
11.3.2 Enumeration Type Documentation . . . 542
11.3.2.1 anonymous enum . . . 542
11.3.3 Function Documentation . . . 543
11.3.3.1 cap_cast . . . 543
11.3.3.2 cap_dynamic_cast . . . 544
11.3.3.3 cap_reinterpret_cast . . . 545
11.3.3.4 throw_ipc_exception . . . 546
11.3.3.5 throw_ipc_exception . . . 546
11.4 L4::Ipc Namespace Reference . . . 547
11.4.1 Detailed Description . . . 549
11.4.2 Function Documentation . . . 549
11.4.2.1 buf_cp_in . . . 549
11.4.2.2 buf_cp_out . . . 549
11.4.2.3 buf_in . . . 550
11.4.2.4 make_cap . . . 550
11.4.2.5 make_cap_rw . . . 550
11.4.2.6 make_cap_rws . . . 551

11.4.2.7 msg_ptr . . . 552
11.4.2.8 read . . . 552
11.4.2.9 str_cp_in . . . 552
11.5 L4::Ipc::Msg Namespace Reference . . . 552
11.5.1 Detailed Description . . . 554
11.5.2 Enumeration Type Documentation . . . 554
11.5.2.1 anonymous enum . . . 554
11.5.3 Function Documentation . . . 554
11.5.3.1 align_to . . . 554
11.5.3.2 align_to . . . 554
11.5.3.3 check_size . . . 555
11.5.3.4 check_size . . . 555
11.5.3.5 msg_add . . . 556
11.5.3.6 msg_get . . . 556
11.6 L4::Ipc_svr Namespace Reference . . . 557
11.6.1 Detailed Description . . . 557
11.7 L4::Typeid Namespace Reference . . . 557
11.7.1 Detailed Description . . . 558
11.8 L4::Types Namespace Reference . . . 558
11.8.1 Detailed Description . . . 558
11.9 L4Re Namespace Reference . . . 558
11.9.1 Detailed Description . . . 560
11.9.2 Function Documentation . . . 560
11.9.2.1 chkcap . . . 560
11.9.2.2 chksys . . . 560
11.9.2.3 chksys . . . 561
11.9.2.4 chksys . . . 562
11.10 L4Re::Vfs Namespace Reference . . . 562
11.10.1 Detailed Description . . . 563
11.11 L4virtio Namespace Reference . . . 563
11.11.1 Detailed Description . . . 563

12 Data Structure Documentation **565**
12.1 cxx::Auto_ptr< T > Class Template Reference . . . 565
12.1.1 Detailed Description . . . 566
12.1.2 Member Typedef Documentation . . . 566
12.1.2.1 Ref_type . . . 566
12.1.3 Constructor & Destructor Documentation . . . 566
12.1.3.1 Auto_ptr . . . 566
12.1.3.2 Auto_ptr . . . 566

12.1.3.3 ~Auto_ptr . . . 567
12.1.4 Member Function Documentation . . . 567
12.1.4.1 get . . . 567
12.1.4.2 operator Priv_type * . . . 567
12.1.4.3 operator* . . . 567
12.1.4.4 operator-> . . . 567
12.1.4.5 operator= . . . 567
12.1.4.6 release . . . 568
12.2 cxx::Avl_map< KEY_TYPE, DATA_TYPE, COMPARE, ALLOC > Class Template Reference . . . 568
12.2.1 Detailed Description . . . 571
12.2.2 Constructor & Destructor Documentation . . . 571
12.2.2.1 Avl_map . . . 571
12.2.3 Member Function Documentation . . . 571
12.2.3.1 insert . . . 571
12.2.3.2 operator[] . . . 572
12.2.3.3 operator[] . . . 572
12.3 cxx::Avl_set< ITEM_TYPE, COMPARE, ALLOC > Class Template Reference . . . 573
12.3.1 Detailed Description . . . 575
12.4 cxx::Avl_tree< Node, Get_key, Compare > Class Template Reference . . . 576
12.4.1 Detailed Description . . . 579
12.4.2 Member Function Documentation . . . 579
12.4.2.1 insert . . . 579
12.4.2.2 remove . . . 580
12.5 cxx::Avl_tree_node Class Reference . . . 581
12.5.1 Detailed Description . . . 582
12.6 cxx::Base_slab< Obj_size, Slab_size, Max_free, Alloc > Class Template Reference . . . 583
12.6.1 Detailed Description . . . 584
12.6.2 Member Function Documentation . . . 584
12.6.2.1 free_objects . . . 585
12.6.2.2 total_objects . . . 585
12.7 cxx::Base_slab_static< Obj_size, Slab_size, Max_free, Alloc > Class Template Reference . . . 586
12.7.1 Detailed Description . . . 587
12.7.2 Member Function Documentation . . . 588
12.7.2.1 alloc . . . 588
12.7.2.2 free . . . 588
12.7.2.3 free_objects . . . 588
12.7.2.4 total_objects . . . 588
12.8 cxx::Bitfield< T, LSB, MSB > Class Template Reference . . . 589
12.8.1 Detailed Description . . . 589
12.9 cxx::Bitmap< BITS > Class Template Reference . . . 589

12.9.1 Detailed Description . . . 591
12.9.2 Constructor & Destructor Documentation . . . 592
12.9.2.1 Bitmap . . . 592
12.9.3 Member Function Documentation . . . 592
12.9.3.1 scan_zero . . . 592
12.10 cxx::Bitmap_base Class Reference . . . 592
12.10.1 Detailed Description . . . 595
12.10.2 Member Function Documentation . . . 595
12.10.2.1 bit . . . 595
12.10.2.2 bit . . . 596
12.10.2.3 bit_index . . . 597
12.10.2.4 chars . . . 597
12.10.2.5 clear_bit . . . 598
12.10.2.6 operator[] . . . 598
12.10.2.7 operator[] . . . 599
12.10.2.8 scan_zero . . . 600
12.10.2.9 set_bit . . . 600
12.10.2.10 word_index . . . 601
12.10.2.11 words . . . 601
12.11 cxx::Bitmap_base::Bit Class Reference . . . 602
12.11.1 Detailed Description . . . 602
12.12 cxx::Bitmap_base::Char< BITS > Class Template Reference . . . 602
12.12.1 Detailed Description . . . 603
12.13 cxx::Bitmap_base::Word< BITS > Class Template Reference . . . 603
12.13.1 Detailed Description . . . 604
12.14 cxx::Bits::Avl_map_get_key< KEY_TYPE > Struct Template Reference . . . 604
12.14.1 Detailed Description . . . 604
12.15 cxx::Bits::Avl_set_get_key< KEY_TYPE > Struct Template Reference . . . 605
12.15.1 Detailed Description . . . 605
12.16 cxx::Bits::Base_avl_set< ITEM_TYPE, COMPARE, ALLOC, GET_KEY > Class Template Reference 605
12.16.1 Detailed Description . . . 608
12.16.2 Constructor & Destructor Documentation . . . 609
12.16.2.1 Base_avl_set . . . 609
12.16.2.2 Base_avl_set . . . 609
12.16.3 Member Function Documentation . . . 609
12.16.3.1 begin . . . 610
12.16.3.2 begin . . . 610
12.16.3.3 end . . . 610
12.16.3.4 end . . . 611
12.16.3.5 erase . . . 611

12.16.3.6 find_node . . . 611
12.16.3.7 insert . . . 612
12.16.3.8 lower_bound_node . . . 613
12.16.3.9 rbegin . . . 614
12.16.3.10 rbegin . . . 614
12.16.3.11 remove . . . 614
12.16.3.12 rend . . . 614
12.16.3.13 rend . . . 615
12.17 cxx::Bits::Base_avl_set< ITEM_TYPE, COMPARE, ALLOC, GET_KEY >::Node Class Reference . 615
12.17.1 Detailed Description . . . 616
12.17.2 Member Function Documentation . . . 616
12.17.2.1 operator* . . . 616
12.17.2.2 operator-> . . . 616
12.17.2.3 valid . . . 616
12.18 cxx::Bits::Basic_list< POLICY > Class Template Reference . . . 617
12.18.1 Detailed Description . . . 618
12.18.2 Member Function Documentation . . . 619
12.18.2.1 clear . . . 619
12.18.2.2 iter . . . 619
12.19 cxx::Bits::Bst< Node, Get_key, Compare > Class Template Reference . . . 619
12.19.1 Detailed Description . . . 623
12.19.2 Member Function Documentation . . . 623
12.19.2.1 begin . . . 623
12.19.2.2 begin . . . 624
12.19.2.3 dir . . . 624
12.19.2.4 dir . . . 625
12.19.2.5 end . . . 625
12.19.2.6 end . . . 625
12.19.2.7 find . . . 626
12.19.2.8 find_node . . . 627
12.19.2.9 lower_bound_node . . . 628
12.19.2.10 rbegin . . . 629
12.19.2.11 rbegin . . . 629
12.19.2.12 remove_all . . . 629
12.19.2.13 remove_tree . . . 630
12.19.2.14 rend . . . 630
12.19.2.15 rend . . . 631
12.20 cxx::Bits::Bst_node Class Reference . . . 631
12.20.1 Detailed Description . . . 633
12.21 cxx::Bits::Direction Struct Reference . . . 633

12.21.1 Detailed Description . . . 634
12.21.2 Member Enumeration Documentation . . . 635
12.21.2.1 Direction_e . . . 635
12.21.3 Member Function Documentation . . . 635
12.21.3.1 operator"! . . . 635
12.22cxx::H_list< T, POLICY > Class Template Reference . . . 635
12.22.1 Detailed Description . . . 638
12.22.2 Member Function Documentation . . . 638
12.22.2.1 erase . . . 638
12.22.2.2 insert . . . 639
12.22.2.3 insert_after . . . 639
12.22.2.4 insert_before . . . 639
12.22.2.5 iter . . . 640
12.22.2.6 pop_front . . . 641
12.22.2.7 remove . . . 641
12.22.2.8 replace . . . 642
12.23cxx::H_list_item_t< ELEM_TYPE > Class Template Reference . . . 643
12.23.1 Detailed Description . . . 645
12.23.2 Constructor & Destructor Documentation . . . 645
12.23.2.1 H_list_item_t . . . 645
12.23.2.2 ~H_list_item_t . . . 645
12.24cxx::H_list_t< T > Struct Template Reference . . . 646
12.24.1 Detailed Description . . . 647
12.25cxx::List< D, Alloc > Class Template Reference . . . 648
12.25.1 Detailed Description . . . 649
12.25.2 Member Function Documentation . . . 649
12.25.2.1 items . . . 649
12.25.2.2 operator[] . . . 649
12.25.2.3 operator[] . . . 649
12.25.2.4 push_back . . . 649
12.25.2.5 push_front . . . 649
12.25.2.6 remove . . . 650
12.25.2.7 size . . . 650
12.26cxx::List< D, Alloc >::Iter Class Reference . . . 650
12.26.1 Detailed Description . . . 650
12.27cxx::List_alloc Class Reference . . . 651
12.27.1 Detailed Description . . . 651
12.27.2 Constructor & Destructor Documentation . . . 651
12.27.2.1 List_alloc . . . 652
12.27.3 Member Function Documentation . . . 652

12.27.3.1 alloc . . . 652
12.27.3.2 alloc_max . . . 652
12.27.3.3 avail . . . 653
12.27.3.4 free . . . 653
12.28 cxx::List_item Class Reference . . . 653
12.28.1 Detailed Description . . . 655
12.28.2 Member Function Documentation . . . 655
12.28.2.1 get_next_item . . . 655
12.28.2.2 get_prev_item . . . 655
12.28.2.3 insert_next_item . . . 655
12.28.2.4 insert_prev_item . . . 655
12.28.2.5 push_back . . . 655
12.28.2.6 push_front . . . 656
12.28.2.7 remove . . . 656
12.28.2.8 remove_me . . . 656
12.29 cxx::List_item::Iter Class Reference . . . 657
12.29.1 Detailed Description . . . 658
12.29.2 Member Function Documentation . . . 658
12.29.2.1 remove_me . . . 658
12.30 cxx::List_item::T_iter< T, Poly > Class Template Reference . . . 659
12.30.1 Detailed Description . . . 660
12.31 cxx::Lt_functor< Obj > Struct Template Reference . . . 661
12.31.1 Detailed Description . . . 661
12.32 cxx::New_allocator< _Type > Class Template Reference . . . 661
12.32.1 Detailed Description . . . 662
12.33 cxx::Nothrow Class Reference . . . 662
12.33.1 Detailed Description . . . 662
12.34 cxx::Pair< First, Second > Struct Template Reference . . . 662
12.34.1 Detailed Description . . . 663
12.34.2 Constructor & Destructor Documentation . . . 664
12.34.2.1 Pair . . . 664
12.35 cxx::Pair_first_compare< Cmp, Typ > Class Template Reference . . . 664
12.35.1 Detailed Description . . . 664
12.35.2 Constructor & Destructor Documentation . . . 665
12.35.2.1 Pair_first_compare . . . 665
12.35.3 Member Function Documentation . . . 665
12.35.3.1 operator() . . . 665
12.36 cxx::Ref_ptr< T, CNT > Class Template Reference . . . 665
12.36.1 Detailed Description . . . 667
12.36.2 Constructor & Destructor Documentation . . . 668

12.36.2.1 Ref_ptr . . . 668
12.36.2.2 Ref_ptr . . . 668
12.36.2.3 Ref_ptr . . . 668
12.36.3 Member Function Documentation . . . 668
12.36.3.1 get . . . 668
12.36.3.2 ptr . . . 669
12.36.3.3 release . . . 669
12.37 cxx::S_list< T, POLICY > Class Template Reference . . . 669
12.37.1 Detailed Description . . . 671
12.37.2 Member Function Documentation . . . 673
12.37.2.1 pop_front . . . 673
12.38 cxx::Slab< Type, Slab_size, Max_free, Alloc > Class Template Reference . . . 673
12.38.1 Detailed Description . . . 676
12.38.2 Member Function Documentation . . . 676
12.38.2.1 alloc . . . 676
12.38.2.2 free . . . 676
12.39 cxx::Slab_static< Type, Slab_size, Max_free, Alloc > Class Template Reference . . . 676
12.39.1 Detailed Description . . . 678
12.39.2 Member Function Documentation . . . 679
12.39.2.1 alloc . . . 679
12.40 cxx::String Class Reference . . . 679
12.40.1 Detailed Description . . . 681
12.41 cxx::Weak_ref< T > Class Template Reference . . . 681
12.41.1 Detailed Description . . . 683
12.42 cxx::Weak_ref_base Class Reference . . . 684
12.42.1 Detailed Description . . . 686
12.43 Elf32_Dyn Struct Reference . . . 686
12.43.1 Detailed Description . . . 687
12.43.2 Field Documentation . . . 687
12.43.2.1 d_val . . . 687
12.44 Elf32_Ehdr Struct Reference . . . 687
12.44.1 Detailed Description . . . 689
12.44.2 Field Documentation . . . 689
12.44.2.1 e_phnum . . . 689
12.44.2.2 e_shnum . . . 689
12.45 Elf32_Phdr Struct Reference . . . 689
12.45.1 Detailed Description . . . 690
12.46 Elf32_Shdr Struct Reference . . . 691
12.46.1 Detailed Description . . . 692
12.47 Elf32_Sym Struct Reference . . . 692

12.47.1 Detailed Description . . . 692
12.48 Elf64_Dyn Struct Reference . . . 693
12.48.1 Detailed Description . . . 693
12.48.2 Field Documentation . . . 693
12.48.2.1 d_val . . . 693
12.49 Elf64_Ehdr Struct Reference . . . 694
12.49.1 Detailed Description . . . 695
12.49.2 Field Documentation . . . 695
12.49.2.1 e_phnum . . . 695
12.49.2.2 e_shnum . . . 695
12.50 Elf64_Phdr Struct Reference . . . 695
12.50.1 Detailed Description . . . 696
12.51 Elf64_Shdr Struct Reference . . . 697
12.51.1 Detailed Description . . . 698
12.52 Elf64_Sym Struct Reference . . . 698
12.52.1 Detailed Description . . . 698
12.53 gfxbitmap_offset Struct Reference . . . 699
12.53.1 Detailed Description . . . 699
12.54 L4::Alloc_list Class Reference . . . 699
12.54.1 Detailed Description . . . 700
12.55 L4::Base_exception Class Reference . . . 700
12.55.1 Detailed Description . . . 701
12.56 L4::Basic_registry Class Reference . . . 701
12.56.1 Detailed Description . . . 703
12.56.2 Member Function Documentation . . . 703
12.56.2.1 dispatch . . . 703
12.56.2.2 find . . . 703
12.57 L4::Bounds_error Class Reference . . . 704
12.57.1 Detailed Description . . . 705
12.58 L4::Cap< T > Class Template Reference . . . 706
12.58.1 Detailed Description . . . 707
12.58.2 Constructor & Destructor Documentation . . . 707
12.58.2.1 Cap . . . 707
12.58.2.2 Cap . . . 707
12.58.2.3 Cap . . . 707
12.58.2.4 Cap . . . 708
12.58.3 Member Function Documentation . . . 708
12.58.3.1 copy . . . 708
12.58.3.2 move . . . 708
12.59 L4::Com_error Class Reference . . . 708

12.59.1 Detailed Description . . . 711
12.59.2 Constructor & Destructor Documentation . . . 711
12.59.2.1 Com_error . . . 711
12.60 L4::Debugger Class Reference . . . 711
12.60.1 Detailed Description . . . 714
12.60.2 Member Function Documentation . . . 714
12.60.2.1 get_object_name . . . 714
12.60.2.2 global_id . . . 714
12.60.2.3 kobj_to_id . . . 715
12.60.2.4 query_log_name . . . 715
12.60.2.5 query_log_typeid . . . 716
12.60.2.6 set_object_name . . . 717
12.60.2.7 switch_log . . . 717
12.61 L4::Element_already_exists Class Reference . . . 718
12.61.1 Detailed Description . . . 720
12.62 L4::Element_not_found Class Reference . . . 721
12.62.1 Detailed Description . . . 722
12.63 L4::Exception_tracer Class Reference . . . 723
12.63.1 Detailed Description . . . 724
12.64 L4::Factory Class Reference . . . 724
12.64.1 Detailed Description . . . 727
12.64.2 Member Function Documentation . . . 727
12.64.2.1 create . . . 727
12.64.2.2 create_factory . . . 728
12.64.2.3 create_gate . . . 729
12.64.2.4 create_irq . . . 730
12.64.2.5 create_task . . . 731
12.64.2.6 create_thread . . . 731
12.64.2.7 create_vm . . . 732
12.65 L4::Factory::Lstr Struct Reference . . . 733
12.65.1 Detailed Description . . . 734
12.66 L4::Factory::Nil Struct Reference . . . 734
12.66.1 Detailed Description . . . 734
12.67 L4::Factory::S Class Reference . . . 734
12.67.1 Detailed Description . . . 735
12.67.2 Constructor & Destructor Documentation . . . 736
12.67.2.1 S . . . 736
12.67.3 Member Function Documentation . . . 736
12.67.3.1 operator l4_msgtag_t . . . 736
12.67.3.2 operator<< . . . 736

12.67.3.3 operator<< . . . 736
12.67.3.4 operator<< . . . 736
12.67.3.5 operator<< . . . 737
12.67.3.6 operator<< . . . 737
12.68 L4::Icu Class Reference . . . 737
12.68.1 Detailed Description . . . 740
12.68.2 Member Function Documentation . . . 740
12.68.2.1 bind . . . 740
12.68.2.2 info . . . 741
12.68.2.3 mask . . . 742
12.68.2.4 msi_info . . . 742
12.68.2.5 set_mode . . . 742
12.68.2.6 unbind . . . 743
12.69 L4::Icu::Info Class Reference . . . 744
12.69.1 Detailed Description . . . 745
12.70 L4::Invalid_capability Class Reference . . . 745
12.70.1 Detailed Description . . . 746
12.70.2 Constructor & Destructor Documentation . . . 747
12.70.2.1 Invalid_capability . . . 747
12.70.3 Member Function Documentation . . . 747
12.70.3.1 cap . . . 747
12.71 L4::Iommu Class Reference . . . 747
12.71.1 Detailed Description . . . 749
12.71.2 Member Function Documentation . . . 749
12.71.2.1 bind . . . 750
12.71.2.2 unbind . . . 751
12.72 L4::IOModifier Class Reference . . . 751
12.72.1 Detailed Description . . . 751
12.73 L4::Ipc::Array< ELEM_TYPE, LEN_TYPE > Struct Template Reference . . . 752
12.73.1 Detailed Description . . . 753
12.74 L4::Ipc::Array_in_buf< ELEM_TYPE, LEN_TYPE, MAX > Struct Template Reference . . . 754
12.74.1 Detailed Description . . . 754
12.75 L4::Ipc::Array_ref< ELEM_TYPE, LEN_TYPE > Struct Template Reference . . . 755
12.75.1 Detailed Description . . . 757
12.76 L4::Ipc::As_value< T > Struct Template Reference . . . 757
12.76.1 Detailed Description . . . 758
12.77 L4::Ipc::Buf_item Class Reference . . . 758
12.77.1 Detailed Description . . . 759
12.78 L4::Ipc::Call Struct Reference . . . 759
12.78.1 Detailed Description . . . 759

12.79 L4::Ipc::Call_t< RIGHTS > Struct Template Reference . . . 760
12.79.1 Detailed Description . . . 761
12.80 L4::Ipc::Call_zero_send_timeout Struct Reference . . . 761
12.80.1 Detailed Description . . . 762
12.81 L4::Ipc::Cap< T > Class Template Reference . . . 763
12.81.1 Detailed Description . . . 764
12.81.2 Member Enumeration Documentation . . . 764
12.81.2.1 anonymous enum . . . 764
12.81.3 Member Function Documentation . . . 764
12.81.3.1 from_ci . . . 764
12.82 L4::Ipc::Gen_fpage< T > Class Template Reference . . . 765
12.82.1 Detailed Description . . . 766
12.82.2 Member Function Documentation . . . 766
12.82.2.1 cap_received . . . 766
12.82.2.2 id_received . . . 766
12.82.2.3 local_id_received . . . 767
12.83 L4::Ipc::In_out< T > Struct Template Reference . . . 767
12.83.1 Detailed Description . . . 767
12.84 L4::Ipc::Iostream Class Reference . . . 767
12.84.1 Detailed Description . . . 770
12.84.2 Constructor & Destructor Documentation . . . 770
12.84.2.1 Iostream . . . 770
12.84.3 Member Function Documentation . . . 770
12.84.3.1 call . . . 770
12.84.3.2 reply_and_wait . . . 771
12.84.3.3 reply_and_wait . . . 771
12.84.3.4 reset . . . 772
12.85 L4::Ipc::Istream Class Reference . . . 772
12.85.1 Detailed Description . . . 775
12.85.2 Constructor & Destructor Documentation . . . 775
12.85.2.1 Istream . . . 775
12.85.3 Member Function Documentation . . . 776
12.85.3.1 get . . . 776
12.85.3.2 get . . . 776
12.85.3.3 get . . . 776
12.85.3.4 receive . . . 776
12.85.3.5 reset . . . 777
12.85.3.6 skip . . . 777
12.85.3.7 tag . . . 777
12.85.3.8 tag . . . 778

12.85.3.9 wait . . . 778
12.85.3.10 wait . . . 779
12.86 L4::Ipc::Msg::Clnt_val_ops< MTYPE, DIR, CLASS > Struct Template Reference . . . 779
12.86.1 Detailed Description . . . 780
12.87 L4::Ipc::Msg::Cls_buffer Struct Reference . . . 781
12.87.1 Detailed Description . . . 781
12.88 L4::Ipc::Msg::Cls_data Struct Reference . . . 782
12.88.1 Detailed Description . . . 782
12.89 L4::Ipc::Msg::Cls_item Struct Reference . . . 783
12.89.1 Detailed Description . . . 783
12.90 L4::Ipc::Msg::Dir_in Struct Reference . . . 783
12.90.1 Detailed Description . . . 784
12.91 L4::Ipc::Msg::Dir_out Struct Reference . . . 784
12.91.1 Detailed Description . . . 785
12.92 L4::Ipc::Msg::Do_in_data Struct Reference . . . 785
12.92.1 Detailed Description . . . 786
12.93 L4::Ipc::Msg::Do_in_items Struct Reference . . . 787
12.93.1 Detailed Description . . . 787
12.94 L4::Ipc::Msg::Do_out_data Struct Reference . . . 788
12.94.1 Detailed Description . . . 789
12.95 L4::Ipc::Msg::Do_out_items Struct Reference . . . 789
12.95.1 Detailed Description . . . 790
12.96 L4::Ipc::Msg::Do_rcv_buffers Struct Reference . . . 790
12.96.1 Detailed Description . . . 791
12.97 L4::Ipc::Msg::Elem< Array< A, LEN > & > Struct Template Reference . . . 792
12.97.1 Detailed Description . . . 792
12.98 L4::Ipc::Msg::Elem< Array< A, LEN > > Struct Template Reference . . . 792
12.98.1 Detailed Description . . . 793
12.99 L4::Ipc::Msg::Elem< Array_ref< A, LEN > & > Struct Template Reference . . . 793
12.99.1 Detailed Description . . . 794
12.100 L4::Ipc::Msg::Is_valid_rpc_type< T > Struct Template Reference . . . 794
12.100.1 Detailed Description . . . 796
12.101 L4::Ipc::Msg::Svr_arg_pack< IPC_TYPE > Struct Template Reference . . . 797
12.101.1 Detailed Description . . . 797
12.102 L4::Ipc::Msg::Svr_val_ops< MTYPE, DIR, CLASS > Struct Template Reference . . . 797
12.102.1 Detailed Description . . . 798
12.103 L4::Ipc::Msg_ptr< T > Class Template Reference . . . 798
12.103.1 Detailed Description . . . 799
12.103.2 Constructor & Destructor Documentation . . . 799
12.103.2.1 Msg_ptr . . . 799

12.104 L4::Ipc::Opt< T > Struct Template Reference . . . 799
12.104.1 Detailed Description . . . 801
12.105 L4::Ipc::Ostream Class Reference . . . 801
12.105.1 Detailed Description . . . 804
12.105.2 Member Function Documentation . . . 804
12.105.2.1 put . . . 804
12.105.2.2 put . . . 804
12.105.2.3 send . . . 805
12.105.2.4 tag . . . 805
12.105.2.5 tag . . . 805
12.106 L4::Ipc::Out< T > Struct Template Reference . . . 806
12.106.1 Detailed Description . . . 806
12.107 L4::Ipc::Ret_array< T > Struct Template Reference . . . 806
12.107.1 Detailed Description . . . 807
12.108 L4::Ipc::Send_only Struct Reference . . . 807
12.108.1 Detailed Description . . . 808
12.109 L4::Ipc::Small_buf Class Reference . . . 808
12.109.1 Detailed Description . . . 808
12.109.2 Constructor & Destructor Documentation . . . 809
12.109.2.1 Small_buf . . . 809
12.109.2.2 Small_buf . . . 810
12.110 L4::Ipc::Snd_item Class Reference . . . 810
12.110.1 Detailed Description . . . 810
12.111 L4::Ipc::Str_cp_in< T > Class Template Reference . . . 810
12.111.1 Detailed Description . . . 811
12.111.2 Constructor & Destructor Documentation . . . 811
12.111.2.1 Str_cp_in . . . 811
12.112 L4::Ipc::Varg Class Reference . . . 812
12.112.1 Detailed Description . . . 813
12.112.2 Member Function Documentation . . . 813
12.112.2.1 data . . . 813
12.112.2.2 get_value . . . 813
12.112.2.3 is_nil . . . 814
12.112.2.4 is_of . . . 814
12.112.2.5 is_of_int . . . 814
12.112.2.6 length . . . 815
12.112.2.7 tag . . . 815
12.112.2.8 type . . . 816
12.112.2.9 value . . . 816
12.113 L4::Ipc::Varg_list< MAX > Class Template Reference . . . 817

12.113.1 Detailed Description . . . 818
12.114 L4::Ipc::Varg_list_ref Class Reference . . . 818
12.114.1 Detailed Description . . . 819
12.114.2 Constructor & Destructor Documentation . . . 820
12.114.2.1 Varg_list_ref . . . 820
12.115 L4::Ipc_gate Class Reference . . . 820
12.115.1 Detailed Description . . . 822
12.115.2 Member Function Documentation . . . 823
12.115.2.1 bind_thread . . . 823
12.115.2.2 get_infos . . . 823
12.116 L4::Ipc_svr::Br_manager_no_buffers Class Reference . . . 823
12.116.1 Detailed Description . . . 826
12.116.2 Member Function Documentation . . . 826
12.116.2.1 alloc_buffer_demand . . . 826
12.117 L4::Ipc_svr::Compound_reply Struct Reference . . . 826
12.117.1 Detailed Description . . . 827
12.118 L4::Ipc_svr::Default_loop_hooks Struct Reference . . . 827
12.118.1 Detailed Description . . . 829
12.119 L4::Ipc_svr::Default_setup_wait Struct Reference . . . 829
12.119.1 Detailed Description . . . 829
12.120 L4::Ipc_svr::Default_timeout Struct Reference . . . 829
12.120.1 Detailed Description . . . 830
12.121 L4::Ipc_svr::Direct_dispatch< R > Struct Template Reference . . . 830
12.121.1 Detailed Description . . . 832
12.122 L4::Ipc_svr::Direct_dispatch< R * > Struct Template Reference . . . 832
12.122.1 Detailed Description . . . 832
12.123 L4::Ipc_svr::Exc_dispatch< R, Exc > Struct Template Reference . . . 833
12.123.1 Detailed Description . . . 834
12.124 L4::Ipc_svr::Ignore_errors Struct Reference . . . 834
12.124.1 Detailed Description . . . 835
12.125 L4::Ipc_svr::Server_iface Class Reference . . . 835
12.125.1 Detailed Description . . . 837
12.125.2 Member Function Documentation . . . 838
12.125.2.1 add_timeout . . . 838
12.125.2.2 alloc_buffer_demand . . . 838
12.125.2.3 get_rcv_cap . . . 838
12.125.2.4 rcv_cap . . . 839
12.125.2.5 rcv_cap . . . 840
12.125.2.6 realloc_rcv_cap . . . 841
12.125.2.7 remove_timeout . . . 842

12.126 L4::Ipc_svr::Timed_work< HOOKS > Class Template Reference . . . 842
12.126.1 Detailed Description . . . 843
12.127 L4::Ipc_svr::Timeout Class Reference . . . 843
12.127.1 Detailed Description . . . 844
12.127.2 Member Function Documentation . . . 844
12.127.2.1 expired . . . 844
12.127.2.2 timeout . . . 845
12.128 L4::Ipc_svr::Timeout_queue Class Reference . . . 845
12.128.1 Detailed Description . . . 846
12.128.2 Member Function Documentation . . . 846
12.128.2.1 add . . . 846
12.128.2.2 handle_expired_timeouts . . . 847
12.128.2.3 next_timeout . . . 848
12.128.2.4 remove . . . 849
12.128.2.5 timeout_expired . . . 850
12.129 L4::Ipc_svr::Timeout_queue_hooks< HOOKS, BR_MAN > Class Template Reference . . . 851
12.129.1 Detailed Description . . . 854
12.129.2 Member Function Documentation . . . 854
12.129.2.1 add_timeout . . . 854
12.129.2.2 remove_timeout . . . 854
12.130 L4::Irq Class Reference . . . 855
12.130.1 Detailed Description . . . 858
12.130.2 Member Function Documentation . . . 858
12.130.2.1 attach . . . 858
12.130.2.2 detach . . . 859
12.130.2.3 receive . . . 860
12.130.2.4 unmask . . . 860
12.130.2.5 wait . . . 861
12.131 L4::Irq_eoi Class Reference . . . 861
12.131.1 Detailed Description . . . 862
12.131.2 Member Function Documentation . . . 863
12.131.2.1 unmask . . . 863
12.132 L4::Irq_handler_object Struct Reference . . . 863
12.132.1 Detailed Description . . . 865
12.133 L4::Irq_mux Struct Reference . . . 866
12.133.1 Detailed Description . . . 868
12.133.2 Member Function Documentation . . . 868
12.133.2.1 chain . . . 868
12.134 L4::Kip::Mem_desc Class Reference . . . 868
12.134.1 Detailed Description . . . 870

12.134.2 Constructor & Destructor Documentation . . . 870
12.134.2.1 Mem_desc . . . 870
12.134.3 Member Function Documentation . . . 870
12.134.3.1 count . . . 870
12.134.3.2 count . . . 871
12.134.3.3 end . . . 871
12.134.3.4 first . . . 872
12.134.3.5 is_virtual . . . 872
12.134.3.6 set . . . 872
12.134.3.7 size . . . 872
12.134.3.8 start . . . 873
12.134.3.9 sub_type . . . 873
12.134.3.10 type . . . 874
12.135 L4::Kobject Class Reference . . . 874
12.135.1 Detailed Description . . . 875
12.135.2 Member Function Documentation . . . 875
12.135.2.1 cap . . . 875
12.135.2.2 dec_refcnt . . . 875
12.136 L4::Kobject_2t< Derived, Base1, Base2, PROTO, S_DEMAND > Class Template Reference . . . 876
12.136.1 Detailed Description . . . 877
12.136.2 Member Typedef Documentation . . . 878
12.136.2.1 __Iface . . . 878
12.136.2.2 __Iface_list . . . 878
12.136.2.3 Class . . . 878
12.136.3 Member Function Documentation . . . 878
12.136.3.1 __check_protocols__ . . . 878
12.136.3.2 c . . . 879
12.137 L4::Kobject_3t< Derived, Base1, Base2, Base3, PROTO, S_DEMAND > Struct Template Reference 879
12.137.1 Detailed Description . . . 881
12.137.2 Member Typedef Documentation . . . 882
12.137.2.1 __Iface . . . 882
12.137.2.2 __Iface_list . . . 882
12.137.2.3 Class . . . 882
12.137.3 Member Function Documentation . . . 882
12.137.3.1 __check_protocols__ . . . 882
12.137.3.2 c . . . 883
12.138 L4::Kobject_demand< T > Struct Template Reference . . . 883
12.138.1 Detailed Description . . . 883
12.139 L4::Kobject_t< Derived, Base, PROTO, S_DEMAND > Class Template Reference . . . 883
12.139.1 Detailed Description . . . 884

12.140 L4::Kobject_typeid< T > Struct Template Reference . . . 885
12.140.1 Detailed Description . . . 886
12.140.2 Member Typedef Documentation . . . 886
12.140.2.1 Demand . . . 886
12.140.3 Member Function Documentation . . . 886
12.140.3.1 demand . . . 886
12.140.3.2 id . . . 886
12.140.3.3 proto_dispatch . . . 887
12.141 L4::Kobject_x< Derived, ARGS > Struct Template Reference . . . 887
12.141.1 Detailed Description . . . 888
12.142 L4::Out_of_memory Class Reference . . . 890
12.142.1 Detailed Description . . . 893
12.143 L4::Platform_control Class Reference . . . 893
12.143.1 Detailed Description . . . 895
12.143.2 Member Enumeration Documentation . . . 895
12.143.2.1 Opcode . . . 895
12.143.3 Member Function Documentation . . . 895
12.143.3.1 cpu_disable . . . 895
12.143.3.2 cpu_enable . . . 895
12.143.3.3 system_shutdown . . . 896
12.143.3.4 system_suspend . . . 897
12.144 L4::Poll_timeout_kipclock Class Reference . . . 897
12.144.1 Detailed Description . . . 897
12.144.2 Constructor & Destructor Documentation . . . 898
12.144.2.1 Poll_timeout_kipclock . . . 898
12.144.3 Member Function Documentation . . . 898
12.144.3.1 set . . . 898
12.144.3.2 test . . . 899
12.144.3.3 timed_out . . . 900
12.145 L4::Proto_t< P > Struct Template Reference . . . 900
12.145.1 Detailed Description . . . 900
12.146 L4::Runtime_error Class Reference . . . 900
12.146.1 Detailed Description . . . 902
12.146.2 Constructor & Destructor Documentation . . . 903
12.146.2.1 Runtime_error . . . 903
12.146.3 Member Function Documentation . . . 903
12.146.3.1 err_no . . . 903
12.146.3.2 extra_str . . . 903
12.147 L4::Scheduler Class Reference . . . 903
12.147.1 Detailed Description . . . 906

12.148 L4::Semaphore Struct Reference . . . 906
12.148.1 Detailed Description . . . 909
12.148.2 Member Function Documentation . . . 909
12.148.2.1 down . . . 909
12.148.2.2 up . . . 909
12.149 L4::Server< LOOP_HOOKS > Class Template Reference . . . 910
12.149.1 Detailed Description . . . 912
12.149.2 Constructor & Destructor Documentation . . . 912
12.149.2.1 Server . . . 912
12.149.3 Member Function Documentation . . . 912
12.149.3.1 internal_loop . . . 912
12.149.3.2 loop . . . 913
12.150 L4::Server_object Class Reference . . . 914
12.150.1 Detailed Description . . . 914
12.150.2 Member Function Documentation . . . 915
12.150.2.1 dispatch . . . 915
12.151 L4::Server_object_t< IFACE, BASE > Struct Template Reference . . . 915
12.151.1 Detailed Description . . . 917
12.151.2 Member Function Documentation . . . 917
12.151.2.1 dispatch_meta_request . . . 917
12.151.2.2 get_buffer_demand . . . 917
12.151.2.3 proto_dispatch . . . 918
12.152 L4::Server_object_x< Derived, IFACE, BASE > Struct Template Reference . . . 918
12.152.1 Detailed Description . . . 920
12.153 L4::Smart_cap< T, SMART > Class Template Reference . . . 920
12.153.1 Detailed Description . . . 921
12.153.2 Constructor & Destructor Documentation . . . 921
12.153.2.1 Smart_cap . . . 921
12.154 L4::String Class Reference . . . 922
12.154.1 Detailed Description . . . 922
12.155 L4::Task Class Reference . . . 922
12.155.1 Detailed Description . . . 925
12.155.2 Member Function Documentation . . . 925
12.155.2.1 add_ku_mem . . . 925
12.155.2.2 cap_equal . . . 926
12.155.2.3 cap_has_child . . . 926
12.155.2.4 cap_valid . . . 927
12.155.2.5 delete_obj . . . 927
12.155.2.6 map . . . 928
12.155.2.7 release_cap . . . 929

12.155.2.8 unmap . . . 929
12.155.2.9 unmap_batch . . . 930
12.156 L4::Thread Class Reference . . . 931
12.156.1 Detailed Description . . . 934
12.156.2 Member Function Documentation . . . 935
12.156.2.1 control . . . 935
12.156.2.2 ex_regs . . . 935
12.156.2.3 ex_regs . . . 936
12.156.2.4 modify_senders . . . 936
12.156.2.5 register_del_irq . . . 937
12.156.2.6 stats_time . . . 938
12.156.2.7 switch_to . . . 938
12.156.2.8 vcpu_control . . . 939
12.156.2.9 vcpu_control_ext . . . 939
12.156.2.10 vcpu_resume_commit . . . 940
12.156.2.11 vcpu_resume_start . . . 941
12.157 L4::Thread::Attr Class Reference . . . 941
12.157.1 Detailed Description . . . 942
12.157.2 Constructor & Destructor Documentation . . . 942
12.157.2.1 Attr . . . 942
12.157.3 Member Function Documentation . . . 942
12.157.3.1 bind . . . 942
12.157.3.2 exc_handler . . . 942
12.157.3.3 exc_handler . . . 943
12.157.3.4 pager . . . 943
12.157.3.5 pager . . . 944
12.157.3.6 ux_host_syscall . . . 944
12.158 L4::Thread::Modify_senders Class Reference . . . 945
12.158.1 Detailed Description . . . 945
12.158.2 Member Function Documentation . . . 945
12.158.2.1 add . . . 945
12.159 L4::Triggerable Struct Reference . . . 946
12.159.1 Detailed Description . . . 948
12.159.2 Member Function Documentation . . . 948
12.159.2.1 trigger . . . 948
12.160 L4::Type_info Struct Reference . . . 949
12.160.1 Detailed Description . . . 950
12.161 L4::Type_info::Demand Class Reference . . . 950
12.161.1 Detailed Description . . . 952
12.161.2 Constructor & Destructor Documentation . . . 952

12.161.2.1 Demand . . . 952
12.161.3 Member Function Documentation . . . 953
12.161.3.1 no_demand . . . 953
12.162 L4::Type_info::Demand_t< CAPS, FLAGS, MEM, PORTS > Struct Template Reference . . . 953
12.162.1 Detailed Description . . . 955
12.163 L4::Type_info::Demand_union_t< D1, D2 > Struct Template Reference . . . 956
12.163.1 Detailed Description . . . 957
12.164 L4::Typeid::Detail::_Rpcs< OPCODE, O, X > Struct Template Reference . . . 958
12.164.1 Detailed Description . . . 959
12.165 L4::Typeid::Detail::_Rpcs< OPCODE, O, Default_op< R > >::Rpc< Y > Struct Template Reference 959
12.165.1 Detailed Description . . . 960
12.166 L4::Typeid::Detail::_Rpcs< OPCODE, O, R, X...> Struct Template Reference . . . 960
12.166.1 Detailed Description . . . 961
12.167 L4::Typeid::Detail::_Rpcs< OPCODE, O, R, X...>::Rpc< Y > Struct Template Reference . . . 961
12.167.1 Detailed Description . . . 962
12.168 L4::Typeid::Detail::Rpcs_end Struct Reference . . . 962
12.168.1 Detailed Description . . . 963
12.169 L4::Typeid::P_dispatch< LIST > Struct Template Reference . . . 963
12.169.1 Detailed Description . . . 963
12.170 L4::Typeid::Raw_ipc< CLASS > Struct Template Reference . . . 963
12.170.1 Detailed Description . . . 964
12.171 L4::Typeid::Rpc_nocode< OPERATION > Struct Template Reference . . . 964
12.171.1 Detailed Description . . . 966
12.172 L4::Typeid::Rpcs< RPCS > Struct Template Reference . . . 967
12.172.1 Detailed Description . . . 968
12.173 L4::Typeid::Rpcs_code< OPCODE_TYPE > Struct Template Reference . . . 969
12.173.1 Detailed Description . . . 969
12.174 L4::Typeid::Rpcs_code< OPCODE_TYPE >::F< RPCS > Struct Template Reference . . . 969
12.174.1 Detailed Description . . . 971
12.175 L4::Typeid::Rpcs_sys< ARG > Struct Template Reference . . . 972
12.175.1 Detailed Description . . . 973
12.176 L4::Types::Bool< V > Struct Template Reference . . . 974
12.176.1 Detailed Description . . . 974
12.177 L4::Types::False Struct Reference . . . 975
12.177.1 Detailed Description . . . 976
12.178 L4::Types::Flags< BITS_ENUM, UNDERLYING > Class Template Reference . . . 976
12.178.1 Detailed Description . . . 978
12.178.2 Member Enumeration Documentation . . . 979
12.178.2.1 None_type . . . 979
12.178.3 Constructor & Destructor Documentation . . . 979

12.178.3.1 Flags . . . 979
12.178.3.2 Flags . . . 979
12.178.4 Member Function Documentation . . . 979
12.178.4.1 clear . . . 979
12.178.4.2 from_raw . . . 980
12.179 L4::Types::Same< A, B > Struct Template Reference . . . 980
12.179.1 Detailed Description . . . 982
12.180 L4::Types::True Struct Reference . . . 983
12.180.1 Detailed Description . . . 984
12.181 L4::Unknown_error Class Reference . . . 984
12.181.1 Detailed Description . . . 986
12.182 L4::Vcon Class Reference . . . 987
12.182.1 Detailed Description . . . 989
12.182.2 Member Function Documentation . . . 989
12.182.2.1 get_attr . . . 989
12.182.2.2 read . . . 990
12.182.2.3 read_with_flags . . . 991
12.182.2.4 send . . . 992
12.182.2.5 set_attr . . . 993
12.182.2.6 write . . . 994
12.183 L4::Vm Class Reference . . . 995
12.183.1 Detailed Description . . . 996
12.184 l4_buf_regs_t Struct Reference . . . 997
12.184.1 Detailed Description . . . 997
12.185 l4_exc_regs_t Struct Reference . . . 997
12.185.1 Detailed Description . . . 1000
12.185.2 Field Documentation . . . 1000
12.185.2.1 flags . . . 1000
12.186 l4_fpage_t Union Reference . . . 1000
12.186.1 Detailed Description . . . 1000
12.187 l4_icu_info_t Struct Reference . . . 1001
12.187.1 Detailed Description . . . 1002
12.187.2 Field Documentation . . . 1002
12.187.2.1 features . . . 1002
12.188 l4_icu_msi_info_t Struct Reference . . . 1002
12.188.1 Detailed Description . . . 1003
12.188.2 Field Documentation . . . 1003
12.188.2.1 msi_data . . . 1003
12.189 l4_kernel_info_mem_desc_t Struct Reference . . . 1003
12.189.1 Detailed Description . . . 1003

12.190 l4_msg_regs_t Union Reference . . . 1004
12.190.1 Detailed Description . . . 1004
12.191 l4_msgtag_t Struct Reference . . . 1004
12.191.1 Detailed Description . . . 1006
12.191.2 Member Function Documentation . . . 1006
12.191.2.1 flags . . . 1006
12.192 l4_sched_cpu_set_t Struct Reference . . . 1006
12.192.1 Detailed Description . . . 1007
12.192.2 Member Function Documentation . . . 1007
12.192.2.1 granularity . . . 1007
12.192.2.2 offset . . . 1008
12.192.3 Field Documentation . . . 1008
12.192.3.1 gran_offset . . . 1008
12.193 l4_sched_param_t Struct Reference . . . 1008
12.193.1 Detailed Description . . . 1009
12.194 l4_snd_fpage_t Struct Reference . . . 1009
12.194.1 Detailed Description . . . 1010
12.195 l4_thread_regs_t Struct Reference . . . 1010
12.195.1 Detailed Description . . . 1011
12.196 l4_timeout_s Struct Reference . . . 1012
12.196.1 Detailed Description . . . 1012
12.197 l4_timeout_t Union Reference . . . 1012
12.197.1 Detailed Description . . . 1013
12.198 l4_tracebuffer_status_t Struct Reference . . . 1013
12.198.1 Detailed Description . . . 1015
12.198.2 Field Documentation . . . 1015
12.198.2.1 cnt_iobmap_tlb_flush . . . 1015
12.199 l4_tracebuffer_status_window_t Struct Reference . . . 1015
12.199.1 Detailed Description . . . 1016
12.200 l4_vcon_attr_t Struct Reference . . . 1016
12.200.1 Detailed Description . . . 1017
12.201 l4_vcpu_ipc_regs_t Struct Reference . . . 1017
12.201.1 Detailed Description . . . 1018
12.202 l4_vcpu_regs_t Struct Reference . . . 1018
12.202.1 Detailed Description . . . 1020
12.202.2 Field Documentation . . . 1020
12.202.2.1 ax . . . 1020
12.202.2.2 bp . . . 1020
12.202.2.3 bx . . . 1021
12.202.2.4 cx . . . 1021

12.202.2.5 di . . . 1021
12.202.2.6 dx . . . 1021
12.202.2.7 si . . . 1021
12.203 l4_vcpu_state_t Struct Reference . . . 1021
12.203.1 Detailed Description . . . 1024
12.204 l4_vhw_descriptor Struct Reference . . . 1024
12.204.1 Detailed Description . . . 1025
12.204.2 Field Documentation . . . 1026
12.204.2.1 count . . . 1026
12.204.2.2 descs . . . 1026
12.204.2.3 magic . . . 1026
12.204.2.4 version . . . 1026
12.205 l4_vhw_entry Struct Reference . . . 1026
12.205.1 Detailed Description . . . 1027
12.205.2 Field Documentation . . . 1027
12.205.2.1 fd . . . 1027
12.205.2.2 irq_no . . . 1028
12.205.2.3 mem_size . . . 1028
12.205.2.4 mem_start . . . 1028
12.205.2.5 provider_pid . . . 1028
12.205.2.6 type . . . 1028
12.206 l4_vm_svm_vmcb_control_area Struct Reference . . . 1029
12.206.1 Detailed Description . . . 1029
12.207 l4_vm_svm_vmcb_state_save_area Struct Reference . . . 1029
12.207.1 Detailed Description . . . 1030
12.208 l4_vm_svm_vmcb_state_save_area_seg Struct Reference . . . 1031
12.208.1 Detailed Description . . . 1031
12.209 l4_vm_svm_vmcb_t Struct Reference . . . 1031
12.209.1 Detailed Description . . . 1032
12.210 l4_vm_tz_state Struct Reference . . . 1033
12.210.1 Detailed Description . . . 1033
12.211 L4Re::Cap_alloc Class Reference . . . 1033
12.211.1 Detailed Description . . . 1034
12.211.2 Member Function Documentation . . . 1034
12.211.2.1 alloc . . . 1034
12.211.2.2 alloc . . . 1035
12.211.2.3 free . . . 1035
12.211.2.4 get_cap_alloc . . . 1036
12.212 L4Re::Console Class Reference . . . 1036
12.212.1 Detailed Description . . . 1038

12.213 L4Re::Dataspace Class Reference . . . 1039
12.213.1 Detailed Description . . . 1041
12.213.2 Member Enumeration Documentation . . . 1041
12.213.2.1 Map_flags . . . 1041
12.213.3 Member Function Documentation . . . 1042
12.213.3.1 allocate . . . 1042
12.213.3.2 clear . . . 1042
12.213.3.3 copy_in . . . 1043
12.213.3.4 flags . . . 1044
12.213.3.5 info . . . 1045
12.213.3.6 map . . . 1046
12.213.3.7 map_region . . . 1047
12.213.3.8 phys . . . 1048
12.213.3.9 size . . . 1048
12.214 L4Re::Dataspace::Stats Struct Reference . . . 1049
12.214.1 Detailed Description . . . 1049
12.215 L4Re::Debug_obj Class Reference . . . 1050
12.215.1 Detailed Description . . . 1051
12.215.2 Member Function Documentation . . . 1051
12.215.2.1 debug . . . 1052
12.216 L4Re::Dma_space Class Reference . . . 1053
12.216.1 Detailed Description . . . 1054
12.216.2 Member Typedef Documentation . . . 1054
12.216.2.1 Attributes . . . 1054
12.216.3 Member Enumeration Documentation . . . 1054
12.216.3.1 Attribute . . . 1054
12.216.3.2 Direction . . . 1055
12.216.3.3 Space_attrib . . . 1055
12.216.4 Member Function Documentation . . . 1055
12.216.4.1 associate . . . 1055
12.216.4.2 disassociate . . . 1056
12.216.4.3 map . . . 1056
12.216.4.4 unmap . . . 1056
12.217 L4Re::Env Class Reference . . . 1057
12.217.1 Detailed Description . . . 1059
12.217.2 Member Function Documentation . . . 1059
12.217.2.1 env . . . 1059
12.217.2.2 factory . . . 1059
12.217.2.3 factory . . . 1060
12.217.2.4 first_free_cap . . . 1061

12.217.2.5 first_free_cap . . . 1061
12.217.2.6 first_free_utcb . . . 1061
12.217.2.7 first_free_utcb . . . 1061
12.217.2.8 get . . . 1061
12.217.2.9 get_cap . . . 1062
12.217.2.10 get_cap . . . 1062
12.217.2.11 initial_caps . . . 1063
12.217.2.12 initial_caps . . . 1063
12.217.2.13 log . . . 1063
12.217.2.14 log . . . 1063
12.217.2.15 main_thread . . . 1063
12.217.2.16 main_thread . . . 1063
12.217.2.17 mem_alloc . . . 1064
12.217.2.18 mem_alloc . . . 1064
12.217.2.19 parent . . . 1064
12.217.2.20 parent . . . 1064
12.217.2.21 rm . . . 1065
12.217.2.22 rm . . . 1065
12.217.2.23 scheduler . . . 1065
12.217.2.24 scheduler . . . 1065
12.217.2.25 task . . . 1065
12.217.2.26 utcb_area . . . 1066
12.217.2.27 utcb_area . . . 1066
12.218 L4Re::Event Class Reference . . . 1066
12.218.1 Detailed Description . . . 1069
12.218.2 Member Function Documentation . . . 1069
12.218.2.1 get_buffer . . . 1069
12.219 L4Re::Event_buffer_t< PAYLOAD > Class Template Reference . . . 1069
12.219.1 Detailed Description . . . 1071
12.219.2 Constructor & Destructor Documentation . . . 1071
12.219.2.1 Event_buffer_t . . . 1071
12.219.3 Member Function Documentation . . . 1072
12.219.3.1 next . . . 1072
12.219.3.2 put . . . 1072
12.220 L4Re::Event_buffer_t< PAYLOAD >::Event Struct Reference . . . 1072
12.220.1 Detailed Description . . . 1073
12.221 L4Re::Inhibitor Class Reference . . . 1073
12.221.1 Detailed Description . . . 1075
12.221.2 Member Enumeration Documentation . . . 1076
12.221.2.1 anonymous enum . . . 1076

12.221.3 Member Function Documentation . . . 1076
12.221.3.1 acquire . . . 1076
12.221.3.2 next_lock_info . . . 1076
12.221.3.3 release . . . 1077
12.222 L4Re::Log Class Reference . . . 1077
12.222.1 Detailed Description . . . 1080
12.222.2 Member Function Documentation . . . 1080
12.222.2.1 print . . . 1080
12.222.2.2 printn . . . 1080
12.223 L4Re::Mem_alloc Class Reference . . . 1080
12.223.1 Detailed Description . . . 1083
12.223.2 Member Enumeration Documentation . . . 1083
12.223.2.1 Mem_alloc_flags . . . 1083
12.223.3 Member Function Documentation . . . 1083
12.223.3.1 alloc . . . 1083
12.223.3.2 free . . . 1084
12.224 L4Re::Namespace Class Reference . . . 1084
12.224.1 Detailed Description . . . 1087
12.224.2 Member Function Documentation . . . 1087
12.224.2.1 query . . . 1087
12.224.2.2 query . . . 1087
12.224.2.3 register_obj . . . 1088
12.224.2.4 unlink . . . 1089
12.225 L4Re::Parent Class Reference . . . 1089
12.225.1 Detailed Description . . . 1091
12.225.2 Member Function Documentation . . . 1091
12.225.2.1 signal . . . 1092
12.226 L4Re::Rm Class Reference . . . 1093
12.226.1 Detailed Description . . . 1096
12.226.2 Member Function Documentation . . . 1097
12.226.2.1 attach . . . 1097
12.226.2.2 attach . . . 1098
12.226.2.3 detach . . . 1099
12.226.2.4 detach . . . 1099
12.226.2.5 detach . . . 1100
12.226.2.6 find . . . 1100
12.226.2.7 free_area . . . 1101
12.226.2.8 reserve_area . . . 1101
12.226.2.9 reserve_area . . . 1102
12.227 L4Re::Smart_cap_auto< Unmap_flags > Class Template Reference . . . 1103

12.227.1 Detailed Description . . . 1103
12.228 L4Re::Util::Auto_cap< T > Struct Template Reference . . . 1103
12.228.1 Detailed Description . . . 1104
12.229 L4Re::Util::Auto_del_cap< T > Struct Template Reference . . . 1105
12.229.1 Detailed Description . . . 1106
12.230 L4Re::Util::Br_manager Class Reference . . . 1106
12.230.1 Detailed Description . . . 1109
12.230.2 Member Function Documentation . . . 1109
12.230.2.1 alloc_buffer_demand . . . 1109
12.230.2.2 get_rcv_cap . . . 1109
12.230.2.3 realloc_rcv_cap . . . 1110
12.230.2.4 set_rcv_cap_flags . . . 1110
12.231 L4Re::Util::Br_manager_hooks Struct Reference . . . 1111
12.231.1 Detailed Description . . . 1112
12.232 L4Re::Util::Br_manager_timeout_hooks Struct Reference . . . 1112
12.232.1 Detailed Description . . . 1114
12.233 L4Re::Util::Cap_alloc_base Class Reference . . . 1115
12.233.1 Detailed Description . . . 1115
12.234 L4Re::Util::Counting_cap_alloc< COUNTERTYPE > Class Template Reference . . . 1116
12.234.1 Detailed Description . . . 1117
12.234.2 Constructor & Destructor Documentation . . . 1117
12.234.2.1 Counting_cap_alloc . . . 1117
12.234.3 Member Function Documentation . . . 1117
12.234.3.1 alloc . . . 1117
12.234.3.2 alloc . . . 1118
12.234.3.3 free . . . 1118
12.234.3.4 release . . . 1119
12.234.3.5 setup . . . 1120
12.234.3.6 take . . . 1120
12.235 L4Re::Util::Dataspace_svr Class Reference . . . 1120
12.235.1 Detailed Description . . . 1122
12.235.2 Member Function Documentation . . . 1122
12.235.2.1 allocate . . . 1122
12.235.2.2 clear . . . 1122
12.235.2.3 copy . . . 1122
12.235.2.4 is_static . . . 1123
12.235.2.5 map . . . 1123
12.235.2.6 map_hook . . . 1123
12.235.2.7 page_shift . . . 1124
12.235.2.8 phys . . . 1124

12.235.2.9 release . . . 1124
12.235.2.10 take . . . 1124
12.236 L4Re::Util::Event_buffer_consumer_t< PAYLOAD > Class Template Reference . . . 1125
12.236.1 Detailed Description . . . 1126
12.236.2 Member Function Documentation . . . 1127
12.236.2.1 foreach_available_event . . . 1127
12.236.2.2 process . . . 1127
12.237 L4Re::Util::Event_buffer_t< PAYLOAD > Class Template Reference . . . 1128
12.237.1 Detailed Description . . . 1130
12.237.2 Member Function Documentation . . . 1130
12.237.2.1 attach . . . 1130
12.237.2.2 buf . . . 1131
12.237.2.3 detach . . . 1131
12.238 L4Re::Util::Event_t< PAYLOAD > Class Template Reference . . . 1132
12.238.1 Detailed Description . . . 1132
12.238.2 Member Enumeration Documentation . . . 1133
12.238.2.1 Mode . . . 1133
12.238.3 Member Function Documentation . . . 1133
12.238.3.1 buffer . . . 1133
12.238.3.2 init . . . 1133
12.238.3.3 init_poll . . . 1134
12.238.3.4 irq . . . 1134
12.239 L4Re::Util::Item_alloc_base Class Reference . . . 1135
12.239.1 Detailed Description . . . 1135
12.240 L4Re::Util::Names::Name Class Reference . . . 1135
12.240.1 Detailed Description . . . 1137
12.241 L4Re::Util::Names::Name_space Class Reference . . . 1137
12.241.1 Detailed Description . . . 1138
12.241.2 Member Function Documentation . . . 1139
12.241.2.1 alloc_dynamic_entry . . . 1139
12.241.2.2 copy_receive_cap . . . 1139
12.241.2.3 free_capability . . . 1139
12.241.2.4 free_dynamic_entry . . . 1139
12.241.2.5 free_epiface . . . 1139
12.241.2.6 get_epiface . . . 1140
12.242 L4Re::Util::Object_registry Class Reference . . . 1141
12.242.1 Detailed Description . . . 1143
12.242.2 Constructor & Destructor Documentation . . . 1143
12.242.2.1 Object_registry . . . 1143
12.242.2.2 Object_registry . . . 1143

12.242.3 Member Function Documentation . . . 1143
12.242.3.1 register_irq_obj . . . 1143
12.242.3.2 register_irq_obj . . . 1143
12.242.3.3 register_obj . . . 1144
12.242.3.4 register_obj . . . 1144
12.242.3.5 unregister_obj . . . 1144
12.243 L4Re::Util::Ref_cap< T > Struct Template Reference . . . 1145
12.243.1 Detailed Description . . . 1145
12.244 L4Re::Util::Ref_del_cap< T > Struct Template Reference . . . 1146
12.244.1 Detailed Description . . . 1146
12.245 L4Re::Util::Registry_server< LOOP_HOOKS > Class Template Reference . . . 1147
12.245.1 Detailed Description . . . 1148
12.245.2 Constructor & Destructor Documentation . . . 1149
12.245.2.1 Registry_server . . . 1149
12.245.2.2 Registry_server . . . 1149
12.245.3 Member Function Documentation . . . 1149
12.245.3.1 registry . . . 1149
12.245.3.2 registry . . . 1149
12.246 L4Re::Util::Smart_cap_auto< Unmap_flags > Class Template Reference . . . 1149
12.246.1 Detailed Description . . . 1150
12.247 L4Re::Util::Smart_count_cap< Unmap_flags > Class Template Reference . . . 1150
12.247.1 Detailed Description . . . 1151
12.248 L4Re::Util::Vcon_svr< SVR > Class Template Reference . . . 1151
12.248.1 Detailed Description . . . 1152
12.249 L4Re::Util::Video::Goos_svr Class Reference . . . 1152
12.249.1 Detailed Description . . . 1154
12.249.2 Member Function Documentation . . . 1154
12.249.2.1 get_fb . . . 1154
12.249.2.2 init_infos . . . 1154
12.249.2.3 refresh . . . 1154
12.249.2.4 screen_info . . . 1155
12.249.2.5 view_info . . . 1155
12.250 L4Re::Vfs::Be_file Class Reference . . . 1155
12.250.1 Detailed Description . . . 1158
12.250.2 Member Function Documentation . . . 1159
12.250.2.1 data_space . . . 1159
12.250.2.2 fstat64 . . . 1159
12.250.2.3 unlock_all_locks . . . 1159
12.251 L4Re::Vfs::Be_file_system Class Reference . . . 1160
12.251.1 Detailed Description . . . 1161

12.251.2 Constructor & Destructor Documentation . . . 1161
12.251.2.1 Be_file_system . . . 1162
12.251.2.2 ~Be_file_system . . . 1163
12.251.3 Member Function Documentation . . . 1163
12.251.3.1 type . . . 1163
12.252 L4Re::Vfs::Directory Class Reference . . . 1163
12.252.1 Detailed Description . . . 1165
12.252.2 Member Function Documentation . . . 1165
12.252.2.1 faccessat . . . 1166
12.252.2.2 link . . . 1166
12.252.2.3 mkdir . . . 1166
12.252.2.4 rename . . . 1167
12.252.2.5 rmdir . . . 1168
12.252.2.6 symlink . . . 1168
12.252.2.7 unlink . . . 1168
12.253 L4Re::Vfs::File Class Reference . . . 1169
12.253.1 Detailed Description . . . 1170
12.254 L4Re::Vfs::File_system Class Reference . . . 1170
12.254.1 Detailed Description . . . 1172
12.254.2 Member Function Documentation . . . 1172
12.254.2.1 mount . . . 1172
12.254.2.2 type . . . 1173
12.255 L4Re::Vfs::Fs Class Reference . . . 1173
12.255.1 Detailed Description . . . 1174
12.255.2 Member Function Documentation . . . 1175
12.255.2.1 alloc_fd . . . 1175
12.255.2.2 free_fd . . . 1175
12.255.2.3 get_file . . . 1175
12.255.2.4 mount . . . 1175
12.255.2.5 set_fd . . . 1176
12.256 L4Re::Vfs::Generic_file Class Reference . . . 1176
12.256.1 Detailed Description . . . 1178
12.256.2 Member Function Documentation . . . 1178
12.256.2.1 fchmod . . . 1178
12.256.2.2 fstat64 . . . 1179
12.256.2.3 get_status_flags . . . 1179
12.256.2.4 set_status_flags . . . 1179
12.256.2.5 unlock_all_locks . . . 1179
12.257 L4Re::Vfs::Mman Class Reference . . . 1180
12.257.1 Detailed Description . . . 1181

12.258 L4Re::Vfs::Ops Class Reference . . . 1182
12.258.1 Detailed Description . . . 1183
12.259 L4Re::Vfs::Regular_file Class Reference . . . 1183
12.259.1 Detailed Description . . . 1185
12.259.2 Member Function Documentation . . . 1186
12.259.2.1 data_space . . . 1186
12.259.2.2 fdatasync . . . 1186
12.259.2.3 fsync . . . 1186
12.259.2.4 ftruncate64 . . . 1186
12.259.2.5 get_lock . . . 1186
12.259.2.6 lseek64 . . . 1187
12.259.2.7 readv . . . 1187
12.259.2.8 set_lock . . . 1187
12.259.2.9 writev . . . 1187
12.260 L4Re::Vfs::Special_file Class Reference . . . 1188
12.260.1 Detailed Description . . . 1190
12.260.2 Member Function Documentation . . . 1190
12.260.2.1 ioctl . . . 1190
12.261 L4Re::Video::Color_component Class Reference . . . 1190
12.261.1 Detailed Description . . . 1191
12.261.2 Constructor & Destructor Documentation . . . 1191
12.261.2.1 Color_component . . . 1191
12.261.3 Member Function Documentation . . . 1192
12.261.3.1 dump . . . 1192
12.261.3.2 get . . . 1192
12.261.3.3 operator== . . . 1192
12.261.3.4 set . . . 1192
12.261.3.5 shift . . . 1193
12.261.3.6 size . . . 1193
12.262 L4Re::Video::Goos Class Reference . . . 1193
12.262.1 Detailed Description . . . 1196
12.262.2 Member Enumeration Documentation . . . 1196
12.262.2.1 Flags . . . 1196
12.262.3 Member Function Documentation . . . 1196
12.262.3.1 create_buffer . . . 1196
12.262.3.2 create_view . . . 1197
12.262.3.3 delete_buffer . . . 1197
12.262.3.4 delete_view . . . 1197
12.262.3.5 get_static_buffer . . . 1197
12.262.3.6 info . . . 1198

12.262.3.7 view . . . 1198
12.263 L4Re::Video::Goos::Info Struct Reference . . . 1198
12.263.1 Detailed Description . . . 1200
12.263.2 Member Function Documentation . . . 1200
12.263.2.1 auto_refresh . . . 1200
12.264 L4Re::Video::Pixel_info Class Reference . . . 1200
12.264.1 Detailed Description . . . 1201
12.264.2 Constructor & Destructor Documentation . . . 1201
12.264.2.1 Pixel_info . . . 1201
12.264.2.2 Pixel_info . . . 1202
12.264.3 Member Function Documentation . . . 1202
12.264.3.1 a . . . 1202
12.264.3.2 a . . . 1202
12.264.3.3 b . . . 1202
12.264.3.4 b . . . 1203
12.264.3.5 bits_per_pixel . . . 1204
12.264.3.6 bytes_per_pixel . . . 1204
12.264.3.7 bytes_per_pixel . . . 1204
12.264.3.8 dump . . . 1204
12.264.3.9 g . . . 1205
12.264.3.10 g . . . 1205
12.264.3.11 has_alpha . . . 1205
12.264.3.12 operator== . . . 1206
12.264.3.13 r . . . 1206
12.264.3.14 r . . . 1206
12.265 L4Re::Video::View Class Reference . . . 1206
12.265.1 Detailed Description . . . 1208
12.265.2 Member Enumeration Documentation . . . 1208
12.265.2.1 Flags . . . 1208
12.265.2.2 V_flags . . . 1208
12.265.3 Member Function Documentation . . . 1208
12.265.3.1 info . . . 1208
12.265.3.2 refresh . . . 1209
12.265.3.3 set_info . . . 1209
12.265.3.4 set_viewport . . . 1209
12.265.3.5 stack . . . 1210
12.266 L4Re::Video::View::Info Struct Reference . . . 1210
12.266.1 Detailed Description . . . 1212
12.267 L4re_aux_t Struct Reference . . . 1212
12.267.1 Detailed Description . . . 1213

12.268 L4re_ds_stats_t Struct Reference . . . 1213
12.268.1 Detailed Description . . . 1214
12.269 L4re_elf_aux_mword_t Struct Reference . . . 1214
12.269.1 Detailed Description . . . 1215
12.270 L4re_elf_aux_t Struct Reference . . . 1215
12.270.1 Detailed Description . . . 1215
12.271 L4re_elf_aux_vma_t Struct Reference . . . 1215
12.271.1 Detailed Description . . . 1216
12.272 L4re_env_cap_entry_t Struct Reference . . . 1216
12.272.1 Detailed Description . . . 1217
12.272.2 Constructor & Destructor Documentation . . . 1217
12.272.2.1 L4re_env_cap_entry_t . . . 1217
12.272.3 Field Documentation . . . 1217
12.272.3.1 flags . . . 1217
12.273 L4re_env_t Struct Reference . . . 1218
12.273.1 Detailed Description . . . 1219
12.274 L4re_event_t Struct Reference . . . 1219
12.274.1 Detailed Description . . . 1220
12.275 L4re_video_color_component_t Struct Reference . . . 1220
12.275.1 Detailed Description . . . 1220
12.276 L4re_video_goos_info_t Struct Reference . . . 1221
12.276.1 Detailed Description . . . 1222
12.277 L4re_video_pixel_info_t Struct Reference . . . 1222
12.277.1 Detailed Description . . . 1223
12.278 L4re_video_view_info_t Struct Reference . . . 1223
12.278.1 Detailed Description . . . 1224
12.279 L4re_video_view_t Struct Reference . . . 1224
12.279.1 Detailed Description . . . 1224
12.280 L4util_idt_desc_t Struct Reference . . . 1225
12.280.1 Detailed Description . . . 1225
12.281 L4util_idt_header_t Struct Reference . . . 1225
12.281.1 Detailed Description . . . 1226
12.282 L4util_mb_addr_range_t Struct Reference . . . 1226
12.282.1 Detailed Description . . . 1227
12.283 L4util_mb_apm_t Struct Reference . . . 1227
12.283.1 Detailed Description . . . 1228
12.284 L4util_mb_drive_t Struct Reference . . . 1228
12.284.1 Detailed Description . . . 1229
12.284.2 Field Documentation . . . 1229
12.284.2.1 drive_cylinders . . . 1229

12.284.2.2 drive_mode . . . 1229
12.284.2.3 drive_number . . . 1229
12.285 L4util_mb_info_t Struct Reference . . . 1229
12.285.1 Detailed Description . . . 1231
12.286 L4util_mb_mod_t Struct Reference . . . 1231
12.286.1 Detailed Description . . . 1232
12.286.2 Field Documentation . . . 1232
12.286.2.1 mod_end . . . 1232
12.286.2.2 mod_start . . . 1232
12.287 L4util_mb_vbe_ctrl_t Struct Reference . . . 1232
12.287.1 Detailed Description . . . 1233
12.288 L4util_mb_vbe_mode_t Struct Reference . . . 1233
12.288.1 Detailed Description . . . 1235
12.289 L4vbus::Device Class Reference . . . 1236
12.289.1 Detailed Description . . . 1238
12.289.2 Member Function Documentation . . . 1238
12.289.2.1 bus_cap . . . 1238
12.289.2.2 dev_handle . . . 1239
12.289.2.3 device_by_hid . . . 1239
12.289.2.4 get_resource . . . 1240
12.289.2.5 is_compatible . . . 1241
12.289.2.6 next_device . . . 1241
12.289.2.7 operator"!= . . . 1242
12.289.2.8 operator== . . . 1242
12.289.3 Field Documentation . . . 1242
12.289.3.1 _bus . . . 1242
12.290 L4vbus::Gpio_module Class Reference . . . 1243
12.290.1 Detailed Description . . . 1246
12.290.2 Member Function Documentation . . . 1246
12.290.2.1 config_pad . . . 1246
12.290.2.2 get . . . 1247
12.290.2.3 pin . . . 1248
12.290.2.4 set . . . 1248
12.290.2.5 setup . . . 1249
12.291 L4vbus::Gpio_module::Pin_slice Struct Reference . . . 1249
12.291.1 Detailed Description . . . 1250
12.292 L4vbus::Gpio_pin Class Reference . . . 1250
12.292.1 Detailed Description . . . 1253
12.292.2 Member Function Documentation . . . 1253
12.292.2.1 config_get . . . 1253

12.292.2.2 config_pad . . . 1253
12.292.2.3 config_pull . . . 1254
12.292.2.4 get . . . 1254
12.292.2.5 pin . . . 1255
12.292.2.6 set . . . 1255
12.292.2.7 setup . . . 1256
12.292.2.8 to_irq . . . 1257
12.293 L4vbus::Icu Class Reference . . . 1257
12.293.1 Detailed Description . . . 1260
12.294 L4vbus::Pci_dev Class Reference . . . 1260
12.294.1 Detailed Description . . . 1263
12.294.2 Member Function Documentation . . . 1263
12.294.2.1 cfg_read . . . 1263
12.294.2.2 cfg_write . . . 1263
12.294.2.3 irq_enable . . . 1264
12.295 L4vbus::Pci_host_bridge Class Reference . . . 1265
12.295.1 Detailed Description . . . 1267
12.295.2 Member Function Documentation . . . 1267
12.295.2.1 cfg_read . . . 1267
12.295.2.2 cfg_write . . . 1267
12.295.2.3 irq_enable . . . 1268
12.296 L4vbus::Pm< DEC > Class Template Reference . . . 1269
12.296.1 Detailed Description . . . 1271
12.297 L4vbus::Vbus Class Reference . . . 1271
12.297.1 Detailed Description . . . 1274
12.297.2 Member Function Documentation . . . 1274
12.297.2.1 release_resource . . . 1274
12.297.2.2 request_resource . . . 1274
12.297.2.3 root . . . 1275
12.298 L4vbus_device_t Struct Reference . . . 1275
12.298.1 Detailed Description . . . 1276
12.299 L4vbus_resource_t Struct Reference . . . 1276
12.299.1 Detailed Description . . . 1277
12.300 L4vcpu::State Class Reference . . . 1277
12.300.1 Detailed Description . . . 1277
12.300.2 Constructor & Destructor Documentation . . . 1277
12.300.2.1 State . . . 1277
12.300.3 Member Function Documentation . . . 1278
12.300.3.1 add . . . 1278
12.300.3.2 clear . . . 1278

12.300.3.3 set . . . 1278
12.301 L4vcpu::Vcpu Class Reference . . . 1278
12.301.1 Detailed Description . . . 1282
12.301.2 Member Function Documentation . . . 1282
12.301.2.1 cast . . . 1282
12.301.2.2 cast . . . 1282
12.301.2.3 entry_ip . . . 1282
12.301.2.4 entry_sp . . . 1282
12.301.2.5 ext_alloc . . . 1283
12.301.2.6 i . . . 1283
12.301.2.7 i . . . 1283
12.301.2.8 irq_disable_save . . . 1283
12.301.2.9 irq_enable . . . 1284
12.301.2.10 irq_restore . . . 1284
12.301.2.11 is_irq_entry . . . 1285
12.301.2.12 is_page_fault_entry . . . 1285
12.301.2.13 i3 . . . 1286
12.301.2.14 i4 . . . 1286
12.301.2.15 saved_state . . . 1286
12.301.2.16 saved_state . . . 1286
12.301.2.17 state . . . 1287
12.301.2.18 state . . . 1287
12.301.2.19 task . . . 1287
12.301.2.20 wait_for_event . . . 1287
12.302 L4virtio::Driver::Virtqueue Class Reference . . . 1288
12.302.1 Detailed Description . . . 1291
12.302.2 Member Function Documentation . . . 1291
12.302.2.1 alloc_descriptor . . . 1291
12.302.2.2 desc . . . 1291
12.302.2.3 enqueue_descriptor . . . 1291
12.302.2.4 find_next_used . . . 1292
12.302.2.5 free_descriptor . . . 1292
12.302.2.6 init_queue . . . 1292
12.303 L4virtio::Ptr< T > Class Template Reference . . . 1293
12.303.1 Detailed Description . . . 1294
12.303.2 Member Enumeration Documentation . . . 1294
12.303.2.1 Invalid_type . . . 1294
12.303.3 Member Function Documentation . . . 1294
12.303.3.1 get . . . 1294
12.303.3.2 is_valid . . . 1295

12.304 L4virtio::Svr::Bad_descriptor Struct Reference . . . 1295
12.304.1 Detailed Description . . . 1296
12.304.2 Constructor & Destructor Documentation . . . 1297
12.304.2.1 Bad_descriptor . . . 1297
12.305 L4virtio::Svr::Data_buffer Struct Reference . . . 1298
12.305.1 Detailed Description . . . 1299
12.305.2 Constructor & Destructor Documentation . . . 1299
12.305.2.1 Data_buffer . . . 1299
12.305.3 Member Function Documentation . . . 1299
12.305.3.1 copy_to . . . 1299
12.305.3.2 done . . . 1300
12.305.3.3 set . . . 1300
12.305.3.4 skip . . . 1300
12.306 L4virtio::Svr::Dev_config Class Reference . . . 1301
12.306.1 Detailed Description . . . 1302
12.306.2 Constructor & Destructor Documentation . . . 1302
12.306.2.1 Dev_config . . . 1302
12.306.3 Member Function Documentation . . . 1303
12.306.3.1 change_queue_config . . . 1303
12.306.3.2 ds . . . 1303
12.306.3.3 hdr . . . 1303
12.306.3.4 qconfig . . . 1304
12.306.3.5 reset_queue . . . 1305
12.306.3.6 set_failed . . . 1306
12.306.3.7 set_status . . . 1307
12.306.3.8 status . . . 1308
12.307 L4virtio::Svr::Dev_features Struct Reference . . . 1308
12.307.1 Detailed Description . . . 1309
12.307.2 Member Typedef Documentation . . . 1310
12.307.2.1 ring_event_idx_bfm_t . . . 1310
12.307.2.2 ring_indirect_desc_bfm_t . . . 1310
12.307.3 Member Function Documentation . . . 1310
12.307.3.1 ring_event_idx . . . 1310
12.307.3.2 ring_event_idx . . . 1310
12.307.3.3 ring_indirect_desc . . . 1310
12.307.3.4 ring_indirect_desc . . . 1310
12.308 L4virtio::Svr::Dev_status Struct Reference . . . 1310
12.308.1 Detailed Description . . . 1312
12.308.2 Member Typedef Documentation . . . 1312
12.308.2.1 acked_bfm_t . . . 1312

12.308.2.2 driver_bfm_t . . . 1312
12.308.2.3 driver_ok_bfm_t . . . 1312
12.308.2.4 failed_bfm_t . . . 1313
12.308.2.5 feature_ok_bfm_t . . . 1313
12.308.3 Member Function Documentation . . . 1313
12.308.3.1 acked . . . 1313
12.308.3.2 acked . . . 1313
12.308.3.3 driver . . . 1313
12.308.3.4 driver . . . 1313
12.308.3.5 driver_ok . . . 1313
12.308.3.6 driver_ok . . . 1313
12.308.3.7 failed . . . 1313
12.308.3.8 failed . . . 1314
12.308.3.9 feature_ok . . . 1314
12.308.3.10 feature_ok . . . 1314
12.308.3.11 running . . . 1314
12.309 L4virtio::Svr::Device_t< DATA > Class Template Reference . . . 1314
12.309.1 Detailed Description . . . 1316
12.309.2 Member Function Documentation . . . 1316
12.309.2.1 device_error . . . 1316
12.309.2.2 init_mem_info . . . 1316
12.309.2.3 reset_queue_config . . . 1317
12.309.2.4 setup_queue . . . 1317
12.310 L4virtio::Svr::Driver_mem_list_t< DATA > Class Template Reference . . . 1318
12.310.1 Detailed Description . . . 1320
12.310.2 Member Function Documentation . . . 1320
12.310.2.1 add . . . 1320
12.310.2.2 find . . . 1320
12.310.2.3 find . . . 1321
12.310.2.4 full . . . 1321
12.310.2.5 init . . . 1322
12.310.2.6 load_desc . . . 1323
12.310.2.7 load_desc . . . 1324
12.310.2.8 load_desc . . . 1325
12.310.2.9 remove . . . 1325
12.311 L4virtio::Svr::Driver_mem_region_t< DATA > Class Template Reference . . . 1326
12.311.1 Detailed Description . . . 1327
12.311.2 Constructor & Destructor Documentation . . . 1327
12.311.2.1 Driver_mem_region_t . . . 1327
12.311.3 Member Function Documentation . . . 1328

12.311.3.1 contains . . . 1328
12.311.3.2 drv_base . . . 1328
12.311.3.3 ds . . . 1329
12.311.3.4 ds_offset . . . 1329
12.311.3.5 empty . . . 1329
12.311.3.6 flags . . . 1329
12.311.3.7 is_writable . . . 1329
12.311.3.8 local . . . 1330
12.311.3.9 local_base . . . 1330
12.311.3.10 size . . . 1331
12.312 L4virtio::Svr::Request_processor Class Reference . . . 1331
12.312.1 Detailed Description . . . 1332
12.312.2 Member Function Documentation . . . 1332
12.312.2.1 current_flags . . . 1332
12.312.2.2 has_more . . . 1332
12.312.2.3 next . . . 1333
12.312.2.4 start . . . 1334
12.312.2.5 start . . . 1335
12.313 L4virtio::Svr::Virtqueue Class Reference . . . 1336
12.313.1 Detailed Description . . . 1338
12.313.2 Member Function Documentation . . . 1338
12.313.2.1 consumed . . . 1338
12.313.2.2 desc . . . 1338
12.313.2.3 desc_avail . . . 1339
12.313.2.4 disable_notify . . . 1339
12.313.2.5 enable_notify . . . 1339
12.313.2.6 next_avail . . . 1340
12.314 L4virtio::Svr::Virtqueue::Head_desc Class Reference . . . 1340
12.314.1 Detailed Description . . . 1341
12.314.2 Member Function Documentation . . . 1341
12.314.2.1 desc . . . 1341
12.314.2.2 operator Null_ptr_check const * . . . 1342
12.314.2.3 valid . . . 1342
12.315 L4virtio::Virtqueue Class Reference . . . 1342
12.315.1 Detailed Description . . . 1345
12.315.2 Member Function Documentation . . . 1345
12.315.2.1 disable . . . 1345
12.315.2.2 dump . . . 1346
12.315.2.3 no_notify_guest . . . 1346
12.315.2.4 no_notify_host . . . 1347

12.315.2.5 no_notify_host . . . 1347
12.315.2.6 num . . . 1348
12.315.2.7 ready . . . 1348
12.315.2.8 setup . . . 1349
12.315.2.9 total_size . . . 1350
12.315.2.10 total_size . . . 1350
12.316 L4virtio::Virtqueue::Avail Class Reference . . . 1351
12.316.1 Detailed Description . . . 1352
12.317 L4virtio::Virtqueue::Avail::Flags Struct Reference . . . 1352
12.317.1 Detailed Description . . . 1353
12.317.2 Member Typedef Documentation . . . 1353
12.317.2.1 no_irq_bfm_t . . . 1353
12.317.3 Member Function Documentation . . . 1353
12.317.3.1 no_irq . . . 1353
12.317.3.2 no_irq . . . 1353
12.318 L4virtio::Virtqueue::Desc Class Reference . . . 1354
12.318.1 Detailed Description . . . 1355
12.319 L4virtio::Virtqueue::Desc::Flags Struct Reference . . . 1355
12.319.1 Detailed Description . . . 1357
12.319.2 Member Typedef Documentation . . . 1357
12.319.2.1 indirect_bfm_t . . . 1357
12.319.2.2 next_bfm_t . . . 1357
12.319.2.3 write_bfm_t . . . 1357
12.319.3 Member Function Documentation . . . 1357
12.319.3.1 indirect . . . 1357
12.319.3.2 indirect . . . 1358
12.319.3.3 next . . . 1358
12.319.3.4 next . . . 1358
12.319.3.5 write . . . 1358
12.319.3.6 write . . . 1359
12.320 L4virtio::Virtqueue::Used Class Reference . . . 1359
12.320.1 Detailed Description . . . 1360
12.321 L4virtio::Virtqueue::Used::Flags Struct Reference . . . 1360
12.321.1 Detailed Description . . . 1361
12.321.2 Member Typedef Documentation . . . 1361
12.321.2.1 no_notify_bfm_t . . . 1361
12.321.3 Member Function Documentation . . . 1361
12.321.3.1 no_notify . . . 1361
12.321.3.2 no_notify . . . 1361
12.322 L4virtio::Virtqueue::Used_elem Struct Reference . . . 1362

12.322.1 Detailed Description . . . 1362
12.322.2 Constructor & Destructor Documentation . . . 1362
12.322.2.1 Used_elem . . . 1362
12.323 l4virtio_block_config_t Struct Reference . . . 1363
12.323.1 Detailed Description . . . 1363
12.323.2 Field Documentation . . . 1363
12.323.2.1 blk_size . . . 1363
12.324 l4virtio_block_header_t Struct Reference . . . 1364
12.324.1 Detailed Description . . . 1364
12.325 l4virtio_config_hdr_t Struct Reference . . . 1364
12.325.1 Detailed Description . . . 1366
12.325.2 Field Documentation . . . 1366
12.325.2.1 status . . . 1366
12.326 l4virtio_config_queue_t Struct Reference . . . 1366
12.326.1 Detailed Description . . . 1367

13 Example Documentation **1369**
13.1 examples/clntsrv/client.cc . . . 1369
13.2 examples/clntsrv/clntsrv.cfg . . . 1369
13.3 examples/clntsrv/server.cc . . . 1370
13.4 examples/libs/l4re/c++/mem_alloc/ma+rm.cc . . . 1371
13.5 examples/libs/l4re/c++/shared_ds/ds_clnt.cc . . . 1372
13.6 examples/libs/l4re/c++/shared_ds/ds_srv.cc . . . 1373
13.7 examples/libs/l4re/c++/shared_ds/shared_ds.cfg . . . 1375
13.8 examples/libs/l4re/c/ma+rm.c . . . 1376
13.9 examples/libs/l4re/streammap/client.cc . . . 1377
13.10 examples/libs/l4re/streammap/server.cc . . . 1378
13.11 examples/libs/l4re/streammap/streammap.cfg . . . 1379
13.12 examples/libs/libirq/async_isr.c . . . 1379
13.13 examples/libs/libirq/loop.c . . . 1380
13.14 examples/libs/shmc/prodcons.c . . . 1381
13.15 examples/sys/aliens/main.c . . . 1382
13.16 examples/sys/ipc/ipc.cfg . . . 1384
13.17 examples/sys/ipc/ipc_example.c . . . 1384
13.18 examples/sys/isr/main.c . . . 1386
13.19 examples/sys/migrate/thread_migrate.cc . . . 1387
13.20 examples/sys/migrate/thread_migrate.cfg . . . 1388
13.21 examples/sys/singlestep/main.c . . . 1388
13.22 examples/sys/start-with-exc/main.c . . . 1390
13.23 examples/sys/utcb-ipc/main.c . . . 1392

13.24 examples/sys/ux-vhw/main.c 1394
13.25 hello/server/src/main.c 1394
13.26 tmpfs/lib/src/fs.cc 1395

Index **1402**

Chapter 1

Overview

Welcome to the documentation of the L4 Runtime Environment, or L4Re for short. There are two parts to this documentation: a user manual, which provides a birds eye view of L4Re and its environment, and a reference section which documents the complete programming API.

User Manual

1. Introduction shortly explains the concept of microkernels and introduces the basic terminology.
2. Tutorial helps you getting started with setting up the development environment and writing your own first L4Re application.
3. Programming for L4Re explains in detail the most important programming concepts.

Reference

The second part provides the complete reference of all classes and functions of the L4Re environment as well as a list of example code.

Chapter 2

Introduction

The intention of this section is to provide a short overview about the L4Re environment.

The general structure of a microkernel-based system will be introduced and the principal functionality of the servers in the basic environment outlined.

2.1 Fiasco.OC

The Fiasco.OC microkernel is the lowest-level piece of software running in an L4-based system. The microkernel is the only program that runs in privileged processor mode. It does not include complex services such as program loading, device drivers, or file systems; those are implemented in user-level programs on top of it (a basic set these services and abstractions is provided by the L4 Runtime Environment).

Fiasco.OC kernel services are implemented in kernel objects. Tasks hold references to kernel objects in their respective *"object space"*, which is a kernel-protected table. These references are called *capabilities*. Fiasco system calls are function invocations on kernel objects through the corresponding capabilities. These can be thought of as function invocations on object references in an object-oriented programming environment. Furthermore, if a task owns a capability, it may grant other tasks the same (or fewer) rights on this object by passing the capability from its own to the other task's object space.

From a design perspective, capabilities are a concept that enables flexibility in the system structure. A thread that invokes an object through a capability does not need to care about where this object is implemented. In fact, it is possible to implement all objects either in the kernel or in a user-level server and replace one implementation with the other transparently for clients.

2.1.1 Communication

The basic communication mechanism in L4-based systems is called *"Inter Process Communication (IPC)"*. It is always synchronous, i.e. both communication partners need to actively rendezvous for IPC. In addition to transmitting arbitrary data between threads, IPC is also used to resolve hardware exceptions, faults and for virtual memory management.

2.1.2 Kernel Objects

The following list gives a short overview of the kernel objects provided by the Fiasco.OC microkernel:

- **Task** A task comprises a memory address space (represented by the task's page table), an object space (holding the kernel protected capabilities), and on X86 an IO-port address space.
- **Thread** A thread is bound to a task and executes code. Multiple threads can coexist in one task and are scheduled by the Fiasco scheduler.

- **Factory** A factory is used by applications to create new kernel objects. Access to a factory is required to create any new kernel object. Factories can control and restrict object creation.
- **IPC Gate** An IPC gate is used to create a secure communication channel between different tasks. It embeds a label (kernel protected payload) that securely identifies the gate through which a message is received. The gate label is not visible to and cannot be altered by the sender.
- **IRQ** IRQ objects provide access to hardware interrupts. Additionally, programs can create new virtual interrupt objects and trigger them. This allows to implement a signaling mechanism. The receiver cannot decide whether the interrupt is a physical or virtual one.
- **Vcon** Provides access to the in-kernel debugging console (input and output). There is only one such object in the kernel and it is only available, if the kernel is built with debugging enabled. This object is typically interposed through a user-level service or without debugging in the kernel can be completely based on user-level services.
- **Scheduler** Implements scheduling policy and assignment of threads to CPUs, including CPU statistics.

2.2 L4Re System Structure

The system has a multi-tier architecture consisting of the following layers depicted in the figure below:

- **Microkernel** The microkernel is the component at the lowest level of the software stack. It is the only piece of software that is running in the privileged mode of the processor.
- **Tasks** Tasks are the basic containers (address spaces) in which system services and applications are executed. They run in the processor's deprivileged user mode.

Figure 2.1: Basic Structure of an L4Re based system

In terms of functionality, the system is structured as follows:

- **Microkernel** The kernel provides primitives to execute programs in tasks, to enforce isolation among them, and to provide means of secure communication in order to let them cooperate. As the kernel is the most privileged, security-critical software component in the system, it is a general design goal to make it as small as possible in order to reduce its attack surface. It provides only a minimal set of mechanisms that are necessary to support applications.

- **Runtime Environment** The small kernel offers a concise set of interfaces, but these are not necessarily suited for building applications directly on top of it. The L4 Runtime Environment aims at providing more convenient abstractions for application development. It comprises low-level software components that interface directly with the microkernel. The root pager *sigma0* and the root task *Moe* are the most basic components of the runtime environment. Other services (e.g., for device enumeration) use interfaces provided by them.

- **Applications** Applications run on top of the system and use services provided by the Runtime Environment – or by other applications. There may be several types of applications in the system and even virtual machine monitors and device drivers are considered applications in the terminology used in this document. They are running alongside other applications on the system.

Lending terminology from the distributed systems area, applications offering services to other applications are usually called *servers*, whereas applications using those services are named *clients*. Being in both roles is also common, for instance, a file system server may be viewed as a server with respect to clients using the file system, while the server itself may also act as a client of a hard disk driver.

2.3 L4 Runtime Environment (L4Re)

The L4 Runtime Environment (L4Re) provides a basic set of services and abstractions, which are useful to implement and run user-level applications on top of the Fiasco.OC microkernel.

L4Re consists of a set of libraries and servers. Libraries as well as server interfaces are completely object oriented. They implement prototype implementations for the classes defined by the L4Re specification.

A minimal L4Re-based application needs 3 components to be booted beforehand: the Fiasco microkernel, the root pager (Sigma0), and the root task (Moe). The Sigma0 root pager initially owns all system resources, but is usually used only to resolve page faults for the Moe root task. Moe provides the essential services to normal user applications such as an initial program loader, a region-map service for virtual memory management, and a memory (data space) allocator.

Chapter 3

Tutorial

This tutorial assumes that the reader is familiar with the basic L4 concepts that were discussed in the Introduction section and has a working knowledge of C++.

Here you can find the first steps to boot a very simple setup. The setup consists of the following components:

- Fiasco.OC — Microkernel
- Sigma0 — Root Pager
- Moe — Root Task
- Ned — Init Process
- hello — Hello World Application

The guide assumes that you already compiled the base components and describes how to generate an ISO image, with GRUB 1 or GRUB 2 as a boot loader, that can for example be booted within QEMU.

First you need a `modules.list` file that contains an entry for the scenario.

```
modaddr  0x002000000

entry hello
  kernel   fiasco -serial_esc
  roottask moe rom/hello.cfg
  module   l4re
  module   ned
  module   hello.cfg
  module   hello
```

This file describes all the binaries and scripts to put into the ISO image, and also describes the GRUB `menu.lst` contents. What you need to do is to set the `make` variable `MODULE_SEARCH_PATH` to contain the path to your Fiasco.OC build directory and the directory containing your `hello.cfg` script.

The `hello.cfg` script should look like the following. A ready to use version can be found in l4/conf/examples.

```
local L4 = require("L4");
L4.default_loader:start({}, "rom/hello");
```

The first line of this script ensures that the `L4` package is available for the script. The second line uses the default loader object defined in that package and starts the binary `rom/hello`.

Note

> All modules defined in `modules.list` are available as data spaces (L4Re::Dataspace) and registered in a name space (L4Re::Namespace). This name space is in turn available as 'rom' to the init process (Ned).

Now you can go to your L4Re build directory and run the following command.

Note

The example assumes that you have created the `modules.list` and `hello.cfg` files in the /tmp directory. Adapt if you created them somewhere else.

```
make grub1iso E=hello MODULES_LIST=/tmp/modules.list MODULE_SEARCH_PATH=/tmp:<path_to_fiasco_builddir>
```

Or as an alternative use GRUB 2:

```
make grub2iso E=hello MODULES_LIST=/tmp/modules.list MODULE_SEARCH_PATH=/tmp:<path_to_fiasco_builddir>
```

Now you should be able to boot the image in QEMU by running:

```
qemu-system-i386 -cdrom images/hello.iso -serial stdio
```

If you press `<ESC>` in the terminal that shows you the serial output you enter the Fiasco.OC kernel debugger... Have fun.

3.1 Customizations

A basic set of bootable entries can be found in `l4/conf/modules.list`. This file is the default for any image creation as shown above. It is recommeded that local modification regarding image creation are done in `conf/Makeconf.boot`. Initially you may copy `Makeconf.boot.example` to `Makeconf.boot`. You can overwrite `MODULES_LIST` to set your own modules-list file. Set `MODULE_SEARCH_PATH` to your setup according to the examples given in the file. When configured a `make` call is reduced to:

```
make grub2iso E=hello
```

All other local configuration can be done in a `Makeconf.local` file located in the `l4` directory.

Chapter 4

Programming for L4Re

This part of the documentation discusses the concept of microkernel-based programming in more detail.

You should have already a basic understanding of the L4Re programming environment from the tutorial.

- Capabilities and Naming
- Initial Environment and Application Bootstrapping
- Memory management - Data Spaces and the Region Map
- Program Input and Output
- Initial Memory Allocator and Factory
- Application and Server Building Blocks
- Pthread Support
- tasks and threads
- communication channels
- server loops
- Interface Definition Language
- hardware access
- L4Re Servers
- L4Re Build System

4.1 Capabilities and Naming

The L4Re system is a capability based system which uses and offers capabilities to implement fine-grained access control.

Generally, owning a capability means to be allowed to communicate with the object the capability points to. All user-visible kernel objects, such as tasks, threads, and IRQs, can be accessed only through a capability. Please refer to the Kernel Objects documentation for details. Capabilities are stored in per-task capability tables (the object space) and are referenced by capability selectors or object flex pages. In a simplified view, a capability selector is a natural number indexing into the capability table of the current task.

As a matter of fact, a system designed solely based on capabilities, uses so-called 'local names', because each task can only access those objects made available to this task. Other objects are not visible to and accessible by the task.

Figure 4.1: Capabilities and Local Naming in L4

So how does an application get access to a service? In general all applications are started with an initial set of objects available. This set of objects is predetermined by the creator of a new application process and granted directly to into the new task before starting the first application thread. The application can then use these initial objects to request access to further objects or to transfer capabilities to own objects to other applications. A central L4Re object for exchanging capabilities at runtime is the name-space object, implementing a store of named capabilities.

From a security perspective, the set of initial capabilities (access rights to objects) completely define the execution environment of an application. Mandatory security policies can be defined by well known properties of the initial objects and carefully handled access rights to them.

4.2 Initial Environment and Application Bootstrapping

New applications that are started by a loader conforming to L4Re get provided an Initial Environment.

This environment comprises a set of capabilities to initial L4Re objects that are required to bootstrap and run this application. These capabilities include:

- A capability to an initial memory allocator for obtaining memory in the form of data spaces
- A capability to a factory which can be used to create additional kernel objects
- A capability to a Vcon object for debugging output and maybe input
- A set of named capabilities to application specific objects

During the bootstrapping of the application, the loader establishes data spaces for each individual region in the ELF binary. These include data spaces for the code and data sections, and a data space backed with RAM for the stack of the program's first thread.

One loader implementation is the `moe` root task. Moe usually starts an *init* process that is responsible for coordinating the further boot process. The default *init* process is `ned`, which implements a script-based configuration and startup of other processes. Ned uses Lua (`http://www.lua.org`) as its scripting language, see Ned Script example for more details.

4.2.1 Configuring an application before startup

The default L4Re init process (Ned) provides a Lua script based configuration of initial capabilities and application startup. Ned itself also has a set of initial objects available that can be used to create the environment for an application. The most important object is a kernel object factory that allows creation of kernel objects such as IPC gates (communication channels), tasks, threads, etc. Ned uses Lua tables (associative arrays) to represent sets of capabilities that shall be granted to application processes.

```
local caps = {
    name = some_capability
}
```

The L4 Lua package in Ned also has support functions to create application tasks, region-map objects, etc. to start an ELF binary in a new task. The package also contains Lua bindings for basic L4Re objects, for example, to generic factory objects, which are used to create kernel objects and also user-level objects provided by user-level servers.

```
L4.default_loader:start({ caps = { some_service = service } }, "rom/program --arg");
```

4.2.2 Connecting clients and servers

In general, a connection between a client and a server is represented by a communication channel (IPC gate). That is available to the client and the server. You can see the simplest connection between a client and a server in the following example.

```
local loader = L4.default_loader; -- which is Moe
local svc = loader:new_channel();  -- create an IPC gate
loader:start({ caps = { service = svc:full() }}, "rom/my_server");
loader:start({ caps = { service = svc:m("rw") }}, "rom/my_client");
```

As you can see in the snippet, the first action is to create a new channel (IPC gate) using `loader:new_channel()`. The capability to the gate is stored in the variable `svc`. Then the binary `my_server` is started in a new task, and full (`:full()`) access to the IPC gate is granted to the server as initial object. The gate is accessible to the server application as "service" in the set of its initial capabilities. Virtually in parallel a second task, running the client application, is started and also given access to the IPC gate with less rights (`:m("rw")`, note, this is essential). The server can now receive messages via the IPC gate and provide some service and the client can call operations on the IPC gate to communicate with the server.

Services that keep client specific state need to implement per-client server objects. Usually it is the responsibility of some authority (e.g., Ned) to request such an object from the service via a generic factory object that the service provides initially.

```
local loader = L4.default_loader; -- which is Moe
local svc = loader:new_channel():m("rws");  -- create an IPC gate with rws rights
loader:start({ caps = { service = svc:full() } }, "rom/my-service");
loader:start({ caps = { foo_service = svc:create(object_to_create, "param") }}, "rom/client");
```

This example is quite similar to the first one, however, the difference is that Ned itself calls the create method on the factory object provided by the server and passes the returned capability of that request as "foo_service" to the client process.

Note

The `svc:create(..)` call blocks on the server. This means the script execution blocks until the my-service application handles the create request.

4.3 Memory management - Data Spaces and the Region Map

4.3.1 User-level paging

Memory management in L4-based systems is done by user-level applications, the role is usually called *pager*. Tasks can give other tasks full or restricted access rights to parts of their own memory. The kernel offers means to grant the memory in a secure way, often referred to as *memory* mapping.

The described mechanism can be used to construct a memory hierarchy among tasks. The root of the hierarchy is `sigma0`, which initially gets all system resources and hands them out once on a first-come-first-served basis. Memory resources can be mapped between tasks at a page-size granularity. This size is predetermined by the CPU's memory management unit and is commonly set to 4 kB.

4.3.1.1 Data spaces

A data space is the L4Re abstraction for objects which may be accessed in a memory mapped fashion (i.e., using normal memory read and write instructions). Examples include the sections of a binary which the loader attaches to the application's address space, files in the ROM or on disk provided by a file server, the registers of memory-mapped devices and anonymous memory such as the heap or the stack.

Anonymous memory data spaces in particular (but in general all data spaces except memory mapped IO) can either be constructed entirely from a portion of the RAM or the current working set may be multiplexed on some portion of the RAM. In the first case it is possible to eagerly insert all pages (more precisely page-frame capabilities) into the application's address space such that no further page faults occur when this data space is accessed. In general, however, only the pages for the some portion are provided and further pages are inserted by the pager as a result of page faults.

4.3.1.2 Virtual Memory Handling

The virtual memory of each task is constructed from data spaces, backing virtual memory regions (VMRs). The management of the VMRs is provided by an object called *region map*. A dedicated region-map object is associated with each task, it allows to attach and detach data spaces to an address space as well as to reserve areas of virtual memory. Since the region-map object possesses all knowledge about virtual memory layout of a task, it also serves as an application's default pager.

4.3.1.3 Memory Allocation

Operating systems commonly use anonymous memory for implementing dynamic memory allocation (e.g., using `malloc` or `new`). In an L4Re-based system, each task gets assigned a memory allocator providing anonymous memory using data spaces.

See also

L4Re::Dataspace and L4Re::Rm.

4.4 Program Input and Output

The initial environment provides a Vcon capability used as the standard input/output stream.

Output is usually connected to the parent of the program and displayed as debugging output. The standard output is also used as a back end to the C-style printf functions and the C++ streams.

Vcon services are implemented in Moe and the loader as well as by the Fiasco kernel and connected either to the serial line or to the screen if available.

See also

Virtual Console

4.5 Initial Memory Allocator and Factory

The purpose of the memory allocator and of the factory is to provide the application with the means to allocate memory (in the form of data spaces) and kernel objects respectively.

An initial memory allocator and an initial factory are accessible via the allocation L4Re environment.

See also

L4Re::Mem_alloc

The factory is a kernel object that provides the ability to create new kernel objects dynamically. A factory imposes a resource limit for kernel memory, and is thus a means to prevent denial of service attacks on kernel resources. A factory can also be used to create new factory objects.

See also

Factory

4.6 Application and Server Building Blocks

So far we have discussed the environment of applications in which a single thread runs and which may invoke services provided through their initial objects.

In the following we describe some building blocks to extend the application in various dimensions and to eventually implement a server which implements user-level objects that may in turn be accessed by other applications and servers.

4.6.1 Creating Additional Application Threads

To create application threads, one must allocate a stack on which this thread may execute, create a thread kernel object and setup the information required at startup time (instruction pointer, stack pointer, etc.). In L4Re this functionality is encapsulated in the pthread library.

4.6.2 Providing a Service

In capability systems, services are typically provided by transferring a capability to those applications that are authorised to access the object to which the capability refers to.

Let us discuss an example to illustrate how two parties can communicate with each other: Assume a simple file server, which implements an interface for accessing individual files: read(pos, buf, length) and write(pos, data, length).

L4Re provides support for building servers based on the class L4::Server_object. L4::Server_object provides an abstract interface to be used with the L4::Server class. Specific server objects such as, in our case, files inherit from L4::Server_object. Let us call this class File_object. When invoked upon receiving a message, the L4::Server will automatically identify the corresponding server object based on the capability that has been provided to its clients and invoke this object's *dispatch* function with the incoming message as a parameter. Based on this message, the server must then decide which of the protocols it implements was invoked (if any). Usually, it will evaluate a protocol specific opcode that clients are required to transmit as one of the first words in the message. For example, assume our server assigns the following opcodes: Read = 0 and Write = 1. The `dispatch` function calls the corresponding server function (i.e., `File_object::read()` or `File_object::write()`), which will in turn parse additional parameters given to the function. In our case, this would be the position and the amount

of data to be read or written. In case the write function was called the server will now update the contents of the file with the data supplied. In case of a read it will store the requested part of the file in the message buffer. A reply to the client finishes the client request.

4.7 Pthread Support

L4Re supports the standard pthread library functionality.

Therefore L4Re itself does not contain any documentation for pthreads itself. Please refer to the standard pthread documentation instead.

The L4Re specific parts will be described herein.

- Include pthread-l4.h header file:

```
#include <pthread-l4.h>
```

- Return the local thread capability of a pthread thread:

  Use `pthread_l4_cap(pthread_t t)` to get the capability index of the pthread t.

  For example:

```
pthread_l4_cap(pthread_self());
```

- Setting the L4 priority of an L4 thread works with a special scheduling policy (other policies do not affect the L4 thread priority):

```
pthread_t t;
pthread_attr_t a;
struct sched_param sp;

pthread_attr_init(&a);
sp.sched_priority = l4_priority;
pthread_attr_setschedpolicy(&a, SCHED_L4);
pthread_attr_setschedparam(&a, &sp);
pthread_attr_setinheritsched(&a, PTHREAD_EXPLICIT_SCHED);

if (pthread_create(&t, &a, pthread_func, NULL))
  // failure...

pthread_attr_destroy(&a);
```

4.8 Interface Definition Language

An interface definition in L4Re is normally declared as a class derived from L4::Kobject_t.

For example, the simple calculator example declares its interface like that:

```
struct Calc : L4::Kobject_t<Calc, L4::Kobject>
{
  L4_INLINE_RPC(int, sub, (l4_uint32_t a, l4_uint32_t b,
      l4_uint32_t *res));
  L4_INLINE_RPC(int, neg, (l4_uint32_t a, l4_uint32_t *res));

  typedef L4::Typeid::Rpcs<sub_t, neg_t> Rpcs;
};
```

The signature of each function is first declared using on of the RPC macros (see below) and then all the functions need to be listed in the Rpcs type.

4.8.1 Parameter types for RPC

Generally all value parameters, const reference parameters, and const pointer parameters to an RPC interface are considered as input parameters for the RPC and are transmitted from the client to the server.

Note

This means that `char const *` is treated as an input `char` and not as a zero terminated string value, for strings see L4::Ipc::String<>.

Parameters that are non-const references or non-const pointers are treated as output parameters going from the server to the client.

```
L4_RPC(long, test, (int arg1, char const *arg2, unsigned *ret1));
```

The example shows the declaration of a method called `test` with `long` as return type, `arg1` is an `int` input, `arg2` a `char` input, and `ret1` an `unsigned` output parameter.

Type	Direction	Client-Type	Server-Type
`T`	Input	`T`	`T`
`T const &`	Input	`T const &`	`T const &`
`T const *`	Input	`T const *`	`T const &`
`T &`	Output	`T &`	`T &`
`T *`	Output	`T *`	`T &`
`L4::Ipc::In_out<T &>`	In/Out	`T &`	`T &`
`L4::Ipc::In_out<T *>`	In/Out	`T *`	`T &`
`L4::Ipc::Cap<T>`	Input	`L4::Ipc::Cap<T>`	`L4::Ipc::Snd_fpage`
`L4::Ipc::Out<L4::Cap<T>>`	Output	`L4::Cap<T>`	`L4::Ipc::Cap<T> &`

Array types can be used to transmit arrays of variable length. They can either be stored in a client-provided buffer (L4::Ipc::Array), copied into a server-provided buffer (L4::Ipc::Array_in_buf) or directly read and written into the UTCB (L4::Ipc::Array_ref). For latter type, the start position of an array type needs to be known in advance. That implies that only one such array can be transmitted per direction and it must be the last part in the message.

Type	Direction	Client-Type	Server-Type
`L4::Ipc::Array<const T>`	Input	`L4::Ipc::Array<const T>`	`L4::Ipc::Array_ref<const T>`
`L4::Ipc::Array<const T>`	Input	`L4::Ipc::Array<const T>`	`L4::Ipc::Array_in_buf<T> const &`
`L4::Ipc::Array<T> &`	Output	`L4::Ipc::Array<T> &`	`L4::Ipc::Array_ref<T> &`
`L4::Ipc::Array_ref<T> &`	Output	`L4::Ipc::Array_ref<T> &`	`L4::Ipc::Array_ref<T> &`

Finally, there are some optional types where the sender can choose if the parameter should be included in the message. These types are for the implementation of some legacy message formats and should generally not be needed for the definition of ordinary interfaces.

Type	Direction	Client-Type	Server-Type
L4::Ipc::Opt<T>	Input	L4::Ipc::Opt<T>	T
L4::Ipc::↩ Opt<const T*>	Input	L4::Ipc::↩ Opt<const T*>	T
L4::Ipc::Opt<T &>	Output	T &	L4::Ipc::Opt<T> &
L4::Ipc::Opt<T *>	Output	T *	L4::Ipc::Opt<T> &
L4::Ipc::↩ Opt<Array_ref<T> &>	Output	Array_ref<T> &	L4::Ipc::↩ Opt<Array_↩ ref<T>> &

4.8.2 RPC Return Types

On the server side, the return type of an RPC handling function is always `long`. The return value is transmitted via the label field in l4_msgtag_t and is therefore restricted to its length. Per convention, a negative return value is interpreted as an error condition. If the return value is negative, output parameters are not transmitted back to the client.

Attention

The client must never rely on the content of output parameters when the return value is negative.

On the client-side, the return value of the RPC is set as defined in the RPC macro. If `l4_msgtag_t` is given, then the client has access to the full message tag, otherwise the return type should be `long`. Note that the client might not only receive the server return value in response but also an IPC error code.

4.8.3 RPC Method Declaration

RPC member functions can be declared using one of the following C++ macros.

For inline RPC stubs, where the RPC stub code itself is `inline`:

- `L4_INLINE_RPC(res, name, (args...), flags)`
- `L4_INLINE_RPC_OP(op, res, name, (args...), flags)`
- `L4_INLINE_RPC_NF(res, name, (args...), flags)`
- `L4_INLINE_RPC_NF_OP(opcode, Ret_type, func_name, (args...), flags)`

For external RPC stubs, where the RPC stub code must be defined in a separate compile unit (usually a `.cc` file):

- `L4_RPC(Ret_type, func_name, (args...), flags)`
- `L4_RPC_OP(opcode, Ret_type, func_name, (args...), flags)`
- `L4_RPC_NF(Ret_type, func_name, (args...), flags)`
- `L4_RPC_NF_OP(opcode, Ret_type, func_name, (args...), flags)`

To generate the implementation of an external RPC stub:

- `L4_RPC_DEF(class_name::func_name)`

The `NF` versions of the macro generally do not generate a callable member function named `<name>` but do only generate the type `<name>_t`. This data type can be used to call the RPC stub explicitly using `<name>_t↩ ::call(L4::Cap<Iface_class> cap, args...)`.

4.9 L4Re Servers

Here you shall find a quick overview over the standard services running on Fiasco.OC and L4Re.

Sigma0, the Root Pager

Sigma0 is a special server running on L4 because it is responsible of resolving page faults for the root task, the first useful task on L4Re. Sigma0 can be seen as part of the kernel, however it runs in unprivileged mode. To run something useful on Fiasco.OC you usually need to run Sigma0, nevertheless it is possible to replace Sigma0 by a different implementation.

Moe, the Root Task

Moe is our implementation of the L4 root task that is responsible for bootstrapping the system, and to provide basic resource management services to the applications on top. Therefore Moe provides L4Re resource management an multiplexing services:

- **Memory** in the form of memory allocators (L4Re::Mem_alloc, L4::Factory) and data spaces (L4Re::↩ Dataspace)
- **Cpu** in the form of basic scheduler objects (L4::Scheduler)
- **Vcon** multiplexing for debug output (output only)
- **Virtual memory management** for applications, L4Re::Rm

Moe further provides an implementation of L4Re name spaces (L4Re::Namespace), which are for example used to provide a read only directory of all multi-boot modules. In the case of a boot loader, like grub that enables a VESA frame buffer, there is also a single instance of an L4Re graphics session (L4Re::Goos).

To start the system Moe starts a single ELF program, this init process. The init process (usually Ned, see the next section) gets access to all resources managed by Moe and to the Sigma0 root pager interface.

For more details see Moe, the Root-Task.

Ned, the Default Init Process

To keep the root task free from complicated scripting engines and to avoid circular dependencies in application startup (that could lead to dead locks) the configuration and startup of the real system is managed by an extra task, the init process.

Ned is such an init process that allows system configuration via Lua scripts.

For more information see Ned.

Io, the Platform and Device Resource Manager

Because all peripheral management of Fiasco.OC is done in user-level applications, there is the need to have a centralized management of the resources belonging to the platform and to peripheral devices.

This is the job of Io. Io provides portable abstractions for iterating and accessing devices and their resources (IRQ's, IO Memory...), as well as delegating access to those resources to other applications (e.g., device drivers).

For more details see Io, the Io Server.

Mag, the GUI Multiplexer

Our default multiplexer for the graphics hardware is Mag. Mag is a Nitpicker (TODO: ref) derivate that allows secure multiplexing of the graphics and input hardware among multiple applications and multiple complete windowing environments.

For more information see Mag.

fb-drv, the Low-Level Graphics Driver

The fb-drv server provides low-level access and initialization of various graphics hardware. It has support for running VESA BIOS calls on Intel x86 platforms, as well as support for various ARM display controllers. *fb-drv*, provides a single instance of the L4Re::Goos interface and can serve as a back end for the Mag server, in particular, if there is no graphics support in the boot loader.

Rtc, the Real-Time Clock Server

Rtc is a simple multiplexer for real-time clock hardware on your platform.

4.9.1 Moe, the Root-Task

Moe is the default root-task implementation for L4Re-based systems.

Moe is the first task which is usually started in L4Re-based systems. The micro kernel starts *Moe* as the Root-Task.

Moe provides a default implementation for the basic L4Re abstractions, such as data spaces (L4Re::Dataspace), region maps (L4Re::Rm), memory allocators (L4Re::Mem_alloc, L4::Factory), name spaces (L4Re::Namespace) and so on (see L4Re Interface).

Moe consists of the following subsystems:

- Name-Space Provider — provides instances of name spaces
- Boot FS — provides access to the files loaded during platform boot (e.g., linked into the boot image or loaded via GRUB boot loader)
- Log Subsystem — provides tagged log output for applications
- Scheduler Subsystem — provides simple scheduler objects for scheduling policy enforcement
- Memory Allocator, Generic Factory — provides allocation of physical RAM as data spaces, as well as allocation of the other L4Re objects provided by Moe

4.9.1.1 Memory Allocator, Generic Factory

The generic factory in Moe is responsible for all kinds of dynamic object allocation. The interface is a combination of L4::Factory and, for traditional reasons, L4Re::Mem_alloc.

The generic factory interface allows allocation of the following objects:

- L4Re::Namespace
- L4Re::Dataspace, RAM allocation
- L4Re::Dma_space, memory management for DMA-capable devices
- L4Re::Rm, Virtual memory management for application tasks

- L4::Vcon (output only)
- L4::Scheduler, to provide a restricted priority / CPU range for clients
- L4::Factory, to provide a quota limited allocation for clients

The memory allocator in Moe is the alternative interface for allocating memory (RAM) in terms of L4Re::Dataspace-s (see also L4Re::Mem_alloc).

Dataspaces can be allocated with an arbitrary size. The granularity for memory allocation however is the machine page size (L4_PAGESIZE). A dataspace user must be aware that, as a result of this page-size granularity, there may be padding memory at the end of a dataspace which is accessible to each client. Moe currently allows most dataspace operations on this padding area. Nonetheless, the client must not make any assumptions about the size or content of the padding area, as this behaviour might change in the future.

The provided data spaces can have different characteristics:

- Physically contiguous and pre-allocated
- Non contiguous and on-demand allocated with possible copy on write (COW)

4.9.1.2 Name-Space Provider

Moe provides a name spaces conforming to the L4Re::Namespace interface (see Name-space API). Per default Moe creates a single name space for the Boot FS. That is available as `rom` in the initial objects of the init process.

4.9.1.2.1 Boot FS

The Boot FS subsystem provides read only access to the files loaded during the platform boot (or available in ROM). These files are either linked into the boot image or loaded via a flexible boot loader, such as GRUB.

The subsystem provides an L4Re::Namespace object as directory and an L4Re::Dataspace object for each file.

4.9.1.3 Log Subsystem

The logging facility of Moe provides per application tagged and synchronized log output.

4.9.1.4 Scheduler Subsystem

The scheduler subsystem provides a simple scheduler proxy.

4.9.1.5 Command-Line Options

Moe's command-line syntax is:

```
moe [--debug=<flags>] [--init=<binary>] [--l4re-dbg=<flags>] [--ldr-flags=<flags>] [-- <init options>]
```

--debug=<debug flags>

This option enables debug messages from Moe itself, the `<debug flags>` values are a combination of `info`, `warn`, `boot`, `server`, `loader`, and `ns` (or `all` for full verbosity).

--init=<init process>

This options allows to override the default init process binary, which is 'rom/ned'.

--l4re-dbg=<debug flags>

This option allows to set the debug options for the L4Re runtime environment of the init process. The flags are the same as for `--debug=`.

--ldr-flags=<loader flags>

This option allows setting some loader options for the L4Re runtime environment. The flags are `pre_alloc`, `all_segs_cow`,and `pinned_segs`.

-- <init options>

All command-line parameters after the special `--` option are passed directly to the init process.

4.9.2 Ned, the Init Process

Ned's job is to bootstrap the system running on L4Re.

The main thing to do here is to coordinate the startup of services and applications as well as to provide the communication channels for them. The central facility in Ned is the Lua (`http://www.lua.org`) script interpreter with the L4Re and ELF-loader bindings.

The boot process is based on the execution of one or more Lua scripts that create communication channels (IPC gates), instantiate other L4Re objects, organize capabilities to these objects in sets, and start application processes with access to those objects (or based on those objects).

For starting applications, Ned depends on the services of Moe, the Root-Task or another *loader*, which must provide data spaces and region maps. Ned also uses the 'rom' capability as source for Lua scripts and at least the 'l4re' binary (the runtime environment core) running in each application.

Each application Ned starts is equipped with an L4Re::Env environment that provides information about all the initial objects made accessible to this application.

4.9.2.1 Lua Bindings for L4Re

Ned provides various bindings for L4Re abstractions. These bindings are located in the 'L4' package (`require "L4"`).

4.9.2.1.1 Capabilities in Lua

Capabilities are handled as normal values in Lua. They can be stored in normal variables or Lua compound structures (tables). A capability in Lua possesses additional information about the access rights that shall be transfered to other tasks when the capability is transfered. To support implementation of the Principle of Least Privilege, minimal rights are assigned by default. Extended rights can be added using the method `mode`("...") (short `m`("...")) that returns a new reference to the capability with the given rights.

Note

It is generally impossible to elevate the real access rights to an object. This means that if Ned has only restricted rights to an object it is not possible to upgrade the access rights with the `mode` method.

The capabilities in Lua also carry dynamic type information about the referenced objects. They thereby provide type-specific operations on the objects, such as the `create` operation on a generic factory or the `query` and `register` operations on a name space.

4.9.2.1.2 Access to L4Re::Env Capabilities

The initial objects provided to Ned itself are accessible via the table `L4.Env`. The default (usually unnamed) capabilities are accessible as `factory`, `log`, `mem_alloc`, `parent`, `rm`, and `scheduler` in the `L4.Env` table.

4.9.2.1.3 Constants

Protocols

The protocol constants are defined by default in the L4 package's table `L4.Proto`. The definition is not complete and only covers what is usually needed to configure and start applications. The protocols are for example used as first argument to the `Factory:create` method.

```
Proto = {
  Dataspace = 0x4000,
  Namespace = 0x4001,
  Goos      = 0x4003,
  Mem_alloc = 0x4004,
  Rm        = 0x4005,
  Event     = 0x4006,
  Inhibitor = 0x4007,
  Irq       = -1,
  Sigma0    = -6,
  Log       = -13,
  Scheduler = -14,
  Factory   = -15,
  Ipc_gate  = 0,
}
```

Debugging Flags

Debugging flags used for the applications L4Re core:

```
Dbg = {
  Info       = 1,
  Warn       = 2,
  Boot       = 4,
  Server     = 0x10,
  Exceptions = 0x20,
  Cmd_line   = 0x40,
  Loader     = 0x80,
  Name_space = 0x400,
  All        = 0xffffffff,
}
```

Loader Flags

Flags for configuring the loading process of an application.

```
Ldr_flags = {
  eager_map    = 0x1, -- L4RE_AUX_LDR_FLAG_EAGER_MAP
  all_segs_cow = 0x2, -- L4RE_AUX_LDR_FLAG_ALL_SEGS_COW
  pinned_segs  = 0x4, -- L4RE_AUX_LDR_FLAG_PINNED_SEGS
}
```

4.9.2.1.4 Application Startup Details

The central facility for starting a new task with Ned is the class `L4.Loader`. This class provides interfaces for conveniently configuring and starting programs. It provides three operations:

- `new_channel()` Returns a new IPC gate that can be used to connect two applications
- `start()` and `startv()` Start a new application process and return a process object

The `new_channel()` call is used to provide a service application with a communication channel to bind its initial service to. The concrete behavior of the object and the number of IPC gates required by a server depends on the server implementation. The channel can the be passed to client applications as well or can be used for operations within the script itself.

`start()` and `startv()` always require at least two arguments. The first one is a table that contains information about the initial objects an application shall get. The second argument is a string, which for `start()` is the program name plus a white-space-separated list of program arguments (argv). For `startv()` the second argument is just the program binary name – which may contain spaces –, and the program arguments are provided as separate string arguments following the binary name (allowing spaces in arguments, too). The last optional argument is a table containing the POSIX environment variables for the program.

The Loader class uses reasonable defaults for most of the initial objects. However, you can override any initial object with some user-defined values. The main elements of the initial object table are:

- `factory` The factory used by the new process to create new kernel objects, such as threads etc. This must be a capability to an object implementing the L4::Factory protocol and defaults to the factory object provided to Ned.
- `mem` The memory allocator provided to the application and used by Ned allocates data spaces for the process. This defaults to Ned's memory allocator object (see L4Re::Mem_alloc).
- `rm_fab` The generic factory object used to allocate the region-map object for the process. (defaults to Ned's memory allocator).
- `log_fab` The generic factory to create the L4Re::Log object for the application's output (defaults to Ned's memory allocator). The `create` method of the `log_fab` object is called with `log_tag` and `log_color`, from this table, as arguments.
- `log_tag` The string used for tagging log output of this process (defaults to the program name) (see `log_fab`).
- `log_color` The color used for the log tag (defaults to "white").
- `scheduler` The scheduler object used for the process' threads (defaults to Ned's own scheduler).
- `caps` The table with application-specific named capabilities (default is an empty table). If the table does not contain a capability with the name 'rom', the 'rom' capability from Ned's initial caps is inserted into the table.

4.9.3 Io, the Io Server

The Io server handles all platform devices and resources such as I/O memory, ports (on x86) and interrupts, and grants access to those to clients.

Upon startup Io discovers all platform devices using available means on the system, e.g. on x86 the PCI bus is scanned and the ACPI subsystem initialised. Available I/O resource can also be configured via configuration scripts.

Io uses configuration can be considered as two parts:

- the description of the real hardware
- the description of virtual buses

Both descriptions represent hierarchical (tree) structure of device nodes. Where each device has a set of resources attached to it. And a device that has child devices can be considered a bus.

Hardware Description

The hardware description represents the devices that are available on the particular platform including their resource descriptions, such as MMIO regions, IO-Port regions, IRQs, bus numbers etc.

The root of the hardware devices is formed by a system bus device (accessible in the configuration via Io.system_bus()). As mentioned before, platforms that support methods for device discovery may populate the hardware description automatically, for example from ACPI. On platforms that do not have support for such methods you have to specify the hardware description by hand. A simple example for this is `x86-legacy.devs`.

Virtual Bus Description

Each Io server client is provided with its own virtual bus which it can iterate to find devices. A virtual PCI bus may be a part of this virtual bus.

Figure 4.2: IO Service Architecture Overview

The Io server must be configured to create virtual buses for its clients.

This is done with at least one configuration file specifying static resources as well as virtual buses for clients. The configuration may be split across several configuration files passed to Io through the command line.

To allow clients access to a available devices, a virtual system bus needs to be created that lists the devices and their resources that should be available to that client. The names of the busses correspond to the capabilities given to Io in its launch configuration.

A very simple configuration for Io could look like this:

```
-- vim:ft=lua
-- Example configuration for io

-- Configure two platform devices to be known to io
Io.Dt.add_children(Io.system_bus(), function()

  -- create a new hardware device called "FOODEVICE"
  FOODEVICE = Io.Hw.Device(function()
    -- set the compatibility IDs for this device
    -- a client tries to match against these IDs and configures
    -- itself accordingly
    -- the list should be sorted from specific to less specific IDs
    compatible = {"dev-foo,mmio", "dev-foo"};

    -- set the 'hid' property of the device, the hid can also be used
    -- as a compatible ID when matching clients
    Property.hid  = "dev-foo,Example";

    -- note: names for resources are truncated to 4 letters and a client
    -- can determine the name from the ID field of a l4vbus_resource_t
    -- add two resources 'int' and 'regs' to the device
    Resource.int  = Io.Res.irq(17);
    Resource.regs = Io.Res.mmio(0x6f000000, 0x6f007fff);
  end);

  -- create a new hardware device called "BARDEVICE"
  BARDEVICE = Io.Hw.Device(function()
    -- set the compatibility IDs for this device
    -- a client tries to match against these IDs and configures
    -- itself accordingly
    -- the list should be sorted from specific to less specific IDs
```

```
    compatible = {"dev-bar,mmio", "dev-bar"};

    -- set the 'hid' property of the device, the hid can also be used
    -- as a compatible ID when matching clients
    Property.hid  = "dev-bar,Example";

    -- Specify that this device is able to use direct memory access (DMA).
    -- This is needed to allow clients to gain access to DMA addresses
    -- used by this device to directly access memory.
    Property.flags = Io.Hw_device_DF_dma_supported;

    -- note: names for resources are truncated to 4 letters and a client
    -- can determine the name from the ID field of a l4vbus_resource_t
    -- add three resources 'intA', 'intB', and 'regs' to the device
    Resource.intA = Io.Res.irq(19);
    Resource.intB = Io.Res.irq(20);
    Resource.regs = Io.Res.mmio(0x6f100000, 0x6f100fff);
  end);
end);


Io.add_vbusses
{
-- Create a virtual bus for a client and give access to FOODEVICE
  client1 = Io.Vi.System_bus(function ()
    dev = wrap(Io.system_bus():match("FOODEVICE"));
  end);

-- Create a virtual bus for another client and give it access to BARDEVICE
  client2 = Io.Vi.System_bus(function ()
    dev = wrap(Io.system_bus():match("BARDEVICE"));
  end);
}
```

Each device supports a 'compatible' property. It is a list of compatibility strings. A client matches itself against one (or multiple) compatibility IDs and configures itself accordingly. All other device members are handled according to their type. If the type is a resource (Io.Res) it is added as a named resource. Note that resource names are truncated to 4 letters and are stored in the ID field of a l4vbus_resource_t. If the type is a device it is added as a child device to the current one. All other types are treated as a device property which can be used to configure a device driver. Right now, device properties are internal to Io only.

Assigning clients PCI devices could look like this:

```
-- This is a configuration snippet for PCI device selection

local hw = Io.system_bus();

Io.add_vbusses
{
  pciclient = Io.Vi.System_bus(function ()
    PCI = Io.Vi.PCI_bus(function ()
      pci_mm      = wrap(hw:match("PCI/CC_04"));
      pci_net     = wrap(hw:match("PCI/CC_02"));
      pci_storage = wrap(hw:match("PCI/CC_01"));
    end)
  end)
}
```

The CC numbers are PCI class codes. You can also use `REV_`, `VEN_`, `DEV_` and `SUBSYS_` to specify revision, vendor, device and subsystem with a hex number.

4.9.4 Mag, the GUI Multiplexer

Mag is the default multiplexer for graphics hardware.

It is not, and does not attempt to be, a fully-fledged window manager.

Command Line Options

As Mag's only command line option it supports loading additional plugins via the application's command line. Plugins must be either a Lua file or a shared library. Shared libraries must be named `libmag-$LIBNAME.so`.

Mag Sessions

Mag provides two types of sessions which a client can create via the Factory interface.

1. Mag client session

   A client with a mag client session gets access to the whole screen. The client has to allocate and manage it's own buffers and has to position them on the screen on its own. Mag provides the factory to create client sessions via the capability named `mag`.

2. Client framebuffer session

   For a client framebuffer session mag allocates a view of the requested size and displays it at the requested coordinates on the screen. Mag provides the factory to create framebuffer sessions via the capability named `svc`.

The options described below are options the client provides to the L4::Factory::create() call. These options influence the appearance and behaviour of the newly created session.

Session Factory Options

As a simple nitpicker clone Mag supports the so-called Xray mode. This mode displays all session labels and draws a colored frame around them. The session that currently has the input focus is highlighted. The Xray mode is actived via the special keys *Scroll* or *NEXTSONG*.

Mag allows to define a text label and a color for all client session types. The label and the color are displayed when Mag enters the Xray mode.

- Label Option: **label**

  `l=LABEL`, `label=LABEL` Set the session's text label to LABEL. The label is restricted to a length of 256 characters.

- Color Option: **col**

  `col=COLOR` Set the session's color which is used in Xray mode to tint the session's screen area and the border drawn around it. The argument can be either one of the following letters or a hexdecimal representation of the RGB values.

  - `r`, `R` Red color
  - `g`, `G` Green color
  - `b`, `B` Blue color
  - `w`, `W` White color
  - `y`, `Y` Yellow color
  - `v`, `V` Magenta color

Example

```
-- set label to "Linux" and use a light blue color
fb = mag_client:create(L4.Proto.Goos, "l=Linux", "col=98d9ff");
```

Mag Client Session Options

These options only apply to Mag client sessions.

- Default Background Option: **default-background**

  `dfl-bg`, `default-background` Marks this session as the default background.

Mag Client Framebuffer Session Options

These options only apply to Mag client framebuffer sessions.

- Geometry Option: **geometry**

  `g=GEOMETRY`, `geometry=GEOMETRY` Set the session's geometry and position on the screen. GEOM↩ETRY is provided in an X11-style format, e.g. `g=WIDTHxHEIGHT+X_OFFSET+Y_OFFSET`.

- Focus Option: **focus**

  `focus` Set the focus to this session.

- Collapsed Option: **shaded**

  `shaded` The window is collapsed and only the title bar is visible. The window can be expanded by clicking into the title bar with the middle mouse button. Collapsing and expanding works also independently of this option.

- Fixed Option: **fixed** `fixed` The window cannot be moved on the screen.

- Barheight Option: **barheight**

  `barheight=X` Set the height of the title bar in pixels.

Example

```
-- create a window of 640x480 pixels at position (100,100) on the screen.
fb = mag_fb:create(L4.Proto.Goos, "g=640x480+100+100");
```

4.10 L4Re Build System

L4Re uses a custom make-based build system, often simply referred to as *BID*.

This section explains how to use BID when writing applications and libraries for L4Re.

4.10.1 Building L4Re

Setting up the Build Directory

L4Re must be built out-of-source. Therefore the first mandatory step is creating and populating a build directory. From the root of the L4Re source tree run

```
make B=<builddir>
```

Other targets that can be executed in the source directory are

update

Update the source directory from svn. Only makes sense when you have downloaded L4Re from the official subversion repository.

help

Show a short help with the most important targets.

Invoking Make

Once the build directory is set up, BID make can be invoked in one of two ways:

1. Go to the build directory and invoke make without special options.
2. Go to a source directory with a BID make file and invoke `make O=<builddir> ....`

The default target builds the source (as you would expect), other targets that are available in build mode are

cleanfast

Quickly cleans the build directory by removing all subdirectories that contain generated files. The configuration will remain untouched.

clean

Remove generated files. Slower than `make cleanfast` but can be used on selected packages. Use `S=...` to select the target package.

In addition to these targets, there are a number of targets to generate images which are explained elsewhere.

4.10.2 Writing BID Make Files

The BID build system exports different roles that define what should be done in the subdirectory. So a BID make file essentially consists of defining the role and a number of role-dependent make variables. The basic layout should look like this:

```
PKGDIR  ?= <path to package's root directory>  # e.g., '.' or '..'
L4DIR   ?= <path to L4Re source directory>     # e.g. '$(PKGDIR)/../..'

<various definitions>

include $(L4DIR)/mk/<role>.mk
```

`PKGDIR` in the first line defines the root directory of the current package. `L4DIR` in the next line must be pointed to the root of the L4Re source tree the package should be built against. After this custom variable definitions for the role follow. In the final line of the file, the make file with the role-specific rules must be sourced.

The following roles are currently defined:

- project.mk - Sub-project Role
- subdir.mk - Directory Role
- prog.mk - Application Role
- lib.mk - Library Role
- include.mk - Header File Role
- doc.mk - Documentation Role
- test.mk - Test Application Role
- idl.mk - IDL File Role (currently unused)
- runux.mk - Tests in FiascoUX Role

BID-global Variables

This section lists variables that configure how the BID build system behaves. They are applicable for all roles.

Variable	Description
CC	C compiler for target
CXX	C++ compiler for target
HOST_CC	C compiler for host
HOST_CXX	C++ compiler for host

4.10.3 prog.mk - Application Role

The prog role is used to build executable programs.

General Configuration Variables

The following variables can only be set globally for the Makefile:

MODE

Kind of target to build for. The following values are possible:

- `static` - build a statically linked binary (default)
- `shared` - build a dynamically linked binary
- `l4linux` - build a binary for running on L4Linux on the target platform
- `host` - build for host system

SYSTEMS

List of architectures the target can be built for. The entries must be space-separated entries either naming an architecture (e.g. amd64) or an architecture and ABI (e.g, arm-l4f). When not defined, the target will be built for all possible platforms.

TARGET

Name or names of the binaries to compile. This variable may also be postfixed with a specific architecture.

SRC_CC_IS_CXX11

C++ standard to use. Default is `c++0x`.

Target-specific Configuration Variables

The following variables may either be used with or without a description suffix. Without suffix they will be used for all operations. With a specific description their use is restricted to a subset. These specifications include a target file and the architecture, both optional but in this order, separated by underscores. The specific variables will be used in addition to the more general ones.

SRC_C / SRC_CC / SRC_F / SRC_S

.c, .cc, .f90, .S source files.

REQUIRES_LIBS

List of libraries the binary depends on. This works only with libraries that export a pkg_config configuration file. Automatically adds any required include and link options.

DEPENDS_PKGS

List of packages this binary depends on. If one these packages is missing then building of the binary will be skipped.

CPPFLAGS / CFLAGS / CXXFLAGS / FFLAGS / ASFLAGS

Options for the C preprocessor, C compiler, C++ compiler, Fortran compiler and assembler. When used with suffix, the referred element is the source file, not the target file.

LDFLAGS

Options for the linker ld.

LIBS

Additional libraries to link against (with -l).

PRIVATE_LIBDIR

Additional directories to search for libraries.

CRT0 / CRTN

(expert use only) Files containing custom startup and finish code.

LDSCRIPT

(expert use only) Custom link script to use.

4.10.4 include.mk - Header File Role

The header file role is responsible for installing header file at the appropriate location.

The following variables can be used for customizing the process:

INCSRC_DIR

Source directory where the headers can be found. Default is the directory where the Makefile resides.

TARGET

List of header files to install. If left undefined, then `INCSRC_DIR` will be scanned for files with suffix `.h` or `.i`.

EXTRA_TARGET

When TARGET is undefined, then add these files to the headers found by scanning the source directory. Has no effect if `TARGET` has been defined.

CONTRIB_HEADERS

When set, the headers will be installed in `${BUILDDIR}/include/contrib/${PKGNAME}` rather than `${BUILDDIR}/include/l4/${PKGNAME}`.

INSTALL_INC_PREFIX

Base directory where to install the headers. Overwrites `CONTRIB_HEADERS`. The headers will then be found under `${BUILDDIR}/include/${INSTALL_INC_PREFIX}`.

PC_FILENAME

When set, a pkg_config configuration file is created with the given name.

4.10.5 test.mk - Test Application Role

The test role is very similar to the application role, it also builds an executable binary.

The difference is that is also builds for each target a test script that executes the test target either on the host (MODE=host) or a target platform (currently only qemu).

The role accepts all make variables that are accepted by the application role. The only difference is that the `TARGET` variable is not required. If it is missing, the source directory will be scanned for source files that fit the pattern `test_*.c[c]` and create one target for each of them.

Note

It is possible to still use SRC_C[C] when targets are determined automatically. In that case the specified sources will be used in addition* to the main `test_*.c[c]` source.

In addition to the variables above, there are a number of variables that control how the test is executed. All these variables may be used as a global variable that applies to all test or, if the target name is added as a suffix, set for a specific target only.

TEST_TARGET

Name of binary containing the test (default: same as `TARGET`).

TEST_EXPECTED

File containing expected output. By default the variable is empty, which means the test binary is expected to produce TAP test output, that can be directly processed.

TEST_EXPECTED_REPEAT

Number of times the expected output should be repeated, by default 1. When set to 0 then output is expected to repeat forever. This is particularly useful to make sure that stress tests that are meant to run in an endless loop are still alive. Note that such endless tests can only be run by directly executing the test script. They will be skipped when run in a test harness like `prove`.

TEST_TIMEOUT

Non-standard timeout after which the test run is aborted (useful for tests involving sleep).

MOE_CFG

LUA configuration file for startup to give to moe

REQUIRED_MODS

Additional modules needed to run the test.

QEMU_ARGS

Additional parameters to supply to QEMU.

MOE_ARGS

Additional parameters to supply to moe.

KERNEL_CONF

Features the Fiasco kernel must have been compiled with. A space-separated list of config options as used by Kconfig. `run_test` looks for a `globalconfig.out` file in the same directory as the kernel and checks that all options are enabled. If not, the test is skipped. Has only an effect if the `globalconfig.out` file is present.

L4LINUX_CONF

Features the L4Linux kernel must have been compiled with. Similar to `KERNEL_CONF` but checks for a `.config` file in the directory of the L4Linux kernel.

TEST_SETUP

Command to execute before the test is run. The test will only be executed if the command returns 0. If the exit code is 2, the test is marked as skipped with the reason provided in the final line of stdout.

In addition to compiled tests, it is also possible to create tests where the test binary or script comes from a different source. These tests must be listed in `EXTRA_TARGET` and for each target a custom `TEST_TARGET` must be provided.

Running Tests

The make role creates a test script which can be found in `<builddir>/test/t/<arch>/<api>`. It is possible to organise the tests further in subdirectories below by specifying a TEST_GROUP.

To be able to execute the test, a minimal test environment needs to be set up by exporting the following environment variables:

KERNEL_<arch>_<platform>, KERNEL_<arch>, KERNEL

Fiasco kernel binary to use. The test runner is able to check if the kernel has all features necessary for the test and skip tests accordingly. In order for this to work, the `globalconfig.out` config file from the build directory needs to be available in the same directory as the kernel.

L4LX_KERNEL_<arch>_<platform>, L4LX_KERNEL_<arch>, L4_LX_KERNEL

L4Linux binary to use. This is only required to run tests in `mode=l4linux`. If no L4Linux kernel is set then these tests will simply be skipped. The test runner is also able to check if the kernel has all features compiled in that are required to run the test successfully (see make variable `L4LINUX_CONF` above). For this to work, the `.config` configuration file from the build directory needs to be available in the same directory as the kernel.

LINUX_RAMDISK_<arch>_<platform>, LINUX_RAMDISK_<arch>, LINUX_RAMDISK

Ramdisk to mount as root in L4Linux. This is only required to run tests in `mode=l4linux`. If not supplied, L4Linux tests will be skipped. The ramdisk must be set up to start the test directly after the initial startup is finished. The name of the test binary is supplied via the kernel command line option `l4re_testprog`. The `tool/test` directory contains an example script `launch-l4linux-test`. which can be copied onto the ramdisk and started by the init script.

In addition to these variables, the following BID variables can be overwritten at runtime: `PLATFORM_TYPE` and `TEST_TIMEOUT`. You may also supply `QEMU_ARGS` and `MOE_ARGS` which will be appended to the parameters specified in the BID test make file.

Once the environment is set up, the tests can be run either by simply executing all of them from the build directory with

```
make test
```

or executing them directly, like

```
test/t/amd64_amdfam10/l4f/l4re-core/moe/test_namespace.t
```

or running one or more tests through the test harness `prove`, like

```
prove test/t/amd64_amdfam10/l4f/l4re-core/moe/test_namespace.t
prove -r test/t/amd64_amdfam10/l4f/l4re-core/
prove -rv test/t/amd64_amdfam10/l4f/l4re-core/
```

Debugging Tests

The test script is only a thin wrapper that sets up the test environment as it was defined in the make file and then executes two scripts: `tapper-wrapper` and `run_test`.

The main work horse of the two is `tool/bin/run_test`. It collects the necessary files and starts qemu to execute the test. This script is always required.

There is then a second script wrapped around the test runner: `tool/bin/tapper-wrapper`. This tool inspects the output of the test runner and reformats it, so that it can be read by tools like `prove`. If the test produces tap output, then the script scans for this output and filters away all the debug output. If `TEST_EXPECTED` was defined, then the script scans the output for the expected lines and prints a suitable TAP message with success or failure. It also makes sure that qemu is killed as soon as the test is finished.

There are a number of command-line parameters that allow to quickly change test parameters for debugging purposes. Run the test with '–help' for more information about available parameters.

Chapter 5

Deprecated List

Class L4::lpc_svr::Timed_work< HOOKS >

Use L4::lpc_svr::Timeout_queue_hooks

Global L4::lrq::unmask (l4_utcb_t *utcb=l4_utcb())

Use L4::lrq_eoi::unmask()

Note

wait() and receive() are doing the unmask already.

Global L4Re::Dataspace::phys (l4_addr_t offset, l4_addr_t &phys_addr, l4_size_t &phys_size)

Use L4Re::Dma_space instead. This function will be removed in future releases.

Global L4Re::Mem_alloc::free (L4::Cap< Dataspace > mem) const

This function is an empty stub which remains here for backward compatibility only. Use `L4::Task::unmap(mem, L4_FP_DELETE_OBJ)` to remove a dataspace.

Chapter 6

Module Index

6.1 Modules

Here is a list of all modules:

- Base API . . . 72
  - Basic Macros . . . 75
  - C++ IPC Interface Definition. . . . 86
    - Internal Helpers . . . 246
  - Cache Consistency . . . 90
  - Capabilities . . . 93
  - Error codes . . . 149
  - Fiasco extensions . . . 172
    - Fiasco real time scheduling extensions . . . 178
    - Kernel Debugger . . . 263
  - Flex pages . . . 180
  - Integer Types . . . 234
  - Kernel Interface Page . . . 271
    - Fiasco-UX Virtual devices . . . 179
    - Memory descriptors (C version) . . . 343
  - Kernel Objects . . . 277
    - Factory . . . 158
    - IPC-Gate API . . . 211
    - IRQs . . . 214
    - Interrupt controller . . . 250
    - L4 kernel object type information . . . 301
    - Platform Control C API . . . 390
    - Scheduler . . . 411
    - Task . . . 433
    - Thread . . . 442
      - Thread control . . . 460
      - vCPU API . . . 526
    - Virtual Console . . . 508
    - Virtual Machines . . . 520
      - VM API for SVM . . . 489
      - VM API for TZ . . . 490
      - VM API for VMX . . . 491
  - Memory operations. . . . 347
  - Memory related . . . 350
  - Object Invocation . . . 375
    - Error Handling . . . 144
    - Message Items . . . 355

Message Tag . . . 359
Realtime API . . . 397
Timeouts . . . 468
Virtual Registers (UTCBs) . . . 521
ARM Virtual Registers (UTCB) . . . 63
Buffer Registers (BRs) . . . 84
Message Registers (MRs) . . . 358
Exception registers . . . 155
Thread Control Registers (TCRs) . . . 459
amd64 Virtual Registers (UTCB) . . . 525
x86 Virtual Registers (UTCB) . . . 535
Client/Server IPC Framework . . . 102
Server-Side IPC framework . . . 415
DMA API for L4Re . . . 114
EDID parsing functionality . . . 122
IO interface . . . 203
IPC Messaging Framework . . . 209
IPC Streams . . . 210
IRQ handling library . . . 213
Interface for asynchronous ISR handlers. . . . 238
Interface for asynchronous ISR handlers with a given IRQ capability. . . . 237
Interface using direct functionality. . . . 240
Interface using direct functionality. . . . 244
L4 IPC Opcodes . . . 281
L4 V-BUS functions . . . 284
L4vbus GPIO functions . . . 316
L4vbus PCI functions . . . 325
L4 VIRTIO Interface . . . 294
L4 VIRTIO Block Device . . . 292
L4 VIRTIO Transport Layer . . . 295
L4Re C Interface . . . 303
Capability allocator . . . 97
DMA Space Interface . . . 115
Dataspace interface . . . 117
Debug interface . . . 120
Event interface . . . 152
Kumem allocator utility . . . 279
L4Re Util C Interface . . . 314
Log interface . . . 331
Memory allocator . . . 337
Namespace interface . . . 372
Region map interface . . . 401
Video API . . . 502
L4Re C++ Interface . . . 306
Auxiliary data . . . 71
C++ Exceptions . . . 85
Console API . . . 105
Debugging API . . . 121
Event API . . . 151
Initial Environment . . . 229
L4Re ELF Auxiliary Information . . . 311
L4Re Protocol identifiers . . . 313
L4Re Util C++ Interface . . . 315
Kumem utiltities . . . 280
L4Re Capability API . . . 308
Logging interface . . . 334

Name-space API . . . 370
Parent API . . . 389
Region map API . . . 398
Shared Memory Library . . . 417
Chunks . . . 98
Consumer . . . 106
Producer . . . 393
Signals . . . 426
Consumer . . . 110
Producer . . . 395
Sigma0 API . . . 422
Internal constants . . . 247
Small C++ Template Library . . . 429
Utility Functions . . . 483
Atomic Instructions . . . 64
Bit Manipulation . . . 77
Bitmap graphics and fonts . . . 83
Functions for rendering bitmap data in frame buffers . . . 192
Functions for rendering bitmap fonts to frame buffers . . . 195
CPU related functions . . . 87
Comfortable Command Line Parsing . . . 103
ELF binary format . . . 125
Functions to manipulate the local IDT . . . 198
IA32 Port I/O API . . . 199
Internal functions . . . 249
Kernel Interface Page API . . . 275
Low-Level Thread Functions . . . 335
Machine Restarting Function . . . 336
Random number support . . . 396
Timestamp Counter . . . 476
vCPU Support Library . . . 528
Extended vCPU support . . . 157

Chapter 7

Namespace Index

7.1 Namespace List

Here is a list of all documented namespaces with brief descriptions:

cxx
Our C++ library 537
cxx::Bits
Internal helpers for the cxx package 539
L4
L4 low-level kernel interface 539
L4::Ipc
IPC related functionality 547
L4::Ipc::Msg
IPC Message related functionality 552
L4::Ipc_svr
Helper classes for L4::Server instantiation 557
L4::Typeid
Definition of interface data-type helpers 557
L4::Types
L4 basic type helpers for C++ 558
L4Re
L4Re C++ Interfaces 558
L4Re::Vfs
Virtual file system for interfaces POSIX libc 562
L4virtio
L4-VIRTIO Transport C++ API 563

Chapter 8

Hierarchical Index

8.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

L4::Ipc::Array_ref< CHAR, LEN > 755
L4::Ipc::Array< CHAR, LEN > 752
L4Re::Util::Auto_cap< L4Re::Dataspace > 1103
L4Re::Util::Auto_cap< L4Re::L4Re::Dataspace > 1103
L4Re::Util::Auto_del_cap< L4::Triggerable > 1105
cxx::Bits::Base_avl_set< ITEM_TYPE, COMPARE, ALLOC, Bits::Avl_set_get_key< ITEM_TYPE > > . 605
cxx::Avl_set< ITEM_TYPE, COMPARE, ALLOC > 573
cxx::Bits::Base_avl_set< Pair< KEY_TYPE, DATA_TYPE >, COMPARE< KEY_TYPE >, ALLOC, Bits↩
::Avl_map_get_key< KEY_TYPE > > 605
cxx::Avl_map< KEY_TYPE, DATA_TYPE, COMPARE, ALLOC > 568
cxx::Bits::Base_avl_set< Pair< Region, Hdlr >, cxx::Lt_functor< Region >, Alloc, Bits::Avl_map_get_↩
key< Region > > 605
cxx::Avl_map< Region, Hdlr, cxx::Lt_functor, Alloc > 568
cxx::Base_slab< sizeof(Type), Slab_size, Max_free, Alloc > 583
cxx::Slab< Type, Slab_size, Max_free, Alloc > 673
cxx::Base_slab_static< sizeof(Type), Slab_size, Max_free, Alloc > 586
cxx::Slab_static< Type, Slab_size, Max_free, Alloc > 676
cxx::Bits::Basic_list< Bits::Basic_list_policy< T, H_list_item_t< T > > > 617
cxx::H_list< T, Bits::Basic_list_policy< T, H_list_item_t< T > > > 635
cxx::H_list_t< T > 646
cxx::Bits::Basic_list< Bits::Basic_list_policy< Weak_ref_base, H_list_item_t< Weak_ref_base > > > . 617
cxx::H_list< Weak_ref_base, Bits::Basic_list_policy< Weak_ref_base, H_list_item_t< Weak_ref_↩
base > > > 635
cxx::H_list_t< Weak_ref_base > 646
L4::Types::Bool< __Iface_conflict< I, I2 >::value||_Iface_conflict< I, LIST::type >::value > 974
L4::Types::Bool< false > 974
L4::Types::False 975
L4::Types::Same< A, B > 980
L4::Types::Bool< I1::Proto!=PROTO_EMPTY &&I1::Proto==I2::Proto > 974
L4::Types::Bool< Iface_conflict< I, L2::type >::value||_Conflict< L1::type, L2::type >::value > 974
L4::Types::Bool< true > 974
L4::Types::True 983
L4::Ipc::Msg::Is_valid_rpc_type< A * > 794
L4::Ipc::Msg::Is_valid_rpc_type< T > 794

L4::Types::Bool< Typeid::Conflict< L1::type, L2::type >::value||Conflict< L1, LIST...>::value||Conflict< L2, LIST...>::value > . . . 974
L4::Types::Bool< Typeid::Iface_conflict< I::type, L::type >::value||Iface_conflict< I, LIST...>::value > . . 974
cxx::Bits::Bst< _Node, Bits::Avl_map_get_key< KEY_TYPE >, COMPARE< KEY_TYPE > > 619
cxx::Avl_tree< _Node, Bits::Avl_map_get_key< KEY_TYPE >, COMPARE< KEY_TYPE > > 576
cxx::Bits::Bst< _Node, Bits::Avl_map_get_key< Region >, cxx::Lt_functor< Region > > 619
cxx::Avl_tree< _Node, Bits::Avl_map_get_key< Region >, cxx::Lt_functor< Region > > 576
cxx::Bits::Bst< _Node, Bits::Avl_set_get_key< ITEM_TYPE >, COMPARE > 619
cxx::Avl_tree< _Node, Bits::Avl_set_get_key< ITEM_TYPE >, COMPARE > 576
cxx::Bits::Bst< _Node, GET_KEY, COMPARE > 619
cxx::Avl_tree< _Node, GET_KEY, COMPARE > 576
cxx::Bits::Bst< Entry, Names_get_key, Compare > 619
cxx::Avl_tree< Entry, Names_get_key > 576
L4::Cap< L4::Factory > 706
L4::Cap< L4::Irq > 706
L4::Cap< L4::Semaphore > 706
L4::Cap< L4::Thread > 706
L4::Cap< L4::Vcon > 706
L4::Cap< L4Re::Dataspace > 706
L4::Cap< L4Re::L4Re::Dataspace > 706
L4::Cap< L4Re::L4Re::Namespace > 706
L4::Cap< L4Re::Rm > 706
L4::Cap< L4Re::Video::Goos > 706
L4::Cap< L4vbus::Vbus > 706
L4::Cap< void > 706
L4::Ipc::Msg::Clnt_val_ops< Detail::_Plain< T >::type, DIR, CLASS > 779
cxx::Auto_ptr< T > 565
cxx::Base_slab< Obj_size, Slab_size, Max_free, Alloc > 583
cxx::Base_slab_static< Obj_size, Slab_size, Max_free, Alloc > 586
cxx::Bitfield< T, LSB, MSB > 589
cxx::Bitmap_base 592
cxx::Bitmap< BITS > 589
cxx::Bitmap_base::Bit 602
cxx::Bitmap_base::Char< BITS > 602
cxx::Bitmap_base::Word< BITS > 603
cxx::Bits::Avl_map_get_key< KEY_TYPE > 604
cxx::Bits::Avl_set_get_key< KEY_TYPE > 605
cxx::Bits::Base_avl_set< ITEM_TYPE, COMPARE, ALLOC, GET_KEY > 605
cxx::Bits::Base_avl_set< ITEM_TYPE, COMPARE, ALLOC, GET_KEY >::Node 615
cxx::Bits::Basic_list< POLICY > 617
cxx::H_list< T, POLICY > 635
cxx::S_list< T, POLICY > 669
cxx::H_list< Observer > 635
cxx::H_list< Slab_i > 635
cxx::H_list< T > 635
cxx::H_list< Timeout > 635
cxx::S_list< T > 669
cxx::Bits::Bst< Node, Get_key, Compare > 619
cxx::Avl_tree< Node, Get_key, Compare > 576
cxx::Bits::Bst_node 631
cxx::Avl_tree_node 581
cxx::Bits::Direction 633
cxx::H_list_item_t< ELEM_TYPE > 643
L4::Ipc_svr::Timeout 843
cxx::List< D, Alloc > 648

cxx::List< D, Alloc >::Iter . . . 650
cxx::List_alloc . . . 651
cxx::List_item . . . 653
cxx::List_item::Iter . . . 657
  cxx::List_item::T_iter< T, Poly > . . . 659
  cxx::List_item::T_iter< E > . . . 659
cxx::Lt_functor< Obj > . . . 661
cxx::New_allocator< _Type > . . . 661
cxx::Nothrow . . . 662
cxx::Pair< First, Second > . . . 662
cxx::Pair_first_compare< Cmp, Typ > . . . 664
cxx::Ref_ptr< T, CNT > . . . 665
cxx::String . . . 679
  L4Re::Util::Names::Name . . . 1135
Elf32_Dyn . . . 686
Elf32_Ehdr . . . 687
Elf32_Phdr . . . 689
Elf32_Shdr . . . 691
Elf32_Sym . . . 692
Elf64_Dyn . . . 693
Elf64_Ehdr . . . 694
Elf64_Phdr . . . 695
Elf64_Shdr . . . 697
Elf64_Sym . . . 698
gfxbitmap_offset . . . 699
cxx::H_list_item_t< Weak_ref_base > . . . 643
  cxx::Weak_ref_base . . . 684
    cxx::Weak_ref< T > . . . 681
L4::Kobject_2t< Console, Video::Goos, Event, L4::PROTO_EMPTY > . . . 876
  L4Re::Console . . . 1036
L4::Kobject_2t< Debug_obj_t< BASE >, BASE, Debug_obj, L4::PROTO_EMPTY > . . . 876
L4::Kobject_t< Pager, Io_pager, L4_PROTO_PAGE_FAULT > . . . 883
L4::Kobject_t< Rm, L4::Pager, L4RE_PROTO_RM, L4::Type_info::Demand_t< 1 > > . . . 883
  L4Re::Rm . . . 1093
L4::Kobject_x< Iommu, Proto_t< L4_PROTO_IOMMU >, Type_info::Demand_t< 1 > > . . . 887
  L4::Iommu . . . 747
L4::Alloc_list . . . 699
L4::Basic_registry . . . 701
  L4Re::Util::Object_registry . . . 1141
L4::Cap< T > . . . 706
L4::Exception_tracer . . . 723
  L4::Base_exception . . . 700
    L4::Invalid_capability . . . 745
    L4::Runtime_error . . . 900
      L4::Bounds_error . . . 704
      L4::Com_error . . . 708
      L4::Element_already_exists . . . 718
      L4::Element_not_found . . . 721
      L4::Out_of_memory . . . 890
    L4::Unknown_error . . . 984
L4::Factory::Lstr . . . 733
L4::Factory::Nil . . . 734
L4::Factory::S . . . 734
L4::IOModifier . . . 751
L4::Ipc::Array_in_buf< ELEM_TYPE, LEN_TYPE, MAX > . . . 754
L4::Ipc::Array_ref< ELEM_TYPE, LEN_TYPE > . . . 755

L4::Ipc::Array< ELEM_TYPE, LEN_TYPE > . . . 752
L4::Ipc::As_value< T > . . . 757
L4::Ipc::Buf_item . . . 758
L4::Ipc::Call . . . 759
L4::Ipc::Call_t< RIGHTS > . . . 760
L4::Ipc::Call_zero_send_timeout . . . 761
L4::Ipc::Cap< T > . . . 763
L4::Ipc::Gen_fpage< T > . . . 765
L4::Ipc::In_out< T > . . . 767
L4::Ipc::Istream . . . 772
L4::Ipc::Iostream . . . 767
L4::Ipc::Msg::Clnt_val_ops< MTYPE, DIR, CLASS > . . . 779
L4::Ipc::Msg::Cls_buffer . . . 781
L4::Ipc::Msg::Do_rcv_buffers . . . 790
L4::Ipc::Msg::Cls_data . . . 782
L4::Ipc::Msg::Do_in_data . . . 785
L4::Ipc::Msg::Do_out_data . . . 788
L4::Ipc::Msg::Cls_item . . . 783
L4::Ipc::Msg::Do_in_items . . . 787
L4::Ipc::Msg::Do_out_items . . . 789
L4::Ipc::Msg::Dir_in . . . 783
L4::Ipc::Msg::Do_in_data . . . 785
L4::Ipc::Msg::Do_in_items . . . 787
L4::Ipc::Msg::Do_rcv_buffers . . . 790
L4::Ipc::Msg::Dir_out . . . 784
L4::Ipc::Msg::Do_out_data . . . 788
L4::Ipc::Msg::Do_out_items . . . 789
L4::Ipc::Msg::Elem< Array< A, LEN > & > . . . 792
L4::Ipc::Msg::Elem< Array< A, LEN > > . . . 792
L4::Ipc::Msg::Elem< Array_ref< A, LEN > & > . . . 793
L4::Ipc::Msg::Svr_arg_pack< IPC_TYPE > . . . 797
L4::Ipc::Msg::Svr_val_ops< MTYPE, DIR, CLASS > . . . 797
L4::Ipc::Msg_ptr< T > . . . 798
L4::Ipc::Opt< T > . . . 799
L4::Ipc::Ostream . . . 801
L4::Ipc::Iostream . . . 767
L4::Ipc::Out< T > . . . 806
L4::Ipc::Ret_array< T > . . . 806
L4::Ipc::Send_only . . . 807
L4::Ipc::Small_buf . . . 808
L4::Ipc::Snd_item . . . 810
L4::Ipc::Str_cp_in< T > . . . 810
L4::Ipc::Varg . . . 812
L4::Ipc::Varg_list_ref . . . 818
L4::Ipc::Varg_list< MAX > . . . 817
L4::Ipc_svr::Compound_reply . . . 826
L4::Ipc_svr::Default_loop_hooks . . . 827
L4Re::Util::Br_manager_hooks . . . 1111
L4::Ipc_svr::Default_setup_wait . . . 829
L4::Ipc_svr::Default_timeout . . . 829
L4::Ipc_svr::Default_loop_hooks . . . 827
L4Re::Util::Br_manager_hooks . . . 1111
L4::Ipc_svr::Direct_dispatch< R > . . . 830
L4::Ipc_svr::Exc_dispatch< R, Exc > . . . 833
L4::Ipc_svr::Direct_dispatch< R * > . . . 832

L4::Ipc_svr::Ignore_errors . . . 834
L4::Ipc_svr::Default_loop_hooks . . . 827
L4Re::Util::Br_manager_hooks . . . 1111
L4Re::Util::Br_manager_timeout_hooks . . . 1112
L4::Ipc_svr::Server_iface . . . 835
L4::Ipc_svr::Br_manager_no_buffers . . . 823
L4::Ipc_svr::Default_loop_hooks . . . 827
L4::Ipc_svr::Timeout_queue_hooks< HOOKS, BR_MAN > . . . 851
L4Re::Util::Br_manager . . . 1106
L4Re::Util::Br_manager_hooks . . . 1111
L4::Ipc_svr::Timeout_queue_hooks< Br_manager_timeout_hooks, Br_manager > . . . 851
L4Re::Util::Br_manager_timeout_hooks . . . 1112
L4::Ipc_svr::Timed_work< HOOKS > . . . 842
L4::Ipc_svr::Timeout_queue . . . 845
L4::Irq_eoi . . . 861
L4::Kobject_t< Icu, Irq_eoi, L4_PROTO_IRQ, Type_info::Demand_t< 1 > > . . . 883
L4::Icu . . . 737
L4::Kobject_t< Event, L4::Icu, L4RE_PROTO_EVENT > . . . 883
L4Re::Event . . . 1066
L4::Kobject_3t< Vbus, L4Re::Dataspace, L4Re::Inhibitor, L4Re::Event > . . . 879
L4vbus::Vbus . . . 1271
L4::Kobject_t< Scheduler, Icu, L4_PROTO_SCHEDULER, Type_info::Demand_t< 1 > > . . . 883
L4::Scheduler . . . 903
L4::Kobject_t< Vcon, Icu, L4_PROTO_LOG > . . . 883
L4::Vcon . . . 987
L4::Kobject_t< Log, L4::Vcon, L4::PROTO_EMPTY > . . . 883
L4Re::Log . . . 1077
L4::Kobject_t< Triggerable, Irq_eoi, L4_PROTO_IRQ > . . . 883
L4::Triggerable . . . 946
L4::Kobject_t< Irq, Triggerable, L4_PROTO_IRQ_SENDER > . . . 883
L4::Irq . . . 855
L4::Kobject_t< Irq_mux, Triggerable, L4_PROTO_IRQ_MUX > . . . 883
L4::Irq_mux . . . 866
L4::Kobject_t< Semaphore, Triggerable, L4_PROTO_SEMAPHORE > . . . 883
L4::Semaphore . . . 906
L4::Kip::Mem_desc . . . 868
L4::Kobject . . . 874
L4::Kobject_t< Dataspace, L4::Kobject, L4RE_PROTO_DATASPACE, L4::Type_info::Demand_t< 1 > > . . . 883
L4Re::Dataspace . . . 1039
L4::Kobject_3t< Vbus, L4Re::Dataspace, L4Re::Inhibitor, L4Re::Event > . . . 879
L4::Kobject_t< Debug_obj, L4::Kobject, L4RE_PROTO_DEBUG > . . . 883
L4Re::Debug_obj . . . 1050
L4::Kobject_t< Debugger, Kobject, L4_PROTO_DEBUGGER > . . . 883
L4::Debugger . . . 711
L4::Kobject_t< Derived, L4::Kobject, PROTO, S_DEMAND > . . . 883
L4::Kobject_t< Device, L4::Kobject, L4VIRTIO_PROTOCOL, L4::Type_info::Demand_t< 1 > > . . . 883
L4::Kobject_t< Dma_space, L4::Kobject, PROTO, L4::Type_info::Demand_t< 1 > > . . . 883
L4::Kobject_t< Exception, L4::Kobject, PROTO, S_DEMAND > . . . 883
L4::Kobject_t< Factory, Kobject, L4_PROTO_FACTORY > . . . 883
L4::Factory . . . 724
L4::Kobject_t< Mem_alloc, L4::Factory, L4::PROTO_EMPTY > . . . 883
L4Re::Mem_alloc . . . 1080
L4::Kobject_t< Goos, L4::Kobject, L4RE_PROTO_GOOS > . . . 883
L4Re::Video::Goos . . . 1193

L4::Kobject_t< Inhibitor, L4::Kobject, L4RE_PROTO_INHIBITOR > . . . 883
L4Re::Inhibitor . . . 1073
L4::Kobject_3t< Vbus, L4Re::Dataspace, L4Re::Inhibitor, L4Re::Event > . . . 879
L4::Kobject_t< Io_pager, L4::Kobject, PROTO, S_DEMAND > . . . 883
L4::Kobject_t< Ipc_gate, Kobject, L4_PROTO_KOBJECT, Type_info::Demand_t< 1 > > . . . 883
L4::Ipc_gate . . . 820
L4::Kobject_t< Irq_eoi, L4::Kobject, PROTO, S_DEMAND > . . . 883
L4::Kobject_t< Meta, Kobject, L4_PROTO_META > . . . 883
L4::Kobject_t< Namespace, L4::Kobject, L4RE_PROTO_NAMESPACE, L4::Type_info::Demand_t< 1 > > . . . 883
L4Re::Namespace . . . 1084
L4::Kobject_t< Parent, L4::Kobject, L4RE_PROTO_PARENT > . . . 883
L4Re::Parent . . . 1089
L4::Kobject_t< Platform_control, Kobject, L4_PROTO_PLATFORM_CTL > . . . 883
L4::Platform_control . . . 893
L4::Kobject_t< Task, Kobject, L4_PROTO_TASK, Type_info::Demand_t< 2 > > . . . 883
L4::Task . . . 922
L4::Kobject_t< Vm, Task, L4_PROTO_VM > . . . 883
L4::Vm . . . 995
L4::Kobject_t< Thread, Kobject, L4_PROTO_THREAD, Type_info::Demand_t< 1 > > . . . 883
L4::Thread . . . 931
L4::Kobject_2t< Derived, Base1, Base2, PROTO, S_DEMAND > . . . 876
L4::Kobject_3t< Derived, Base1, Base2, Base3, PROTO, S_DEMAND > . . . 879
L4::Kobject_demand< T > . . . 883
L4::Kobject_t< Derived, Base, PROTO, S_DEMAND > . . . 883
L4::Kobject_typeid< T > . . . 885
L4::Kobject_x< Derived, ARGS > . . . 887
L4::Poll_timeout_kipclock . . . 897
L4::Proto_t< P > . . . 900
L4::Server< LOOP_HOOKS > . . . 910
L4Re::Util::Registry_server< LOOP_HOOKS > . . . 1147
L4::Server_object . . . 914
L4::Server_object_x< Derived, IFACE, BASE > . . . 918
L4::Server_object_t< IFACE, BASE > . . . 915
L4::Server_object_x< Derived, IFACE, BASE > . . . 918
L4::Smart_cap< T, SMART > . . . 920
L4::String . . . 922
L4::Thread::Attr . . . 941
L4::Thread::Modify_senders . . . 945
L4::Type_info . . . 949
L4::Type_info::Demand . . . 950
L4::Type_info::Demand_t< __I::Max< D1::Caps, D2::Caps >::Res, D1::Flags|D2::Flags, __I::Max< D1::Mem, D2::Mem >::Res, __I::Max< D1::Ports, D2::Ports >::Res > . . . 953
L4::Type_info::Demand_union_t< D1, D2 > . . . 956
L4::Type_info::Demand_t< __I::Max< Kobject_typeid< T1 >::Demand::Caps, Kobject_demand< T2...>::Caps >::Res, Kobject_typeid< T1 >::Demand::Flags|Kobject_demand< T2...>:↩ :Flags, __I::Max< Kobject_typeid< T1 >::Demand::Mem, Kobject_demand< T2...>::Mem >::Res, __I::Max< Kobject_typeid< T1 >::Demand::Ports, Kobject_demand< T2...>::Ports >::Res > . . . 953
L4::Type_info::Demand_union_t< Kobject_typeid< T1 >::Demand, Kobject_demand< T2...> > . . . 956
L4::Type_info::Demand_t<> . . . 953
L4::Type_info::Demand_t< CAPS, FLAGS, MEM, PORTS > . . . 953
L4::Typeid::Detail::_Rpcs< OPCODE, O, Default_op< R > >::Rpc< Y > . . . 959
L4::Typeid::Detail::_Rpcs< OPCODE, O, R, X...> . . . 960
L4::Typeid::Detail::_Rpcs< OPCODE, O, R, X...>::Rpc< Y > . . . 961
L4::Typeid::Detail::Rpcs_end . . . 962

L4::Typeid::Detail::_Rpcs< L4::Opcode, 0, RPCS...> . . . 958
L4::Typeid::Rpcs< RPCS > . . . 967
L4::Typeid::Detail::_Rpcs< l4_umword_t, 0, ARG...> . . . 958
L4::Typeid::Rpcs_sys< ARG > . . . 972
L4::Typeid::Detail::_Rpcs< OPCODE_TYPE, 0, RPCS...> . . . 958
L4::Typeid::Rpcs_code< OPCODE_TYPE >::F< RPCS > . . . 969
L4::Typeid::Detail::_Rpcs< void, 0, OPERATION > . . . 958
L4::Typeid::Rpc_nocode< OPERATION > . . . 964
L4::Typeid::Detail::_Rpcs< OPCODE, O, X > . . . 958
L4::Typeid::P_dispatch< LIST > . . . 963
L4::Typeid::Raw_ipc< CLASS > . . . 963
L4::Typeid::Rpcs_code< OPCODE_TYPE > . . . 969
L4::Types::Bool< V > . . . 974
L4::Types::Flags< BITS_ENUM, UNDERLYING > . . . 976
l4_buf_regs_t . . . 997
l4_exc_regs_t . . . 997
l4_fpage_t . . . 1000
l4_icu_info_t . . . 1001
L4::Icu::Info . . . 744
l4_icu_msi_info_t . . . 1002
l4_kernel_info_mem_desc_t . . . 1003
l4_msg_regs_t . . . 1004
l4_msgtag_t . . . 1004
l4_sched_cpu_set_t . . . 1006
l4_sched_param_t . . . 1008
l4_snd_fpage_t . . . 1009
l4_thread_regs_t . . . 1010
l4_timeout_s . . . 1012
l4_timeout_t . . . 1012
l4_tracebuffer_status_t . . . 1013
l4_tracebuffer_status_window_t . . . 1015
l4_vcon_attr_t . . . 1016
l4_vcpu_ipc_regs_t . . . 1017
l4_vcpu_regs_t . . . 1018
l4_vcpu_state_t . . . 1021
L4vcpu::Vcpu . . . 1278
l4_vhw_descriptor . . . 1024
l4_vhw_entry . . . 1026
l4_vm_svm_vmcb_control_area . . . 1029
l4_vm_svm_vmcb_state_save_area . . . 1029
l4_vm_svm_vmcb_state_save_area_seg . . . 1031
l4_vm_svm_vmcb_t . . . 1031
l4_vm_tz_state . . . 1033
L4Re::Cap_alloc . . . 1033
L4Re::Dataspace::Stats . . . 1049
L4Re::Dma_space . . . 1053
L4Re::Env . . . 1057
L4Re::Event_buffer_t< PAYLOAD > . . . 1069
L4Re::Util::Event_buffer_t< PAYLOAD > . . . 1128
L4Re::Util::Event_buffer_consumer_t< PAYLOAD > . . . 1125
L4Re::Event_buffer_t< PAYLOAD >::Event . . . 1072
L4Re::Smart_cap_auto< Unmap_flags > . . . 1103
L4Re::Util::Auto_cap< T > . . . 1103
L4Re::Util::Auto_del_cap< T > . . . 1105
L4Re::Util::Cap_alloc_base . . . 1115
L4Re::Util::Counting_cap_alloc< COUNTERTYPE > . . . 1116
L4Re::Util::Dataspace_svr . . . 1120

L4Re::Util::Event_t< PAYLOAD > . . . 1132
L4Re::Util::Item_alloc_base . . . 1135
L4Re::Util::Names::Name_space . . . 1137
L4Re::Util::Ref_cap< T > . . . 1145
L4Re::Util::Ref_del_cap< T > . . . 1146
L4Re::Util::Smart_cap_auto< Unmap_flags > . . . 1149
L4Re::Util::Smart_count_cap< Unmap_flags > . . . 1150
L4Re::Util::Vcon_svr< SVR > . . . 1151
L4Re::Util::Video::Goos_svr . . . 1152
L4Re::Vfs::Directory . . . 1163
  L4Re::Vfs::File . . . 1169
    L4Re::Vfs::Be_file . . . 1155
L4Re::Vfs::File_system . . . 1170
  L4Re::Vfs::Be_file_system . . . 1160
L4Re::Vfs::Fs . . . 1173
  L4Re::Vfs::Ops . . . 1182
L4Re::Vfs::Generic_file . . . 1176
  L4Re::Vfs::File . . . 1169
L4Re::Vfs::Mman . . . 1180
  L4Re::Vfs::Ops . . . 1182
L4Re::Vfs::Regular_file . . . 1183
  L4Re::Vfs::File . . . 1169
L4Re::Vfs::Special_file . . . 1188
  L4Re::Vfs::File . . . 1169
L4Re::Video::Color_component . . . 1190
L4Re::Video::Goos::Info . . . 1198
L4Re::Video::Pixel_info . . . 1200
L4Re::Video::View . . . 1206
L4Re::Video::View::Info . . . 1210
l4re_aux_t . . . 1212
l4re_ds_stats_t . . . 1213
l4re_elf_aux_mword_t . . . 1214
l4re_elf_aux_t . . . 1215
l4re_elf_aux_vma_t . . . 1215
l4re_env_cap_entry_t . . . 1216
l4re_env_t . . . 1218
l4re_event_t . . . 1219
l4re_video_color_component_t . . . 1220
l4re_video_goos_info_t . . . 1221
l4re_video_pixel_info_t . . . 1222
l4re_video_view_info_t . . . 1223
l4re_video_view_t . . . 1224
l4util_idt_desc_t . . . 1225
l4util_idt_header_t . . . 1225
l4util_mb_addr_range_t . . . 1226
l4util_mb_apm_t . . . 1227
l4util_mb_drive_t . . . 1228
l4util_mb_info_t . . . 1229
l4util_mb_mod_t . . . 1231
l4util_mb_vbe_ctrl_t . . . 1232
l4util_mb_vbe_mode_t . . . 1233
L4vbus::Gpio_module::Pin_slice . . . 1249
L4vbus::Pm< DEC > . . . 1269
l4vbus_device_t . . . 1275
l4vbus_resource_t . . . 1276
L4vcpu::State . . . 1277

L4virtio::Ptr< T > . . . 1293
L4virtio::Svr::Bad_descriptor . . . 1295
L4virtio::Svr::Data_buffer . . . 1298
L4virtio::Svr::Dev_config . . . 1301
L4virtio::Svr::Dev_features . . . 1308
L4virtio::Svr::Dev_status . . . 1310
L4virtio::Svr::Device_t< DATA > . . . 1314
L4virtio::Svr::Driver_mem_list_t< DATA > . . . 1318
L4virtio::Svr::Driver_mem_region_t< DATA > . . . 1326
L4virtio::Svr::Request_processor . . . 1331
L4virtio::Svr::Virtqueue::Head_desc . . . 1340
L4virtio::Virtqueue . . . 1342
  L4virtio::Driver::Virtqueue . . . 1288
  L4virtio::Svr::Virtqueue . . . 1336
L4virtio::Virtqueue::Avail . . . 1351
L4virtio::Virtqueue::Avail::Flags . . . 1352
L4virtio::Virtqueue::Desc . . . 1354
L4virtio::Virtqueue::Desc::Flags . . . 1355
L4virtio::Virtqueue::Used . . . 1359
L4virtio::Virtqueue::Used::Flags . . . 1360
L4virtio::Virtqueue::Used_elem . . . 1362
l4virtio_block_config_t . . . 1363
l4virtio_block_header_t . . . 1364
l4virtio_config_hdr_t . . . 1364
l4virtio_config_queue_t . . . 1366
L4vbus::Pm< Device > . . . 1269
  L4vbus::Device . . . 1236
    L4vbus::Gpio_module . . . 1243
    L4vbus::Gpio_pin . . . 1250
    L4vbus::Icu . . . 1257
    L4vbus::Pci_dev . . . 1260
    L4vbus::Pci_host_bridge . . . 1265
L4virtio::Ptr< void > . . . 1293
cxx::Ref_ptr< L4Re::Vfs::File > . . . 665
cxx::Ref_ptr< Mount_tree > . . . 665
L4::Server_object_t< Kobject > . . . 915
  L4::Irq_handler_object . . . 863
L4::Ipc::Msg::Svr_val_ops< Array_ref< A, LEN >, Dir_in, CLASS > . . . 797
L4::Ipc::Msg::Svr_val_ops< Array_ref< A, LEN >, Dir_out, CLASS > . . . 797
L4::Ipc::Msg::Svr_val_ops< L4::Ipc::Snd_fpage, Dir_in, CLASS > . . . 797
cxx::Bitmap_base::Word< Bits > . . . 603
cxx::Bitmap_base::Word< Size > . . . 603

Chapter 9

Data Structure Index

9.1 Data Structures

Here are the data structures with brief descriptions:

cxx::Auto_ptr< T >
Smart pointer with automatic deletion 565
cxx::Avl_map< KEY_TYPE, DATA_TYPE, COMPARE, ALLOC >
AVL tree based associative container 568
cxx::Avl_set< ITEM_TYPE, COMPARE, ALLOC >
AVL set for simple compareable items 573
cxx::Avl_tree< Node, Get_key, Compare >
A generic AVL tree 576
cxx::Avl_tree_node
Node of an AVL tree 581
cxx::Base_slab< Obj_size, Slab_size, Max_free, Alloc >
Basic slab allocator 583
cxx::Base_slab_static< Obj_size, Slab_size, Max_free, Alloc >
Merged slab allocator (allocators for objects of the same size are merged together) 586
cxx::Bitfield< T, LSB, MSB >
Definition for a member (part) of a bit field 589
cxx::Bitmap< BITS >
A static bit map 589
cxx::Bitmap_base
Basic bitmap abstraction 592
cxx::Bitmap_base::Bit
A writeable bit in a bitmap 602
cxx::Bitmap_base::Char< BITS >
Helper abstraction for a byte contained in the bitmap 602
cxx::Bitmap_base::Word< BITS >
Helper abstraction for a word contained in the bitmap 603
cxx::Bits::Avl_map_get_key< KEY_TYPE >
Key-getter for Avl_map 604
cxx::Bits::Avl_set_get_key< KEY_TYPE >
Internal, key-getter for Avl_set nodes 605
cxx::Bits::Base_avl_set< ITEM_TYPE, COMPARE, ALLOC, GET_KEY >
Internal: AVL set with internally managed nodes 605
cxx::Bits::Base_avl_set< ITEM_TYPE, COMPARE, ALLOC, GET_KEY >::Node
A smart pointer to a tree item 615
cxx::Bits::Basic_list< POLICY >
Internal: Common functions for all head-based list implementations 617
cxx::Bits::Bst< Node, Get_key, Compare >
Basic binary search tree (BST) 619

cxx::Bits::Bst_node
Basic type of a node in a binary search tree (BST) 631
cxx::Bits::Direction
The direction to go in a binary search tree 633
cxx::H_list< T, POLICY >
General double-linked list of unspecified cxx::H_list_item elements 635
cxx::H_list_item_t< ELEM_TYPE >
Basic element type for a double-linked H_list 643
cxx::H_list_t< T >
Double-linked list of typed H_list_item_t elements 646
cxx::List< D, Alloc >
Doubly linked list, with internal allocation 648
cxx::List< D, Alloc >::Iter
Iterator 650
cxx::List_alloc
Standard list-based allocator 651
cxx::List_item
Basic list item 653
cxx::List_item::Iter
Iterator for a list of ListItem-s 657
cxx::List_item::T_iter< T, Poly >
Iterator for derived classes from ListItem 659
cxx::Lt_functor< Obj >
Generic comparator class that defaults to the less-than operator 661
cxx::New_allocator< _Type >
Standard allocator based on `operator new ()` 661
cxx::Nothrow
Helper type to distinguish the `oeprator new` version that does not throw exceptions . . . 662
cxx::Pair< First, Second >
Pair of two values 662
cxx::Pair_first_compare< Cmp, Typ >
Comparison functor for Pair 664
cxx::Ref_ptr< T, CNT >
A reference-counting pointer with automatic cleanup 665
cxx::S_list< T, POLICY >
Simple single-linked list 669
cxx::Slab< Type, Slab_size, Max_free, Alloc >
Slab allocator for object of type *Type* 673
cxx::Slab_static< Type, Slab_size, Max_free, Alloc >
Merged slab allocator (allocators for objects of the same size are merged together) 676
cxx::String
This class is used to group characters of a string which belong to one syntactical token types number, identifier, string, whitespace or another single character 679
cxx::Weak_ref< T >
Typed weak reference to an object of type `T` 681
cxx::Weak_ref_base
Generic (base) weak reference to some object 684
Elf32_Dyn
ELF32 dynamic entry 686
Elf32_Ehdr
ELF32 header 687
Elf32_Phdr
ELF32 program header 689
Elf32_Shdr
ELF32 section header - figure 1-9, page 1-9 691
Elf32_Sym
ELF32 symbol table entry 692

Elf64_Dyn
ELF64 dynamic entry . . . 693
Elf64_Ehdr
ELF64 header . . . 694
Elf64_Phdr
ELF64 program header . . . 695
Elf64_Shdr
ELF64 section header . . . 697
Elf64_Sym
ELF64 symbol table entry . . . 698
gfxbitmap_offset
Offsets in pmap[] and bmap[] . . . 699
L4::Alloc_list
A simple list-based allocator . . . 699
L4::Base_exception
Base class for all exceptions, thrown by the L4Re framework . . . 700
L4::Basic_registry
This registry returns the corresponding server object based on the label of an Ipc_gate . . . 701
L4::Bounds_error
Access out of bounds . . . 704
L4::Cap< T >
C++ interface for capabilities . . . 706
L4::Com_error
Error conditions during IPC . . . 708
L4::Debugger
C++ debugger interface . . . 711
L4::Element_already_exists
Exception for duplicate element insertions . . . 718
L4::Element_not_found
Exception for a failed lookup (element not found) . . . 721
L4::Exception_tracer
Back-trace support for exceptions . . . 723
L4::Factory
C++ Factory interface to create kernel objects . . . 724
L4::Factory::Lstr
Special type to add a pascal string into the factory create stream . . . 733
L4::Factory::Nil
Special type to add a void argument into the factory create stream . . . 734
L4::Factory::S
Stream class for the create() argument stream . . . 734
L4::Icu
C++ Icu interface . . . 737
L4::Icu::Info
This class encapsulates information about an ICU . . . 744
L4::Invalid_capability
Indicates that an invalid object was invoked . . . 745
L4::Iommu
Interface for IO-MMUs used for DMA remapping . . . 747
L4::IOModifier
Modifier class for the IO stream . . . 751
L4::Ipc::Array< ELEM_TYPE, LEN_TYPE >
Array data type for dynamically sized arrays in RPCs . . . 752
L4::Ipc::Array_in_buf< ELEM_TYPE, LEN_TYPE, MAX >
Server-side copy in buffer for Array . . . 754
L4::Ipc::Array_ref< ELEM_TYPE, LEN_TYPE >
Array reference data type for arrays located in the message . . . 755
L4::Ipc::As_value< T >
Pass the argument as plain data value . . . 757

L4::Ipc::Buf_item
RPC warpper for a receive item . . . 758
L4::Ipc::Call
RPC attribute for a standard RPC call . . . 759
L4::Ipc::Call_t< RIGHTS >
RPC attribute for an RPC call with required rights . . . 760
L4::Ipc::Call_zero_send_timeout
RPC attribute for an RPC call, with zero send timeout . . . 761
L4::Ipc::Cap< T >
Capability type for RPC interfaces (see `L4::Cap<T>`) . . . 763
L4::Ipc::Gen_fpage< T >
Generic RPC wrapper for L4 flex-pages . . . 765
L4::Ipc::In_out< T >
Mark an argument as in-out argument . . . 767
L4::Ipc::Iostream
Input/Output stream for IPC [un]marshalling . . . 767
L4::Ipc::Istream
Input stream for IPC unmarshalling . . . 772
L4::Ipc::Msg::Clnt_val_ops< MTYPE, DIR, CLASS >
Defines client-side handling of 'MTYPE' as RPC argument . . . 779
L4::Ipc::Msg::Cls_buffer
Marker type for receive buffer values . . . 781
L4::Ipc::Msg::Cls_data
Marker type for data values . . . 782
L4::Ipc::Msg::Cls_item
Marker type for item values . . . 783
L4::Ipc::Msg::Dir_in
Marker type for input values . . . 783
L4::Ipc::Msg::Dir_out
Marker type for output values . . . 784
L4::Ipc::Msg::Do_in_data
Marker for Input data . . . 785
L4::Ipc::Msg::Do_in_items
Marker for Input items . . . 787
L4::Ipc::Msg::Do_out_data
Marker for Output data . . . 788
L4::Ipc::Msg::Do_out_items
Marker for Output items . . . 789
L4::Ipc::Msg::Do_rcv_buffers
Marker for receive buffers . . . 790
L4::Ipc::Msg::Elem< Array< A, LEN > & >
Array as output argument . . . 792
L4::Ipc::Msg::Elem< Array< A, LEN > >
Array as input arguments . . . 792
L4::Ipc::Msg::Elem< Array_ref< A, LEN > & >
Array_ref as output argument . . . 793
L4::Ipc::Msg::Is_valid_rpc_type< T >
Type trait defining a valid RPC parameter type . . . 794
L4::Ipc::Msg::Svr_arg_pack< IPC_TYPE >
Server-side RPC arguments data structure used to provide arguments to the server-side implementation of an RPC function . . . 797
L4::Ipc::Msg::Svr_val_ops< MTYPE, DIR, CLASS >
Defines server-side handling for `MTYPE` server arguments . . . 797
L4::Ipc::Msg_ptr< T >
Pointer to an element of type T in an Ipc::Istream . . . 798
L4::Ipc::Opt< T >
Attribute for defining an optional RPC argument . . . 799

L4::Ipc::Ostream
Output stream for IPC marshalling . . . 801
L4::Ipc::Out< T >
Mark an argument as a output value in an RPC signature . . . 806
L4::Ipc::Ret_array< T >
Dynamically sized output array of type T . . . 806
L4::Ipc::Send_only
RPC attribute for a send-only RPC . . . 807
L4::Ipc::Small_buf
A receive item for receiving a single capability . . . 808
L4::Ipc::Snd_item
RPC wrapper for a send item . . . 810
L4::Ipc::Str_cp_in< T >
Abstraction for extracting a zero-termintaed string from an Ipc::Istream . . . 810
L4::Ipc::Varg
Variably sized RPC argument . . . 812
L4::Ipc::Varg_list< MAX >
Self-contained list of variable-sized RPC parameters . . . 817
L4::Ipc::Varg_list_ref
List of variable-sized RPC parameters as received by the server . . . 818
L4::Ipc_gate
The C++ IPC gate interface . . . 820
L4::Ipc_svr::Br_manager_no_buffers
Empty implementation of Server_iface . . . 823
L4::Ipc_svr::Compound_reply
Mix in for LOOP_HOOKS to always use compound reply and wait . . . 826
L4::Ipc_svr::Default_loop_hooks
Default LOOP_HOOKS . . . 827
L4::Ipc_svr::Default_setup_wait
Mix in for LOOP_HOOKS for setup_wait no op . . . 829
L4::Ipc_svr::Default_timeout
Mix in for LOOP_HOOKS to use a 0 send and a infinite receive timeout . . . 829
L4::Ipc_svr::Direct_dispatch< R >
Direct disptach helper, for forwarding dispatch calls a registry *R* . . . 830
L4::Ipc_svr::Direct_dispatch< R * >
Direct disptach helper, for forwarding dispatch calls via a pointer to a registry *R* . . . 832
L4::Ipc_svr::Exc_dispatch< R, Exc >
Dispatch helper wrapping try {} catch {} around the dispatch call . . . 833
L4::Ipc_svr::Ignore_errors
Mix in for LOOP_HOOKS to ignore IPC errors . . . 834
L4::Ipc_svr::Server_iface
Interface for server-loop related functions . . . 835
L4::Ipc_svr::Timed_work< HOOKS >
DEPRECATED . . . 842
L4::Ipc_svr::Timeout
Callback interface for Timeout_queue . . . 843
L4::Ipc_svr::Timeout_queue
Timeout queue to be used in l4re server loop . . . 845
L4::Ipc_svr::Timeout_queue_hooks< HOOKS, BR_MAN >
Loop hooks mixin for integrating a timeout queue into the server loop . . . 851
L4::Irq
C++ Irq interface . . . 855
L4::Irq_eoi
Interface for sending an acknowledge message to an object . . . 861
L4::Irq_handler_object
Server object base class for handling IRQ messages . . . 863
L4::Irq_mux
IRQ multiplexer for shared IRQs . . . 866

L4::Kip::Mem_desc
Memory descriptors stored in the kernel interface page 868
L4::Kobject
Base class for all kinds of kernel objects and remote objects, referenced by capabilities 874
L4::Kobject_2t< Derived, Base1, Base2, PROTO, S_DEMAND >
Helper class to create an L4Re interface class that is derived from two base classes (see L4↩ ::Kobject_t) 876
L4::Kobject_3t< Derived, Base1, Base2, Base3, PROTO, S_DEMAND >
Helper class to create an L4Re interface class that is derived from three base classes (see L4::Kobject_t) 879
L4::Kobject_demand< T >
Get the combined server-side resource requirements for all type T.. 883
L4::Kobject_t< Derived, Base, PROTO, S_DEMAND >
Helper class to create an L4Re interface class that is derived from a single base class 883
L4::Kobject_typeid< T >
Meta object for handling access to type information of Kobjects 885
L4::Kobject_x< Derived, ARGS >
Generic Kobject inheritance template 887
L4::Out_of_memory
Exception signalling insufficient memory 890
L4::Platform_control
L4 C++ interface for controlling platform-wide properties 893
L4::Poll_timeout_kipclock
A polling timeout based on the L4Re clock 897
L4::Proto_t< P >
Data type for defining protocol numbers 900
L4::Runtime_error
Exception for an abstract runtime error 900
L4::Scheduler
Scheduler object 903
L4::Semaphore
Kernel-provided semaphore object 906
L4::Server< LOOP_HOOKS >
Basic server loop for handling client requests 910
L4::Server_object
Abstract server object to be used with L4::Server and L4::Basic_registry 914
L4::Server_object_t< IFACE, BASE >
Base class (template) for server implementing server objects 915
L4::Server_object_x< Derived, IFACE, BASE >
Helper class to implement p_dispatch based server objects 918
L4::Smart_cap< T, SMART >
Smart capability class 920
L4::String
A null-terminated string container class 922
L4::Task
C++ interface of the Task kernel object 922
L4::Thread
C++ L4 kernel thread interface 931
L4::Thread::Attr
Thread attributes used for control_commit() 941
L4::Thread::Modify_senders
Wrapper class for modifying senders 945
L4::Triggerable
Interface that allows an object to be triggered by some source 946
L4::Type_info
Dynamic Type Information for L4Re Interfaces 949
L4::Type_info::Demand
Data type for expressing the needed receive buffers at the server-side of an interface 950

L4::Type_info::Demand_t< CAPS, FLAGS, MEM, PORTS >
Template type statically describing demand of receive buffers . . . 953
L4::Type_info::Demand_union_t< D1, D2 >
Template type statically describing the combination of two Demand object . . . 956
L4::Typeid::Detail::_Rpcs< OPCODE, O, X >
Empty list of RPCs . . . 958
L4::Typeid::Detail::_Rpcs< OPCODE, O, Default_op< R > >::Rpc< Y >
Find the given RPC in the list . . . 959
L4::Typeid::Detail::_Rpcs< OPCODE, O, R, X...>
Non-empty list of RPCs . . . 960
L4::Typeid::Detail::_Rpcs< OPCODE, O, R, X...>::Rpc< Y >
Find the given RPC in the list . . . 961
L4::Typeid::Detail::Rpcs_end
Internal end-of-list marker . . . 962
L4::Typeid::P_dispatch< LIST >
Use for protocol based dispatch stage . . . 963
L4::Typeid::Raw_ipc< CLASS >
RPCs list for passing raw incoming IPC to the server object . . . 963
L4::Typeid::Rpc_nocode< OPERATION >
List of RPCs of an interface using a single operation without an opcode . . . 964
L4::Typeid::Rpcs< RPCS >
Standard list of RPCs of an interface . . . 967
L4::Typeid::Rpcs_code< OPCODE_TYPE >
List of RPCs of an interface using a special opcode type . . . 969
L4::Typeid::Rpcs_code< OPCODE_TYPE >::F< RPCS > . . . 969
L4::Typeid::Rpcs_sys< ARG >
List of RPCs typically used for kernel interfaces . . . 972
L4::Types::Bool< V >
Boolean meta type . . . 974
L4::Types::False
False meta value . . . 975
L4::Types::Flags< BITS_ENUM, UNDERLYING >
Template for defining typical Flags bitmaps . . . 976
L4::Types::Same< A, B >
Compare two data types for equality . . . 980
L4::Types::True
True meta value . . . 983
L4::Unknown_error
Exception for an unknown condition . . . 984
L4::Vcon
C++ L4 Vcon interface . . . 987
L4::Vm
Virtual machine . . . 995
l4_buf_regs_t
Encapsulation of the buffer-registers block in the UTCB . . . 997
l4_exc_regs_t
UTCB structure for exceptions . . . 997
l4_fpage_t
L4 flexpage type . . . 1000
l4_icu_info_t
Info structure for an ICU . . . 1001
l4_icu_msi_info_t
Info to use for a specific MSI . . . 1002
l4_kernel_info_mem_desc_t
Memory descriptor data structure . . . 1003
l4_msg_regs_t
Encapsulation of the message-register block in the UTCB . . . 1004

l4_msgtag_t
Message tag data structure . . . 1004
l4_sched_cpu_set_t
CPU sets . . . 1006
l4_sched_param_t
Scheduler parameter set . . . 1008
l4_snd_fpage_t
Send-flex-page types . . . 1009
l4_thread_regs_t
Encapsulation of the thread-control-register block of the UTCB . . . 1010
l4_timeout_s
Basic timeout specification . . . 1012
l4_timeout_t
Timeout pair . . . 1012
l4_tracebuffer_status_t
Trace-buffer status . . . 1013
l4_tracebuffer_status_window_t
Trace-buffer status window descriptor . . . 1015
l4_vcon_attr_t
Vcon attribute structure . . . 1016
l4_vcpu_ipc_regs_t
VCPU message registers . . . 1017
l4_vcpu_regs_t
VCPU registers . . . 1018
l4_vcpu_state_t
State of a vCPU . . . 1021
l4_vhw_descriptor
Virtual hardware devices description . . . 1024
l4_vhw_entry
Description of a device . . . 1026
l4_vm_svm_vmcb_control_area
VMCB structure for SVM VMs . . . 1029
l4_vm_svm_vmcb_state_save_area
State save area structure for SVM VMs . . . 1029
l4_vm_svm_vmcb_state_save_area_seg
State save area segment selector struct . . . 1031
l4_vm_svm_vmcb_t
Control structure for SVM VMs . . . 1031
l4_vm_tz_state
State structure for TrustZone VMs . . . 1033
L4Re::Cap_alloc
Capability allocator interface . . . 1033
L4Re::Console
Console class . . . 1036
L4Re::Dataspace
Interface for memory-like objects . . . 1039
L4Re::Dataspace::Stats
Information about the dataspace . . . 1049
L4Re::Debug_obj
Debug interface . . . 1050
L4Re::Dma_space
DMA Address Space . . . 1053
L4Re::Env
Initial Environment (C++ version) . . . 1057
L4Re::Event
Event class . . . 1066
L4Re::Event_buffer_t< PAYLOAD >
Event buffer class . . . 1069

L4Re::Event_buffer_t< PAYLOAD >::Event
Event structure used in buffer . . . 1072
L4Re::Inhibitor
Set of inhibitor locks, which inhibit specific actions when held . . . 1073
L4Re::Log
Log interface class . . . 1077
L4Re::Mem_alloc
Memory allocation interface . . . 1080
L4Re::Namespace
Name-space interface . . . 1084
L4Re::Parent
Parent interface . . . 1089
L4Re::Rm
Region map . . . 1093
L4Re::Smart_cap_auto< Unmap_flags >
Helper for Auto_cap and Auto_del_cap . . . 1103
L4Re::Util::Auto_cap< T >
Automatic capability that implements automatic free and unmap of the capability selector . . . 1103
L4Re::Util::Auto_del_cap< T >
Automatic capability that implements automatic free and unmap+delete of the capability selector 1105
L4Re::Util::Br_manager
Buffer-register (BR) manager for L4::Server . . . 1106
L4Re::Util::Br_manager_hooks
Predefined server-loop hooks for a server loop using the Br_manager . . . 1111
L4Re::Util::Br_manager_timeout_hooks
Predefined server-loop hooks for a server with using the Br_manager and a timeout queue . . . 1112
L4Re::Util::Cap_alloc_base
Capability allocator . . . 1115
L4Re::Util::Counting_cap_alloc< COUNTERTYPE >
Reference-counting cap allocator . . . 1116
L4Re::Util::Dataspace_svr
Dataspace server class . . . 1120
L4Re::Util::Event_buffer_consumer_t< PAYLOAD >
An event buffer consumer . . . 1125
L4Re::Util::Event_buffer_t< PAYLOAD >
Event_buffer utility class . . . 1128
L4Re::Util::Event_t< PAYLOAD >
Convenience wrapper for getting access to an event object . . . 1132
L4Re::Util::Item_alloc_base
Item allocator . . . 1135
L4Re::Util::Names::Name
Name class . . . 1135
L4Re::Util::Names::Name_space
Abstract server-side implementation of the L4::Namespace interface . . . 1137
L4Re::Util::Object_registry
A registry that manages server objects and their attached IPC gates for a single server loop for a specific thread . . . 1141
L4Re::Util::Ref_cap< T >
Automatic capability that implements automatic free and unmap of the capability selector . . . 1145
L4Re::Util::Ref_del_cap< T >
Automatic capability that implements automatic free and unmap+delete of the capability selector 1146
L4Re::Util::Registry_server< LOOP_HOOKS >
A server loop object which has a Object_registry included . . . 1147
L4Re::Util::Smart_cap_auto< Unmap_flags >
Helper for Auto_cap and Auto_del_cap . . . 1149
L4Re::Util::Smart_count_cap< Unmap_flags >
Helper for Ref_cap and Ref_del_cap . . . 1150

L4Re::Util::Vcon_svr< SVR >
Console server template class . . . 1151
L4Re::Util::Video::Goos_svr
Goos server class . . . 1152
L4Re::Vfs::Be_file
Boiler plate class for implementing an open file for L4Re::Vfs . . . 1155
L4Re::Vfs::Be_file_system
Boilerplate class for implementing a L4Re::Vfs::File_system . . . 1160
L4Re::Vfs::Directory
Interface for a POSIX file that is a directory . . . 1163
L4Re::Vfs::File
The basic interface for an open POSIX file . . . 1169
L4Re::Vfs::File_system
Basic interface for an L4Re::Vfs file system . . . 1170
L4Re::Vfs::Fs
POSIX File-system related functionality . . . 1173
L4Re::Vfs::Generic_file
The common interface for an open POSIX file . . . 1176
L4Re::Vfs::Mman
Interface for the POSIX memory management . . . 1180
L4Re::Vfs::Ops
Interface for the POSIX backends for an application . . . 1182
L4Re::Vfs::Regular_file
Interface for a POSIX file that provides regular file semantics . . . 1183
L4Re::Vfs::Special_file
Interface for a POSIX file that provides special file semantics . . . 1188
L4Re::Video::Color_component
A color component . . . 1190
L4Re::Video::Goos
A goos . . . 1193
L4Re::Video::Goos::Info
Information structure of a goos . . . 1198
L4Re::Video::Pixel_info
Pixel information . . . 1200
L4Re::Video::View
View . . . 1206
L4Re::Video::View::Info
Information structure of a view . . . 1210
l4re_aux_t
Auxiliary descriptor . . . 1212
l4re_ds_stats_t
Information about the data space . . . 1213
l4re_elf_aux_mword_t
Auxiliary vector element for a single unsigned data word . . . 1214
l4re_elf_aux_t
Generic header for each auxiliary vector element . . . 1215
l4re_elf_aux_vma_t
Auxiliary vector element for a reserved virtual memory area . . . 1215
l4re_env_cap_entry_t
Entry in the L4Re environment array for the named inital objects . . . 1216
l4re_env_t
Initial Environment structure (C version) . . . 1218
l4re_event_t
Event structure used in buffer . . . 1219
l4re_video_color_component_t
Color component structure . . . 1220
l4re_video_goos_info_t
Goos information structure . . . 1221

l4re_video_pixel_info_t
Pixel_info structure . . . 1222
l4re_video_view_info_t
View information structure . . . 1223
l4re_video_view_t
C representation of a goos view . . . 1224
l4util_idt_desc_t
IDT entry . . . 1225
l4util_idt_header_t
Header of an IDT table . . . 1225
l4util_mb_addr_range_t
INT-15, AX=E820 style "AddressRangeDescriptor" ...with a "size" parameter on the front which is the structure size - 4, pointing to the next one, up until the full buffer length of the memory map has been reached . . . 1226
l4util_mb_apm_t
APM BIOS info . . . 1227
l4util_mb_drive_t
Drive Info structure . . . 1228
l4util_mb_info_t . . . 1229
l4util_mb_mod_t . . . 1231
l4util_mb_vbe_ctrl_t
VBE controller information . . . 1232
l4util_mb_vbe_mode_t
VBE mode information . . . 1233
L4vbus::Device
Device on a virtual bus (V-BUS) . . . 1236
L4vbus::Gpio_module
A Gpio_module groups multiple GPIO pins together . . . 1243
L4vbus::Gpio_module::Pin_slice
A slice of the pins provided by this module . . . 1249
L4vbus::Gpio_pin
A GPIO pin . . . 1250
L4vbus::Icu
V-BUS Interrupt controller API (ICU) . . . 1257
L4vbus::Pci_dev
A PCI device . . . 1260
L4vbus::Pci_host_bridge
A Pci host bridge . . . 1265
L4vbus::Pm< DEC >
Power-management API mixin . . . 1269
L4vbus::Vbus
The virtual BUS . . . 1271
l4vbus_device_t
Detailed information about a vbus device . . . 1275
l4vbus_resource_t
Description of a single vbus resource . . . 1276
L4vcpu::State
C++ implementation of state word in the vCPU area . . . 1277
L4vcpu::Vcpu
C++ implementation of the vCPU save state area . . . 1278
L4virtio::Driver::Virtqueue
Driver-side implementation of a Virtqueue . . . 1288
L4virtio::Ptr< T >
Pointer used in virtio descriptors . . . 1293
L4virtio::Svr::Bad_descriptor
Exception used by Queue to indicate descriptor errors . . . 1295
L4virtio::Svr::Data_buffer
Abstract data buffer . . . 1298

L4virtio::Svr::Dev_config
Abstraction for L4-Virtio device config memory . . . 1301
L4virtio::Svr::Dev_features
Type for device feature bitmap . . . 1308
L4virtio::Svr::Dev_status
Type of the device status register . . . 1310
L4virtio::Svr::Device_t< DATA >
Server-side L4-VIRTIO device stub . . . 1314
L4virtio::Svr::Driver_mem_list_t< DATA >
List of driver memory regions assigned to a single L4-VIRTIO transport instance . . . 1318
L4virtio::Svr::Driver_mem_region_t< DATA >
Region of driver memory, that shall be managed locally . . . 1326
L4virtio::Svr::Request_processor
Encapsulate the state for processing a VIRTIO request . . . 1331
L4virtio::Svr::Virtqueue
Virtqueue implementation for the device . . . 1336
L4virtio::Svr::Virtqueue::Head_desc
VIRTIO request, essentially a descriptor from the available ring . . . 1340
L4virtio::Virtqueue
Low-level Virtqueue . . . 1342
L4virtio::Virtqueue::Avail
Type of available ring, this is read-only for the host . . . 1351
L4virtio::Virtqueue::Avail::Flags
Flags of the available ring . . . 1352
L4virtio::Virtqueue::Desc
Descriptor in the descriptor table . . . 1354
L4virtio::Virtqueue::Desc::Flags
Type for descriptor flags . . . 1355
L4virtio::Virtqueue::Used
Used ring . . . 1359
L4virtio::Virtqueue::Used::Flags
Flags for the used ring . . . 1360
L4virtio::Virtqueue::Used_elem
Type of an element of the used ring . . . 1362
l4virtio_block_config_t
Device configuration for block devices . . . 1363
l4virtio_block_header_t
Header structure of a request for a block device . . . 1364
l4virtio_config_hdr_t
L4-VIRTIO config header, provided in shared data space . . . 1364
l4virtio_config_queue_t
Queue configuration entry . . . 1366

Chapter 10

Module Documentation

10.1 ARM Virtual Registers (UTCB)

Collaboration diagram for ARM Virtual Registers (UTCB):

Data Structures

- struct l4_exc_regs_t

  UTCB structure for exceptions.

Typedefs

- typedef struct l4_exc_regs_t l4_exc_regs_t

  UTCB structure for exceptions.

Enumerations

- enum L4_utcb_consts_arm

  UTCB constants for ARM.

10.1.1 Detailed Description

10.2 Atomic Instructions

Collaboration diagram for Atomic Instructions:

Files

- file atomic.h

  atomic operations header and generic implementations

Functions

- int l4util_cmpxchg64 (volatile l4_uint64_t *dest, l4_uint64_t cmp_val, l4_uint64_t new_val)

  Atomic compare and exchange (64 bit version)
- int l4util_cmpxchg32 (volatile l4_uint32_t *dest, l4_uint32_t cmp_val, l4_uint32_t new_val)

  Atomic compare and exchange (32 bit version)
- int l4util_cmpxchg16 (volatile l4_uint16_t *dest, l4_uint16_t cmp_val, l4_uint16_t new_val)

  Atomic compare and exchange (16 bit version)
- int l4util_cmpxchg8 (volatile l4_uint8_t *dest, l4_uint8_t cmp_val, l4_uint8_t new_val)

  Atomic compare and exchange (8 bit version)
- int l4util_cmpxchg (volatile l4_umword_t *dest, l4_umword_t cmp_val, l4_umword_t new_val)

  Atomic compare and exchange (machine wide fields)
- l4_uint32_t l4util_xchg32 (volatile l4_uint32_t *dest, l4_uint32_t val)

  Atomic exchange (32 bit version)
- l4_uint16_t l4util_xchg16 (volatile l4_uint16_t *dest, l4_uint16_t val)

  Atomic exchange (16 bit version)
- l4_uint8_t l4util_xchg8 (volatile l4_uint8_t *dest, l4_uint8_t val)

  Atomic exchange (8 bit version)
- l4_umword_t l4util_xchg (volatile l4_umword_t *dest, l4_umword_t val)

  Atomic exchange (machine wide fields)
- void l4util_atomic_add (volatile long *dest, long val)

  Atomic add.
- void l4util_atomic_inc (volatile long *dest)

  Atomic increment.

Atomic add/sub/and/or (8,16,32 bit version) without result

- void l4util_add8 (volatile l4_uint8_t *dest, l4_uint8_t val)
- void **l4util_add16** (volatile l4_uint16_t *dest, l4_uint16_t val)
- void **l4util_add32** (volatile l4_uint32_t *dest, l4_uint32_t val)
- void **l4util_sub8** (volatile l4_uint8_t *dest, l4_uint8_t val)
- void **l4util_sub16** (volatile l4_uint16_t *dest, l4_uint16_t val)

- void **l4util_sub32** (volatile l4_uint32_t *dest, l4_uint32_t val)
- void **l4util_and8** (volatile l4_uint8_t *dest, l4_uint8_t val)
- void **l4util_and16** (volatile l4_uint16_t *dest, l4_uint16_t val)
- void **l4util_and32** (volatile l4_uint32_t *dest, l4_uint32_t val)
- void **l4util_or8** (volatile l4_uint8_t *dest, l4_uint8_t val)
- void **l4util_or16** (volatile l4_uint16_t *dest, l4_uint16_t val)
- void **l4util_or32** (volatile l4_uint32_t *dest, l4_uint32_t val)

Atomic add/sub/and/or operations (8,16,32 bit) with result

- l4_uint8_t l4util_add8_res (volatile l4_uint8_t *dest, l4_uint8_t val)
- l4_uint16_t **l4util_add16_res** (volatile l4_uint16_t *dest, l4_uint16_t val)
- l4_uint32_t **l4util_add32_res** (volatile l4_uint32_t *dest, l4_uint32_t val)
- l4_uint8_t **l4util_sub8_res** (volatile l4_uint8_t *dest, l4_uint8_t val)
- l4_uint16_t **l4util_sub16_res** (volatile l4_uint16_t *dest, l4_uint16_t val)
- l4_uint32_t **l4util_sub32_res** (volatile l4_uint32_t *dest, l4_uint32_t val)
- l4_uint8_t **l4util_and8_res** (volatile l4_uint8_t *dest, l4_uint8_t val)
- l4_uint16_t **l4util_and16_res** (volatile l4_uint16_t *dest, l4_uint16_t val)
- l4_uint32_t **l4util_and32_res** (volatile l4_uint32_t *dest, l4_uint32_t val)
- l4_uint8_t **l4util_or8_res** (volatile l4_uint8_t *dest, l4_uint8_t val)
- l4_uint16_t **l4util_or16_res** (volatile l4_uint16_t *dest, l4_uint16_t val)
- l4_uint32_t **l4util_or32_res** (volatile l4_uint32_t *dest, l4_uint32_t val)

Atomic inc/dec (8,16,32 bit) without result

- void l4util_inc8 (volatile l4_uint8_t *dest)
- void **l4util_inc16** (volatile l4_uint16_t *dest)
- void **l4util_inc32** (volatile l4_uint32_t *dest)
- void **l4util_dec8** (volatile l4_uint8_t *dest)
- void **l4util_dec16** (volatile l4_uint16_t *dest)
- void **l4util_dec32** (volatile l4_uint32_t *dest)

Atomic inc/dec (8,16,32 bit) with result

- l4_uint8_t l4util_inc8_res (volatile l4_uint8_t *dest)
- l4_uint16_t **l4util_inc16_res** (volatile l4_uint16_t *dest)
- l4_uint32_t **l4util_inc32_res** (volatile l4_uint32_t *dest)
- l4_uint8_t **l4util_dec8_res** (volatile l4_uint8_t *dest)
- l4_uint16_t **l4util_dec16_res** (volatile l4_uint16_t *dest)
- l4_uint32_t **l4util_dec32_res** (volatile l4_uint32_t *dest)

10.2.1 Detailed Description

10.2.2 Function Documentation

10.2.2.1 void l4util_add8 (volatile **l4_uint8_t** * *dest*, **l4_uint8_t** *val*) [inline]

Parameters

dest	destination operand
val	value to add/sub/and/or

Definition at line 413 of file atomic.h.

10.2.2.2 l4_uint8_t l4util_add8_res (volatile l4_uint8_t * *dest*, l4_uint8_t *val*) [inline]

Parameters

dest	destination operand
val	value to add/sub/and/or

Returns

res

Definition at line 486 of file atomic.h.

10.2.2.3 void l4util_atomic_add (volatile long * *dest*, long *val*) [inline]

Atomic add.

Parameters

dest	destination operand
val	value to add

Definition at line 54 of file atomic_arch.h.

10.2.2.4 void l4util_atomic_inc (volatile long * *dest*) [inline]

Atomic increment.

Parameters

dest	destination operand

Definition at line 61 of file atomic_arch.h.

10.2.2.5 int l4util_cmpxchg (volatile l4_umword_t * *dest*, l4_umword_t *cmp_val*, l4_umword_t *new_val*) [inline]

Atomic compare and exchange (machine wide fields)

Parameters

dest	destination operand
cmp_val	compare value
new_val	new value for dest

Returns

0 if comparison failed, 1 otherwise

Compare the value in *dest* with *cmp_val*, if equal set *dest* to *new_val*

Definition at line 382 of file atomic.h.

Referenced by l4util_btr(), l4util_bts(), l4util_clear_bit(), and l4util_set_bit().

Here is the caller graph for this function:

10.2.2.6 int l4util_cmpxchg16 (volatile l4_uint16_t * *dest*, l4_uint16_t *cmp_val*, l4_uint16_t *new_val*) [inline]

Atomic compare and exchange (16 bit version)

Parameters

dest	destination operand
cmp_val	compare value
new_val	new value for dest

Returns

0 if comparison failed, !=0 otherwise

Compare the value in *dest* with *cmp_val*, if equal set *dest* to *new_val*

Definition at line 375 of file atomic.h.

10.2.2.7 int l4util_cmpxchg32 (volatile l4_uint32_t * *dest*, l4_uint32_t *cmp_val*, l4_uint32_t *new_val*) [inline]

Atomic compare and exchange (32 bit version)

Parameters

dest	destination operand
cmp_val	compare value
new_val	new value for dest

Returns

0 if comparison failed, !=0 otherwise

Compare the value in *dest* with *cmp_val*, if equal set *dest* to *new_val*

Definition at line 24 of file atomic_arch.h.

10.2.2.8 int l4util_cmpxchg64 (volatile l4_uint64_t * *dest*, l4_uint64_t *cmp_val*, l4_uint64_t *new_val*) [inline]

Atomic compare and exchange (64 bit version)

Parameters

dest	destination operand
cmp_val	compare value
new_val	new value for dest

Returns

0 if comparison failed, 1 otherwise

Compare the value in *dest* with *cmp_val*, if equal set *dest* to *new_val*

Definition at line 361 of file atomic.h.

10.2.2.9 int l4util_cmpxchg8 (volatile l4_uint8_t * *dest*, l4_uint8_t *cmp_val*, l4_uint8_t *new_val*) [inline]

Atomic compare and exchange (8 bit version)

Parameters

dest	destination operand
cmp_val	compare value
new_val	new value for dest

Returns

0 if comparison failed, !=0 otherwise

Compare the value in *dest* with *cmp_val*, if equal set *dest* to *new_val*

Definition at line 368 of file atomic.h.

10.2.2.10 void l4util_inc8 (volatile l4_uint8_t * *dest*) [inline]

Parameters

dest	destination operand

Definition at line 311 of file atomic.h.

10.2.2.11 l4_uint8_t l4util_inc8_res (volatile l4_uint8_t * *dest*) [inline]

Parameters

dest	destination operand

Returns

res

Definition at line 337 of file atomic.h.

10.2.2.12 l4_umword_t l4util_xchg (volatile l4_umword_t * *dest*, l4_umword_t *val*) [inline]

Atomic exchange (machine wide fields)

Parameters

dest	destination operand
val	new value for dest

Returns

old value at destination

Definition at line 407 of file atomic.h.

10.2.2.13 l4_uint16_t l4util_xchg16 (volatile l4_uint16_t * *dest*, l4_uint16_t *val*) [inline]

Atomic exchange (16 bit version)

Parameters

dest	destination operand
val	new value for dest

Returns

old value at destination

Definition at line 395 of file atomic.h.

10.2.2.14 l4_uint32_t l4util_xchg32 (volatile l4_uint32_t * *dest*, l4_uint32_t *val*) [inline]

Atomic exchange (32 bit version)

Parameters

dest	destination operand
val	new value for dest

Returns

old value at destination

Definition at line 389 of file atomic.h.

10.2.2.15 l4_uint8_t l4util_xchg8 (volatile l4_uint8_t * *dest*, l4_uint8_t *val*) [inline]

Atomic exchange (8 bit version)

Parameters

dest	destination operand
val	new value for dest

Returns

old value at destination

Definition at line 401 of file atomic.h.

10.3 Auxiliary data

Collaboration diagram for Auxiliary data:

Data Structures

- struct l4re_aux_t

  Auxiliary descriptor.

Typedefs

- typedef struct l4re_aux_t l4re_aux_t

  Auxiliary descriptor.

Enumerations

- enum l4re_aux_ldr_flags_t

  Flags for program loading.

10.3.1 Detailed Description

10.4 Base API

Interfaces for all kinds of base functionality.

Collaboration diagram for Base API:

Modules

- Basic Macros

  L4 standard macros for header files, function definitions, and public APIs etc.

- C++ IPC Interface Definition.

  APIs for defining IPC interfaces using C++ as language.
- Cache Consistency

  Various functions for cache consistency.
- Capabilities

  C interface for capabilities.
- Error codes

  Common error codes.
- Fiasco extensions

  Kernel debugger extensions of the Fiasco L4 implementation.
- Flex pages

  Flex-page related API.
- Integer Types

  `#include<l4/sys/l4int.h>`
- Kernel Interface Page

  Kernel Interface Page.
- Kernel Objects

  API of kernel objects.
- Memory operations.

  Operations for memory access.
- Memory related

  Memory related constants, data types and functions.
- Object Invocation

  API for L4 object invocation.

Files

- file cache.h

  Cache-consistency functions.
- file compiler.h

  L4 compiler related defines.
- file consts.h

  Common constants.
- file debugger.h

  Debugger related definitions.
- file factory.h

  Common factory related definitions.
- file icu

  Interrupt controller.
- file icu.h

  Interrupt controller.
- file ipc.h

  Common IPC interface.
- file irq.h

  C Irq interface.
- file kip

  L4::Kip class, memory descriptors.
- file kip.h

  Kernel Info Page access functions.
- file memdesc.h

Memory description functions.

- file meta

  Meta interface for getting dynamic type information about objects behind capabilities.
- file types.h

  Common L4 ABI Data Types.
- file vhw.h

  Descriptors for virtual hardware (under UX).
- file consts.h

  Common L4 constants, arm version.
- file consts.h

  Common L4 constants, amd64 version.
- file ipc.h

  L4 IPC System Calls, x86.
- file consts.h

  Common L4 constants, x86 version.

10.4.1 Detailed Description

Interfaces for all kinds of base functionality.

Some notes on Inter Process Communication (IPC)

IPC in L4 is always synchronous and unbuffered: a message is transferred from the sender to the recipient if and only if the recipient has invoked a corresponding IPC operation. The sender blocks until this happens or a timeout specified by the sender elapsed without the destination becoming ready to receive.

10.5 Basic Macros

L4 standard macros for header files, function definitions, and public APIs etc.

Collaboration diagram for Basic Macros:

Macros

- #define L4_ALWAYS_INLINE

  L4 Inline function attribute.
- #define __END_DECLS

  End section with C types and functions.
- #define EXTERN_C_BEGIN

  Start section with C types and functions.
- #define EXTERN_C_END

  End section with C types and functions.
- #define EXTERN_C

  Mark C types and functions.
- #define L4_NOTHROW

  Mark a function declaration and definition as never throwing an exception.
- #define L4_HIDDEN

  Attribute to mark functions, variables, and data types as being explicitly hidden from users of a library.
- #define L4_NORETURN

  Noreturn function attribute.
- #define L4_NOINSTRUMENT

  No instrumentation function attribute.
- #define L4_LIKELY(x)

  Expression is likely to execute.
- #define L4_UNLIKELY(x)

  Expression is unlikely to execute.
- #define L4_STICKY(x)

  Mark symbol sticky (even not there)
- #define L4_DEPRECATED(s)

  Mark symbol deprecated.
- #define L4_stringify_helper(x)

  stringify helper.
- #define L4_stringify(x)

  stringify.
- #define L4_CV

  Define calling convention.
- #define L4_CV

  Define calling convention.
- #define L4_CV __attribute__((regparm(0)))

  Define calling convention.

Functions

- void l4_barrier (void)

  Memory barrier.
- void l4_mb (void)

  Memory barrier.
- void l4_wmb (void)

  Write memory barrier.

10.5.1 Detailed Description

L4 standard macros for header files, function definitions, and public APIs etc.

```
#include <l4/sys/compiler.h>
```

10.5.2 Macro Definition Documentation

10.5.2.1 #define L4_ALWAYS_INLINE

L4 Inline function attribute.

Always inline a function

Definition at line 67 of file compiler.h.

10.5.2.2 #define L4_HIDDEN

Attribute to mark functions, variables, and data types as being explicitly hidden from users of a library.

This attribute is intended for functions, data, and data types that shall never be visible outside of a library. In particular, for shared libraries this may result in much faster code within the library and short linking times.

```
class L4_HIDDEN My_class
{
  ...
};

int L4_HIDDEN function(void);

int L4_HIDDEN global_data; // global data is not recommended
```

Definition at line 212 of file compiler.h.

10.5.2.3 #define L4_NOTHROW

Mark a function declaration and definition as never throwing an exception.

(Also for C code).

This macro shall be used to mark C and C++ functions that never throw any exception. Note that also C functions may throw exceptions according to the compilers ABI and shall be marke with L4_NOTHROW if they never do. In C++ this is equvalent to `throw()`.

```
int foo() L4_NOTHROW;
...
int foo() L4_NOTHROW
{
  ...
  return result;
}
```

Definition at line 185 of file compiler.h.

10.6 Bit Manipulation

Collaboration diagram for Bit Manipulation:

Files

- file bitops.h

  bit manipulation functions

Functions

- void l4util_set_bit (int b, volatile l4_umword_t *dest)

  Set bit in memory.
- void l4util_clear_bit (int b, volatile l4_umword_t *dest)

  Clear bit in memory.
- void l4util_complement_bit (int b, volatile l4_umword_t *dest)

  Complement bit in memory.
- int l4util_test_bit (int b, const volatile l4_umword_t *dest)

  Test bit (return value of bit)
- int l4util_bts (int b, volatile l4_umword_t *dest)

  Bit test and set.
- int l4util_btr (int b, volatile l4_umword_t *dest)

  Bit test and reset.
- int l4util_btc (int b, volatile l4_umword_t *dest)

  Bit test and complement.
- int l4util_bsr (l4_umword_t word)

  Bit scan reverse.
- int l4util_bsf (l4_umword_t word)

  Bit scan forward.
- int l4util_find_first_set_bit (const void *dest, l4_size_t size)

  Find the first set bit in a memory region.
- int l4util_find_first_zero_bit (const void *dest, l4_size_t size)

  Find the first zero bit in a memory region.
- int l4util_next_power2 (const unsigned long val)

  Find the next power of 2 for a given number.

10.6.1 Detailed Description

10.6.2 Function Documentation

10.6.2.1 int l4util_bsf (l4_umword_t *word*) [inline]

Bit scan forward.

Parameters

word	value (machine size)

Returns

index of least significant bit set in word, -1 if no bit is set (word == 0)

"bit scan forward", find least significant bit set in word.

Definition at line 351 of file bitops.h.

Referenced by l4util_splitlog2_size().

Here is the caller graph for this function:

10.6.2.2 int l4util_bsr (l4_umword_t *word*) [inline]

Bit scan reverse.

Parameters

word	value (machine size)

Returns

index of most significant set bit in word, -1 if no bit is set (word == 0)

"bit scan reverse", find most significant set bit in word (-> LOG2(word))

Definition at line 334 of file bitops.h.

Referenced by l4util_splitlog2_size().

Here is the caller graph for this function:

10.6.2.3 int l4util_btc (int *b*, volatile l4_umword_t * *dest*) [inline]

Bit test and complement.

Parameters

b	bit position
dest	destination operand

Returns

Old value of bit *b*.

Complement bit *b* and return old value.

Definition at line 435 of file bitops.h.

10.6.2.4 int l4util_btr (int *b*, volatile l4_umword_t * *dest*) [inline]

Bit test and reset.

Parameters

b	bit position
dest	destination operand

Returns

Old value of bit *b*.

Reset bit *b* and return old value.

Definition at line 313 of file bitops.h.

References l4util_cmpxchg().

Here is the call graph for this function:

10.6.2.5 int l4util_bts (int *b*, volatile l4_umword_t * *dest*) [inline]

Bit test and set.

Parameters

b	bit position
dest	destination operand

Returns

Old value of bit *b*.

Set the *b* bit of *dest* to 1 and return the old value.

Definition at line 291 of file bitops.h.

References l4util_cmpxchg().

Here is the call graph for this function:

10.6.2.6 void l4util_clear_bit (int *b*, volatile l4_umword_t * *dest*) [inline]

Clear bit in memory.

Parameters

b	bit position
dest	destination operand

Definition at line 250 of file bitops.h.

References l4util_cmpxchg().

Here is the call graph for this function:

10.6.2.7 void l4util_complement_bit (int *b*, volatile l4_umword_t * *dest*) [inline]

Complement bit in memory.

Parameters

b	bit position
dest	destination operand

Definition at line 394 of file bitops.h.

10.6.2.8 int l4util_find_first_set_bit (const void * *dest*, l4_size_t *size*) [inline]

Find the first set bit in a memory region.

Parameters

dest	bit string
size	size of string in bits (must be a multiple of 32!)

Returns

number of the first set bit, >= size if no bit is set

Definition at line 441 of file bitops.h.

10.6.2.9 int l4util_find_first_zero_bit (const void * *dest*, l4_size_t *size*) [inline]

Find the first zero bit in a memory region.

Parameters

dest	bit string
size	size of string in bits (must be a multiple of 32!)

Returns

number of the first zero bit, >= size if no bit is set

Definition at line 368 of file bitops.h.

10.6.2.10 int l4util_next_power2 (const unsigned long *val*) [inline]

Find the next power of 2 for a given number.

Parameters

val	initial value

Returns

next-highest power of 2

Definition at line 408 of file bitops.h.

10.6.2.11 void l4util_set_bit (int *b*, volatile l4_umword_t * *dest*) [inline]

Set bit in memory.

Parameters

b	bit position
dest	destination operand

Definition at line 231 of file bitops.h.

References l4util_cmpxchg().

Here is the call graph for this function:

10.6.2.12 int l4util_test_bit (int *b*, const volatile l4_umword_t * *dest*) [inline]

Test bit (return value of bit)

Parameters

b	bit position
dest	destination operand

Returns

Value of bit *b*.

Definition at line 268 of file bitops.h.

10.7 Bitmap graphics and fonts

This library provides some functions for bitmap handling in frame buffers.

Collaboration diagram for Bitmap graphics and fonts:

Modules

- Functions for rendering bitmap data in frame buffers
- Functions for rendering bitmap fonts to frame buffers

10.7.1 Detailed Description

This library provides some functions for bitmap handling in frame buffers.

Includes simple functions like filling or copying an area of the frame buffer going up to rendering text into the frame buffer using bitmap fonts.

10.8 Buffer Registers (BRs)

Collaboration diagram for Buffer Registers (BRs):

Data Structures

- struct l4_buf_regs_t

  Encapsulation of the buffer-registers block in the UTCB.

Typedefs

- typedef struct l4_buf_regs_t l4_buf_regs_t

  Encapsulation of the buffer-registers block in the UTCB.

Enumerations

- enum l4_buffer_desc_consts_t { L4_BDR_MEM_SHIFT = 0, L4_BDR_IO_SHIFT = 5, L4_BDR_OBJ_SHIFT = 10 }

  Constants for buffer descriptors.

Functions

- void l4_utcb_inherit_fpu (int switch_on) L4_NOTHROW

  Enable or disable inheritance of FPU state to receiver.

10.8.1 Detailed Description

10.8.2 Enumeration Type Documentation

10.8.2.1 enum l4_buffer_desc_consts_t

Constants for buffer descriptors.

Enumerator

L4_BDR_MEM_SHIFT Bit offset for the memory-buffer index.

L4_BDR_IO_SHIFT Bit offset for the IO-buffer index.

L4_BDR_OBJ_SHIFT Bit offset for the capability-buffer index.

Definition at line 219 of file consts.h.

10.9 C++ Exceptions

Collaboration diagram for C++ Exceptions:

Files

- file exceptions

  Base exceptions.
- file std_exc_io

  Base exceptions std stream operator.

Data Structures

- class L4::Exception_tracer

  Back-trace support for exceptions.
- class L4::Base_exception

  Base class for all exceptions, thrown by the L4Re framework.
- class L4::Runtime_error

  Exception for an abstract runtime error.
- class L4::Out_of_memory

  Exception signalling insufficient memory.
- class L4::Element_already_exists

  Exception for duplicate element insertions.
- class L4::Unknown_error

  Exception for an unknown condition.
- class L4::Element_not_found

  Exception for a failed lookup (element not found).
- class L4::Invalid_capability

  Indicates that an invalid object was invoked.
- class L4::Com_error

  Error conditions during IPC.
- class L4::Bounds_error

  Access out of bounds.

Macros

- #define L4_CXX_EXCEPTION_BACKTRACE 20

  Number of instruction pointers in backtrace.

10.9.1 Detailed Description

10.10 C++ IPC Interface Definition.

APIs for defining IPC interfaces using C++ as language.

Collaboration diagram for C++ IPC Interface Definition.:

Modules

- Internal Helpers

Namespaces

- L4::Typeid

  Definition of interface data-type helpers.

Data Structures

- struct L4::Typeid::Detail::Rpcs_end

  Internal end-of-list marker.
- struct L4::Typeid::Detail::_Rpcs< OPCODE, O, R, X...>

  Non-empty list of RPCs.
- struct L4::Typeid::Raw_ipc< CLASS >

  RPCs list for passing raw incoming IPC to the server object.
- struct L4::Typeid::Rpcs_code< OPCODE_TYPE >

  List of RPCs of an interface using a special opcode type.

Enumerations

- enum

  The opcode value to use for this RPC, may be bogus if the opcode_type is void.
- enum

  The opcode value to use for this RPC, may be bogus if the opcode_type is void.

10.10.1 Detailed Description

APIs for defining IPC interfaces using C++ as language.

10.11 CPU related functions

Collaboration diagram for CPU related functions:

Functions

- int l4util_cpu_has_cpuid (void)

  Check whether the CPU supports the "cpuid" instruction.
- unsigned int l4util_cpu_capabilities (void)

  Returns the CPU capabilities if the "cpuid" instruction is available.
- unsigned int l4util_cpu_capabilities_nocheck (void)

  Returns the CPU capabilities.
- void l4util_cpu_cpuid (unsigned long mode, unsigned long ∗eax, unsigned long ∗ebx, unsigned long ∗ecx, unsigned long ∗edx)

  Generic CPUID access function.

10.11.1 Detailed Description

10.11.2 Function Documentation

10.11.2.1 unsigned int l4util_cpu_capabilities (void) [inline]

Returns the CPU capabilities if the "cpuid" instruction is available.

Returns

CPU capabilities if the "cpuid" instruction is available, 0 if the "cpuid" instruction is not supported.

Definition at line 97 of file cpu.h.

References l4util_cpu_capabilities_nocheck(), and l4util_cpu_has_cpuid().

Here is the call graph for this function:

10.11.2.2 unsigned int l4util_cpu_capabilities_nocheck (void) [inline]

Returns the CPU capabilities.

Returns

CPU capabilities.

Definition at line 86 of file cpu.h.

References l4util_cpu_cpuid().

Referenced by l4util_cpu_capabilities().

Here is the call graph for this function:

Here is the caller graph for this function:

10.11.2.3 int l4util_cpu_has_cpuid (void) [inline]

Check whether the CPU supports the "cpuid" instruction.

Returns

1 if it has, 0 if it has not

Definition at line 66 of file cpu.h.

Referenced by l4util_cpu_capabilities().

Here is the caller graph for this function:

10.12 Cache Consistency

Various functions for cache consistency.

Collaboration diagram for Cache Consistency:

Functions

- void l4_cache_clean_data (unsigned long start, unsigned long end) L4_NOTHROW

  Cache clean a range in D-cache.
- void l4_cache_flush_data (unsigned long start, unsigned long end) L4_NOTHROW

  Cache flush a range.
- void l4_cache_inv_data (unsigned long start, unsigned long end) L4_NOTHROW

  Cache invalidate a range.
- void l4_cache_coherent (unsigned long start, unsigned long end) L4_NOTHROW

  Make memory coherent between I-cache and D-cache.
- void l4_cache_dma_coherent (unsigned long start, unsigned long end) L4_NOTHROW

  Make memory coherent for use with external memory.
- void l4_cache_dma_coherent_full (void) L4_NOTHROW

  Make memory coherent for use with external memory.

10.12.1 Detailed Description

Various functions for cache consistency.

```
#include <l4/sys/cache.h>
```

10.12.2 Function Documentation

10.12.2.1 void l4_cache_clean_data (unsigned long *start*, unsigned long *end*) [inline]

Cache clean a range in D-cache.

Parameters

start	Start of range (inclusive)
end	End of range (exclusive)

Examples:

examples/libs/l4re/c++/shared_ds/ds_clnt.cc.

Definition at line 84 of file cache.h.

10.12.2.2 void l4_cache_coherent (unsigned long *start*, unsigned long *end*) [inline]

Make memory coherent between I-cache and D-cache.

Parameters

start	Start of range (inclusive)
end	End of range (exclusive)

Definition at line 105 of file cache.h.

10.12.2.3 void l4_cache_dma_coherent (unsigned long *start*, unsigned long *end*) [inline]

Make memory coherent for use with external memory.

Parameters

start	Start of range (inclusive)
end	End of range (exclusive)

Definition at line 112 of file cache.h.

10.12.2.4 void l4_cache_flush_data (unsigned long *start*, unsigned long *end*) [inline]

Cache flush a range.

Parameters

start	Start of range (inclusive)
end	End of range (exclusive)

Definition at line 91 of file cache.h.

10.12.2.5 void l4_cache_inv_data (unsigned long *start*, unsigned long *end*) [inline]

Cache invalidate a range.

Parameters

start	Start of range (inclusive)
end	End of range (exclusive)

Definition at line 98 of file cache.h.

10.13 Capabilities

C interface for capabilities.

Collaboration diagram for Capabilities:

Typedefs

- typedef unsigned long l4_cap_idx_t

  L4 Capability selector Type.

Enumerations

- enum l4_cap_consts_t { L4_CAP_SHIFT, L4_CAP_SIZE , L4_CAP_MASK, L4_INVALID_CAP }

  Constants related to capability selectors.
- enum l4_default_caps_t { L4_BASE_TASK_CAP, L4_BASE_FACTORY_CAP, L4_BASE_THREAD_CAP, L4_BASE_PAGER_CAP, L4_BASE_LOG_CAP, L4_BASE_ICU_CAP, L4_BASE_SCHEDULER_CAP, L4_BASE_IOMMU_CAP, L4_BASE_DEBUGGER_CAP }

  Default capabilities setup for the initial tasks.

Functions

- unsigned l4_is_invalid_cap (l4_cap_idx_t c) L4_NOTHROW

  Test if a capability selector is the invalid capability.
- unsigned l4_is_valid_cap (l4_cap_idx_t c) L4_NOTHROW

  Test if a capability selector is a valid selector.
- unsigned l4_capability_equal (l4_cap_idx_t c1, l4_cap_idx_t c2) L4_NOTHROW

  Test if two capability selectors are equal.

10.13.1 Detailed Description

C interface for capabilities.

Add

```
#include <l4/sys/types.h>
#include <l4/sys/consts.h>
```

to your code to use the functions and definitions explained here.

10.13.2 Enumeration Type Documentation

10.13.2.1 enum l4_cap_consts_t

Constants related to capability selectors.

Enumerator

L4_CAP_SHIFT Capability index shift.

L4_CAP_SIZE Offset of two consecutive capability selectors.

L4_CAP_MASK Mask to get only the relevant bits of an l4_cap_idx_t.

L4_INVALID_CAP Invalid capability selector.

Definition at line 128 of file consts.h.

10.13.2.2 enum l4_default_caps_t

Default capabilities setup for the initial tasks.

These capability selectors are setup per default by the micro kernel for the two initial tasks, the Root-Pager (Sigma0) and the Root-Task (Moe).

Attention

This constants do not have any particular meaning for applications started by Moe, see Initial Environment for this kind of information.

See also

Initial Environment for information useful for normal user applications.

Enumerator

L4_BASE_TASK_CAP Capability selector for the current task.

L4_BASE_FACTORY_CAP Capability selector for the factory.

L4_BASE_THREAD_CAP Capability selector for the first thread.

L4_BASE_PAGER_CAP Capability selector for the pager gate.

L4_BASE_LOG_CAP Capability selector for the log object.

L4_BASE_ICU_CAP Capability selector for the base icu object.

L4_BASE_SCHEDULER_CAP Capability selector for the scheduler cap.

L4_BASE_IOMMU_CAP Capability selector for the IO-MMU cap.

L4_BASE_DEBUGGER_CAP Capability selector for the debugger cap.

Definition at line 240 of file consts.h.

10.13.3 Function Documentation

10.13.3.1 unsigned l4_capability_equal (l4_cap_idx_t *c1*, l4_cap_idx_t *c2*) [inline]

Test if two capability selectors are equal.

Parameters

c1	Capability
c2	Capability

Return values

0	The given capabilities are not in the same slot.
1	The given capabilities are in the same slot.

Definition at line 368 of file types.h.

References L4_CAP_SHIFT.

10.13.3.2 unsigned l4_is_invalid_cap (l4_cap_idx_t *c*) [inline]

Test if a capability selector is the invalid capability.

Parameters

c	Capability selector

Return values

0	The capability selector is not the invalid capability.
>0	The capability selector is the invalid capability.

Examples:

examples/libs/l4re/c/ma+rm.c, examples/sys/aliens/main.c, examples/sys/isr/main.c, examples/sys/singlestep/main.c, examples/sys/start-with-exc/main.c, and examples/sys/utcb-ipc/main.c.

Definition at line 360 of file types.h.

Referenced by l4_irq_attach_u().

Here is the caller graph for this function:

10.13.3.3 unsigned l4_is_valid_cap (l4_cap_idx_t *c*) [inline]

Test if a capability selector is a valid selector.

Parameters

c	Capability selector

Return values

0	The capability selector is not valid.
>0	The capability selector is valid.

Definition at line 364 of file types.h.

10.14 Capability allocator

Capability allocator C interface.

Collaboration diagram for Capability allocator:

Functions

- l4_cap_idx_t l4re_util_cap_alloc (void) L4_NOTHROW

  Get free capability index at capability allocator.
- void l4re_util_cap_free (l4_cap_idx_t cap) L4_NOTHROW

  Return capability index to capability allocator.
- void l4re_util_cap_free_um (l4_cap_idx_t cap) L4_NOTHROW

  Return capability index to capability allocator, and unmaps the object.
- long l4re_util_cap_last (void) L4_NOTHROW

  Return last capability index the allocator can return.

10.14.1 Detailed Description

Capability allocator C interface.

10.14.2 Function Documentation

10.14.2.1 long l4re_util_cap_last (void)

Return last capability index the allocator can return.

Returns

last/biggest capability index the allocator can return

10.15 Chunks

Collaboration diagram for Chunks:

Modules

- Consumer
- Producer

Functions

- long l4shmc_add_chunk (l4shmc_area_t ∗shmarea, const char ∗chunk_name, l4_umword_t chunk_capacity, l4shmc_chunk_t ∗chunk)

  Add a chunk in the shared memory area.
- long l4shmc_get_chunk (l4shmc_area_t ∗shmarea, const char ∗chunk_name, l4shmc_chunk_t ∗chunk)

  Get chunk out of shared memory area.
- long l4shmc_get_chunk_to (l4shmc_area_t ∗shmarea, const char ∗chunk_name, l4_umword_t timeout_ms, l4shmc_chunk_t ∗chunk)

  Get chunk out of shared memory area, with timeout.
- long l4shmc_iterate_chunk (l4shmc_area_t ∗shmarea, const char ∗∗chunk_name, long offs)

  Iterate over names of all existing chunks.
- void ∗ l4shmc_chunk_ptr (l4shmc_chunk_t ∗chunk)

  Get data pointer to chunk.
- long l4shmc_chunk_capacity (l4shmc_chunk_t ∗chunk)

  Get capacity of a chunk.
- l4shmc_signal_t ∗ l4shmc_chunk_signal (l4shmc_chunk_t ∗chunk)

  Get the signal of a chunk.

10.15.1 Detailed Description

10.15.2 Function Documentation

10.15.2.1 long l4shmc_add_chunk (l4shmc_area_t ∗ *shmarea,* const char ∗ *chunk_name,* l4_umword_t *chunk_capacity,* l4shmc_chunk_t ∗ *chunk*)

Add a chunk in the shared memory area.

Parameters

	shmarea	The shared memory area to put the chunk in.
	chunk_name	Name of the chunk.
	chunk_capacity	Capacity for payload of the chunk in bytes.
out	*chunk*	Chunk structure to fill in.

Return values

0	Success.
<0	Error.

Examples:

examples/libs/shmc/prodcons.c.

10.15.2.2 long l4shmc_chunk_capacity (l4shmc_chunk_t * *chunk*) [inline]

Get capacity of a chunk.

Parameters

chunk	Chunk.

Return values

0	Success.
<0	Error.

10.15.2.3 void* l4shmc_chunk_ptr (l4shmc_chunk_t * *chunk*) [inline]

Get data pointer to chunk.

Parameters

chunk	Chunk.

Return values

0	Success.
<0	Error.

Examples:

examples/libs/shmc/prodcons.c.

10.15.2.4 l4shmc_signal_t* l4shmc_chunk_signal (l4shmc_chunk_t * *chunk*) [inline]

Get the signal of a chunk.

Parameters

chunk	Chunk.

Return values

0	No signal has been registered with this chunk.
>0	Pointer to signal otherwise.

10.15.2.5 long l4shmc_get_chunk (l4shmc_area_t * *shmarea*, const char * *chunk_name*, l4shmc_chunk_t * *chunk*) [inline]

Get chunk out of shared memory area.

Parameters

	shmarea	Shared memory area.
	chunk_name	Name of the chunk.
out	*chunk*	Chunk data structure to fill.

Return values

0	Success.
<0	Error.

Examples:

examples/libs/shmc/prodcons.c.

10.15.2.6 long l4shmc_get_chunk_to (l4shmc_area_t * *shmarea*, const char * *chunk_name*, l4_umword_t *timeout_ms*, l4shmc_chunk_t * *chunk*)

Get chunk out of shared memory area, with timeout.

Parameters

	shmarea	Shared memory area.
	chunk_name	Name of the chunk.
	timeout_ms	Timeout in milliseconds to wait for the chunk to appear in the shared memory area.
out	*chunk*	Chunk data structure to fill.

Return values

0	Success.
<0	Error.

10.15.2.7 long l4shmc_iterate_chunk (l4shmc_area_t * *shmarea*, const char ** *chunk_name*, long *offs*)

Iterate over names of all existing chunks.

Parameters

shmarea	Shared memory area.
chunk_name	Where the name of the current chunk will be stored
offs	0 to start iteration, return value of previous call to l4shmc_iterate_chunk() to get next chunk

Return values

0	No more chunks available.
<0	Error.
>0	Iterator value for the next call.

10.16 Client/Server IPC Framework

Collaboration diagram for Client/Server IPC Framework:

Modules

- Server-Side IPC framework

  Server-Side framework for implementing object-oriented servers.

10.16.1 Detailed Description

10.17 Comfortable Command Line Parsing

Collaboration diagram for Comfortable Command Line Parsing:

Typedefs

- typedef void(∗ parse_cmd_fn_t)(int)

  Function type for PARSE_CMD_FN.
- typedef void(∗ parse_cmd_fn_arg_t)(int, const char ∗, int)

  Function type for PARSE_CMD_FN_ARG.

Enumerations

- enum parse_cmd_type

  Types for parsing.

Functions

- int parse_cmdline (int ∗argc, const char ∗∗∗argv, char arg0,...)

  Parse the command-line for specified arguments and store the values into variables.

10.17.1 Detailed Description

10.17.2 Function Documentation

10.17.2.1 int parse_cmdline (int ∗ *argc,* const char ∗∗∗ *argv,* char *arg0,* ...)

Parse the command-line for specified arguments and store the values into variables.

This Functions gets the command-line, and a list of command-descriptors. Then, the command-line is parsed according to the given descriptors, storing strings, switches and numeric arguments at given addresses, and possibly calling specified functions. A default help descriptor is added. Its purpose is to present a short command overview in the case the given command-line does not fit to the descriptors.

Each command-descriptor has the following form:

short option char, *long option name*, *comment*, *type*, *val*, *addr*.

The *short option char* specifies the short form of the described option. The short form will be recognized after a single dash, or in a group of short options preceeded by a single dash. Specify ' ' if no short form should be used.

The *long option name* specifies the long form of the described option. The long form will be recognized after two dashes. Specify 0 if no long form should be used for this option.

The *comment* is a string that will be used when presenting the short command-line help.

The *type* specifies, if the option should be recognized as

- a number (PARSE_CMD_INT),
- a switch (PARSE_CMD_SWITCH),
- a string (PARSE_CMD_STRING),
- a function call (PARSE_CMD_FN, PARSE_CMD_FN_ARG),
- an increment/decrement operator (PARSE_CMD_INC, PARSE_CMD_DEC).

If type is PARSE_CMD_INT, the option requires a second argument on the command-line after the option. This argument is parsed as a number. It can be preceeded by 0x to present a hex-value or by 0 to present an octal form. *addr* is interpreted as an int-pointer. The scanned argument from the command-line is stored in this pointer.

If *type* is PARSE_CMD_SWITCH, *addr* must be a pointer to int, and the value from *val* is stored at this pointer.

With PARSE_CMD_STRING, an additional argument is expected at the cmdline. *addr* must be a pointer to const char*, and a pointer to the argument on the command line is stored at this pointer. The value in *val* is a default value, which is stored at *addr* if the corresponding option is not given on the command line.

With PARSE_CMD_FN_ARG, *addr* is interpreted as a function pointer of type parse_cmd_fn_t. It will be called with *val* as argument if the corresponding option is found.

If *type* is PARSE_CMD_FN_ARG, *addr* is as a function pointer of type parse_cmd_fn_arg_t, and handled similar to PARSE_CMD_FN. An additional argument is expected at the command line, however. It is given to the called function as 2nd argument, and parsed as an integer as with PARSE_CMD_INT as a third argument.

If *type* is PARSE_CMD_INC or PARSE_CMD_DEC, *addr* is interpreted as an int-pointer. The value of *val* is stored to this pointer first. For every occurence of the option in the command line, the integer referenced by *addr* is incremented or decremented, respectively.

The list of command-descriptors is terminated by specifying a binary 0 for the short option char.

Note: The short option char 'h' and the long option name "help" must not be specified. They are used for the default help descriptor and produce a short command-options help when specified on the command-line.

Parameters

argc	pointer to number of command line parameters as passed to main
argv	pointer to array of command line parameters as passed to main
arg0	format list describing the command line options to parse for

Returns

0 if the command-line was successfully parsed, otherwise:

- -1 if the given descriptors are somehow wrong.
- -2 if not enough memory was available to hold temporary structs.
- -3 if the given command-line args did not meet the specified set.
- -4 if the help-option was given.

Upon return, argc and argv point to a list of arguments that were not scanned as arguments. See getoptlong for details on scanning.

10.18 Console API

Console interface.

Collaboration diagram for Console API:

Data Structures

- class L4Re::Console

  Console class.

10.18.1 Detailed Description

Console interface.

10.19 Consumer

Collaboration diagram for Consumer:

Functions

- long l4shmc_enable_chunk (l4shmc_chunk_t ∗chunk)

  Enable a signal connected with a chunk.
- long l4shmc_wait_chunk (l4shmc_chunk_t ∗chunk)

  Wait on a specific chunk.
- long l4shmc_wait_chunk_to (l4shmc_chunk_t ∗chunk, l4_timeout_t timeout)

  Check whether a specific chunk has an event pending, with timeout.
- long l4shmc_wait_chunk_try (l4shmc_chunk_t ∗chunk)

  Check whether a specific chunk has an event pending.
- long l4shmc_chunk_consumed (l4shmc_chunk_t ∗chunk)

  Mark a chunk as free.
- long l4shmc_is_chunk_ready (l4shmc_chunk_t ∗chunk)

  Check whether data is available.
- long l4shmc_chunk_size (l4shmc_chunk_t ∗chunk)

  Get current size of a chunk.

10.19.1 Detailed Description

10.19.2 Function Documentation

10.19.2.1 long l4shmc_chunk_consumed (l4shmc_chunk_t ∗ *chunk*) [inline]

Mark a chunk as free.

Parameters

chunk	Chunk to mark as free.

Return values

0	Success.
<0	Error.

Examples:

examples/libs/shmc/prodcons.c.

10.19.2.2 long l4shmc_chunk_size (l4shmc_chunk_t ∗ *chunk*) [inline]

Get current size of a chunk.

Parameters

chunk	Chunk.

Return values

0	Success.
<0	Error.

Examples:

examples/libs/shmc/prodcons.c.

10.19.2.3 long l4shmc_enable_chunk (l4shmc_chunk_t * *chunk*)

Enable a signal connected with a chunk.

Parameters

chunk	Chunk to enable.

Return values

0	Success.
<0	Error.

A signal must be enabled before waiting when the consumer waits on any signal. Enabling is not needed if the consumer waits for a specific signal or chunk.

10.19.2.4 long l4shmc_is_chunk_ready (l4shmc_chunk_t * *chunk*) [inline]

Check whether data is available.

Parameters

chunk	Chunk to check.

Return values

0	Success.
<0	Error.

10.19.2.5 long l4shmc_wait_chunk (l4shmc_chunk_t * *chunk*) [inline]

Wait on a specific chunk.

Parameters

chunk	Chunk to wait for.

Return values

0	Success.
<0	Error.

Examples:

examples/libs/shmc/prodcons.c.

10.19.2.6 long l4shmc_wait_chunk_to (l4shmc_chunk_t * *chunk,* l4_timeout_t *timeout*)

Check whether a specific chunk has an event pending, with timeout.

Parameters

chunk	Chunk to check.
timeout	Timeout.

Return values

0	Success.
<0	Error.

The return code indicates whether an event was pending or not. Success means an event was pending, if an receive timeout error is returned no event was pending.

10.19.2.7 long l4shmc_wait_chunk_try (l4shmc_chunk_t * *chunk*) [inline]

Check whether a specific chunk has an event pending.

Parameters

chunk	Chunk to check.

Return values

0	Success.
<0	Error.

The return code indicates whether an event was pending or not. Success means an event was pending, if an receive timeout error is returned no event was pending.

10.20 Consumer

Collaboration diagram for Consumer:

Functions

- long l4shmc_enable_signal (l4shmc_signal_t *signal)

  Enable a signal.
- long l4shmc_wait_any (l4shmc_signal_t **retsignal)

  Wait on any signal.
- long l4shmc_wait_any_try (l4shmc_signal_t **retsignal)

  Check whether any waited signal has an event pending.
- long l4shmc_wait_any_to (l4_timeout_t timeout, l4shmc_signal_t **retsignal)

  Wait for any signal with timeout.
- long l4shmc_wait_signal (l4shmc_signal_t *signal)

  Wait on a specific signal.
- long l4shmc_wait_signal_to (l4shmc_signal_t *signal, l4_timeout_t timeout)

  Wait on a specific signal, with timeout.
- long l4shmc_wait_signal_try (l4shmc_signal_t *signal)

  Check whether a specific signal has an event pending.

10.20.1 Detailed Description

10.20.2 Function Documentation

10.20.2.1 long l4shmc_enable_signal (l4shmc_signal_t * *signal*)

Enable a signal.

Parameters

signal	Signal to enable.

Return values

0	Success.
<0	Error.

A signal must be enabled before waiting when the consumer waits on any signal. Enabling is not needed if the consumer waits for a specific signal or chunk.

10.20.2.2 long l4shmc_wait_any (l4shmc_signal_t ** *retsignal*) [inline]

Wait on any signal.

Parameters

out	*retsignal*	Signal received.

Return values

0	Success.
<0	Error.

10.20.2.3 long l4shmc_wait_any_to (l4_timeout_t *timeout*, l4shmc_signal_t ** *retsignal*)

Wait for any signal with timeout.

Parameters

	timeout	Timeout.
out	*retsignal*	Signal that has the event pending if any.

Return values

0	Success.
<0	Error.

The return code indicates whether an event was pending or not. Success means an event was pending, if an receive timeout error is returned no event was pending.

10.20.2.4 long l4shmc_wait_any_try (l4shmc_signal_t ** *retsignal*) [inline]

Check whether any waited signal has an event pending.

Parameters

out	*retsignal*	Signal that has the event pending if any.

Return values

0	Success.
<0	Error.

The return code indicates whether an event was pending or not. Success means an event was pending, if an receive timeout error is returned no event was pending.

10.20.2.5 long l4shmc_wait_signal (l4shmc_signal_t * *signal*) [inline]

Wait on a specific signal.

Parameters

signal	Signal to wait for.

Return values

0	Success.
<0	Error.

Examples:

examples/libs/shmc/prodcons.c.

10.20.2.6 long l4shmc_wait_signal_to (l4shmc_signal_t * *signal,* l4_timeout_t *timeout*)

Wait on a specific signal, with timeout.

Parameters

signal	Signal to wait for.
timeout	Timeout.

Return values

0	Success.
<0	Error.

10.20.2.7 long l4shmc_wait_signal_try (l4shmc_signal_t * *signal*) [inline]

Check whether a specific signal has an event pending.

Parameters

signal	Signal to check.

Return values

0	Success.
<0	Error.

The return code indicates whether an event was pending or not. Success means an event was pending, if an receive timeout error is returned no event was pending.

10.21 DMA API for L4Re

10.22 DMA Space Interface

DMA Space C interface.

Collaboration diagram for DMA Space Interface:

Typedefs

- typedef l4_cap_idx_t l4re_dma_space_t

  DMA space capability type.

Functions

- long l4re_dma_space_map (l4re_dma_space_t dma, l4re_ds_t src, l4_addr_t offset, l4_size_t *size, unsigned long attrs, enum l4re_dma_space_direction dir, l4re_dma_space_dma_addr_t *dma_addr) L4_NOTHROW

  Map the given part of this data space into the DMA address space.
- long l4re_dma_space_unmap (l4re_dma_space_t dma, l4re_dma_space_dma_addr_t dma_addr, l4_size_t size, unsigned long attrs, enum l4re_dma_space_direction dir) L4_NOTHROW

  Unmap the given part of this data space from the DMA address space.

10.22.1 Detailed Description

DMA Space C interface.

10.22.2 Typedef Documentation

10.22.2.1 typedef l4_cap_idx_t l4re_dma_space_t

DMA space capability type.

DMA Address Space. A Dma_space represents the DMA address space of a device. Whenever a device needs direct access to parts of an L4Re::Dataspace, that part of the data space must be mapped to the DMA address space that is assigned to that device. After the DMA accesses to the memory are finished the memory must be unmapped from the device's DMA address space.

Mapping to a DMA address space, using map(), makes the given parts of the data space visible to the associated device at the returned DMA address. As long as the memory is mapped into a DMA space it is 'pinned' and cannot be subject to dynamic memory management such as swapping. Additionally, map() is responsible for the necessary syncing operations before the DMA.

unmap() is the reverse operation to map() and unmaps the given data-space part for the DMA address space. unmap() is responsible for the necessary sync operations after the DMA.

Definition at line 50 of file dma_space.h.

10.22.3 Function Documentation

10.22.3.1 long l4re_dma_space_map (l4re_dma_space_t *dma*, l4re_ds_t *src*, l4_addr_t *offset*, l4_size_t * *size*, unsigned long *attrs*, enum l4re_dma_space_direction *dir*, l4re_dma_space_dma_addr_t * *dma_addr*)

Map the given part of this data space into the DMA address space.

Parameters

dma	DMA space capability

Precondition

requires capability rights: {R}

Parameters

in	*src*	Source data space (that describes the memory). Caller needs write rights to the data space.
in	*offset*	The offset (bytes) with in `src`.
in,out	*size*	The size (bytes) of the of the region to be mapped to for DMA, after successful mapping the size returned is the size mapped for DMA as single block, this size might be smaller than the original input size, in this case the caller might call map() again with a new offset and the remaining size.
in	*attrs*	The attributes used for this DMA mapping (a combination of Dma_space::↩ Attribute values).
in	*dir*	The direction of the DMA transfer issued with this mapping. The same value must later be passed to unmap().
out	*dma_addr*	The DMA address to use for DMA with the associated device.

Returns

0 in the case of success, a negative error code otherwise.

10.22.3.2 long l4re_dma_space_unmap (l4re_dma_space_t *dma*, l4re_dma_space_dma_addr_t *dma_addr*, l4_size_t *size*, unsigned long *attrs*, enum l4re_dma_space_direction *dir*)

Unmap the given part of this data space from the DMA address space.

Parameters

dma	DMA space capability

Precondition

requires capability rights: {R}

Parameters

dma_addr	The DMA address (returned by Dma_space::map()).
size	The size (bytes) of the memory region to unmap.
attrs	The attributes for the unmap (currently none).
dir	The direction of the finished DMA operation.

10.23 Dataspace interface

Dataspace C interface.

Collaboration diagram for Dataspace interface:

Data Structures

- struct l4re_ds_stats_t

  Information about the data space.

Typedefs

- typedef l4_cap_idx_t l4re_ds_t

  Dataspace type.
- typedef l4_cap_idx_t l4re_namespace_t

  Namespace type.

Enumerations

- enum l4re_ds_map_flags { , L4RE_DS_MAP_NORMAL = 0x00, L4RE_DS_MAP_CACHEABLE = L4RE_DS_MAP_NORMAL, L4RE_DS_MAP_BUFFERABLE = 0x10, L4RE_DS_MAP_UNCACHEABLE = 0x20, L4RE_DS_MAP_CACHING_MASK = 0x30, L4RE_DS_MAP_CACHING_SHIFT = 4 }

  Flags to specify the memory mapping type of a request.

Functions

- long l4re_ds_clear (const l4re_ds_t ds, l4_addr_t offset, unsigned long size) L4_NOTHROW
- long l4re_ds_allocate (const l4re_ds_t ds, l4_addr_t offset, l4_size_t size) L4_NOTHROW
- int l4re_ds_copy_in (const l4re_ds_t ds, l4_addr_t dst_offs, const l4re_ds_t src, l4_addr_t src_offs, unsigned long size) L4_NOTHROW
- long l4re_ds_size (const l4re_ds_t ds) L4_NOTHROW
- long l4re_ds_flags (const l4re_ds_t ds) L4_NOTHROW
- int l4re_ds_info (const l4re_ds_t ds, l4re_ds_stats_t *stats) L4_NOTHROW
- int l4re_ds_phys (const l4re_ds_t ds, l4_addr_t offset, l4_addr_t *phys_addr, l4_size_t *phys_size) L4_NOTHROW

  Return physical address.

10.23.1 Detailed Description

Dataspace C interface.

10.23.2 Enumeration Type Documentation

10.23.2.1 enum l4re_ds_map_flags

Flags to specify the memory mapping type of a request.

Enumerator

- ***L4RE_DS_MAP_NORMAL*** request normal memory mapping
- ***L4RE_DS_MAP_CACHEABLE*** request normal memory mapping
- ***L4RE_DS_MAP_BUFFERABLE*** request bufferable (write buffered) mappings
- ***L4RE_DS_MAP_UNCACHEABLE*** request uncacheable memory mappings
- ***L4RE_DS_MAP_CACHING_MASK*** mask for caching flags
- ***L4RE_DS_MAP_CACHING_SHIFT*** shift value for caching flags

Definition at line 54 of file dataspace.h.

10.23.3 Function Documentation

10.23.3.1 long l4re_ds_allocate (const l4re_ds_t *ds*, l4_addr_t *offset*, l4_size_t *size*)

Returns

0 on success, <0 on errors

See also

L4Re::Dataspace::allocate

10.23.3.2 long l4re_ds_clear (const l4re_ds_t *ds*, l4_addr_t *offset*, unsigned long *size*)

Returns

0 on success, <0 on errors

See also

L4Re::Dataspace::clear

10.23.3.3 int l4re_ds_copy_in (const l4re_ds_t *ds*, l4_addr_t *dst_offs*, const l4re_ds_t *src*, l4_addr_t *src_offs*, unsigned long *size*)

Returns

0 on success, <0 on errors

See also

L4Re::Dataspace::copy_in

10.23.3.4 long l4re_ds_flags (const l4re_ds_t *ds*)

See also

L4Re::Dataspace::flags

10.23.3.5 int l4re_ds_info (const l4re_ds_t *ds*, l4re_ds_stats_t * *stats*)

See also

L4Re::Dataspace::info

10.23.3.6 int l4re_ds_phys (const l4re_ds_t *ds*, l4_addr_t *offset*, l4_addr_t * *phys_addr*, l4_size_t * *phys_size*)

Return physical address.

Parameters

ds	Dataspace
offset	Offset in bytes in dataspace

Return values

phys_addr	Physical address
phys_size	Size of physically contiguous region starting from *phys_addr* (in bytes).

Returns

0 for success, <0 on error

The function returns the physical address of an offset in a dataspace. Use multiple calls of the function to get all physical regions in case of physically non-contiguous dataspaces.

See also

L4Re::Dataspace::phys

10.23.3.7 long l4re_ds_size (const l4re_ds_t *ds*)

Returns

size of dataspace, <0 on errors

See also

L4Re::Dataspace::size

10.24 Debug interface

Collaboration diagram for Debug interface:

Functions

- void l4re_debug_obj_debug (l4_cap_idx_t srv, unsigned long function) L4_NOTHROW

  Call debug function of L4Re service.

10.24.1 Detailed Description

10.24.2 Function Documentation

10.24.2.1 void l4re_debug_obj_debug (l4_cap_idx_t *srv*, unsigned long *function*)

Call debug function of L4Re service.

Parameters

srv	Object to call.
function	Function to call.

See also

L4Re::Debug_obj::debug

10.25 Debugging API

Debugging Interface.

Collaboration diagram for Debugging API:

Data Structures

- class L4Re::Debug_obj

  Debug interface.

10.25.1 Detailed Description

Debugging Interface.

The debugging interface can be provided to retrieve, or log debugging information for an object. Each class may realize the debug interface to provide debugging functionality. For example, the region-map objects provide a facility to dump the currently established memory regions.

See also

L4::Debug_obj for more information.

10.26 EDID parsing functionality

Enumerations

- enum Libedid_consts { Libedid_block_size = 128 }

  EDID constants.

Functions

- int libedid_check_header (const unsigned char ∗edid)

  Check for valid EDID header.
- int libedid_checksum (const unsigned char ∗edid)

  Calculates the EDID checksum.
- unsigned libedid_version (const unsigned char ∗edid)

  Returns the EDID version number.
- unsigned libedid_revision (const unsigned char ∗edid)

  Returns the EDID revision number.
- void libedid_pnp_id (const unsigned char ∗edid, unsigned char ∗id)

  Extracts the display's PnP ID.
- void libedid_prefered_resolution (const unsigned char ∗edid, unsigned ∗w, unsigned ∗h)

  Extract the display's prefered mode.
- unsigned libedid_num_ext_blocks (const unsigned char ∗edid)

  Get the number of EDID extension blocks.
- unsigned libedid_dump_standard_timings (const unsigned char ∗edid)

  Dump the standard timings to stdout.
- void libedid_dump (const unsigned char ∗edid)

  Dump raw EDID data to stdout.

10.26.1 Detailed Description

10.26.2 Enumeration Type Documentation

10.26.2.1 enum Libedid_consts

EDID constants.

Enumerator

Libedid_block_size Size of one EDID block in bytes.

Definition at line 23 of file edid.h.

10.26.3 Function Documentation

10.26.3.1 int libedid_check_header (const unsigned char ∗ *edid*)

Check for valid EDID header.

Parameters

edid	Pointer to a 128byte EDID block

Returns

0 if the header is correct, -EINVAL otherwise

10.26.3.2 int libedid_checksum (const unsigned char * *edid*)

Calculates the EDID checksum.

Parameters

edid	Pointer to a 128byte EDID block

Returns

0 if checksum is correct, -EINVAL otherwise

10.26.3.3 void libedid_dump (const unsigned char * *edid*)

Dump raw EDID data to stdout.

Parameters

edid	Pointer to a 128byte EDID block

10.26.3.4 unsigned libedid_dump_standard_timings (const unsigned char * *edid*)

Dump the standard timings to stdout.

Parameters

edid	Pointer to a 128byte EDID block

Returns

Number of standard timings stored in EDID

10.26.3.5 unsigned libedid_num_ext_blocks (const unsigned char * *edid*)

Get the number of EDID extension blocks.

Parameters

edid	Pointer to a 128byte EDID block

Returns

Number of EDID extension blocks

10.26.3.6 void libedid_pnp_id (const unsigned char * *edid*, unsigned char * *id*)

Extracts the display's PnP ID.

Parameters

edid	Pointer to a 128byte EDID block

Return values

id	Return the PnP id. Must point to 4 bytes.

10.26.3.7 void libedid_prefered_resolution (const unsigned char * *edid*, unsigned * *w*, unsigned * *h*)

Extract the display's prefered mode.

Parameters

edid	Pointer to a 128byte EDID block

Return values

w	X resolution of prefered video mode in pixels.
h	Y resolution of prefered video mode in pixels.

10.26.3.8 unsigned libedid_revision (const unsigned char * *edid*)

Returns the EDID revision number.

Parameters

edid	Pointer to a 128 EDID block

Returns

Revision number

10.26.3.9 unsigned libedid_version (const unsigned char * *edid*)

Returns the EDID version number.

Parameters

edid	Pointer to a 128byte EDID block

Returns

Version number

10.27 ELF binary format

Functions and types related to ELF binaries.

Collaboration diagram for ELF binary format:

Files

- file elf.h

  ELF definition.

Data Structures

- struct Elf32_Ehdr

  ELF32 header.
- struct Elf64_Ehdr

  ELF64 header.
- struct Elf32_Shdr

  ELF32 section header - figure 1-9, page 1-9.
- struct Elf64_Shdr

  ELF64 section header.
- struct Elf32_Phdr

  ELF32 program header.
- struct Elf64_Phdr

  ELF64 program header.
- struct Elf32_Dyn

  ELF32 dynamic entry.
- struct Elf64_Dyn

  ELF64 dynamic entry.
- struct Elf32_Sym

  ELF32 symbol table entry.
- struct Elf64_Sym

  ELF64 symbol table entry.

Macros

- #define EI_NIDENT 16

  number of characters
- #define EI_CLASS 4

  ELF class byte index.
- #define EI_CLASS 4

  ELF class byte index.

- #define ELFCLASSNONE 0

  Invalid ELF class.
- #define ELFCLASSNONE 0

  Invalid ELF class.
- #define ELFCLASS32 1

  32-bit objects
- #define ELFCLASS64 2

  64-bit objects
- #define ELFCLASSNUM 3

  Mask for 32-bit or 64-bit class.
- #define EI_DATA 5

  Data encoding byte index.
- #define EI_DATA 5

  Data encoding byte index.
- #define ELFDATANONE 0

  Invalid data encoding.
- #define ELFDATANONE 0

  Invalid data encoding.
- #define ELFDATA2LSB 1

  2's complement, little endian
- #define ELFDATA2LSB 1

  2's complement, little endian
- #define ELFDATA2MSB 2

  2's complement, big endian
- #define ELFDATA2MSB 2

  2's complement, big endian
- #define EI_VERSION 6

  File version byte index.
- #define EI_VERSION 6

  File version byte index.
- #define EI_OSABI 7

  OS ABI identification.
- #define EI_OSABI 7

  OS ABI identification.
- #define ELFOSABI_NONE 0

  UNIX System V ABI.
- #define ELFOSABI_SYSV 0

  Alias.
- #define ELFOSABI_SYSV 0

  Alias.
- #define ELFOSABI_HPUX 1

  HP-UX.
- #define ELFOSABI_HPUX 1

  HP-UX.
- #define ELFOSABI_NETBSD 2

  NetBSD.
- #define ELFOSABI_LINUX 3

  Linux.
- #define ELFOSABI_SOLARIS 6

  Sun Solaris.
- #define ELFOSABI_AIX 7

IBM AIX.

- #define ELFOSABI_IRIX 8

  SGI Irix.
- #define ELFOSABI_FREEBSD 9

  FreeBSD.
- #define ELFOSABI_TRU64 10

  Compaq TRU64 UNIX.
- #define ELFOSABI_MODESTO 11

  Novell Modesto.
- #define ELFOSABI_OPENBSD 12

  OpenBSD.
- #define ELFOSABI_ARM 97

  ARM.
- #define ELFOSABI_STANDALONE 255

  Standalone (embedded) application.
- #define ELFOSABI_STANDALONE 255

  Standalone (embedded) application.
- #define EI_ABIVERSION 8

  ABI version.
- #define EI_ABIVERSION 8

  ABI version.
- #define EI_PAD 9

  Byte index of padding bytes.
- #define EI_PAD 9

  Byte index of padding bytes.
- #define ET_NONE 0

  no file type
- #define ET_REL 1

  relocatable file
- #define ET_EXEC 2

  executable file
- #define ET_DYN 3

  shared object file
- #define ET_CORE 4

  core file
- #define ET_LOPROC 0xff00

  processor-specific
- #define ET_HIPROC 0xffff

  processor-specific
- #define EM_NONE 0

  no machine
- #define EM_M32 1

  AT&T WE 32100.
- #define EM_SPARC 2

  SPARC.
- #define EM_386 3

  Intel 80386.
- #define EM_68K 4

  Motorola 68000.
- #define EM_88K 5

  Motorola 88000.

- #define EM_860 7

  Intel 80860.
- #define EM_MIPS 8

  MIPS RS3000 big-endian.
- #define EM_MIPS_RS4_BE 10

  MIPS RS4000 big-endian.
- #define EM_SPARC64 11

  SPARC 64-bit.
- #define EM_PARISC 15

  HP PA-RISC.
- #define EM_VPP500 17

  Fujitsu VPP500.
- #define EM_SPARC32PLUS 18

  Sun's V8plus.
- #define EM_960 19

  Intel 80960.
- #define EM_PPC 20

  PowerPC.
- #define EM_V800 36

  NEC V800.
- #define EM_FR20 37

  Fujitsu FR20.
- #define EM_RH32 38

  TRW RH-32.
- #define EM_RCE 39

  Motorola RCE.
- #define EM_ARM 40

  Advanced RISC Machines ARM.
- #define EM_ALPHA 41

  Digital Alpha.
- #define EM_SH 42

  Hitachi SuperH.
- #define EM_SPARCV9 43

  SPARC v9 64-bit.
- #define EM_TRICORE 44

  Siemens Tricore embedded processor.
- #define EM_ARC 45

  Argonaut RISC Core, Argonaut Techn Inc.
- #define EM_H8_300 46

  Hitachi H8/300.
- #define EM_H8_300H 47

  Hitachi H8/300H.
- #define EM_H8S 48

  Hitachi H8/S.
- #define EM_H8_500 49

  Hitachi H8/500.
- #define EM_IA_64 50

  HP/Intel IA-64.
- #define EM_MIPS_X 51

  Stanford MIPS-X.
- #define EM_COLDFIRE 52

Motorola Coldfire.

- #define EM_68HC12 53

  Motorola M68HC12.
- #define EM_X86_64 62

  Advanced Micro Devices x86-64.
- #define EV_NONE 0

  Invalid version.
- #define EV_CURRENT 1

  Current version.
- #define EI_MAG0 0

  file id
- #define EI_MAG1 1

  file id
- #define EI_MAG2 2

  file id
- #define EI_MAG3 3

  file id
- #define ELFMAG0 0x7f

  e_ident[EI_MAG0]
- #define ELFMAG1 'E'

  e_ident[EI_MAG1]
- #define ELFMAG2 'L'

  e_ident[EI_MAG2]
- #define ELFMAG3 'F'

  e_ident[EI_MAG3]
- #define ELFCLASSS32 1

  32-bit object
- #define ELFCLASSS64 2

  64-bit object
- #define SHN_UNDEF 0

  undefined section header entry
- #define SHN_LORESERVE 0xff00

  lower bound of reserved indexes
- #define SHN_LOPROC 0xff00

  lower bound of proc spec entr
- #define SHN_HIPROC 0xff1f

  upper bound of proc spec entr
- #define SHN_ABS 0xfff1

  absolute values for ref
- #define SHN_COMMON 0xfff2

  common symbols
- #define SHN_HIRESERVE 0xffff

  upper bound of reserved indexes
- #define SHT_INIT_ARRAY 14

  Array of constructors.
- #define SHT_FINI_ARRAY 15

  Array of destructors.
- #define SHT_PREINIT_ARRAY 16

  Array of pre-constructors.
- #define SHT_GROUP 17

  Section group.

- #define SHT_SYMTAB_SHNDX 18

  Extended section indeces.
- #define SHT_NUM 19

  Number of defined types.
- #define SHF_WRITE 0x1

  writeable during execution
- #define SHF_ALLOC 0x2

  section occupies virt memory
- #define SHF_EXECINSTR 0x4

  code section
- #define SHF_MERGE 0x10

  Might be merged.
- #define SHF_STRINGS 0x20

  Contains nul-terminated strings.
- #define SHF_INFO_LINK 0x40

  'sh_info' contains SHT index
- #define SHF_LINK_ORDER 0x80

  Preserve order after combining.
- #define SHF_OS_NONCONFORMING 0x100

  Non-standard OS specific handling required.
- #define SHF_GROUP 0x200

  Section is member of a group.
- #define SHF_TLS 0x400

  Section hold thread-local data.
- #define SHF_MASKOS 0x0ff00000

  OS-specific.
- #define SHF_MASKPROC 0xf0000000

  proc spec mask
- #define PT_NULL 0

  array is unused
- #define PT_LOAD 1

  loadable
- #define PT_DYNAMIC 2

  dynamic linking information
- #define PT_INTERP 3

  path to interpreter
- #define PT_NOTE 4

  auxiliary information
- #define PT_SHLIB 5

  reserved
- #define PT_PHDR 6

  location of the pht itself
- #define PT_TLS 7

  Thread-local storage segment.
- #define PT_NUM 8

  Number of defined types.
- #define PT_LOOS 0x60000000

  os spec.
- #define PT_HIOS 0x6fffffff

  os spec.
- #define PT_LOPROC 0x70000000

processor spec.

- #define PT_HIPROC 0x7fffffff

  processor spec.
- #define PT_GNU_EH_FRAME (PT_LOOS + 0x474e550)

  EH frame information.
- #define PT_GNU_STACK (PT_LOOS + 0x474e551)

  Flags for stack.
- #define PT_GNU_RELRO (PT_LOOS + 0x474e552)

  Read only after reloc.
- #define PT_L4_STACK (PT_LOOS + 0x12)

  Address of the stack.
- #define PT_L4_KIP (PT_LOOS + 0x13)

  Address of the KIP.
- #define PT_L4_AUX (PT_LOOS + 0x14)

  Address of the AUX strcutures.
- #define NT_PRSTATUS 1

  Contains copy of prstatus struct.
- #define NT_FPREGSET 2

  Contains copy of fpregset struct.
- #define NT_PRPSINFO 3

  Contains copy of prpsinfo struct.
- #define NT_PRXREG 4

  Contains copy of prxregset struct.
- #define NT_TASKSTRUCT 4

  Contains copy of task structure.
- #define NT_PLATFORM 5

  String from sysinfo(SI_PLATFORM)
- #define NT_AUXV 6

  Contains copy of auxv array.
- #define NT_GWINDOWS 7

  Contains copy of gwindows struct.
- #define NT_ASRS 8

  Contains copy of asrset struct.
- #define NT_PSTATUS 10

  Contains copy of pstatus struct.
- #define NT_PSINFO 13

  Contains copy of psinfo struct.
- #define NT_PRCRED 14

  Contains copy of prcred struct.
- #define NT_UTSNAME 15

  Contains copy of utsname struct.
- #define NT_LWPSTATUS 16

  Contains copy of lwpstatus struct.
- #define NT_LWPSINFO 17

  Contains copy of lwpinfo struct.
- #define NT_PRFPXREG 20

  Contains copy of fprxregset struct.
- #define NT_VERSION 1

  Contains a version string.
- #define DT_NULL 0

  Dynamic Array Tags, d_tag - figure 2-10, page 2-12.

- #define DT_NEEDED 1

  name of a needed library
- #define DT_PLTRELSZ 2

  total size of relocation entry
- #define DT_PLTGOT 3

  address assoc with prog link table
- #define DT_HASH 4

  address of symbol hash table
- #define DT_STRTAB 5

  address of string table
- #define DT_SYMTAB 6

  address of symbol table
- #define DT_RELA 7

  address of relocation table
- #define DT_RELASZ 8

  total size of relocation table
- #define DT_RELAENT 9

  size of DT_RELA relocation entry
- #define DT_STRSZ 10

  size of the string table
- #define DT_SYMENT 11

  size of a symbol table entry
- #define DT_INIT 12

  address of initialization function
- #define DT_FINI 13

  address of termination function
- #define DT_SONAME 14

  name of the shared object
- #define DT_RPATH 15

  search library path
- #define DT_SYMBOLIC 16

  alter symbol resolution algorithm
- #define DT_REL 17

  address of relocation table
- #define DT_RELSZ 18

  total size of DT_REL relocation table
- #define DT_RELENT 19

  size of the DT_REL relocation entry
- #define DT_PTRREL 20

  type of relocation entry
- #define DT_DEBUG 21

  for debugging purposes
- #define DT_TEXTREL 22

  at least on entry changes r/o section
- #define DT_JMPREL 23

  address of relocation entries
- #define DT_BIND_NOW 24

  Process relocations of object.
- #define DT_INIT_ARRAY 25

  Array with addresses of init fct.
- #define DT_FINI_ARRAY 26

Array with addresses of fini fct.

- #define DT_INIT_ARRAYSZ 27

  Size in bytes of DT_INIT_ARRAY.
- #define DT_FINI_ARRAYSZ 28

  Size in bytes of DT_FINI_ARRAY.
- #define DT_RUNPATH 29

  Library search path.
- #define DT_FLAGS 30

  Flags for the object being loaded.
- #define DT_ENCODING 32

  Start of encoded range.
- #define DT_PREINIT_ARRAY 32

  Array with addresses of preinit fct.
- #define DT_PREINIT_ARRAYSZ 33

  size in bytes of DT_PREINIT_ARRAY
- #define DT_NUM 34

  Number used.
- #define DT_LOOS 0x6000000d

  Start of OS-specific.
- #define DT_HIOS 0x6ffff000

  End of OS-specific.
- #define DT_LOPROC 0x70000000

  processor spec.
- #define DT_HIPROC 0x7fffffff

  processor spec.
- #define DF_ORIGIN 0x00000001

  Object may use DF_ORIGIN.
- #define DF_SYMBOLIC 0x00000002

  Symbol resolutions starts here.
- #define DF_TEXTREL 0x00000004

  Object contains text relocations.
- #define DF_BIND_NOW 0x00000008

  No lazy binding for this object.
- #define DF_STATIC_TLS 0x00000010

  Module uses the static TLS model.
- #define DF_1_NOW 0x00000001

  Set RTLD_NOW for this object.
- #define DF_1_GLOBAL 0x00000002

  Set RTLD_GLOBAL for this object.
- #define DF_1_GROUP 0x00000004

  Set RTLD_GROUP for this object.
- #define DF_1_NODELETE 0x00000008

  Set RTLD_NODELETE for this object.
- #define DF_1_LOADFLTR 0x00000010

  Trigger filtee loading at runtime.
- #define DF_1_INITFIRST 0x00000020

  Set RTLD_INITFIRST for this object.
- #define DF_1_NOOPEN 0x00000040

  Set RTLD_NOOPEN for this object.
- #define DF_1_ORIGIN 0x00000080

  $ORIGIN must be handled.

- #define DF_1_DIRECT 0x00000100

  Direct binding enabled.
- #define DF_1_INTERPOSE 0x00000400

  Object is used to interpose.
- #define DF_1_NODEFLIB 0x00000800

  Ignore default lib search path.
- #define DF_1_NODUMP 0x00001000

  Object can't be dldump'ed.
- #define DF_1_CONFALT 0x00002000

  Configuration alternative created.
- #define DF_1_ENDFILTEE 0x00004000

  Filtee terminates filters search.
- #define DF_1_DISPRELDNE 0x00008000

  Disp reloc applied at build time.
- #define DF_1_DISPRELPND 0x00010000

  Disp reloc applied at run-time.
- #define DF_P1_LAZYLOAD 0x00000001

  Lazyload following object.
- #define DF_P1_GROUPPERM 0x00000002

  Symbols from next object are not generally available.
- #define R_386_NONE 0

  none
- #define R_386_32 1

  S + A.
- #define R_386_PC32 2

  S + A - P.
- #define R_386_GOT32 3

  G + A - P.
- #define R_386_PLT32 4

  L + A - P.
- #define R_386_COPY 5

  none
- #define R_386_GLOB_DAT 6

  S.
- #define R_386_JMP_SLOT 7

  S.
- #define R_386_RELATIVE 8

  B + A.
- #define R_386_GOTOFF 9

  S + A - GOT.
- #define R_386_GOTPC 10

  GOT + A - P.
- #define STB_LOCAL 0

  not visible outside object file
- #define STB_GLOBAL 1

  visible to all objects beeing combined
- #define STB_WEAK 2

  resemble global symbols
- #define STB_LOOS 10

  os specific
- #define STB_HIOS 12

os specific

- #define STB_LOPROC 13

  proc specific
- #define STB_HIPROC 15

  proc specific
- #define STT_NOTYPE 0

  symbol's type not specified
- #define STT_OBJECT 1

  associated with a data object
- #define STT_FUNC 2

  associated with a function or other code
- #define STT_SECTION 3

  associated with a section
- #define STT_FILE 4

  source file name associated with object
- #define STT_LOOS 10

  os specific
- #define STT_HIOS 12

  os specific
- #define STT_LOPROC 13

  proc specific
- #define STT_HIPROC 15

  proc specific

ELF types

- typedef l4_uint32_t Elf32_Addr

  size 4 align 4
- typedef l4_uint32_t Elf32_Off

  size 4 align 4
- typedef l4_uint16_t Elf32_Half

  size 2 align 2
- typedef l4_uint32_t Elf32_Word

  size 4 align 4
- typedef l4_int32_t Elf32_Sword

  size 4 align 4
- typedef l4_uint64_t Elf64_Addr

  size 8 align 8
- typedef l4_uint64_t Elf64_Off

  size 8 align 8
- typedef l4_uint16_t Elf64_Half

  size 2 align 2
- typedef l4_uint32_t Elf64_Word

  size 4 align 4
- typedef l4_int32_t Elf64_Sword

  size 4 align 4
- typedef l4_uint64_t Elf64_Xword

  size 8 align 8
- typedef l4_int64_t Elf64_Sxword

  size 8 align 8

10.27.1 Detailed Description

Functions and types related to ELF binaries.

10.27.2 Macro Definition Documentation

10.27.2.1 #define DF_1_CONFALT 0x00002000

Configuration alternative created.

Definition at line 541 of file elf.h.

10.27.2.2 #define DF_1_DIRECT 0x00000100

Direct binding enabled.

Definition at line 536 of file elf.h.

10.27.2.3 #define DF_1_DISPRELDNE 0x00008000

Disp reloc applied at build time.

Definition at line 543 of file elf.h.

10.27.2.4 #define DF_1_DISPRELPND 0x00010000

Disp reloc applied at run-time.

Definition at line 544 of file elf.h.

10.27.2.5 #define DF_1_ENDFILTEE 0x00004000

Filtee terminates filters search.

Definition at line 542 of file elf.h.

10.27.2.6 #define DF_1_GLOBAL 0x00000002

Set RTLD_GLOBAL for this object.

Definition at line 529 of file elf.h.

10.27.2.7 #define DF_1_GROUP 0x00000004

Set RTLD_GROUP for this object.

Definition at line 530 of file elf.h.

10.27.2.8 #define DF_1_INTERPOSE 0x00000400

Object is used to interpose.

Definition at line 538 of file elf.h.

10.27.2.9 #define DF_1_LOADFLTR 0x00000010

Trigger filtee loading at runtime.

Definition at line 532 of file elf.h.

10.27.2.10 #define DF_1_NODEFLIB 0x00000800

Ignore default lib search path.

Definition at line 539 of file elf.h.

10.27.2.11 #define DF_1_NODELETE 0x00000008

Set RTLD_NODELETE for this object.

Definition at line 531 of file elf.h.

10.27.2.12 #define DF_1_NODUMP 0x00001000

Object can't be dldump'ed.

Definition at line 540 of file elf.h.

10.27.2.13 #define DF_1_NOOPEN 0x00000040

Set RTLD_NOOPEN for this object.

Definition at line 534 of file elf.h.

10.27.2.14 #define DF_1_NOW 0x00000001

Set RTLD_NOW for this object.

Definition at line 528 of file elf.h.

10.27.2.15 #define DF_1_ORIGIN 0x00000080

$ORIGIN must be handled.

Definition at line 535 of file elf.h.

10.27.2.16 #define DF_P1_GROUPPERM 0x00000002

Symbols from next object are not generally available.

Definition at line 552 of file elf.h.

10.27.2.17 #define DF_P1_LAZYLOAD 0x00000001

Lazyload following object.

Definition at line 551 of file elf.h.

10.27.2.18 #define DT_HIPROC 0x7fffffff

processor spec.

Definition at line 517 of file elf.h.

10.27.2.19 #define DT_LOPROC 0x70000000

processor spec.

Definition at line 516 of file elf.h.

10.27.2.20 #define DT_NULL 0

Dynamic Array Tags, d_tag - figure 2-10, page 2-12.

end of _DYNAMIC array

Definition at line 479 of file elf.h.

10.27.2.21 #define EI_CLASS 4

ELF class byte index.

file class

Definition at line 254 of file elf.h.

10.27.2.22 #define EI_CLASS 4

ELF class byte index.

file class

Definition at line 254 of file elf.h.

10.27.2.23 #define EI_DATA 5

Data encoding byte index.

data encoding

Definition at line 255 of file elf.h.

10.27.2.24 #define EI_DATA 5

Data encoding byte index.

data encoding

Definition at line 255 of file elf.h.

10.27.2.25 #define EI_OSABI 7

OS ABI identification.

Operating system / ABI identification.

Definition at line 257 of file elf.h.

10.27.2.26 #define EI_OSABI 7

OS ABI identification.

Operating system / ABI identification.

Definition at line 257 of file elf.h.

10.27.2.27 #define EI_PAD 9

Byte index of padding bytes.

start of padding bytes

Definition at line 259 of file elf.h.

10.27.2.28 #define EI_PAD 9

Byte index of padding bytes.

start of padding bytes

Definition at line 259 of file elf.h.

10.27.2.29 #define EI_VERSION 6

File version byte index.

file version

Value must be EV_CURRENT

Definition at line 256 of file elf.h.

10.27.2.30 #define EI_VERSION 6

File version byte index.

file version

Value must be EV_CURRENT

Definition at line 256 of file elf.h.

10.27.2.31 #define ELFCLASSNONE 0

Invalid ELF class.

Invalid class.

Definition at line 270 of file elf.h.

10.27.2.32 #define ELFCLASSNONE 0

Invalid ELF class.

Invalid class.

Definition at line 270 of file elf.h.

10.27.2.33 #define ELFDATA2LSB 1

2's complement, little endian

0x01020304 => [0x04|0x03|0x02|0x01]

Definition at line 277 of file elf.h.

10.27.2.34 #define ELFDATA2LSB 1

2's complement, little endian

0x01020304 => [0x04|0x03|0x02|0x01]

Definition at line 277 of file elf.h.

10.27.2.35 #define ELFDATA2MSB 2

2's complement, big endian

0x01020304 => [0x01|0x02|0x03|0x04]

Definition at line 278 of file elf.h.

10.27.2.36 #define ELFDATA2MSB 2

2's complement, big endian

0x01020304 => [0x01|0x02|0x03|0x04]

Definition at line 278 of file elf.h.

10.27.2.37 #define ELFDATANONE 0

Invalid data encoding.

invalid data encoding

Definition at line 276 of file elf.h.

10.27.2.38 #define ELFDATANONE 0

Invalid data encoding.

invalid data encoding

Definition at line 276 of file elf.h.

10.27.2.39 #define ELFOSABI_AIX 7

IBM AIX.

Definition at line 177 of file elf.h.

10.27.2.40 #define ELFOSABI_FREEBSD 9

FreeBSD.

Definition at line 179 of file elf.h.

10.27.2.41 #define ELFOSABI_HPUX 1

HP-UX.

HP-UX operating system.

Definition at line 283 of file elf.h.

10.27.2.42 #define ELFOSABI_HPUX 1

HP-UX.

HP-UX operating system.

Definition at line 283 of file elf.h.

10.27.2.43 #define ELFOSABI_IRIX 8

SGI Irix.

Definition at line 178 of file elf.h.

10.27.2.44 #define ELFOSABI_LINUX 3

Linux.

Definition at line 175 of file elf.h.

10.27.2.45 #define ELFOSABI_MODESTO 11

Novell Modesto.

Definition at line 181 of file elf.h.

10.27.2.46 #define ELFOSABI_NETBSD 2

NetBSD.

Definition at line 174 of file elf.h.

10.27.2.47 #define ELFOSABI_OPENBSD 12

OpenBSD.

Definition at line 182 of file elf.h.

10.27.2.48 #define ELFOSABI_SOLARIS 6

Sun Solaris.

Definition at line 176 of file elf.h.

10.27.2.49 #define ELFOSABI_SYSV 0

Alias.

UNIX System V ABI (this specification)

Definition at line 282 of file elf.h.

10.27.2.50 #define ELFOSABI_SYSV 0

Alias.

UNIX System V ABI (this specification)

Definition at line 282 of file elf.h.

10.27.2.51 #define ELFOSABI_TRU64 10

Compaq TRU64 UNIX.

Definition at line 180 of file elf.h.

10.27.2.52 #define EM_ARC 45

Argonaut RISC Core, Argonaut Techn Inc.

Definition at line 226 of file elf.h.

10.27.2.53 #define NT_VERSION 1

Contains a version string.

Definition at line 455 of file elf.h.

10.27.2.54 #define PT_GNU_EH_FRAME (PT_LOOS + 0x474e550)

EH frame information.

Definition at line 418 of file elf.h.

10.27.2.55 #define PT_GNU_RELRO (PT_LOOS + 0x474e552)

Read only after reloc.

Definition at line 420 of file elf.h.

10.27.2.56 #define PT_GNU_STACK (PT_LOOS + 0x474e551)

Flags for stack.

Definition at line 419 of file elf.h.

10.27.2.57 #define PT_HIOS 0x6fffffff

os spec.

Definition at line 414 of file elf.h.

10.27.2.58 #define PT_HIPROC 0x7fffffff

processor spec.

Definition at line 416 of file elf.h.

10.27.2.59 #define PT_L4_AUX (PT_LOOS + 0x14)

Address of the AUX strcutures.

Definition at line 424 of file elf.h.

10.27.2.60 #define PT_L4_KIP (PT_LOOS + 0x13)

Address of the KIP.

Definition at line 423 of file elf.h.

10.27.2.61 #define PT_L4_STACK (PT_LOOS + 0x12)

Address of the stack.

Definition at line 422 of file elf.h.

10.27.2.62 #define PT_LOOS 0x60000000

os spec.

Definition at line 413 of file elf.h.

10.27.2.63 #define PT_LOPROC 0x70000000

processor spec.

Definition at line 415 of file elf.h.

10.27.2.64 #define SHF_GROUP 0x200

Section is member of a group.

Definition at line 368 of file elf.h.

10.27.2.65 #define SHF_MASKOS 0x0ff00000

OS-specific.

Definition at line 370 of file elf.h.

10.27.2.66 #define SHF_TLS 0x400

Section hold thread-local data.

Definition at line 369 of file elf.h.

10.27.2.67 #define SHT_NUM 19

Number of defined types.

Definition at line 348 of file elf.h.

10.28 Error Handling

Error handling for L4 object invocation.

Collaboration diagram for Error Handling:

Enumerations

- enum l4_ipc_tcr_error_t {
  L4_IPC_ERROR_MASK = 0x1F, L4_IPC_SND_ERR_MASK = 0x01, L4_IPC_ENOT_EXISTENT = 0x04, L4_IPC_RETIMEOUT = 0x03,
  L4_IPC_SETIMEOUT = 0x02, L4_IPC_RECANCELED = 0x07, L4_IPC_SECANCELED = 0x06, L4_IPC_REMAPFAILED = 0x11,
  L4_IPC_SEMAPFAILED = 0x10, L4_IPC_RESNDPFTO = 0x0b, L4_IPC_SESNDPFTO = 0x0a, L4_IPC_RERCVPFTO = 0x0d,
  L4_IPC_SERCVPFTO = 0x0c, L4_IPC_REABORTED = 0x0f, L4_IPC_SEABORTED = 0x0e, L4_IPC_REMSGCUT = 0x09,
  L4_IPC_SEMSGCUT = 0x08 }

  Error codes in the error TCR.

Functions

- l4_umword_t l4_ipc_error (l4_msgtag_t tag, l4_utcb_t *utcb) L4_NOTHROW

  Get the error code for an object invocation.
- long l4_error (l4_msgtag_t tag) L4_NOTHROW

  Return error code of a system call return message tag.
- int l4_ipc_is_snd_error (l4_utcb_t *utcb) L4_NOTHROW

  Returns whether an error occurred in send phase of an invocation.
- int l4_ipc_is_rcv_error (l4_utcb_t *utcb) L4_NOTHROW

  Returns whether an error occurred in receive phase of an invocation.
- int l4_ipc_error_code (l4_utcb_t *utcb) L4_NOTHROW

  Get the error condition of the last invocation from the TCR.

10.28.1 Detailed Description

Error handling for L4 object invocation.

```
#include <l4/sys/ipc.h>
```

10.28.2 Enumeration Type Documentation

10.28.2.1 enum l4_ipc_tcr_error_t

Error codes in the *error* TCR.

The error codes are accessible via the *error* TCR, see l4_thread_regs_t.error.

Enumerator

L4_IPC_ERROR_MASK Mask for error bits.

L4_IPC_SND_ERR_MASK Send error mask.

L4_IPC_ENOT_EXISTENT Non-existing destination or source.

L4_IPC_RETIMEOUT Timeout during receive operation.

L4_IPC_SETIMEOUT Timeout during send operation.

L4_IPC_RECANCELED Receive operation canceled.

L4_IPC_SECANCELED Send operation canceled.

L4_IPC_REMAPFAILED Map flexpage failed in receive operation.

L4_IPC_SEMAPFAILED Map flexpage failed in send operation.

L4_IPC_RESNDPFTO Send-pagefault timeout in receive operation.

L4_IPC_SESNDPFTO Send-pagefault timeout in send operation.

L4_IPC_RERCVPFTO Receive-pagefault timeout in receive operation.

L4_IPC_SERCVPFTO Receive-pagefault timeout in send operation.

L4_IPC_REABORTED Receive operation aborted.

L4_IPC_SEABORTED Send operation aborted.

L4_IPC_REMSGCUT Cut receive message, due to message buffer is too small.

L4_IPC_SEMSGCUT Cut send message. due to message buffer is too small,

Definition at line 75 of file ipc.h.

10.28.3 Function Documentation

10.28.3.1 long l4_error (l4_msgtag_t *tag*) [inline]

Return error code of a system call return message tag.

Parameters

tag	System call return message type

Returns

0 for no error, error number in case of error

Examples:

examples/libs/l4re/streammap/client.cc, examples/sys/aliens/main.c, examples/sys/isr/main.c, examples/sys/migrate/thread↩_migrate.cc, examples/sys/singlestep/main.c, examples/sys/start-with-exc/main.c, and examples/sys/utcb-ipc/main.c.

Definition at line 513 of file ipc.h.

References l4_utcb().

Referenced by L4Re::Mem_alloc::alloc(), L4::cap_dynamic_cast(), L4Re::Util::Event_t< PAYLOAD >::init(), l4_↩vcon_write_u(), and L4Re::Util::Event_buffer_consumer_t< PAYLOAD >::process().

Here is the call graph for this function:

Here is the caller graph for this function:

10.28.3.2 l4_umword_t l4_ipc_error (l4_msgtag_t *tag*, l4_utcb_t * *utcb*) [inline]

Get the error code for an object invocation.

Parameters

tag	Return value of the invocation.
utcb	UTCB that was used for the invocation.

Returns

0 if no error condition is set, error code otherwise (see l4_ipc_tcr_error_t).

Examples:

examples/sys/ipc/ipc_example.c, examples/sys/isr/main.c, and examples/sys/start-with-exc/main.c.

Definition at line 496 of file ipc.h.

References l4_thread_regs_t::error, L4_IPC_ERROR_MASK, and l4_msgtag_has_error().

Referenced by L4Re::Util::Event_buffer_consumer_t< PAYLOAD >::process().

Here is the call graph for this function:

Here is the caller graph for this function:

10.28.3.3 int l4_ipc_error_code (l4_utcb_t * *utcb*) [inline]

Get the error condition of the last invocation from the TCR.

Precondition

l4_msgtag_has_error(tag) == true

Parameters

utcb	UTCB to check.

Returns

Error condition of type l4_ipc_tcr_error_t.

Definition at line 525 of file ipc.h.

References l4_thread_regs_t::error, and L4_IPC_ERROR_MASK.

10.28.3.4 int l4_ipc_is_rcv_error (l4_utcb_t * *utcb*) [inline]

Returns whether an error occurred in receive phase of an invocation.

Precondition

l4_msgtag_has_error(tag) == true

Parameters

utcb	UTCB to check.

Returns

Boolean value.

Definition at line 522 of file ipc.h.

References l4_thread_regs_t::error.

10.28.3.5 int l4_ipc_is_snd_error (l4_utcb_t * *utcb*) [inline]

Returns whether an error occurred in send phase of an invocation.

Precondition

l4_msgtag_has_error(tag) == true

Parameters

utcb	UTCB to check.

Returns

Boolean value.

Definition at line 519 of file ipc.h.

10.29 Error codes

Common error codes.

Collaboration diagram for Error codes:

Enumerations

- enum l4_error_code_t {
  L4_EOK = 0, L4_EPERM = 1, L4_ENOENT = 2, L4_EIO = 5,
  L4_EAGAIN = 11, L4_ENOMEM = 12, L4_EACCESS = 13, L4_EBUSY = 16,
  L4_EEXIST = 17, L4_ENODEV = 19, L4_EINVAL = 22, L4_ERANGE = 34,
  L4_ENAMETOOLONG = 36, L4_ENOSYS = 38, L4_EBADPROTO = 39, L4_EADDRNOTAVAIL = 99,
  L4_ERRNOMAX = 100, L4_ENOREPLY = 1000, L4_EMSGTOOSHORT = 1001, L4_EMSGTOOLONG = 1002,
  L4_EMSGMISSARG = 1003, L4_EIPC_LO = 2000, L4_EIPC_HI = 2000 + 0x1f }

  L4 error codes.

10.29.1 Detailed Description

Common error codes.

```
#include <l4/sys/err.h>
```

10.29.2 Enumeration Type Documentation

10.29.2.1 enum l4_error_code_t

L4 error codes.

Those error codes are used by both the kernel and the user programs.

Enumerator

L4_EOK Ok.

L4_EPERM No permission.

L4_ENOENT No such entity.

L4_EIO I/O error.

L4_EAGAIN Try again.

L4_ENOMEM No memory.

L4_EACCESS Permission denied.

L4_EBUSY Object currently busy, try later.

L4_EEXIST Already exists.

L4_ENODEV No such thing.

L4_EINVAL Invalid argument.

L4_ERANGE Range error.

L4_ENAMETOOLONG Name too long.

L4_ENOSYS No sys.

L4_EBADPROTO Unsupported protocol.

L4_EADDRNOTAVAIL Address not available.

L4_ERRNOMAX Maximum error value.

L4_ENOREPLY No reply.

L4_EMSGTOOSHORT Message too short.

L4_EMSGTOOLONG Message too long.

L4_EMSGMISSARG Message has invalid capability.

L4_EIPC_LO Communication error-range low.

L4_EIPC_HI Communication error-range high.

Definition at line 41 of file err.h.

10.30 Event API

Event interface.

Collaboration diagram for Event API:

Data Structures

- class L4Re::Event

  Event class.
- class L4Re::Event_buffer_t< PAYLOAD >

  Event buffer class.

10.30.1 Detailed Description

Event interface.

10.31 Event interface

Event C interface.

Collaboration diagram for Event interface:

Functions

- long l4re_event_get_buffer (const l4_cap_idx_t server, const l4re_ds_t ds) L4_NOTHROW

  Get an event signal buffer.
- long l4re_event_get_num_streams (const l4_cap_idx_t server) L4_NOTHROW

  Get number of streams.
- long l4re_event_get_stream_info (const l4_cap_idx_t server, int idx, l4re_event_stream_info_t ∗info) L4_N↩ OTHROW

  Get information on a stream.
- long l4re_event_get_stream_info_for_id (const l4_cap_idx_t server, l4_umword_t stream_id, l4re_event_↩ stream_info_t ∗info) L4_NOTHROW

  Get info for a stream given a stream id.
- long l4re_event_get_axis_info (const l4_cap_idx_t server, l4_umword_t id, unsigned naxes, unsigned const ∗axis, l4re_event_absinfo_t ∗info) L4_NOTHROW

  Get Axis information for a stream.

10.31.1 Detailed Description

Event C interface.

10.31.2 Function Documentation

10.31.2.1 long l4re_event_get_axis_info (const l4_cap_idx_t *server*, l4_umword_t *id*, unsigned *naxes*, unsigned const ∗ *axis*, l4re_event_absinfo_t ∗ *info*)

Get Axis information for a stream.

Parameters

	server	Server to talk to.
	id	Id of the stream to get information from.
	naxes	Number of axes in `axis` array.
`in`	*axis*	Array of axis IDs whose information should be retrieved.
`out`	*info*	Information buffer to store the retrieved axis infos.

Return values

0	Success
<0	Error

See also

L4Re::Event::get_axis_info

10.31.2.2 long l4re_event_get_buffer (const l4_cap_idx_t *server*, const l4re_ds_t *ds*)

Get an event signal buffer.

Parameters

server	Server to talk to.
ds	Buffer to event data.

Returns

0 for success, <0 on error

See also

L4Re::Event::get_buffer

10.31.2.3 long l4re_event_get_num_streams (const l4_cap_idx_t *server*)

Get number of streams.

Parameters

server	Server to talk to.

Returns

0 for success, <0 on error

See also

L4Re::Event::get_num_streams

10.31.2.4 long l4re_event_get_stream_info (const l4_cap_idx_t *server*, int *idx*, l4re_event_stream_info_t * *info*)

Get information on a stream.

Parameters

server	Server to talk to.
idx	Index value.

Return values

info	Information buffer.

Returns

0 for success, <0 on error

See also

L4Re::Event::get_stream_info

10.31.2.5 long l4re_event_get_stream_info_for_id (const l4_cap_idx_t *server*, l4_umword_t *stream_id*, l4re_event_stream_info_t * *info*)

Get info for a stream given a stream id.

Parameters

server	Server to talk to.
stream_id	Stream ID.

Return values

info	Information buffer.

Returns

0 for success, <0 on error

See also

L4Re::Event::get_stream_info_for_id

10.32 Exception registers

Overly definition of the MRs for exception messages.

Collaboration diagram for Exception registers:

Functions

- l4_exc_regs_t * l4_utcb_exc (void) L4_NOTHROW L4_PURE

  Get the message-register block of a UTCB (for an exception IPC).
- l4_umword_t l4_utcb_exc_pc (l4_exc_regs_t *u) L4_NOTHROW L4_PURE

  Access function to get the program counter of the exception state.
- void l4_utcb_exc_pc_set (l4_exc_regs_t *u, l4_addr_t pc) L4_NOTHROW

  Set the program counter register in the exception state.
- unsigned long l4_utcb_exc_typeval (l4_exc_regs_t *u) L4_NOTHROW L4_PURE

  Get the value out of an exception UTCB that describes the type of exception.
- int l4_utcb_exc_is_pf (l4_exc_regs_t *u) L4_NOTHROW L4_PURE

  Check whether an exception IPC is a page fault.
- l4_addr_t l4_utcb_exc_pfa (l4_exc_regs_t *u) L4_NOTHROW L4_PURE

  Function to get the L4 style page fault address out of an exception.

10.32.1 Detailed Description

Overly definition of the MRs for exception messages.

10.32.2 Function Documentation

10.32.2.1 l4_exc_regs_t * l4_utcb_exc (void) [inline]

Get the message-register block of a UTCB (for an exception IPC).

Returns

A pointer to the exception message in `u`.

Examples:

examples/sys/aliens/main.c, and examples/sys/singlestep/main.c.

Definition at line 351 of file utcb.h.

References l4_utcb().

Here is the call graph for this function:

10.32.2.2 int l4_utcb_exc_is_pf (l4_exc_regs_t * u) [inline]

Check whether an exception IPC is a page fault.

Returns

0 if not, != 0 if yes

Function to check whether an exception IPC is a page fault, also applies to I/O pagefaults.

Definition at line 105 of file utcb.h.

10.32.2.3 l4_umword_t l4_utcb_exc_pc (l4_exc_regs_t * u) [inline]

Access function to get the program counter of the exception state.

Parameters

u	UTCB

Returns

The program counter register out of the exception state.

Examples:

examples/sys/aliens/main.c, and examples/sys/singlestep/main.c.

Definition at line 90 of file utcb.h.

10.32.2.4 void l4_utcb_exc_pc_set (l4_exc_regs_t * u, l4_addr_t pc) [inline]

Set the program counter register in the exception state.

Parameters

u	UTCB
pc	The program counter to set.

Definition at line 95 of file utcb.h.

10.33 Extended vCPU support

extended vCPU handling functionality.

Collaboration diagram for Extended vCPU support:

Functions

- int l4vcpu_ext_alloc (l4_vcpu_state_t **vcpu, l4_addr_t *ext_state, l4_cap_idx_t task, l4_cap_idx_t regmgr) L4_NOTHROW

  Allocate state area for an extended vCPU.

10.33.1 Detailed Description

extended vCPU handling functionality.

10.33.2 Function Documentation

10.33.2.1 int l4vcpu_ext_alloc (l4_vcpu_state_t ** *vcpu*, l4_addr_t * *ext_state*, l4_cap_idx_t *task*, l4_cap_idx_t *regmgr*)

Allocate state area for an extended vCPU.

Return values

vcpu	Allocated vcpu-state area.
ext_state	Allocated extended vcpu-state area.

Parameters

task	Task to use for allocation.
regmgr	Region manager to use for allocation.

Returns

0 for success, error code otherwise

10.34 Factory

C factory interface to create kernel objects.

Collaboration diagram for Factory:

Functions

- l4_msgtag_t l4_factory_create_task (l4_cap_idx_t factory, l4_cap_idx_t target_cap, l4_fpage_t const utcb_area) L4_NOTHROW

  Create a new task.
- l4_msgtag_t l4_factory_create_task_u (l4_cap_idx_t factory, l4_cap_idx_t target_cap, l4_fpage_t const utcb_area, l4_utcb_t *utcb) L4_NOTHROW

  Create a new task.
- l4_msgtag_t l4_factory_create_thread (l4_cap_idx_t factory, l4_cap_idx_t target_cap) L4_NOTHROW

  Create a new thread.
- l4_msgtag_t l4_factory_create_thread_u (l4_cap_idx_t factory, l4_cap_idx_t target_cap, l4_utcb_t *utcb) L4_NOTHROW

  Create a new thread.
- l4_msgtag_t l4_factory_create_factory (l4_cap_idx_t factory, l4_cap_idx_t target_cap, unsigned long limit) L4_NOTHROW

  Create a new factory.
- l4_msgtag_t l4_factory_create_factory_u (l4_cap_idx_t factory, l4_cap_idx_t target_cap, unsigned long limit, l4_utcb_t *utcb) L4_NOTHROW

  Create a new factory.
- l4_msgtag_t l4_factory_create_gate (l4_cap_idx_t factory, l4_cap_idx_t target_cap, l4_cap_idx_t thread_cap, l4_umword_t label) L4_NOTHROW

  Create a new IPC gate.
- l4_msgtag_t l4_factory_create_gate_u (l4_cap_idx_t factory, l4_cap_idx_t target_cap, l4_cap_idx_t thread_cap, l4_umword_t label, l4_utcb_t *utcb) L4_NOTHROW

  Create a new IPC gate.
- l4_msgtag_t l4_factory_create_irq (l4_cap_idx_t factory, l4_cap_idx_t target_cap) L4_NOTHROW

  Create a new IRQ sender.
- l4_msgtag_t l4_factory_create_irq_u (l4_cap_idx_t factory, l4_cap_idx_t target_cap, l4_utcb_t *utcb) L4_NOTHROW

  Create a new IRQ sender.
- l4_msgtag_t l4_factory_create_vm (l4_cap_idx_t factory, l4_cap_idx_t target_cap) L4_NOTHROW

  Create a new virtual machine.
- l4_msgtag_t l4_factory_create_vm_u (l4_cap_idx_t factory, l4_cap_idx_t target_cap, l4_utcb_t *utcb) L4_NOTHROW

  Create a new virtual machine.

10.34.1 Detailed Description

C factory interface to create kernel objects.

Add

```
#include <l4/sys/factory.h>
```

to your code to use the factory interface. A factory is used to create all kinds of kernel objects:

- Task
- Thread
- Factory
- IPC-Gate API
- IRQs
- Virtual Machines

The factory is equipped with a limit that limits the amount of kernel memory available for that factory.

Note

The limit does not give any guarantee for the amount of available kernel memory.

For the C++ interface refer to L4::Factory.

10.34.2 Function Documentation

10.34.2.1 l4_msgtag_t l4_factory_create_factory (l4_cap_idx_t *factory*, l4_cap_idx_t *target_cap*, unsigned long *limit*) [inline]

Create a new factory.

Parameters

	factory	Capability selector for factory to use for creation.
out	*target_cap*	Capability selector for the root capability of the new factory. The caller must allocate the capability slot. The kernel stores the new factory's capability into this slot.
	limit	Limit for the new factory in bytes.
	utcb	The UTCB to use for the operation.

Returns

Syscall return tag

Note

The limit of the new factory is subtracted from the available amount of the factory used for creation.

Definition at line 316 of file factory.h.

References l4_factory_create_factory_u(), and l4_utcb().

Here is the call graph for this function:

10.34.2.2 l4_msgtag_t l4_factory_create_factory_u (l4_cap_idx_t *factory*, l4_cap_idx_t *target_cap*, unsigned long *limit*, l4_utcb_t * *utcb*) [inline]

Create a new factory.

Parameters

	factory	Capability selector for factory to use for creation.
out	*target_cap*	Capability selector for the root capability of the new factory. The caller must allocate the capability slot. The kernel stores the new factory's capability into this slot.
	limit	Limit for the new factory in bytes.
	utcb	The UTCB to use for the operation.

Returns

Syscall return tag

Note

The limit of the new factory is subtracted from the available amount of the factory used for creation.

Definition at line 250 of file factory.h.

References L4_PROTO_FACTORY.

Referenced by L4::Factory::create_factory(), and l4_factory_create_factory().

Here is the caller graph for this function:

10.34.2.3 l4_msgtag_t l4_factory_create_gate (l4_cap_idx_t *factory,* l4_cap_idx_t *target_cap,* l4_cap_idx_t *thread_cap,* l4_umword_t *label*) [inline]

Create a new IPC gate.

Parameters

	factory	Capability selector for factory to use for creation.
out	*target_cap*	Capability selector for the root capability of the new IPC gate. The caller must allocate the capability slot. The kernel stores the new IPC gate's capability into this slot.
	thread_cap	Capability selector of the thread to bind the gate to.
	label	Label of the gate.
	utcb	The UTCB to use for the operation.

Returns

Syscall return tag

See also

L4::Ipc_gate
IPC-Gate API

Definition at line 324 of file factory.h.

References l4_factory_create_gate_u(), and l4_utcb().

Here is the call graph for this function:

10.34.2.4 l4_msgtag_t l4_factory_create_gate_u (l4_cap_idx_t *factory*, l4_cap_idx_t *target_cap*, l4_cap_idx_t *thread_cap*, l4_umword_t *label*, l4_utcb_t * *utcb*) [inline]

Create a new IPC gate.

Parameters

	factory	Capability selector for factory to use for creation.
out	*target_cap*	Capability selector for the root capability of the new IPC gate. The caller must allocate the capability slot. The kernel stores the new IPC gate's capability into this slot.
	thread_cap	Capability selector of the thread to bind the gate to.
	label	Label of the gate.
	utcb	The UTCB to use for the operation.

Returns

Syscall return tag

See also

L4::Ipc_gate
IPC-Gate API

Definition at line 261 of file factory.h.

References L4_FPAGE_RWX, l4_map_obj_control(), l4_msgtag(), l4_msgtag_label(), l4_msgtag_words(), l4_obj↩_fpage(), l4_msg_regs_t::mr, and l4_fpage_t::raw.

Referenced by L4::Factory::create_gate(), and l4_factory_create_gate().

Here is the call graph for this function:

Here is the caller graph for this function:

10.34.2.5 l4_msgtag_t l4_factory_create_irq (l4_cap_idx_t *factory*, l4_cap_idx_t *target_cap*) [inline]

Create a new IRQ sender.

Parameters

	factory	Factory to use for creation.
out	*target_cap*	Slot to receive the capability for the new IRQ sender.

Returns

Syscall return tag

See also

IRQs

Examples:

examples/sys/isr/main.c.

Definition at line 332 of file factory.h.

References l4_factory_create_irq_u(), and l4_utcb().

Here is the call graph for this function:

10.34.2.6 l4_msgtag_t l4_factory_create_irq_u (l4_cap_idx_t *factory*, l4_cap_idx_t *target_cap*, l4_utcb_t * *utcb*) [inline]

Create a new IRQ sender.

Parameters

	factory	Factory to use for creation.
out	*target_cap*	Slot to receive the capability for the new IRQ sender.

Returns

Syscall return tag

See also

IRQs

Definition at line 283 of file factory.h.

References L4_PROTO_IRQ_SENDER.

Referenced by L4::Factory::create_irq(), and l4_factory_create_irq().

Here is the caller graph for this function:

10.34.2.7 l4_msgtag_t l4_factory_create_task (l4_cap_idx_t *factory*, l4_cap_idx_t *target_cap*, l4_fpage_t const *utcb_area*) [inline]

Create a new task.

Parameters

	factory	Capability selector for factory to use for creation.
out	*target_cap*	Capability selector for the root capability of the new task. The caller must allocate the capability slot. The kernel stores the new task's capability into this slot.
	utcb_area	Flexpage that describes the area used for UTCBs and vCPU state-save-areas of the new task.
	utcb	The UTCB to use for the operation.

Returns

Syscall return tag

Note

The size of the UTCB area specifies indirectly the number of UTCBs available for this task. Refer to L4::Task::add_ku_mem / l4_task_add_ku_mem() for adding more of this type of memory.

See also

L4::Task
Task

Definition at line 302 of file factory.h.

References l4_factory_create_task_u(), and l4_utcb().

Here is the call graph for this function:

10.34.2.8 l4_msgtag_t l4_factory_create_task_u (l4_cap_idx_t *factory*, l4_cap_idx_t *target_cap*, l4_fpage_t const *utcb_area*, l4_utcb_t * *utcb*) [inline]

Create a new task.

Parameters

	factory	Capability selector for factory to use for creation.
out	*target_cap*	Capability selector for the root capability of the new task. The caller must allocate the capability slot. The kernel stores the new task's capability into this slot.
	utcb_area	Flexpage that describes the area used for UTCBs and vCPU state-save-areas of the new task.
	utcb	The UTCB to use for the operation.

Returns

Syscall return tag

Note

The size of the UTCB area specifies indirectly the number of UTCBs available for this task. Refer to L4::Task::add_ku_mem / l4_task_add_ku_mem() for adding more of this type of memory.

See also

L4::Task
Task

Definition at line 232 of file factory.h.

References L4_PROTO_TASK.

Referenced by L4::Factory::create_task(), and l4_factory_create_task().

Here is the caller graph for this function:

10.34.2.9 l4_msgtag_t l4_factory_create_thread (l4_cap_idx_t *factory*, l4_cap_idx_t *target_cap*) [inline]

Create a new thread.

Parameters

	factory	Capability selector for factory to use for creation.
out	*target_cap*	Capability selector for the root capability of the new thread. The caller must allocate the capability slot. The kernel stores the new thread's capability into this slot.
	utcb	The UTCB to use for the operation.

Returns

Syscall return tag

See also

L4::Thread
Thread

Examples:

examples/sys/aliens/main.c, examples/sys/singlestep/main.c, examples/sys/start-with-exc/main.c, and examples/sys/utcb-ipc/main.c.

Definition at line 309 of file factory.h.

References l4_factory_create_thread_u(), and l4_utcb().

Here is the call graph for this function:

10.34.2.10 l4_msgtag_t l4_factory_create_thread_u (l4_cap_idx_t *factory*, l4_cap_idx_t *target_cap*, l4_utcb_t * *utcb*) [inline]

Create a new thread.

Parameters

	factory	Capability selector for factory to use for creation.
out	*target_cap*	Capability selector for the root capability of the new thread. The caller must allocate the capability slot. The kernel stores the new thread's capability into this slot.
	utcb	The UTCB to use for the operation.

Returns

Syscall return tag

See also

L4::Thread
Thread

Definition at line 243 of file factory.h.

References L4_PROTO_THREAD.

Referenced by L4::Factory::create_thread(), and l4_factory_create_thread().

Here is the caller graph for this function:

10.34.2.11 l4_msgtag_t l4_factory_create_vm (l4_cap_idx_t *factory,* l4_cap_idx_t *target_cap*) [inline]

Create a new virtual machine.

Parameters

	factory	Capability selector for factory to use for creation.
out	*target_cap*	Capability selector for the root capability of the new VM. The caller must allocate the capability slot. The kernel stores the new VM's capability into this slot.
	utcb	The UTCB to use for the operation.

Returns

Syscall return tag

See also

L4::Vm
Virtual Machines

Definition at line 339 of file factory.h.

References l4_factory_create_vm_u(), and l4_utcb().

Here is the call graph for this function:

10.34.2.12 l4_msgtag_t l4_factory_create_vm_u (l4_cap_idx_t *factory*, l4_cap_idx_t *target_cap*, l4_utcb_t * *utcb*) [inline]

Create a new virtual machine.

Parameters

	factory	Capability selector for factory to use for creation.
out	*target_cap*	Capability selector for the root capability of the new VM. The caller must allocate the capability slot. The kernel stores the new VM's capability into this slot.
	utcb	The UTCB to use for the operation.

Returns

Syscall return tag

See also

L4::Vm
Virtual Machines

Definition at line 290 of file factory.h.

References L4_PROTO_VM.

Referenced by L4::Factory::create_vm(), and l4_factory_create_vm().

Here is the caller graph for this function:

10.35 Fiasco extensions

Kernel debugger extensions of the Fiasco L4 implementation.

Collaboration diagram for Fiasco extensions:

Modules

- Fiasco real time scheduling extensions

  Real time scheduling extension for the Fiasco L4 implementation.
- Kernel Debugger

  Kernel debugger related functionality.

Files

- file segment.h

  l4f specific fs/gs manipulation
- file segment.h

  l4f specific segment manipulation

Data Structures

- struct l4_tracebuffer_status_window_t

  Trace-buffer status window descriptor.
- struct l4_tracebuffer_status_t

  Trace-buffer status.

Functions

- l4_tracebuffer_status_t ∗ fiasco_tbuf_get_status (void)

  Return trace-buffer status.
- l4_addr_t fiasco_tbuf_get_status_phys (void)

  Return the physical address of the trace-buffer status struct.
- l4_umword_t fiasco_tbuf_log (const char ∗text)

  Create new trace-buffer entry with describing <text>.
- l4_umword_t fiasco_tbuf_log_3val (const char ∗text, l4_umword_t v1, l4_umword_t v2, l4_umword_t v3)

  Create new trace-buffer entry with describing <text> and three additional values.
- l4_umword_t fiasco_tbuf_log_binary (const unsigned char ∗data)

  Create new trace-buffer entry with binary data.
- void fiasco_tbuf_clear (void)

Clear trace-buffer.

- void fiasco_tbuf_dump (void)

  Dump trace-buffer to kernel console.

- void fiasco_profile_start (void) L4_NOTHROW

  Start profiling.

- void fiasco_profile_stop_and_dump (void) L4_NOTHROW

  Stop profiling and dump result to console.

- void fiasco_profile_stop (void) L4_NOTHROW

  Stop profiling.

- void fiasco_watchdog_enable (void) L4_NOTHROW

  Enable Fiasco watchdog.

- void fiasco_watchdog_disable (void) L4_NOTHROW

  Disable Fiasco watchdog.

- void fiasco_watchdog_takeover (void) L4_NOTHROW

  Disable automatic resetting of watchdog.

- void fiasco_watchdog_giveback (void) L4_NOTHROW

  Reenable automatic resetting of watchdog.

- void fiasco_watchdog_touch (void) L4_NOTHROW

  Reset watchdog from user land.

- long fiasco_ldt_set (l4_cap_idx_t task, void *ldt, unsigned int num_desc, unsigned int entry_number_start, l4_utcb_t *utcb)

  Set LDT segments descriptors.

- long fiasco_gdt_set (l4_cap_idx_t thread, void *desc, unsigned int size, unsigned int entry_number_start, l4_utcb_t *utcb)

  Set GDT segment descriptors.

- unsigned fiasco_gdt_get_entry_offset (l4_cap_idx_t thread, l4_utcb_t *utcb)

  Return the offset of the entry in the GDT.

10.35.1 Detailed Description

Kernel debugger extensions of the Fiasco L4 implementation.

10.35.2 Function Documentation

10.35.2.1 unsigned fiasco_gdt_get_entry_offset (l4_cap_idx_t *thread*, l4_utcb_t * *utcb*) [inline]

Return the offset of the entry in the GDT.

Parameters

thread	Thread to get info from.
utcb	UTCB of the caller.

Definition at line 136 of file segment.h.

References l4_ipc_call(), L4_IPC_NEVER, l4_msgtag(), L4_PROTO_THREAD, L4_THREAD_X86_GDT_OP, and l4_msg_regs_t::mr.

Here is the call graph for this function:

10.35.2.2 long fiasco_gdt_set (l4_cap_idx_t *thread*, void * *desc*, unsigned int *size*, unsigned int *entry_number_start*, l4_utcb_t * *utcb*) [inline]

Set GDT segment descriptors.

Fiasco supports 3 consecutive entries, starting at the value returned by fiasco_gdt_get_entry_offset()

Parameters

thread	Thread to set the GDT entry for.
desc	Pointer to GDT descriptors.
size	Size of the descriptors in bytes (multiple of 8).
entry_number_start	Entry number to start (valid values: 0-2).
utcb	UTCB of the caller.

Returns

System call error

Definition at line 52 of file segment.h.

References l4_ipc_call(), L4_IPC_NEVER, l4_msgtag(), L4_PROTO_THREAD, L4_THREAD_X86_GDT_OP, and l4_msg_regs_t::mr.

Here is the call graph for this function:

10.35.2.3 long fiasco_ldt_set (l4_cap_idx_t *task*, void * *ldt*, unsigned int *num_desc*, unsigned int *entry_number_start*, l4_utcb_t * *utcb*) [inline]

Set LDT segments descriptors.

Parameters

task	Task to set the segment for.
ldt	Pointer to LDT hardware descriptors.
num_desc	Number of descriptors.
entry_number↩ _start	Entry number to start.
utcb	UTCB of the caller.

Definition at line 123 of file segment.h.

References L4_EINVAL, l4_ipc_call(), L4_IPC_NEVER, l4_msgtag(), L4_PROTO_TASK, L4_TASK_LDT_SET_↩ X86_OP, and l4_msg_regs_t::mr.

Here is the call graph for this function:

10.35.2.4 l4_tracebuffer_status_t* fiasco_tbuf_get_status (void) [inline]

Return trace-buffer status.

Returns

Pointer to trace-buffer status struct.

Definition at line 33 of file ktrace-arch.h.

10.35.2.5 l4_addr_t fiasco_tbuf_get_status_phys (void) [inline]

Return the physical address of the trace-buffer status struct.

Returns

physical address of status struct.

Definition at line 40 of file ktrace-arch.h.

References enter_kdebug.

10.35.2.6 l4_umword_t fiasco_tbuf_log (const char * *text*) [inline]

Create new trace-buffer entry with describing <text>.

Parameters

text	Logging text

Returns

Pointer to trace-buffer entry

Definition at line 47 of file ktrace-arch.h.

10.35.2.7 l4_umword_t fiasco_tbuf_log_3val (const char * *text*, l4_umword_t *v1*, l4_umword_t *v2*, l4_umword_t *v3*) [inline]

Create new trace-buffer entry with describing <text> and three additional values.

Parameters

text	Logging text
v1	first value
v2	second value
v3	third value

Returns

Pointer to trace-buffer entry

Definition at line 53 of file ktrace-arch.h.

10.35.2.8 l4_umword_t fiasco_tbuf_log_binary (const unsigned char * *data*) [inline]

Create new trace-buffer entry with binary data.

Parameters

data	binary data

Returns

Pointer to trace-buffer entry

Definition at line 83 of file ktrace-arch.h.

10.35.2.9 void fiasco_watchdog_takeover (void) [inline]

Disable automatic resetting of watchdog.

User is responsible to call `fiasco_watchdog_touch` from time to time to ensure that the watchdog does not trigger.

Definition at line 407 of file kdebug.h.

10.35.2.10 void fiasco_watchdog_touch (void) [inline]

Reset watchdog from user land.

This function **must** be called from time to time to prevent the watchdog from triggering if the watchdog is activated and if `fiasco_watchdog_takeover` was performed.

Definition at line 419 of file kdebug.h.

10.36 Fiasco real time scheduling extensions

Real time scheduling extension for the Fiasco L4 implementation.

Collaboration diagram for Fiasco real time scheduling extensions:

Real time scheduling extension for the Fiasco L4 implementation.

10.37 Fiasco-UX Virtual devices

Virtual hardware devices, provided by Fiasco-UX.

Collaboration diagram for Fiasco-UX Virtual devices:

Data Structures

- struct l4_vhw_entry

  Description of a device.
- struct l4_vhw_descriptor

  Virtual hardware devices description.

Enumerations

- enum l4_vhw_entry_type { L4_TYPE_VHW_NONE, L4_TYPE_VHW_FRAMEBUFFER, L4_TYPE_VHW_I↩NPUT, L4_TYPE_VHW_NET }

  Type of device.

10.37.1 Detailed Description

Virtual hardware devices, provided by Fiasco-UX.

```
#include <l4/sys/vhw.h>
```

10.37.2 Enumeration Type Documentation

10.37.2.1 enum l4_vhw_entry_type

Type of device.

Enumerator

L4_TYPE_VHW_NONE None entry.

L4_TYPE_VHW_FRAMEBUFFER Framebuffer device.

L4_TYPE_VHW_INPUT Input device.

L4_TYPE_VHW_NET Network device.

Definition at line 44 of file vhw.h.

10.38 Flex pages

Flex-page related API.

Collaboration diagram for Flex pages:

Data Structures

- union l4_fpage_t

  L4 flexpage type.
- struct l4_snd_fpage_t

  Send-flex-page types.

Enumerations

- enum l4_fpage_consts {
  L4_FPAGE_RIGHTS_SHIFT = 0, L4_FPAGE_TYPE_SHIFT = 4, L4_FPAGE_SIZE_SHIFT = 6, L4_FPAGE_ADDR_SHIFT = 12,
  L4_FPAGE_RIGHTS_BITS = 4, L4_FPAGE_TYPE_BITS = 2, L4_FPAGE_SIZE_BITS = 6, L4_FPAGE_ADDR_BITS = L4_MWORD_BITS - L4_FPAGE_ADDR_SHIFT,
  L4_FPAGE_RIGHTS_MASK }

  L4 flexpage structure.
- enum { L4_WHOLE_ADDRESS_SPACE = 63 }

  Constants for flexpages.
- enum L4_fpage_rights {
  L4_FPAGE_X = 1, L4_FPAGE_W = 2, L4_FPAGE_RO = 4, L4_FPAGE_RW = L4_FPAGE_RO | L4_FPAGE_W,
  L4_FPAGE_RX = L4_FPAGE_RO | L4_FPAGE_X, L4_FPAGE_RWX = L4_FPAGE_RW | L4_FPAGE_X }

  Flex-page rights.
- enum L4_cap_fpage_rights {
  L4_CAP_FPAGE_W = 0x1, L4_CAP_FPAGE_S = 0x2, L4_CAP_FPAGE_R = 0x4, L4_CAP_FPAGE_RO = 0x4,
  L4_CAP_FPAGE_D = 0x8, L4_CAP_FPAGE_RW = L4_CAP_FPAGE_R | L4_CAP_FPAGE_W, L4_CAP_FPAGE_RS = L4_CAP_FPAGE_R | L4_CAP_FPAGE_S, L4_CAP_FPAGE_RWS = L4_CAP_FPAGE_RW | L4_CAP_FPAGE_S,
  L4_CAP_FPAGE_RWSD = L4_CAP_FPAGE_RWS | L4_CAP_FPAGE_D, L4_CAP_FPAGE_RWD = L4_CAP_FPAGE_RW | L4_CAP_FPAGE_D, L4_CAP_FPAGE_RSD = L4_CAP_FPAGE_RS | L4_CAP_FPAGE_D }

  Cap-flex-page rights.
- enum L4_fpage_type

  Flex-page type.
- enum L4_fpage_control

  Flex-page map control flags.
- enum L4_obj_fpage_ctl { L4_FPAGE_C_NO_REF_CNT = 0x10 }

Flex-page map control for capabilities (snd_base)

- enum l4_fpage_cacheability_opt_t { L4_FPAGE_CACHE_OPT = 0x1, L4_FPAGE_CACHEABLE = 0x3, L4_FPAGE_BUFFERABLE = 0x5, L4_FPAGE_UNCACHEABLE = 0x1 }

  Flex-page cacheability option.

- enum { L4_WHOLE_IOADDRESS_SPACE = 16, L4_IOPORT_MAX = (1L << L4_WHOLE_IOADDRESS_SPACE) }

  Special constants for IO flex pages.

Functions

- l4_fpage_t l4_fpage (unsigned long address, unsigned int size, unsigned char rights) L4_NOTHROW

  Create a memory flex page.

- l4_fpage_t l4_fpage_all (void) L4_NOTHROW

  Get a flex page, describing all address spaces at once.

- l4_fpage_t l4_fpage_invalid (void) L4_NOTHROW

  Get an invalid flex page.

- l4_fpage_t l4_iofpage (unsigned long port, unsigned int size) L4_NOTHROW

  Create an IO-port flex page.

- l4_fpage_t l4_obj_fpage (l4_cap_idx_t obj, unsigned int order, unsigned char rights) L4_NOTHROW

  Create a kernel-object flex page.

- int l4_is_fpage_writable (l4_fpage_t fp) L4_NOTHROW

  Test if the flex page is writable.

- unsigned l4_fpage_rights (l4_fpage_t f) L4_NOTHROW

  Return rights from a flex page.

- unsigned l4_fpage_type (l4_fpage_t f) L4_NOTHROW

  Return type from a flex page.

- unsigned l4_fpage_size (l4_fpage_t f) L4_NOTHROW

  Return size from a flex page.

- unsigned long l4_fpage_page (l4_fpage_t f) L4_NOTHROW

  Return page from a flex page.

- l4_fpage_t l4_fpage_set_rights (l4_fpage_t src, unsigned char new_rights) L4_NOTHROW

  Set new right in a flex page.

- int l4_fpage_contains (l4_fpage_t fpage, l4_addr_t addr, unsigned size) L4_NOTHROW

  Test whether a given range is completely within an fpage.

- unsigned char l4_fpage_max_order (unsigned char order, l4_addr_t addr, l4_addr_t min_addr, l4_addr_t max_addr, l4_addr_t hotspot=0)

  Determine maximum flex page size of a region.

10.38.1 Detailed Description

Flex-page related API.

A flex page is a page with a variable size, that can describe memory, IO-Ports (IA32 only), and sets of kernel objects.

A flex page describes an always size aligned region of an address space. The size is given in a log2 scale. This means the size in elements (bytes for memory, ports for IO-Ports, and capabilities for kernel objects) is always a power of two.

A flex page also carries type and access right information for the described region. The type information selects the address space in which the flex page is valid. Access rights have a meaning depending on the specific address space (type).

There exists a special type for defining *receive windows* or for the l4_task_unmap() method, that can be used to describe all address spaces (all types) with a single flex page.

10.38.2 Enumeration Type Documentation

10.38.2.1 anonymous enum

Constants for flexpages.

Enumerator

L4_WHOLE_ADDRESS_SPACE Whole address space size.

Definition at line 89 of file __l4_fpage.h.

10.38.2.2 anonymous enum

Special constants for IO flex pages.

Enumerator

L4_WHOLE_IOADDRESS_SPACE Whole I/O address space size.

L4_IOPORT_MAX Maximum I/O port address.

Definition at line 263 of file __l4_fpage.h.

10.38.2.3 enum L4_cap_fpage_rights

Cap-flex-page rights.

Capabilities are modified or transfered with map and unmap operations. For that capabilities are wrapped into flex-page objects. The flex-page carries a set of rights the sender wants to hand over to the receiver along with the capability.

For the user only the 'S' and the 'W' right are visible. Other rights such as the 'D' right are internal to the corresponding kernel object and cannot be evaluated by the receiver.

Note

A thread can also map a capability from its task's capability table with a reduced set of rights into another slot of its own capability table.

Enumerator

L4_CAP_FPAGE_W Interface specific 'W' right for capability flex-pages. The semantics of the 'W' right is defined by the protocol. For example in case of a dataspace cap, the 'W' right is needed to get a writable dataspace.

L4_CAP_FPAGE_S Interface specific 'S' right for capability flex-pages. The semantics of the 'S' right is defined by the interface. The kernel masks this right with the 'S' right of the IPC gate over which the capability is mapped. That means that the receiver capability will only have the 'S' right set if both the flex-page and the IPC gate have the 'S' bit set.

L4_CAP_FPAGE_R Read right for capability flex-pages. This is always required, otherwise no capability is mapped.

L4_CAP_FPAGE_RO Read right for capability flex-pages. This is always required, otherwise no capability is mapped.

L4_CAP_FPAGE_D Delete right for capability flex-pages. This allows the receiver to delete the corresponding kernel object using unmap() regardless of other tasks still holding a capability to the kernel object. Such capabilities are set to an empty capability if the object is deleted.

L4_CAP_FPAGE_RW Read and interface specific 'W' right for capability flex-pages. The semantics of the 'W' right is defined by the interface.

See also

L4_CAP_FPAGE_W

L4_CAP_FPAGE_RS Read and interface specific 'S' right for capability flex-pages. The semantics of the 'S' right is defined by the interface.

See also

L4_CAP_FPAGE_S

L4_CAP_FPAGE_RWS Read, interface specific 'W', and 'S' rights for capability flex-pages. The semantics of the 'W' and 'S' right are defined by the interface.

See also

L4_CAP_FPAGE_R, L4_CAP_FPAGE_W, and L4_CAP_FPAGE_S

L4_CAP_FPAGE_RWSD Full rights for capability flex-pages.

See also

L4_CAP_FPAGE_R, L4_CAP_FPAGE_W, L4_CAP_FPAGE_S, and L4_CAP_FPAGE_D

L4_CAP_FPAGE_RWD Read, write, and delete right for capability flex-pages.

See also

L4_CAP_FPAGE_R, L4_CAP_FPAGE_W, and L4_CAP_FPAGE_D

L4_CAP_FPAGE_RSD Read, 'S', and delete right for capability flex-pages.

See also

L4_CAP_FPAGE_R, L4_CAP_FPAGE_S, and L4_CAP_FPAGE_D

Definition at line 134 of file __l4_fpage.h.

10.38.2.4 enum l4_fpage_cacheability_opt_t

Flex-page cacheability option.

Enumerator

L4_FPAGE_CACHE_OPT Enable the cacheability option in a send flex page.

L4_FPAGE_CACHEABLE Cacheability option to enable caches for the mapping.

L4_FPAGE_BUFFERABLE Cacheability option to enable buffered writes for the mapping.

L4_FPAGE_UNCACHEABLE Cacheability option to disable caching for the mapping.

Definition at line 244 of file __l4_fpage.h.

10.38.2.5 enum l4_fpage_consts

L4 flexpage structure.

Enumerator

L4_FPAGE_RIGHTS_SHIFT Access permissions shift.

L4_FPAGE_TYPE_SHIFT Flexpage type shift (memory, IO port, obj...)

L4_FPAGE_SIZE_SHIFT Flexpage size shift (log2-based)

L4_FPAGE_ADDR_SHIFT Page address shift.

L4_FPAGE_RIGHTS_BITS Access permissions size.

L4_FPAGE_TYPE_BITS Flexpage type size (memory, IO port, obj...)

L4_FPAGE_SIZE_BITS Flexpage size size (log2-based)

L4_FPAGE_ADDR_BITS Page address size.

L4_FPAGE_RIGHTS_MASK Mask to get the flexpage rights.

Definition at line 55 of file __l4_fpage.h.

10.38.2.6 enum L4_fpage_rights

Flex-page rights.

Enumerator

L4_FPAGE_X Executable flex page.
L4_FPAGE_W Writable flex page.
L4_FPAGE_RO Read-only flex page.
L4_FPAGE_RW Read-write flex page.
L4_FPAGE_RX Read-execute flex page.
L4_FPAGE_RWX Read-write-execute flex page.

Definition at line 107 of file __l4_fpage.h.

10.38.2.7 enum L4_obj_fpage_ctl

Flex-page map control for capabilities (snd_base)

Enumerator

L4_FPAGE_C_NO_REF_CNT Don't increase the reference counter.

Definition at line 234 of file __l4_fpage.h.

10.38.3 Function Documentation

10.38.3.1 l4_fpage_t l4_fpage (unsigned long *address*, unsigned int *size*, unsigned char *rights*) [inline]

Create a memory flex page.

Parameters

address	Flex-page start address
size	Flex-page size (log2), L4_WHOLE_ADDRESS_SPACE to specify the whole address space (with *address* 0)
rights	Access rights, see l4_fpage_rights

Returns

Memory flex page

Definition at line 533 of file __l4_fpage.h.

10.38.3.2 l4_fpage_t l4_fpage_all (void) [inline]

Get a flex page, describing all address spaces at once.

Returns

Special *all-spaces* flex page.

Definition at line 551 of file __l4_fpage.h.

References L4_WHOLE_ADDRESS_SPACE.

10.38.3.3 int l4_fpage_contains (l4_fpage_t *fpage*, l4_addr_t *addr*, unsigned *size*) [inline]

Test whether a given range is completely within an fpage.

Parameters

fpage	Flex page
addr	Address
size	Size of range in log2.

Definition at line 583 of file __l4_fpage.h.

References L4_FPAGE_ADDR_SHIFT, and l4_fpage_page().

Here is the call graph for this function:

10.38.3.4 l4_fpage_t l4_fpage_invalid (void) [inline]

Get an invalid flex page.

Returns

Special *invalid* flex page.

Definition at line 557 of file __l4_fpage.h.

10.38.3.5 unsigned char l4_fpage_max_order (unsigned char *order*, l4_addr_t *addr*, l4_addr_t *min_addr*, l4_addr_t *max_addr*, l4_addr_t *hotspot* = 0) [inline]

Determine maximum flex page size of a region.

Parameters

order	Order value to start with (e.g. for memory L4_LOG2_PAGESIZE would be used)
addr	Address to be covered by the flex page.
min_addr	Start of region / minimal address (including).
max_addr	End of region / maximal address (excluding).
hotspot	(Optional) hot spot.

Returns

Maximum order (log2-size) possible.

Note

The start address of the flex-page can be determined with l4_trunc_size(addr, returnvalue)

Definition at line 591 of file __l4_fpage.h.

References l4_trunc_size().

Referenced by L4Re::Dataspace::map(), and L4Re::Dataspace::map_region().

Here is the call graph for this function:

Here is the caller graph for this function:

10.38.3.6 unsigned long l4_fpage_page (l4_fpage_t *f*) [inline]

Return page from a flex page.

Parameters

f	Flex page

Returns

Page part of the given flex page.

Definition at line 501 of file __l4_fpage.h.

References L4_FPAGE_ADDR_SHIFT.

Referenced by l4_fpage_contains().

Here is the caller graph for this function:

10.38.3.7 unsigned l4_fpage_rights (l4_fpage_t *f*) [inline]

Return rights from a flex page.

Parameters

f	Flex page

Returns

Size part of the given flex page.

Definition at line 483 of file __l4_fpage.h.

References L4_FPAGE_RIGHTS_MASK, and L4_FPAGE_RIGHTS_SHIFT.

Referenced by l4_is_fpage_writable().

Here is the caller graph for this function:

10.38.3.8 l4_fpage_t l4_fpage_set_rights (l4_fpage_t *src*, unsigned char *new_rights*) [inline]

Set new right in a flex page.

Parameters

src	Flex page
new_rights	New rights

Returns

Modified flex page with new rights.

Definition at line 524 of file __l4_fpage.h.

References L4_FPAGE_RIGHTS_MASK, L4_FPAGE_RIGHTS_SHIFT, and l4_fpage_t::raw.

10.38.3.9 unsigned l4_fpage_size (l4_fpage_t *f*) [inline]

Return size from a flex page.

Parameters

f	Flex page

Returns

Size part of the given flex page.

Definition at line 495 of file __l4_fpage.h.

References L4_FPAGE_SIZE_SHIFT.

10.38.3.10 unsigned l4_fpage_type (l4_fpage_t *f*) [inline]

Return type from a flex page.

Parameters

f	Flex page

Returns

Type part of the given flex page.

Definition at line 489 of file __l4_fpage.h.

References L4_FPAGE_TYPE_SHIFT.

10.38.3.11 l4_fpage_t l4_iofpage (unsigned long *port*, unsigned int *size*) [inline]

Create an IO-port flex page.

Parameters

port	I/O-flex-page port base
size	I/O-flex-page size, L4_WHOLE_IOADDRESS_SPACE to specify the whole I/O address space (with *port* 0)

Returns

I/O flex page

Definition at line 539 of file __l4_fpage.h.

References L4_FPAGE_ADDR_SHIFT, and L4_FPAGE_RW.

10.38.3.12 int l4_is_fpage_writable (l4_fpage_t *fp*) [inline]

Test if the flex page is writable.

Parameters

fp	Flex page.

Returns

!= 0 if flex page is writable, 0 if not

Definition at line 564 of file __l4_fpage.h.

References l4_fpage_rights(), and L4_FPAGE_W.

Here is the call graph for this function:

10.38.3.13 l4_fpage_t l4_obj_fpage (l4_cap_idx_t *obj*, unsigned int *order*, unsigned char *rights*) [inline]

Create a kernel-object flex page.

Parameters

obj	Base capability selector.
order	Log2 size (number of capabilities).
rights	Access rights

Returns

Flex page for a set of kernel objects.

Definition at line 545 of file __l4_fpage.h.

Referenced by l4_factory_create_gate_u(), l4_icu_bind_u(), l4_icu_unbind_u(), l4_irq_attach_u(), and l4_irq_mux↩_chain_u().

Here is the caller graph for this function:

10.39 Functions for rendering bitmap data in frame buffers

Collaboration diagram for Functions for rendering bitmap data in frame buffers:

Data Structures

- struct gfxbitmap_offset

  offsets in pmap[] and bmap[]

Typedefs

- typedef unsigned int gfxbitmap_color_t

  Standard color type.
- typedef unsigned int gfxbitmap_color_pix_t

  Specific color type.

Functions

- gfxbitmap_color_pix_t gfxbitmap_convert_color (l4re_video_view_info_t *vi, gfxbitmap_color_t rgb)

  Convert a color.
- void gfxbitmap_fill (l4_uint8_t *vfb, l4re_video_view_info_t *vi, int x, int y, int w, int h, gfxbitmap_color_pix_t color)

  Fill a rectangular area with a color.
- void gfxbitmap_bmap (l4_uint8_t *vfb, l4re_video_view_info_t *vi, l4_int16_t x, l4_int16_t y, l4_uint32_t w, l4_uint32_t h, l4_uint8_t *bmap, gfxbitmap_color_pix_t fgc, gfxbitmap_color_pix_t bgc, struct gfxbitmap_offset *offset, l4_uint8_t mode)

  Fill a rectangular area with a bicolor bitmap pattern.
- void gfxbitmap_set (l4_uint8_t *vfb, l4re_video_view_info_t *vi, l4_int16_t x, l4_int16_t y, l4_uint32_t w, l4_uint32_t h, l4_uint32_t xoffs, l4_uint32_t yoffs, l4_uint8_t *pmap, struct gfxbitmap_offset *offset, l4_uint32_t pwidth)

  Set area from source area.
- void gfxbitmap_copy (l4_uint8_t *dest, l4_uint8_t *src, l4re_video_view_info_t *vi, int x, int y, int w, int h, int dx, int dy)

  Copy a rectangular area.

10.39.1 Detailed Description

10.39.2 Typedef Documentation

10.39.2.1 typedef unsigned int gfxbitmap_color_pix_t

Specific color type.

This color type is specific for a particular framebuffer, it can be use to write pixel on a framebuffer. Use gfxbitmap↩_convert_color to convert from gfxbitmap_color_t to gfxbitmap_color_pix_t.

Definition at line 66 of file bitmap.h.

10.39.2.2 typedef unsigned int gfxbitmap_color_t

Standard color type.

It's a RGB type with 8bits for each channel, regardless of the framebuffer used.

Definition at line 57 of file bitmap.h.

10.39.3 Function Documentation

10.39.3.1 void gfxbitmap_bmap (l4_uint8_t * *vfb*, l4re_video_view_info_t * *vi*, l4_int16_t *x*, l4_int16_t *y*, l4_uint32_t *w*, l4_uint32_t *h*, l4_uint8_t * *bmap*, gfxbitmap_color_pix_t *fgc*, gfxbitmap_color_pix_t *bgc*, struct gfxbitmap_offset * *offset*, l4_uint8_t *mode*)

Fill a rectangular area with a bicolor bitmap pattern.

Parameters

vfb	Frame buffer.
vi	Frame buffer information structure.
x	X position of area.
y	Y position of area.
w	Width of area.
h	Height of area.
bmap	Bitmap pattern.
fgc	Foreground color.
bgc	Background color.
offset	Offsets.
mode	Mode

See also

#pSLIM_BMAP_START_MSB and #pSLIM_BMAP_START_LSB.

10.39.3.2 gfxbitmap_color_pix_t gfxbitmap_convert_color (l4re_video_view_info_t * *vi*, gfxbitmap_color_t *rgb*)

Convert a color.

Converts a given color in standard format to the format used in the framebuffer.

10.39.3.3 void gfxbitmap_copy (l4_uint8_t * *dest*, l4_uint8_t * *src*, l4re_video_view_info_t * *vi*, int *x*, int *y*, int *w*, int *h*, int *dx*, int *dy*)

Copy a rectangular area.

Parameters

dest	Destination frame buffer.
src	Source frame buffer.
vi	Frame buffer information structure.
x	Source X position of area.
y	Source Y position of area.
w	Width of area.
h	Height of area.
dx	Source X position of area.
dy	Source Y position of area.

10.39.3.4 void gfxbitmap_fill (l4_uint8_t * *vfb*, l4re_video_view_info_t * *vi*, int *x*, int *y*, int *w*, int *h*, gfxbitmap_color_pix_t *color*)

Fill a rectangular area with a color.

Parameters

vfb	Frame buffer.
vi	Frame buffer information structure.
x	X position of area.
y	Y position of area.
w	Width of area.
h	Height of area.
color	Color of area.

10.39.3.5 void gfxbitmap_set (l4_uint8_t * *vfb*, l4re_video_view_info_t * *vi*, l4_int16_t *x*, l4_int16_t *y*, l4_uint32_t *w*, l4_uint32_t *h*, l4_uint32_t *xoffs*, l4_uint32_t *yoffs*, l4_uint8_t * *pmap*, struct gfxbitmap_offset * *offset*, l4_uint32_t *pwidth*)

Set area from source area.

Parameters

vfb	Frame buffer.
vi	Frame buffer information structure.
x	X position of area.
y	Y position of area.
w	Width of area.
h	Height of area.
pmap	Source.
xoffs	X offset.
yoffs	Y offset.
offset	Offsets.
pwidth	Width of source in bytes.

10.40 Functions for rendering bitmap fonts to frame buffers

Collaboration diagram for Functions for rendering bitmap fonts to frame buffers:

Macros

- #define GFXBITMAP_DEFAULT_FONT (void ∗)0

  Constant to use for the default font.

Typedefs

- typedef void ∗ gfxbitmap_font_t

  Font.

Enumerations

- enum

  Constant for length field.

Functions

- int gfxbitmap_font_init (void)

  Initialize the library.
- gfxbitmap_font_t gfxbitmap_font_get (const char ∗name)

  Get a font descriptor.
- unsigned gfxbitmap_font_width (gfxbitmap_font_t font)

  Get the font width.
- unsigned gfxbitmap_font_height (gfxbitmap_font_t font)

  Get the font height.
- void ∗ gfxbitmap_font_data (gfxbitmap_font_t font, unsigned c)

  Get bitmap font data for a specific character.
- void gfxbitmap_font_text (void ∗fb, l4re_video_view_info_t ∗vi, gfxbitmap_font_t font, const char ∗text, unsigned len, unsigned x, unsigned y, gfxbitmap_color_pix_t fg, gfxbitmap_color_pix_t bg)

  Render a string to a framebuffer.
- void gfxbitmap_font_text_scale (void ∗fb, l4re_video_view_info_t ∗vi, gfxbitmap_font_t font, const char ∗text, unsigned len, unsigned x, unsigned y, gfxbitmap_color_pix_t fg, gfxbitmap_color_pix_t bg, int scale_x, int scale_y)

  Render a string to a framebuffer, including scaling.

10.40.1 Detailed Description

10.40.2 Enumeration Type Documentation

10.40.2.1 anonymous enum

Constant for length field.

Use this if the function should call strlen on the text argument itself.

Definition at line 38 of file font.h.

10.40.3 Function Documentation

10.40.3.1 void* gfxbitmap_font_data (gfxbitmap_font_t *font*, unsigned *c*)

Get bitmap font data for a specific character.

Parameters

font	Font.
c	Character.

Returns

Pointer to bmap data, NULL on error.

10.40.3.2 gfxbitmap_font_t gfxbitmap_font_get (const char * *name*)

Get a font descriptor.

Parameters

name	Name of the font.

Returns

A (opaque) font descriptor, or NULL if font could not be found.

10.40.3.3 unsigned gfxbitmap_font_height (gfxbitmap_font_t *font*)

Get the font height.

Parameters

font	Font.

Returns

Font height, 0 if font height could not be retrieved.

10.40.3.4 int gfxbitmap_font_init (void)

Initialize the library.

This function must be called before any other font function of this library.

Returns

0 on success, other on error

10.40.3.5 void gfxbitmap_font_text (void * *fb*, l4re_video_view_info_t * *vi*, gfxbitmap_font_t *font*, const char * *text*, unsigned *len*, unsigned *x*, unsigned *y*, gfxbitmap_color_pix_t *fg*, gfxbitmap_color_pix_t *bg*)

Render a string to a framebuffer.

Parameters

fb	Pointer to frame buffer.
vi	Frame buffer info structure.
font	Font.
text	Text string.
len	Length of the text string.
x	Horizontal position in the frame buffer.
y	Vertical position in the frame buffer.
fg	Foreground color.
bg	Background color.

10.40.3.6 void gfxbitmap_font_text_scale (void * *fb*, l4re_video_view_info_t * *vi*, gfxbitmap_font_t *font*, const char * *text*, unsigned *len*, unsigned *x*, unsigned *y*, gfxbitmap_color_pix_t *fg*, gfxbitmap_color_pix_t *bg*, int *scale_x*, int *scale_y*)

Render a string to a framebuffer, including scaling.

Parameters

fb	Pointer to frame buffer.
vi	Frame buffer info structure.
font	Font.
text	Text string.
len	Length of the text string.
x	Horizontal position in the frame buffer.
y	Vertical position in the frame buffer.
fg	Foreground color.
bg	Background color.
scale_x	Horizonal scale factor.
scale_y	Vertical scale factor.

10.40.3.7 unsigned gfxbitmap_font_width (gfxbitmap_font_t *font*)

Get the font width.

Parameters

font	Font.

Returns

Font width, 0 if font width could not be retrieved.

10.41 Functions to manipulate the local IDT

Collaboration diagram for Functions to manipulate the local IDT:

Data Structures

- struct l4util_idt_desc_t

  IDT entry.
- struct l4util_idt_header_t

  Header of an IDT table.

10.41.1 Detailed Description

10.42 IA32 Port I/O API

Collaboration diagram for IA32 Port I/O API:

Functions

- l4_uint8_t l4util_in8 (l4_uint16_t port)

  Read byte from I/O port.
- l4_uint16_t l4util_in16 (l4_uint16_t port)

  Read 16-bit-value from I/O port.
- l4_uint32_t l4util_in32 (l4_uint16_t port)

  Read 32-bit-value from I/O port.
- void l4util_ins8 (l4_uint16_t port, l4_umword_t addr, l4_umword_t count)

  Read a block of 8-bit-values from I/O ports.
- void l4util_ins16 (l4_uint16_t port, l4_umword_t addr, l4_umword_t count)

  Read a block of 16-bit-values from I/O ports.
- void l4util_ins32 (l4_uint16_t port, l4_umword_t addr, l4_umword_t count)

  Read a block of 32-bit-values from I/O ports.
- void l4util_out8 (l4_uint8_t value, l4_uint16_t port)

  Write byte to I/O port.
- void l4util_out16 (l4_uint16_t value, l4_uint16_t port)

  Write 16-bit-value to I/O port.
- void l4util_out32 (l4_uint32_t value, l4_uint16_t port)

  Write 32-bit-value to I/O port.
- void l4util_outs8 (l4_uint16_t port, l4_umword_t addr, l4_umword_t count)

  Write a block of bytes to I/O port.
- void l4util_outs16 (l4_uint16_t port, l4_umword_t addr, l4_umword_t count)

  Write a block of 16-bit-values to I/O port.
- void l4util_outs32 (l4_uint16_t port, l4_umword_t addr, l4_umword_t count)

  Write block of 32-bit-values to I/O port.
- void l4util_iodelay (void)

  delay I/O port access by writing to port 0x80

10.42.1 Detailed Description

10.42.2 Function Documentation

10.42.2.1 l4_uint16_t l4util_in16 (l4_uint16_t *port*) [inline]

Read 16-bit-value from I/O port.

Parameters

port	I/O port address

Returns

value

Definition at line 180 of file port_io.h.

10.42.2.2 l4_uint32_t l4util_in32 (l4_uint16_t *port*) [inline]

Read 32-bit-value from I/O port.

Parameters

port	I/O port address

Returns

value

Definition at line 188 of file port_io.h.

10.42.2.3 l4_uint8_t l4util_in8 (l4_uint16_t *port*) [inline]

Read byte from I/O port.

Parameters

port	I/O port address

Returns

value

Definition at line 172 of file port_io.h.

10.42.2.4 void l4util_ins16 (l4_uint16_t *port*, l4_umword_t *addr*, l4_umword_t *count*) [inline]

Read a block of 16-bit-values from I/O ports.

Parameters

port	I/O port address
addr	address of buffer
count	number of I/O operations

Definition at line 205 of file port_io.h.

10.42.2.5 void l4util_ins32 (l4_uint16_t *port*, l4_umword_t *addr*, l4_umword_t *count*) [inline]

Read a block of 32-bit-values from I/O ports.

Parameters

port	I/O port address
addr	address of buffer
count	number of I/O operations

Definition at line 214 of file port_io.h.

10.42.2.6 void l4util_ins8 (l4_uint16_t *port*, l4_umword_t *addr*, l4_umword_t *count*) [inline]

Read a block of 8-bit-values from I/O ports.

Parameters

port	I/O port address
addr	address of buffer
count	number of I/O operations

Definition at line 196 of file port_io.h.

10.42.2.7 void l4util_out16 (l4_uint16_t *value*, l4_uint16_t *port*) [inline]

Write 16-bit-value to I/O port.

Parameters

port	I/O port address
value	value to write

Definition at line 229 of file port_io.h.

10.42.2.8 void l4util_out32 (l4_uint32_t *value*, l4_uint16_t *port*) [inline]

Write 32-bit-value to I/O port.

Parameters

port	I/O port address
value	value to write

Definition at line 235 of file port_io.h.

10.42.2.9 void l4util_out8 (l4_uint8_t *value*, l4_uint16_t *port*) [inline]

Write byte to I/O port.

Parameters

port	I/O port address
value	value to write

Definition at line 223 of file port_io.h.

10.42.2.10 void l4util_outs16 (l4_uint16_t *port*, l4_umword_t *addr*, l4_umword_t *count*) [inline]

Write a block of 16-bit-values to I/O port.

Parameters

port	I/O port address
addr	address of buffer
count	number of I/O operations

Definition at line 250 of file port_io.h.

10.42.2.11 void l4util_outs32 (l4_uint16_t *port*, l4_umword_t *addr*, l4_umword_t *count*) [inline]

Write block of 32-bit-values to I/O port.

Parameters

port	I/O port address
addr	address of buffer
count	number of I/O operations

Definition at line 259 of file port_io.h.

10.42.2.12 void l4util_outs8 (l4_uint16_t *port*, l4_umword_t *addr*, l4_umword_t *count*) [inline]

Write a block of bytes to I/O port.

Parameters

port	I/O port address
addr	address of buffer
count	number of I/O operations

Definition at line 241 of file port_io.h.

10.43 IO interface

Typedefs

- typedef l4vbus_resource_t l4io_resource_t

  Resource descriptor.
- typedef l4vbus_device_t l4io_device_t

  Device descriptor.

Enumerations

- enum l4io_iomem_flags_t {
  L4IO_MEM_NONCACHED = 0, L4IO_MEM_CACHED = 1, L4IO_MEM_USE_MTRR = 2 , L4IO_MEM_US↩
  E_RESERVED_AREA = 0x40 << 8,
  L4IO_MEM_EAGER_MAP = 0x80 << 8 }

  Flags for IO memory.
- enum l4io_device_types_t {
  L4IO_DEVICE_INVALID = 0, L4IO_DEVICE_PCI, L4IO_DEVICE_USB, L4IO_DEVICE_OTHER,
  L4IO_DEVICE_ANY = ∼0 }

  Device types.
- enum l4io_resource_types_t {
  L4IO_RESOURCE_INVALID = L4VBUS_RESOURCE_INVALID, L4IO_RESOURCE_IRQ = L4VBUS_RE↩
  SOURCE_IRQ, L4IO_RESOURCE_MEM = L4VBUS_RESOURCE_MEM, L4IO_RESOURCE_PORT = L4↩
  VBUS_RESOURCE_PORT,
  L4IO_RESOURCE_ANY = ∼0 }

  Resource types.

Functions

- long l4io_request_iomem (l4_addr_t phys, unsigned long size, int flags, l4_addr_t *virt)

  Request an IO memory region.
- long l4io_request_iomem_region (l4_addr_t phys, l4_addr_t virt, unsigned long size, int flags)

  Request an IO memory region and map to a specified region.
- long l4io_release_iomem (l4_addr_t virt, unsigned long size)

  Release an IO memory region.
- long l4io_search_iomem_region (l4_addr_t phys, l4_addr_t size, l4_addr_t *rstart, l4_addr_t *rsize)

  Search for a IO memory region.
- long l4io_request_ioport (unsigned portnum, unsigned len)

  Request an IO port region.
- long l4io_release_ioport (unsigned portnum, unsigned len)

  Release an IO port region.
- int l4io_lookup_device (const char *devname, l4io_device_handle_t *dev_handle, l4io_device_t *dev, l4io↩
  _resource_handle_t *res_handle)

  Find a device by name.
- int l4io_lookup_resource (l4io_device_handle_t devhandle, enum l4io_resource_types_t type, l4io_↩
  resource_handle_t *reshandle, l4io_resource_t *res)

  Request a specific resource from a device description.
- l4_addr_t l4io_request_resource_iomem (l4io_device_handle_t devhandle, l4io_resource_handle_↩
  t *reshandle)

  Request IO memory.
- int l4io_has_resource (enum l4io_resource_types_t type, l4vbus_paddr_t start, l4vbus_paddr_t end)

  Check if a resource is available.

10.43.1 Detailed Description

10.43.2 Typedef Documentation

10.43.2.1 typedef l4vbus_resource_t l4io_resource_t

Resource descriptor.

For IRQ types, the end field is not used, i.e. only a single interrupt can be described with a l4io_resource_t

Definition at line 69 of file types.h.

10.43.3 Enumeration Type Documentation

10.43.3.1 enum l4io_device_types_t

Device types.

Enumerator

L4IO_DEVICE_INVALID Invalid type.

L4IO_DEVICE_PCI PCI device.

L4IO_DEVICE_USB USB device.

L4IO_DEVICE_OTHER Any other device without unique IDs.

L4IO_DEVICE_ANY any type

Definition at line 38 of file types.h.

10.43.3.2 enum l4io_iomem_flags_t

Flags for IO memory.

Enumerator

L4IO_MEM_NONCACHED Non-cache memory.

L4IO_MEM_CACHED Cache memory.

L4IO_MEM_USE_MTRR Use MTRR.

L4IO_MEM_USE_RESERVED_AREA Use reserved area for mapping I/O memory. Flag only valid for l4io↩_request_iomem_region()

L4IO_MEM_EAGER_MAP Eagerly map the I/O memory. Passthrough to the l4re-rm.

Definition at line 16 of file types.h.

10.43.3.3 enum l4io_resource_types_t

Resource types.

Enumerator

L4IO_RESOURCE_INVALID Invalid type.

L4IO_RESOURCE_IRQ Interrupt resource.

L4IO_RESOURCE_MEM I/O memory resource.

L4IO_RESOURCE_PORT I/O port resource (x86 only)

L4IO_RESOURCE_ANY any type

Definition at line 50 of file types.h.

10.43.4 Function Documentation

10.43.4.1 int l4io_has_resource (enum l4io_resource_types_t *type*, l4vbus_paddr_t *start*, l4vbus_paddr_t *end*)

Check if a resource is available.

Parameters

type	Type of resource
start	Minimal value.
end	Maximum value.

10.43.4.2 int l4io_lookup_device (const char * *devname*, l4io_device_handle_t * *dev_handle*, l4io_device_t * *dev*, l4io_resource_handle_t * *res_handle*)

Find a device by name.

Parameters

devname	Name of device

Return values

dev_handle	Device handle for found device, can be NULL.
dev	Device information, filled by the function, can be NULL.
res_handle	Resource handle, can be NULL.

Returns

0 on success, error code otherwise

10.43.4.3 int l4io_lookup_resource (l4io_device_handle_t *devhandle*, enum l4io_resource_types_t *type*, l4io_resource_handle_t * *reshandle*, l4io_resource_t * *res*)

Request a specific resource from a device description.

Parameters

devhandle	Device handle.
type	Type of resource to request (see #l4io_resource_types_t)
reshandle	Resource handle, start with handle returned by device functions.

Return values

reshandle	Next resource handle.
res	Device descriptor

Returns

0 on success, error code otherwise, esp. -L4_ENOENT if no more resources found

10.43.4.4 long l4io_release_iomem (l4_addr_t *virt*, unsigned long *size*)

Release an IO memory region.

Parameters

virt	Virtual address of region to free, see l4io_request_iomem
size	Size of the region to release.

Returns

0 on success, <0 on error

10.43.4.5 long l4io_release_ioport (unsigned *portnum*, unsigned *len*)

Release an IO port region.

Parameters

portnum	Start of port range to release
len	Length of range to request

Returns

0 on success, <0 on error

Note

X86 architecture only

10.43.4.6 long l4io_request_iomem (l4_addr_t *phys*, unsigned long *size*, int *flags*, l4_addr_t * *virt*)

Request an IO memory region.

Parameters

phys	Physical address of the I/O memory region
size	Size of the region in Bytes, granularity pages.
flags	See l4io_iomem_flags_t

Return values

virt	Virtual address the region is available at.

Returns

0 on success, <0 on error

Note

This function uses L4Re functionality to reserve a part of the virtual address space of the caller.

10.43.4.7 long l4io_request_iomem_region (l4_addr_t *phys*, l4_addr_t *virt*, unsigned long *size*, int *flags*)

Request an IO memory region and map to a specified region.

Parameters

phys	Physical address of the I/O memory region
virt	Virtual address.
size	Size of the region in Bytes, granularity pages.
flags	See l4io_iomem_flags_t

Returns

0 on success, <0 on error

Note

This function uses L4Re functionality to reserve a part of the virtual address space of the caller.

10.43.4.8 long l4io_request_ioport (unsigned *portnum*, unsigned *len*)

Request an IO port region.

Parameters

portnum	Start of port range to request
len	Length of range to request

Returns

0 on success, <0 on error

Note

X86 architecture only

10.43.4.9 l4_addr_t l4io_request_resource_iomem (l4io_device_handle_t *devhandle*, l4io_resource_handle_t * *reshandle*)

Request IO memory.

Parameters

devhandle	Device handle.

Return values

reshandle	Resource handle, input and ouput, return next resource handle

Returns

0 on error, virtual address otherwise

10.43.4.10 long l4io_search_iomem_region (l4_addr_t *phys*, l4_addr_t *size*, l4_addr_t * *rstart*, l4_addr_t * *rsize*)

Search for a IO memory region.

Parameters

phys	Physical address to look for
size	Size of requested memory area

Return values

rstart	Start address for region
rsize	Size of region in bytes

Returns

0 if an IO region was found, $<$0 if not

10.44 IPC Messaging Framework

10.45 IPC Streams

10.46 IPC-Gate API

Secure communication object.

Collaboration diagram for IPC-Gate API:

Functions

- l4_msgtag_t l4_ipc_gate_bind_thread (l4_cap_idx_t gate, l4_cap_idx_t thread, l4_umword_t label)

  Bind the IPC gate to a thread.
- l4_msgtag_t l4_ipc_gate_get_infos (l4_cap_idx_t gate, l4_umword_t *label)

  Get information about the IPC-gate.

10.46.1 Detailed Description

Secure communication object.

IPC-Gate objects provide a means to establish secure communication channels to L4 Threads (Thread). An IPC-Gate object can be created using a Factory (l4_factory_create_gate()) and get assigned a specific L4 thread and a *label* as protected payload. The *label* has the size of one machine word and can only be seen by the Task running the thread that is assigned of the IPC-gate. The *label* is received as part of the IPC message. The *label* can thus be used to securely identify the IPC-gate that was used to send a message.

An IPC-gate is usually used to represent an user-level object and may be the address of the data structure for the object in the server task.

With client privileges an IPC-gate does not provide any direct API and thus an IPC-gate kernel object cannot be modified by invocations. Each invocation of an IPC-gate kernel object is translated into an IPC message to the assigned thread.

Include File

```
#include <l4/sys/ipc_gate.h>
```

For the C++ interface refer to the L4::Ipc_gate documentation.

See also

Object Invocation

10.46.2 Function Documentation

10.46.2.1 l4_msgtag_t l4_ipc_gate_bind_thread (l4_cap_idx_t *gate*, l4_cap_idx_t *thread*, l4_umword_t *label*) [inline]

Bind the IPC gate to a thread.

Parameters

gate	The IPC gate object.
thread	The thread object that shall be bound to `gate`.
label	Label to assign to `gate`. The two least significant bits should usually be set to zero.

Returns

System call return tag.

Definition at line 128 of file ipc_gate.h.

References l4_utcb().

Here is the call graph for this function:

10.46.2.2 l4_msgtag_t l4_ipc_gate_get_infos (l4_cap_idx_t *gate*, l4_umword_t * *label*) [inline]

Get information about the IPC-gate.

Parameters

	gate	The IPC gate object to get information about.
`out`	*label*	The label of the IPC gate is returned here.

Returns

System call return tag.

Definition at line 135 of file ipc_gate.h.

References l4_utcb().

Here is the call graph for this function:

10.47 IRQ handling library

Collaboration diagram for IRQ handling library:

Modules

- Interface for asynchronous ISR handlers.

  This interface has just two (main) functions.
- Interface using direct functionality.

10.47.1 Detailed Description

10.48 IRQs

C IRQ interface.

Collaboration diagram for IRQs:

Enumerations

- enum L4_irq_mode {
  L4_IRQ_F_NONE = 0, L4_IRQ_F_LEVEL = 0x2, L4_IRQ_F_EDGE = 0x0, L4_IRQ_F_POS = 0x0, L4_IRQ_F_NEG = 0x4, L4_IRQ_F_BOTH = 0x8, L4_IRQ_F_LEVEL_HIGH = 0x3, L4_IRQ_F_LEVEL_LOW = 0x7,
  L4_IRQ_F_POS_EDGE = 0x1, L4_IRQ_F_NEG_EDGE = 0x5, L4_IRQ_F_BOTH_EDGE = 0x9, L4_IRQ_↩F_MASK = 0xf,
  L4_IRQ_F_SET_WAKEUP = 0x10, L4_IRQ_F_CLEAR_WAKEUP = 0x20 }

  Interrupt attributes.

Functions

- l4_msgtag_t l4_irq_attach (l4_cap_idx_t irq, l4_umword_t label, l4_cap_idx_t thread) L4_NOTHROW

  Attach a thread to an interrupt source.
- l4_msgtag_t l4_irq_attach_u (l4_cap_idx_t irq, l4_umword_t label, l4_cap_idx_t thread, l4_utcb_t ∗utcb) L4↩_NOTHROW

  Attach a thread to this interrupt.
- l4_msgtag_t l4_irq_mux_chain (l4_cap_idx_t irq, l4_cap_idx_t slave) L4_NOTHROW

  Chain an IRQ to another master IRQ source.
- l4_msgtag_t l4_irq_mux_chain_u (l4_cap_idx_t irq, l4_cap_idx_t slave, l4_utcb_t ∗utcb) L4_NOTHROW

  Attach an IRQ to this multiplexer.
- l4_msgtag_t l4_irq_detach (l4_cap_idx_t irq) L4_NOTHROW

  Detach from an interrupt source.
- l4_msgtag_t l4_irq_detach_u (l4_cap_idx_t irq, l4_utcb_t ∗utcb) L4_NOTHROW

  Detach from this interrupt.
- l4_msgtag_t l4_irq_trigger (l4_cap_idx_t irq) L4_NOTHROW

  Trigger an IRQ.
- l4_msgtag_t l4_irq_trigger_u (l4_cap_idx_t irq, l4_utcb_t ∗utcb) L4_NOTHROW

  Trigger.
- l4_msgtag_t l4_irq_receive (l4_cap_idx_t irq, l4_timeout_t to) L4_NOTHROW

  Unmask and wait for specified IRQ.
- l4_msgtag_t l4_irq_receive_u (l4_cap_idx_t irq, l4_timeout_t timeout, l4_utcb_t ∗utcb) L4_NOTHROW

  Unmask and wait for this IRQ.
- l4_msgtag_t l4_irq_wait (l4_cap_idx_t irq, l4_umword_t ∗label, l4_timeout_t to) L4_NOTHROW

  Unmask IRQ and wait for any message.

- l4_msgtag_t l4_irq_wait_u (l4_cap_idx_t irq, l4_umword_t *label, l4_timeout_t timeout, l4_utcb_t *utcb) L4_NOTHROW

  Unmask IRQ and (open) wait for any message.
- l4_msgtag_t l4_irq_unmask (l4_cap_idx_t irq) L4_NOTHROW

  Unmask IRQ.
- l4_msgtag_t l4_irq_unmask_u (l4_cap_idx_t irq, l4_utcb_t *utcb) L4_NOTHROW

  Unmask IRQ.

10.48.1 Detailed Description

C IRQ interface.

The IRQ interface provides access to abstract interrupts provided by the microkernel. Interrupts may be

- hardware interrupts provided by the platform interrupt controller,
- virtual device interrupts provided by the microkernel's virtual devices (virtual serial or trace buffer) or
- virtual interrupts that can be triggered by user programs (IRQs)

IRQ objects can be created using a factory, see the Factory API (use l4_factory_create_irq()).

Include File

```
#include <l4/sys/irq.h>
```

For the C++ interface refer to the L4::Irq API for an overview.

10.48.2 Enumeration Type Documentation

10.48.2.1 enum L4_irq_mode

Interrupt attributes.

Enumerator

L4_IRQ_F_NONE Flow types. None

L4_IRQ_F_LEVEL Level triggered.

L4_IRQ_F_EDGE Edge triggered.

L4_IRQ_F_POS Positive trigger.

L4_IRQ_F_NEG Negative trigger.

L4_IRQ_F_BOTH Both edges trigger.

L4_IRQ_F_LEVEL_HIGH Level high trigger.

L4_IRQ_F_LEVEL_LOW Level low trigger.

L4_IRQ_F_POS_EDGE Positive edge trigger.

L4_IRQ_F_NEG_EDGE Negative edge trigger.

L4_IRQ_F_BOTH_EDGE Both edges trigger.

L4_IRQ_F_MASK Mask.

L4_IRQ_F_SET_WAKEUP Wakeup source? Use irq as wakeup source

L4_IRQ_F_CLEAR_WAKEUP Do not use irq as wakeup source.

Definition at line 67 of file icu.h.

10.48.3 Function Documentation

10.48.3.1 l4_msgtag_t l4_irq_attach (l4_cap_idx_t *irq*, l4_umword_t *label*, l4_cap_idx_t *thread*) [inline]

Attach a thread to an interrupt source.

Parameters

irq	IRQ object where `thread` is attached to.
label	Identifier of the IRQ.
thread	The thread object to attach `irq` to.

Returns

Syscall return tag

Examples:

examples/sys/isr/main.c.

Definition at line 324 of file irq.h.

References l4_irq_attach_u(), and l4_utcb().

Here is the call graph for this function:

10.48.3.2 l4_msgtag_t l4_irq_attach_u (l4_cap_idx_t *irq*, l4_umword_t *label*, l4_cap_idx_t *thread*, l4_utcb_t * *utcb*) [inline]

Attach a thread to this interrupt.

Parameters

irq	IRQ object where `thread` is attached to.
label	Identifier of the IRQ (*protected label* used for messages)
thread	Capability of the thread to attach the IRQ to.
utcb	UTCB to be used for this operation, usually the UTCB of the calling thread.

Returns

Syscall return tag

Definition at line 253 of file irq.h.

References L4_FPAGE_RWX, l4_ipc_call(), L4_IPC_NEVER, l4_is_invalid_cap(), l4_map_obj_control(), l4_msgtag(), l4_obj_fpage(), L4_PROTO_IRQ_SENDER, l4_msg_regs_t::mr, and l4_fpage_t::raw.

Referenced by L4::Irq::attach(), and l4_irq_attach().

Here is the call graph for this function:

Here is the caller graph for this function:

10.48.3.3 l4_msgtag_t l4_irq_detach (l4_cap_idx_t *irq*) [inline]

Detach from an interrupt source.

Parameters

irq	The IRQ object that shall be detached.

Returns

Syscall return tag

Examples:

examples/sys/isr/main.c.

Definition at line 337 of file irq.h.

References l4_irq_detach_u(), and l4_utcb().

Here is the call graph for this function:

10.48.3.4 l4_msgtag_t l4_irq_detach_u (l4_cap_idx_t *irq*, l4_utcb_t * *utcb*) [inline]

Detach from this interrupt.

Parameters

irq	The IRQ object that shall be detached.
utcb	UTCB to be used for this operation, usually the UTCB of the calling thread.

Returns

Syscall return tag

Definition at line 284 of file irq.h.

References l4_ipc_call(), L4_IPC_NEVER, l4_msgtag(), L4_PROTO_IRQ_SENDER, and l4_msg_regs_t::mr.

Referenced by L4::Irq::detach(), and l4_irq_detach().

Here is the call graph for this function:

Here is the caller graph for this function:

10.48.3.5 l4_msgtag_t l4_irq_mux_chain (l4_cap_idx_t *irq*, l4_cap_idx_t *slave*) [inline]

Chain an IRQ to another master IRQ source.

Parameters

irq	The master IRQ object.
slave	The slave that shall be attached to the master.

Returns

Syscall return tag

The chaining feature of IRQ objects allows to deal with shared IRQs. For chaining IRQs there must be a master IRQ object, bound to the real IRQ source. Note, the master IRQ must not have a thread attached to it.

This function allows to add a limited number of slave IRQs to this master IRQ, with the semantics that each of the slave IRQs is triggered whenever the master IRQ is triggered. The master IRQ will be masked automatically when an IRQ is delivered and shall be unmasked when all attached slave IRQs are unmasked.

Definition at line 331 of file irq.h.

References l4_irq_mux_chain_u(), and l4_utcb().

Here is the call graph for this function:

10.48.3.6 l4_msgtag_t l4_irq_mux_chain_u (l4_cap_idx_t *irq*, l4_cap_idx_t *slave*, l4_utcb_t * *utcb*) [inline]

Attach an IRQ to this multiplexer.

Parameters

irq	The master IRQ object.
slave	The slave that shall be attached to the master.
utcb	UTCB to be used for this operation, usually the UTCB of the calling thread.

Returns

Syscall return tag

The chaining feature of IRQ objects allows to deal with shared IRQs. For chaining IRQs there must be an IRQ multiplexer (Irq_mux) bound to the real IRQ source. This function allows to add slave IRQs to this multiplexer.

Definition at line 272 of file irq.h.

References L4_FPAGE_RWX, l4_ipc_call(), L4_IPC_NEVER, l4_map_obj_control(), l4_msgtag(), l4_obj_fpage(), L4_PROTO_IRQ_MUX, l4_msg_regs_t::mr, and l4_fpage_t::raw.

Referenced by L4::Irq_mux::chain(), and l4_irq_mux_chain().

Here is the call graph for this function:

Here is the caller graph for this function:

10.48.3.7 l4_msgtag_t l4_irq_receive (l4_cap_idx_t *irq*, l4_timeout_t *to*) [inline]

Unmask and wait for specified IRQ.

Parameters

irq	The IRQ object that shall be unmasked.
to	Timeout.

Returns

Syscall return tag

Examples:

examples/sys/isr/main.c.

Definition at line 349 of file irq.h.

References l4_irq_receive_u(), and l4_utcb().

Referenced by L4Re::Util::Event_buffer_consumer_t< PAYLOAD >::process().

Here is the call graph for this function:

Here is the caller graph for this function:

10.48.3.8 l4_msgtag_t l4_irq_receive_u (l4_cap_idx_t *irq*, l4_timeout_t *timeout*, l4_utcb_t * *utcb*) [inline]

Unmask and wait for this IRQ.

Parameters

irq	The IRQ object that shall be unmasked.
timeout	Timeout.

utcb	UTCB to be used for this operation, usually the UTCB of the calling thread.

Returns

Syscall return tag

Note

If this is the function normally used for your IRQs consider using L4::Semaphore instead of L4::Irq.

Definition at line 300 of file irq.h.

References l4_ipc_call(), l4_msgtag(), L4_PROTO_IRQ, and l4_msg_regs_t::mr.

Referenced by l4_irq_receive(), and L4::Irq::receive().

Here is the call graph for this function:

Here is the caller graph for this function:

10.48.3.9 l4_msgtag_t l4_irq_trigger (l4_cap_idx_t *irq*) [inline]

Trigger an IRQ.

Parameters

irq	The IRQ object that shall be triggered.

Returns

Syscall return tag.

Note that this function is a send only operation, i.e. there is no return value except for a failed send operation. Especially l4_error() will return an error value from the message tag which still contains the IRQ protocol used for the send operation.

Use l4_ipc_error() to check for (send) errors.

Definition at line 343 of file irq.h.

References l4_irq_trigger_u(), and l4_utcb().

Here is the call graph for this function:

10.48.3.10 l4_msgtag_t l4_irq_trigger_u (l4_cap_idx_t *irq*, l4_utcb_t * *utcb*) [inline]

Trigger.

Parameters

irq	The IRQ object that shall be triggered.
utcb	UTCB to be used for this operation, usually the UTCB of the calling thread.

Returns

Syscall return tag for a send-only operation, use l4_ipc_error() to check for errors (**do not** use l4_error()).

Note that this function is a send-only operation, this means there is no return value except for a failed send operation. Use l4_ipc_error() to check for errors, **do not** use l4_error(), because l4_error() will always return an error.

Use l4_ipc_error() to check for (send) errors.

Definition at line 292 of file irq.h.

References L4_IPC_BOTH_TIMEOUT_0, l4_ipc_send(), l4_msgtag(), L4_PROTO_IRQ, and l4_msg_regs_t::mr.

Referenced by l4_irq_trigger(), and L4::Triggerable::trigger().

Here is the call graph for this function:

Here is the caller graph for this function:

10.48.3.11 l4_msgtag_t l4_irq_unmask (l4_cap_idx_t *irq*) [inline]

Unmask IRQ.

Parameters

irq	The IRQ object that shall be unmasked.

Returns

Syscall return tag

Note

l4_irq_wait() and l4_irq_receive() are doing the unmask themselves.

Definition at line 362 of file irq.h.

References l4_irq_unmask_u(), and l4_utcb().

Here is the call graph for this function:

10.48.3.12 l4_msgtag_t l4_irq_unmask_u (l4_cap_idx_t *irq*, l4_utcb_t * *utcb*) [inline]

Unmask IRQ.

Parameters

irq	The IRQ object that shall be unmasked.
utcb	UTCB to be used for this operation, usually the UTCB of the calling thread.

Returns

Syscall return tag

Deprecated Use L4::Irq_eoi::unmask()

Note

wait() and receive() are doing the unmask already.

Definition at line 316 of file irq.h.

References L4_IPC_NEVER, l4_ipc_send(), l4_msgtag(), L4_PROTO_IRQ, and l4_msg_regs_t::mr.

Referenced by l4_irq_unmask().

Here is the call graph for this function:

Here is the caller graph for this function:

10.48.3.13 l4_msgtag_t l4_irq_wait (l4_cap_idx_t *irq*, l4_umword_t * *label*, l4_timeout_t *to*) [inline]

Unmask IRQ and wait for any message.

Parameters

irq	The IRQ object that shall be unmasked.
label	Receive label.
to	Timeout.

Returns

Syscall return tag

Definition at line 355 of file irq.h.

References l4_irq_wait_u(), and l4_utcb().

Here is the call graph for this function:

10.48.3.14 l4_msgtag_t l4_irq_wait_u (l4_cap_idx_t *irq*, l4_umword_t * *label*, l4_timeout_t *timeout*, l4_utcb_t * *utcb*) [inline]

Unmask IRQ and (open) wait for any message.

Parameters

irq	The IRQ object that shall be unmasked.
label	The *protected label* shall be received here.

timeout	Timeout.
utcb	UTCB to be used for this operation, usually the UTCB of the calling thread.

Returns

Syscall return tag

Definition at line 307 of file irq.h.

References l4_ipc_send_and_wait(), l4_msgtag(), L4_PROTO_IRQ, and l4_msg_regs_t::mr.

Referenced by l4_irq_wait().

Here is the call graph for this function:

Here is the caller graph for this function:

10.49 Initial Environment

Environment that is initially provided to an L4 task.

Collaboration diagram for Initial Environment:

Data Structures

- class L4Re::Env

  Initial Environment (C++ version).
- struct l4re_env_cap_entry_t

  Entry in the L4Re environment array for the named inital objects.
- struct l4re_env_t

  Initial Environment structure (C version)

Typedefs

- typedef struct l4re_env_cap_entry_t l4re_env_cap_entry_t

  Entry in the L4Re environment array for the named inital objects.
- typedef struct l4re_env_t l4re_env_t

  Initial Environment structure (C version)

Functions

- l4re_env_t ∗ l4re_env (void) L4_NOTHROW

  Get L4Re initial environment (C version).
- l4_kernel_info_t ∗ l4re_kip (void) L4_NOTHROW

  Get Kernel Info Page.
- l4_cap_idx_t l4re_env_get_cap (char const ∗name) L4_NOTHROW

  Get the capability selector for the object named name.
- l4_cap_idx_t l4re_env_get_cap_e (char const ∗name, l4re_env_t const ∗e) L4_NOTHROW

  Get the capability selector for the object named name.
- l4re_env_cap_entry_t const ∗ l4re_env_get_cap_l (char const ∗name, unsigned l, l4re_env_t const ∗e) L4↩_NOTHROW

  Get the full l4re_env_cap_entry_t for the object named name.

10.49.1 Detailed Description

Environment that is initially provided to an L4 task.

The initial environment is provided to each L4 task that is started by an L4Re conform loader, such as the Moe root task. The initial environment provides access to a set of initial capabilities and some additional information about the

available resources, such as free UTCBs (see Virtual Registers) and available entries in capability table (provided by the micro kernel).

The initial set of capabilities is:

- C[parent:L4Re::Parent] — parent object
- C[mem_alloc:L4Re::Mem_alloc] — initial memory allocator
- C[log:L4Re::Log] — logging facility
- C[main_thread:L4::Thread] — first application thread
- C[rm:L4::Rm] — region manager
- C[factory:L4::Factory] — factory to create kernel objects
- C[task:L4::Task] — the task itself

Additional information is:

- First free entry in capability table
- The UTCB area (as flex page)
- First free UTCB (address in the UTCB area)

See also

L4Re::Env, l4re_env_t for more information.

10.49.2 Typedef Documentation

10.49.2.1 typedef struct l4re_env_t l4re_env_t

Initial Environment structure (C version)

See also

Initial environment

10.49.3 Function Documentation

10.49.3.1 l4re_env_t * l4re_env (void) [inline]

Get L4Re initial environment (C version).

Returns

Pointer to L4Re initial environment (C version).

Examples:

examples/sys/aliens/main.c, examples/sys/isr/main.c, examples/sys/singlestep/main.c, examples/sys/start-with-exc/main.c, and examples/sys/utcb-ipc/main.c.

Definition at line 185 of file env.h.

Referenced by l4re_env_get_cap().

Here is the caller graph for this function:

10.49.3.2 l4_cap_idx_t l4re_env_get_cap (char const * *name*) [inline]

Get the capability selector for the object named *name*.

Parameters

name	is the name of the object to lookup in the initial objects.

Returns

A valid capability selector if the object exists or an invalid capability selector if not (l4_is_invalid_cap()).

Examples:

examples/sys/isr/main.c.

Definition at line 227 of file env.h.

References l4re_env(), and l4re_env_get_cap_e().

Here is the call graph for this function:

10.49.3.3 l4_cap_idx_t l4re_env_get_cap_e (char const * *name*, l4re_env_t const * *e*) [inline]

Get the capability selector for the object named *name*.

Parameters

name	is the name of the object to lookup in the initial objects.
e	is the environment structure to use for the operation.

Returns

A valid capability selector if the object exists or an invalid capability selector if not (l4_is_invalid_cap()).

Definition at line 214 of file env.h.

References l4re_env_cap_entry_t::cap, L4_INVALID_CAP, and l4re_env_get_cap_l().

Referenced by l4re_env_get_cap().

Here is the call graph for this function:

Here is the caller graph for this function:

10.49.3.4 l4re_env_cap_entry_t const * l4re_env_get_cap_l (char const * *name*, unsigned *l*, l4re_env_t const * *e*) [inline]

Get the full l4re_env_cap_entry_t for the object named *name*.

Parameters

name	is the name of the object to lookup in the initial objects.
l	is the length of the name string, thus *name* might not be zero terminated.
e	is the environment structure to use for the operation.

Returns

A pointer to an l4re_env_cap_entry_t if the object exists or NULL if not.

Definition at line 196 of file env.h.

References l4re_env_cap_entry_t::flags, and l4re_env_cap_entry_t::name.

Referenced by L4Re::Env::get(), and l4re_env_get_cap_e().

Here is the caller graph for this function:

10.49.3.5 l4_kernel_info_t * l4re_kip (void) [inline]

Get Kernel Info Page.

Returns

Pointer to Kernel Info Page (KIP) structure.

Examples:

examples/sys/aliens/main.c, and examples/sys/ux-vhw/main.c.

Definition at line 189 of file env.h.

Referenced by L4::Poll_timeout_kipclock::set(), and L4::Poll_timeout_kipclock::test().

Here is the caller graph for this function:

10.50 Integer Types

`#include<l4/sys/l4int.h>`

Collaboration diagram for Integer Types:

Files

- file l4int.h

  Fixed sized integer types, generic version.
- file l4int.h

  Fixed sized integer types, arm version.
- file l4int.h

  Fixed sized integer types, amd64 version.
- file l4int.h

  Fixed sized integer types, x86 version.

Macros

- #define L4_MWORD_BITS 32

  Size of machine words in bits.
- #define L4_MWORD_BITS 64

  Size of machine words in bits.
- #define L4_MWORD_BITS 32

  Size of machine words in bits.

Typedefs

- typedef signed char l4_int8_t

  Signed 8bit value.
- typedef unsigned char l4_uint8_t

  Unsigned 8bit value.
- typedef signed short int l4_int16_t

  Signed 16bit value.
- typedef unsigned short int l4_uint16_t

  Unsigned 16bit value.
- typedef signed int l4_int32_t

  Signed 32bit value.
- typedef unsigned int l4_uint32_t

  Unsigned 32bit value.
- typedef signed long long l4_int64_t

  Signed 64bit value.

- typedef unsigned long long l4_uint64_t
  Unsigned 64bit value.
- typedef unsigned long l4_addr_t
  Address type.
- typedef signed long l4_mword_t
  Signed machine word.
- typedef unsigned long l4_umword_t
  Unsigned machine word.
- typedef l4_uint64_t l4_cpu_time_t
  CPU clock type.
- typedef l4_uint64_t l4_kernel_clock_t
  Kernel clock type.
- typedef unsigned int l4_size_t
  Unsigned size type.
- typedef signed int l4_ssize_t
  Signed size type.
- typedef unsigned long l4_size_t
  Unsigned size type.
- typedef signed long l4_ssize_t
  Signed size type.
- typedef unsigned int l4_size_t
  Unsigned size type.
- typedef signed int l4_ssize_t
  Signed size type.

10.50.1 Detailed Description

```
#include<l4/sys/l4int.h>
```

10.50.2 Typedef Documentation

10.50.2.1 typedef signed short int l4_int16_t

Signed 16bit value.

Definition at line 37 of file l4int.h.

10.50.2.2 typedef signed int l4_int32_t

Signed 32bit value.

Definition at line 39 of file l4int.h.

10.50.2.3 typedef signed long long l4_int64_t

Signed 64bit value.

Definition at line 41 of file l4int.h.

10.50.2.4 typedef signed char l4_int8_t

Signed 8bit value.

Definition at line 35 of file l4int.h.

10.50.2.5 typedef unsigned short int l4_uint16_t

Unsigned 16bit value.

Definition at line 38 of file l4int.h.

10.50.2.6 typedef unsigned int l4_uint32_t

Unsigned 32bit value.

Definition at line 40 of file l4int.h.

10.50.2.7 typedef unsigned long long l4_uint64_t

Unsigned 64bit value.

Definition at line 42 of file l4int.h.

10.50.2.8 typedef unsigned char l4_uint8_t

Unsigned 8bit value.

Definition at line 36 of file l4int.h.

10.51 Interface for asynchronous ISR handlers with a given IRQ capability.

This group is just an enhanced version to l4irq_request() which takes a capability object instead of a plain number.

Collaboration diagram for Interface for asynchronous ISR handlers with a given IRQ capability.:

Functions

- l4irq_t ∗ l4irq_request_cap (l4_cap_idx_t irqcap, void(∗isr_handler)(void ∗), void ∗isr_data, int irq_thread_↩ prio, unsigned mode)

  Attach asychronous ISR handler to IRQ.

10.51.1 Detailed Description

This group is just an enhanced version to l4irq_request() which takes a capability object instead of a plain number.

10.51.2 Function Documentation

10.51.2.1 l4irq_t∗ l4irq_request_cap (l4_cap_idx_t *irqcap,* void(∗)(void ∗) *isr_handler,* void ∗ *isr_data,* int *irq_thread_prio,* unsigned *mode*)

Attach asychronous ISR handler to IRQ.

Parameters

irqcap	IRQ capability
isr_handler	Handler routine that is called when an interrupt triggers
isr_data	Pointer given as argument to isr_handler
irq_thread_prio	L4 thread priority of the ISR handler. Give -1 for same priority as creator.
mode	Interrupt type,

See also

L4_irq_mode

Returns

Pointer to l4irq_t structure, 0 on error

10.52 Interface for asynchronous ISR handlers.

This interface has just two (main) functions.

Collaboration diagram for Interface for asynchronous ISR handlers.:

Modules

- Interface for asynchronous ISR handlers with a given IRQ capability.

  This group is just an enhanced version to l4irq_request() which takes a capability object instead of a plain number.

Functions

- l4irq_t ∗ l4irq_request (int irqnum, void(∗isr_handler)(void ∗), void ∗isr_data, int irq_thread_prio, unsigned mode)

  Attach asychronous ISR handler to IRQ.
- long l4irq_release (l4irq_t ∗irq)

  Release asynchronous ISR handler and free resources.

10.52.1 Detailed Description

This interface has just two (main) functions.

l4irq_request to install a handler for an interrupt and l4irq_release to uninstall the handler again and release all resources associated with it.

10.52.2 Function Documentation

10.52.2.1 long l4irq_release (l4irq_t ∗ *irq*)

Release asynchronous ISR handler and free resources.

Parameters

irq	IRQ data structure

Returns

0 sucess, != 0 failure

Examples:

examples/libs/libirq/async_isr.c.

10.52.2.2 l4irq_t∗ l4irq_request (int *irqnum,* void(∗)(void ∗) *isr_handler,* void ∗ *isr_data,* int *irq_thread_prio,* unsigned *mode*)

Attach asychronous ISR handler to IRQ.

Parameters

irqnum	IRQ number to request
isr_handler	Handler routine that is called when an interrupt triggers
isr_data	Pointer given as argument to isr_handler
irq_thread_prio	L4 thread priority of the ISR handler. Give -1 for same priority as creator.
mode	Interrupt type,

See also

L4_irq_mode

Returns

Pointer to l4irq_t structure, 0 on error

Examples:

examples/libs/libirq/async_isr.c.

10.53 Interface using direct functionality.

Collaboration diagram for Interface using direct functionality.:

Functions

- l4irq_t ∗ l4irq_attach (int irqnum)

  Attach/connect to IRQ.
- l4irq_t ∗ l4irq_attach_ft (int irqnum, unsigned mode)

  Attach/connect to IRQ using given type.
- l4irq_t ∗ l4irq_attach_thread (int irqnum, l4_cap_idx_t to_thread)

  Attach/connect to IRQ.
- l4irq_t ∗ l4irq_attach_thread_ft (int irqnum, l4_cap_idx_t to_thread, unsigned mode)

  Attach/connect to IRQ using given type.
- long l4irq_wait (l4irq_t ∗irq)

  Wait for specified IRQ.
- long l4irq_unmask_and_wait_any (l4irq_t ∗unmask_irq, l4irq_t ∗∗ret_irq)

  Unmask a specific IRQ and wait for any attached IRQ.
- long l4irq_wait_any (l4irq_t ∗∗irq)

  Wait for any attached IRQ.
- long l4irq_unmask (l4irq_t ∗irq)

  Unmask a specific IRQ.
- long l4irq_detach (l4irq_t ∗irq)

  Detach from IRQ.

10.53.1 Detailed Description

10.53.2 Function Documentation

10.53.2.1 l4irq_t∗ l4irq_attach (int *irqnum*)

Attach/connect to IRQ.

Parameters

irqnum	IRQ number to request

Returns

Pointer to l4irq_t structure, 0 on error

This l4irq_attach has to be called in the same thread as l4irq_wait and caller has to be a pthread thread.

Examples:

examples/libs/libirq/loop.c.

10.53.2.2 l4irq_t* l4irq_attach_ft (int *irqnum*, unsigned *mode*)

Attach/connect to IRQ using given type.

Parameters

irqnum	IRQ number to request
mode	Interrupt type,

See also

L4_irq_mode

Returns

Pointer to l4irq_t structure, 0 on error

This l4irq_attach has to be called in the same thread as l4irq_wait and caller has to be a pthread thread.

10.53.2.3 l4irq_t* l4irq_attach_thread (int *irqnum*, l4_cap_idx_t *to_thread*)

Attach/connect to IRQ.

Parameters

irqnum	IRQ number to request
to_thread	Attach IRQ to this specified thread.

Returns

Pointer to l4irq_t structure, 0 on error

The pointer to the IRQ structure is used as a label in the IRQ object.

10.53.2.4 l4irq_t* l4irq_attach_thread_ft (int *irqnum*, l4_cap_idx_t *to_thread*, unsigned *mode*)

Attach/connect to IRQ using given type.

Parameters

irqnum	IRQ number to request
to_thread	Attach IRQ to this specified thread.
mode	Interrupt type,

See also

L4_irq_mode

Returns

Pointer to l4irq_t structure, 0 on error

The pointer to the IRQ structure is used as a label in the IRQ object.

10.53.2.5 long l4irq_detach (l4irq_t * *irq*)

Detach from IRQ.

Parameters

irq	IRQ data structure

Returns

0 on success, != 0 on error

10.53.2.6 long l4irq_unmask (l4irq_t * *irq*)

Unmask a specific IRQ.

Parameters

irq	IRQ data structure

Returns

0 on success, != 0 on error

This function is useful if a thread wants to wait for multiple IRQs using l4_ipc_wait.

10.53.2.7 long l4irq_unmask_and_wait_any (l4irq_t * *unmask_irq,* l4irq_t ** *ret_irq*)

Unmask a specific IRQ and wait for any attached IRQ.

Parameters

unmask_irq	IRQ data structure for unmask.

Return values

ret_irq	Received interrupt.

Returns

0 on success, != 0 on error

10.53.2.8 long l4irq_wait (l4irq_t * *irq*)

Wait for specified IRQ.

Parameters

irq	IRQ data structure

Returns

0 on success, != 0 on error

Examples:

examples/libs/libirq/loop.c.

10.53.2.9 long l4irq_wait_any (l4irq_t ** *irq*)

Wait for any attached IRQ.

Return values

irq	Received interrupt.

Returns

0 on success, != 0 on error

10.54 Interface using direct functionality.

Functions

- l4irq_t * l4irq_attach_cap (l4_cap_idx_t irqcap)

  Attach/connect to IRQ.
- l4irq_t * l4irq_attach_cap_ft (l4_cap_idx_t irqcap, unsigned mode)

  Attach/connect to IRQ using given type.
- l4irq_t * l4irq_attach_thread_cap (l4_cap_idx_t irqcap, l4_cap_idx_t to_thread)

  Attach/connect to IRQ.
- l4irq_t * l4irq_attach_thread_cap_ft (l4_cap_idx_t irqcap, l4_cap_idx_t to_thread, unsigned mode)

  Attach/connect to IRQ using given type.

10.54.1 Detailed Description

10.54.2 Function Documentation

10.54.2.1 l4irq_t* l4irq_attach_cap (l4_cap_idx_t *irqcap*)

Attach/connect to IRQ.

Parameters

irqcap	IRQ capability

Returns

Pointer to l4irq_t structure, 0 on error

This l4irq_attach has to be called in the same thread as l4irq_wait and caller has to be a pthread thread.

10.54.2.2 l4irq_t* l4irq_attach_cap_ft (l4_cap_idx_t *irqcap,* unsigned *mode*)

Attach/connect to IRQ using given type.

Parameters

irqcap	IRQ capability
mode	Interrupt type,

See also

L4_irq_mode

Returns

Pointer to l4irq_t structure, 0 on error

This l4irq_attach has to be called in the same thread as l4irq_wait and caller has to be a pthread thread.

10.54.2.3 l4irq_t* l4irq_attach_thread_cap (l4_cap_idx_t *irqcap,* l4_cap_idx_t *to_thread*)

Attach/connect to IRQ.

Parameters

irqcap	IRQ capability
to_thread	Attach IRQ to this thread.

Returns

Pointer to l4irq_t structure, 0 on error

The pointer to the IRQ structure is used as a label in the IRQ object.

10.54.2.4 l4irq_t* l4irq_attach_thread_cap_ft (l4_cap_idx_t *irqcap*, l4_cap_idx_t *to_thread*, unsigned *mode*)

Attach/connect to IRQ using given type.

Parameters

irqcap	IRQ capability
to_thread	Attach IRQ to this thread.
mode	Interrupt type,

See also

L4_irq_mode

Returns

Pointer to l4irq_t structure, 0 on error

The pointer to the IRQ structure is used as a label in the IRQ object.

10.55 Internal Helpers

Collaboration diagram for Internal Helpers:

Data Structures

- struct L4::Types::Bool< V >

  Boolean meta type.
- struct L4::Types::False

  False meta value.
- struct L4::Types::True

  True meta value.
- struct L4::Types::Same< A, B >

  Compare two data types for equality.

10.55.1 Detailed Description

10.56 Internal constants

Internal sigma0 definitions.

Collaboration diagram for Internal constants:

Macros

- #define SIGMA0_REQ_MAGIC ~0xFFUL

  Request magic.
- #define SIGMA0_REQ_MASK ~0xFFUL

  Request mask.
- #define SIGMA0_REQ_ID_MASK 0xF0

  ID mask.
- #define SIGMA0_REQ_ID_FPAGE_RAM 0x60

  RAM.
- #define SIGMA0_REQ_ID_FPAGE_IOMEM 0x70

  I/O memory.
- #define SIGMA0_REQ_ID_FPAGE_IOMEM_CACHED 0x80

  Cached I/O memory.
- #define SIGMA0_REQ_ID_FPAGE_ANY 0x90

  Any.
- #define SIGMA0_REQ_ID_KIP 0xA0

  KIP.
- #define SIGMA0_REQ_ID_TBUF 0xB0

  TBUF.
- #define SIGMA0_REQ_ID_DEBUG_DUMP 0xC0

  Debug dump.
- #define SIGMA0_REQ_ID_NEW_CLIENT 0xD0

  New client.
- #define SIGMA0_IS_MAGIC_REQ(d1) ((d1 & SIGMA0_REQ_MASK) == SIGMA0_REQ_MAGIC)

  Check if magic.
- #define SIGMA0_REQ(x) (SIGMA0_REQ_MAGIC + SIGMA0_REQ_ID_ ## x)

  Construct.
- #define SIGMA0_REQ_FPAGE_RAM (SIGMA0_REQ(FPAGE_RAM))

  RAM.
- #define SIGMA0_REQ_FPAGE_IOMEM (SIGMA0_REQ(FPAGE_IOMEM))

  I/O memory.
- #define SIGMA0_REQ_FPAGE_IOMEM_CACHED (SIGMA0_REQ(FPAGE_IOMEM_CACHED))

  Cache I/O memory.
- #define SIGMA0_REQ_FPAGE_ANY (SIGMA0_REQ(FPAGE_ANY))

  Any.

- #define SIGMA0_REQ_KIP (SIGMA0_REQ(KIP))

  KIP.
- #define SIGMA0_REQ_TBUF (SIGMA0_REQ(TBUF))

  TBUF.
- #define SIGMA0_REQ_DEBUG_DUMP (SIGMA0_REQ(DEBUG_DUMP))

  Debug dump.
- #define SIGMA0_REQ_NEW_CLIENT (SIGMA0_REQ(NEW_CLIENT))

  New client.

10.56.1 Detailed Description

Internal sigma0 definitions.

10.57 Internal functions

Collaboration diagram for Internal functions:

Functions

- void base64_encode (const char ∗infile, unsigned int in_size, char ∗∗outfile)

  base-64-encode string infile
- void base64_decode (const char ∗infile, unsigned int in_size, char ∗∗outfile)

  decode base-64-encoded string infile

10.57.1 Detailed Description

10.58 Interrupt controller

The C Icu interface.

Collaboration diagram for Interrupt controller:

Data Structures

- struct l4_icu_info_t

  Info structure for an ICU.

Typedefs

- typedef struct l4_icu_info_t l4_icu_info_t

  Info structure for an ICU.

Enumerations

- enum L4_icu_flags { L4_ICU_FLAG_MSI }

  Flags for IRQ numbers used for the ICU.

Functions

- l4_msgtag_t l4_icu_bind (l4_cap_idx_t icu, unsigned irqnum, l4_cap_idx_t irq) L4_NOTHROW

  Bind an interrupt line of an interrupt controller to an interrupt object.
- l4_msgtag_t l4_icu_bind_u (l4_cap_idx_t icu, unsigned irqnum, l4_cap_idx_t irq, l4_utcb_t *utcb) L4_NOTHROW

  Bind an interrupt line of an interrupt controller to an interrupt object.
- l4_msgtag_t l4_icu_unbind (l4_cap_idx_t icu, unsigned irqnum, l4_cap_idx_t irq) L4_NOTHROW

  Remove binding of an interrupt line from the interrupt controller object.
- l4_msgtag_t l4_icu_unbind_u (l4_cap_idx_t icu, unsigned irqnum, l4_cap_idx_t irq, l4_utcb_t *utcb) L4_NOTHROW

  Remove binding of an interrupt line from the interrupt controller object.
- l4_msgtag_t l4_icu_set_mode (l4_cap_idx_t icu, unsigned irqnum, l4_umword_t mode) L4_NOTHROW

  Set interrupt mode.
- l4_msgtag_t l4_icu_set_mode_u (l4_cap_idx_t icu, unsigned irqnum, l4_umword_t mode, l4_utcb_t *utcb) L4_NOTHROW

  Set interrupt mode.
- l4_msgtag_t l4_icu_info (l4_cap_idx_t icu, l4_icu_info_t *info) L4_NOTHROW

  Get information about the capabilites of the ICU.
- l4_msgtag_t l4_icu_info_u (l4_cap_idx_t icu, l4_icu_info_t *info, l4_utcb_t *utcb) L4_NOTHROW

  Get information about the capabilites of the ICU.

- l4_msgtag_t l4_icu_msi_info (l4_cap_idx_t icu, unsigned irqnum, l4_uint64_t source, l4_icu_msi_info_t *msi_info) L4_NOTHROW

  Get MSI info about IRQ.
- l4_msgtag_t l4_icu_msi_info_u (l4_cap_idx_t icu, unsigned irqnum, l4_uint64_t source, l4_icu_msi_info_t *msi_info, l4_utcb_t *utcb) L4_NOTHROW

  Get MSI info about IRQ.
- l4_msgtag_t l4_icu_unmask (l4_cap_idx_t icu, unsigned irqnum, l4_umword_t *label, l4_timeout_t to) L4_NOTHROW

  Unmask an IRQ line.
- l4_msgtag_t l4_icu_unmask_u (l4_cap_idx_t icu, unsigned irqnum, l4_umword_t *label, l4_timeout_t to, l4_utcb_t *utcb) L4_NOTHROW

  Acknowledge the given interrupt line.
- l4_msgtag_t l4_icu_mask (l4_cap_idx_t icu, unsigned irqnum, l4_umword_t *label, l4_timeout_t to) L4_NOTHROW

  Mask an IRQ line.
- l4_msgtag_t l4_icu_mask_u (l4_cap_idx_t icu, unsigned irqnum, l4_umword_t *label, l4_timeout_t to, l4_utcb_t *utcb) L4_NOTHROW

  Mask an IRQ line.

10.58.1 Detailed Description

The C Icu interface.

To setup an IRQ line the following steps are required:

1. l4_icu_set_mode() (optional if IRQ has a default mode)
2. l4_irq_attach() to attach the IRQ capability to a thread
3. l4_icu_bind()
4. l4_icu_unmask() to receive the first IRQ

Include File

```
#include <l4/sys/icu.h>
```

10.58.2 Typedef Documentation

10.58.2.1 typedef struct l4_icu_info_t l4_icu_info_t

Info structure for an ICU.

This structure contains information about the features of an ICU.

See also

l4_icu_info().

10.58.3 Enumeration Type Documentation

10.58.3.1 enum L4_icu_flags

Flags for IRQ numbers used for the ICU.

Enumerator

L4_ICU_FLAG_MSI Flag to denote that the IRQ is actually an MSI. This flag may be used for l4_icu_bind() and l4_icu_unbind() functions to denote that the IRQ number is meant to be an MSI.

Definition at line 50 of file icu.h.

10.58.4 Function Documentation

10.58.4.1 l4_msgtag_t l4_icu_bind (l4_cap_idx_t *icu*, unsigned *irqnum*, l4_cap_idx_t *irq*) [inline]

Bind an interrupt line of an interrupt controller to an interrupt object.

Parameters

icu	ICU object to bind `irq` to.
irqnum	IRQ line at the ICU.
irq	IRQ object to bind to this ICU.

Returns

Syscall return tag

Examples:

examples/sys/isr/main.c.

Definition at line 467 of file icu.h.

References l4_icu_bind_u(), and l4_utcb().

Here is the call graph for this function:

10.58.4.2 l4_msgtag_t l4_icu_bind_u (l4_cap_idx_t *icu*, unsigned *irqnum*, l4_cap_idx_t *irq*, l4_utcb_t * *utcb*) [inline]

Bind an interrupt line of an interrupt controller to an interrupt object.

Parameters

icu	The ICU objecte to bind `irq` to.

irqnum	IRQ line at the ICU.
irq	IRQ object for the given IRQ line to bind to this ICU.
utcb	UTCB to be used for this operation, usually the UTCB of the calling thread.

Returns

Syscall return tag

Definition at line 367 of file icu.h.

References L4_FPAGE_RWX, L4_ICU_OP_BIND, l4_ipc_call(), L4_IPC_NEVER, l4_map_obj_control(), l4_msgtag(), l4_obj_fpage(), L4_PROTO_IRQ, l4_msg_regs_t::mr, and l4_fpage_t::raw.

Referenced by L4::Icu::bind(), and l4_icu_bind().

Here is the call graph for this function:

Here is the caller graph for this function:

10.58.4.3 l4_msgtag_t l4_icu_info (l4_cap_idx_t *icu*, l4_icu_info_t * *info*) [inline]

Get information about the capabilites of the ICU.

Parameters

	icu	The ICU object from which information shall be retrieved.
out	*info*	Pointer to an info structure to be filled with information. The memory for this structure has to be allocated by the caller.

Returns

Syscall return tag

Definition at line 475 of file icu.h.

References l4_icu_info_u(), and l4_utcb().

Here is the call graph for this function:

10.58.4.4 l4_msgtag_t l4_icu_info_u (l4_cap_idx_t *icu*, l4_icu_info_t * *info*, l4_utcb_t * *utcb*) [inline]

Get information about the capabilites of the ICU.

Parameters

	icu	The ICU object from which MSI information shall be retrieved.
out	*info*	Info structure to be filled with information.
	utcb	UTCB to be used for this operation, usually the UTCB of the calling thread.

Returns

Syscall return tag

Definition at line 391 of file icu.h.

References L4_ICU_OP_INFO, l4_ipc_call(), L4_IPC_NEVER, l4_msgtag(), L4_PROTO_IRQ, and l4_msg_regs↩_t::mr.

Referenced by L4::Icu::info(), and l4_icu_info().

Here is the call graph for this function:

Here is the caller graph for this function:

10.58.4.5 l4_msgtag_t l4_icu_mask (l4_cap_idx_t *icu*, unsigned *irqnum*, l4_umword_t * *label*, l4_timeout_t *to*) [inline]

Mask an IRQ line.

Parameters

icu	The ICU object where the IRQ line 'irqnum' shall be masked.
irqnum	IRQ line at the ICU.
label	If non-NULL the function also waits for the next message.
to	Timeout for message to ICU, if unsure use L4_IPC_NEVER.

Returns

Syscall return tag

Definition at line 489 of file icu.h.

References l4_utcb().

Here is the call graph for this function:

10.58.4.6 l4_msgtag_t l4_icu_mask_u (l4_cap_idx_t *icu*, unsigned *irqnum*, l4_umword_t * *label*, l4_timeout_t *to*, l4_utcb_t * *utcb*) [inline]

Mask an IRQ line.

Parameters

icu	The ICU object where the IRQ line 'irqnum' shall be masked.
irqnum	IRQ line at the ICU.
label	If NULL this function is a send-only message to the ICU. If not NULL this function will enter an open wait after sending the mask message.
to	The timeout-pair (send and receive) that shall be used for this operation. The receive timeout is used with a non-NULL `label` only.
utcb	UTCB to be used for this operation, usually the UTCB of the calling thread.

Returns

Syscall return tag

Definition at line 454 of file icu.h.

Referenced by L4::Icu::mask().

Here is the caller graph for this function:

10.58.4.7 l4_msgtag_t l4_icu_msi_info (l4_cap_idx_t *icu*, unsigned *irqnum*, l4_uint64_t *source*, l4_icu_msi_info_t * *msi_info*) [inline]

Get MSI info about IRQ.

Parameters

icu	The ICU object from which MSI information shall be retrieved.
irqnum	IRQ line at the ICU.
source	Platform dependent requester ID for MSIs. On IA32 we use a 20bit source filter value as described in the Intel IRQ remapping specification.
msi_info	A l4_icu_msi_info_t structure receiving the address and the data value to trigger this MSI.

Returns

Syscall return tag

Definition at line 479 of file icu.h.

References l4_icu_msi_info_u(), and l4_utcb().

Here is the call graph for this function:

10.58.4.8 l4_msgtag_t l4_icu_msi_info_u (l4_cap_idx_t *icu*, unsigned *irqnum*, l4_uint64_t *source*, l4_icu_msi_info_t * *msi_info*, l4_utcb_t * *utcb*) [inline]

Get MSI info about IRQ.

Parameters

utcb	UTCB to be used for this operation, usually the UTCB of the calling thread.
icu	The ICU object from which MSI information shall be retrieved.
irqnum	IRQ line at the ICU.
source	Platform dependent requester ID for MSIs. On IA32 we use a 20bit source filter value as described in the Intel IRQ remapping specification.
msi_info	A l4_icu_msi_info_t structure receiving the address and the data value to trigger this MSI.

Returns

Syscall return tag

Definition at line 405 of file icu.h.

References L4_ICU_OP_MSI_INFO, l4_ipc_call(), L4_IPC_NEVER, l4_msgtag(), l4_msgtag_has_error(), l4_msgtag_words(), L4_PROTO_IRQ, L4_UNLIKELY, l4_utcb_mr64_idx(), l4_msg_regs_t::mr, and l4_msg_regs_t::mr64.

Referenced by l4_icu_msi_info().

Here is the call graph for this function:

Here is the caller graph for this function:

10.58.4.9 l4_msgtag_t l4_icu_set_mode (l4_cap_idx_t *icu*, unsigned *irqnum*, l4_umword_t *mode*) [inline]

Set interrupt mode.

Parameters

icu	The ICU object.
irqnum	IRQ line at the ICU.
mode	Mode, see L4_irq_mode.

Returns

Syscall return tag

Definition at line 494 of file icu.h.

References l4_icu_set_mode_u(), and l4_utcb().

Here is the call graph for this function:

10.58.4.10 l4_msgtag_t l4_icu_set_mode_u (l4_cap_idx_t *icu*, unsigned *irqnum*, l4_umword_t *mode*, l4_utcb_t * *utcb*) [inline]

Set interrupt mode.

Parameters

icu	The ICU object.
irqnum	IRQ line at the ICU.
mode	Mode, see L4_irq_mode.
utcb	UTCB to be used for this operation, usually the UTCB of the calling thread.

Returns

Syscall return tag

Definition at line 428 of file icu.h.

References L4_ICU_OP_SET_MODE, l4_ipc_call(), L4_IPC_NEVER, l4_msgtag(), L4_PROTO_IRQ, and l4_msg_regs_t::mr.

Referenced by l4_icu_set_mode(), and L4::Icu::set_mode().

Here is the call graph for this function:

Here is the caller graph for this function:

10.58.4.11 l4_msgtag_t l4_icu_unbind (l4_cap_idx_t *icu*, unsigned *irqnum*, l4_cap_idx_t *irq*) [inline]

Remove binding of an interrupt line from the interrupt controller object.

Parameters

icu	The ICU object from where the binding shall be removed.
irqnum	IRQ line at the ICU.
irq	IRQ object to remove from the ICU.

Returns

Syscall return tag

Definition at line 471 of file icu.h.

References l4_icu_unbind_u(), and l4_utcb().

Here is the call graph for this function:

10.58.4.12 l4_msgtag_t l4_icu_unbind_u (l4_cap_idx_t *icu*, unsigned *irqnum*, l4_cap_idx_t *irq*, l4_utcb_t * *utcb*) [inline]

Remove binding of an interrupt line from the interrupt controller object.

Parameters

icu	The ICU object from where the binding shall be removed.
irqnum	IRQ line at the ICU.
irq	IRQ object to remove from the ICU.
utcb	UTCB to be used for this operation, usually the UTCB of the calling thread.

Returns

Syscall return tag

Definition at line 379 of file icu.h.

References L4_FPAGE_RWX, L4_ICU_OP_UNBIND, l4_ipc_call(), L4_IPC_NEVER, l4_map_obj_control(), l4_msgtag(), l4_obj_fpage(), L4_PROTO_IRQ, l4_msg_regs_t::mr, and l4_fpage_t::raw.

Referenced by l4_icu_unbind(), and L4::Icu::unbind().

Here is the call graph for this function:

Here is the caller graph for this function:

10.58.4.13 l4_msgtag_t l4_icu_unmask (l4_cap_idx_t *icu*, unsigned *irqnum*, l4_umword_t * *label*, l4_timeout_t *to*) [inline]

Unmask an IRQ line.

Parameters

icu	The ICU object where the IRQ line shall be unmasked.
irqnum	IRQ line at the ICU.
label	If non-NULL the function also waits for the next message.
to	Timeout for message to ICU, if unsure use L4_IPC_NEVER.

Returns

Syscall return tag, the error values therein are undefined because l4_icu_unmask() is a sender-only IPC.

Definition at line 484 of file icu.h.

References l4_utcb().

Here is the call graph for this function:

10.58.4.14 l4_msgtag_t l4_icu_unmask_u (l4_cap_idx_t *icu*, unsigned *irqnum*, l4_umword_t * *label*, l4_timeout_t *to*, l4_utcb_t * *utcb*) [inline]

Acknowledge the given interrupt line.

Parameters

	icu	The ICU object where the IRQ line shall be unmasked.
	irqnum	The interrupt line that shall be acknowledged.
out	*label*	If NULL this is a send-only unmask, if not NULL then this operation enters an open wait and the *protected label* shall be received here.
	to	The timeout-pair (send and receive) that shall be used for this operation. The receive timeout is used with a non-NULL `label` only.
	utcb	UTCB to be used for this operation, usually the UTCB of the calling thread.

Definition at line 459 of file icu.h.

10.59 Kernel Debugger

Kernel debugger related functionality.

Collaboration diagram for Kernel Debugger:

Macros

- #define enter_kdebug(text)

  Enter L4 kernel debugger.
- #define asm_enter_kdebug(text)

  Enter L4 kernel debugger (plain assembler version)
- #define kd_display(text)

  Show message with L4 kernel debugger, but do not enter debugger.
- #define ko(c)

  Output character with L4 kernel debugger.
- #define enter_kdebug(text)

  Enter L4 kernel debugger.
- #define asm_enter_kdebug(text)

  Enter L4 kernel debugger (plain assembler version)
- #define kd_display(text)

  Show message with L4 kernel debugger, but do not enter debugger.
- #define ko(c)

  Output character with L4 kernel debugger.

Functions

- l4_msgtag_t l4_debugger_set_object_name (l4_cap_idx_t cap, const char ∗name) L4_NOTHROW

  Set the name of a kernel object.
- unsigned long l4_debugger_global_id (l4_cap_idx_t cap) L4_NOTHROW

  Get the globally unique ID of the object behind a capability.
- unsigned long l4_debugger_kobj_to_id (l4_cap_idx_t cap, l4_addr_t kobjp) L4_NOTHROW

  Get the globally unique ID of the object behind the kobject pointer.
- void outchar (char c) L4_NOTHROW

  Print character.
- void outstring (const char ∗text) L4_NOTHROW

  Print character string.
- void outnstring (char const ∗text, unsigned len) L4_NOTHROW

  Print character string.
- void outhex32 (int number) L4_NOTHROW

  Print 32 bit number (hexadecimal)
- void outhex20 (int number) L4_NOTHROW

Print 20 bit number (hexadecimal)

- void outhex16 (int number) L4_NOTHROW

  Print 16 bit number (hexadecimal)

- void outhex12 (int number) L4_NOTHROW

  Print 12 bit number (hexadecimal)

- void outhex8 (int number) L4_NOTHROW

  Print 8 bit number (hexadecimal)

- void outdec (int number) L4_NOTHROW

  Print number (decimal)

- char l4kd_inchar (void) L4_NOTHROW

  Read character from console, non blocking.

10.59.1 Detailed Description

Kernel debugger related functionality.

Attention

This API is subject to change!

This is a debugging facility, any call to any function might be invalid. Do not rely on it in any real code.

Include File

```
#include <l4/sys/debugger.h>
```

10.59.2 Macro Definition Documentation

10.59.2.1 #define asm_enter_kdebug(*text*)

Value:

```
"int  $3  \n\t"\
    "jmp  1f  \n\t"\
    ".ascii \"" text "\"\n\t"\
    "1:     \n\t"
```

Enter L4 kernel debugger (plain assembler version)

Parameters

text	Text to be shown at kernel debugger prompt

Definition at line 63 of file kdebug.h.

10.59.2.2 #define asm_enter_kdebug(*text*)

Value:

```
"int  $3  \n\t"\
    "jmp  1f  \n\t"\
    ".ascii \"" text "\"\n\t"\
    "1:     \n\t"
```

Enter L4 kernel debugger (plain assembler version)

Parameters

text	Text to be shown at kernel debugger prompt

Definition at line 63 of file kdebug.h.

10.59.2.3 #define enter_kdebug(*text*)

Value:

```
asm(\
    "int  $3  \n\t"\
    "jmp  1f  \n\t"\
    ".ascii \"" text "\"\n\t"\
    "1:     \n\t"\
    )
```

Enter L4 kernel debugger.

Parameters

text	Text to be shown at kernel debugger prompt

Examples:

examples/sys/singlestep/main.c.

Definition at line 41 of file kdebug.h.

Referenced by fiasco_tbuf_get_status_phys().

10.59.2.4 #define enter_kdebug(*text*)

Value:

```
asm(\
    "int  $3  \n\t"\
    "jmp  1f  \n\t"\
    ".ascii \"" text "\"\n\t"\
    "1:     \n\t"\
    )
```

Enter L4 kernel debugger.

Parameters

text	Text to be shown at kernel debugger prompt

Definition at line 41 of file kdebug.h.

10.59.2.5 #define kd_display(*text*)

Value:

```
asm(\
    "int  $3  \n\t"\
    "nop    \n\t"\
    "jmp  1f  \n\t"\
    ".ascii \"" text "\"\n\t"\
    "1:     \n\t"\
    )
```

Show message with L4 kernel debugger, but do not enter debugger.

Parameters

text	Text to be shown

Definition at line 76 of file kdebug.h.

10.59.2.6 #define kd_display(*text*)

Value:

```
asm(\
    "int  $3  \n\t"\
    "nop    \n\t"\
    "jmp  1f  \n\t"\
    ".ascii \"" text "\"\n\t"\
    "1:     \n\t"\
    )
```

Show message with L4 kernel debugger, but do not enter debugger.

Parameters

text	Text to be shown

Definition at line 76 of file kdebug.h.

10.59.2.7 #define ko(*c*)

Value:

```
asm(            \
      "int  $3  \n\t"     \
      "cmpb %0,%%al \n\t"     \
      : /* No output */       \
      : "N" (c)         \
      )
```

Output character with L4 kernel debugger.

Parameters

c	Character to be shown

Definition at line 92 of file kdebug.h.

10.59.2.8 #define ko(*c*)

Value:

```
asm(            \
      "int  $3  \n\t"     \
      "cmpb %0,%%al \n\t"     \
      : /* No output */       \
      : "N" (c)         \
      )
```

Output character with L4 kernel debugger.

Parameters

c	Character to be shown

Definition at line 92 of file kdebug.h.

10.59.3 Function Documentation

10.59.3.1 unsigned long l4_debugger_global_id (l4_cap_idx_t *cap*) [inline]

Get the globally unique ID of the object behind a capability.

Parameters

cap	Capability

Return values

~0UL	Capability is not valid.
>=0	Global debugger id.

This is a debugging facility, the call might be invalid.

Definition at line 331 of file debugger.h.

References l4_utcb().

Here is the call graph for this function:

10.59.3.2 unsigned long l4_debugger_kobj_to_id (l4_cap_idx_t *cap*, l4_addr_t *kobjp*) [inline]

Get the globally unique ID of the object behind the kobject pointer.

Parameters

cap	Capability
kobjp	Kobject pointer

Return values

~0UL	The capability or the kobject pointer are invalid.
>=0	The globally unique id.

This is a debugging facility, the call might be invalid.

Definition at line 337 of file debugger.h.

References l4_utcb().

Here is the call graph for this function:

10.59.3.3 l4_msgtag_t l4_debugger_set_object_name (l4_cap_idx_t *cap*, const char * *name*) [inline]

Set the name of a kernel object.

Parameters

cap	Capability which refers to the kernel object.
name	Name of the kernel object that is e.g. displayed in the kernel debugger.

This is a debugging facility, the call might be invalid.

Examples:

examples/sys/aliens/main.c.

Definition at line 324 of file debugger.h.

References l4_utcb().

Here is the call graph for this function:

10.59.3.4 char l4kd_inchar (void) [inline]

Read character from console, non blocking.

Returns

Input character, -1 if no character to read

Definition at line 369 of file kdebug.h.

10.59.3.5 void outchar (char *c*) [inline]

Print character.

Parameters

c	Character

Definition at line 263 of file kdebug.h.

10.59.3.6 void outdec (int *number*) [inline]

Print number (decimal)

Parameters

number	Number

Definition at line 358 of file kdebug.h.

10.59.3.7 void outhex12 (int *number*) [inline]

Print 12 bit number (hexadecimal)

Parameters

number	12 bit number

Definition at line 336 of file kdebug.h.

10.59.3.8 void outhex16 (int *number*) [inline]

Print 16 bit number (hexadecimal)

Parameters

number	16 bit number

Definition at line 325 of file kdebug.h.

10.59.3.9 void outhex20 (int *number*) [inline]

Print 20 bit number (hexadecimal)

Parameters

number	20 bit number

Definition at line 314 of file kdebug.h.

10.59.3.10 void outhex32 (int *number*) [inline]

Print 32 bit number (hexadecimal)

Parameters

number	32 bit number

Definition at line 303 of file kdebug.h.

10.59.3.11 void outhex8 (int *number*) [inline]

Print 8 bit number (hexadecimal)

Parameters

number	8 bit number

Definition at line 347 of file kdebug.h.

10.59.3.12 void outnstring (char const * *text*, unsigned *len*) [inline]

Print character string.

Parameters

text	Character string
len	Number of characters
text	String
len	Number of characters

Examples:

examples/sys/aliens/main.c.

Definition at line 288 of file kdebug.h.

10.59.3.13 void outstring (const char * *text*) [inline]

Print character string.

Parameters

text	Character string
text	String

Examples:

examples/sys/aliens/main.c.

Definition at line 275 of file kdebug.h.

10.60 Kernel Interface Page

Kernel Interface Page.

Collaboration diagram for Kernel Interface Page:

Modules

- Fiasco-UX Virtual devices

  Virtual hardware devices, provided by Fiasco-UX.
- Memory descriptors (C version)

  C Interface for KIP memory descriptors.

Data Structures

- class L4::Kip::Mem_desc

  Memory descriptors stored in the kernel interface page.

Macros

- #define L4_KERNEL_INFO_MAGIC (0x4BE6344CL) /∗ "L4µK" ∗/

  Kernel Info Page identifier ("L4µK").

Enumerations

- enum L4::Kip::Mem_desc::Mem_type

  Memory types.

Functions

- l4_umword_t l4_kip_version (l4_kernel_info_t ∗kip) L4_NOTHROW

  Get the kernel version.
- const char ∗ l4_kip_version_string (l4_kernel_info_t ∗kip) L4_NOTHROW

  Get the kernel version string.
- int l4_kernel_info_version_offset (l4_kernel_info_t ∗kip) L4_NOTHROW

  Return offset in bytes of version_strings relative to the KIP base.
- l4_cpu_time_t l4_kip_clock (l4_kernel_info_t ∗kip) L4_NOTHROW

  Return clock value from the KIP.
- l4_umword_t l4_kip_clock_lw (l4_kernel_info_t ∗kip) L4_NOTHROW

  Return least significant machine word of clock value from the KIP.

10.60.1 Detailed Description

Kernel Interface Page.

C interface for the Kernel Interface Page:
`#include <l4/sys/kip.h>`

C++ interface for the Kernel Interface Page:
`#include <l4/sys/kip>`

10.60.2 Function Documentation

10.60.2.1 int l4_kernel_info_version_offset (l4_kernel_info_t * *kip*) [inline]

Return offset in bytes of version_strings relative to the KIP base.

Parameters

kip	Pointer to the kernel info page (KIP).

Returns

offset of version_strings relative to the KIP base address, in bytes.

Definition at line 130 of file kip.h.

Referenced by l4_kip_version_string().

Here is the caller graph for this function:

10.60.2.2 l4_cpu_time_t l4_kip_clock (l4_kernel_info_t * *kip*) [inline]

Return clock value from the KIP.

Parameters

kip	Pointer to the kernel info page (KIP).

Returns

Value of the clock field in the KIP.

Definition at line 134 of file kip.h.

References l4_mb().

Referenced by L4::Poll_timeout_kipclock::set(), and L4::Poll_timeout_kipclock::test().

Here is the call graph for this function:

Here is the caller graph for this function:

10.60.2.3 l4_umword_t l4_kip_clock_lw (l4_kernel_info_t * *kip*) [inline]

Return least significant machine word of clock value from the KIP.

Parameters

kip	Pointer to the kernel info page (KIP).

Returns

Lower machine word of clock value from the KIP.

Definition at line 155 of file kip.h.

References l4_mb().

Here is the call graph for this function:

10.60.2.4 l4_umword_t l4_kip_version (l4_kernel_info_t * *kip*) [inline]

Get the kernel version.

Parameters

kip	Kernel Info Page.

Returns

Kernel version string. 0 if KIP could not be mapped.

Definition at line 122 of file kip.h.

10.60.2.5 const char * l4_kip_version_string (l4_kernel_info_t * *kip*) [inline]

Get the kernel version string.

Parameters

kip	Kernel Info Page.

Returns

Kernel version string.

Definition at line 126 of file kip.h.

References l4_kernel_info_version_offset().

Here is the call graph for this function:

10.61 Kernel Interface Page API

Collaboration diagram for Kernel Interface Page API:

Files

- file kip.h

Macros

- #define l4util_kip_for_each_feature(s) for (s += strlen(s) + 1; ∗s; s += strlen(s) + 1)

  Cycle through kernel features given in the KIP.

Functions

- int l4util_kip_kernel_is_ux (l4_kernel_info_t ∗)

  Return whether the kernel is running native or under UX.
- int l4util_kip_kernel_has_feature (l4_kernel_info_t ∗, const char ∗str)

  Check if kernel supports a feature.
- unsigned long l4util_kip_kernel_abi_version (l4_kernel_info_t ∗)

  Return kernel ABI version.
- l4_addr_t l4util_memdesc_vm_high (l4_kernel_info_t ∗kinfo)

  Return end of virtual memory.

10.61.1 Detailed Description

10.61.2 Macro Definition Documentation

10.61.2.1 #define l4util_kip_for_each_feature(s) for (s += strlen(s) + 1; ∗s; s += strlen(s) + 1)

Cycle through kernel features given in the KIP.

Cycles through all KIP kernel feature strings. s must be a character pointer (char ∗) initialized with l4util_kip_↩version_string().

Definition at line 74 of file kip.h.

10.61.3 Function Documentation

10.61.3.1 unsigned long l4util_kip_kernel_abi_version (l4_kernel_info_t ∗)

Return kernel ABI version.

Returns

Kernel ABI version.

10.61.3.2 int l4util_kip_kernel_has_feature (l4_kernel_info_t *, const char * *str*)

Check if kernel supports a feature.

Parameters

str	Feature name to check.

Returns

1 if the kernel supports the feature, 0 if not.

Checks the feature field in the KIP for the given string. The KIP will be mapped if not already mapped. The KIP will not be unmapped again.

10.61.3.3 int l4util_kip_kernel_is_ux (l4_kernel_info_t *)

Return whether the kernel is running native or under UX.

Returns whether the kernel is running natively or under UX. The KIP will be mapped if not already mapped. The KIP will not be unmapped again.

Returns

1 when running under UX, 0 if not running under UX

Examples:

examples/sys/ux-vhw/main.c.

10.61.3.4 l4_addr_t l4util_memdesc_vm_high (l4_kernel_info_t * *kinfo*)

Return end of virtual memory.

Returns

0 if memory descriptor could not be found, last address of address space otherwise

10.62 Kernel Objects

API of kernel objects.

Collaboration diagram for Kernel Objects:

Modules

- Factory

  C factory interface to create kernel objects.
- IPC-Gate API

  Secure communication object.
- IRQs

  C IRQ interface.
- Interrupt controller

  The C Icu interface.
- L4 kernel object type information

  Type information for L4 server objects that can be called via IPC.
- Platform Control C API

  C interface for controlling platform-wide properties.

- Scheduler

  Scheduler object.
- Task

  C interface of the Task kernel object.
- Thread

  Thread object.
- Virtual Console

  Virtual console for simple character based input and output.
- Virtual Machines

  Virtual Machine API.

Data Structures

- class L4::Kobject

  Base class for all kinds of kernel objects and remote objects, referenced by capabilities.

10.62.1 Detailed Description

API of kernel objects.

Meta interface that shall be implemented by each L4Re object and gives access to the dynamic type information for L4Re objects.

```
#include <l4/sys/kernel_object.h>
```

10.63 Kumem allocator utility

Kumem allocator utility C interface.

Collaboration diagram for Kumem allocator utility:

Functions

- int l4re_util_kumem_alloc (l4_addr_t *mem, unsigned pages_order, l4_cap_idx_t task, l4_cap_idx_t regmgr) L4_NOTHROW

  Allocate state area.

10.63.1 Detailed Description

Kumem allocator utility C interface.

10.63.2 Function Documentation

10.63.2.1 int l4re_util_kumem_alloc (l4_addr_t * *mem*, unsigned *pages_order*, l4_cap_idx_t *task*, l4_cap_idx_t *regmgr*)

Allocate state area.

Return values

mem	Pointer to memory that has been allocated.
pages_order	Size to allocate, in log2 pages.

Parameters

task	Task to use for allocation.
regmgr	Region manager to use for allocation.

Returns

0 for success, error code otherwise

10.64 Kumem utiltities

Collaboration diagram for Kumem utiltities:

Functions

- int L4Re::Util::kumem_alloc (l4_addr_t *mem, unsigned pages_order, L4::Cap< L4::Task > task=L4Re::Env::env() ->task(), L4::Cap< L4Re::Rm > rm=L4Re::Env::env() ->rm()) throw ()

  Allocate state area.

10.64.1 Detailed Description

10.64.2 Function Documentation

10.64.2.1 int L4Re::Util::kumem_alloc (l4_addr_t * *mem*, unsigned *pages_order*, L4::Cap< L4::Task > *task* = L4Re::Env::env() ->task(), L4::Cap< L4Re::Rm > *rm* = L4Re::Env::env() ->rm()) throw)

Allocate state area.

Return values

mem	Pointer to memory that has been allocated.

Parameters

pages_order	Size to allocate, in log2 pages.
task	Task to use for allocation.
rm	Region manager to use for allocation.

Returns

0 for success, error code otherwise

10.65 L4 IPC Opcodes

List of protocol specific opcodes used for communication with L4Re and Kernel objects.

Enumerations

- enum L4_icu_opcode {
  L4_ICU_OP_BIND, L4_ICU_OP_UNBIND, L4_ICU_OP_INFO, L4_ICU_OP_MSI_INFO,
  L4_ICU_OP_UNMASK, L4_ICU_OP_MASK, L4_ICU_OP_SET_MODE }

  Opcodes to the ICU interface.
- enum L4_ipc_gate_ops { L4_IPC_GATE_BIND_OP = 0x10, L4_IPC_GATE_GET_INFO_OP = 0x11 }

  Operations on the IPC-gate.
- enum L4_platform_ctl_ops { L4_PLATFORM_CTL_SYS_SUSPEND_OP = 0UL, L4_PLATFORM_CTL_S↩YS_SHUTDOWN_OP = 1UL, L4_PLATFORM_CTL_CPU_ENABLE_OP = 3UL, L4_PLATFORM_CTL_CP↩U_DISABLE_OP = 4UL }

  Operations on platform-control objects.
- enum L4_task_ops {
  L4_TASK_MAP_OP = 0UL, L4_TASK_UNMAP_OP = 1UL, L4_TASK_CAP_INFO_OP = 2UL, L4_TASK_↩ADD_KU_MEM_OP = 3UL,
  L4_TASK_LDT_SET_X86_OP = 0x11UL }

  Operations on task objects.
- enum L4_thread_ops {
  L4_THREAD_CONTROL_OP = 0UL, L4_THREAD_EX_REGS_OP = 1UL, L4_THREAD_SWITCH_OP = 2↩UL, L4_THREAD_STATS_OP = 3UL,
  L4_THREAD_VCPU_RESUME_OP = 4UL, L4_THREAD_REGISTER_DELETE_IRQ_OP = 5UL, L4_THR↩EAD_MODIFY_SENDER_OP = 6UL, L4_THREAD_VCPU_CONTROL_OP = 7UL ,
  L4_THREAD_X86_GDT_OP = 0x10UL, L4_THREAD_ARM_TPIDRURO_OP = 0x10UL, L4_THREAD_A↩MD64_SET_SEGMENT_BASE_OP = 0x12UL, L4_THREAD_OPCODE_MASK = 0xffff }

  Operations on thread objects.
- enum L4_vcon_ops { L4_VCON_WRITE_OP = 0UL, L4_VCON_READ_OP = 1UL, L4_VCON_SET_ATTR↩_OP = 2UL, L4_VCON_GET_ATTR_OP = 3UL }

  Operations on vcon objects.

10.65.1 Detailed Description

List of protocol specific opcodes used for communication with L4Re and Kernel objects.

10.65.2 Enumeration Type Documentation

10.65.2.1 enum L4_icu_opcode

Opcodes to the ICU interface.

Enumerator

L4_ICU_OP_BIND Bind opcode.

See also

l4_icu_bind()

L4_ICU_OP_UNBIND Unbind opcode.

See also

l4_icu_unbind()

L4_ICU_OP_INFO Info opcode.

See also

l4_icu_info()

L4_ICU_OP_MSI_INFO Msi-info opcode.

See also

l4_icu_msi_info()

L4_ICU_OP_UNMASK Unmask opcode.

See also

l4_icu_unmask()

L4_ICU_OP_MASK Mask opcode.

See also

l4_icu_mask()

L4_ICU_OP_SET_MODE Set-mode opcode.

See also

l4_icu_set_mode()

Definition at line 93 of file icu.h.

10.65.2.2 enum L4_ipc_gate_ops

Operations on the IPC-gate.

Enumerator

L4_IPC_GATE_BIND_OP Bind operation.

L4_IPC_GATE_GET_INFO_OP Info operation.

Definition at line 86 of file ipc_gate.h.

10.65.2.3 enum L4_platform_ctl_ops

Operations on platform-control objects.

See L4_PROTO_PLATFORM_CTL for the protocol type to use for messages to platform-control objects.

Enumerator

L4_PLATFORM_CTL_SYS_SUSPEND_OP Suspend.

L4_PLATFORM_CTL_SYS_SHUTDOWN_OP shutdown/reboot

L4_PLATFORM_CTL_CPU_ENABLE_OP enable an offline CPU

L4_PLATFORM_CTL_CPU_DISABLE_OP disable an online CPU

Definition at line 134 of file platform_control.h.

10.65.2.4 enum L4_task_ops

Operations on task objects.

Enumerator

L4_TASK_MAP_OP Map.

L4_TASK_UNMAP_OP Unmap.

L4_TASK_CAP_INFO_OP Cap info.

L4_TASK_ADD_KU_MEM_OP Add kernel-user memory.

L4_TASK_LDT_SET_X86_OP x86: LDT set

Definition at line 265 of file task.h.

10.65.2.5 enum L4_thread_ops

Operations on thread objects.

Enumerator

L4_THREAD_CONTROL_OP Control operation.

L4_THREAD_EX_REGS_OP Exchange registers operation.

L4_THREAD_SWITCH_OP Do a thread switch.

L4_THREAD_STATS_OP Thread statistics.

L4_THREAD_VCPU_RESUME_OP VCPU resume.

L4_THREAD_REGISTER_DELETE_IRQ_OP Register an IPC-gate deletion IRQ.

L4_THREAD_MODIFY_SENDER_OP Modify all senders IDs that match the given pattern.

L4_THREAD_VCPU_CONTROL_OP Enable / disable VCPU feature.

L4_THREAD_X86_GDT_OP Gdt.

L4_THREAD_ARM_TPIDRURO_OP Set TPIDRURO register.

L4_THREAD_AMD64_SET_SEGMENT_BASE_OP Set segment base.

L4_THREAD_OPCODE_MASK Mask for opcodes.

Definition at line 608 of file thread.h.

10.65.2.6 enum L4_vcon_ops

Operations on vcon objects.

Enumerator

L4_VCON_WRITE_OP Write.

L4_VCON_READ_OP Read.

L4_VCON_SET_ATTR_OP Get console attributes.

L4_VCON_GET_ATTR_OP Set console attributes.

Definition at line 259 of file vcon.h.

10.66 L4 V-BUS functions

Collaboration diagram for L4 V-BUS functions:

Modules

- L4vbus GPIO functions
- L4vbus PCI functions

Data Structures

- class L4vbus::Pm< DEC >

  Power-management API mixin.
- class L4vbus::Device

  Device on a virtual bus (V-BUS)
- class L4vbus::Icu

  V-BUS Interrupt controller API (ICU)
- class L4vbus::Vbus

  The virtual BUS.

Enumerations

- enum L4vbus_dma_domain_assign_flags { L4VBUS_DMAD_UNBIND = 0, L4VBUS_DMAD_BIND = 1, L4VBUS_DMAD_L4RE_DMA_SPACE = 0, L4VBUS_DMAD_KERNEL_DMA_SPACE = 2 }

  Flags for l4vbus_assign_dma_domain().

Functions

- int l4vbus_get_device_by_hid (l4_cap_idx_t vbus, l4vbus_device_handle_t parent, l4vbus_device_handle_t *child, char const *hid, int depth, l4vbus_device_t *devinfo)

  Find a device by the HID.
- int l4vbus_get_next_device (l4_cap_idx_t vbus, l4vbus_device_handle_t parent, l4vbus_device_handle_t *child, int depth, l4vbus_device_t *devinfo)

  Find next child following child.
- int l4vbus_get_resource (l4_cap_idx_t vbus, l4vbus_device_handle_t dev, int res_idx, l4vbus_resource_t *res)

  Iterate over the resources of a device.
- int l4vbus_is_compatible (l4_cap_idx_t vbus, l4vbus_device_handle_t dev, char const *cid)

Check if the given device has a compatibility ID (CID) or HID that matches cid.

- int l4vbus_get_hid (l4_cap_idx_t vbus, l4vbus_device_handle_t dev, char ∗hid, unsigned long max_len)

  Get the HID (hardware identifier) if a device.
- int l4vbus_request_resource (l4_cap_idx_t vbus, l4vbus_resource_t const ∗res, int flags)

  Request a resource of a specific type.
- int l4vbus_assign_dma_domain (l4_cap_idx_t vbus, unsigned domain_id, unsigned flags, l4_cap_idx_t dma↩_space)

  Bind or unbind a kernel DMA space (L4::Task) or a L4Re::Dma_space to a DMA domain.
- int l4vbus_release_resource (l4_cap_idx_t vbus, l4vbus_resource_t const ∗res)

  Release a previously requested resource.
- int l4vbus_vicu_get_cap (l4_cap_idx_t vbus, l4vbus_device_handle_t icu, l4_cap_idx_t cap)

  Get capability of ICU.

10.66.1 Detailed Description

10.66.2 Enumeration Type Documentation

10.66.2.1 enum L4vbus_dma_domain_assign_flags

Flags for l4vbus_assign_dma_domain().

Enumerator

L4VBUS_DMAD_UNBIND Unbind the given DMA space from the DMA domain.

L4VBUS_DMAD_BIND Bind the given DMA space to the DMA domain.

L4VBUS_DMAD_L4RE_DMA_SPACE The given DMA space is an L4Re::Dma_space.

L4VBUS_DMAD_KERNEL_DMA_SPACE The goven DMA space is a kernel DMA space (L4::Task)

Definition at line 120 of file vbus.h.

10.66.3 Function Documentation

10.66.3.1 int l4vbus_assign_dma_domain (l4_cap_idx_t *vbus*, unsigned *domain_id*, unsigned *flags*, l4_cap_idx_t *dma_space*)

Bind or unbind a kernel DMA space (L4::Task) or a L4Re::Dma_space to a DMA domain.

Parameters

vbus	Capability of the system bus
domain_id	DMA domain ID (resource address of DMA domain found on the vBUS).
flags	A combination of L4vbus_dma_domain_assign_flags.
dma_space	The DMA space capability to bind or unbind, this must either be an L4Re::Dma_space or a kernel DMA space (L4::Task created with L4_PROTO_DMA_SPACE) and the type must be reflected in the `flags`.

Return values

0	Operation completed successfully.
-L4_ENOENT	The vbus does not support a global DMA domain or no DMA domain could be found.

-L4_EINVAL	Invalid argument used.
-L4_EBUSY	DMA domain is already active, this means another DMA space is already assigned.

10.66.3.2 int l4vbus_get_device_by_hid (l4_cap_idx_t *vbus*, l4vbus_device_handle_t *parent*, l4vbus_device_handle_t * *child*, char const * *hid*, int *depth*, l4vbus_device_t * *devinfo*)

Find a device by the HID.

Parameters

vbus	Capability of the system bus
parent	Handle to the parent to start the search This function searches the vbus for a device with the given HID and returns a handle to the first matching device. The HID usually conforms to an ACPI HID or a Linux device tree compatible ID.

It is possible to have multiple devices with the same HID on a vbus. In order to find all matching devices this function has to be called repeatedly with `child` pointing to the device found in the previous iteration. The iteration starts at `child` that might be any device node in the tree.

Parameters

`in,out`	*child*	Handle of the device from where in the device tree the search should start. To start searching from the beginning `child` must be initialized using the default (#L4VBUS_NULL). If a matching device is found its handle is returned through this parameter.
	hid	HID of the device
	depth	Maximum depth for the recursive lookup
`out`	*devinfo*	Device information structure (might be NULL)

Return values

>=	0 A device with the given HID was found.
-L4_ENODEV	No device with the given HID could be found on the vbus.
-L4_ENOENT	Invalid or no HID provided.

Referenced by L4vbus::Device::device_by_hid().

Here is the caller graph for this function:

10.66.3.3 int l4vbus_get_hid (l4_cap_idx_t *vbus*, l4vbus_device_handle_t *dev*, char * *hid*, unsigned long *max_len*)

Get the HID (hardware identifier) if a device.

Parameters

vbus	Capability of the system bus
dev	Handle of the device
hid	Pointer to a buffer for the HID string
max_len	The size of the buffer (*hid*)

Returns

the length of the HID string on success, else failure

10.66.3.4 int l4vbus_get_next_device (l4_cap_idx_t *vbus*, l4vbus_device_handle_t *parent*, l4vbus_device_handle_t * *child*, int *depth*, l4vbus_device_t * *devinfo*)

Find next child following *child*.

Parameters

	vbus	Capability of the system bus
	parent	Handle to the parent device (use 0 for the system bus)
	child	Handle to the child device (use 0 to get the first child)
	depth	Depth to look for
out	*devinfo*	device information (might be NULL)

Returns

0 on success, else failure

Referenced by L4vbus::Device::next_device().

Here is the caller graph for this function:

10.66.3.5 int l4vbus_get_resource (l4_cap_idx_t *vbus*, l4vbus_device_handle_t *dev*, int *res_idx*, l4vbus_resource_t * *res*)

Iterate over the resources of a device.

Parameters

	vbus	Capability of the system bus
	dev	Handle of the device

out	*res_idx*	Index of the resource, the number of resources is available in the devinfo from get device functions.
out	*res*	Descriptor of the resource

Returns

0 on success, else failure

Referenced by L4vbus::Device::get_resource().

Here is the caller graph for this function:

10.66.3.6 int l4vbus_is_compatible (l4_cap_idx_t *vbus*, l4vbus_device_handle_t *dev*, char const * *cid*)

Check if the given device has a compatibility ID (CID) or HID that matches *cid*.

Parameters

vbus	Capability of the system bus
dev	device handle for which the CID shall be tested
cid	the compatibility ID to test

Returns

1 when the given ID (*cid*) matches this device, 0 when the given ID does not match, <0 on error.

Referenced by L4vbus::Device::is_compatible().

Here is the caller graph for this function:

10.66.3.7 int l4vbus_release_resource (l4_cap_idx_t *vbus*, l4vbus_resource_t const * *res*)

Release a previously requested resource.

Parameters

vbus	Capability of the system bus.
res	Descriptor of the resource.

Returns

0 on success, else failure

Referenced by L4vbus::Vbus::release_resource().

Here is the caller graph for this function:

10.66.3.8 int l4vbus_request_resource (l4_cap_idx_t *vbus*, l4vbus_resource_t const * *res*, int *flags*)

Request a resource of a specific type.

Parameters

vbus	Capability of the system bus
res	Descriptor of the resource
flags	Optional flags

Returns

0 on success, else failure

If any resource is found that contains the requested type and addresses this resource is returned.

Flags are only relevant to control the memory caching. If io-memory is requested.

Returns

0 on success, else failure

Referenced by L4vbus::Vbus::request_resource().

Here is the caller graph for this function:

10.66.3.9 int l4vbus_vicu_get_cap (l4_cap_idx_t *vbus*, l4vbus_device_handle_t *icu*, l4_cap_idx_t *cap*)

Get capability of ICU.

Parameters

vbus	Capability of the system bus.
icu	ICU device handle.
cap	Capability slot for the capability.

Returns

0 on success, else failure

Referenced by L4vbus::Icu::vicu().

Here is the caller graph for this function:

10.67 L4 VIRTIO Block Device

Collaboration diagram for L4 VIRTIO Block Device:

Data Structures

- struct l4virtio_block_header_t

  Header structure of a request for a block device.
- struct l4virtio_block_config_t

  Device configuration for block devices.

Typedefs

- typedef struct l4virtio_block_header_t l4virtio_block_header_t

  Header structure of a request for a block device.
- typedef struct l4virtio_block_config_t l4virtio_block_config_t

  Device configuration for block devices.

Enumerations

- enum L4virtio_block_operations { L4VIRTIO_BLOCK_T_IN = 0, L4VIRTIO_BLOCK_T_OUT = 1, L4VIRTIO_BLOCK_T_FLUSH = 4 }

  Kinds of operation over a block device.
- enum L4virtio_block_status { L4VIRTIO_BLOCK_S_OK = 0, L4VIRTIO_BLOCK_S_IOERR = 1, L4VIRTIO_BLOCK_S_UNSUPP = 2 }

  Status of a finished block request.

10.67.1 Detailed Description

10.67.2 Enumeration Type Documentation

10.67.2.1 enum L4virtio_block_operations

Kinds of operation over a block device.

Enumerator

L4VIRTIO_BLOCK_T_IN Read from device.

L4VIRTIO_BLOCK_T_OUT Write to device.

L4VIRTIO_BLOCK_T_FLUSH Flush disk.

Definition at line 31 of file virtio_block.h.

10.67.2.2 enum L4virtio_block_status

Status of a finished block request.

Enumerator

L4VIRTIO_BLOCK_S_OK Request finished successfully.

L4VIRTIO_BLOCK_S_IOERR IO error on device.

L4VIRTIO_BLOCK_S_UNSUPP Operation is not supported.

Definition at line 41 of file virtio_block.h.

10.68 L4 VIRTIO Interface

Collaboration diagram for L4 VIRTIO Interface:

Modules

- L4 VIRTIO Block Device
- L4 VIRTIO Transport Layer

  L4 specific VIRTIO Transport layer.

10.68.1 Detailed Description

10.69 L4 VIRTIO Transport Layer

L4 specific VIRTIO Transport layer.

Collaboration diagram for L4 VIRTIO Transport Layer:

Namespaces

- L4virtio

  L4-VIRTIO Transport C++ API.

Data Structures

- class L4virtio::Svr::Dev_config

  Abstraction for L4-Virtio device config memory.
- class L4virtio::Svr::Driver_mem_region_t< DATA >

  Region of driver memory, that shall be managed locally.
- class L4virtio::Svr::Driver_mem_list_t< DATA >

  List of driver memory regions assigned to a single L4-VIRTIO transport instance.
- class L4virtio::Svr::Device_t< DATA >

  Server-side L4-VIRTIO device stub.
- struct L4virtio::Svr::Dev_status

  Type of the device status register.
- struct L4virtio::Svr::Dev_features

  Type for device feature bitmap.
- class L4virtio::Svr::Virtqueue

  Virtqueue implementation for the device.
- class L4virtio::Svr::Virtqueue::Head_desc

  VIRTIO request, essentially a descriptor from the available ring.
- struct L4virtio::Svr::Data_buffer

  Abstract data buffer.
- struct L4virtio::Svr::Bad_descriptor

  Exception used by Queue to indicate descriptor errors.
- class L4virtio::Svr::Request_processor

  Encapsulate the state for processing a VIRTIO request.
- struct l4virtio_config_hdr_t

  L4-VIRTIO config header, provided in shared data space.
- struct l4virtio_config_queue_t

  Queue configuration entry.

Typedefs

- typedef struct l4virtio_config_hdr_t l4virtio_config_hdr_t

  L4-VIRTIO config header, provided in shared data space.
- typedef struct l4virtio_config_queue_t l4virtio_config_queue_t

  Queue configuration entry.

Enumerations

- enum L4virtio::Svr::Bad_descriptor::Error { L4virtio::Svr::Bad_descriptor::Bad_address, L4virtio::Svr::Bad_descriptor::Bad_rights, L4virtio::Svr::Bad_descriptor::Bad_flags, L4virtio::Svr::Bad_descriptor::Bad_next, L4virtio::Svr::Bad_descriptor::Bad_size }

  The error code.
- enum L4_virtio_protocol

  L4-VIRTIO protocol number.
- enum L4_virtio_opcodes { L4VIRTIO_OP_SET_STATUS = 0, L4VIRTIO_OP_CONFIG_QUEUE, L4VIRTIO_OP_REGISTER_IFACE, L4VIRTIO_OP_REGISTER_DS }

  L4-VIRTIO opcodes.
- enum L4virtio_device_ids { L4VIRTIO_ID_NET = 1, L4VIRTIO_ID_BLOCK = 2, L4VIRTIO_ID_CONSOLE = 3, L4VIRTIO_ID_RNG = 4, L4VIRTIO_ID_BALLOON = 5, L4VIRTIO_ID_RPMSG = 7, L4VIRTIO_ID_SCSI = 8, L4VIRTIO_ID_9P = 9, L4VIRTIO_ID_RPROC_SERIAL = 11, L4VIRTIO_ID_CAIF = 12, L4VIRTIO_ID_SOCK = 0x9999 }

  Virtio device IDs as reported in the driver's config space.
- enum L4virtio_device_status { L4VIRTIO_STATUS_ACKNOWLEDGE = 1, L4VIRTIO_STATUS_DRIVER = 2, L4VIRTIO_STATUS_DRIVER_OK = 4, L4VIRTIO_STATUS_FEATURES_OK = 8, L4VIRTIO_STATUS_FAILED = 0x80 }

  Virtio device status bits.
- enum L4_virtio_irq_status { L4VIRTIO_IRQ_STATUS_VRING = 1, L4VIRTIO_IRQ_STATUS_CONFIG = 2 }

  VIRTIO IRQ status codes (l4virtio_config_hdr_t::irq_status).

Functions

- l4virtio_config_queue_t * l4virtio_config_queues (l4virtio_config_hdr_t const *cfg)

  Get the pointer to the first queue config.
- void * l4virtio_device_config (l4virtio_config_hdr_t const *cfg)

  Get the pointer to the device configuration.
- int l4virtio_set_status (l4_cap_idx_t cap, unsigned status) L4_NOTHROW

  Write the VIRTIO status register.
- int l4virtio_config_queue (l4_cap_idx_t cap, unsigned queue) L4_NOTHROW

  Trigger queue configuration of the given queue.
- int l4virtio_register_ds (l4_cap_idx_t cap, l4_cap_idx_t ds_cap, l4_uint64_t base, l4_umword_t offset, l4_umword_t size) L4_NOTHROW

  Register a shared data space with VIRTIO host.
- int l4virtio_register_iface (l4_cap_idx_t cap, l4_cap_idx_t guest_irq, l4_cap_idx_t host_irq, l4_cap_idx_t config_ds) L4_NOTHROW

  Register client to the given L4-VIRTIO host.

10.69.1 Detailed Description

L4 specific VIRTIO Transport layer.

The L4 specific VIRTIO Transport layer is based on L4Re::Dataspace as shared memory and L4::Irq for signaling. The VIRTIO configuration space is mostly based on a shared memory implementation too and accompanied by two IPC functions to synchronize the configuration between device and driver.

10.69.2 Typedef Documentation

10.69.2.1 typedef struct l4virtio_config_queue_t l4virtio_config_queue_t

Queue configuration entry.

An array of such entries is available at the l4virtio_config_hdr_t::queues_offset in the config data space.

10.69.3 Enumeration Type Documentation

10.69.3.1 enum L4virtio::Svr::Bad_descriptor::Error

The error code.

Enumerator

Bad_address Address cannot be translated.

Bad_rights Missing access rights on memory.

Bad_flags Invalid combination of descriptor flags.

Bad_next Invalid next index.

Bad_size Invalid size of memory block.

Definition at line 311 of file virtio.

10.69.3.2 enum L4_virtio_irq_status

VIRTIO IRQ status codes (l4virtio_config_hdr_t::irq_status).

Note

l4virtio_config_hdr_t::irq_status is currently unused.

Enumerator

L4VIRTIO_IRQ_STATUS_VRING VRING IRQ pending flag.

L4VIRTIO_IRQ_STATUS_CONFIG CONFIG IRQ pending flag.

Definition at line 89 of file virtio.h.

10.69.3.3 enum L4_virtio_opcodes

L4-VIRTIO opcodes.

Enumerator

L4VIRTIO_OP_SET_STATUS Set status register in device config.

L4VIRTIO_OP_CONFIG_QUEUE Set queue config in device config.

L4VIRTIO_OP_REGISTER_IFACE Register a transport driver to the device.

L4VIRTIO_OP_REGISTER_DS Register a data space as transport memory.

Definition at line 50 of file virtio.h.

10.69.3.4 enum L4virtio_device_ids

Virtio device IDs as reported in the driver's config space.

Enumerator

L4VIRTIO_ID_NET Virtual ethernet card.

L4VIRTIO_ID_BLOCK General block device.

L4VIRTIO_ID_CONSOLE Simple device for data IO via ports.

L4VIRTIO_ID_RNG Entropy source.

L4VIRTIO_ID_BALLOON Memory balooning device.

L4VIRTIO_ID_RPMSG Device using rpmsg protocol.

L4VIRTIO_ID_SCSI SCSI host device.

L4VIRTIO_ID_9P Device using 9P transport protocol.

L4VIRTIO_ID_RPROC_SERIAL Rproc serial device.

L4VIRTIO_ID_CAIF Device using CAIF network protocol.

L4VIRTIO_ID_SOCK Inofficial socket device.

Definition at line 59 of file virtio.h.

10.69.3.5 enum L4virtio_device_status

Virtio device status bits.

Enumerator

L4VIRTIO_STATUS_ACKNOWLEDGE Guest OS has found device.

L4VIRTIO_STATUS_DRIVER Guest OS knows how to drive device.

L4VIRTIO_STATUS_DRIVER_OK Driver is set up.

L4VIRTIO_STATUS_FEATURES_OK Driver has acknowledged feature set. (virtio 1.0 only)

L4VIRTIO_STATUS_FAILED Fatal error in driver or device.

Definition at line 76 of file virtio.h.

10.69.4 Function Documentation

10.69.4.1 int l4virtio_config_queue (l4_cap_idx_t *cap*, unsigned *queue*)

Trigger queue configuration of the given queue.

Usually all queues are configured when the status is written to running. However, in some cases queues shall be disabled or enabled dynamically, in this case this function triggers a reconfiguration from the shared memory register of the queue config.

Parameters

cap	Capability to the VIRTIO host.
queue	Queue index for the queue to be configured.

Return values

0	on success.
-L4_EIO	The queue's status is invalid.
-L4_ERANGE	The queue index exceeds the number of queues.
-L4_EINVAL	Otherwise.

10.69.4.2 l4virtio_config_queue_t* l4virtio_config_queues (l4virtio_config_hdr_t const * *cfg*) [inline]

Get the pointer to the first queue config.

Parameters

cfg	Pointer to the config header.

Returns

pointer to queue config of queue 0.

Definition at line 148 of file virtio.h.

References l4virtio_config_hdr_t::queues_offset.

10.69.4.3 void* l4virtio_device_config (l4virtio_config_hdr_t const * *cfg*) [inline]

Get the pointer to the device configuration.

Parameters

cfg	Pointer to the config header.

Returns

pointer to device configuration structure.

Definition at line 159 of file virtio.h.

References l4virtio_config_hdr_t::dev_cfg_offset.

10.69.4.4 int l4virtio_register_ds (l4_cap_idx_t *cap*, l4_cap_idx_t *ds_cap*, l4_uint64_t *base*, l4_umword_t *offset*, l4_umword_t *size*)

Register a shared data space with VIRTIO host.

Parameters

cap	Capability to the VIRTIO host
ds_cap	Data-space capability to register. The lower 8 bits determine the rights mask with which the guest's rights are masked during the registration of the dataspace at the VIRTIO host.

base	VIRTIO guest physical start address of shared memory region
offset	Offset within the data space that is attached to the given *base* in the guest physical memory.
size	Size of the memory region in the guest

Returns

0 on success, $<$ 0 on error

10.69.4.5 int l4virtio_register_iface (l4_cap_idx_t *cap,* l4_cap_idx_t *guest_irq,* l4_cap_idx_t *host_irq,* l4_cap_idx_t *config_ds*)

Register client to the given L4-VIRTIO host.

Parameters

cap	Capability to the L4-VIRTIO host
guest_irq	IRQ capability for valid IRQ object for host-to-guest notifications
host_irq	Capability selector for receiving the guest-to-host notifications IRQ capability.
config_ds	Capability for receiving the data-space capability for the shared L4-VIRTIO config data space.

Return values

0	on success.
-L4_EINVAL	The host did not receive the *guest_irq* cap.

10.69.4.6 int l4virtio_set_status (l4_cap_idx_t *cap,* unsigned *status*)

Write the VIRTIO status register.

Note

All other registers are accessed via shared memory.

Parameters

cap	Capability to the VIRTIO host
status	Status word to write to the VIRTIO status.

Returns

0 on success, $<$0 on error.

10.70 L4 kernel object type information

Type information for L4 server objects that can be called via IPC.

Collaboration diagram for L4 kernel object type information:

Data Structures

- struct L4::Type_info

  Dynamic Type Information for L4Re Interfaces.
- class L4::Type_info::Demand

  Data type for expressing the needed receive buffers at the server-side of an interface.
- struct L4::Type_info::Demand_t< CAPS, FLAGS, MEM, PORTS >

  Template type statically describing demand of receive buffers.
- struct L4::Kobject_typeid< T >

  Meta object for handling access to type information of Kobjects.
- class L4::Kobject_t< Derived, Base, PROTO, S_DEMAND >

  Helper class to create an L4Re interface class that is derived from a single base class.
- class L4::Kobject_2t< Derived, Base1, Base2, PROTO, S_DEMAND >

  Helper class to create an L4Re interface class that is derived from two base classes (see `L4::Kobject_t`).
- struct L4::Kobject_3t< Derived, Base1, Base2, Base3, PROTO, S_DEMAND >

  Helper class to create an L4Re interface class that is derived from three base classes (see `L4::Kobject_t`).
- struct L4::Kobject_x< Derived, ARGS >

  Generic Kobject inheritance template.

Enumerations

- enum { L4::Type_info::Demand_t< CAPS, FLAGS, MEM, PORTS >::Caps = CAPS, L4::Type_info::Demand_t< CAPS, FLAGS, MEM, PORTS >::Flags = FLAGS, L4::Type_info::Demand_t< CAPS, FLAGS, MEM, PORTS >::Mem = MEM, L4::Type_info::Demand_t< CAPS, FLAGS, MEM, PORTS >::Ports = PORTS }

Functions

- template<typename T >
  Type_info const * L4::kobject_typeid ()

  Get the L4::Type_info for the L4Re interface given in `T`.

10.70.1 Detailed Description

Type information for L4 server objects that can be called via IPC.

This type information consists of inheritance information, the protocol number assigned to an interface as well as the demand on server-side resources.

10.70.2 Enumeration Type Documentation

10.70.2.1 template<unsigned char CAPS = 0, unsigned char FLAGS = 0, unsigned char MEM = 0, unsigned char PORTS = 0> anonymous enum

Enumerator

Caps number of capability receive buffers.

Flags flags, such as the need for timeouts.

Mem number of memory receive windows.

Ports number of IO-port receive windows.

Definition at line 565 of file __typeinfo.h.

10.70.3 Function Documentation

10.70.3.1 template<typename T > Type_info const* L4::kobject_typeid ( ) [inline]

Get the L4::Type_info for the L4Re interface given in `T`.

Template Parameters

T	The type (L4Re interface) for which the information shall be returned.

Returns

A pointer to the L4::Type_info structure for `T`.

Definition at line 686 of file __typeinfo.h.

References L4::Kobject_typeid< T >::id().

Here is the call graph for this function:

10.71 L4Re C Interface

Documentation for the L4Re C Interface.

Collaboration diagram for L4Re C Interface:

Modules

- Capability allocator

  Capability allocator C interface.
- DMA Space Interface

  DMA Space C interface.

- Dataspace interface

  Dataspace C interface.
- Debug interface
- Event interface

  Event C interface.
- Kumem allocator utility

  Kumem allocator utility C interface.
- L4Re Util C Interface

  Documentation of the L4 Runtime Environment utility functionality in C.
- Log interface

  Log C interface.
- Memory allocator

  Memory allocator C interface.
- Namespace interface

  Namespace C interface.
- Region map interface

  Region map C interface.
- Video API

Functions

- long l4re_inhibitor_acquire (l4_cap_idx_t cap, l4_umword_t id, char const ∗reason)

  Inhibitor C interface.

10.71.1 Detailed Description

Documentation for the L4Re C Interface.

The interface functions closely align with the C++ functions and add no further functionalities.

For new programs it is advised to use the C++ interface.

10.71.2 Function Documentation

10.71.2.1 long l4re_inhibitor_acquire (l4_cap_idx_t *cap*, l4_umword_t *id*, char const ∗ *reason*)

Inhibitor C interface.

Acquire an inhibitor lock.

Parameters

cap	Capability for the Inhibitor object (

See also

L4Re::Inhibitor)

Parameters

id	ID of the inhibitor lock that shall be acquired.

reason	Reason why the inhibitor lock is acquired. (Used for informing the user or debugging.)

Returns

0 for success, <0 on error

See also

L4Re::Inhibitor::acquire()

10.72 L4Re C++ Interface

Documentation of the L4 Runtime Environment C++ API.

Collaboration diagram for L4Re C++ Interface:

Modules

- Auxiliary data
- C++ Exceptions

- Console API

  Console interface.
- Debugging API

  Debugging Interface.
- Event API

  Event interface.
- Initial Environment

  Environment that is initially provided to an L4 task.
- L4Re ELF Auxiliary Information

  API for embedding auxiliary information into binary programs.
- L4Re Protocol identifiers

  L4Re Protocol Constants (C version)
- L4Re Util C++ Interface

  Documentation of the L4 Runtime Environment utility functionality in C++.
- Logging interface

  Interface for log output.
- Name-space API

  API for name spaces that store capabilities.
- Parent API

  Parent interface.
- Region map API

  Virtual address-space management.

10.72.1 Detailed Description

Documentation of the L4 Runtime Environment C++ API.

10.73 L4Re Capability API

Collaboration diagram for L4Re Capability API:

Data Structures

- class L4Re::Cap_alloc

  Capability allocator interface.
- class L4Re::Smart_cap_auto< Unmap_flags >

  Helper for Auto_cap and Auto_del_cap.
- class L4Re::Util::Smart_count_cap< Unmap_flags >

  Helper for Ref_cap and Ref_del_cap.
- struct L4Re::Util::Ref_cap< T >

  Automatic capability that implements automatic free and unmap of the capability selector.
- struct L4Re::Util::Ref_del_cap< T >

  Automatic capability that implements automatic free and unmap+delete of the capability selector.

Functions

- template<typename T >
  Auto_cap< T >::Cap L4Re::Util::make_auto_cap ()

  Allocate a capability slot and wrap it in an Auto_cap.
- template<typename T >
  Auto_del_cap< T >::Cap L4Re::Util::make_auto_del_cap ()

  Allocate a capability slot and wrap it in an Auto_del_cap.
- template<typename T >
  Ref_cap< T >::Cap L4Re::Util::make_ref_cap ()

  Allocate a capability slot and wrap it in a Ref_cap.
- template<typename T >
  Ref_del_cap< T >::Cap L4Re::Util::make_ref_del_cap ()

  Allocate a capability slot and wrap it in a Ref_del_cap.
- virtual L4Re::Cap_alloc::~Cap_alloc ()=0

  Destructor.

Variables

- _Cap_alloc & L4Re::Util::cap_alloc

  Capability allocator.

10.73.1 Detailed Description

10.73.2 Function Documentation

10.73.2.1 template<typename T > Auto_cap<T>::Cap L4Re::Util::make_auto_cap ( )

Allocate a capability slot and wrap it in an Auto_cap.

Template Parameters

T	Type of capability the slot is used for.

Definition at line 284 of file cap_alloc.

10.73.2.2 template<typename T > Auto_del_cap<T>::Cap L4Re::Util::make_auto_del_cap ( )

Allocate a capability slot and wrap it in an Auto_del_cap.

Template Parameters

T	Type of capability the slot is used for.

Definition at line 294 of file cap_alloc.

10.73.2.3 template<typename T > Ref_cap<T>::Cap L4Re::Util::make_ref_cap ( )

Allocate a capability slot and wrap it in a Ref_cap.

Template Parameters

T	Type of capability the slot is used for.

Definition at line 304 of file cap_alloc.

10.73.2.4 template<typename T > Ref_del_cap<T>::Cap L4Re::Util::make_ref_del_cap ( )

Allocate a capability slot and wrap it in a Ref_del_cap.

Template Parameters

T	Type of capability the slot is used for.

Definition at line 313 of file cap_alloc.

10.73.3 Variable Documentation

10.73.3.1 _Cap_alloc& L4Re::Util::cap_alloc

Capability allocator.

This is the instance of the capability allocator that is used by usual applications. The actual implementation of the allocator depends on the configuration of the system.

Per default we use Counting_cap_alloc, a reference-counting capability allocator, that keeps a reference counter for each managed capability selector.

Note

This capability allocator is not thread-safe.

Examples:

examples/libs/l4re/c++/mem_alloc/ma+rm.cc, examples/libs/l4re/c++/shared_ds/ds_clnt.cc, examples/libs/l4re/c++/shared_ds/ds_srv.cc, and examples/libs/l4re/streammap/client.cc.

Referenced by L4Re::Util::Br_manager::alloc_buffer_demand(), L4virtio::Svr::Dev_config::Dev_config(), and L4Re::Util::Br_manager::realloc_rcv_cap().

10.74 L4Re ELF Auxiliary Information

API for embedding auxiliary information into binary programs.

Collaboration diagram for L4Re ELF Auxiliary Information:

Data Structures

- struct l4re_elf_aux_t

  Generic header for each auxiliary vector element.
- struct l4re_elf_aux_vma_t

  Auxiliary vector element for a reserved virtual memory area.
- struct l4re_elf_aux_mword_t

  Auxiliary vector element for a single unsigned data word.

Macros

- #define L4RE_ELF_AUX_ELEM const __attribute__((used, section(".rol4re_elf_aux"), aligned(sizeof(l4_↩umword_t))))

  Define an auxiliary vector element.
- #define L4RE_ELF_AUX_ELEM_T(type, id, tag, val...) static L4RE_ELF_AUX_ELEM type id = {tag, sizeof(type), val}

  Define an auxiliary vector element.

Typedefs

- typedef struct l4re_elf_aux_t l4re_elf_aux_t

  Generic header for each auxiliary vector element.
- typedef struct l4re_elf_aux_vma_t l4re_elf_aux_vma_t

  Auxiliary vector element for a reserved virtual memory area.
- typedef struct l4re_elf_aux_mword_t l4re_elf_aux_mword_t

  Auxiliary vector element for a single unsigned data word.

Enumerations

- enum {
  L4RE_ELF_AUX_T_NONE = 0, L4RE_ELF_AUX_T_VMA, L4RE_ELF_AUX_T_STACK_SIZE, L4RE_ELF↩_AUX_T_STACK_ADDR,
  L4RE_ELF_AUX_T_KIP_ADDR }

10.74.1 Detailed Description

API for embedding auxiliary information into binary programs.

This API allows information for the binary loader to be embedded into a binary application. This information can be reserved areas in the virtual memory of an application and things such as the stack size to be allocated for the first application thread.

10.74.2 Macro Definition Documentation

10.74.2.1 #define L4RE_ELF_AUX_ELEM const __attribute__((used, section(".rol4re_elf_aux"), aligned(sizeof(l4_umword_t))))

Define an auxiliary vector element.

This is the generic method for defining auxiliary vector elements. A more convenient way is to use L4RE_ELF_A↩UX_ELEM_T.

Usage:

```
L4RE_ELF_AUX_ELEM l4re_elf_aux_vma_t decl_name =
  { L4RE_ELF_AUX_T_VMA, sizeof(l4re_elf_aux_vma_t), 0x2000, 0x4000 };
```

Definition at line 52 of file elf_aux.h.

10.74.2.2 #define L4RE_ELF_AUX_ELEM_T(*type, id, tag, val...*) static L4RE_ELF_AUX_ELEM type id = {tag, sizeof(type), val}

Define an auxiliary vector element.

Parameters

type	is the data type for the element (e.g., l4re_elf_aux_vma_t)
id	is the identifier (variable name) for the declaration (the variable is defined with `static` storage class)
tag	is the tag value for the element e.g., L4RE_ELF_AUX_T_VMA
val	are the values to be set in the descriptor

Usage:

```
L4RE_ELF_AUX_ELEM_T(l4re_elf_aux_vma_t, decl_name,
      L4RE_ELF_AUX_T_VMA, 0x2000, 0x4000 };
```

Definition at line 67 of file elf_aux.h.

10.74.3 Enumeration Type Documentation

10.74.3.1 anonymous enum

Enumerator

L4RE_ELF_AUX_T_NONE Tag for an invalid element in the auxiliary vector.

L4RE_ELF_AUX_T_VMA Tag for descriptor for a reserved virtual memory area.

L4RE_ELF_AUX_T_STACK_SIZE Tag for descriptor that defines the stack size for the first application thread.

L4RE_ELF_AUX_T_STACK_ADDR Tag for descriptor that defines the stack address for the first application thread.

L4RE_ELF_AUX_T_KIP_ADDR Tag for descriptor that defines the KIP address for the binaries address space.

Definition at line 70 of file elf_aux.h.

10.75 L4Re Protocol identifiers

L4Re Protocol Constants (C version)

Collaboration diagram for L4Re Protocol identifiers:

Enumerations

- enum L4Re::Dataspace_::Opcodes

  Data-space communication-protocol opcodes.
- enum L4Re::Event_::Opcodes

  Event communication-protocol opcodes.
- enum L4Re::Inhibitor_::Opcodes

  Inhibitor communication-protocol opcodes.
- enum L4Re::Log_::Opcodes

  Logging-service communication-protocol opcodes.
- enum L4Re::Mem_alloc_::Opcodes

  Memory-allocator communication-protocol opcodes.
- enum L4Re::Namespace_::Opcodes

  Name-space communication-protocol opcodes.
- enum L4Re::Parent_::Opcodes

  Parent communication-protocol opcodes.
- enum L4Re::Rm_::Opcodes

  Region-map communication-protocol opcodes.
- enum L4Re::Video::Goos_::Opcodes

  Frame buffer communication-protocol opcodes.

10.75.1 Detailed Description

L4Re Protocol Constants (C version)

Fix L4Re Protocol Constants.

10.76 L4Re Util C Interface

Documentation of the L4 Runtime Environment utility functionality in C.

Collaboration diagram for L4Re Util C Interface:

Documentation of the L4 Runtime Environment utility functionality in C.

The interface functions closely align with the C++ functions and add no further functionalities.

For new programs it is advised to use the C++ interface.

10.77 L4Re Util C++ Interface

Documentation of the L4 Runtime Environment utility functionality in C++.

Collaboration diagram for L4Re Util C++ Interface:

Modules

- Kumem utiltities
- L4Re Capability API

Data Structures

- class L4Re::Smart_cap_auto< Unmap_flags >

  Helper for Auto_cap and Auto_del_cap.
- class L4Re::Util::Cap_alloc_base

  Capability allocator.
- class L4Re::Util::Br_manager

  Buffer-register (BR) manager for L4::Server.
- class L4Re::Util::Counting_cap_alloc< COUNTERTYPE >

  Reference-counting cap allocator.
- class L4Re::Util::Event_buffer_t< PAYLOAD >

  Event_buffer utility class.
- class L4Re::Util::Event_buffer_consumer_t< PAYLOAD >

  An event buffer consumer.
- class L4Re::Util::Vcon_svr< SVR >

  Console server template class.
- class L4Re::Util::Video::Goos_svr

  Goos server class.

10.77.1 Detailed Description

Documentation of the L4 Runtime Environment utility functionality in C++.

10.78 L4vbus GPIO functions

Collaboration diagram for L4vbus GPIO functions:

Data Structures

- class L4vbus::Gpio_pin

  A GPIO pin.
- class L4vbus::Gpio_module

  A Gpio_module groups multiple GPIO pins together.
- struct L4vbus::Gpio_module::Pin_slice

  A slice of the pins provided by this module.

Enumerations

- enum L4vbus_gpio_generic_func { L4VBUS_GPIO_SETUP_INPUT = 0x100, L4VBUS_GPIO_SETUP_O↩UTPUT = 0x200, L4VBUS_GPIO_SETUP_IRQ = 0x300 }

  Constants for generic GPIO functions.
- enum L4vbus_gpio_pull_modes { L4VBUS_GPIO_PIN_PULL_NONE = 0x100, L4VBUS_GPIO_PIN_PUL↩L_UP = 0x200, L4VBUS_GPIO_PIN_PULL_DOWN = 0x300 }

  Constants for generic GPIO pull up/down resistor configuration.

Functions

- int l4vbus_gpio_setup (l4_cap_idx_t vbus, l4vbus_device_handle_t handle, unsigned pin, unsigned mode, int value)

  Configure the function of a GPIO pin.
- int l4vbus_gpio_config_pull (l4_cap_idx_t vbus, l4vbus_device_handle_t handle, unsigned pin, unsigned mode)

  Generic function to set pull up/down mode.
- int l4vbus_gpio_config_pad (l4_cap_idx_t vbus, l4vbus_device_handle_t handle, unsigned pin, unsigned func, unsigned value)

  Hardware specific configuration function.
- int l4vbus_gpio_config_get (l4_cap_idx_t vbus, l4vbus_device_handle_t handle, unsigned pin, unsigned func, unsigned *value)

  Read hardware specific configuration.
- int l4vbus_gpio_get (l4_cap_idx_t vbus, l4vbus_device_handle_t handle, unsigned pin)

  Read value of GPIO input pin.
- int l4vbus_gpio_set (l4_cap_idx_t vbus, l4vbus_device_handle_t handle, unsigned pin, int value)

  Set GPIO output pin.
- int l4vbus_gpio_multi_setup (l4_cap_idx_t vbus, l4vbus_device_handle_t handle, unsigned offset, unsigned mask, unsigned mode, unsigned value)

Configure function of multiple GPIO pins at once.

- int l4vbus_gpio_multi_config_pad (l4_cap_idx_t vbus, l4vbus_device_handle_t handle, unsigned offset, unsigned mask, unsigned func, unsigned value)

  Hardware specific configuration function for multiple GPIO pins.
- int l4vbus_gpio_multi_get (l4_cap_idx_t vbus, l4vbus_device_handle_t handle, unsigned offset, unsigned *data)

  Read values of multiple GPIO pins at once.
- int l4vbus_gpio_multi_set (l4_cap_idx_t vbus, l4vbus_device_handle_t handle, unsigned offset, unsigned mask, unsigned data)

  Set multiple GPIO output pins at once.
- int l4vbus_gpio_to_irq (l4_cap_idx_t vbus, l4vbus_device_handle_t handle, unsigned pin)

  Create IRQ for GPIO pin.

10.78.1 Detailed Description

10.78.2 Enumeration Type Documentation

10.78.2.1 enum L4vbus_gpio_generic_func

Constants for generic GPIO functions.

Enumerator

L4VBUS_GPIO_SETUP_INPUT Set GPIO pin to input.

L4VBUS_GPIO_SETUP_OUTPUT Set GPIO pin to output.

L4VBUS_GPIO_SETUP_IRQ Set GPIO pin to IRQ.

Definition at line 26 of file vbus_gpio.h.

10.78.2.2 enum L4vbus_gpio_pull_modes

Constants for generic GPIO pull up/down resistor configuration.

Enumerator

L4VBUS_GPIO_PIN_PULL_NONE No pull up or pull down resistors.

L4VBUS_GPIO_PIN_PULL_UP enable pull up resistor

L4VBUS_GPIO_PIN_PULL_DOWN enable pull down resistor

Definition at line 36 of file vbus_gpio.h.

10.78.3 Function Documentation

10.78.3.1 int l4vbus_gpio_config_get (l4_cap_idx_t *vbus*, l4vbus_device_handle_t *handle*, unsigned *pin*, unsigned *func*, unsigned * *value*)

Read hardware specific configuration.

Parameters

	vbus	V-BUS capability
	handle	Device handle for the GPIO chip
	pin	GPIO pin number
	func	Hardware specific configuration register to read from. Usually this is an offset to the GPIO chip's base address.
`out`	*value*	The configuration value.

Returns

0 if OK, error code otherwise

Referenced by L4vbus::Gpio_pin::config_get().

Here is the caller graph for this function:

10.78.3.2 int l4vbus_gpio_config_pad (l4_cap_idx_t *vbus*, l4vbus_device_handle_t *handle*, unsigned *pin*, unsigned *func*, unsigned *value*)

Hardware specific configuration function.

Parameters

vbus	V-BUS capability
handle	Device handle for the GPIO chip
pin	GPIO pin number
func	Hardware specific configuration register, usually offset to the GPIO chip's base address
value	Value which is written into the hardware specific configuration register for the specified pin

Returns

0 if OK, error code otherwise

Referenced by L4vbus::Gpio_pin::config_pad().

Here is the caller graph for this function:

10.78.3.3 int l4vbus_gpio_config_pull (l4_cap_idx_t *vbus*, l4vbus_device_handle_t *handle*, unsigned *pin*, unsigned *mode*)

Generic function to set pull up/down mode.

Parameters

vbus	V-BUS capability
handle	Device handle for the GPIO chip
pin	GPIO pin number
mode	mode for pull up/down resistors, see L4vbus_gpio_pull_modes

Returns

0 if OK, error code otherwise

Referenced by L4vbus::Gpio_pin::config_pull().

Here is the caller graph for this function:

10.78.3.4 int l4vbus_gpio_get (l4_cap_idx_t *vbus*, l4vbus_device_handle_t *handle*, unsigned *pin*)

Read value of GPIO input pin.

Parameters

vbus	V-BUS capability
handle	Device handle for the GPIO chip
pin	GPIO pin number to read from

Returns

Value of GPIO pin (usually 0 or 1), negative error code otherwise.

Referenced by L4vbus::Gpio_pin::get().

Here is the caller graph for this function:

10.78.3.5 int l4vbus_gpio_multi_config_pad (l4_cap_idx_t *vbus*, l4vbus_device_handle_t *handle*, unsigned *offset*, unsigned *mask*, unsigned *func*, unsigned *value*)

Hardware specific configuration function for multiple GPIO pins.

Parameters

vbus	V-BUS capability
handle	Device handle for the GPIO chip
offset	Pin corresponding to the LSB in *mask*. Note: allowed may be hardware specific.
mask	Mask of GPIO pins to configure. A bit set to 1 configures this pin. A maximum of 32 pins can be configured at once. The real number depends on the hardware and the driver implementation.
mask	Mask of GPIO pins to configure. A bit set to 1 configures this pin. A maximum of 32 pins can be configured at once. The real number depends on the hardware and the driver implementation.
func	Hardware specific configuration register, usually offset to the GPIO chip's base address.
value	Value which is written into the hardware specific configuration register for the specified pins

Returns

0 if OK, error code otherwise

Referenced by L4vbus::Gpio_module::config_pad().

Here is the caller graph for this function:

10.78.3.6 int l4vbus_gpio_multi_get (l4_cap_idx_t *vbus*, l4vbus_device_handle_t *handle*, unsigned *offset*, unsigned * *data*)

Read values of multiple GPIO pins at once.

Parameters

	vbus	V-BUS capability
	handle	Device handle for the GPIO chip
	offset	Pin corresponding to the LSB in *data*. Note: allowed may be hardware specific.
`out`	*data*	Each bit returns the value (0 or 1) for the corresponding GPIO pin. The value of pins that are not accessible is undefined.

Returns

0 if OK, error code otherwise

Referenced by L4vbus::Gpio_module::get().

Here is the caller graph for this function:

10.78.3.7 int l4vbus_gpio_multi_set (l4_cap_idx_t *vbus*, l4vbus_device_handle_t *handle*, unsigned *offset*, unsigned *mask*, unsigned *data*)

Set multiple GPIO output pins at once.

Parameters

vbus	V-BUS capability
handle	Device handle for the GPIO chip
offset	Pin corresponding to the LSB in *data*. Note: allowed may be hardware specific.
mask	Mask of GPIO pins to set. A bit set to 1 selects this pin. A maximum of 32 pins can be set at once. The real number depends on the hardware and the driver implementation.
data	Each bit corresponds to the GPIO pin in *mask*. The value of each bit is written to the GPIO pin if its bit in *mask* is set.

Returns

0 if OK, error code otherwise

Referenced by L4vbus::Gpio_module::set().

Here is the caller graph for this function:

10.78.3.8 int l4vbus_gpio_multi_setup (l4_cap_idx_t *vbus*, l4vbus_device_handle_t *handle*, unsigned *offset*, unsigned *mask*, unsigned *mode*, unsigned *value*)

Configure function of multiple GPIO pins at once.

Parameters

vbus	V-BUS capability
handle	Device handle for the GPIO chip
offset	Pin corresponding to the LSB in *mask*. Note: allowed may be hardware specific.
mask	Mask of GPIO pins to configure. A bit set to 1 configures this pin. A maximum of 32 pins can be configured at once. The real number depends on the hardware and the driver implementation.
mask	Mask of GPIO pins to configure. A bit set to 1 configures this pin. A maximum of 32 pins can be configured at once. The real number depends on the hardware and the driver implementation.
mode	GPIO function, see L4vbus_gpio_generic_func for generic functions. Hardware specific functions must be provided in the lower 8 bits.
value	Optional value to set the GPIO pins to if they are configured as output pins

Returns

0 if OK, error code otherwise

Referenced by L4vbus::Gpio_module::setup().

Here is the caller graph for this function:

10.78.3.9 int l4vbus_gpio_set (l4_cap_idx_t *vbus*, l4vbus_device_handle_t *handle*, unsigned *pin*, int *value*)

Set GPIO output pin.

Parameters

vbus	V-BUS capability
handle	Device handle for the GPIO chip
pin	GPIO pin number to write to
value	Value to write to the GPIO pin (usually 0 or 1)

Returns

0 if OK, error code otherwise

Referenced by L4vbus::Gpio_pin::set().

Here is the caller graph for this function:

10.78.3.10 int l4vbus_gpio_setup (l4_cap_idx_t *vbus*, l4vbus_device_handle_t *handle*, unsigned *pin*, unsigned *mode*, int *value*)

Configure the function of a GPIO pin.

Parameters

vbus	V-BUS capability
handle	Device handle for the GPIO chip
pin	GPIO pin number
mode	GPIO function, see L4vbus_gpio_generic_func for generic functions. Hardware specific functions must be provided in the lower 8 bits.
value	Optional value to set the GPIO pin to if it is configured as an output pin

Returns

0 if OK, error code otherwise

Referenced by L4vbus::Gpio_pin::setup().

Here is the caller graph for this function:

10.78.3.11 int l4vbus_gpio_to_irq (l4_cap_idx_t *vbus*, l4vbus_device_handle_t *handle*, unsigned *pin*)

Create IRQ for GPIO pin.

Parameters

vbus	V-BUS capability
handle	Device handle for the GPIO chip
pin	GPIO pin to create an IRQ for.

Returns

IRQ number if OK, negative error code otherwise

Referenced by L4vbus::Gpio_pin::to_irq().

Here is the caller graph for this function:

10.79 L4vbus PCI functions

Collaboration diagram for L4vbus PCI functions:

Functions

- int l4vbus_pci_cfg_read (l4_cap_idx_t vbus, l4vbus_device_handle_t handle, l4_uint32_t bus, l4_uint32_t devfn, l4_uint32_t reg, l4_uint32_t *value, l4_uint32_t width)

  Read from the vPCI configuration space using the PCI root bridge.
- int l4vbus_pci_cfg_write (l4_cap_idx_t vbus, l4vbus_device_handle_t handle, l4_uint32_t bus, l4_uint32_t devfn, l4_uint32_t reg, l4_uint32_t value, l4_uint32_t width)

  Write to the vPCI configuration space using the PCI root bridge.
- int l4vbus_pci_irq_enable (l4_cap_idx_t vbus, l4vbus_device_handle_t handle, l4_uint32_t bus, l4_uint32_t devfn, int pin, unsigned char *trigger, unsigned char *polarity)

  Enable PCI interrupt for a specific device using the PCI root bridge.
- int l4vbus_pcidev_cfg_read (l4_cap_idx_t vbus, l4vbus_device_handle_t handle, l4_uint32_t reg, l4_uint32_t *value, l4_uint32_t width)

  Read from the device's vPCI configuration space.
- int l4vbus_pcidev_cfg_write (l4_cap_idx_t vbus, l4vbus_device_handle_t handle, l4_uint32_t reg, l4_uint32_t value, l4_uint32_t width)

  Write to the device's vPCI configuration space.
- int l4vbus_pcidev_irq_enable (l4_cap_idx_t vbus, l4vbus_device_handle_t handle, unsigned char *trigger, unsigned char *polarity)

  Enable the device's PCI interrupt.

10.79.1 Detailed Description

10.79.2 Function Documentation

10.79.2.1 int l4vbus_pci_cfg_read (l4_cap_idx_t *vbus*, l4vbus_device_handle_t *handle*, l4_uint32_t *bus*, l4_uint32_t *devfn*, l4_uint32_t *reg*, l4_uint32_t * *value*, l4_uint32_t *width*)

Read from the vPCI configuration space using the PCI root bridge.

Parameters

	vbus	Capability of the system bus
	handle	Device handle of the PCI root bridge
	bus	Bus number
	devfn	Device id (upper 16bit) and function (lower 16bit)

	reg	Register in configuration space to read
`out`	*value*	Value that has been read
	width	Width to read in bits (e.g. 8, 16, 32)

Returns

0 on success, else failure

Referenced by L4vbus::Pci_host_bridge::cfg_read().

Here is the caller graph for this function:

10.79.2.2 int l4vbus_pci_cfg_write (l4_cap_idx_t *vbus*, l4vbus_device_handle_t *handle*, l4_uint32_t *bus*, l4_uint32_t *devfn*, l4_uint32_t *reg*, l4_uint32_t *value*, l4_uint32_t *width*)

Write to the vPCI configuration space using the PCI root bridge.

Parameters

vbus	Capability of the system bus
handle	Device handle of the PCI root bridge
bus	Bus number
devfn	Device id (upper 16bit) and function (lower 16bit)
reg	Register in configuration space to write
value	Value to write
width	Width to write in bits (e.g. 8, 16, 32)

Returns

0 on success, else failure

Referenced by L4vbus::Pci_host_bridge::cfg_write().

Here is the caller graph for this function:

10.79.2.3 int l4vbus_pci_irq_enable (l4_cap_idx_t *vbus*, l4vbus_device_handle_t *handle*, l4_uint32_t *bus*, l4_uint32_t *devfn*, int *pin*, unsigned char * *trigger*, unsigned char * *polarity*)

Enable PCI interrupt for a specific device using the PCI root bridge.

Parameters

	vbus	Capability of the system bus
	handle	Device handle of the PCI root bridge
	bus	Bus number
	devfn	Device id (upper 16bit) and function (lower 16bit)
	pin	Interrupt pin (normally as reported in configuration register INTR)
out	*trigger*	False if interrupt is level-triggered
out	*polarity*	True if interrupt is of low polarity

Returns

On success: Interrupt line to be used, else failure

Referenced by L4vbus::Pci_host_bridge::irq_enable().

Here is the caller graph for this function:

10.79.2.4 int l4vbus_pcidev_cfg_read (l4_cap_idx_t *vbus*, l4vbus_device_handle_t *handle*, l4_uint32_t *reg*, l4_uint32_t * *value*, l4_uint32_t *width*)

Read from the device's vPCI configuration space.

Parameters

	vbus	Capability of the system bus
	handle	Device handle of the PCI device
	reg	Register in configuration space to read
out	*value*	Value that has been read
	width	Width to read in bits (e.g. 8, 16, 32)

Returns

0 on success, else failure

Referenced by L4vbus::Pci_dev::cfg_read().

Here is the caller graph for this function:

10.79.2.5 int l4vbus_pcidev_cfg_write (l4_cap_idx_t *vbus*, l4vbus_device_handle_t *handle*, l4_uint32_t *reg*, l4_uint32_t *value*, l4_uint32_t *width*)

Write to the device's vPCI configuration space.

Parameters

vbus	Capability of the system bus
handle	Device handle of the PCI device
reg	Register in configuration space to write
value	Value to write
width	Width to write in bits (e.g. 8, 16, 32)

Returns

0 on success, else failure

Referenced by L4vbus::Pci_dev::cfg_write().

Here is the caller graph for this function:

10.79.2.6 int l4vbus_pcidev_irq_enable (l4_cap_idx_t *vbus*, l4vbus_device_handle_t *handle*, unsigned char * *trigger*, unsigned char * *polarity*)

Enable the device's PCI interrupt.

Parameters

	vbus	Capability of the system bus
	handle	Device handle of the PCI device
out	*trigger*	False if interrupt is level-triggered
out	*polarity*	True if interrupt is of low polarity

Returns

On success: Interrupt line to be used, else failure

Referenced by L4vbus::Pci_dev::irq_enable().

Here is the caller graph for this function:

10.80 Log interface

Log C interface.

Collaboration diagram for Log interface:

Functions

- void l4re_log_print (char const *string) L4_NOTHROW

  Write a null terminated string to the default log.
- void l4re_log_printn (char const *string, int len) L4_NOTHROW

  Write a string of a given length to the default log.
- void l4re_log_print_srv (const l4_cap_idx_t logcap, char const *string) L4_NOTHROW

  Write a null terminated string to a log.
- void l4re_log_printn_srv (const l4_cap_idx_t logcap, char const *string, int len) L4_NOTHROW

  Write a string of a given length to a log.

10.80.1 Detailed Description

Log C interface.

10.80.2 Function Documentation

10.80.2.1 void l4re_log_print (char const * *string*) [inline]

Write a null terminated string to the default log.

Parameters

string	Text to print, null terminated.

Returns

0 for success, <0 on error

See also

L4Re::Log::print

Definition at line 99 of file log.h.

References l4re_log_print_srv(), and l4re_env_t::log.

Here is the call graph for this function:

10.80.2.2 void l4re_log_print_srv (const l4_cap_idx_t *logcap*, char const * *string*)

Write a null terminated string to a log.

Parameters

logcap	Log capability (service).
string	Text to print, null terminated.

Returns

0 for success, <0 on error

See also

L4Re::Log::print

Referenced by l4re_log_print().

Here is the caller graph for this function:

10.80.2.3 void l4re_log_printn (char const * *string*, int *len*) [inline]

Write a string of a given length to the default log.

Parameters

string	Text to print, null terminated.
len	Length of string in bytes.

Returns

0 for success, <0 on error

See also

L4Re::Log::printn

Definition at line 105 of file log.h.

References l4re_log_printn_srv(), and l4re_env_t::log.

Here is the call graph for this function:

10.80.2.4 void l4re_log_printn_srv (const l4_cap_idx_t *logcap*, char const * *string*, int *len*)

Write a string of a given length to a log.

Parameters

logcap	Log capability (service).
string	Text to print, null terminated.
len	Length of string in bytes.

Returns

0 for success, <0 on error

See also

L4Re::Log::printn

Referenced by l4re_log_printn().

Here is the caller graph for this function:

10.81 Logging interface

Interface for log output.

Collaboration diagram for Logging interface:

Data Structures

- class L4Re::Log

  Log interface class.

10.81.1 Detailed Description

Interface for log output.

The logging interface provides a facility sending log output. One purpose of the interface is to serialize the output and provide the possibility to tag output sent to a specific log object.

10.82 Low-Level Thread Functions

Collaboration diagram for Low-Level Thread Functions:

10.83 Machine Restarting Function

Collaboration diagram for Machine Restarting Function:

Functions

- void l4util_reboot (void))

  Machine reboot.

10.83.1 Detailed Description

10.84 Memory allocator

Memory allocator C interface.

Collaboration diagram for Memory allocator:

Enumerations

- enum l4re_ma_flags

  Flags for requesting memory at the memory allocator.

Functions

- long l4re_ma_alloc (long size, l4re_ds_t const mem, unsigned long flags) L4_NOTHROW

  Allocate memory.
- long l4re_ma_alloc_align (long size, l4re_ds_t const mem, unsigned long flags, unsigned long align) L4_N↩OTHROW

  Allocate memory.
- long l4re_ma_free (l4re_ds_t const mem) L4_NOTHROW

  Free memory.
- long l4re_ma_alloc_align_srv (l4_cap_idx_t srv, long size, l4re_ds_t const mem, unsigned long flags, unsigned long align) L4_NOTHROW

  Allocate memory.
- long l4re_ma_free_srv (l4_cap_idx_t srv, l4re_ds_t const mem) L4_NOTHROW

  Free memory.

10.84.1 Detailed Description

Memory allocator C interface.

10.84.2 Enumeration Type Documentation

10.84.2.1 enum l4re_ma_flags

Flags for requesting memory at the memory allocator.

See also

L4Re::Mem_alloc::Mem_alloc_flags

Definition at line 42 of file mem_alloc.h.

10.84.3 Function Documentation

10.84.3.1 long l4re_ma_alloc (long *size,* l4re_ds_t const *mem,* unsigned long *flags*) [inline]

Allocate memory.

Parameters

size	Size to be requested in bytes (granularity is (super)pages and the size is rounded up to this granularity).
mem	Capability slot to put the requested dataspace in
flags	Flags, see l4re_ma_flags

Returns

0 on success, <0 on error

See also

L4Re::Mem_alloc::alloc

The memory allocator returns a dataspace.

Note

This function is using the L4Re::Env::env()->mem_alloc() service.

Examples:

examples/libs/l4re/c/ma+rm.c.

Definition at line 153 of file mem_alloc.h.

References l4re_ma_alloc_align_srv(), and l4re_env_t::mem_alloc.

Here is the call graph for this function:

10.84.3.2 long l4re_ma_alloc_align (long *size*, l4re_ds_t const *mem*, unsigned long *flags*, unsigned long *align*) [inline]

Allocate memory.

Parameters

size	Size to be requested in bytes (granularity is (super)pages and the size is rounded up to this granularity).
mem	Capability slot to put the requested dataspace in
flags	Flags, see l4re_ma_flags
align	Log2 alignment of dataspace if supported by allocator, will be at least L4_PAGESHIFT, with Super_pages flag set at least L4_SUPERPAGESHIFT, default 0

Returns

0 on success, <0 on error

See also

L4Re::Mem_alloc::alloc and
l4re_ma_alloc

The memory allocator returns a dataspace.

Note

This function is using the L4Re::Env::env()->mem_alloc() service.

Definition at line 161 of file mem_alloc.h.

References l4re_ma_alloc_align_srv(), and l4re_env_t::mem_alloc.

Here is the call graph for this function:

10.84.3.3 long l4re_ma_alloc_align_srv (l4_cap_idx_t *srv*, long *size*, l4re_ds_t const *mem*, unsigned long *flags*, unsigned long *align*)

Allocate memory.

Parameters

srv	Memory allocator service.
size	Size to be requested.
mem	Capability slot to put the requested dataspace in
flags	Flags, see l4re_ma_flags
align	Log2 alignment of dataspace if supported by allocator, will be at least L4_PAGESHIFT, with Super_pages flag set at least L4_SUPERPAGESHIFT, default 0

Returns

0 on success, <0 on error

See also

L4Re::Mem_alloc::alloc

The memory allocator returns a dataspace.

Referenced by l4re_ma_alloc(), and l4re_ma_alloc_align().

Here is the caller graph for this function:

10.84.3.4 long l4re_ma_free (l4re_ds_t const *mem*) [inline]

Free memory.

Parameters

mem	Dataspace to free.

Returns

0 on success, <0 on error

See also

L4Re::Mem_alloc::free

Note

This function is using the L4Re::Env::env()->mem_alloc() service.

Examples:

examples/libs/l4re/c/ma+rm.c.

Definition at line 169 of file mem_alloc.h.

References l4re_ma_free_srv(), and l4re_env_t::mem_alloc.

Here is the call graph for this function:

10.84.3.5 long l4re_ma_free_srv (l4_cap_idx_t *srv*, l4re_ds_t const *mem*)

Free memory.

Parameters

srv	Memory allocator service.
mem	Dataspace to free.

Returns

0 on success, <0 on error

See also

L4Re::Mem_alloc::free

Referenced by l4re_ma_free().

Here is the caller graph for this function:

10.85 Memory descriptors (C version)

C Interface for KIP memory descriptors.

Collaboration diagram for Memory descriptors (C version):

Data Structures

- struct l4_kernel_info_mem_desc_t

  Memory descriptor data structure.

Typedefs

- typedef struct l4_kernel_info_mem_desc_t l4_kernel_info_mem_desc_t

  Memory descriptor data structure.

Enumerations

- enum l4_mem_type_t { l4_mem_type_undefined = 0x0, l4_mem_type_conventional = 0x1, l4_mem_type_reserved = 0x2, l4_mem↩_type_dedicated = 0x3, l4_mem_type_shared = 0x4, l4_mem_type_bootloader = 0xe, l4_mem_type_archspecific = 0xf }

  Type of a memory descriptor.

Functions

- l4_kernel_info_mem_desc_t * l4_kernel_info_get_mem_descs (l4_kernel_info_t *kip) L4_NOTHROW

  Get pointer to memory descriptors from KIP.
- unsigned l4_kernel_info_get_num_mem_descs (l4_kernel_info_t *kip) L4_NOTHROW

  Get number of memory descriptors in KIP.
- void l4_kernel_info_set_mem_desc (l4_kernel_info_mem_desc_t *md, l4_addr_t start, l4_addr_t end, unsigned type, unsigned virt, unsigned sub_type) L4_NOTHROW

  Populate a memory descriptor.
- l4_umword_t l4_kernel_info_get_mem_desc_start (l4_kernel_info_mem_desc_t *md) L4_NOTHROW

  Get start address of the region described by the memory descriptor.
- l4_umword_t l4_kernel_info_get_mem_desc_end (l4_kernel_info_mem_desc_t *md) L4_NOTHROW

  Get end address of the region described by the memory descriptor.
- l4_umword_t l4_kernel_info_get_mem_desc_type (l4_kernel_info_mem_desc_t *md) L4_NOTHROW

  Get type of the memory region.
- l4_umword_t l4_kernel_info_get_mem_desc_subtype (l4_kernel_info_mem_desc_t *md) L4_NOTHROW

Get sub-type of memory region.

- l4_umword_t l4_kernel_info_get_mem_desc_is_virtual (l4_kernel_info_mem_desc_t *md) L4_NOTHROW

  Get virtual flag of the memory descriptor.

10.85.1 Detailed Description

C Interface for KIP memory descriptors.

```
#include <l4/sys/memdesc.h>
```

This module contains the C functions to access the memory descriptor in the kernel interface page (KIP).

10.85.2 Typedef Documentation

10.85.2.1 typedef struct l4_kernel_info_mem_desc_t l4_kernel_info_mem_desc_t

Memory descriptor data structure.

Note

This data type is opaque, and must be accessed by the accessor functions defined in this module.

10.85.3 Enumeration Type Documentation

10.85.3.1 enum l4_mem_type_t

Type of a memory descriptor.

Enumerator

l4_mem_type_undefined Undefined, unused descriptor.

l4_mem_type_conventional Conventional memory.

l4_mem_type_reserved Reserved memory for kernel etc.

l4_mem_type_dedicated Dedicated memory (some device memory)

l4_mem_type_shared Shared memory (not implemented)

l4_mem_type_bootloader Memory owned by the boot loader.

l4_mem_type_archspecific Architecture specific memory (e.g., ACPI memory)

Definition at line 44 of file memdesc.h.

10.85.4 Function Documentation

10.85.4.1 l4_umword_t l4_kernel_info_get_mem_desc_end (l4_kernel_info_mem_desc_t * *md*) [inline]

Get end address of the region described by the memory descriptor.

Returns

End address.

Definition at line 207 of file memdesc.h.

10.85.4.2 l4_umword_t l4_kernel_info_get_mem_desc_is_virtual (l4_kernel_info_mem_desc_t * *md*) [inline]

Get virtual flag of the memory descriptor.

Returns

1 if region is virtual memory, 0 if region is physical memory

Definition at line 228 of file memdesc.h.

10.85.4.3 l4_umword_t l4_kernel_info_get_mem_desc_start (l4_kernel_info_mem_desc_t * *md*) [inline]

Get start address of the region described by the memory descriptor.

Returns

Start address.

Definition at line 200 of file memdesc.h.

10.85.4.4 l4_umword_t l4_kernel_info_get_mem_desc_subtype (l4_kernel_info_mem_desc_t * *md*) [inline]

Get sub-type of memory region.

Returns

Sub-type.

The sub type is defined for architecture specific memory descriptors (see l4_mem_type_archspecific) and has architecture specific meaning.

Definition at line 221 of file memdesc.h.

10.85.4.5 l4_umword_t l4_kernel_info_get_mem_desc_type (l4_kernel_info_mem_desc_t * *md*) [inline]

Get type of the memory region.

Returns

Type of the region (see l4_mem_type_t).

Definition at line 214 of file memdesc.h.

10.85.4.6 unsigned l4_kernel_info_get_num_mem_descs (l4_kernel_info_t * *kip*) [inline]

Get number of memory descriptors in KIP.

Returns

Number of memory descriptors.

Definition at line 178 of file memdesc.h.

10.85.4.7 void l4_kernel_info_set_mem_desc (l4_kernel_info_mem_desc_t * *md*, l4_addr_t *start*, l4_addr_t *end*, unsigned *type*, unsigned *virt*, unsigned *sub_type*) [inline]

Populate a memory descriptor.

Parameters

md	Pointer to memory descriptor
start	Start of region
end	End of region
type	Type of region
virt	1 if virtual region, 0 if physical region
sub_type	Sub type.

Definition at line 185 of file memdesc.h.

10.86 Memory operations.

Operations for memory access.

Collaboration diagram for Memory operations.:

Enumerations

- enum L4_mem_op_widths { L4_MEM_WIDTH_1BYTE = 0, L4_MEM_WIDTH_2BYTE = 1, L4_MEM_WIDTH_4BYTE = 2 }

  Memory access width definitions.

Functions

- unsigned long l4_mem_read (unsigned long virtaddress, unsigned width)

  Read memory from kernel privilege level.
- void l4_mem_write (unsigned long virtaddress, unsigned width, unsigned long value)

  Write memory from kernel privilege level.

10.86.1 Detailed Description

Operations for memory access.

This modules provides functionality to access user task memory from the kernel. This is needed for some devices that are only accessible from privileged processor mode. Only use this when absolutely required. This functionality is only available on the ARM architecture.

```
#include <l4/sys/mem_op.h>
```

10.86.2 Enumeration Type Documentation

10.86.2.1 enum L4_mem_op_widths

Memory access width definitions.

Enumerator

L4_MEM_WIDTH_1BYTE	Access one byte (8-bit width)
L4_MEM_WIDTH_2BYTE	Access two bytes (16-bit width)
L4_MEM_WIDTH_4BYTE	Access four bytes (32-bit width)

Definition at line 51 of file mem_op.h.

10.86.3 Function Documentation

10.86.3.1 unsigned long l4_mem_read (unsigned long *virtaddress,* unsigned *width*) [inline]

Read memory from kernel privilege level.

Parameters

virtaddress	Virtual address in the calling task.
width	Width of access in bytes in log2,

See also

L4_mem_op_widths

Returns

Read value.

Upon an given invalid address or invalid width value the function does nothing.

Definition at line 141 of file mem_op.h.

10.86.3.2 void l4_mem_write (unsigned long *virtaddress*, unsigned *width*, unsigned long *value*) [inline]

Write memory from kernel privilege level.

Parameters

virtaddress	Virtual address in the calling task.
width	Width of access in bytes in log2 (i.e. allowed values: 0, 1, 2)
value	Value to write.

Upon an given invalid address or invalid width value the function does nothing.

Definition at line 147 of file mem_op.h.

10.87 Memory related

Memory related constants, data types and functions.

Collaboration diagram for Memory related:

Macros

- #define L4_PAGESIZE

  Minimal page size (in bytes).
- #define L4_PAGEMASK

  Mask for the page number.
- #define L4_LOG2_PAGESIZE

  Number of bits used for page offset.
- #define L4_SUPERPAGESIZE

  Size of a large page.
- #define L4_SUPERPAGEMASK

  Mask for the number of a large page.
- #define L4_LOG2_SUPERPAGESIZE

  Number of bits used as offset for a large page.
- #define L4_INVALID_PTR ((void∗)L4_INVALID_ADDR)

  Invalid address as pointer type.
- #define L4_PAGESHIFT 12

  Size of a page, log2-based.
- #define L4_SUPERPAGESHIFT 21

  Size of a large page, log2-based.
- #define L4_PAGESHIFT 12

  Size of a page, log2-based.
- #define L4_SUPERPAGESHIFT 21

  Size of a large page, log2-based.
- #define L4_PAGESHIFT 12

  Size of a page log2-based.
- #define L4_SUPERPAGESHIFT 22

  Size of a large page log2-based.

Enumerations

- enum l4_addr_consts_t { L4_INVALID_ADDR = ∼0UL }

  Address related constants.

Functions

- l4_addr_t l4_trunc_page (l4_addr_t address) L4_NOTHROW

  Round an address down to the next lower page boundary.
- l4_addr_t l4_trunc_size (l4_addr_t address, unsigned char bits) L4_NOTHROW

  Round an address down to the next lower flex page with size bits.
- l4_addr_t l4_round_page (l4_addr_t address) L4_NOTHROW

  Round address up to the next page.
- l4_addr_t l4_round_size (l4_umword_t value, unsigned char bits) L4_NOTHROW

  Round value up to the next alignment with bits size.

10.87.1 Detailed Description

Memory related constants, data types and functions.

10.87.2 Macro Definition Documentation

10.87.2.1 #define L4_LOG2_PAGESIZE

Number of bits used for page offset.

Size of page in log2.

Definition at line 290 of file consts.h.

Referenced by L4Re::Dataspace::map(), L4Re::Dataspace::map_region(), and L4Re::Util::Dataspace_svr::page↩_shift().

10.87.2.2 #define L4_LOG2_SUPERPAGESIZE

Number of bits used as offset for a large page.

Size of large page in log2

Definition at line 316 of file consts.h.

10.87.2.3 #define L4_PAGEMASK

Mask for the page number.

Note

The most significant bits are set.

Definition at line 281 of file consts.h.

Referenced by l4_round_page(), and l4_trunc_page().

10.87.2.4 #define L4_SUPERPAGEMASK

Mask for the number of a large page.

Note

The most significant bits are set.

Definition at line 308 of file consts.h.

10.87.2.5 #define L4_SUPERPAGESIZE

Size of a large page.

A large page is a *super page* on IA32 or a *section* on ARM.

Definition at line 299 of file consts.h.

10.87.3 Enumeration Type Documentation

10.87.3.1 enum l4_addr_consts_t

Address related constants.

Enumerator

L4_INVALID_ADDR Invalid address.

Definition at line 372 of file consts.h.

10.87.4 Function Documentation

10.87.4.1 l4_addr_t l4_round_page (l4_addr_t *address*) [inline]

Round address up to the next page.

The address is rounded up to the next minimal page boundary. On most architectures this is a 4k page. Check L4_PAGESIZE for the minimal page size.

Parameters

address	The address to round up.

Definition at line 354 of file consts.h.

References L4_PAGEMASK, and L4_PAGESIZE.

Referenced by L4virtio::Svr::Driver_mem_region_t< DATA >::Driver_mem_region_t(), L4Re::Dataspace::map(), and L4Re::Dataspace::map_region().

Here is the caller graph for this function:

10.87.4.2 l4_addr_t l4_round_size (l4_umword_t *value*, unsigned char *bits*) [inline]

Round value up to the next alignment with *bits* size.

Parameters

value	The value to round up to the next size-alignment.
bits	The size of the alignment (log2).

Definition at line 365 of file consts.h.

Referenced by cxx::List_alloc::alloc_max(), and L4Re::Dataspace::map_region().

Here is the caller graph for this function:

10.87.4.3 l4_addr_t l4_trunc_page (l4_addr_t *address*) [inline]

Round an address down to the next lower page boundary.

The address is rounded down to the next lower mininmal page boundary. On most architectures this is a 4k page. Check L4_PAGESIZE for the minimal page size.

Parameters

address	The address to round.

Examples:

examples/libs/l4re/c++/mem_alloc/ma+rm.cc, and examples/libs/l4re/c/ma+rm.c.

Definition at line 329 of file consts.h.

References L4_PAGEMASK.

Referenced by l4_touch_ro(), l4_touch_rw(), L4Re::Dataspace::map(), and L4Re::Dataspace::map_region().

Here is the caller graph for this function:

10.87.4.4 l4_addr_t l4_trunc_size (l4_addr_t *address*, unsigned char *bits*) [inline]

Round an address down to the next lower flex page with size *bits*.

Parameters

address	The address to round.
bits	The size of the flex page (log2).

Definition at line 340 of file consts.h.

Referenced by cxx::List_alloc::alloc_max(), l4_fpage_max_order(), and L4Re::Dataspace::map_region().

Here is the caller graph for this function:

10.88 Message Items

Message item related functions.

Collaboration diagram for Message Items:

Enumerations

- enum l4_msg_item_consts_t {
  L4_ITEM_MAP = 8, L4_ITEM_CONT = 1, L4_MAP_ITEM_GRANT = 2, L4_MAP_ITEM_MAP = 0,
  L4_RCV_ITEM_SINGLE_CAP = L4_ITEM_MAP | 2, L4_RCV_ITEM_LOCAL_ID = 4 }

  Constants for message items.

Functions

- l4_umword_t l4_map_control (l4_umword_t spot, unsigned char cache, unsigned grant) L4_NOTHROW

  Create the first word for a map item for the memory space.
- l4_umword_t l4_map_obj_control (l4_umword_t spot, unsigned grant) L4_NOTHROW

  Create the first word for a map item for the object space.

10.88.1 Detailed Description

Message item related functions.

Message items are typed items that can be transferred via IPC operations. Message items are also used to specify receive windows for typed items to be received. Message items are placed in the message registers (MRs) of the UTCB of the sending thread. Receive items are placed in the buffer registers (BRs) of the UTCB of the receiving thread.

Message items are usually two-word data structures. The first word denotes the type of the message item (for example a memory flex-page, io flex-page or object flex-page) and the second word contains information depending on the type. There is actually one exception that is a small (one word) receive buffer item for a single capability.

10.88.2 Enumeration Type Documentation

10.88.2.1 enum l4_msg_item_consts_t

Constants for message items.

Enumerator

L4_ITEM_MAP Identify a message item as *map item*.

L4_ITEM_CONT Donote that the following item shall be put into the same receive item as this one.

L4_MAP_ITEM_GRANT Flag as *grant* instead of *map* operation.

L4_MAP_ITEM_MAP Flag as usual *map* operation.

L4_RCV_ITEM_SINGLE_CAP Mark the receive buffer to be a small receive item that describes a buffer for a single capability.

L4_RCV_ITEM_LOCAL_ID The receiver requests to receive a local ID instead of a mapping whenever possible.

Definition at line 187 of file consts.h.

10.88.3 Function Documentation

10.88.3.1 l4_umword_t l4_map_control (l4_umword_t *spot*, unsigned char *cache*, unsigned *grant*) [inline]

Create the first word for a map item for the memory space.

Parameters

spot	Hot spot address, used to determine what is actually mapped when send and receive flex page have differing sizes.
cache	Cacheability hints for memory flex pages. See Cacheability options
grant	Indicates if it is a map or a grant item.

Returns

The value to be used as first word in a map item for memory.

Definition at line 570 of file __l4_fpage.h.

References L4_ITEM_MAP.

Referenced by l4_map_obj_control().

Here is the caller graph for this function:

10.88.3.2 l4_umword_t l4_map_obj_control (l4_umword_t *spot*, unsigned *grant*) [inline]

Create the first word for a map item for the object space.

Parameters

spot	Hot spot address, used to determine what is actually mapped when send and receive flex pages have different size.
grant	Indicates if it is a map item or a grant item.

Returns

The value to be used as first word in a map item for kernel objects or IO-ports.

Definition at line 577 of file __l4_fpage.h.

References l4_map_control().

Referenced by l4_factory_create_gate_u(), l4_icu_bind_u(), l4_icu_unbind_u(), l4_irq_attach_u(), and l4_irq_mux↩_chain_u().

Here is the call graph for this function:

Here is the caller graph for this function:

10.89 Message Registers (MRs)

Collaboration diagram for Message Registers (MRs):

Modules

- Exception registers

  Overly definition of the MRs for exception messages.

Data Structures

- union l4_msg_regs_t

  Encapsulation of the message-register block in the UTCB.

Typedefs

- typedef union l4_msg_regs_t l4_msg_regs_t

  Encapsulation of the message-register block in the UTCB.

10.89.1 Detailed Description

10.90 Message Tag

API related to the message tag data type.

Collaboration diagram for Message Tag:

Data Structures

- struct l4_msgtag_t

  Message tag data structure.

Typedefs

- typedef struct l4_msgtag_t l4_msgtag_t

  Message tag data structure.

Enumerations

- enum L4_platform_ctl_proto { L4_PROTO_PLATFORM_CTL = 0 }

  Predefined protocol type for messages to platform-control objects.
- enum l4_msgtag_protocol {
  L4_PROTO_NONE = 0, L4_PROTO_ALLOW_SYSCALL = 1, L4_PROTO_PF_EXCEPTION = 1, L4_PROTO_IRQ = -1L,
  L4_PROTO_PAGE_FAULT = -2L, L4_PROTO_PREEMPTION = -3L, L4_PROTO_SYS_EXCEPTION = -4L, L4_PROTO_EXCEPTION = -5L,
  L4_PROTO_SIGMA0 = -6L, L4_PROTO_IO_PAGE_FAULT = -8L, L4_PROTO_KOBJECT = -10L, L4_PROTO_TASK = -11L,
  L4_PROTO_THREAD = -12L, L4_PROTO_LOG = -13L, L4_PROTO_SCHEDULER = -14L, L4_PROTO_FACTORY = -15L,
  L4_PROTO_VM = -16L, L4_PROTO_DMA_SPACE = -17L, L4_PROTO_IRQ_SENDER = -18L, L4_PROTO_IRQ_MUX = -19L,
  L4_PROTO_SEMAPHORE = -20L, L4_PROTO_META = -21L, L4_PROTO_IOMMU = -22L, L4_PROTO_DEBUGGER = -23L }

  Message tag for IPC operations.
- enum l4_msgtag_flags {
  L4_MSGTAG_ERROR, L4_MSGTAG_XCPU, L4_MSGTAG_TRANSFER_FPU, L4_MSGTAG_SCHEDULE,
  L4_MSGTAG_PROPAGATE, L4_MSGTAG_FLAGS }

  Flags for message tags.

Functions

- l4_msgtag_t l4_msgtag (long label, unsigned words, unsigned items, unsigned flags) L4_NOTHROW

Create a message tag from the specified values.

- long l4_msgtag_label (l4_msgtag_t t) L4_NOTHROW

  Get the protocol of tag.
- unsigned l4_msgtag_words (l4_msgtag_t t) L4_NOTHROW

  Get the number of untyped words.
- unsigned l4_msgtag_items (l4_msgtag_t t) L4_NOTHROW

  Get the number of typed items.
- unsigned l4_msgtag_flags (l4_msgtag_t t) L4_NOTHROW

  Get the flags.
- unsigned l4_msgtag_has_error (l4_msgtag_t t) L4_NOTHROW

  Test for error indicator flag.
- unsigned l4_msgtag_is_page_fault (l4_msgtag_t t) L4_NOTHROW

  Test for page-fault protocol.
- unsigned l4_msgtag_is_preemption (l4_msgtag_t t) L4_NOTHROW

  Test for preemption protocol.
- unsigned l4_msgtag_is_sys_exception (l4_msgtag_t t) L4_NOTHROW

  Test for system-exception protocol.
- unsigned l4_msgtag_is_exception (l4_msgtag_t t) L4_NOTHROW

  Test for exception protocol.
- unsigned l4_msgtag_is_sigma0 (l4_msgtag_t t) L4_NOTHROW

  Test for sigma0 protocol.
- unsigned l4_msgtag_is_io_page_fault (l4_msgtag_t t) L4_NOTHROW

  Test for IO-page-fault protocol.

10.90.1 Detailed Description

API related to the message tag data type.

```
#include <l4/sys/types.h>
```

10.90.2 Typedef Documentation

10.90.2.1 typedef struct l4_msgtag_t l4_msgtag_t

Message tag data structure.

```
#include <l4/sys/types.h>
```

Describes the details of an IPC operation, in particular which parts of the UTCB have to be transmitted, and also flags to enable real-time and FPU extensions.

The message tag also contains a user-defined label that could be used to specify a protocol ID. Some negative values are reserved for kernel protocols such as page faults and exceptions.

The type must be treated completely opaque.

10.90.3 Enumeration Type Documentation

10.90.3.1 enum l4_msgtag_flags

Flags for message tags.

Enumerator

L4_MSGTAG_ERROR Error indicator flag.

L4_MSGTAG_XCPU Cross-CPU invocation indicator flag.

L4_MSGTAG_TRANSFER_FPU Enable FPU transfer flag for IPC. By enabling this flag when sending IPC, the sender indicates that the contents of the FPU shall be transfered to the receiving thread. However, the receiver has to indicate its willingness to receive FPU context in its buffer descriptor register (BDR).

L4_MSGTAG_SCHEDULE Enable schedule in IPC flag. Usually IPC operations donate the remaining time slice of a thread to the called thread. Enabling this flag when sending IPC does a real scheduling decision. However, this flag decreases IPC performance.

L4_MSGTAG_PROPAGATE Enable IPC propagation. This flag enables IPC propagation, which means an IPC reply-connection from the current caller will be propagated to the new IPC receiver. This makes it possible to propagate an IPC call to a third thread, which may then directly answer to the caller.

L4_MSGTAG_FLAGS Mask for all flags.

Definition at line 94 of file types.h.

10.90.3.2 enum l4_msgtag_protocol

Message tag for IPC operations.

All predefined protocols used by the kernel.

Enumerator

L4_PROTO_NONE Default protocol tag to reply to kernel.

L4_PROTO_ALLOW_SYSCALL Allow an alien the system call.

L4_PROTO_PF_EXCEPTION Make an exception out of a page fault.

L4_PROTO_IRQ IRQ message.

L4_PROTO_PAGE_FAULT Page fault message.

L4_PROTO_PREEMPTION Preemption message.

L4_PROTO_SYS_EXCEPTION System exception.

L4_PROTO_EXCEPTION Exception.

L4_PROTO_SIGMA0 Sigma0 protocol.

L4_PROTO_IO_PAGE_FAULT I/O page fault message.

L4_PROTO_KOBJECT Protocol for messages to a a generic kobject.

L4_PROTO_TASK Protocol for messages to a task object.

L4_PROTO_THREAD Protocol for messages to a thread object.

L4_PROTO_LOG Protocol for messages to a log object.

L4_PROTO_SCHEDULER Protocol for messages to a scheduler object.

L4_PROTO_FACTORY Protocol for messages to a factory object.

L4_PROTO_VM Protocol for messages to a virtual machine object.

L4_PROTO_DMA_SPACE Protocol for (creating) kernel DMA space objects.

L4_PROTO_IRQ_SENDER Protocol for IRQ senders (IRQ -> IPC)

L4_PROTO_IRQ_MUX Protocol for IRQ mux (IRQ -> n x IRQ)

L4_PROTO_SEMAPHORE Protocol for semaphore objects.

L4_PROTO_META Meta information protocol.

L4_PROTO_IOMMU Protocol ID for IO-MMUs.

L4_PROTO_DEBUGGER Protocol ID for the ddebugger.

Definition at line 49 of file types.h.

10.90.3.3 enum L4_platform_ctl_proto

Predefined protocol type for messages to platform-control objects.

Enumerator

L4_PROTO_PLATFORM_CTL Protocol messages to a platform control object. See L4_platform_ctl_ops for allowed operations.

Definition at line 146 of file platform_control.h.

10.90.4 Function Documentation

10.90.4.1 l4_msgtag_t l4_msgtag (long *label*, unsigned *words*, unsigned *items*, unsigned *flags*) [inline]

Create a message tag from the specified values.

Message tag functions.

Parameters

label	the user-defined label
words	the number of untyped words within the UTCB
items	the number of typed items (e.g., flex pages) within the UTCB
flags	the IPC flags for realtime and FPU extensions

Returns

Message tag

Examples:

examples/sys/aliens/main.c, examples/sys/ipc/ipc_example.c, examples/sys/singlestep/main.c, examples/sys/start-with-exc/main.c, and examples/sys/utcb-ipc/main.c.

Definition at line 376 of file types.h.

Referenced by L4::Basic_registry::dispatch(), fiasco_gdt_get_entry_offset(), fiasco_gdt_set(), fiasco_ldt_set(), L4::Server< LOOP_HOOKS >::internal_loop(), l4_factory_create_gate_u(), l4_icu_bind_u(), l4_icu_info_u(), l4_icu_msi_info_u(), l4_icu_set_mode_u(), l4_icu_unbind_u(), l4_irq_attach_u(), l4_irq_detach_u(), l4_irq_mux_chain_u(), l4_irq_receive_u(), l4_irq_trigger_u(), l4_irq_unmask_u(), l4_irq_wait_u(), l4_thread_ex_regs_u(), l4_thread_vcpu_control_ext_u(), l4_thread_vcpu_control_u(), l4_thread_yield(), l4_vcon_get_attr_u(), l4_vcon_send_u(), l4_vcon_set_attr_u(), L4::Thread::modify_senders(), and L4::Ipc_svr::Exc_dispatch< R, Exc >::operator()().

Here is the caller graph for this function:

10.90.4.2 unsigned l4_msgtag_flags (l4_msgtag_t *t*) [inline]

Get the flags.

The flag are defined by l4_msgtag_flags.

Parameters

t	the tag

Returns

Flags

Definition at line 399 of file types.h.

10.90.4.3 unsigned l4_msgtag_has_error (l4_msgtag_t *t*) [inline]

Test for error indicator flag.

Parameters

t	the tag

Returns

>0 for yes, 0 for no

Return whether the kernel operation caused a communication error, e.g. with IPC. if true: utcb->error is valid, otherwise utcb->error is not valid

Examples:

examples/sys/aliens/main.c, examples/sys/singlestep/main.c, and examples/sys/utcb-ipc/main.c.

Definition at line 404 of file types.h.

References L4_MSGTAG_ERROR.

Referenced by l4_icu_msi_info_u(), l4_ipc_error(), and l4_vcon_write_u().

Here is the caller graph for this function:

10.90.4.4 unsigned l4_msgtag_is_exception (l4_msgtag_t *t*) [inline]

Test for exception protocol.

Parameters

t	the tag

Returns

Boolean value

Examples:

examples/sys/aliens/main.c, examples/sys/singlestep/main.c, and examples/sys/start-with-exc/main.c.

Definition at line 418 of file types.h.

References l4_msgtag_label(), and L4_PROTO_EXCEPTION.

Here is the call graph for this function:

10.90.4.5 unsigned l4_msgtag_is_io_page_fault (l4_msgtag_t *t*) [inline]

Test for IO-page-fault protocol.

Parameters

t	the tag

Returns

Boolean value

Definition at line 424 of file types.h.

References l4_msgtag_label(), and L4_PROTO_IO_PAGE_FAULT.

Here is the call graph for this function:

10.90.4.6 unsigned l4_msgtag_is_page_fault (l4_msgtag_t *t*) [inline]

Test for page-fault protocol.

Parameters

t	the tag

Returns

Boolean value

Definition at line 409 of file types.h.

References l4_msgtag_label(), and L4_PROTO_PAGE_FAULT.

Here is the call graph for this function:

10.90.4.7 unsigned l4_msgtag_is_preemption (l4_msgtag_t *t*) [inline]

Test for preemption protocol.

Parameters

t	the tag

Returns

Boolean value

Definition at line 412 of file types.h.

References l4_msgtag_label(), and L4_PROTO_PREEMPTION.

Here is the call graph for this function:

10.90.4.8 unsigned l4_msgtag_is_sigma0 (l4_msgtag_t *t*) [inline]

Test for sigma0 protocol.

Parameters

t	the tag

Returns

Boolean value

Definition at line 421 of file types.h.

References l4_msgtag_label(), and L4_PROTO_SIGMA0.

Here is the call graph for this function:

10.90.4.9 unsigned l4_msgtag_is_sys_exception (l4_msgtag_t *t*) [inline]

Test for system-exception protocol.

Parameters

t	the tag

Returns

Boolean value

Definition at line 415 of file types.h.

References l4_msgtag_label(), and L4_PROTO_SYS_EXCEPTION.

Here is the call graph for this function:

10.90.4.10 unsigned l4_msgtag_items (l4_msgtag_t *t*) [inline]

Get the number of typed items.

Parameters

t	The tag

Returns

Number of items.

Definition at line 395 of file types.h.

10.90.4.11 long l4_msgtag_label (l4_msgtag_t *t*) [inline]

Get the protocol of tag.

Parameters

t	The tag

Returns

Label

Examples:

examples/sys/singlestep/main.c, and examples/sys/start-with-exc/main.c.

Definition at line 387 of file types.h.

Referenced by l4_factory_create_gate_u(), l4_msgtag_is_exception(), l4_msgtag_is_io_page_fault(), l4_msgtag↩_is_page_fault(), l4_msgtag_is_preemption(), l4_msgtag_is_sigma0(), and l4_msgtag_is_sys_exception().

Here is the caller graph for this function:

10.90.4.12 unsigned l4_msgtag_words (l4_msgtag_t *t*) [inline]

Get the number of untyped words.

Parameters

t	The tag

Returns

Number of words

Examples:

examples/sys/utcb-ipc/main.c.

Definition at line 391 of file types.h.

Referenced by l4_factory_create_gate_u(), and l4_icu_msi_info_u().

Here is the caller graph for this function:

10.91 Name-space API

API for name spaces that store capabilities.

Collaboration diagram for Name-space API:

Data Structures

- class L4Re::Namespace

  Name-space interface.

Enumerations

- enum L4Re::Namespace::Register_flags { L4Re::Namespace::Ro = L4_CAP_FPAGE_RO, L4Re::Namespace::Rw = L4_CAP_FPAGE_RW, L4Re::↩ Namespace::Rs = L4_CAP_FPAGE_RS, L4Re::Namespace::Rws = L4_CAP_FPAGE_RWS, L4Re::Namespace::Strong = L4_CAP_FPAGE_S, L4Re::Namespace::Trusted = 0x008 , L4Re::↩ Namespace::Link = 0x100, L4Re::Namespace::Overwrite = 0x200 }

  Flags for registering name spaces.
- enum L4Re::Namespace::Query_result_flags { L4Re::Namespace::Partly_resolved = 0x020 }

  Flags returned by query IPC, only used internally.

10.91.1 Detailed Description

API for name spaces that store capabilities.

This is a basic abstraction for managing a mapping from human-readable names to capabilities. In particular, a name can also be mapped to a capability that refers to another name space object. By this means name spaces can be constructed hierarchically.

Name spaces play a central role in L4Re, because the implementation of the name space objects determine the policy which capabilities (which objects) are accessible to a client of a name space.

10.91.2 Enumeration Type Documentation

10.91.2.1 enum L4Re::Namespace::Query_result_flags

Flags returned by query IPC, only used internally.

Enumerator

Partly_resolved Name was only partly resolved.

Definition at line 88 of file namespace.

10.91.2.2 enum L4Re::Namespace::Register_flags

Flags for registering name spaces.

Enumerator

Ro Read-only.

Rw Read-write.

Rs Read-only + strong.

Rws Read-write + strong.

Strong Strong.

Trusted Obsolete, do not use.

Link Obsolete, do not use.

Overwrite If entry already exists, overwrite it.

Definition at line 68 of file namespace.

10.92 Namespace interface

Namespace C interface.

Collaboration diagram for Namespace interface:

Enumerations

- enum l4re_ns_register_flags

  Namespace register flags.

Functions

- long l4re_ns_query_to_srv (l4re_namespace_t srv, char const ∗name, l4_cap_idx_t const cap, int timeout) L4_NOTHROW

  Query the name space for the object named by `name`.
- long l4re_ns_query_srv (l4re_namespace_t srv, char const ∗name, l4_cap_idx_t const cap) L4_NOTHROW

  Query the name space for the object named by `name`.
- long l4re_ns_register_obj_srv (l4re_namespace_t srv, char const ∗name, l4_cap_idx_t const obj, unsigned flags) L4_NOTHROW

  Register an object with a name.

10.92.1 Detailed Description

Namespace C interface.

10.92.2 Enumeration Type Documentation

10.92.2.1 enum l4re_ns_register_flags

Namespace register flags.

See also

L4Re::Namespace::Register_flags

Definition at line 39 of file namespace.h.

10.92.3 Function Documentation

10.92.3.1 long l4re_ns_query_srv (l4re_namespace_t *srv*, char const ∗ *name*, l4_cap_idx_t const *cap*) [inline]

Query the name space for the object named by `name`.

Parameters

srv	Name space server to use for the query.
name	String to query.
cap	Capability slot where the received capability will be stored.

Return values

0	Name could be fully resolved.
>0	Name could only be partly resolved. The number of remaining characters is returned.
-L4_ENOENT	Entry could not be found.
-L4_EAGAIN	Entry exists but no object is yet attached. Try again later.
<0	IPC errors, see l4_error_code_t.

Definition at line 105 of file namespace.h.

References l4re_ns_query_to_srv().

Here is the call graph for this function:

10.92.3.2 long l4re_ns_query_to_srv (l4re_namespace_t *srv*, char const * *name*, l4_cap_idx_t const *cap*, int *timeout*)

Query the name space for the object named by `name`.

Parameters

timeout	Timeout of query in milliseconds. The client will only wait if a name already has been registered with the server but no object has been attached yet.
srv	Name space server to use for the query.
name	String to query.
cap	Capability slot where the received capability will be stored.

Return values

0	Name could be fully resolved.
>0	Name could only be partly resolved. The number of remaining characters is returned.
-L4_ENOENT	Entry could not be found.
-L4_EAGAIN	Entry exists but no object is yet attached. Try again later.
<0	IPC errors, see l4_error_code_t.

Referenced by l4re_ns_query_srv().

Here is the caller graph for this function:

10.92.3.3 long l4re_ns_register_obj_srv (l4re_namespace_t *srv*, char const * *name*, l4_cap_idx_t const *obj*, unsigned *flags*)

Register an object with a name.

Parameters

srv	Name space server to use for the query.
name	Name under which the object should be registered.
obj	Capability to object to register. An invalid capability may be given to only reserve the name for later use.
flags	Flags to assign to the entry, see L4Re::Namespace::Register_flags. Note that the rights that are assigned to a capability are not only determined by the rights given in these flags but also by the rights with which the `obj` capability was mapped to the name space.

Return values

0	Object was successfully registered with *name*.
-L4_EEXIST	Name already registered.
-L4_EPERM	Caller doesn't have necessary permissions.
-L4_ENOMEM	Server has insufficient resources.
-L4_EINVAL	Invalid parameter.
<0	IPC errors, see l4_error_code_t.

Precondition

requires capability rights: {RW}

10.93 Object Invocation

API for L4 object invocation.

Collaboration diagram for Object Invocation:

Modules

- Error Handling

  Error handling for L4 object invocation.
- Message Items

  Message item related functions.
- Message Tag

  API related to the message tag data type.
- Realtime API
- Timeouts

  All kinds of timeouts and time related functions.
- Virtual Registers (UTCBs)

  L4 Virtual Registers (UTCB).

Files

- file utcb.h

  UTCB definitions.

Enumerations

- enum l4_syscall_flags_t {
  L4_SYSF_NONE, L4_SYSF_SEND, L4_SYSF_RECV, L4_SYSF_OPEN_WAIT,
  L4_SYSF_REPLY, L4_SYSF_CALL, L4_SYSF_WAIT, L4_SYSF_SEND_AND_WAIT,
  L4_SYSF_REPLY_AND_WAIT }

Capability selector flags.

Functions

- l4_msgtag_t l4_ipc_send (l4_cap_idx_t dest, l4_utcb_t *utcb, l4_msgtag_t tag, l4_timeout_t timeout) L4_NOTHROW

  *Send a message to an object (do **not** wait for a reply).*
- l4_msgtag_t l4_ipc_wait (l4_utcb_t *utcb, l4_umword_t *label, l4_timeout_t timeout) L4_NOTHROW

  Wait for an incoming message from any possible sender.
- l4_msgtag_t l4_ipc_receive (l4_cap_idx_t object, l4_utcb_t *utcb, l4_timeout_t timeout) L4_NOTHROW

  Wait for a message from a specific source.
- l4_msgtag_t l4_ipc_call (l4_cap_idx_t object, l4_utcb_t *utcb, l4_msgtag_t tag, l4_timeout_t timeout) L4_NOTHROW

  Object call (usual invocation).
- l4_msgtag_t l4_ipc_reply_and_wait (l4_utcb_t *utcb, l4_msgtag_t tag, l4_umword_t *label, l4_timeout_t timeout) L4_NOTHROW

  Reply and wait operation (uses the reply capability).
- l4_msgtag_t l4_ipc_send_and_wait (l4_cap_idx_t dest, l4_utcb_t *utcb, l4_msgtag_t tag, l4_umword_t *label, l4_timeout_t timeout) L4_NOTHROW

  Send a message and do an open wait.
- L4_ALWAYS_INLINE l4_msgtag_t l4_ipc (l4_cap_idx_t dest, l4_utcb_t *utcb, l4_umword_t flags, l4_umword_t slabel, l4_msgtag_t tag, l4_umword_t *rlabel, l4_timeout_t timeout) L4_NOTHROW

  Generic L4 object invocation.
- l4_msgtag_t l4_ipc_sleep (l4_timeout_t timeout) L4_NOTHROW

  Sleep for an amount of time.
- int l4_sndfpage_add (l4_fpage_t const snd_fpage, unsigned long snd_base, l4_msgtag_t *tag) L4_NOTHROW

  Add a flex-page to be sent to the UTCB.

10.93.1 Detailed Description

API for L4 object invocation.

```
#include <l4/sys/ipc.h>
```

General abstractions for L4 object invocation. The basic principle is that all objects are denoted by a capability that is accessed via a capability selector (see Capabilities).

This set of functions is common to all kinds of objects provided by the L4 micro kernel. The concrete semantics of an invocation depends on the object that shall be invoked.

Objects may be invoked in various ways, the most common way is to use a *call* operation (l4_ipc_call()). However, there are a lot more flavours available that have a semantics depending on the object.

See also

IPC-Gate API

10.93.2 Enumeration Type Documentation

10.93.2.1 enum l4_syscall_flags_t

Capability selector flags.

These flags determine the concrete operation when a kernel object is invoked.

Enumerator

L4_SYSF_NONE Default flags (call to a kernel object). Using this value as flags in the capability selector for an invocation indicates a call (send and wait for a reply).

L4_SYSF_SEND Send-phase flag. Setting this flag in a capability selector induces a send phase, this means a message is send to the object denoted by the capability. For receive phase see L4_SYSF_RECV.

L4_SYSF_RECV Receive-phase flag. Setting this flag in a capability selector induces a receive phase, this means the invoking thread waits for a message from the object denoted by the capability. For a send phase see L4_SYSF_SEND.

L4_SYSF_OPEN_WAIT Open-wait flag. This flag indicates that the receive operation (see L4_SYSF_RECV) shall be an *open wait*. *Open wait* means that the invoking thread shall wait for a message from any possible sender and *not* from the sender denoted by the capability.

L4_SYSF_REPLY Reply flag. This flag indicates that the send phase shall use the in-kernel reply capability instead of the capability denoted by the selector index.

L4_SYSF_CALL Call flags (combines send and receive). Combines L4_SYSF_SEND and L4_SYSF_RECV.

L4_SYSF_WAIT Wait flags (combines receive and open wait). Combines L4_SYSF_RECV and L4_SYSF_OPEN_WAIT.

L4_SYSF_SEND_AND_WAIT Send-and-wait flags. Combines L4_SYSF_SEND and L4_SYSF_WAIT.

L4_SYSF_REPLY_AND_WAIT Reply-and-wait flags. Combines L4_SYSF_SEND, L4_SYSF_REPLY, and L4_SYSF_WAIT.

Definition at line 39 of file consts.h.

10.93.3 Function Documentation

10.93.3.1 L4_ALWAYS_INLINE l4_msgtag_t l4_ipc (l4_cap_idx_t *dest*, l4_utcb_t * *utcb*, l4_umword_t *flags*, l4_umword_t *slabel*, l4_msgtag_t *tag*, l4_umword_t * *rlabel*, l4_timeout_t *timeout*) [inline]

Generic L4 object invocation.

Parameters

dest	Destination object.
utcb	UTCB of the caller.
flags	Invocation flags (see l4_syscall_flags_t).
slabel	Send label if applicable (may be seen by the receiver).
tag	Sending message tag.

Return values

rlabel	Receiving label.

Parameters

timeout	Timeout pair (see l4_timeout_t).

Returns

return tag

Definition at line 34 of file ipc.h.

Referenced by l4_ipc_call(), l4_ipc_receive(), l4_ipc_reply_and_wait(), l4_ipc_send(), l4_ipc_send_and_wait(), and l4_ipc_wait().

Here is the caller graph for this function:

10.93.3.2 **l4_msgtag_t l4_ipc_call (l4_cap_idx_t *object*, l4_utcb_t * *utcb*, l4_msgtag_t *tag*, l4_timeout_t *timeout*)** `[inline]`

Object call (usual invocation).

Parameters

object	Capability selector for the object to call.
utcb	UTCB of the caller.
tag	Message tag to describe the message to be sent.
timeout	Timeout pair for send an receive phase (see l4_timeout_t).

Returns

result tag

A message is sent to the object and the invoker waits for a reply from the object. Messages from other sources are not accepted.

Note

The send-to-receive transition needs no time, the object can reply with a send timeout of zero.

Examples:

examples/sys/aliens/main.c, examples/sys/ipc/ipc_example.c, and examples/sys/singlestep/main.c.

Definition at line 441 of file ipc.h.

References l4_ipc(), and L4_SYSF_CALL.

Referenced by L4::Ipc::Iostream::call(), fiasco_gdt_get_entry_offset(), fiasco_gdt_set(), fiasco_ldt_set(), l4_icu↩_bind_u(), l4_icu_info_u(), l4_icu_msi_info_u(), l4_icu_set_mode_u(), l4_icu_unbind_u(), l4_irq_attach_u(), l4_↩irq_detach_u(), l4_irq_mux_chain_u(), l4_irq_receive_u(), l4_thread_ex_regs_u(), l4_thread_vcpu_control_ext_u(), l4_thread_vcpu_control_u(), l4_vcon_get_attr_u(), l4_vcon_set_attr_u(), and L4::Thread::modify_senders().

Here is the call graph for this function:

Here is the caller graph for this function:

10.93.3.3 l4_msgtag_t l4_ipc_receive (l4_cap_idx_t *object*, l4_utcb_t * *utcb*, l4_timeout_t *timeout*) [inline]

Wait for a message from a specific source.

Parameters

object	Object to receive a message from.
timeout	Timeout pair (see l4_timeout_t, only the receive part matters).
utcb	UTCB of the caller.

Returns

result tag.

This operation waits for a message from the specified object. Messages from other sources are not accepted by this operation. The operation does not include a send phase, this means no message is sent to the object.

Note

This operation is usually used to receive messages from a specific IRQ or thread. However, it is not common to use this operation for normal applications.

Examples:

examples/sys/aliens/main.c, examples/sys/singlestep/main.c, examples/sys/start-with-exc/main.c, and examples/sys/utcb-ipc/main.c.

Definition at line 483 of file ipc.h.

References l4_ipc(), L4_SYSF_RECV, and l4_msgtag_t::raw.

Referenced by l4_ipc_sleep(), and l4_thread_yield().

Here is the call graph for this function:

Here is the caller graph for this function:

10.93.3.4 l4_msgtag_t l4_ipc_reply_and_wait (l4_utcb_t * *utcb*, l4_msgtag_t *tag*, l4_umword_t * *label*, l4_timeout_t *timeout*) [inline]

Reply and wait operation (uses the *reply* capability).

Parameters

tag	Describes the message to be sent as reply.
utcb	UTCB of the caller.

Return values

label	Label assigned to the source object of the received message.

Parameters

timeout	Timeout pair (see l4_timeout_t).

Returns

result tag

A message is sent to the previous caller using the implicit reply capability. Afterwards the invoking thread waits for a message from any source.

Note

This is the standard server operation: it sends a reply to the actual client and waits for the next incoming request, which may come from any other client.

Examples:

examples/sys/ipc/ipc_example.c.

Definition at line 449 of file ipc.h.

References L4_INVALID_CAP, l4_ipc(), and L4_SYSF_REPLY_AND_WAIT.

Referenced by L4::Ipc::Iostream::reply_and_wait(), and L4::Server< LOOP_HOOKS >::reply_n_wait().

Here is the call graph for this function:

Here is the caller graph for this function:

10.93.3.5 l4_msgtag_t l4_ipc_send (l4_cap_idx_t *dest*, l4_utcb_t * *utcb*, l4_msgtag_t *tag*, l4_timeout_t *timeout*) [inline]

Send a message to an object (do **not** wait for a reply).

Parameters

dest	Capability selector for the destination object.
utcb	UTCB of the caller.
tag	Descriptor for the message to be sent.
timeout	Timeout pair (see l4_timeout_t) only send part is relevant.

Returns

result tag

A message is sent to the destination object. There is no receive phase included. The invoker continues working after sending the message.

Attention

This is a special-purpose message transfer, objects usually support only invocation via l4_ipc_call().

Examples:

examples/sys/start-with-exc/main.c, and examples/sys/utcb-ipc/main.c.

Definition at line 466 of file ipc.h.

References l4_ipc(), and L4_SYSF_SEND.

Referenced by l4_irq_trigger_u(), l4_irq_unmask_u(), l4_vcon_send_u(), L4::Server< LOOP_HOOKS >::reply_↩n_wait(), and L4::Ipc::Ostream::send().

Here is the call graph for this function:

Here is the caller graph for this function:

10.93.3.6 l4_msgtag_t l4_ipc_send_and_wait (l4_cap_idx_t *dest*, l4_utcb_t * *utcb*, l4_msgtag_t *tag*, l4_umword_t * *label*, l4_timeout_t *timeout*) [inline]

Send a message and do an open wait.

Parameters

dest	Object to send a message to.
utcb	UTCB of the caller.
tag	Describes the message that shall be sent.

Return values

label	Label assigned to the source object of the receive phase.

Parameters

timeout	Timeout pair (see l4_timeout_t).

Returns

result tag

A message is sent to the destination object and the invoking thread waits for a reply from any source.

Note

This is a special-purpose operation and shall not be used in general applications.

Definition at line 457 of file ipc.h.

References l4_ipc(), and L4_SYSF_SEND_AND_WAIT.

Referenced by l4_irq_wait_u().

Here is the call graph for this function:

Here is the caller graph for this function:

10.93.3.7 l4_msgtag_t l4_ipc_sleep (l4_timeout_t *timeout*) [inline]

Sleep for an amount of time.

Parameters

timeout	Timeout pair (see l4_timeout_t, the receive part matters).

Returns

error code:

- L4_IPC_RETIMEOUT: success
- L4_IPC_RECANCELED woken up by a different thread (l4_thread_ex_regs()).

The invoking thread waits until the timeout is expired or the wait was aborted by another thread by l4_thread_ex_regs().

Definition at line 492 of file ipc.h.

References L4_INVALID_CAP, and l4_ipc_receive().

Referenced by l4_sleep_forever().

Here is the call graph for this function:

Here is the caller graph for this function:

10.93.3.8 l4_msgtag_t l4_ipc_wait (l4_utcb_t * *utcb*, l4_umword_t * *label*, l4_timeout_t *timeout*) [inline]

Wait for an incoming message from any possible sender.

Parameters

utcb	UTCB of the caller.

Return values

label	Label assigned to the source object (IPC gate or IRQ).

Parameters

timeout	Timeout pair (see l4_timeout_t, only the receive part is used).

Returns

return tag

This operation does an open wait, and therefore needs no capability to denote the possible source of a message. This means the calling thread waits for an incoming message from any possible source. There is no send phase included in this operation.

The usual usage of this function is to call that function when entering a server loop in a user-level server that implements user-level objects, see also l4_ipc_reply_and_wait().

Examples:

examples/sys/ipc/ipc_example.c.

Definition at line 474 of file ipc.h.

References L4_INVALID_CAP, l4_ipc(), L4_SYSF_WAIT, and l4_msgtag_t::raw.

Referenced by L4::Server< LOOP_HOOKS >::reply_n_wait(), and L4::Ipc::Istream::wait().

Here is the call graph for this function:

Here is the caller graph for this function:

10.93.3.9 int l4_sndfpage_add (l4_fpage_t const *snd_fpage*, unsigned long *snd_base*, l4_msgtag_t * *tag*) [inline]

Add a flex-page to be sent to the UTCB.

Parameters

snd_fpage	Flex-page.
snd_base	Send base.
tag	Tag to be modified.

Return values

tag	Modified tag, the number of items will be increased, all other values in the tag will be retained.

Returns

0 on success, negative error code otherwise

Definition at line 552 of file ipc.h.

References l4_utcb().

Here is the call graph for this function:

10.94 Parent API

Parent interface.

Collaboration diagram for Parent API:

Data Structures

- class L4Re::Parent

  Parent interface.

10.94.1 Detailed Description

Parent interface.

The parent interface provides means for an L4 task to signal changes in its execution state. The main purpose is to signal program termination.

See also

L4Re::Parent for information about the concrete interface.

10.95 Platform Control C API

C interface for controlling platform-wide properties.

Collaboration diagram for Platform Control C API:

Functions

- l4_msgtag_t l4_platform_ctl_system_suspend (l4_cap_idx_t pfc, l4_umword_t extras) L4_NOTHROW

  Enter suspend to RAM.
- l4_msgtag_t l4_platform_ctl_system_shutdown (l4_cap_idx_t pfc, l4_umword_t reboot) L4_NOTHROW

  Shutdown or reboot the system.
- l4_msgtag_t l4_platform_ctl_cpu_enable (l4_cap_idx_t pfc, l4_umword_t phys_id) L4_NOTHROW

  Enable an offline CPU.
- l4_msgtag_t l4_platform_ctl_cpu_disable (l4_cap_idx_t pfc, l4_umword_t phys_id) L4_NOTHROW

  Disable an online CPU.

10.95.1 Detailed Description

C interface for controlling platform-wide properties.

Add

```
#include <l4/sys/platform_control.h>
```

to your code to use the platform control functions. The API allows a client to suspend, reboot or shutdown the system.

For the C++ interface refer to L4::Platform_control

10.95.2 Function Documentation

10.95.2.1 l4_msgtag_t l4_platform_ctl_cpu_disable (l4_cap_idx_t *pfc*, l4_umword_t *phys_id*) [inline]

Disable an online CPU.

Parameters

pfc	Capability to the platform control object.
phys_id	Physical CPU id of CPU (e.g. local APIC id) to disable.

Returns

System call message tag

Definition at line 231 of file platform_control.h.

References l4_utcb().

Here is the call graph for this function:

10.95.2.2 l4_msgtag_t l4_platform_ctl_cpu_enable (l4_cap_idx_t *pfc*, l4_umword_t *phys_id*) [inline]

Enable an offline CPU.

Parameters

pfc	Capability to the platform control object.
phys_id	Physical CPU id of CPU (e.g. local APIC id) to enable.

Returns

System call message tag

Definition at line 224 of file platform_control.h.

References l4_utcb().

Here is the call graph for this function:

10.95.2.3 l4_msgtag_t l4_platform_ctl_system_shutdown (l4_cap_idx_t *pfc*, l4_umword_t *reboot*) [inline]

Shutdown or reboot the system.

Parameters

pfc	Capability selector for the platform-control object
reboot	Shutdown when 0, or reboot when 1.

Returns

Syscall return tag

Definition at line 193 of file platform_control.h.

References l4_utcb().

Here is the call graph for this function:

10.95.2.4 l4_msgtag_t l4_platform_ctl_system_suspend (l4_cap_idx_t *pfc*, l4_umword_t *extras*) [inline]

Enter suspend to RAM.

Parameters

pfc	Capability selector for the platform-control object
extras	some extra platform-specific information needed to enter suspend to RAM.

Returns

Syscall return tag

Definition at line 186 of file platform_control.h.

References l4_utcb().

Here is the call graph for this function:

10.96 Producer

Collaboration diagram for Producer:

Functions

- long l4shmc_chunk_try_to_take (l4shmc_chunk_t ∗chunk)

  Try to mark chunk busy.
- long l4shmc_chunk_ready (l4shmc_chunk_t ∗chunk, l4_umword_t size)

  Mark chunk as filled (ready).
- long l4shmc_chunk_ready_sig (l4shmc_chunk_t ∗chunk, l4_umword_t size)

  Mark chunk as filled (ready) and signal consumer.
- long l4shmc_is_chunk_clear (l4shmc_chunk_t ∗chunk)

  Check whether chunk is free.

10.96.1 Detailed Description

10.96.2 Function Documentation

10.96.2.1 long l4shmc_chunk_ready (l4shmc_chunk_t ∗ *chunk*, l4_umword_t *size*) [inline]

Mark chunk as filled (ready).

Parameters

chunk	chunk.
size	Size of data in the chunk, in bytes.

Return values

0	Success.
<0	Error.

10.96.2.2 long l4shmc_chunk_ready_sig (l4shmc_chunk_t ∗ *chunk*, l4_umword_t *size*) [inline]

Mark chunk as filled (ready) and signal consumer.

Parameters

chunk	chunk.
size	Size of data in the chunk, in bytes.

Return values

0	Success.
<0	Error.

Examples:

examples/libs/shmc/prodcons.c.

10.96.2.3 long l4shmc_chunk_try_to_take (l4shmc_chunk_t * *chunk*) [inline]

Try to mark chunk busy.

Parameters

chunk	chunk to mark.

Return values

0	Chunk could be taken.
<0	Chunk could not be taken, try again.

Examples:

examples/libs/shmc/prodcons.c.

10.96.2.4 long l4shmc_is_chunk_clear (l4shmc_chunk_t * *chunk*) [inline]

Check whether chunk is free.

Parameters

chunk	Chunk to check.

Return values

0	Success.
<0	Error.

10.97 Producer

Collaboration diagram for Producer:

Functions

- long l4shmc_trigger (l4shmc_signal_t *signal)

  Trigger a signal.

10.97.1 Detailed Description

10.97.2 Function Documentation

10.97.2.1 long l4shmc_trigger (l4shmc_signal_t * *signal*) [inline]

Trigger a signal.

Parameters

signal	Signal to trigger.

Return values

0	Success.
<0	Error.

Examples:

examples/libs/shmc/prodcons.c.

10.98 Random number support

Collaboration diagram for Random number support:

Functions

- l4_uint32_t l4util_rand (void)

  Deliver next random number.
- void l4util_srand (l4_uint32_t seed)

  Initialize random number generator.

10.98.1 Detailed Description

10.98.2 Function Documentation

10.98.2.1 l4_uint32_t l4util_rand (void)

Deliver next random number.

Returns

A new random number

10.98.2.2 void l4util_srand (l4_uint32_t *seed*)

Initialize random number generator.

Parameters

seed	Value to initialize

10.99 Realtime API

Collaboration diagram for Realtime API:

10.100 Region map API

Virtual address-space management.

Collaboration diagram for Region map API:

Data Structures

- class L4Re::Rm

  Region map.

Enumerations

- enum L4Re::Rm::Detach_result { L4Re::Rm::Detached_ds = 0, L4Re::Rm::Kept_ds = 1, L4Re::Rm::Split_ds = 2 , L4Re::Rm::Detach_again = 4 }

  Result values for detach operation.
- enum L4Re::Rm::Region_flags { L4Re::Rm::Read_only = 0x01, L4Re::Rm::Detach_free = 0x02, L4Re::Rm::Pager = 0x04, L4Re::Rm::Reserved = 0x08, L4Re::Rm::Caching_shift = 8, L4Re::Rm::Caching_ds_shift = Caching_shift - Dataspace::Map_caching_shift, L4Re::Rm::Caching = Dataspace::Map_caching_mask << Caching_ds_shift, L4Re::Rm::Cache_normal = Dataspace::Map_normal << Caching_ds_shift, L4Re::Rm::Cache_buffered = Dataspace::Map_bufferable << Caching_ds_shift, L4Re::Rm::Cache_uncached = Dataspace::Map_uncacheable << Caching_ds_shift, L4Re::Rm::Region_flags = Caching | 0x0f }

  Flags for regions.
- enum L4Re::Rm::Attach_flags { L4Re::Rm::Search_addr = 0x20, L4Re::Rm::In_area = 0x40, L4Re::Rm::Eager_map = 0x80, L4Re::Rm::Attach_flags = 0xf0 }

  Flags for attach operation.
- enum L4Re::Rm::Detach_flags { L4Re::Rm::Detach_exact = 1, L4Re::Rm::Detach_overlap = 2, L4Re::Rm::Detach_keep = 4 }

  Flags for detach operation.

10.100.1 Detailed Description

Virtual address-space management.

The central purpose of the region-map API is to provide means to manage the virtual memory address space of an L4 task. A region-map object implements two protocols. The first protocol is the kernel page-fault protocol, to resolve page faults for threads running in an L4 task. The second protocol is the region-map protocol itself, that allows to attach a data-space object to a region of the virtual address space.

There are two basic concepts provided by a region-map abstraction:

- Regions provide a means to create a view to a data space (or parts of a data space).

- Areas provide a means to reserve areas in a virtual memory address space for special purposes. A reserved area is skipped when searching for an available range of virtual memory, or may be explicitly used to search only within that area.

See also

L4Re::Dataspace, L4Re::Rm

10.100.2 Enumeration Type Documentation

10.100.2.1 enum L4Re::Rm::Attach_flags

Flags for attach operation.

Enumerator

Search_addr Search for a suitable address range.

In_area Search only in area, or map into area.

Eager_map Eagerly map the attached data space in.

Attach_flags Mask of all attach flags.

Definition at line 111 of file rm.

10.100.2.2 enum L4Re::Rm::Detach_flags

Flags for detach operation.

Enumerator

Detach_exact Do an unmap of the exact region given.

Detach_overlap Do an unmap of all overlapping regions.

Detach_keep Do not free the detached data space, ignore the Detach_free.

Definition at line 121 of file rm.

10.100.2.3 enum L4Re::Rm::Detach_result

Result values for detach operation.

Enumerator

Detached_ds Detached data sapce.

Kept_ds Kept data space.

Split_ds Splitted data space, and done.

Detach_again Detached data space, more to do.

Definition at line 75 of file rm.

10.100.2.4 enum L4Re::Rm::Region_flags

Flags for regions.

Enumerator

Read_only Region is read-only.

Detach_free Free the portion of the data space after detach.

Pager Region has a pager.

Reserved Region is reserved (blocked)

Caching_shift Start of Rm cache bits.

Caching_ds_shift Shift value for Dataspace to Rm cache bits.

Caching Mask of all Rm cache bits.

Cache_normal Cache bits for normal cacheable memory.

Cache_buffered Cache bits for buffered (write combining) memory.

Cache_uncached Cache bits for uncached memory.

Region_flags Mask of all region flags.

Definition at line 86 of file rm.

10.101 Region map interface

Region map C interface.

Collaboration diagram for Region map interface:

Enumerations

- enum l4re_rm_flags_t {
  L4RE_RM_READ_ONLY = 0x01, L4RE_RM_NO_ALIAS = 0x02, L4RE_RM_PAGER = 0x04, L4RE_RM_↩
  RESERVED = 0x08,
  L4RE_RM_REGION_FLAGS = 0x0f, L4RE_RM_OVERMAP = 0x10, L4RE_RM_SEARCH_ADDR = 0x20,
  L4RE_RM_IN_AREA = 0x40,
  L4RE_RM_EAGER_MAP = 0x80, L4RE_RM_ATTACH_FLAGS = 0xf0 }

  Flags for region operations.

Functions

- int l4re_rm_reserve_area (l4_addr_t ∗start, unsigned long size, unsigned flags, unsigned char align) L4_N↩
  OTHROW
- int l4re_rm_free_area (l4_addr_t addr) L4_NOTHROW
- int l4re_rm_attach (void ∗∗start, unsigned long size, unsigned long flags, l4re_ds_t const mem, l4_addr_t offs, unsigned char align) L4_NOTHROW
- int l4re_rm_detach (void ∗addr) L4_NOTHROW

  Detach and unmap in current task.
- int l4re_rm_detach_ds (void ∗addr, l4re_ds_t ∗ds) L4_NOTHROW

  Detach, unmap and return affected dataspace in current task.
- int l4re_rm_detach_unmap (l4_addr_t addr, l4_cap_idx_t task) L4_NOTHROW

  Detach and unmap in specified task.
- int l4re_rm_detach_ds_unmap (void ∗addr, l4re_ds_t ∗ds, l4_cap_idx_t task) L4_NOTHROW

  Detach and unmap in specified task.
- int l4re_rm_find (l4_addr_t ∗addr, unsigned long ∗size, l4_addr_t ∗offset, unsigned ∗flags, l4re_ds_t ∗m) L4_NOTHROW
- void l4re_rm_show_lists (void) L4_NOTHROW

  Dump region map internal data structures.
- int l4re_rm_reserve_area_srv (l4_cap_idx_t rm, l4_addr_t ∗start, unsigned long size, unsigned flags, unsigned char align) L4_NOTHROW
- int l4re_rm_free_area_srv (l4_cap_idx_t rm, l4_addr_t addr) L4_NOTHROW
- int l4re_rm_attach_srv (l4_cap_idx_t rm, void ∗∗start, unsigned long size, unsigned long flags, l4re_ds_t const mem, l4_addr_t offs, unsigned char align) L4_NOTHROW
- int l4re_rm_detach_srv (l4_cap_idx_t rm, l4_addr_t addr, l4re_ds_t ∗ds, l4_cap_idx_t task) L4_NOTHROW
- int l4re_rm_find_srv (l4_cap_idx_t rm, l4_addr_t ∗addr, unsigned long ∗size, l4_addr_t ∗offset, unsigned ∗flags, l4re_ds_t ∗m) L4_NOTHROW
- void l4re_rm_show_lists_srv (l4_cap_idx_t rm) L4_NOTHROW

  Dump region map internal data structures.

10.101.1 Detailed Description

Region map C interface.

10.101.2 Enumeration Type Documentation

10.101.2.1 enum l4re_rm_flags_t

Flags for region operations.

Enumerator

L4RE_RM_READ_ONLY Region is read-only.

L4RE_RM_NO_ALIAS The region contains exclusive memory that is not mapped anywhere else.

L4RE_RM_PAGER Region has a pager.

L4RE_RM_RESERVED Region is reserved (blocked)

L4RE_RM_REGION_FLAGS Mask of all region flags.

L4RE_RM_OVERMAP Unmap memory already mapped in the region.

L4RE_RM_SEARCH_ADDR Search for a suitable address range.

L4RE_RM_IN_AREA Search only in area, or map into area.

L4RE_RM_EAGER_MAP Eagerly map the attached data space in.

L4RE_RM_ATTACH_FLAGS Mask of all attach flags.

Definition at line 40 of file rm.h.

10.101.3 Function Documentation

10.101.3.1 int l4re_rm_attach (void ** *start*, unsigned long *size*, unsigned long *flags*, l4re_ds_t const *mem*, l4_addr_t *offs*, unsigned char *align*) [inline]

Parameters

in,out	*start*	Virtual start address where the region manager shall attach the data space. If L4Re::Rm::Search_addr is given this value is used as the start address to search for a free virtual memory region and the resulting address is returned here. If L4Re::Rm::In_area is given the value is used as a selector for the area (see L4Re::Rm::reserve_area) to attach the data space to.
	size	Size of the data space to attach (in bytes)
	flags	Flags, see L4Re::Rm::Attach_flags and L4Re::Rm::Region_flags
	mem	Data space
	offs	Offset into the data space to use
	align	Alignment of the virtual region, log2-size, default: a page (L4_PAGESHIFT). This is only meaningful if the L4Re::Rm::Search_addr flag is used.

Return values

0	Success
-L4_ENOENT	No area could be found (see L4Re::Rm::In_area)
-L4_EPERM	Operation not allowed.

-L4_EINVAL	
-L4_EADDRNOTAVAIL	The given address is not available.
<0	IPC errors

Makes the whole or parts of a data space visible in the virtual memory of the corresponding task. The corresponding region in the virtual address space is backed with the contents of the dataspace.

Note

When searching for a free place in the virtual address space, the space between *start* and the end of the virtual address space is searched.
There is no region object created, instead the region is defined by a virtual address within this range (see L4Re::Rm::find).

Returns

0 on success, <0 on error

See also

L4Re::Rm::attach

This function is using the L4::Env::env()->rm() service.

Examples:

examples/libs/l4re/c/ma+rm.c.

Definition at line 243 of file rm.h.

References l4re_rm_attach_srv(), and l4re_env_t::rm.

Here is the call graph for this function:

10.101.3.2 int l4re_rm_attach_srv (l4_cap_idx_t *rm*, void ** *start*, unsigned long *size*, unsigned long *flags*, l4re_ds_t const *mem*, l4_addr_t *offs*, unsigned char *align*)

See also

L4Re::Rm::attach

Referenced by l4re_rm_attach().

Here is the caller graph for this function:

10.101.3.3 int l4re_rm_detach (void * *addr*) [inline]

Detach and unmap in current task.

Parameters

addr	Address of the region to detach.

Returns

0 on success, <0 on error

Also

See also

L4Re::Rm::detach

This function is using the L4::Env::env()->rm() service.

Definition at line 253 of file rm.h.

References L4_BASE_TASK_CAP, l4re_rm_detach_srv(), and l4re_env_t::rm.

Here is the call graph for this function:

10.101.3.4 int l4re_rm_detach_ds (void * *addr*, l4re_ds_t * *ds*) [inline]

Detach, unmap and return affected dataspace in current task.

Parameters

addr	Address of the region to detach.

Return values

ds	Returns dataspace that is affected.

Returns

0 on success, <0 on error

Also

See also

L4Re::Rm::detach

This function is using the L4::Env::env()->rm() service.

Examples:

examples/libs/l4re/c/ma+rm.c.

Definition at line 266 of file rm.h.

References L4_BASE_TASK_CAP, l4re_rm_detach_srv(), and l4re_env_t::rm.

Here is the call graph for this function:

10.101.3.5 int l4re_rm_detach_ds_unmap (void * *addr*, l4re_ds_t * *ds*, l4_cap_idx_t *task*) [inline]

Detach and unmap in specified task.

Parameters

addr	Address of the region to detach.

Return values

ds	Returns dataspace that is affected.

Parameters

task	Task to unmap pages from, specify L4_INVALID_CAP to not unmap

Returns

0 on success, <0 on error

Also

See also

L4Re::Rm::detach

This function is using the L4::Env::env()->rm() service.

Definition at line 273 of file rm.h.

References l4re_rm_detach_srv(), and l4re_env_t::rm.

Here is the call graph for this function:

10.101.3.6 int l4re_rm_detach_srv (l4_cap_idx_t *rm*, l4_addr_t *addr*, l4re_ds_t * *ds*, l4_cap_idx_t *task*)

See also

L4Re::Rm::detach

Referenced by l4re_rm_detach(), l4re_rm_detach_ds(), l4re_rm_detach_ds_unmap(), and l4re_rm_detach_unmap().

Here is the caller graph for this function:

10.101.3.7 int l4re_rm_detach_unmap (l4_addr_t *addr*, l4_cap_idx_t *task*) [inline]

Detach and unmap in specified task.

Parameters

addr	Address of the region to detach.
task	Task to unmap pages from, specify L4_INVALID_CAP to not unmap

Returns

0 on success, <0 on error

Also

See also

L4Re::Rm::detach

This function is using the L4::Env::env()->rm() service.

Definition at line 260 of file rm.h.

References l4re_rm_detach_srv(), and l4re_env_t::rm.

Here is the call graph for this function:

10.101.3.8 int l4re_rm_find (l4_addr_t * *addr*, unsigned long * *size*, l4_addr_t * *offset*, unsigned * *flags*, l4re_ds_t * *m*) [inline]

Returns

0 on success, <0 on error

See also

L4Re::Rm::find

Definition at line 280 of file rm.h.

References l4re_rm_find_srv(), and l4re_env_t::rm.

Here is the call graph for this function:

10.101.3.9 int l4re_rm_find_srv (l4_cap_idx_t *rm*, l4_addr_t * *addr*, unsigned long * *size*, l4_addr_t * *offset*, unsigned * *flags*, l4re_ds_t * *m*)

See also

L4Re::Rm::find

Referenced by l4re_rm_find().

Here is the caller graph for this function:

10.101.3.10 int l4re_rm_free_area (l4_addr_t *addr*) [inline]

Returns

0 on success, <0 on error

See also

L4Re::Rm::free_area

This function is using the L4::Env::env()->rm() service.

Definition at line 237 of file rm.h.

References l4re_rm_free_area_srv(), and l4re_env_t::rm.

Here is the call graph for this function:

10.101.3.11 int l4re_rm_free_area_srv (l4_cap_idx_t *rm*, l4_addr_t *addr*)

See also

L4Re::Rm::free_area

Referenced by l4re_rm_free_area().

Here is the caller graph for this function:

10.101.3.12 int l4re_rm_reserve_area (l4_addr_t * *start*, unsigned long *size*, unsigned *flags*, unsigned char *align*) [inline]

Returns

0 on success, <0 on error

See also

L4Re::Rm::reserve_area

This function is using the L4::Env::env()->rm() service.

Definition at line 229 of file rm.h.

References l4re_rm_reserve_area_srv(), and l4re_env_t::rm.

Here is the call graph for this function:

10.101.3.13 int l4re_rm_reserve_area_srv (l4_cap_idx_t *rm*, l4_addr_t * *start*, unsigned long *size*, unsigned *flags*, unsigned char *align*)

See also

L4Re::Rm::reserve_area

Referenced by l4re_rm_reserve_area().

Here is the caller graph for this function:

10.101.3.14 void l4re_rm_show_lists (void) [inline]

Dump region map internal data structures.

This function is using the L4::Env::env()->rm() service.

Definition at line 287 of file rm.h.

References l4re_rm_show_lists_srv(), and l4re_env_t::rm.

Here is the call graph for this function:

10.102 Scheduler

Scheduler object.

Collaboration diagram for Scheduler:

Data Structures

- struct l4_sched_cpu_set_t

  CPU sets.
- struct l4_sched_param_t

  Scheduler parameter set.

Typedefs

- typedef struct l4_sched_cpu_set_t l4_sched_cpu_set_t

  CPU sets.
- typedef struct l4_sched_param_t l4_sched_param_t

  Scheduler parameter set.

Enumerations

- enum L4_scheduler_ops { L4_SCHEDULER_INFO_OP = 0UL, L4_SCHEDULER_RUN_THREAD_OP = 1↩ UL, L4_SCHEDULER_IDLE_TIME_OP = 2UL }

  Operations on the Scheduler object.

Functions

- l4_msgtag_t L4::Scheduler::info (l4_umword_t ∗cpu_max, l4_sched_cpu_set_t ∗cpus, l4_utcb_t ∗utcb=l4_↩ utcb()) const throw ()

  ABI function for 'info' call.
- l4_msgtag_t L4::Scheduler::run_thread (Ipc::Cap< Thread > thread, l4_sched_param_t const &sp)

  Run a thread on a Scheduler.
- l4_msgtag_t L4::Scheduler::idle_time (l4_sched_cpu_set_t const &cpus, l4_kernel_clock_t ∗us)

  Query the idle time (in µs) of a CPU.
- bool L4::Scheduler::is_online (l4_umword_t cpu, l4_utcb_t ∗utcb=l4_utcb()) const throw ()

  Query if a CPU is online.
- l4_sched_cpu_set_t l4_sched_cpu_set (l4_umword_t offset, unsigned char granularity, l4_umword_t map=1) L4_NOTHROW
- l4_sched_param_t l4_sched_param (unsigned prio, l4_cpu_time_t quantum=0) L4_NOTHROW

  Construct scheduler parameter.

10.102.1 Detailed Description

Scheduler object.

```
#include <l4/sys/scheduler.h>
```

10.102.2 Enumeration Type Documentation

10.102.2.1 enum L4_scheduler_ops

Operations on the Scheduler object.

Enumerator

L4_SCHEDULER_INFO_OP Query infos about the scheduler.

L4_SCHEDULER_RUN_THREAD_OP Run a thread on this scheduler.

L4_SCHEDULER_IDLE_TIME_OP Query idle time for the scheduler.

Definition at line 179 of file scheduler.h.

10.102.3 Function Documentation

10.102.3.1 l4_msgtag_t L4::Scheduler::idle_time (l4_sched_cpu_set_t const & *cpus*, l4_kernel_clock_t * *us*)

Query the idle time (in µs) of a CPU.

Parameters

	cpus	Set of CPUs to query. Only the idle time of the first selected CPU is queried.
out	*us*	Idle time of queried CPU in µs.

Returns

0 on success, <0 error code otherwise.

10.102.3.2 l4_msgtag_t L4::Scheduler::info (l4_umword_t * *cpu_max*, l4_sched_cpu_set_t * *cpus*, l4_utcb_t * *utcb* = l4_utcb ()) const throw) [inline]

ABI function for 'info' call.

Get scheduler information.

Parameters

out	*cpu_max*	Maximum number of CPUs ever available.
in,out	*cpus*	*cpus.offset* is first CPU of interest. *cpus.granularity* (see l4_sched_cpu_set_t). *cpus.map* Bitmap of online CPUs.
	utcb	UTCB pointer of the calling thread. This defaults to the UTCB of the current thread.

Returns

0 on success, <0 error code otherwise.

Definition at line 65 of file scheduler.

References l4_sched_cpu_set_t::gran_offset, l4_sched_cpu_set_t::map, and cxx::max().

Here is the call graph for this function:

10.102.3.3 bool L4::Scheduler::is_online (l4_umword_t *cpu*, l4_utcb_t * *utcb* = l4_utcb ()) const throw) [inline]

Query if a CPU is online.

Parameters

cpu	CPU number whose online status should be queried.
utcb	UTCB pointer of the calling thread. Defaults to *l4_utcb()*.

Returns

true if online, false if not (or any other query error).

Definition at line 109 of file scheduler.

10.102.3.4 l4_sched_cpu_set_t l4_sched_cpu_set (l4_umword_t *offset*, unsigned char *granularity*, l4_umword_t *map* = 1) [inline]

Parameters

offset	Offset.
granularity	Granularitry in log2 notation.
map	Bitmap of CPUs, defaults to 1 in C++.

Returns

CPU set.

Examples:

examples/sys/migrate/thread_migrate.cc.

Definition at line 189 of file scheduler.h.

References l4_sched_cpu_set_t::gran_offset, and l4_sched_cpu_set_t::map.

Referenced by l4_sched_param().

Here is the caller graph for this function:

10.102.3.5 l4_msgtag_t L4::Scheduler::run_thread (Ipc::Cap< Thread > *thread*, l4_sched_param_t const & *sp*)

Run a thread on a Scheduler.

Parameters

thread	Capability of the thread to run.
sp	Scheduling parameters.

Returns

0 on success, <0 error code otherwise.

10.103 Server-Side IPC framework

Server-Side framework for implementing object-oriented servers.

Collaboration diagram for Server-Side IPC framework:

Namespaces

- L4::Ipc_svr

  Helper classes for L4::Server instantiation.

Data Structures

- class L4::Ipc_svr::Server_iface

  Interface for server-loop related functions.
- class L4::Basic_registry

  This registry returns the corresponding server object based on the label of an Ipc_gate.
- struct L4::Ipc_svr::Ignore_errors

  Mix in for LOOP_HOOKS to ignore IPC errors.
- struct L4::Ipc_svr::Default_timeout

  Mix in for LOOP_HOOKS to use a 0 send and a infinite receive timeout.
- struct L4::Ipc_svr::Compound_reply

  Mix in for LOOP_HOOKS to always use compound reply and wait.
- struct L4::Ipc_svr::Default_setup_wait

  Mix in for LOOP_HOOKS for setup_wait no op.
- class L4::Ipc_svr::Timed_work< HOOKS >

  DEPRECATED.
- struct L4::Ipc_svr::Direct_dispatch< R >

  Direct disptach helper, for forwarding dispatch calls a registry R.
- struct L4::Ipc_svr::Direct_dispatch< R * >

  Direct disptach helper, for forwarding dispatch calls via a pointer to a registry R.
- struct L4::Ipc_svr::Exc_dispatch< R, Exc >

  Dispatch helper wrapping try {} catch {} around the dispatch call.
- class L4::Ipc_svr::Br_manager_no_buffers

  Empty implementation of Server_iface.
- struct L4::Ipc_svr::Default_loop_hooks

  Default LOOP_HOOKS.
- class L4::Server< LOOP_HOOKS >

  Basic server loop for handling client requests.
- class L4::Server_object

  Abstract server object to be used with L4::Server and L4::Basic_registry.
- struct L4::Server_object_t< IFACE, BASE >

  Base class (template) for server implementing server objects.

- struct L4::Server_object_x< Derived, IFACE, BASE >

  Helper class to implement p_dispatch based server objects.
- struct L4::Irq_handler_object

  Server object base class for handling IRQ messages.
- class L4::Ipc_svr::Timeout

  Callback interface for Timeout_queue.
- class L4::Ipc_svr::Timeout_queue

  Timeout queue to be used in l4re server loop.
- class L4::Ipc_svr::Timeout_queue_hooks< HOOKS, BR_MAN >

  Loop hooks mixin for integrating a timeout queue into the server loop.

Enumerations

- enum L4::Ipc_svr::Reply_mode { L4::Ipc_svr::Reply_compound, L4::Ipc_svr::Reply_separate }

  Reply mode for server loop.

10.103.1 Detailed Description

Server-Side framework for implementing object-oriented servers.

´

10.103.2 Enumeration Type Documentation

10.103.2.1 enum L4::Ipc_svr::Reply_mode

Reply mode for server loop.

The reply mode specifies if the server loop shall do a compound reply and wait operation (Reply_compound), which is the most performant method. Note, setup_wait() is called before the reply. The other way is to call reply and wait separately and call setup_wait in between.

The actual mode is determined by the return value of the before_reply() hook in the LOOP_HOOKS of L4::Server.

Enumerator

Reply_compound Server shall use a compound reply and wait (fast).

Reply_separate Server shall call reply and wait separately.

Definition at line 50 of file ipc_server_loop.

10.104 Shared Memory Library

L4SHM provides a shared memory infrastructure that establishes a shared memory area between multiple parties and uses a fast notification mechanism.

Collaboration diagram for Shared Memory Library:

Modules

- Chunks
- Signals

Functions

- long l4shmc_create (const char *shmc_name, l4_umword_t shm_size)

  Create a shared memory area.
- long l4shmc_attach (const char *shmc_name, l4shmc_area_t *shmarea)

  Attach to a shared memory area.
- long l4shmc_attach_to (const char *shmc_name, l4_umword_t timeout_ms, l4shmc_area_t *shmarea)

  Attach to a shared memory area, with limited waiting.
- long l4shmc_connect_chunk_signal (l4shmc_chunk_t *chunk, l4shmc_signal_t *signal)

  Connect a signal with a chunk.
- long l4shmc_area_size (l4shmc_area_t *shmarea)

  Get size of shared memory area.
- long l4shmc_area_size_free (l4shmc_area_t *shmarea)

  Get free size of shared memory area.
- long l4shmc_area_overhead (void)

  Get memory overhead per area that is not available for chunks.
- long l4shmc_chunk_overhead (void)

  Get memory overhead required in addition to the chunk capacity for adding one chunk.

10.104.1 Detailed Description

L4SHM provides a shared memory infrastructure that establishes a shared memory area between multiple parties and uses a fast notification mechanism.

A shared memory area consists of chunks and signals. A chunk is a defined chunk of memory within the memory area with a maximum size. A chunk is filled (written) by a producer and read by a consumer. When a producer has finished writing to the chunk it signals a data ready notification to the consumer.

A consumer attaches to a chunk and waits for the producer to fill the chunk. After reading out the chunk it marks the chunk free again.

A shared memory area can have multiple chunks.

The interface is divided in three roles.

- The master role, responsible for setting up a shared memory area.
- A producer, generating data into a chunk
- A consumer, receiving data.

A signal can be connected with a chunk or can be used independently (e.g. for multiple chunks).

10.104.2 Function Documentation

10.104.2.1 long l4shmc_area_overhead (void)

Get memory overhead per area that is not available for chunks.

Returns

size of the overhead in bytes

10.104.2.2 long l4shmc_area_size (l4shmc_area_t * *shmarea*) [inline]

Get size of shared memory area.

Parameters

shmarea	Shared memory area.

Return values

>0	Size of the shared memory area.
<0	Error.

10.104.2.3 long l4shmc_area_size_free (l4shmc_area_t * *shmarea*)

Get free size of shared memory area.

To get the max size to pass to l4shmc_add_chunk, substract l4shmc_chunk_overhead().

Parameters

shmarea	Shared memory area.

Return values

0	Free capacity in the area.
<0	Error.

10.104.2.4 long l4shmc_attach (const char * *shmc_name*, l4shmc_area_t * *shmarea*) [inline]

Attach to a shared memory area.

Parameters

	shmc_name	Name of the shared memory area.
out	*shmarea*	Pointer to shared memory area descriptor to be filled with information for the shared memory area.

Return values

0	Success.
<0	Error.

Examples:

examples/libs/shmc/prodcons.c.

10.104.2.5 long l4shmc_attach_to (const char * *shmc_name*, l4_umword_t *timeout_ms*, l4shmc_area_t * *shmarea*)

Attach to a shared memory area, with limited waiting.

Parameters

	shmc_name	Name of the shared memory area.
	timeout_ms	Timeout to wait for shm area in milliseconds.
out	*shmarea*	Pointer to shared memory area descriptor to be filled with information for the shared memory area.

Return values

0	Success.
<0	Error.

10.104.2.6 long l4shmc_chunk_overhead (void)

Get memory overhead required in addition to the chunk capacity for adding one chunk.

Returns

size of the overhead in bytes

10.104.2.7 long l4shmc_connect_chunk_signal (l4shmc_chunk_t * *chunk*, l4shmc_signal_t * *signal*)

Connect a signal with a chunk.

Parameters

chunk	Chunk to attach the signal to.
signal	Signal to attach.

Returns

0 on success, <0 on error

Examples:

examples/libs/shmc/prodcons.c.

10.104.2.8 long l4shmc_create (const char * *shmc_name,* l4_umword_t *shm_size*)

Create a shared memory area.

Parameters

shmc_name	Name of the shared memory area.
shm_size	Size of the whole shared memory area.

Return values

0	Success.
<0	Error.

Examples:

examples/libs/shmc/prodcons.c.

10.105 Sigma0 API

Sigma0 API bindings.

Collaboration diagram for Sigma0 API:

Modules

- Internal constants

  Internal sigma0 definitions.

Files

- file sigma0.h

  Sigma0 interface.

Enumerations

- enum l4sigma0_return_flags_t { L4SIGMA0_OK, L4SIGMA0_NOTALIGNED, L4SIGMA0_IPCERROR, L4SIGMA0_NOFPAGE , L4SIGMA0_SMALLERFPAGE }

  Return flags of libsigma0 functions.

Functions

- l4_kernel_info_t ∗ l4sigma0_map_kip (l4_cap_idx_t sigma0, void ∗addr, unsigned log2_size)

  Map the kernel info page from pager to addr.
- int l4sigma0_map_mem (l4_cap_idx_t sigma0, l4_addr_t phys, l4_addr_t virt, l4_addr_t size)

  Request a memory mapping from sigma0.
- int l4sigma0_map_iomem (l4_cap_idx_t sigma0, l4_addr_t phys, l4_addr_t virt, l4_addr_t size, int cached)

  Request IO memory from sigma0.
- int l4sigma0_map_anypage (l4_cap_idx_t sigma0, l4_addr_t map_area, unsigned log2_map_size, l4_addr↩_t ∗base, unsigned sz)

  Request an arbitrary free page of RAM.
- int l4sigma0_map_tbuf (l4_cap_idx_t sigma0, l4_addr_t virt)

  Request Fiasco trace buffer.
- void l4sigma0_debug_dump (l4_cap_idx_t sigma0)

  Request sigma0 to dump internal debug information.
- int l4sigma0_new_client (l4_cap_idx_t sigma0, l4_cap_idx_t gate)

  Create a new IPC gate for a new Sigma0 client.
- char const ∗ l4sigma0_map_errstr (int err)

  Get a user readable error messages for the return codes.

10.105.1 Detailed Description

Sigma0 API bindings.

Convenience bindings for the Sigma0 protocol.

10.105.2 Enumeration Type Documentation

10.105.2.1 enum l4sigma0_return_flags_t

Return flags of libsigma0 functions.

Enumerator

L4SIGMA0_OK Ok.

L4SIGMA0_NOTALIGNED Phys, virt or size not aligned.

L4SIGMA0_IPCERROR IPC error.

L4SIGMA0_NOFPAGE No fpage received.

L4SIGMA0_SMALLERFPAGE Superpage requested but smaller flexpage received.

Definition at line 81 of file sigma0.h.

10.105.3 Function Documentation

10.105.3.1 void l4sigma0_debug_dump (l4_cap_idx_t *sigma0*)

Request sigma0 to dump internal debug information.

The debug information, such as internal memory maps, as well as statistics about the internal allocators is dumped to the kernel debugger.

Parameters

sigma0	the sigma0 thread id.

10.105.3.2 int l4sigma0_map_anypage (l4_cap_idx_t *sigma0*, l4_addr_t *map_area*, unsigned *log2_map_size*, l4_addr_t * *base*, unsigned *sz*)

Request an arbitrary free page of RAM.

This function requests arbitrary free memory from sigma0. It should be used whenever spare memory is needed, instead of requesting specific physical memory with l4sigma0_map_mem().

Parameters

sigma0	usually the thread id of sigma0.
map_area	the base address of the local virtual memory area where the page should be mapped.
log2_map_size	the size of the requested page log 2 (the size in bytes is 2^log2_map_size). This must be at least the minimal page size. By specifing larger sizes the largest possible hardware page size will be used.

Return values

base	physical address of the page received (i.e., the send base of the received mapping if any).

Parameters

sz	Size to map by the server, in 2^sz bytes.

Returns

0 on success, !0 else (see l4sigma0_map_errstr()).

10.105.3.3 char const * l4sigma0_map_errstr (int *err*) [inline]

Get a user readable error messages for the return codes.

Parameters

err	the error code reported by the *map* functions.

Returns

a string containing the error message.

Definition at line 208 of file sigma0.h.

10.105.3.4 int l4sigma0_map_iomem (l4_cap_idx_t *sigma0*, l4_addr_t *phys*, l4_addr_t *virt*, l4_addr_t *size*, int *cached*)

Request IO memory from sigma0.

This function is similar to l4sigma0_map_mem(), the difference is that it requests IO memory. IO memory is everything that is not known to be normal RAM. Also ACPI tables or the BIOS memory is treated as IO memory.

Parameters

sigma0	usually the thread id of sigma0.
phys	the physical address to be requested (page aligned).
virt	the virtual address where the memory should be mapped to (page aligned).
size	the size of the IO memory area to be mapped (multiple of page size)
cached	requests cacheable IO memory if 1, and uncached if 0.

Returns

0 on success, !0 else (see l4sigma0_map_errstr()).

10.105.3.5 l4_kernel_info_t* l4sigma0_map_kip (l4_cap_idx_t *sigma0*, void * *addr*, unsigned *log2_size*)

Map the kernel info page from pager to addr.

Parameters

sigma0	Capability selector for the sigma0 gate.
addr	Start of the receive window to receive KIP in.
log2_size	Size of the receive window to receive KIP in.

Returns

Address KIP was mapped to, 0 indicates an error.

10.105.3.6 int l4sigma0_map_mem (l4_cap_idx_t *sigma0,* l4_addr_t *phys,* l4_addr_t *virt,* l4_addr_t *size*)

Request a memory mapping from sigma0.

Parameters

sigma0	ID of service talking the sigma0 protocol.
phys	the physical address of the requested page (must be at least aligned to the minimum page size).
virt	the virtual address where the paged should be mapped in the local address space (must be at least aligned to the minimum page size).
size	the size of the requested page, this must be a multiple of the minimum page size.

Returns

0 on success, !0 else (see l4sigma0_map_errstr()).

10.105.3.7 int l4sigma0_map_tbuf (l4_cap_idx_t *sigma0*, l4_addr_t *virt*)

Request Fiasco trace buffer.

This is a Fiasco specific feature. Where you can request the kernel internal trace buffer for user-level evaluation. This is for special debugging tools, such as Ferret.

Parameters

sigma0	as usual the sigma0 thread id.
virt	the virtual address where the trace buffer should be mapped,

Returns

0 on success, !0 else (see l4sigma0_map_errstr()).

10.105.3.8 int l4sigma0_new_client (l4_cap_idx_t *sigma0*, l4_cap_idx_t *gate*)

Create a new IPC gate for a new Sigma0 client.

Parameters

sigma0	Capability selector for sigma0 gate.
gate	Capability selector to use for the new gate.

10.106 Signals

Collaboration diagram for Signals:

Modules

- Consumer
- Producer

Functions

- long l4shmc_add_signal (l4shmc_area_t ∗shmarea, const char ∗signal_name, l4shmc_signal_t ∗signal)

  Add a signal for the shared memory area.
- long l4shmc_attach_signal (l4shmc_area_t ∗shmarea, const char ∗signal_name, l4_cap_idx_t thread, l4shmc_signal_t ∗signal)

  Attach to signal.
- long l4shmc_attach_signal_to (l4shmc_area_t ∗shmarea, const char ∗signal_name, l4_cap_idx_t thread, l4↩_umword_t timeout_ms, l4shmc_signal_t ∗signal)

  Attach to signal, with timeout.
- long l4shmc_get_signal_to (l4shmc_area_t ∗shmarea, const char ∗signal_name, l4_umword_t timeout_ms, l4shmc_signal_t ∗signal)

  Get signal object from the shared memory area.
- l4_cap_idx_t l4shmc_signal_cap (l4shmc_signal_t ∗signal)

  Get the signal capability of a signal.
- long l4shmc_check_magic (l4shmc_chunk_t ∗chunk)

  Check magic value of a chunk.

10.106.1 Detailed Description

10.106.2 Function Documentation

10.106.2.1 long l4shmc_add_signal (l4shmc_area_t ∗ *shmarea,* const char ∗ *signal_name,* l4shmc_signal_t ∗ *signal*)

Add a signal for the shared memory area.

Parameters

	shmarea	The shared memory area to put the chunk in.

	signal_name	Name of the signal.
out	*signal*	Signal structure to fill in.

Return values

0	Success.
<0	Error.

Examples:

examples/libs/shmc/prodcons.c.

10.106.2.2 long l4shmc_attach_signal (l4shmc_area_t * *shmarea*, const char * *signal_name*, l4_cap_idx_t *thread*, l4shmc_signal_t * *signal*) [inline]

Attach to signal.

Parameters

	shmarea	Shared memory area.
	signal_name	Name of the signal.
	thread	Thread capability index to attach the signal to.
out	*signal*	Signal data structure to fill.

Return values

0	Success.
<0	Error.

10.106.2.3 long l4shmc_attach_signal_to (l4shmc_area_t * *shmarea*, const char * *signal_name*, l4_cap_idx_t *thread*, l4_umword_t *timeout_ms*, l4shmc_signal_t * *signal*)

Attach to signal, with timeout.

Parameters

	shmarea	Shared memory area.
	signal_name	Name of the signal.
	thread	Thread capability index to attach the signal to.
	timeout_ms	Timeout in milliseconds to wait for the chunk to appear in the shared memory area.
out	*signal*	Signal data structure to fill.

Return values

0	Success.
<0	Error.

Examples:

examples/libs/shmc/prodcons.c.

10.106.2.4 long l4shmc_check_magic (l4shmc_chunk_t * *chunk*) [inline]

Check magic value of a chunk.

Parameters

chunk	Chunk.

Return values

0	Magic value is not valid.
>0	Chunk is ok, the magic value is valid.

10.106.2.5 long l4shmc_get_signal_to (l4shmc_area_t * *shmarea,* const char * *signal_name,* l4_umword_t *timeout_ms,* l4shmc_signal_t * *signal*)

Get signal object from the shared memory area.

Parameters

	shmarea	Shared memory area.
	signal_name	Name of the signal.
	timeout_ms	Timeout in milliseconds to wait for signal of a chunk to appear in the shared memory area.
`out`	*signal*	Signal data structure to fill.

Return values

0	Success.
<0	Error.

10.106.2.6 l4_cap_idx_t l4shmc_signal_cap (l4shmc_signal_t * *signal*) `[inline]`

Get the signal capability of a signal.

Parameters

signal	Signal.

Returns

Capability of the signal object.

10.107 Small C++ Template Library

Namespaces

- cxx

  Our C++ library.

Data Structures

- class L4::Alloc_list

  A simple list-based allocator.
- class cxx::Auto_ptr< T >

  Smart pointer with automatic deletion.
- class cxx::Bitmap_base

  Basic bitmap abstraction.
- class cxx::Bitmap_base::Bit

  A writeable bit in a bitmap.
- class cxx::Bitmap_base::Word< BITS >

  Helper abstraction for a word contained in the bitmap.
- class cxx::Bitmap_base::Char< BITS >

  Helper abstraction for a byte contained in the bitmap.
- class cxx::Bitmap< BITS >

  A static bit map.
- class cxx::List_item

  Basic list item.
- class cxx::List_item::Iter

  Iterator for a list of ListItem-s.
- class cxx::List_item::T_iter< T, Poly >

  Iterator for derived classes from ListItem.
- struct cxx::Pair< First, Second >

  Pair of two values.
- class cxx::Base_slab< Obj_size, Slab_size, Max_free, Alloc >

  Basic slab allocator.
- class cxx::Slab< Type, Slab_size, Max_free, Alloc >

  Slab allocator for object of type Type.
- class cxx::Base_slab_static< Obj_size, Slab_size, Max_free, Alloc >

  Merged slab allocator (allocators for objects of the same size are merged together).
- class cxx::Slab_static< Type, Slab_size, Max_free, Alloc >

  Merged slab allocator (allocators for objects of the same size are merged together).
- class cxx::Nothrow

  Helper type to distinguish the `oeprator new` version that does not throw exceptions.
- class cxx::New_allocator< _Type >

  Standard allocator based on `operator new ()` .
- class L4::String

  A null-terminated string container class.

Enumerations

- enum { cxx::Bitmap_base::W_bits = sizeof(word_type) ∗ 8, cxx::Bitmap_base::C_bits = 8 }
- enum { cxx::Base_slab< Obj_size, Slab_size, Max_free, Alloc >::object_size = Obj_size, cxx::Base_slab< Obj_size, Slab_size, Max_free, Alloc >::slab_size = Slab_size, cxx::Base_slab< Obj_size, Slab_size, Max_free, Alloc >::objects_per_slab = (Slab_size - sizeof(Slab_head)) / object_size, cxx::Base_slab< Obj_size, Slab_size, Max_free, Alloc >::max_free_slabs = Max_free }
- enum { cxx::Base_slab_static< Obj_size, Slab_size, Max_free, Alloc >::object_size = Obj_size, cxx::Base_slab_static< Obj_size, Slab_size, Max_free, Alloc >::slab_size = Slab_size, cxx::Base_slab_static< Obj_size, Slab_size, Max_free, Alloc >::objects_per_slab = _A::objects_per_slab, cxx::Base_slab_static< Obj_size, Slab_size, Max_free, Alloc >::max_free_slabs = Max_free }

Functions

- template<typename T1 >
  T1 cxx::min (T1 a, T1 b)

  Get the minimum of a and b.
- template<typename T1 >
  T1 cxx::max (T1 a, T1 b)

  Get the maximum of a and b.
- void ∗ operator new (size_t, void ∗mem, cxx::Nothrow const &) throw ()

  Simple placement new operator.
- void ∗ operator new (size_t, cxx::Nothrow const &) throw ()

  New operator that does not throw exceptions.

10.107.1 Detailed Description

10.107.2 Enumeration Type Documentation

10.107.2.1 anonymous enum [protected]

Enumerator

W_bits number of bits in word_type

C_bits number of bits in char

Definition at line 38 of file bitmap.

10.107.2.2 template<int Obj_size, int Slab_size = L4_PAGESIZE, int Max_free = 2, template< typename A > class Alloc = New_allocator> anonymous enum

Enumerator

object_size size of an object.

slab_size size of a slab cache.

objects_per_slab objects per slab cache.

max_free_slabs maximum number of free slab caches.

Definition at line 63 of file slab_alloc.

10.107.2.3 template<int Obj_size, int Slab_size = L4_PAGESIZE, int Max_free = 2, template< typename A > class Alloc = New_allocator> anonymous enum

Enumerator

object_size size of an object.

slab_size size of a slab cache.

objects_per_slab number of objects per slab cache.

max_free_slabs maximum number of free slab caches.

Definition at line 355 of file slab_alloc.

10.107.3 Function Documentation

10.107.3.1 template<typename T1 > T1 cxx::max (T1 *a*, T1 *b*) [inline]

Get the maximum of *a* and *b*.

Parameters

a	the first value.
b	the second value.

Definition at line 45 of file minmax.

Referenced by cxx::List_alloc::alloc_max(), L4::Scheduler::info(), L4virtio::Svr::Driver_mem_list_t< DATA >::init(), and L4::Type_info::Demand::operator|().

Here is the caller graph for this function:

10.107.3.2 template<typename T1 > T1 cxx::min (T1 *a*, T1 *b*) [inline]

Get the minimum of *a* and *b*.

Parameters

a	the first value.

b	the second value.

Examples:

tmpfs/lib/src/fs.cc.

Definition at line 35 of file minmax.

Referenced by L4virtio::Svr::Data_buffer::copy_to(), and L4virtio::Svr::Data_buffer::skip().

Here is the caller graph for this function:

10.107.3.3 void* operator new (size_t , void * *mem,* cxx::Nothrow const &) throw) [inline]

Simple placement new operator.

Parameters

mem	the address of the memory block to place the new object.

Returns

the address given by *mem*.

Definition at line 39 of file std_alloc.

10.108 Task

C interface of the Task kernel object.

Collaboration diagram for Task:

Enumerations

- enum l4_unmap_flags_t { L4_FP_ALL_SPACES, L4_FP_DELETE_OBJ, L4_FP_OTHER_SPACES }

  Flags for the unmap operation.

Functions

- l4_msgtag_t l4_task_map (l4_cap_idx_t dst_task, l4_cap_idx_t src_task, l4_fpage_t snd_fpage, l4_addr_t snd_base) L4_NOTHROW

  Map resources available in the source task to a destination task.
- l4_msgtag_t l4_task_unmap (l4_cap_idx_t task, l4_fpage_t fpage, l4_umword_t map_mask) L4_NOTHROW

  Revoke rights from the task.
- l4_msgtag_t l4_task_unmap_batch (l4_cap_idx_t task, l4_fpage_t const *fpages, unsigned num_fpages, unsigned long map_mask) L4_NOTHROW

  Revoke rights from a task.
- l4_msgtag_t l4_task_delete_obj (l4_cap_idx_t task, l4_cap_idx_t obj) L4_NOTHROW

  Release capability and delete object.
- l4_msgtag_t l4_task_release_cap (l4_cap_idx_t task, l4_cap_idx_t cap) L4_NOTHROW

  Release capability.
- l4_msgtag_t l4_task_cap_valid (l4_cap_idx_t task, l4_cap_idx_t cap) L4_NOTHROW

  Test whether a capability selector points to a valid capability.
- l4_msgtag_t l4_task_cap_has_child (l4_cap_idx_t task, l4_cap_idx_t cap) L4_NOTHROW

  Test whether a capability has child mappings (in another task).
- l4_msgtag_t l4_task_cap_equal (l4_cap_idx_t task, l4_cap_idx_t cap_a, l4_cap_idx_t cap_b) L4_NOTHROW

  Test whether two capabilities point to the same object with the same rights.
- l4_msgtag_t l4_task_add_ku_mem (l4_cap_idx_t task, l4_fpage_t ku_mem) L4_NOTHROW

  Add kernel-user memory.

10.108.1 Detailed Description

C interface of the Task kernel object.

A task represents a combination of the address spaces provided by the L4Re micro kernel. A task consists of at least a memory address space and an object address space. On IA32 there is also an IO-port address space.

Task objects are created using the Factory interface.

Include File

```
#include <l4/sys/task.h>
```

10.108.2 Enumeration Type Documentation

10.108.2.1 enum l4_unmap_flags_t

Flags for the unmap operation.

See also

L4::Task::unmap() and l4_task_unmap()

Enumerator

L4_FP_ALL_SPACES Flag to tell the unmap operation to unmap all child mappings including the mapping in the invoked task.

See also

L4::Task::unmap() l4_task_unmap()

L4_FP_DELETE_OBJ Flag that indicates that the unmap operation on a capability shall try to delete the corresponding objects immediately.

See also

L4::Task::unmap() l4_task_unmap()

L4_FP_OTHER_SPACES Counterpart to L4_FP_ALL_SPACES, unmap only child mappings.

See also

L4::Task::unmap() l4_task_unmap()

Definition at line 157 of file consts.h.

10.108.3 Function Documentation

10.108.3.1 l4_msgtag_t l4_task_add_ku_mem (l4_cap_idx_t *task*, l4_fpage_t *ku_mem*) [inline]

Add kernel-user memory.

Parameters

task	Capability selector of the task to add the memory to
ku_mem	Flexpage describing the virtual area the memory goes to.

Returns

Syscall return tag

Definition at line 428 of file task.h.

References l4_utcb().

Here is the call graph for this function:

10.108.3.2 l4_msgtag_t l4_task_cap_equal (l4_cap_idx_t *task*, l4_cap_idx_t *cap_a*, l4_cap_idx_t *cap_b*) [inline]

Test whether two capabilities point to the same object with the same rights.

Parameters

task	Capability selector of the destination task to do the lookup in
cap_a	Capability selector to compare
cap_b	Capability selector to compare

Returns

label contains 1 if equal, 0 if not equal

Definition at line 421 of file task.h.

References l4_utcb().

Here is the call graph for this function:

10.108.3.3 l4_msgtag_t l4_task_cap_has_child (l4_cap_idx_t *task*, l4_cap_idx_t *cap*) [inline]

Test whether a capability has child mappings (in another task).

Parameters

task	Capability selector of the destination task to do the lookup in
cap	Capability selector to look up in the destination task

Returns

label contains 1 if it has at least one child, 0 if not or invalid

Definition at line 415 of file task.h.

References l4_utcb().

Here is the call graph for this function:

10.108.3.4 l4_msgtag_t l4_task_cap_valid (l4_cap_idx_t *task*, l4_cap_idx_t *cap*) [inline]

Test whether a capability selector points to a valid capability.

Parameters

task	Capability selector of the destination task to do the lookup in
cap	Capability selector to look up in the destination task

Returns

label contains >0 if valid, 0 if invalid

Definition at line 409 of file task.h.

References l4_utcb().

Here is the call graph for this function:

10.108.3.5 l4_msgtag_t l4_task_delete_obj (l4_cap_idx_t *task*, l4_cap_idx_t *obj*) [inline]

Release capability and delete object.

Parameters

task	Capability selector of destination task
obj	Capability selector of object to delete

Returns

Syscall return tag

The object will be deleted if the obj has sufficient rights. No error will be reported if the rights are insufficient, however, the capability is removed in all cases.

This operation calls l4_task_unmap() with L4_FP_DELETE_OBJ.

Definition at line 388 of file task.h.

References l4_utcb().

Here is the call graph for this function:

10.108.3.6 l4_msgtag_t l4_task_map (l4_cap_idx_t *dst_task*, l4_cap_idx_t *src_task*, l4_fpage_t *snd_fpage*, l4_addr_t *snd_base*) [inline]

Map resources available in the source task to a destination task.

Parameters

dst_task	Capability selector of destination task
src_task	Capability selector of source task
snd_fpage	Send flexpage that describes an area in the address space or object space of the source task.
snd_base	Send base that describes an offset in the receive window of the destination task.

Returns

Syscall return tag

This method allows for asynchronous rights delegation from one task to another. It can be used to share memory as well as to delegate access to objects.

Definition at line 358 of file task.h.

References l4_utcb().

Here is the call graph for this function:

10.108.3.7 l4_msgtag_t l4_task_release_cap (l4_cap_idx_t *task*, l4_cap_idx_t *cap*) [inline]

Release capability.

Parameters

task	Capability selector of destination task
cap	Capability selector to release

Returns

Syscall return tag

This operation unmaps the capability from the specified task.

Definition at line 403 of file task.h.

References l4_utcb().

Here is the call graph for this function:

10.108.3.8 l4_msgtag_t l4_task_unmap (l4_cap_idx_t *task*, l4_fpage_t *fpage*, l4_umword_t *map_mask*) [inline]

Revoke rights from the task.

Parameters

task	Capability selector of destination task
fpage	Flexpage that describes an area in the address space or object space of the destination task
map_mask	Unmap mask, see l4_unmap_flags_t

Returns

Syscall return tag

This method allows to revoke rights from the destination task and from all the tasks that got the rights delegated from that task (i.e., this operation does a recursive rights revocation).

Note

Calling this function on the object space can cause a root capability of an object to be destructed, which destroys the object itself.

Definition at line 365 of file task.h.

References l4_utcb().

Referenced by L4Re::Util::Cap_alloc_base::free(), L4Re::Util::Counting_cap_alloc< COUNTERTYPE >::free(), and L4Re::Util::Counting_cap_alloc< COUNTERTYPE >::release().

Here is the call graph for this function:

Here is the caller graph for this function:

10.108.3.9 l4_msgtag_t l4_task_unmap_batch (l4_cap_idx_t *task*, l4_fpage_t const * *fpages*, unsigned *num_fpages*, unsigned long *map_mask*) [inline]

Revoke rights from a task.

Parameters

task	Capability selector of destination task
fpages	An array of flexpages that describes an area in the address space or object space of the destination task each
num_fpages	The size of the fpages array in elements (number of fpages sent).
map_mask	Unmap mask, see l4_unmap_flags_t

Returns

Syscall return tag

This method allows to revoke rights from the destination task and from all the tasks that got the rights delegated from that task (i.e., this operation does a recursive rights revocation).

Precondition

The caller needs to take care that num_fpages is not bigger than L4_UTCB_GENERIC_DATA_SIZE - 2.

Note

Calling this function on the object space can cause a root capability of an object to be destructed, which destroys the object itself.

Definition at line 372 of file task.h.

References l4_utcb().

Here is the call graph for this function:

10.109 Thread

Thread object.

Collaboration diagram for Thread:

Modules

- Thread control

  API for Thread Control method.
- vCPU API

  vCPU API

Enumerations

- enum L4_thread_control_flags { L4_THREAD_CONTROL_SET_PAGER = 0x0010000, L4_THREAD_CONTROL_BIND_TASK = 0x0200000, L4_THREAD_CONTROL_ALIEN = 0x0400000, L4_THREAD_CONTROL_UX_NATIVE = 0x0800000, L4_THREAD_CONTROL_SET_EXC_HANDLER = 0x1000000 }

  Flags for the thread control operation.
- enum L4_thread_control_mr_indices { L4_THREAD_CONTROL_MR_IDX_FLAGS = 0, L4_THREAD_CONTROL_MR_IDX_PAGER = 1, L4_THREAD_CONTROL_MR_IDX_EXC_HANDLER = 2, L4_THREAD_CONTROL_MR_IDX_FLAG_VALS = 4, L4_THREAD_CONTROL_MR_IDX_BIND_UTCB = 5, L4_THREAD_CONTROL_MR_IDX_BIND_TASK = 6 }

  Indices for the values in the message register for thread control.
- enum L4_thread_ex_regs_flags { L4_THREAD_EX_REGS_CANCEL = 0x10000UL, L4_THREAD_EX_REGS_TRIGGER_EXCEPTION = 0x20000UL }

  Flags for the thread ex-regs operation.

Functions

- l4_msgtag_t l4_thread_ex_regs (l4_cap_idx_t thread, l4_addr_t ip, l4_addr_t sp, l4_umword_t flags) L4_NOTHROW

  Exchange basic thread registers.
- l4_msgtag_t l4_thread_ex_regs_u (l4_cap_idx_t thread, l4_addr_t ip, l4_addr_t sp, l4_umword_t flags, l4_utcb_t *utcb) L4_NOTHROW

  Exchange basic thread registers.
- l4_msgtag_t l4_thread_ex_regs_ret (l4_cap_idx_t thread, l4_addr_t *ip, l4_addr_t *sp, l4_umword_t *flags) L4_NOTHROW

  Exchange basic thread registers and return previous values.

- l4_msgtag_t l4_thread_ex_regs_ret_u (l4_cap_idx_t thread, l4_addr_t *ip, l4_addr_t *sp, l4_umword_t *flags, l4_utcb_t *utcb) L4_NOTHROW

  Exchange basic thread registers and return previous values.
- l4_msgtag_t l4_thread_yield (void) L4_NOTHROW

  Yield current time slice.
- l4_msgtag_t l4_thread_switch (l4_cap_idx_t to_thread) L4_NOTHROW

  Switch to another thread (and donate the remaining time slice).
- l4_msgtag_t l4_thread_stats_time (l4_cap_idx_t thread, l4_kernel_clock_t *us) L4_NOTHROW

  Get consumed time of thread in µs.
- l4_msgtag_t l4_thread_vcpu_resume_start (void) L4_NOTHROW

  vCPU return from event handler.
- l4_msgtag_t l4_thread_vcpu_resume_commit (l4_cap_idx_t thread, l4_msgtag_t tag) L4_NOTHROW

  Commit vCPU resume.
- l4_msgtag_t l4_thread_vcpu_control (l4_cap_idx_t thread, l4_addr_t vcpu_state) L4_NOTHROW

  Enable or disable the vCPU feature for the thread.
- l4_msgtag_t l4_thread_vcpu_control_u (l4_cap_idx_t thread, l4_addr_t vcpu_state, l4_utcb_t *utcb) L4_NOTHROW

  Enable or disable the vCPU feature for the thread.
- l4_msgtag_t l4_thread_vcpu_control_ext (l4_cap_idx_t thread, l4_addr_t ext_vcpu_state) L4_NOTHROW

  Enable or disable the extended vCPU feature for the thread.
- l4_msgtag_t l4_thread_vcpu_control_ext_u (l4_cap_idx_t thread, l4_addr_t ext_vcpu_state, l4_utcb_t *utcb) L4_NOTHROW

  Enable or disable the extended vCPU feature for the thread.
- l4_msgtag_t l4_thread_register_del_irq (l4_cap_idx_t thread, l4_cap_idx_t irq) L4_NOTHROW

  Register an IRQ that will trigger upon deletion events.
- l4_msgtag_t l4_thread_modify_sender_start (void) L4_NOTHROW

  Start a thread sender modifiction sequence.
- int l4_thread_modify_sender_add (l4_umword_t match_mask, l4_umword_t match, l4_umword_t del_bits, l4_umword_t add_bits, l4_msgtag_t *tag) L4_NOTHROW

  Add a modification pattern to a sender modification sequence.
- l4_msgtag_t l4_thread_modify_sender_commit (l4_cap_idx_t thread, l4_msgtag_t tag) L4_NOTHROW

  Apply (commit) a sender modification sequence.
- l4_msgtag_t l4_thread_arm_set_tpidruro (l4_cap_idx_t thread, l4_addr_t tpidruro) L4_NOTHROW

  Set the TPIDRURO thread specific register.

10.109.1 Detailed Description

Thread object.

An L4 thread is a thread of execution in the L4 context. Usually user-level and kernel threads are mapped 1:1 to each other. Thread kernel objects are created using a factory, see Factory (l4_factory_create_thread()).

Amongst other things an L4 thread encapsulates:

- CPU state
  - General-purpose registers
  - Program counter
  - Stack pointer
- FPU state
- Scheduling parameters, see the Scheduler API
- Execution state

  - Blocked, Runnable, Running

Thread objects provide an API for

- Thread configuration and manipulation
- Thread switching.

The thread control functions are used to control various aspects of a thread. See l4_thread_control_start() for more information.

Include File

```
#include <l4/sys/thread.h>
```

For the C++ interface refer to L4::Thread.

10.109.2 Enumeration Type Documentation

10.109.2.1 enum L4_thread_control_flags

Flags for the thread control operation.

Enumerator

L4_THREAD_CONTROL_SET_PAGER The pager will be given.
L4_THREAD_CONTROL_BIND_TASK The task to bind the thread to will be given.
L4_THREAD_CONTROL_ALIEN Alien state of the thread is set.
L4_THREAD_CONTROL_UX_NATIVE Fiasco-UX only: pass-through of host system calls is set.
L4_THREAD_CONTROL_SET_EXC_HANDLER The exception handler of the thread will be given.

Definition at line 635 of file thread.h.

10.109.2.2 enum L4_thread_control_mr_indices

Indices for the values in the message register for thread control.

Enumerator	See also
L4_THREAD_CONTROL_MR_IDX_FLAGS	L4_thread_control_flags.
L4_THREAD_CONTROL_MR_IDX_PAGER	Index for pager cap.
L4_THREAD_CONTROL_MR_IDX_EXC_HANDLER	Index for exception handler.
L4_THREAD_CONTROL_MR_IDX_FLAG_VALS	Index for feature values.
L4_THREAD_CONTROL_MR_IDX_BIND_UTCB	Index for UTCB address for bind.
L4_THREAD_CONTROL_MR_IDX_BIND_TASK	Index for task flex-page for bind.

Definition at line 658 of file thread.h.

10.109.2.3 enum L4_thread_ex_regs_flags

Flags for the thread ex-regs operation.

Enumerator

L4_THREAD_EX_REGS_CANCEL Cancel ongoing IPC in the thread.
L4_THREAD_EX_REGS_TRIGGER_EXCEPTION Trigger artificial exception in thread.

Definition at line 673 of file thread.h.

10.109.3 Function Documentation

10.109.3.1 l4_msgtag_t l4_thread_arm_set_tpidruro (l4_cap_idx_t *thread*, l4_addr_t *tpidruro*) [inline]

Set the TPIDRURO thread specific register.

Parameters

thread	Thread to manipulate
tpidruro	The value to be set

Returns

System call return tag

Definition at line 59 of file thread.h.

References l4_utcb().

Here is the call graph for this function:

10.109.3.2 l4_msgtag_t l4_thread_ex_regs (l4_cap_idx_t *thread*, l4_addr_t *ip*, l4_addr_t *sp*, l4_umword_t *flags*) [inline]

Exchange basic thread registers.

Parameters

thread	Capability selector of the thread to manipulate.
ip	New instruction pointer, use ~0UL to leave the instruction pointer unchanged.
sp	New stack pointer, use ~0UL to leave the stack pointer unchanged.
flags	Ex-regs flags, see L4_thread_ex_regs_flags.

Returns

System call return tag

This method allows to manipulate a thread. The basic functionality is to set the instruction pointer and the stack pointer of a thread. Additionally, this method allows also to cancel ongoing IPC operations and to force the thread to raise an artificial exception (see `flags`).

The thread is started using l4_scheduler_run_thread(). However, if at the time l4_scheduler_run_thread() is called, the instruction pointer of the thread is invalid, a later call to l4_thread_ex_regs() with a valid instruction pointer might start the thread.

Examples:

examples/sys/aliens/main.c, examples/sys/singlestep/main.c, examples/sys/start-with-exc/main.c, and examples/sys/utcb-ipc/main.c.

Definition at line 824 of file thread.h.

References l4_thread_ex_regs_u(), and l4_utcb().

Here is the call graph for this function:

10.109.3.3 l4_msgtag_t l4_thread_ex_regs_ret (l4_cap_idx_t *thread,* l4_addr_t * *ip,* l4_addr_t * *sp,* l4_umword_t * *flags*) [inline]

Exchange basic thread registers and return previous values.

Parameters

	thread	Capability selector of the thread to manipulate.
in,out	*ip*	New instruction pointer, use ~0UL to leave the instruction pointer unchanged, return previous instruction pointer.
in,out	*sp*	New stack pointer, use ~0UL to leave the stack pointer unchanged, returns previous stack pointer.
in,out	*flags*	Ex-regs flags, see L4_thread_ex_regs_flags, return previous CPU flags of the thread.

Returns

System call return tag

This method allows to manipulate and start a thread. The basic functionality is to set the instruction pointer and the stack pointer of a thread. Additionally, this method allows also to cancel ongoing IPC operations and to force the thread to raise an artificial exception (see `flags`).

Returned values are valid only if function returns successfully.

Definition at line 831 of file thread.h.

References l4_thread_ex_regs_ret_u(), and l4_utcb().

Here is the call graph for this function:

10.109.3.4 l4_msgtag_t l4_thread_ex_regs_ret_u (l4_cap_idx_t *thread*, l4_addr_t * *ip*, l4_addr_t * *sp*, l4_umword_t * *flags*, l4_utcb_t * *utcb*) [inline]

Exchange basic thread registers and return previous values.

Parameters

	thread	Capability selector of the thread to manipulate.
`in,out`	*ip*	New instruction pointer, use ~0UL to leave the instruction pointer unchanged, return previous instruction pointer.
`in,out`	*sp*	New stack pointer, use ~0UL to leave the stack pointer unchanged, returns previous stack pointer.
`in,out`	*flags*	Ex-regs flags, see L4_thread_ex_regs_flags, return previous CPU flags of the thread.
	utcb	UTCB to use for this operation.

Returns

System call return tag

This method allows to manipulate and start a thread. The basic functionality is to set the instruction pointer and the stack pointer of a thread. Additionally, this method allows also to cancel ongoing IPC operations and to force the thread to raise an artificial exception (see `flags`).

Returned values are valid only if function returns successfully.

Definition at line 697 of file thread.h.

References l4_thread_ex_regs_u(), and l4_msg_regs_t::mr.

Referenced by L4::Thread::ex_regs(), and l4_thread_ex_regs_ret().

Here is the call graph for this function:

Here is the caller graph for this function:

10.109.3.5 l4_msgtag_t l4_thread_ex_regs_u (l4_cap_idx_t *thread*, l4_addr_t *ip*, l4_addr_t *sp*, l4_umword_t *flags*, l4_utcb_t * *utcb*) [inline]

Exchange basic thread registers.

Parameters

thread	Capability selector of the thread to manipulate.
ip	New instruction pointer, use ~0UL to leave the instruction pointer unchanged.
sp	New stack pointer, use ~0UL to leave the stack pointer unchanged.
flags	Ex-regs flags, see L4_thread_ex_regs_flags.
utcb	UTCB to use for this operation.

Returns

System call return tag

This method allows to manipulate a thread. The basic functionality is to set the instruction pointer and the stack pointer of a thread. Additionally, this method allows also to cancel ongoing IPC operations and to force the thread to raise an artificial exception (see `flags`).

The thread is started using L4::Scheduler::run_thread(). However, if at the time L4::Scheduler::run_thread() is called, the instruction pointer of the thread is invalid, a later call to ex_regs() with a valid instruction pointer might start the thread.

Definition at line 686 of file thread.h.

References l4_ipc_call(), L4_IPC_NEVER, l4_msgtag(), L4_PROTO_THREAD, L4_THREAD_EX_REGS_OP, and l4_msg_regs_t::mr.

Referenced by L4::Thread::ex_regs(), l4_thread_ex_regs(), and l4_thread_ex_regs_ret_u().

Here is the call graph for this function:

Here is the caller graph for this function:

10.109.3.6 int l4_thread_modify_sender_add (l4_umword_t *match_mask*, l4_umword_t *match*, l4_umword_t *del_bits*, l4_umword_t *add_bits*, l4_msgtag_t * *tag*) [inline]

Add a modification pattern to a sender modification sequence.

Parameters

tag	Tag received from l4_thread_modify_sender_start() or previous l4_thread_modify_sender_add() calls from the same sequence.
match_mask	Bitmask of bits to match the label.
match	Bitmask that must be equal to the label after applying match_mask.
del_bits	Bits to be deleted from the label.
add_bits	Bits to be added to the label.

Returns

0 on sucess, <0 on error

In pseudo code: if ((sender_label & match_mask) == match) { label = (label & ~del_bits) | add_bits; }

Only the first match is applied.

See also

l4_thread_modify_sender_start
l4_thread_modify_sender_commit

Definition at line 1004 of file thread.h.

References l4_utcb().

Here is the call graph for this function:

10.109.3.7 l4_msgtag_t l4_thread_modify_sender_commit (l4_cap_idx_t *thread*, l4_msgtag_t *tag*) [inline]

Apply (commit) a sender modification sequence.

See also

l4_thread_modify_sender_start
l4_thread_modify_sender_add

Definition at line 1015 of file thread.h.

References l4_utcb().

Here is the call graph for this function:

10.109.3.8 l4_msgtag_t l4_thread_modify_sender_start (void) [inline]

Start a thread sender modifiction sequence.

Add modification rules with l4_thread_modify_sender_add() and commit with l4_thread_modify_sender_commit(). Do not touch the UTCB between l4_thread_modify_sender_start() and l4_thread_modify_sender_commit().

See also

l4_thread_modify_sender_add
l4_thread_modify_sender_commit

Definition at line 998 of file thread.h.

References l4_utcb().

Here is the call graph for this function:

10.109.3.9 l4_msgtag_t l4_thread_register_del_irq (l4_cap_idx_t *thread*, l4_cap_idx_t *irq*) [inline]

Register an IRQ that will trigger upon deletion events.

Parameters

thread	Thread to register IRQ for.
irq	Irq to register.

Returns

System call result message tag.

Definition at line 925 of file thread.h.

References l4_utcb().

Here is the call graph for this function:

10.109.3.10 l4_msgtag_t l4_thread_stats_time (l4_cap_idx_t *thread*, l4_kernel_clock_t * *us*) [inline]

Get consumed time of thread in µs.

Parameters

	thread	Thread to get the consumed time from.
out	*us*	Consumed time in µs.

Returns

system call return tag

Definition at line 893 of file thread.h.

References l4_utcb().

Here is the call graph for this function:

10.109.3.11 l4_msgtag_t l4_thread_switch (l4_cap_idx_t *to_thread*) [inline]

Switch to another thread (and donate the remaining time slice).

Parameters

to_thread	The thread to switch to.

Returns

system call return tag

Definition at line 884 of file thread.h.

References l4_utcb().

Here is the call graph for this function:

10.109.3.12 l4_msgtag_t l4_thread_vcpu_control (l4_cap_idx_t *thread*, l4_addr_t *vcpu_state*) [inline]

Enable or disable the vCPU feature for the thread.

Parameters

thread	Capability selector of the thread for which the vCPU feature shall be enabled or disabled.
vcpu_state	The virtual address where the kernel shall store the vCPU state in case of vCPU exits. The address must be a valid kernel-user-memory address (see l4_task_add_ku_mem()).

Returns

Syscall return tag.

This function enables the vCPU feature of the `thread` if `vcpu_state` is set to a valid kernel-user-memory address, or disables the vCPU feature if `vcpu_state` is 0.

Definition at line 942 of file thread.h.

References l4_thread_vcpu_control_u(), and l4_utcb().

Here is the call graph for this function:

10.109.3.13 l4_msgtag_t l4_thread_vcpu_control_ext (l4_cap_idx_t *thread*, l4_addr_t *ext_vcpu_state*) [inline]

Enable or disable the extended vCPU feature for the thread.

Parameters

thread	Capability selector of the thread for which the extended vCPU feature shall be enabled or disabled.
ext_vcpu_state	The virtual address where the kernel shall store the vCPU state in case of vCPU exits. The address must be a valid kernel-user-memory address (see l4_task_add_ku_mem()).

Returns

Systemcall result message tag.

The extended vCPU feature allows the use of hardware-virtualization features such as Intel's VT or AMD's SVM.

This function enables the extended vCPU feature of the `thread` if `ext_vcpu_state` is set to a valid kernel-user-memory address, or disables the vCPU feature if `ext_vcpu_state` is 0.

Note

The extended vCPU mode includes the normal vCPU mode.

Definition at line 957 of file thread.h.

References l4_thread_vcpu_control_ext_u(), and l4_utcb().

Here is the call graph for this function:

10.109.3.14 l4_msgtag_t l4_thread_vcpu_control_ext_u (l4_cap_idx_t *thread*, l4_addr_t *ext_vcpu_state*, l4_utcb_t * *utcb*) [inline]

Enable or disable the extended vCPU feature for the thread.

Parameters

thread	Capability selector of the thread for which the extended vCPU feature shall be enabled or disabled.
ext_vcpu_state	The virtual address where the kernel shall store the vCPU state in case of vCPU exits. The address must be a valid kernel-user-memory address (see L4::Task::add_ku_mem()).
utcb	UTCB to use for this operation.

Returns

Syscall return tag.

The extended vCPU feature allows the use of hardware-virtualization features such as Intel's VT or AMD's SVM.

This function enables the extended vCPU feature of `this` thread if `ext_vcpu_state` is set to a valid kernel-user-memory address, or disables the vCPU feature if `ext_vcpu_state` is 0.

Note

The extended vCPU mode includes the normal vCPU mode.

Definition at line 947 of file thread.h.

References l4_ipc_call(), L4_IPC_NEVER, l4_msgtag(), L4_PROTO_THREAD, and l4_msg_regs_t::mr.

Referenced by l4_thread_vcpu_control_ext(), and L4::Thread::vcpu_control_ext().

Here is the call graph for this function:

Here is the caller graph for this function:

10.109.3.15 l4_msgtag_t l4_thread_vcpu_control_u (l4_cap_idx_t *thread*, l4_addr_t *vcpu_state*, l4_utcb_t * *utcb*) [inline]

Enable or disable the vCPU feature for the thread.

Parameters

thread	Capability selector of the thread for which the vCPU feature shall be enabled or disabled.
vcpu_state	The virtual address where the kernel shall store the vCPU state in case of vCPU exits. The address must be a valid kernel-user-memory address (see L4::Task::add_ku_mem()).
utcb	UTCB to use for this operation.

Returns

Syscall resturn tag.

This function enables the vCPU feature of `this` thread if `vcpu_state` is set to a valid kernel-user-memory address, or disables the vCPU feature if `vcpu_state` is 0.

Definition at line 932 of file thread.h.

References l4_ipc_call(), L4_IPC_NEVER, l4_msgtag(), L4_PROTO_THREAD, L4_THREAD_VCPU_CONTRO↩L_OP, and l4_msg_regs_t::mr.

Referenced by l4_thread_vcpu_control(), and L4::Thread::vcpu_control().

Here is the call graph for this function:

Here is the caller graph for this function:

10.109.3.16 l4_msgtag_t l4_thread_vcpu_resume_commit (l4_cap_idx_t *thread*, l4_msgtag_t *tag*) [inline]

Commit vCPU resume.

Parameters

thread	Thread to be resumed, the invalid cap can be used for the current thread.
tag	Tag to use, returned by l4_thread_vcpu_resume_start()

Returns

System call result message tag. In extended vCPU mode and when the virtual interrupts are cleared, the return code 1 flags an incoming IPC message, whereas 0 indicates a VM exit. An error is returned upon:

- Insufficient rights on the given task capability (-L4_EPERM).
- Given task capability is invalid (-L4_ENOENT).
- A supplied mapping failed.

To resume into another address space the capability to the target task must be set in the vCPU-state, with all lower bits in the task capability cleared. The kernel adds the L4_SYSF_SEND flag to this field to indicate that the capability has been referenced in the kernel. Consecutive resumes will not reference the task capability again until all bits are cleared again. To release a task use the different task capability or use an invalid capability with the L4_SYSF_REPLY flag set.

See also

l4_vcpu_state_t

Definition at line 905 of file thread.h.

References l4_utcb().

Here is the call graph for this function:

10.109.3.17 l4_msgtag_t l4_thread_vcpu_resume_start (void) [inline]

vCPU return from event handler.

Returns

Message tag to be used for l4_sndfpage_add() and l4_thread_vcpu_commit()

The vCPU resume functionality is split in multiple functions to allow the specification of additional send-flex-pages using l4_sndfpage_add().

Definition at line 899 of file thread.h.

References l4_utcb().

Here is the call graph for this function:

10.109.3.18 l4_msgtag_t l4_thread_yield (void) [inline]

Yield current time slice.

Returns

system call return tag

Definition at line 773 of file thread.h.

References L4_INVALID_CAP, L4_IPC_BOTH_TIMEOUT_0, l4_ipc_receive(), and l4_msgtag().

Here is the call graph for this function:

10.110 Thread Control Registers (TCRs)

Collaboration diagram for Thread Control Registers (TCRs):

Data Structures

- struct l4_thread_regs_t

  Encapsulation of the thread-control-register block of the UTCB.

Typedefs

- typedef struct l4_thread_regs_t l4_thread_regs_t

  Encapsulation of the thread-control-register block of the UTCB.

10.110.1 Detailed Description

10.111 Thread control

API for Thread Control method.

Collaboration diagram for Thread control:

Functions

- void l4_thread_control_start (void) L4_NOTHROW

  Start a thread control API sequence.
- void l4_thread_control_pager (l4_cap_idx_t pager) L4_NOTHROW

  Set the pager.
- void l4_thread_control_exc_handler (l4_cap_idx_t exc_handler) L4_NOTHROW

  Set the exception handler.
- void l4_thread_control_bind (l4_utcb_t *thread_utcb, l4_cap_idx_t task) L4_NOTHROW

  Bind the thread to a task.
- void l4_thread_control_alien (int on) L4_NOTHROW

  Enable alien mode.
- void l4_thread_control_ux_host_syscall (int on) L4_NOTHROW

  Enable pass through of native host (Linux) system calls.
- l4_msgtag_t l4_thread_control_commit (l4_cap_idx_t thread) L4_NOTHROW

  Commit the thread control parameters.

10.111.1 Detailed Description

API for Thread Control method.

The thread control API provides access to almost any parameter of a thread object. The API is based on a single invocation of the thread object. However, because of the huge amount of parameters, the API provides a set of functions to set specific parameters of a thread and a commit function to commit the thread control call (see l4_thread_control_commit()).

A thread control operation must always start with l4_thread_control_start() and be committed with l4_thread_control_commit(). All other thread control parameter setter functions must be called between these two functions.

An example for a sequence of thread control API calls can be found below.

```
l4_utcb_t *u = l4_utcb();
l4_thread_control_start(u);
l4_thread_control_pager(u, pager_cap);
l4_thread_control_bind (u, thread_utcb, task);
l4_thread_control_commit(u, thread_cap);
```

10.111.2 Function Documentation

10.111.2.1 void l4_thread_control_alien (int *on*) [inline]

Enable alien mode.

Parameters

on	Boolean value defining the state of the feature.

Alien mode means the thread is not allowed to invoke L4 kernel objects directly and it is also not allowed to allocate FPU state. All those operations result in an exception IPC that gets sent through the pager capability. The responsible pager can then selectively allow an object invocation or allocate FPU state for the thread.

This feature can be used to attach a debugger to a thread and trace all object invocations.

Examples:

examples/sys/aliens/main.c, and examples/sys/singlestep/main.c.

Definition at line 863 of file thread.h.

References l4_utcb().

Here is the call graph for this function:

10.111.2.2 void l4_thread_control_bind (l4_utcb_t * *thread_utcb*, l4_cap_idx_t *task*) [inline]

Bind the thread to a task.

Parameters

thread_utcb	The address of the UTCB in the target task.
task	The target task of the thread.

Binding a thread to a task has the effect that the thread afterwards executes code within that task and has access to the resources visible within that task.

Note

There should not be more than one thread use a UTCB to prevent data corruption.

Examples:

examples/sys/aliens/main.c, examples/sys/singlestep/main.c, examples/sys/start-with-exc/main.c, and examples/sys/utcb-ipc/main.c.

Definition at line 857 of file thread.h.

References l4_utcb().

Here is the call graph for this function:

10.111.2.3 l4_msgtag_t l4_thread_control_commit (l4_cap_idx_t *thread*) [inline]

Commit the thread control parameters.

Parameters

thread	Capability selector of target thread to commit to.

Returns

system call return tag

Examples:

examples/sys/aliens/main.c, examples/sys/singlestep/main.c, examples/sys/start-with-exc/main.c, and examples/sys/utcb-ipc/main.c.

Definition at line 875 of file thread.h.

References l4_utcb().

Here is the call graph for this function:

10.111.2.4 void l4_thread_control_exc_handler (l4_cap_idx_t *exc_handler*) [inline]

Set the exception handler.

Parameters

exc_handler	Capability selector invoked to send an exception IPC.

Note

The exception-handler capability selector is interpreted in the task the thread is bound to (executes in).

Examples:

examples/sys/aliens/main.c, examples/sys/singlestep/main.c, examples/sys/start-with-exc/main.c, and examples/sys/utcb-ipc/main.c.

Definition at line 850 of file thread.h.

References l4_utcb().

Here is the call graph for this function:

10.111.2.5 void l4_thread_control_pager (l4_cap_idx_t *pager*) [inline]

Set the pager.

Parameters

pager	Capability selector invoked to send a page-fault IPC.

Note

The pager capability selector is interpreted in the task the thread is bound to (executes in).

Examples:

examples/sys/aliens/main.c, examples/sys/singlestep/main.c, examples/sys/start-with-exc/main.c, and examples/sys/utcb-ipc/main.c.

Definition at line 844 of file thread.h.

References l4_utcb().

Here is the call graph for this function:

10.111.2.6 void l4_thread_control_start (void) [inline]

Start a thread control API sequence.

This function starts a sequence of thread control API functions. After this functions any of following functions may be called in any order.

- l4_thread_control_pager()
- l4_thread_control_exc_handler()
- l4_thread_control_bind()
- l4_thread_control_alien()
- l4_thread_control_ux_host_syscall() (Fiasco-UX only)

To commit the changes to the thread l4_thread_control_commit() must be called in the end.

Note

The thread control API calls store the parameters for the thread in the UTCB of the caller, this means between l4_thread_control_start() and l4_thread_control_commit() no functions that modify the UTCB contents must be called.

Examples:

examples/sys/aliens/main.c, examples/sys/singlestep/main.c, examples/sys/start-with-exc/main.c, and examples/sys/utcb-ipc/main.c.

Definition at line 838 of file thread.h.

References l4_utcb().

Here is the call graph for this function:

10.111.2.7 void l4_thread_control_ux_host_syscall (int *on*) [inline]

Enable pass through of native host (Linux) system calls.

Parameters

on	Boolean value defining the state of the feature.

Precondition

Running on Fiasco-UX

This enables the thread to do host system calls. This feature is only available in Fiasco-UX and ignored in other environments.

Definition at line 869 of file thread.h.

References l4_utcb().

Here is the call graph for this function:

10.112 Timeouts

All kinds of timeouts and time related functions.

Collaboration diagram for Timeouts:

Data Structures

- struct l4_timeout_s

  Basic timeout specification.
- union l4_timeout_t

  Timeout pair.

Macros

- #define L4_IPC_TIMEOUT_0 ((l4_timeout_s){0x0400})

  Timeout constants.
- #define L4_IPC_TIMEOUT_NEVER ((l4_timeout_s){0})

  never timeout
- #define L4_IPC_NEVER_INITIALIZER {0}

  never timeout, init
- #define L4_IPC_NEVER ((l4_timeout_t){0})

  never timeout
- #define L4_IPC_RECV_TIMEOUT_0 ((l4_timeout_t){0x00000400})

  0 receive timeout
- #define L4_IPC_SEND_TIMEOUT_0 ((l4_timeout_t){0x04000000})

  0 send timeout
- #define L4_IPC_BOTH_TIMEOUT_0 ((l4_timeout_t){0x04000400})

  0 receive and send timeout

Typedefs

- typedef struct l4_timeout_s l4_timeout_s

  Basic timeout specification.
- typedef union l4_timeout_t l4_timeout_t

  Timeout pair.

Enumerations

- enum l4_timeout_abs_validity

  Intervals of validity for absolute timeouts
  Times are actually 2^x values (e.g.

Functions

- l4_timeout_s l4_timeout_rel (unsigned man, unsigned exp) L4_NOTHROW

  Get relative timeout consisting of mantissa and exponent.
- l4_timeout_t l4_ipc_timeout (unsigned snd_man, unsigned snd_exp, unsigned rcv_man, unsigned rcv_exp) L4_NOTHROW

  Convert explicit timeout values to l4_timeout_t type.
- l4_timeout_t l4_timeout (l4_timeout_s snd, l4_timeout_s rcv) L4_NOTHROW

  Combine send and receive timeout in a timeout.
- void l4_snd_timeout (l4_timeout_s snd, l4_timeout_t *to) L4_NOTHROW

  Set send timeout in given to timeout.
- void l4_rcv_timeout (l4_timeout_s rcv, l4_timeout_t *to) L4_NOTHROW

  Set receive timeout in given to timeout.
- l4_kernel_clock_t l4_timeout_rel_get (l4_timeout_s to) L4_NOTHROW

  Get clock value of out timeout.
- unsigned l4_timeout_is_absolute (l4_timeout_s to) L4_NOTHROW

  Return whether the given timeout is absolute or not.
- l4_kernel_clock_t l4_timeout_get (l4_kernel_clock_t cur, l4_timeout_s to) L4_NOTHROW

  Get clock value for a clock + a timeout.
- l4_timeout_s l4_timeout_abs (l4_kernel_clock_t pint, int br) L4_NOTHROW

  Set an absolute timeout.
- unsigned l4_utcb_mr64_idx (unsigned idx) L4_NOTHROW

  Get index into 64bit message registers alias from native-sized index.

10.112.1 Detailed Description

All kinds of timeouts and time related functions.

10.112.2 Macro Definition Documentation

10.112.2.1 #define L4_IPC_TIMEOUT_0 ((l4_timeout_s){0x0400})

Timeout constants.

0 timeout

Definition at line 77 of file __timeout.h.

Referenced by L4::Ipc_svr::Timeout_queue_hooks< Br_manager_timeout_hooks, Br_manager >::timeout().

10.112.3 Typedef Documentation

10.112.3.1 typedef struct l4_timeout_s l4_timeout_s

Basic timeout specification.

Basically a floating point number with 10 bits mantissa and 5 bits exponent (t = m*2^e).

The timeout can also specify an absolute point in time (bit 16 == 1).

10.112.3.2 typedef union l4_timeout_t l4_timeout_t

Timeout pair.

For IPC there are usually a send and a receive timeout. So this structure contains a pair of timeouts.

10.112.4 Enumeration Type Documentation

10.112.4.1 enum l4_timeout_abs_validity

Intervals of validity for absolute timeouts

Times are actually 2^x values (e.g.

2ms -> 2048µs)

Definition at line 92 of file __timeout.h.

10.112.5 Function Documentation

10.112.5.1 l4_timeout_t l4_ipc_timeout (unsigned *snd_man*, unsigned *snd_exp*, unsigned *rcv_man*, unsigned *rcv_exp*) [inline]

Convert explicit timeout values to l4_timeout_t type.

Parameters

snd_man	Mantissa of send timeout.
snd_exp	Exponent of send timeout.
rcv_man	Mantissa of receive timeout.
rcv_exp	Exponent of receive timeout.

Definition at line 210 of file __timeout.h.

References l4_timeout_t::p, l4_timeout_t::rcv, l4_timeout_t::snd, and l4_timeout_s::t.

10.112.5.2 void l4_rcv_timeout (l4_timeout_s *rcv*, l4_timeout_t * *to*) [inline]

Set receive timeout in given to timeout.

Parameters

rcv	Receive timeout

Return values

to	L4 timeout

Definition at line 238 of file __timeout.h.

10.112.5.3 void l4_snd_timeout (l4_timeout_s *snd*, l4_timeout_t * *to*) [inline]

Set send timeout in given to timeout.

Parameters

snd	Send timeout

Return values

to	L4 timeout

Definition at line 231 of file __timeout.h.

10.112.5.4 l4_timeout_t l4_timeout (l4_timeout_s *snd*, l4_timeout_s *rcv*) [inline]

Combine send and receive timeout in a timeout.

Parameters

snd	Send timeout
rcv	Receive timeout

Returns

L4 timeout

Definition at line 221 of file __timeout.h.

References l4_timeout_t::p, l4_timeout_t::rcv, and l4_timeout_t::snd.

Referenced by L4::Ipc_svr::Timeout_queue_hooks< Br_manager_timeout_hooks, Br_manager >::timeout().

Here is the caller graph for this function:

10.112.5.5 l4_timeout_s l4_timeout_abs (l4_kernel_clock_t *pint*, int *br*) [inline]

Set an absolute timeout.

Parameters

pint	Point in time in clocks
br	The buffer register the timeout shall be placed in. (

Note

On 32bit architectures the timeout needs two consecutive buffers.)
The absolute timeout value will be placed into the buffer register *br* of the current thread.

Returns

timeout value

Definition at line 373 of file utcb.h.

References l4_utcb().

Referenced by L4::Ipc_svr::Timeout_queue_hooks< Br_manager_timeout_hooks, Br_manager >::timeout().

Here is the call graph for this function:

Here is the caller graph for this function:

10.112.5.6 l4_kernel_clock_t l4_timeout_get (l4_kernel_clock_t *cur*, l4_timeout_s *to*) [inline]

Get clock value for a clock + a timeout.

Parameters

cur	Clock value
to	L4 timeout

Returns

Clock sum

Definition at line 268 of file __timeout.h.

References l4_timeout_is_absolute(), and l4_timeout_rel_get().

Here is the call graph for this function:

10.112.5.7 unsigned l4_timeout_is_absolute (l4_timeout_s *to*) [inline]

Return whether the given timeout is absolute or not.

Parameters

to	L4 timeout

Returns

!= 0 if absolute, 0 if relative

Definition at line 261 of file __timeout.h.

Referenced by l4_timeout_get().

Here is the caller graph for this function:

10.112.5.8 l4_timeout_s l4_timeout_rel (unsigned *man*, unsigned *exp*) [inline]

Get relative timeout consisting of mantissa and exponent.

Parameters

man	Mantissa of timeout
exp	Exponent of timeout

Returns

timeout value

Definition at line 245 of file __timeout.h.

10.112.5.9 l4_kernel_clock_t l4_timeout_rel_get (l4_timeout_s *to*) [inline]

Get clock value of out timeout.

Parameters

to	L4 timeout

Returns

Clock value

Definition at line 252 of file __timeout.h.

Referenced by l4_timeout_get().

Here is the caller graph for this function:

10.112.5.10 unsigned l4_utcb_mr64_idx (unsigned *idx*) [inline]

Get index into 64bit message registers alias from native-sized index.

Parameters

idx	Index to native-sized message register

Returns

Index to 64bit message register alias

Definition at line 376 of file utcb.h.

Referenced by l4_icu_msi_info_u().

Here is the caller graph for this function:

10.113 Timestamp Counter

Collaboration diagram for Timestamp Counter:

Files

- file rdtsc.h

  time stamp counter related functions
- file rdtsc.h

  time stamp counter related functions

Macros

- #define L4_TSC_INIT_AUTO 0

  Automatic init.
- #define L4_TSC_INIT_KERNEL 1

  Initialized by kernel.
- #define L4_TSC_INIT_CALIBRATE 2

  Initialized by user-level.
- #define L4_TSC_INIT_AUTO 0

  Automatic init.
- #define L4_TSC_INIT_KERNEL 1

  Initialized by kernel.
- #define L4_TSC_INIT_CALIBRATE 2

  Initialized by user-level.

Functions

- l4_cpu_time_t l4_rdtsc (void)

  Read current value of CPU-internal time stamp counter.
- l4_uint32_t l4_rdtsc_32 (void)

  Read the lest significant 32 bit of the TSC.
- l4_cpu_time_t l4_rdpmc (int nr)

  Return current value of CPU-internal performance measurement counter.
- l4_uint32_t l4_rdpmc_32 (int nr)

  Return the least significant 32 bit of a performance counter.
- l4_uint64_t l4_tsc_to_ns (l4_cpu_time_t tsc)

  Convert time stamp to ns value.
- l4_uint64_t l4_tsc_to_us (l4_cpu_time_t tsc)

  Convert time stamp into micro seconds value.
- void l4_tsc_to_s_and_ns (l4_cpu_time_t tsc, l4_uint32_t *s, l4_uint32_t *ns)

Convert time stamp to s.ns value.

- l4_cpu_time_t l4_ns_to_tsc (l4_uint64_t ns)

  Convert nano seconds into CPU ticks.
- void l4_busy_wait_ns (l4_uint64_t ns)

  Wait busy for a small amount of time.
- void l4_busy_wait_us (l4_uint64_t us)

  Wait busy for a small amount of time.
- l4_uint32_t l4_calibrate_tsc (l4_kernel_info_t *kip)

  Calibrate scalers for time stamp calculations.
- l4_uint32_t l4_tsc_init (int constraint, l4_kernel_info_t *kip)

  Initialitze scaler for TSC calicaltions.
- l4_uint32_t l4_get_hz (void)

  Get CPU frequency in Hz.

10.113.1 Detailed Description

10.113.2 Function Documentation

10.113.2.1 void l4_busy_wait_ns (l4_uint64_t *ns*) [inline]

Wait busy for a small amount of time.

Parameters

ns	nano seconds to wait

Attention

Not intendet for any use!

Definition at line 317 of file rdtsc.h.

References l4_ns_to_tsc(), and l4_rdtsc().

Here is the call graph for this function:

10.113.2.2 void l4_busy_wait_us (l4_uint64_t *us*) [inline]

Wait busy for a small amount of time.

Parameters

us	micro seconds to wait

Attention

Not intendet for any use!

Definition at line 327 of file rdtsc.h.

References l4_ns_to_tsc(), and l4_rdtsc().

Here is the call graph for this function:

10.113.2.3 l4_uint32_t l4_calibrate_tsc (l4_kernel_info_t * *kip*) [inline]

Calibrate scalers for time stamp calculations.

Determine some scalers to be able to convert between real time and CPU ticks. This test uses channel 0 of the PIT (i8254) or the kernel KIP, depending on availability. Just calls l4_tsc_init(L4_TSC_INIT_AUTO).

Examples:

examples/sys/aliens/main.c.

Definition at line 179 of file rdtsc.h.

References l4_tsc_init(), and L4_TSC_INIT_AUTO.

Here is the call graph for this function:

10.113.2.4 l4_uint32_t l4_get_hz (void)

Get CPU frequency in Hz.

Returns

frequency in Hz

10.113.2.5 l4_cpu_time_t l4_ns_to_tsc (l4_uint64_t *ns*) [inline]

Convert nano seconds into CPU ticks.

Parameters

ns	nano seconds

Returns

CPU ticks

Definition at line 303 of file rdtsc.h.

Referenced by l4_busy_wait_ns(), and l4_busy_wait_us().

Here is the caller graph for this function:

10.113.2.6 l4_cpu_time_t l4_rdpmc (int *nr*) [inline]

Return current value of CPU-internal performance measurement counter.

Parameters

nr	Number of counter (0 or 1)

Returns

64-bit PMC

Definition at line 205 of file rdtsc.h.

10.113.2.7 l4_uint32_t l4_rdpmc_32 (int *nr*) [inline]

Return the least significant 32 bit of a performance counter.

Useful for smaller differences, needs less cycles.

Definition at line 227 of file rdtsc.h.

10.113.2.8 l4_cpu_time_t l4_rdtsc (void) [inline]

Read current value of CPU-internal time stamp counter.

Returns

64-bit time stamp

Examples:

examples/sys/aliens/main.c.

Definition at line 185 of file rdtsc.h.

Referenced by l4_busy_wait_ns(), and l4_busy_wait_us().

Here is the caller graph for this function:

10.113.2.9 l4_uint32_t l4_rdtsc_32 (void) [inline]

Read the lest significant 32 bit of the TSC.

Useful for smaller differences, needs less cycles.

Definition at line 246 of file rdtsc.h.

10.113.2.10 l4_uint32_t l4_tsc_init (int *constraint*, l4_kernel_info_t * *kip*)

Initialitze scaler for TSC calicaltions.

Initialize the scalers needed by l4_tsc_to_ns()/l4_ns_to_tsc() and so on. Current versions of Fiasco export these scalers from kernel into userland. The programmer may decide whether he allows to use these scalers or if an calibration should be performed.

Parameters

constraint	programmers constraint: • L4_TSC_INIT_AUTO if the kernel exports the scalers then use them. If not, perform calibration using channel 0 of the PIT (i8254). The latter case may lead into short (unpredictable) periods where interrupts are disabled. • L4_TSC_INIT_KERNEL depend on retrieving the scalers from kernel. If the scalers are not available, return 0. • L4_TSC_INIT_CALIBRATE Ignore possible scalers exported by the scaler, instead insist on calibration using the PIT.
kip	KIP pointer

Returns

0 on error (no scalers exported by kernel, calibrating failed ...) otherwise returns (2^32 / (tsc per µsec)). This value has the same semantics as the value returned by the calibrate_delay_loop() function of the Linux kernel.

Referenced by l4_calibrate_tsc().

Here is the caller graph for this function:

10.113.2.11 l4_uint64_t l4_tsc_to_ns (l4_cpu_time_t *tsc*) [inline]

Convert time stamp to ns value.

Parameters

tsc	time value in CPU ticks

Returns

time value in ns

Examples:

examples/sys/aliens/main.c.

Definition at line 260 of file rdtsc.h.

10.113.2.12 void l4_tsc_to_s_and_ns (l4_cpu_time_t *tsc*, l4_uint32_t * *s*, l4_uint32_t * *ns*) [inline]

Convert time stamp to s.ns value.

Parameters

tsc	time value in CPU ticks

Return values

s	seconds
ns	nano seconds

Definition at line 288 of file rdtsc.h.

10.113.2.13 l4_uint64_t l4_tsc_to_us (l4_cpu_time_t *tsc*) [inline]

Convert time stamp into micro seconds value.

Parameters

tsc	time value in CPU ticks

Returns

time value in micro seconds

Definition at line 274 of file rdtsc.h.

10.114 Utility Functions

Collaboration diagram for Utility Functions:

Modules

- Atomic Instructions
- Bit Manipulation
- Bitmap graphics and fonts

This library provides some functions for bitmap handling in frame buffers.

- CPU related functions
- Comfortable Command Line Parsing
- ELF binary format

  Functions and types related to ELF binaries.
- Functions to manipulate the local IDT
- IA32 Port I/O API
- Internal functions
- Kernel Interface Page API
- Low-Level Thread Functions
- Machine Restarting Function
- Random number support
- Timestamp Counter

Files

- file rand.h

  Simple Pseudo-Random Number Generator.

Functions

- void l4_sleep_forever (void) L4_NOTHROW)

  Go sleep and never wake up.
- long l4util_splitlog2_hdl (l4_addr_t start, l4_addr_t end, long(∗handler)(l4_addr_t s, l4_addr_t e, int log2size))

  Split a range into log2 base and size aligned chunks.
- l4_addr_t l4util_splitlog2_size (l4_addr_t start, l4_addr_t end)

  Return log2 base and size aligned length of a range.
- l4_timeout_s l4util_micros2l4to (unsigned int mus) L4_NOTHROW

  Calculate l4 timeouts.
- void l4_sleep (int ms) L4_NOTHROW

  Suspend thread for a period of ms milliseconds.
- void l4_touch_ro (const void ∗addr, unsigned size) L4_NOTHROW

  Touch data area to force mapping (read-only)
- void l4_touch_rw (const void ∗addr, unsigned size) L4_NOTHROW

  Touch data areas to force mapping (read-write)

10.114.1 Detailed Description

10.114.2 Function Documentation

10.114.2.1 void l4_sleep (int *ms*)

Suspend thread for a period of *ms* milliseconds.

Parameters

ms	Time in milliseconds

Examples:

examples/libs/libirq/async_isr.c, examples/sys/aliens/main.c, examples/sys/singlestep/main.c, and examples/sys/start-with-exc/main.c.

Referenced by L4Re::Namespace::query().

Here is the caller graph for this function:

10.114.2.2 void l4_touch_ro (const void * *addr*, unsigned *size*) [inline]

Touch data area to force mapping (read-only)

Parameters

addr	Start of memory area to touch.
size	Size of area to touch.

Examples:

examples/sys/singlestep/main.c.

Definition at line 93 of file util.h.

References L4_PAGESIZE, and l4_trunc_page().

Here is the call graph for this function:

10.114.2.3 void l4_touch_rw (const void * *addr*, unsigned *size*) [inline]

Touch data areas to force mapping (read-write)

Parameters

addr	Start of memory area to touch.
size	Size of area to touch.

Examples:

examples/sys/aliens/main.c, and examples/sys/singlestep/main.c.

Definition at line 106 of file util.h.

References L4_PAGESIZE, and l4_trunc_page().

Here is the call graph for this function:

10.114.2.4 l4_timeout_s l4util_micros2l4to (unsigned int *mus*)

Calculate l4 timeouts.

Parameters

mus	time in microseconds. Special cases: • 0 - > timeout 0 • ∼0U -> timeout NEVER

Returns

the corresponding l4_timeout value

10.114.2.5 long l4util_splitlog2_hdl (l4_addr_t *start*, l4_addr_t *end*, long(∗)(l4_addr_t s, l4_addr_t e, int log2size) *handler*) [inline]

Split a range into log2 base and size aligned chunks.

Parameters

start	Start of range
end	End of range (inclusive) (e.g. 2-4 is len 3)
handler	Handler function that is called with start and end (both inclusive) of the chunk. On success, the handler must return 0, if it returns !=0 the function will immediately return with the return code of the handler.

Returns

0 on success, != 0 otherwise

Definition at line 53 of file splitlog2.h.

References L4_EINVAL, and l4util_splitlog2_size().

Here is the call graph for this function:

10.114.2.6 l4_addr_t l4util_splitlog2_size (l4_addr_t *start*, l4_addr_t *end*) [inline]

Return log2 base and size aligned length of a range.

Parameters

start	Start of range
end	End of range (inclusive) (e.g. 2-4 is len 3)

Returns

length of elements in log2size (length is 1 << log2size)

Definition at line 72 of file splitlog2.h.

References l4util_bsf(), and l4util_bsr().

Referenced by l4util_splitlog2_hdl().

Here is the call graph for this function:

Here is the caller graph for this function:

10.115 VM API for SVM

Virtual machine API for SVM.

Collaboration diagram for VM API for SVM:

Data Structures

- struct l4_vm_svm_vmcb_control_area

  VMCB structure for SVM VMs.
- struct l4_vm_svm_vmcb_state_save_area_seg

  State save area segment selector struct.
- struct l4_vm_svm_vmcb_state_save_area

  State save area structure for SVM VMs.
- struct l4_vm_svm_vmcb_t

  Control structure for SVM VMs.

Typedefs

- typedef struct l4_vm_svm_vmcb_control_area l4_vm_svm_vmcb_control_area_t

  VMCB structure for SVM VMs.
- typedef struct l4_vm_svm_vmcb_state_save_area_seg l4_vm_svm_vmcb_state_save_area_seg_t

  State save area segment selector struct.
- typedef struct l4_vm_svm_vmcb_state_save_area l4_vm_svm_vmcb_state_save_area_t

  State save area structure for SVM VMs.
- typedef struct l4_vm_svm_vmcb_t l4_vm_svm_vmcb_t

  Control structure for SVM VMs.

10.115.1 Detailed Description

Virtual machine API for SVM.

10.116 VM API for TZ

Virtual Machine API for ARM TrustZone.

Collaboration diagram for VM API for TZ:

Data Structures

- struct l4_vm_tz_state

  state structure for TrustZone VMs

10.116.1 Detailed Description

Virtual Machine API for ARM TrustZone.

10.117 VM API for VMX

Virtual machine API for VMX.

Collaboration diagram for VM API for VMX:

Enumerations

- enum L4_vm_vmx_caps_regs {
  L4_VM_VMX_BASIC_REG = 0, L4_VM_VMX_TRUE_PINBASED_CTLS_REG = 1, L4_VM_VMX_TRUE_↩ PROCBASED_CTLS_REG = 2, L4_VM_VMX_TRUE_EXIT_CTLS_REG = 3,
  L4_VM_VMX_TRUE_ENTRY_CTLS_REG = 4, L4_VM_VMX_MISC_REG = 5, L4_VM_VMX_CR0_FIXE↩ D0_REG = 6, L4_VM_VMX_CR0_FIXED1_REG = 7,
  L4_VM_VMX_CR4_FIXED0_REG = 8, L4_VM_VMX_CR4_FIXED1_REG = 9, L4_VM_VMX_VMCS_ENU↩ M_REG = 0xa, L4_VM_VMX_PROCBASED_CTLS2_REG = 0xb,
  L4_VM_VMX_EPT_VPID_CAP_REG = 0xc, L4_VM_VMX_NUM_CAPS_REGS }

  Exported VMX capability registers.
- enum L4_vm_vmx_dfl1_regs {
  L4_VM_VMX_PINBASED_CTLS_DFL1_REG = 0x1, L4_VM_VMX_PROCBASED_CTLS_DFL1_REG = 0x2, L4_VM_VMX_EXIT_CTLS_DFL1_REG = 0x3, L4_VM_VMX_ENTRY_CTLS_DFL1_REG = 0x4,
  L4_VM_VMX_NUM_DFL1_REGS }

  Exported VMX capability registers (default to 1 bits).
- enum {
  L4_VM_VMX_VMCS_CR2 = 0x683e, L4_VM_VMX_VMCS_XCR0 = 0x6842, L4_VM_VMX_VMCS_MSR_↩ SYSCALL_MASK = 0x6844, L4_VM_VMX_VMCS_MSR_LSTAR = 0x6846,
  L4_VM_VMX_VMCS_MSR_CSTAR = 0x6848, L4_VM_VMX_VMCS_MSR_TSC_AUX = 0x6850, L4_VM_↩ VMX_VMCS_MSR_STAR = 0x6852, L4_VM_VMX_VMCS_MSR_KERNEL_GS_BASE = 0x6854 }

  Additional (virtual) VMCS fields.

Functions

- l4_uint64_t l4_vm_vmx_get_caps (void const ∗vcpu_state, unsigned cap_msr) L4_NOTHROW

  Get a capability register for VMX.
- l4_uint32_t l4_vm_vmx_get_caps_default1 (void const ∗vcpu_state, unsigned cap_msr) L4_NOTHROW

  Get a default to one capability register for VMX.
- unsigned l4_vm_vmx_field_len (unsigned field) L4_NOTHROW

  Return length in bytes of a VMCS field.
- unsigned l4_vm_vmx_field_order (unsigned field) L4_NOTHROW

  Return length in power of two (bytes) of a VMCS field.
- void l4_vm_vmx_clear (void ∗vmcs, void ∗user_vmcs) L4_NOTHROW

  Saves cached state from the kernel VMCS to the user VMCS.
- void l4_vm_vmx_ptr_load (void ∗vmcs, void ∗user_vmcs) L4_NOTHROW

  Loads the user_vmcs as the current VMCS.
- l4_uint32_t l4_vm_vmx_get_cr2_index (void const ∗vmcs) L4_NOTHROW

Get the VMCS field index of the virtual CR2 register.

- l4_umword_t l4_vm_vmx_read_nat (void *vmcs, unsigned field) L4_NOTHROW

  Read a natural width VMCS field.
- l4_uint16_t l4_vm_vmx_read_16 (void *vmcs, unsigned field) L4_NOTHROW

  Read a 16bit VMCS field.
- l4_uint32_t l4_vm_vmx_read_32 (void *vmcs, unsigned field) L4_NOTHROW

  Read a 32bit VMCS field.
- l4_uint64_t l4_vm_vmx_read_64 (void *vmcs, unsigned field) L4_NOTHROW

  Read a 64bit VMCS field.
- l4_uint64_t l4_vm_vmx_read (void *vmcs, unsigned field) L4_NOTHROW

  Read any VMCS field.
- void l4_vm_vmx_write_nat (void *vmcs, unsigned field, l4_umword_t val) L4_NOTHROW

  Write to a natural width VMCS field.
- void l4_vm_vmx_write_16 (void *vmcs, unsigned field, l4_uint16_t val) L4_NOTHROW

  Write to a 16bit VMCS field.
- void l4_vm_vmx_write_32 (void *vmcs, unsigned field, l4_uint32_t val) L4_NOTHROW

  Write to a 32bit VMCS field.
- void l4_vm_vmx_write_64 (void *vmcs, unsigned field, l4_uint64_t val) L4_NOTHROW

  Write to a 64bit VMCS field.
- void l4_vm_vmx_write (void *vmcs, unsigned field, l4_uint64_t val) L4_NOTHROW

  Write to an arbitrary VMCS field.

10.117.1 Detailed Description

Virtual machine API for VMX.

10.117.2 Enumeration Type Documentation

10.117.2.1 anonymous enum

Additional (virtual) VMCS fields.

The VMCS offsets defined here are actually not in the hardware VMCS. However our VMMs run in user mode and need to have access to certain registers available in kernel mode only. So we put them into our version of the VMCS.

Enumerator

L4_VM_VMX_VMCS_CR2 VMCS offset for CR2.

Note

You usually need to check this value against the value you get from l4_vm_vmx_get_cr2_index() to make sure you are running on a compatible kernel.

L4_VM_VMX_VMCS_XCR0 VMCS offset of extended control register XCR0.

L4_VM_VMX_VMCS_MSR_SYSCALL_MASK VMCS offset of system call flag mask MSR.

L4_VM_VMX_VMCS_MSR_LSTAR VMCS offset of IA32e mode system call target address MSR.

L4_VM_VMX_VMCS_MSR_CSTAR VMCS offset of IA32 mode system call target address MSR.

L4_VM_VMX_VMCS_MSR_TSC_AUX VMCS offset of auxiliary TSC signature MSR.

L4_VM_VMX_VMCS_MSR_STAR VMCS offset of system call target address MSR.

L4_VM_VMX_VMCS_MSR_KERNEL_GS_BASE VMCS offset of GS base address swap target MSR.

Definition at line 105 of file __vm-vmx.h.

10.117.2.2 enum L4_vm_vmx_caps_regs

Exported VMX capability registers.

Enumerator

L4_VM_VMX_BASIC_REG Basic VMX capabilities.

L4_VM_VMX_TRUE_PINBASED_CTLS_REG True pin-based control caps.

L4_VM_VMX_TRUE_PROCBASED_CTLS_REG True processor based control caps.

L4_VM_VMX_TRUE_EXIT_CTLS_REG True exit control caps.

L4_VM_VMX_TRUE_ENTRY_CTLS_REG True entry control caps.

L4_VM_VMX_MISC_REG Misc caps.

L4_VM_VMX_CR0_FIXED0_REG Fixed to 0 bits of CR0.

L4_VM_VMX_CR0_FIXED1_REG Fixed to 1 bits of CR0.

L4_VM_VMX_CR4_FIXED0_REG Fixed to 0 bits of CR4.

L4_VM_VMX_CR4_FIXED1_REG Fixed to 1 bits of CR4.

L4_VM_VMX_VMCS_ENUM_REG VMCS enumeration info.

L4_VM_VMX_PROCBASED_CTLS2_REG Processor based control 2 caps.

L4_VM_VMX_EPT_VPID_CAP_REG EPT and VPID caps.

L4_VM_VMX_NUM_CAPS_REGS Total number of VMX capability registers.

Definition at line 39 of file __vm-vmx.h.

10.117.2.3 enum L4_vm_vmx_dfl1_regs

Exported VMX capability registers (default to 1 bits).

Enumerator

L4_VM_VMX_PINBASED_CTLS_DFL1_REG Default 1 bits in pin-based controls.

L4_VM_VMX_PROCBASED_CTLS_DFL1_REG Default 1 bits in processor-based controls.

L4_VM_VMX_EXIT_CTLS_DFL1_REG Default 1 bits in exit controls.

L4_VM_VMX_ENTRY_CTLS_DFL1_REG Default 1 bits in entry controls.

L4_VM_VMX_NUM_DFL1_REGS Total number of default on registers.

Definition at line 62 of file __vm-vmx.h.

10.117.3 Function Documentation

10.117.3.1 void l4_vm_vmx_clear (void * *vmcs*, void * *user_vmcs*) [inline]

Saves cached state from the kernel VMCS to the user VMCS.

Parameters

vmcs	Pointer to the kernel VMCS.
user_vmcs	Pointer to the user VMCS.

This function is comparable to VMX vmclear.

Definition at line 426 of file __vm-vmx.h.

Referenced by l4_vm_vmx_ptr_load().

Here is the caller graph for this function:

10.117.3.2 unsigned l4_vm_vmx_field_len (unsigned *field*) [inline]

Return length in bytes of a VMCS field.

Parameters

field	Field number.

Returns

Width of field in bytes.

Definition at line 350 of file __vm-vmx.h.

References l4_vm_vmx_field_order().

Here is the call graph for this function:

10.117.3.3 unsigned l4_vm_vmx_field_order (unsigned *field*) [inline]

Return length in power of two (bytes) of a VMCS field.

Parameters

field	Field number.

Returns

Width of field in power of two (bytes).

Definition at line 335 of file __vm-vmx.h.

Referenced by l4_vm_vmx_field_len().

Here is the caller graph for this function:

10.117.3.4 l4_uint64_t l4_vm_vmx_get_caps (void const * *vcpu_state*, unsigned *cap_msr*) [inline]

Get a capability register for VMX.

Parameters

vcpu_state	Pointer to the VCPU state of the VCPU.
cap_msr	Caps register index (see L4_vm_vmx_caps_regs).

Returns

The value of the capability register.

Definition at line 522 of file __vm-vmx.h.

References L4_VCPU_OFFSET_EXT_INFOS.

10.117.3.5 l4_uint32_t l4_vm_vmx_get_caps_default1 (void const * *vcpu_state*, unsigned *cap_msr*) [inline]

Get a default to one capability register for VMX.

Parameters

vcpu_state	Pointer to the VCPU state of the VCPU.
cap_msr	Default 1 caps register index (see L4_vm_vmx_dfl1_regs).

Returns

The value of the capability register.

Definition at line 530 of file __vm-vmx.h.

References L4_VCPU_OFFSET_EXT_INFOS, L4_VM_VMX_NUM_CAPS_REGS, and L4_VM_VMX_PINBASE↩D_CTLS_DFL1_REG.

10.117.3.6 l4_uint32_t l4_vm_vmx_get_cr2_index (void const * *vmcs*) [inline]

Get the VMCS field index of the virtual CR2 register.

Parameters

vmcs	Pointer to the software VMCS.

Returns

The field index used for the virtual CR2 register as used by the current Fiasco.OC interface.

The CR2 register is actually not in the hardware VMCS, however our VMMs run in user mode and need to have access to this register so we put it into our software version of the VMCS.

See also

L4_VM_VMX_VMCS_CR2

Definition at line 538 of file __vm-vmx.h.

10.117.3.7 void l4_vm_vmx_ptr_load (void * *vmcs*, void * *user_vmcs*) [inline]

Loads the user_vmcs as the current VMCS.

Parameters

vmcs	Pointer to the kernel VMCS.
user_vmcs	Pointer to the user VMCS.

This function is comparable to VMX vmptrld.

Definition at line 438 of file __vm-vmx.h.

References l4_vm_vmx_clear().

Here is the call graph for this function:

10.117.3.8 l4_uint64_t l4_vm_vmx_read (void * *vmcs*, unsigned *field*) [inline]

Read any VMCS field.

Parameters

vmcs	Pointer to the software VMCS.
field	The VMCS field index as used on VMX hardware.

Returns

The value of the VMCS field with the given index.

Definition at line 474 of file __vm-vmx.h.

References l4_vm_vmx_read_16(), l4_vm_vmx_read_32(), l4_vm_vmx_read_64(), and l4_vm_vmx_read_nat().

Here is the call graph for this function:

10.117.3.9 l4_uint16_t l4_vm_vmx_read_16 (void * *vmcs*, unsigned *field*) [inline]

Read a 16bit VMCS field.

Parameters

vmcs	Pointer to the software VMCS.
field	The VMCS field index as used on VMX hardware.

Returns

The value of the VMCS field with the given index.

Definition at line 459 of file __vm-vmx.h.

Referenced by l4_vm_vmx_read().

Here is the caller graph for this function:

10.117.3.10 l4_uint32_t l4_vm_vmx_read_32 (void * *vmcs*, unsigned *field*) [inline]

Read a 32bit VMCS field.

Parameters

vmcs	Pointer to the software VMCS.
field	The VMCS field index as used on VMX hardware.

Returns

The value of the VMCS field with the given index.

Definition at line 464 of file __vm-vmx.h.

Referenced by l4_vm_vmx_read().

Here is the caller graph for this function:

10.117.3.11 l4_uint64_t l4_vm_vmx_read_64 (void * *vmcs*, unsigned *field*) [inline]

Read a 64bit VMCS field.

Parameters

vmcs	Pointer to the software VMCS.
field	The VMCS field index as used on VMX hardware.

Returns

The value of the VMCS field with the given index.

Definition at line 469 of file __vm-vmx.h.

Referenced by l4_vm_vmx_read().

Here is the caller graph for this function:

10.117.3.12 l4_umword_t l4_vm_vmx_read_nat (void * *vmcs*, unsigned *field*) [inline]

Read a natural width VMCS field.

Parameters

vmcs	Pointer to the software VMCS.
field	The VMCS field index as used on VMX hardware.

Returns

The value of the VMCS field with the given index.

Definition at line 454 of file __vm-vmx.h.

Referenced by l4_vm_vmx_read().

Here is the caller graph for this function:

10.117.3.13 void l4_vm_vmx_write (void * *vmcs*, unsigned *field*, l4_uint64_t *val*) [inline]

Write to an arbitrary VMCS field.

Parameters

vmcs	Pointer to the software VMCS.
field	The VMCS field index as used on VMX hardware.
val	The value that shall be written to the given field.

Definition at line 509 of file __vm-vmx.h.

References l4_vm_vmx_write_16(), l4_vm_vmx_write_32(), l4_vm_vmx_write_64(), and l4_vm_vmx_write_nat().

Here is the call graph for this function:

10.117.3.14 void l4_vm_vmx_write_16 (void * *vmcs*, unsigned *field*, l4_uint16_t *val*) [inline]

Write to a 16bit VMCS field.

Parameters

vmcs	Pointer to the software VMCS.
field	The VMCS field index as used on VMX hardware.
val	The value that shall be written to the given field.

Definition at line 493 of file __vm-vmx.h.

Referenced by l4_vm_vmx_write().

Here is the caller graph for this function:

10.117.3.15 void l4_vm_vmx_write_32 (void * *vmcs*, unsigned *field*, l4_uint32_t *val*) [inline]

Write to a 32bit VMCS field.

Parameters

vmcs	Pointer to the software VMCS.
field	The VMCS field index as used on VMX hardware.
val	The value that shall be written to the given field.

Definition at line 498 of file __vm-vmx.h.

Referenced by l4_vm_vmx_write().

Here is the caller graph for this function:

10.117.3.16 void l4_vm_vmx_write_64 (void * *vmcs*, unsigned *field*, l4_uint64_t *val*) [inline]

Write to a 64bit VMCS field.

Parameters

vmcs	Pointer to the software VMCS.
field	The VMCS field index as used on VMX hardware.
val	The value that shall be written to the given field.

Definition at line 503 of file __vm-vmx.h.

Referenced by l4_vm_vmx_write().

Here is the caller graph for this function:

10.117.3.17 void l4_vm_vmx_write_nat (void * *vmcs*, unsigned *field*, l4_umword_t *val*) [inline]

Write to a natural width VMCS field.

Parameters

vmcs	Pointer to the software VMCS.
field	The VMCS field index as used on VMX hardware.
val	The value that shall be written to the given field.

Definition at line 488 of file __vm-vmx.h.

Referenced by l4_vm_vmx_write().

Here is the caller graph for this function:

10.118 Video API

Collaboration diagram for Video API:

Data Structures

- struct l4re_video_color_component_t

  Color component structure.
- struct l4re_video_pixel_info_t

  Pixel_info structure.
- struct l4re_video_goos_info_t

  Goos information structure.
- struct l4re_video_view_info_t

  View information structure.
- struct l4re_video_view_t

  C representation of a goos view.

Typedefs

- typedef struct l4re_video_color_component_t l4re_video_color_component_t

  Color component structure.
- typedef struct l4re_video_pixel_info_t l4re_video_pixel_info_t

  Pixel_info structure.
- typedef struct l4re_video_view_info_t l4re_video_view_info_t

  View information structure.
- typedef struct l4re_video_view_t l4re_video_view_t

  C representation of a goos view.

Enumerations

- enum l4re_video_goos_info_flags_t { F_l4re_video_goos_auto_refresh = 0x01, F_l4re_video_goos_pointer = 0x02, F_l4re_video_goos_dynamic_views = 0x04, F_l4re_video_goos_dynamic_buffers = 0x08 }

  Flags of information on the goos.
- enum l4re_video_view_info_flags_t { F_l4re_video_view_none = 0x00, F_l4re_video_view_set_buffer = 0x01, F_l4re_video_view_set_buffer_offset = 0x02, F_l4re_video_view_set_bytes_per_line = 0x04, F_l4re_video_view_set_pixel = 0x08, F_l4re_video_view_set_position = 0x10, F_l4re_video_view_dyn_allocated = 0x20, F_l4re_video_view_set_background = 0x40, F_l4re_video_view_set_flags = 0x80 , F_l4re_video_view_above = 0x01000, F_l4re_video_view_flags_mask = 0xff000 }

  Flags of information on a view.

Functions

- int l4re_video_goos_info (l4re_video_goos_t goos, l4re_video_goos_info_t *ginfo) L4_NOTHROW

  Get information on a goos.
- int l4re_video_goos_refresh (l4re_video_goos_t goos, int x, int y, int w, int h) L4_NOTHROW

  Flush a rectangle of pixels of the goos screen.
- int l4re_video_goos_create_buffer (l4re_video_goos_t goos, unsigned long size, l4_cap_idx_t buffer) L4_NOTHROW

  Create a new buffer (memory buffer) for pixel data.
- int l4re_video_goos_delete_buffer (l4re_video_goos_t goos, unsigned idx) L4_NOTHROW

  Delete a pixel buffer.
- int l4re_video_goos_get_static_buffer (l4re_video_goos_t goos, unsigned idx, l4_cap_idx_t buffer) L4_NOTHROW

  Get the data-space capability of the static pixel buffer.
- int l4re_video_goos_create_view (l4re_video_goos_t goos, l4re_video_view_t *view) L4_NOTHROW

  Create a new view (.
- int l4re_video_goos_delete_view (l4re_video_goos_t goos, l4re_video_view_t *view) L4_NOTHROW

  Delete a view.
- int l4re_video_goos_get_view (l4re_video_goos_t goos, unsigned idx, l4re_video_view_t *view) L4_NOTHROW

  Get a view for the given index.
- int l4re_video_view_refresh (l4re_video_view_t *view, int x, int y, int w, int h) L4_NOTHROW

  Flush the given rectangle of pixels of the given view.
- int l4re_video_view_get_info (l4re_video_view_t *view, l4re_video_view_info_t *info) L4_NOTHROW

  Retrieve information about the given view.
- int l4re_video_view_set_info (l4re_video_view_t *view, l4re_video_view_info_t *info) L4_NOTHROW

  Set properties of the view.
- int l4re_video_view_set_viewport (l4re_video_view_t *view, int x, int y, int w, int h, unsigned long bofs) L4_NOTHROW

  Set the viewport parameters of a view.
- int l4re_video_view_stack (l4re_video_view_t *view, l4re_video_view_t *pivot, int behind) L4_NOTHROW

  Change the stacking order in the stack of visible views.

10.118.1 Detailed Description

10.118.2 Typedef Documentation

10.118.2.1 typedef struct l4re_video_view_t l4re_video_view_t

C representation of a goos view.

A view is a visible rectangle that provides a view to the contents of a buffer (frame buffer) memory object and is placed on a real screen.

10.118.3 Enumeration Type Documentation

10.118.3.1 enum l4re_video_goos_info_flags_t

Flags of information on the goos.

Enumerator

F_l4re_video_goos_auto_refresh The graphics display is automatically refreshed.

F_l4re_video_goos_pointer We have a mouse pointer.

F_l4re_video_goos_dynamic_views Supports dynamically allocated views.

F_l4re_video_goos_dynamic_buffers Supports dynamically allocated buffers.

Definition at line 39 of file goos.h.

10.118.3.2 enum l4re_video_view_info_flags_t

Flags of information on a view.

Enumerator

F_l4re_video_view_none everything for this view is static (the VESA-FB case)

F_l4re_video_view_set_buffer buffer object for this view can be changed

F_l4re_video_view_set_buffer_offset buffer offset can be set

F_l4re_video_view_set_bytes_per_line bytes per line can be set

F_l4re_video_view_set_pixel pixel type can be set

F_l4re_video_view_set_position position on screen can be set

F_l4re_video_view_dyn_allocated View is dynamically allocated.

F_l4re_video_view_set_background Set view as background for session.

F_l4re_video_view_set_flags Set view property flags.

F_l4re_video_view_above Flag the view as stay on top.

F_l4re_video_view_flags_mask Mask containing all possible property flags.

Definition at line 33 of file view.h.

10.118.4 Function Documentation

10.118.4.1 int l4re_video_goos_create_buffer (l4re_video_goos_t *goos*, unsigned long *size*, l4_cap_idx_t *buffer*)

Create a new buffer (memory buffer) for pixel data.

Parameters

goos	the target object for the operation.
size	the size in bytes for the pixel buffer.
buffer	a capability index to receive the data-space capability for the buffer.

Returns

>=0: The index of the created buffer (used to assign views and for deletion). < 0: on error

10.118.4.2 int l4re_video_goos_create_view (l4re_video_goos_t *goos*, l4re_video_view_t * *view*)

Create a new view (.

See also

l4re_video_view_t)

Parameters

goos	the goos session to use.

Return values

view	the structure will be initialized for the new view.

10.118.4.3 int l4re_video_goos_delete_buffer (l4re_video_goos_t *goos*, unsigned *idx*)

Delete a pixel buffer.

Parameters

goos	the target goos object.
idx	the buffer index of the buffer to delete (the return value of l4re_video_goos_create_buffer())

10.118.4.4 int l4re_video_goos_delete_view (l4re_video_goos_t *goos*, l4re_video_view_t * *view*)

Delete a view.

Parameters

goos	the goos session to use.
view	the view to delete, the given data-structure is invalid afterwards.

10.118.4.5 int l4re_video_goos_get_static_buffer (l4re_video_goos_t *goos*, unsigned *idx*, l4_cap_idx_t *buffer*)

Get the data-space capability of the static pixel buffer.

Parameters

goos	The target goos object.
idx	Index of the static buffer.
buffer	A capability index to receive the data-space capability.

This function allows access to static, preexisting pixel buffers. Such static buffers exist for static configurations, such as the VESA framebuffer.

10.118.4.6 int l4re_video_goos_get_view (l4re_video_goos_t *goos*, unsigned *idx*, l4re_video_view_t * *view*)

Get a view for the given index.

Parameters

goos	the target goos session.
idx	the index of the view to retrieve.

Return values

view	the structure will be initialized to the view with the given index.

This function allows to access static views as provided by the VESA framebuffer (the monitor). However, it also allows to access dynamic views created with l4re_video_goos_create_view().

10.118.4.7 int l4re_video_goos_info (l4re_video_goos_t *goos*, l4re_video_goos_info_t * *ginfo*)

Get information on a goos.

Parameters

goos	Goos object

Return values

ginfo	Pointer to goos information structure.

Returns

0 for success, <0 on error

- -L4_ENODEV
- IPC errors

10.118.4.8 int l4re_video_goos_refresh (l4re_video_goos_t *goos*, int *x*, int *y*, int *w*, int *h*)

Flush a rectangle of pixels of the goos screen.

Parameters

goos	the target object of the operation.
x	the x-coordinate of the upper left corner of the rectangle
y	the y-coordinate of the upper left corner of the rectangle
w	the width of the rectangle to be flushed
h	the height of the rectangle

10.118.4.9 int l4re_video_view_get_info (l4re_video_view_t * *view*, l4re_video_view_info_t * *info*)

Retrieve information about the given *view*.

Parameters

view	the target view for the operation.

Return values

info	a buffer receiving the information about the view.

10.118.4.10 int l4re_video_view_refresh (l4re_video_view_t * *view*, int *x*, int *y*, int *w*, int *h*)

Flush the given rectangle of pixels of the given *view*.

Parameters

view	the target view of the operation.
x	x-coordinate of the upper left corner
y	y-coordinate of the upper left corner
w	the width of the rectangle
h	the height of the rectangle

10.118.4.11 int l4re_video_view_set_info (l4re_video_view_t * *view*, l4re_video_view_info_t * *info*)

Set properties of the view.

Parameters

view	the target view of the operation.
info	the parameters to be set on the view.

Which parameters can be manipulated on a given view can be figured out with l4re_video_view_get_info() and this depends on the concrete instance the view object.

10.118.4.12 int l4re_video_view_set_viewport (l4re_video_view_t * *view*, int *x*, int *y*, int *w*, int *h*, unsigned long *bofs*)

Set the viewport parameters of a view.

Parameters

view	the target view of the operation.
x	the x-coordinate of the upper left corner on the screen.
y	the y-coordinate of the upper left corner on the screen.
w	the width of the view.
h	the height of the view.
bofs	the offset (in bytes) of the upper left pixel in the memory buffer

This function is a convenience wrapper for l4re_video_view_set_info(), just setting the often changed parameters of a dynamic view. With this function a view can be placed on the real screen and at the same time on its backing buffer.

10.118.4.13 int l4re_video_view_stack (l4re_video_view_t * *view*, l4re_video_view_t * *pivot*, int *behind*)

Change the stacking order in the stack of visible views.

Parameters

view	the target view for the operation.
pivot	the neighbor view, relative to which *view* shall be stacked. a NULL value allows top (*behind* = 1) and bottom (*behind* = 0) placement of the view.
behind	describes the placement of the view relative to the *pivot* view.

10.119 Virtual Console

Virtual console for simple character based input and output.

Collaboration diagram for Virtual Console:

Data Structures

- struct l4_vcon_attr_t

  Vcon attribute structure.

Typedefs

- typedef struct l4_vcon_attr_t l4_vcon_attr_t

  Vcon attribute structure.

Enumerations

- enum L4_vcon_size_consts { L4_VCON_WRITE_SIZE = (L4_UTCB_GENERIC_DATA_SIZE - 2) ∗ sizeof(l4_umword_t), L4_VCON_READ_SIZE = (L4_UTCB_GENERIC_DATA_SIZE - 1) ∗ sizeof(l4_umword_t) }

  Size constants.
- enum L4_vcon_i_flags { L4_VCON_INLCR = 000100, L4_VCON_IGNCR = 000200, L4_VCON_ICRNL = 000400 }

  Input flags.
- enum L4_vcon_o_flags { L4_VCON_ONLCR = 000004, L4_VCON_OCRNL = 000010, L4_VCON_ONLRET = 000040 }

  Output flags.
- enum L4_vcon_l_flags { L4_VCON_ICANON = 000002, L4_VCON_ECHO = 000010 }

  Local flags.

Functions

- l4_msgtag_t l4_vcon_send (l4_cap_idx_t vcon, char const ∗buf, int size) L4_NOTHROW

  Send data to virtual console.
- l4_msgtag_t l4_vcon_send_u (l4_cap_idx_t vcon, char const ∗buf, int size, l4_utcb_t ∗utcb) L4_NOTHROW

  Send data to `this` virtual console.
- long l4_vcon_write (l4_cap_idx_t vcon, char const ∗buf, int size) L4_NOTHROW

  Write data to virtual console.
- long l4_vcon_write_u (l4_cap_idx_t vcon, char const ∗buf, int size, l4_utcb_t ∗utcb) L4_NOTHROW

  Write data to `this` virtual console.
- int l4_vcon_read (l4_cap_idx_t vcon, char ∗buf, int size) L4_NOTHROW

Read data from virtual console.

- int l4_vcon_read_u (l4_cap_idx_t vcon, char *buf, int size, l4_utcb_t *utcb) L4_NOTHROW

  Read data from `this` virtual console.

- int l4_vcon_read_with_flags (l4_cap_idx_t vcon, char *buf, int size) L4_NOTHROW

  Read data from virtual console, extended version including flags.

- l4_msgtag_t l4_vcon_set_attr (l4_cap_idx_t vcon, l4_vcon_attr_t const *attr) L4_NOTHROW

  Set attributes of a Vcon.

- l4_msgtag_t l4_vcon_set_attr_u (l4_cap_idx_t vcon, l4_vcon_attr_t const *attr, l4_utcb_t *utcb) L4_NOTHROW

  Set the attributes of `this` virtual console.

- l4_msgtag_t l4_vcon_get_attr (l4_cap_idx_t vcon, l4_vcon_attr_t *attr) L4_NOTHROW

  Get attributes of a Vcon.

- l4_msgtag_t l4_vcon_get_attr_u (l4_cap_idx_t vcon, l4_vcon_attr_t *attr, l4_utcb_t *utcb) L4_NOTHROW

  Get attributes of `this` virtual console.

10.119.1 Detailed Description

Virtual console for simple character based input and output.

The interrupt for read events is provided by the virtual key interrupt.

Include File

```
#include <l4/sys/vcon.h>
```

See L4::Vcon for the C++ interface.

10.119.2 Enumeration Type Documentation

10.119.2.1 enum L4_vcon_i_flags

Input flags.

Enumerator

L4_VCON_INLCR Translate NL to CR.

L4_VCON_IGNCR Ignore CR.

L4_VCON_ICRNL Translate CR to NL if L4_VCON_IGNCR is not set.

Definition at line 184 of file vcon.h.

10.119.2.2 enum L4_vcon_l_flags

Local flags.

Enumerator

L4_VCON_ICANON Canonical mode.

L4_VCON_ECHO Echo input.

Definition at line 206 of file vcon.h.

10.119.2.3 enum L4_vcon_o_flags

Output flags.

Enumerator

L4_VCON_ONLCR Translate NL to CR-NL.
L4_VCON_OCRNL Translate CR to NL.
L4_VCON_ONLRET Do not output CR.

Definition at line 195 of file vcon.h.

10.119.2.4 enum L4_vcon_size_consts

Size constants.

Enumerator

L4_VCON_WRITE_SIZE Maximum size that can be written with one l4_vcon_write call.
L4_VCON_READ_SIZE Maximum size that can be read with one l4_vcon_read∗ call.

Definition at line 92 of file vcon.h.

10.119.3 Function Documentation

10.119.3.1 l4_msgtag_t l4_vcon_get_attr (l4_cap_idx_t *vcon*, l4_vcon_attr_t ∗ *attr*) [inline]

Get attributes of a Vcon.

Parameters

	vcon	Vcon object.
out	*attr*	Attribute structure.

Returns

Syscall return tag

Definition at line 404 of file vcon.h.

References l4_utcb(), and l4_vcon_get_attr_u().

Here is the call graph for this function:

10.119.3.2 l4_msgtag_t l4_vcon_get_attr_u (l4_cap_idx_t *vcon*, l4_vcon_attr_t ∗ *attr*, l4_utcb_t ∗ *utcb*) [inline]

Get attributes of `this` virtual console.

Parameters

	vcon	Capability index of the vcon object.
`out`	*attr*	Attribute structure. Contains the attributes after a successfull call of this function.
	utcb	UTCB pointer of the calling thread.

Returns

Syscall return tag.

Definition at line 386 of file vcon.h.

References l4_ipc_call(), L4_IPC_NEVER, l4_msgtag(), L4_PROTO_LOG, L4_VCON_GET_ATTR_OP, and l4_msg_regs_t::mr.

Referenced by L4::Vcon::get_attr(), and l4_vcon_get_attr().

Here is the call graph for this function:

Here is the caller graph for this function:

10.119.3.3 int l4_vcon_read (l4_cap_idx_t *vcon*, char * *buf*, int *size*) [inline]

Read data from virtual console.

Parameters

	vcon	Vcon object.
out	*buf*	Pointer to data buffer.
	size	Size of buffer in bytes.

Return values

<0	Error code.
>size	If more bytes are to read, `size` bytes are in the buffer `buf`.
<=size	Number of bytes read.

Note

Size must not exceed L4_VCON_READ_SIZE.

Definition at line 360 of file vcon.h.

References l4_utcb(), and l4_vcon_read_u().

Here is the call graph for this function:

10.119.3.4 int l4_vcon_read_u (l4_cap_idx_t *vcon*, char * *buf*, int *size*, l4_utcb_t * *utcb*) [inline]

Read data from `this` virtual console.

Parameters

	vcon	Capability index of the vcon object.
out	*buf*	Pointer to data buffer.
	size	Size of the data buffer in bytes.
	utcb	UTCB pointer of the calling thread.

Return values

<0	Error code.
>size	More bytes to read, `size` bytes are in the buffer `buf`.
<=size	Number of bytes read.

Note

Size must not exceed L4_VCON_READ_SIZE.

Definition at line 350 of file vcon.h.

Referenced by l4_vcon_read(), and L4::Vcon::read().

Here is the caller graph for this function:

10.119.3.5 int l4_vcon_read_with_flags (l4_cap_idx_t *vcon*, char * *buf*, int *size*) [inline]

Read data from virtual console, extended version including flags.

Parameters

	vcon	Vcon object.
out	*buf*	Pointer to data buffer.
	size	Size of buffer in bytes.

If this function returns a positive value the caller can check the #L4_VCON_READ_STAT_BREAK flag bit for a break condition. The bytes read can be obtained by masking the return value with #L4_VCON_READ_SIZE_MASK.

If a break condition is signaled, it is always the first event in the transmitted content, i.e. all characters supplied by this read call follow the break condition.

Note

Size must not exceed L4_VCON_READ_SIZE.

Return values

<0	Error code.
>size	More bytes to read, `size` bytes are in the buffer `buf`.
<=size	Number of bytes read.

Definition at line 344 of file vcon.h.

References l4_utcb().

Here is the call graph for this function:

10.119.3.6 l4_msgtag_t l4_vcon_send (l4_cap_idx_t *vcon*, char const * *buf*, int *size*) [inline]

Send data to virtual console.

Parameters

vcon	Vcon object.
buf	Pointer to data buffer.
size	Size of buffer in bytes.

Returns

Syscall return tag

Note

Size must not exceed L4_VCON_WRITE_SIZE, a proper value of the `size` parameter is NOT checked.

Definition at line 284 of file vcon.h.

References l4_utcb(), and l4_vcon_send_u().

Here is the call graph for this function:

10.119.3.7 l4_msgtag_t l4_vcon_send_u (l4_cap_idx_t *vcon*, char const * *buf*, int *size*, l4_utcb_t * *utcb*) [inline]

Send data to `this` virtual console.

Parameters

vcon	Capability index of the Vcon object.
buf	Pointer to the data buffer.
size	Size of the data buffer in bytes.
utcb	UTBC pointer of the calling thread.

Returns

Syscall return tag

Note

Size must not exceed L4_VCON_WRITE_SIZE, a proper value of the `size` parameter is NOT checked.

Definition at line 270 of file vcon.h.

References L4_IPC_NEVER, l4_ipc_send(), l4_msgtag(), L4_MSGTAG_SCHEDULE, L4_PROTO_LOG, L4_VCON_WRITE_OP, and l4_msg_regs_t::mr.

Referenced by l4_vcon_send(), l4_vcon_write_u(), and L4::Vcon::send().

Here is the call graph for this function:

Here is the caller graph for this function:

10.119.3.8 l4_msgtag_t l4_vcon_set_attr (l4_cap_idx_t *vcon*, l4_vcon_attr_t const * *attr*) [inline]

Set attributes of a Vcon.

Parameters

vcon	Vcon object.
attr	Attribute structure.

Returns

Syscall return tag

Definition at line 380 of file vcon.h.

References l4_utcb(), and l4_vcon_set_attr_u().

Here is the call graph for this function:

10.119.3.9 l4_msgtag_t l4_vcon_set_attr_u (l4_cap_idx_t *vcon*, l4_vcon_attr_t const * *attr*, l4_utcb_t * *utcb*) [inline]

Set the attributes of `this` virtual console.

Parameters

vcon	Capability index of the vcon object.
attr	Attribute structure with the attributes for the virtual console.
utcb	UTCB pointer of the calling thread.

Returns

Syscall return tag.

Definition at line 366 of file vcon.h.

References l4_ipc_call(), L4_IPC_NEVER, l4_msgtag(), L4_PROTO_LOG, L4_VCON_SET_ATTR_OP, and l4_msg_regs_t::mr.

Referenced by l4_vcon_set_attr(), and L4::Vcon::set_attr().

Here is the call graph for this function:

Here is the caller graph for this function:

10.119.3.10 long l4_vcon_write (l4_cap_idx_t *vcon*, char const * *buf*, int *size*) [inline]

Write data to virtual console.

Parameters

vcon	Vcon object.
buf	Pointer to data buffer.
size	Size of buffer in bytes.

Return values

<0	Error.
>=0	Number of bytes written to the virtual console

Definition at line 305 of file vcon.h.

References l4_utcb(), and l4_vcon_write_u().

Here is the call graph for this function:

10.119.3.11 long l4_vcon_write_u (l4_cap_idx_t *vcon*, char const * *buf*, int *size*, l4_utcb_t * *utcb*) [inline]

Write data to `this` virtual console.

Parameters

vcon	Capability index of the vcon object.

buf	Pointer to the data buffer.
size	Size of the data buffer in bytes.
utcb	UTCB pointer of the calling thread.

Return values

<0	Error.
>=0	Number of bytes written to the virtual console.

Definition at line 290 of file vcon.h.

References l4_error(), l4_msgtag_has_error(), l4_vcon_send_u(), and L4_VCON_WRITE_SIZE.

Referenced by l4_vcon_write(), and L4::Vcon::write().

Here is the call graph for this function:

Here is the caller graph for this function:

10.120 Virtual Machines

Virtual Machine API.

Collaboration diagram for Virtual Machines:

Modules

- VM API for SVM

  Virtual machine API for SVM.
- VM API for TZ

  Virtual Machine API for ARM TrustZone.
- VM API for VMX

  Virtual machine API for VMX.

10.120.1 Detailed Description

Virtual Machine API.

10.121 Virtual Registers (UTCBs)

L4 Virtual Registers (UTCB).

Collaboration diagram for Virtual Registers (UTCBs):

Modules

- ARM Virtual Registers (UTCB)
- Buffer Registers (BRs)
- Message Registers (MRs)
- Thread Control Registers (TCRs)
- amd64 Virtual Registers (UTCB)
- x86 Virtual Registers (UTCB)

Files

- file utcb.h

  UTCB definitions for ARM.
- file utcb.h

  UTCB definitions for amd64.
- file utcb.h

  UTCB definitions for X86.

Typedefs

- typedef struct l4_utcb_t l4_utcb_t

  Opaque type for the UTCB.

Functions

- l4_utcb_t * l4_utcb (void) L4_NOTHROW L4_PURE

  Get the UTCB address.
- l4_msg_regs_t * l4_utcb_mr (void) L4_NOTHROW L4_PURE

  Get the message-register block of a UTCB.
- l4_buf_regs_t * l4_utcb_br (void) L4_NOTHROW L4_PURE

  Get the buffer-register block of a UTCB.
- l4_thread_regs_t * l4_utcb_tcr (void) L4_NOTHROW L4_PURE

  Get the thread-control-register block of a UTCB.

10.121.1 Detailed Description

L4 Virtual Registers (UTCB).

Includes:
`#include <l4/sys/utcb.h>`

The virtual registers are part of the micro-kernel API and are located in the user-level thread control block (UTCB). The UTCB is a data structure defined by the micro kernel and located on kernel-provided memory. Each L4 thread gets a unique UTCB assigned when it is bound to a task (see Thread Control , l4_thread_control_bind() for more information).

The UTCB is arranged in three blocks of virtual registers.

- Thread Control Registers (TCRs)
- Message Registers (MRs)
- Buffer Registers (BRs)

To access the contents of the virtual registers the l4_utcb_mr(), l4_utcb_tcr(), and l4_utcb_br() functions must be used.

10.121.2 Typedef Documentation

10.121.2.1 typedef struct l4_utcb_t l4_utcb_t

Opaque type for the UTCB.

To access the contents of the virtual registers the l4_utcb_mr(), l4_utcb_tcr(), and l4_utcb_br() functions must be used.

Definition at line 68 of file utcb.h.

10.121.3 Function Documentation

10.121.3.1 l4_buf_regs_t * l4_utcb_br (void) [inline]

Get the buffer-register block of a UTCB.

Returns

A pointer to the buffer-register block of `u`.

Definition at line 345 of file utcb.h.

References l4_utcb().

Here is the call graph for this function:

10.121.3.2 l4_msg_regs_t * l4_utcb_mr (void) [inline]

Get the message-register block of a UTCB.

Returns

A pointer to the message-register block of `u`.

Examples:

examples/sys/aliens/main.c, examples/sys/ipc/ipc_example.c, examples/sys/singlestep/main.c, and examples/sys/utcb-ipc/main.c.

Definition at line 342 of file utcb.h.

References l4_utcb().

Here is the call graph for this function:

10.121.3.3 l4_thread_regs_t * l4_utcb_tcr (void) [inline]

Get the thread-control-register block of a UTCB.

Returns

A pointer to the thread-control-register block of `u`.

Definition at line 348 of file utcb.h.

References l4_utcb().

Here is the call graph for this function:

10.122 amd64 Virtual Registers (UTCB)

Collaboration diagram for amd64 Virtual Registers (UTCB):

Data Structures

- struct l4_exc_regs_t

  UTCB structure for exceptions.

Typedefs

- typedef struct l4_exc_regs_t l4_exc_regs_t

  UTCB structure for exceptions.

Enumerations

- enum L4_utcb_consts_amd64

  UTCB constants for AMD64.

10.122.1 Detailed Description

10.123 vCPU API

vCPU API

Collaboration diagram for vCPU API:

Data Structures

- struct l4_vcpu_state_t

  State of a vCPU.
- struct l4_vcpu_regs_t

  vCPU registers.
- struct l4_vcpu_ipc_regs_t

  vCPU message registers.

Typedefs

- typedef struct l4_vcpu_state_t l4_vcpu_state_t

  State of a vCPU.
- typedef struct l4_vcpu_regs_t l4_vcpu_regs_t

  vCPU registers.
- typedef struct l4_vcpu_ipc_regs_t l4_vcpu_ipc_regs_t

  vCPU message registers.
- typedef struct l4_vcpu_regs_t l4_vcpu_regs_t

  vCPU registers.
- typedef struct l4_vcpu_ipc_regs_t l4_vcpu_ipc_regs_t

  vCPU message registers.
- typedef struct l4_vcpu_regs_t l4_vcpu_regs_t

  vCPU registers.
- typedef struct l4_vcpu_ipc_regs_t l4_vcpu_ipc_regs_t

  vCPU message registers.

Enumerations

- enum L4_vcpu_state_flags {
  L4_VCPU_F_IRQ = 0x01, L4_VCPU_F_PAGE_FAULTS = 0x02, L4_VCPU_F_EXCEPTIONS = 0x04, L4↩
  _VCPU_F_DEBUG_EXC = 0x08,
  L4_VCPU_F_USER_MODE = 0x20, L4_VCPU_F_FPU_ENABLED = 0x80 }

  State flags of a vCPU.
- enum L4_vcpu_sticky_flags { L4_VCPU_SF_IRQ_PENDING = 0x01 }

  Sticky flags of a vCPU.
- enum L4_vcpu_state_offset { L4_VCPU_OFFSET_EXT_STATE = 0x400, L4_VCPU_OFFSET_EXT_INFOS = 0x200 }

  Offsets for vCPU state layouts.

10.123.1 Detailed Description

vCPU API

10.123.2 Enumeration Type Documentation

10.123.2.1 enum L4_vcpu_state_flags

State flags of a vCPU.

Enumerator

L4_VCPU_F_IRQ IRQs (events) enabled.

L4_VCPU_F_PAGE_FAULTS Page faults enabled.

L4_VCPU_F_EXCEPTIONS Exception enabled.

L4_VCPU_F_DEBUG_EXC Debug exception enabled.

L4_VCPU_F_USER_MODE User task will be used.

L4_VCPU_F_FPU_ENABLED FPU enabled.

Definition at line 56 of file vcpu.h.

10.123.2.2 enum L4_vcpu_state_offset

Offsets for vCPU state layouts.

Enumerator

L4_VCPU_OFFSET_EXT_STATE Offset where extended state begins.

L4_VCPU_OFFSET_EXT_INFOS Offset where extended infos begin.

Definition at line 79 of file vcpu.h.

10.123.2.3 enum L4_vcpu_sticky_flags

Sticky flags of a vCPU.

Enumerator

L4_VCPU_SF_IRQ_PENDING An event (e.g. IRQ) is pending.

Definition at line 70 of file vcpu.h.

10.124 vCPU Support Library

vCPU handling functionality.

Collaboration diagram for vCPU Support Library:

Modules

- Extended vCPU support

  extended vCPU handling functionality.

Data Structures

- class L4vcpu::State

  C++ implementation of state word in the vCPU area.
- class L4vcpu::Vcpu

  C++ implementation of the vCPU save state area.

Functions

- void l4vcpu_irq_disable (l4_vcpu_state_t ∗vcpu) L4_NOTHROW

  Disable a vCPU for event delivery.
- unsigned l4vcpu_irq_disable_save (l4_vcpu_state_t ∗vcpu) L4_NOTHROW

  Disable a vCPU for event delivery and return previous state.
- void l4vcpu_irq_enable (l4_vcpu_state_t ∗vcpu, l4_utcb_t ∗utcb, l4vcpu_event_hndl_t do_event_work_cb, l4vcpu_setup_ipc_t setup_ipc) L4_NOTHROW

  Enable a vCPU for event delivery.
- void l4vcpu_irq_restore (l4_vcpu_state_t ∗vcpu, unsigned s, l4_utcb_t ∗utcb, l4vcpu_event_hndl_t do_event_work_cb, l4vcpu_setup_ipc_t setup_ipc) L4_NOTHROW

  Restore a previously saved IRQ/event state.
- void l4vcpu_wait_for_event (l4_vcpu_state_t ∗vcpu, l4_utcb_t ∗utcb, l4vcpu_event_hndl_t do_event_work_cb, l4vcpu_setup_ipc_t setup_ipc) L4_NOTHROW

  Wait for event.
- void l4vcpu_print_state (const l4_vcpu_state_t ∗vcpu, const char ∗prefix) L4_NOTHROW

  Print the state of a vCPU.
- int l4vcpu_is_irq_entry (l4_vcpu_state_t const ∗vcpu) L4_NOTHROW

  Return whether the entry reason was an IRQ/IPC message.
- int l4vcpu_is_page_fault_entry (l4_vcpu_state_t const ∗vcpu) L4_NOTHROW

  Return whether the entry reason was a page fault.

10.124.1 Detailed Description

vCPU handling functionality.

This library provides convenience functionality on top of the l4sys vCPU interface to ease programming. It wraps commonly used code and abstracts architecture depends parts as far as reasonable.

10.124.2 Function Documentation

10.124.2.1 void l4vcpu_irq_disable (l4_vcpu_state_t * *vcpu*) [inline]

Disable a vCPU for event delivery.

Parameters

vcpu	Pointer to vCPU area.

Definition at line 208 of file vcpu.h.

References l4_barrier(), and L4_VCPU_F_IRQ.

Referenced by L4vcpu::Vcpu::irq_disable(), l4vcpu_irq_disable_save(), and l4vcpu_irq_restore().

Here is the call graph for this function:

Here is the caller graph for this function:

10.124.2.2 unsigned l4vcpu_irq_disable_save (l4_vcpu_state_t * *vcpu*) [inline]

Disable a vCPU for event delivery and return previous state.

Parameters

vcpu	Pointer to vCPU area.

Returns

IRQ state before disabling IRQs.

Definition at line 216 of file vcpu.h.

References l4vcpu_irq_disable().

Referenced by L4vcpu::Vcpu::irq_disable_save().

Here is the call graph for this function:

Here is the caller graph for this function:

10.124.2.3 void l4vcpu_irq_enable (l4_vcpu_state_t * *vcpu*, l4_utcb_t * *utcb*, l4vcpu_event_hndl_t *do_event_work_cb*, l4vcpu_setup_ipc_t *setup_ipc*) [inline]

Enable a vCPU for event delivery.

Parameters

vcpu	Pointer to vCPU area.
utcb	Utcb pointer of the calling vCPU.
do_event_work_cb	Call-back function that is called in case an event (such as an interrupt) is pending.
setup_ipc	Function call-back that is called right before any IPC operation, and before event delivery is enabled.

Definition at line 239 of file vcpu.h.

References l4_barrier(), L4_IPC_BOTH_TIMEOUT_0, L4_LIKELY, L4_VCPU_F_IRQ, and L4_VCPU_SF_IRQ_PENDING.

Referenced by L4vcpu::Vcpu::irq_enable(), and l4vcpu_irq_restore().

Here is the call graph for this function:

Here is the caller graph for this function:

10.124.2.4 void l4vcpu_irq_restore (l4_vcpu_state_t * *vcpu*, unsigned *s*, l4_utcb_t * *utcb*, l4vcpu_event_hndl_t *do_event_work_cb*, l4vcpu_setup_ipc_t *setup_ipc*) [inline]

Restore a previously saved IRQ/event state.

Parameters

vcpu	Pointer to vCPU area.
s	IRQ state to be restored.
utcb	Utcb pointer of the calling vCPU.
do_event_work_cb	Call-back function that is called in case an event (such as an interrupt) is pending after enabling.
setup_ipc	Function call-back that is called right before any IPC operation, and before event delivery is enabled.

Definition at line 264 of file vcpu.h.

References L4_VCPU_F_IRQ, l4vcpu_irq_disable(), and l4vcpu_irq_enable().

Referenced by L4vcpu::Vcpu::irq_restore().

Here is the call graph for this function:

Here is the caller graph for this function:

10.124.2.5 int l4vcpu_is_irq_entry (l4_vcpu_state_t const * *vcpu*) [inline]

Return whether the entry reason was an IRQ/IPC message.

Parameters

vcpu	Pointer to vCPU area.

return 0 if not, !=0 otherwise.

Referenced by L4vcpu::Vcpu::is_irq_entry().

Here is the caller graph for this function:

10.124.2.6 int l4vcpu_is_page_fault_entry (l4_vcpu_state_t const * *vcpu*) [inline]

Return whether the entry reason was a page fault.

Parameters

vcpu	Pointer to vCPU area.

return 0 if not, !=0 otherwise.

Referenced by L4vcpu::Vcpu::is_page_fault_entry().

Here is the caller graph for this function:

10.124.2.7 void l4vcpu_print_state (const l4_vcpu_state_t * *vcpu*, const char * *prefix*)

Print the state of a vCPU.

Parameters

vcpu	Pointer to vCPU area.
prefix	A prefix for each line printed.

Referenced by L4vcpu::Vcpu::print_state().

Here is the caller graph for this function:

10.124.2.8 void l4vcpu_wait_for_event (l4_vcpu_state_t * *vcpu*, l4_utcb_t * *utcb*, l4vcpu_event_hndl_t *do_event_work_cb*, l4vcpu_setup_ipc_t *setup_ipc*) [inline]

Wait for event.

Parameters

vcpu	Pointer to vCPU area.
utcb	Utcb pointer of the calling vCPU.
do_event_work_cb	Call-back function that is called when the vCPU awakes and needs to handle an event/IRQ.
setup_ipc	Function call-back that is called right before any IPC operation.

Note that event delivery remains disabled after this function returns.

Definition at line 277 of file vcpu.h.

References L4_IPC_NEVER.

Referenced by L4vcpu::Vcpu::wait_for_event().

Here is the caller graph for this function:

10.125 x86 Virtual Registers (UTCB)

Collaboration diagram for x86 Virtual Registers (UTCB):

Data Structures

- struct l4_exc_regs_t

  UTCB structure for exceptions.

Typedefs

- typedef struct l4_exc_regs_t l4_exc_regs_t

  UTCB structure for exceptions.

Enumerations

- enum L4_utcb_consts_x86 {
  L4_UTCB_EXCEPTION_REGS_SIZE = 16, L4_UTCB_GENERIC_DATA_SIZE = 63, L4_UTCB_GENERIC_BUFFERS_SIZE = 58, L4_UTCB_MSG_REGS_OFFSET = 0,
  L4_UTCB_BUF_REGS_OFFSET = 64 ∗ sizeof(l4_umword_t), L4_UTCB_THREAD_REGS_OFFSET = 123 ∗ sizeof(l4_umword_t), L4_UTCB_INHERIT_FPU = 1UL << 24, L4_UTCB_OFFSET = 512 }

  UTCB constants for x86.

10.125.1 Detailed Description

10.125.2 Enumeration Type Documentation

10.125.2.1 enum L4_utcb_consts_x86

UTCB constants for x86.

Enumerator

L4_UTCB_EXCEPTION_REGS_SIZE Number if message registers used for exception IPC.

L4_UTCB_GENERIC_DATA_SIZE Total number of message register (MRs) available.

L4_UTCB_GENERIC_BUFFERS_SIZE Total number of buffer registers (BRs) available.

L4_UTCB_MSG_REGS_OFFSET Offset of MR[0] relative to the UTCB pointer.

L4_UTCB_BUF_REGS_OFFSET Offset of BR[0] relative to the UTCB pointer.

L4_UTCB_THREAD_REGS_OFFSET Offset of TCR[0] relative to the UTCB pointer.

L4_UTCB_INHERIT_FPU BDR flag to accept reception of FPU state.

L4_UTCB_OFFSET Offset of two consecutive UTCBs.

Definition at line 41 of file utcb.h.

Chapter 11

Namespace Documentation

11.1 cxx Namespace Reference

Our C++ library.

Namespaces

- Bits

  Internal helpers for the cxx package.

Data Structures

- class Auto_ptr

  Smart pointer with automatic deletion.
- class Avl_map

  AVL tree based associative container.
- class Avl_set

  AVL set for simple compareable items.
- class Avl_tree

  A generic AVL tree.
- class Avl_tree_node

  Node of an AVL tree.
- class Base_slab

  Basic slab allocator.
- class Base_slab_static

  Merged slab allocator (allocators for objects of the same size are merged together).
- class Bitfield

  Definition for a member (part) of a bit field.
- class Bitmap

  A static bit map.
- class Bitmap_base

  Basic bitmap abstraction.
- class H_list

  General double-linked list of unspecified cxx::H_list_item elements.
- class H_list_item_t

  Basic element type for a double-linked H_list.
- struct H_list_t

Double-linked list of typed H_list_item_t elements.

- class List

  Doubly linked list, with internal allocation.

- class List_alloc

  Standard list-based allocator.

- class List_item

  Basic list item.

- struct Lt_functor

  Generic comparator class that defaults to the less-than operator.

- class New_allocator

  Standard allocator based on `operator new ()` *.*

- class Nothrow

  Helper type to distinguish the `oeprator new` *version that does not throw exceptions.*

- struct Pair

  Pair of two values.

- class Pair_first_compare

  Comparison functor for Pair.

- class Ref_ptr

  A reference-counting pointer with automatic cleanup.

- class S_list

  Simple single-linked list.

- class Slab

  Slab allocator for object of type Type.

- class Slab_static

  Merged slab allocator (allocators for objects of the same size are merged together).

- class String

  This class is used to group characters of a string which belong to one syntactical token types number, identifier, string, whitespace or another single character.

- class Weak_ref

  Typed weak reference to an object of type `T`*.*

- class Weak_ref_base

  Generic (base) weak reference to some object.

Functions

- template<typename T1 >
  T1 min (T1 a, T1 b)

  Get the minimum of a and b.

- template<typename T1 >
  T1 max (T1 a, T1 b)

  Get the maximum of a and b.

11.1.1 Detailed Description

Our C++ library.

Small Low-Level C++ Library.

Various kinds of C++ utilities.

11.2 cxx::Bits Namespace Reference

Internal helpers for the cxx package.

Data Structures

- struct Avl_map_get_key

  Key-getter for Avl_map.
- struct Avl_set_get_key

  Internal, key-getter for Avl_set nodes.
- class Base_avl_set

  Internal: AVL set with internally managed nodes.
- class Basic_list

  Internal: Common functions for all head-based list implementations.
- class Bst

  Basic binary search tree (BST).
- class Bst_node

  Basic type of a node in a binary search tree (BST).
- struct Direction

  The direction to go in a binary search tree.

11.2.1 Detailed Description

Internal helpers for the cxx package.

11.3 L4 Namespace Reference

L4 low-level kernel interface.

Namespaces

- Ipc

  IPC related functionality.
- Ipc_svr

  Helper classes for L4::Server instantiation.
- Typeid

  Definition of interface data-type helpers.
- Types

  L4 basic type helpers for C++.

Data Structures

- class Alloc_list

  A simple list-based allocator.
- class Base_exception

  Base class for all exceptions, thrown by the L4Re framework.
- class Basic_registry

  This registry returns the corresponding server object based on the label of an Ipc_gate.
- class Bounds_error

Access out of bounds.

- class Cap

  C++ interface for capabilities.
- class Com_error

  Error conditions during IPC.
- class Debugger

  C++ debugger interface.
- class Element_already_exists

  Exception for duplicate element insertions.
- class Element_not_found

  Exception for a failed lookup (element not found).
- class Exception_tracer

  Back-trace support for exceptions.
- class Factory

  C++ Factory interface to create kernel objects.
- class Icu

  C++ Icu interface.
- class Invalid_capability

  Indicates that an invalid object was invoked.
- class Iommu

  Interface for IO-MMUs used for DMA remapping.
- class IOModifier

  Modifier class for the IO stream.
- class Ipc_gate

  The C++ IPC gate interface.
- class Irq

  C++ Irq interface.
- class Irq_eoi

  Interface for sending an acknowledge message to an object.
- struct Irq_handler_object

  Server object base class for handling IRQ messages.
- struct Irq_mux

  IRQ multiplexer for shared IRQs.
- class Kobject

  Base class for all kinds of kernel objects and remote objects, referenced by capabilities.
- class Kobject_2t

  Helper class to create an L4Re interface class that is derived from two base classes (see `L4::Kobject_t`).
- struct Kobject_3t

  Helper class to create an L4Re interface class that is derived from three base classes (see `L4::Kobject_t`).
- struct Kobject_demand

  Get the combined server-side resource requirements for all type T...
- class Kobject_t

  Helper class to create an L4Re interface class that is derived from a single base class.
- struct Kobject_typeid

  Meta object for handling access to type information of Kobjects.
- struct Kobject_x

  Generic Kobject inheritance template.
- class Out_of_memory

  Exception signalling insufficient memory.
- class Platform_control

  L4 C++ interface for controlling platform-wide properties.

- class Poll_timeout_kipclock

  A polling timeout based on the L4Re clock.
- struct Proto_t

  Data type for defining protocol numbers.
- class Runtime_error

  Exception for an abstract runtime error.
- class Scheduler

  Scheduler object.
- struct Semaphore

  Kernel-provided semaphore object.
- class Server

  Basic server loop for handling client requests.
- class Server_object

  Abstract server object to be used with L4::Server and L4::Basic_registry.
- struct Server_object_t

  Base class (template) for server implementing server objects.
- struct Server_object_x

  Helper class to implement p_dispatch based server objects.
- class Smart_cap

  Smart capability class.
- class String

  A null-terminated string container class.
- class Task

  C++ interface of the Task kernel object.
- class Thread

  C++ L4 kernel thread interface.
- struct Triggerable

  Interface that allows an object to be triggered by some source.
- struct Type_info

  Dynamic Type Information for L4Re Interfaces.
- class Unknown_error

  Exception for an unknown condition.
- class Vcon

  C++ L4 Vcon interface.
- class Vm

  Virtual machine.

Typedefs

- typedef int Opcode

  Data type for RPC opcodes.

Enumerations

- enum { PROTO_ANY = 0, PROTO_EMPTY = -19 }

Functions

- template<typename T >
  Type_info const * kobject_typeid ()

  Get the L4::Type_info for the L4Re interface given in T.
- template<typename T , typename F >
  Cap< T > cap_dynamic_cast (Cap< F > const &c) throw ()

  dynamic_cast for capabilities.
- template<typename T , typename F >
  Cap< T > cap_cast (Cap< F > const &c) throw ()

  static_cast for capabilities.
- template<typename T , typename F >
  Cap< T > cap_reinterpret_cast (Cap< F > const &c) throw ()

  reinterpret_cast for capabilities.
- template<typename T , typename F , typename SMART >
  Smart_cap< T, SMART > cap_cast (Smart_cap< F, SMART > const &c) throw ()

  static_cast for capabilities.
- template<typename T , typename F , typename SMART >
  Smart_cap< T, SMART > cap_reinterpret_cast (Smart_cap< F, SMART > const &c) throw ()

  reinterpret_cast for capabilities.
- void throw_ipc_exception (L4::Cap< void > const &o, l4_msgtag_t const &err, l4_utcb_t *utcb)

  Throw an L4 IPC error as exception.
- void throw_ipc_exception (void const *o, l4_msgtag_t const &err, l4_utcb_t *utcb)

  Throw an L4 IPC error as exception.

Variables

- IOModifier const hex

  Modifies the stream to print numbers as hexadecimal values.
- IOModifier const dec

  Modifies the stream to print numbers as decimal values.
- BasicOStream cout

  Standard output stream.
- BasicOStream cerr

  Standard error stream.

11.3.1 Detailed Description

L4 low-level kernel interface.

11.3.2 Enumeration Type Documentation

11.3.2.1 anonymous enum

Enumerator

PROTO_ANY Default protocol used by Kobject_t and Kobject_x.

PROTO_EMPTY Empty protocol for empty APIs.

Definition at line 55 of file __typeinfo.h.

11.3.3 Function Documentation

11.3.3.1 template<typename T , typename F > Cap<T> L4::cap_cast (Cap< F > const & *c*) throw) [inline]

static_cast for capabilities.

Template Parameters

T	The target type of the capability
F	The source type (and is usually implicitly set)

Parameters

c	The source capability that shall be casted

Returns

A capability typed to the interface `T`.

The use of this cast operator is similar to the `static_cast<>()` for C++ pointers. It does the same type checking and adjustments like C++ does on pointers.

Example code:

```
L4::Cap<L4::Kobject> obj = ... ;
L4::Cap<L4::Icu> icu = L4::cap_cast<L4::Icu>(obj);
```

Definition at line 356 of file capability.h.

Referenced by L4Re::Cap_alloc::alloc(), L4Re::Util::Counting_cap_alloc< COUNTERTYPE >::alloc(), and L4::↩ Ipc_svr::Server_iface::rcv_cap().

Here is the caller graph for this function:

11.3.3.2 template<typename T , typename F > Cap<T> L4::cap_dynamic_cast (Cap< F > const & *c*) throw) [inline]

dynamic_cast for capabilities.

Template Parameters

T	is the target type of the capability
F	is the source type (and is usually implicitly set)

Parameters

c	is the source capability that shall be casted

Returns

A capability typed to the interface `T`. If the object does not support the target interface `T` or does not support the L4::Meta interface the result is the invalid capability selector.

The use of this cast operator is similar to the `dynamic_cast<>()` for C++ pointers. It also induces overhead, because it uses the meta interface (L4::Meta) to do runtime type checking.

Example code:

```
L4::Cap<L4::Kobject> obj = ... ;
L4::Cap<L4::Icu> icu = L4::cap_dynamic_cast<L4::Icu>(obj);
```

Definition at line 110 of file capability.

References cap_reinterpret_cast(), and l4_error().

Here is the call graph for this function:

11.3.3.3 template<typename T , typename F > Cap<T> L4::cap_reinterpret_cast (Cap< F > const & *c*) throw) [inline]

reinterpret_cast for capabilities.

Template Parameters

T	The target type of the capability
F	The source type (and is usually implicitly set)

Parameters

c	The source capability that shall be casted

Returns

A capability typed to the interface `T`.

The use of this cast operator is similar to the `reinterpret_cast<>()` for C++ pointers. It does not do any type checking or type adjustment.

Example code:

```
L4::Cap<L4::Kobject> obj = ... ;
L4::Cap<L4::Icu> icu = L4::cap_reinterpret_cast<L4::Icu>(obj);
```

Definition at line 387 of file capability.h.

Referenced by cap_dynamic_cast().

Here is the caller graph for this function:

11.3.3.4 void L4::throw_ipc_exception (L4::Cap< void > const & *o*, l4_msgtag_t const & *err*, l4_utcb_t * *utcb*) [inline]

Throw an L4 IPC error as exception.

Parameters

o	The client side object, for which the IPC was invoked.
err	The IPC result code (error code).
utcb	UTCB to be used for this operation, usually the UTCB of the calling thread.

Definition at line 44 of file ipc_helper.

References l4_msgtag_t::has_error().

Referenced by throw_ipc_exception().

Here is the call graph for this function:

Here is the caller graph for this function:

11.3.3.5 void L4::throw_ipc_exception (void const * *o*, l4_msgtag_t const & *err*, l4_utcb_t * *utcb*) [inline]

Throw an L4 IPC error as exception.

Parameters

o	The client side object, for which the IPC was invoked.
err	The IPC result code (error code).
utcb	UTCB to be used for this operation, usually the UTCB of the calling thread.

Definition at line 61 of file ipc_helper.

References throw_ipc_exception().

Here is the call graph for this function:

11.4 L4::Ipc Namespace Reference

IPC related functionality.

Namespaces

- Msg

  IPC Message related functionality.

Data Structures

- struct Array

  Array data type for dynamically sized arrays in RPCs.
- struct Array_in_buf

  Server-side copy in buffer for Array.
- struct Array_ref

  Array reference data type for arrays located in the message.
- struct As_value

  Pass the argument as plain data value.
- class Buf_item

  RPC warpper for a receive item.
- struct Call

  RPC attribute for a standard RPC call.
- struct Call_t

  RPC attribute for an RPC call with required rights.
- struct Call_zero_send_timeout

  RPC attribute for an RPC call, with zero send timeout.
- class Cap

  Capability type for RPC interfaces (see `L4::Cap<T>`).
- class Gen_fpage

  Generic RPC wrapper for L4 flex-pages.
- struct In_out

  Mark an argument as in-out argument.
- class Iostream

Input/Output stream for IPC [un]marshalling.

- class Istream

  Input stream for IPC unmarshalling.
- class Msg_ptr

  Pointer to an element of type T in an lpc::Istream.
- struct Opt

  Attribute for defining an optional RPC argument.
- class Ostream

  Output stream for IPC marshalling.
- struct Out

  Mark an argument as a output value in an RPC signature.
- struct Ret_array

  Dynamically sized output array of type T.
- struct Send_only

  RPC attribute for a send-only RPC.
- class Small_buf

  A receive item for receiving a single capability.
- class Snd_item

  RPC wrapper for a send item.
- class Str_cp_in

  Abstraction for extracting a zero-termintaed string from an lpc::Istream.
- class Varg

  Variably sized RPC argument.
- class Varg_list

  Self-contained list of variable-sized RPC parameters.
- class Varg_list_ref

  List of variable-sized RPC parameters as received by the server.

Typedefs

- typedef unsigned short Array_len_default

  Default type for passing length of an array.
- typedef Gen_fpage< Snd_item > Snd_fpage

  Send flex-page.
- typedef Gen_fpage< Buf_item > Rcv_fpage

  Rcv flex-page.

Functions

- template<typename T >
  Cap< T > make_cap (L4::Cap< T > cap, unsigned rights)

  Make an L4::lpc::Cap<T> for the given capability and rights.
- template<typename T >
  Cap< T > make_cap_rw (L4::Cap< T > cap)

  Make an L4::lpc::Cap<T> for the given capability with L4_CAP_FPAGE_RW rights.
- template<typename T >
  Cap< T > make_cap_rws (L4::Cap< T > cap)

  Make an L4::lpc::Cap<T> for the given capability with L4_CAP_FPAGE_RWS rights.
- template<typename T >
  Internal::Buf_cp_out< T > buf_cp_out (T const *v, unsigned long size)

  Insert an array into an lpc::Ostream.

- template<typename T >
  Internal::Buf_cp_in< T > buf_cp_in (T *v, unsigned long &size)
  Extract an array from an Ipc::Istream.
- template<typename T >
  Str_cp_in< T > str_cp_in (T *v, unsigned long &size)
  Create a Str_cp_in for the given values.
- template<typename T >
  Msg_ptr< T > msg_ptr (T *&p)
  Create an Msg_ptr to adjust the given pointer.
- template<typename T >
  Internal::Buf_in< T > buf_in (T *&v, unsigned long &size)
  Return a pointer to stream array data.
- template<typename T >
  T read (Istream &s)
  Read a value out of a stream.

11.4.1 Detailed Description

IPC related functionality.

11.4.2 Function Documentation

11.4.2.1 template<typename T > Internal::Buf_cp_in<T> L4::Ipc::buf_cp_in (T * *v*, unsigned long & *size*)

Extract an array from an Ipc::Istream.

Parameters

	v	Pointer to the array that shall receive the values from the Ipc::Istream.
in,out	*size*	Input: the number of elements the array can take at most Output: the number of elements found in the stream.

buf_cp_in() can be used to extract an array from an Ipc::Istream. This is the counterpart buf_cp_out(). The data from the received message is thereby copied to the given buffer and size is set to the number of elements found in the stream. To avoid the copy operation buf_in() may be used instead.

See also

buf_in() and buf_cp_out().

Definition at line 172 of file ipc_stream.

11.4.2.2 template<typename T > Internal::Buf_cp_out<T> L4::Ipc::buf_cp_out (T const * *v*, unsigned long *size*)

Insert an array into an Ipc::Ostream.

Parameters

v	Pointer to the array that shall be inserted into an Ipc::Ostream.
size	Number of elements in the array.

This function inserts an array (e.g. a string) into an Ipc::Ostream. The data is copied to the stream. On insertion into the Ipc::Ostream exactly the given number of elements of type T are copied to the message buffer, this means the source buffer is no longer referenced after insertion into the stream.

See also

The counterpart is either buf_cp_in() or buf_in().

Definition at line 114 of file ipc_stream.

11.4.2.3 template<typename T > Internal::Buf_in<T> L4::Ipc::buf_in (T *& *v*, unsigned long & *size*)

Return a pointer to stream array data.

Parameters

out	*v*	Pointer to the array within the Ipc::Istream.
out	*size*	The number of elements found in the stream.

This routine provdes a possibility to extract an array from an Ipc::Istream, without extra copy overhead. In contrast to buf_cp_in() the data is not copied to a buffer, but a pointer to the array is returned. The user must make sure the UTCB is not used for other purposes while the returned pointer is still in use.

The mechanism is comparable to that of Msg_ptr, however it handles arrays inserted with buf_cp_out().

See also

buf_cp_in() and buf_cp_out().

Definition at line 321 of file ipc_stream.

11.4.2.4 template<typename T > Cap<T> L4::Ipc::make_cap (L4::Cap< T > *cap*, unsigned *rights*)

Make an L4::Ipc::Cap<T> for the given capability and rights.

Template Parameters

T	(IMPLICIT) type of the referenced interface

Parameters

cap	source capability (L4::Cap<T>)
rights	rights mask that shall be applied on transfer.

Definition at line 608 of file ipc_types.

Referenced by L4virtio::Svr::Driver_mem_region_t< DATA >::Driver_mem_region_t().

Here is the caller graph for this function:

11.4.2.5 template<typename T > Cap<T> L4::Ipc::make_cap_rw (L4::Cap< T > *cap*)

Make an L4::Ipc::Cap<T> for the given capability with L4_CAP_FPAGE_RW rights.

Template Parameters

T	(IMPLICIT) type of the referenced interface

Parameters

cap	source capability (L4::Cap<T>)

Examples:

examples/libs/l4re/c++/mem_alloc/ma+rm.cc, examples/libs/l4re/c++/shared_ds/ds_clnt.cc, and examples/libs/l4re/c++/shared_ds/ds_srv.cc.

Definition at line 618 of file ipc_types.

References L4_CAP_FPAGE_RW.

Referenced by L4Re::Util::Event_buffer_t< PAYLOAD >::attach(), L4virtio::Svr::Dev_config::Dev_config(), L4Re::Util::Event_t< PAYLOAD >::init(), and L4Re::Util::Event_t< PAYLOAD >::init_poll().

Here is the caller graph for this function:

11.4.2.6 template<typename T > Cap<T> L4::Ipc::make_cap_rws (L4::Cap< T > *cap*)

Make an L4::Ipc::Cap<T> for the given capability with L4_CAP_FPAGE_RWS rights.

Template Parameters

T	(IMPLICIT) type of the referenced interface

Parameters

cap	source capability (L4::Cap<T>)

Definition at line 628 of file ipc_types.

References L4_CAP_FPAGE_RWS.

11.4.2.7 template<typename T > Msg_ptr<T> L4::Ipc::msg_ptr (T *& *p*)

Create an Msg_ptr to adjust the given pointer.

This function makes it more convenient to extract pointers to data in the message buffer itself from an Ipc::Istream. This may be used to avoid copy out of large data structures. (See Msg_ptr.)

Definition at line 264 of file ipc_stream.

11.4.2.8 template<typename T > T L4::Ipc::read (Istream & *s*) [inline]

Read a value out of a stream.

Parameters

s	An Istream.

Returns

The value of type `T`.

The stream position is progressed accordingly.

Definition at line 1383 of file ipc_stream.

11.4.2.9 template<typename T > Str_cp_in<T> L4::Ipc::str_cp_in (T * *v*, unsigned long & *size*)

Create a Str_cp_in for the given values.

Parameters

	v	Pointer to the array that shall receive the values from the Ipc::Istream.
in,out	*size*	Input: the number of elements the array can take at most Output: the number of elements found in the stream.

This function makes it more convenient to extract arrays from an Ipc::Istream (

See also

Str_cp_in.)

Definition at line 225 of file ipc_stream.

11.5 L4::Ipc::Msg Namespace Reference

IPC Message related functionality.

Data Structures

- struct CInt_val_ops

  Defines client-side handling of 'MTYPE' as RPC argument.
- struct Cls_buffer

  Marker type for receive buffer values.
- struct Cls_data

  Marker type for data values.
- struct Cls_item

  Marker type for item values.

- struct Dir_in

  Marker type for input values.
- struct Dir_out

  Marker type for output values.
- struct Do_in_data

  Marker for Input data.
- struct Do_in_items

  Marker for Input items.
- struct Do_out_data

  Marker for Output data.
- struct Do_out_items

  Marker for Output items.
- struct Do_rcv_buffers

  Marker for receive buffers.
- struct Elem< Array< A, LEN > & >

  Array as output argument.
- struct Elem< Array< A, LEN > >

  Array as input arguments.
- struct Elem< Array_ref< A, LEN > & >

  Array_ref as output argument.
- struct Is_valid_rpc_type

  Type trait defining a valid RPC parameter type.
- struct Svr_arg_pack

  Server-side RPC arguments data structure used to provide arguments to the server-side implementation of an RPC function.
- struct Svr_val_ops

  Defines server-side handling for `MTYPE` server arguments.

Enumerations

- enum {
  Word_bytes = sizeof(l4_umword_t), Item_words = 2, Item_bytes = Word_bytes ∗ Item_words, Mr_words = L4_UTCB_GENERIC_DATA_SIZE,
  Mr_bytes = Word_bytes ∗ Mr_words, Br_bytes = Word_bytes ∗ L4_UTCB_GENERIC_BUFFERS_SIZE }

Functions

- unsigned long align_to (unsigned long bytes, unsigned long align)

  Pad bytes to the given alignment align (in bytes)
- template<typename T >
  unsigned long align_to (unsigned long bytes)

  Pad bytes to the alignment of the type T.
- template<typename T >
  bool check_size (unsigned offset, unsigned limit)

  Check if there is enough space for T from offset to limit.
- template<typename T , typename CTYPE >
  bool check_size (unsigned offset, unsigned limit, CTYPE cnt)

  Check if there is enough space for an array of T from offset to limit.
- template<typename T >
  int msg_add (char ∗msg, unsigned offs, unsigned limit, T v)

  Add some data to a message at offs.
- template<typename T >
  int msg_get (char ∗msg, unsigned offs, unsigned limit, T &v)

  Get some data from a message at offs.

11.5.1 Detailed Description

IPC Message related functionality.

11.5.2 Enumeration Type Documentation

11.5.2.1 anonymous enum

Enumerator

Word_bytes number of bytes for one message word

Item_words number of message words for one message item

Item_bytes number of bytes for one message item

Mr_words number of message words available in the UTCB

Mr_bytes number of bytes available in the UTCB message registers

Br_bytes number of bytes available in the UTCB buffer registers

Definition at line 96 of file ipc_basics.

11.5.3 Function Documentation

11.5.3.1 unsigned long L4::Ipc::Msg::align_to (unsigned long *bytes*, unsigned long *align*) [inline]

Pad bytes to the given alignment *align* (in bytes)

Parameters

bytes	The input value in bytes
align	The alignment value in bytes

Returns

the result after padding *bytes* to *align*.

Definition at line 41 of file ipc_basics.

Referenced by align_to().

Here is the caller graph for this function:

11.5.3.2 template<typename T > unsigned long L4::Ipc::Msg::align_to (unsigned long *bytes*) [inline]

Pad *bytes* to the alignment of the type *T*.

Template Parameters

T	The data type used for the alignment

Parameters

bytes	The value to add the padding to

Returns

bytes padded to achieve the alignment of *T*.

Definition at line 51 of file ipc_basics.

References align_to().

Here is the call graph for this function:

11.5.3.3 template<typename T > bool L4::Ipc::Msg::check_size (unsigned *offset*, unsigned *limit*) [inline]

Check if there is enough space for T from offset to limit.

Template Parameters

T	The data type that shall be fitted at *offset*

Parameters

offset	The current offset in bytes (must already be padded if desired).
limit	The limit in bytes that must not be exceeded after adding the size of *T*.

Returns

true if the limit will not be exceeded, false else.

Definition at line 64 of file ipc_basics.

11.5.3.4 template<typename T , typename CTYPE > bool L4::Ipc::Msg::check_size (unsigned *offset*, unsigned *limit*, CTYPE *cnt*) [inline]

Check if there is enough space for an array of T from offset to limit.

Template Parameters

T	The data type that shall be fitted at *offset*

CTYPE	Type of the *cnt* parameter

Parameters

offset	The current offset in bytes (must already be padded if desired).
limit	The limit in bytes that must not be exceeded after adding *cnt* times the size of *T*.
cnt	The number of elements of type *T* that shall be put at *offset*.

Returns

true if the limit will not be exceeded, false else.

Definition at line 82 of file ipc_basics.

References L4_UNLIKELY.

11.5.3.5 template<typename T > int L4::Ipc::Msg::msg_add (char * *msg*, unsigned *offs*, unsigned *limit*, T *v*) [inline]

Add some data to a message at offs.

Template Parameters

T	The type of the data to add

Parameters

msg	pointer to the start of the message
offs	The current offset within the message, this shall be padded to the alignment of *T* if *v* is added.
limit	The size limit in bytes that offset must not exceed.
v	The value to add to the message

Returns

The new offset when successful, a negative value if the given limit will be exceeded.

Definition at line 125 of file ipc_basics.

References L4_EMSGTOOLONG, and L4_UNLIKELY.

11.5.3.6 template<typename T > int L4::Ipc::Msg::msg_get (char * *msg*, unsigned *offs*, unsigned *limit*, T & *v*) [inline]

Get some data from a message at offs.

Template Parameters

T	The type of the data to read

Parameters

msg	Pointer to the start of the message
offs	The current offset within the message, this shall be padded to the alignment of *T* if a *v* can be read.
limit	The size limit in bytes that offset must not exceed.
v	A reference to receive the value from the message

Returns

The new offset when successful, a negative value if the given limit will be exceeded.

Definition at line 146 of file ipc_basics.

References L4_EMSGTOOSHORT, and L4_UNLIKELY.

11.6 L4::Ipc_svr Namespace Reference

Helper classes for L4::Server instantiation.

Data Structures

- class Br_manager_no_buffers

  Empty implementation of Server_iface.
- struct Compound_reply

  Mix in for LOOP_HOOKS to always use compound reply and wait.
- struct Default_loop_hooks

  Default LOOP_HOOKS.
- struct Default_setup_wait

  Mix in for LOOP_HOOKS for setup_wait no op.
- struct Default_timeout

  Mix in for LOOP_HOOKS to use a 0 send and a infinite receive timeout.
- struct Direct_dispatch

  Direct disptach helper, for forwarding dispatch calls a registry R.
- struct Direct_dispatch< R * >

  Direct disptach helper, for forwarding dispatch calls via a pointer to a registry R.
- struct Exc_dispatch

  Dispatch helper wrapping try {} catch {} around the dispatch call.
- struct Ignore_errors

  Mix in for LOOP_HOOKS to ignore IPC errors.
- class Server_iface

  Interface for server-loop related functions.
- class Timed_work

  DEPRECATED.
- class Timeout

  Callback interface for Timeout_queue.
- class Timeout_queue

  Timeout queue to be used in l4re server loop.
- class Timeout_queue_hooks

  Loop hooks mixin for integrating a timeout queue into the server loop.

Enumerations

- enum Reply_mode { Reply_compound, Reply_separate }

  Reply mode for server loop.

11.6.1 Detailed Description

Helper classes for L4::Server instantiation.

11.7 L4::Typeid Namespace Reference

Definition of interface data-type helpers.

Data Structures

- struct P_dispatch

  Use for protocol based dispatch stage.
- struct Raw_ipc

  RPCs list for passing raw incoming IPC to the server object.
- struct Rpc_nocode

  List of RPCs of an interface using a single operation without an opcode.
- struct Rpcs

  Standard list of RPCs of an interface.
- struct Rpcs_code

  List of RPCs of an interface using a special opcode type.
- struct Rpcs_sys

  List of RPCs typically used for kernel interfaces.

11.7.1 Detailed Description

Definition of interface data-type helpers.

Note

These type helpers are intended for internal use, if you look for standard C++ type traits use the `<type_traits>` header for the standard C++ library or use `<l4/cxx/type_traits>`.

11.8 L4::Types Namespace Reference

L4 basic type helpers for C++.

Data Structures

- struct Bool

  Boolean meta type.
- struct False

  False meta value.
- class Flags

  Template for defining typical Flags bitmaps.
- struct Same

  Compare two data types for equality.
- struct True

  True meta value.

11.8.1 Detailed Description

L4 basic type helpers for C++.

11.9 L4Re Namespace Reference

L4Re C++ Interfaces.

Namespaces

- Vfs

  Virtual file system for interfaces POSIX libc.

Data Structures

- class Cap_alloc

  Capability allocator interface.
- class Console

  Console class.
- class Dataspace

  Interface for memory-like objects.
- class Debug_obj

  Debug interface.
- class Dma_space

  DMA Address Space.
- class Env

  Initial Environment (C++ version).
- class Event

  Event class.
- class Event_buffer_t

  Event buffer class.
- class Inhibitor

  Set of inhibitor locks, which inhibit specific actions when held.
- class Log

  Log interface class.
- class Mem_alloc

  Memory allocation interface.
- class Namespace

  Name-space interface.
- class Parent

  Parent interface.
- class Rm

  Region map.
- class Smart_cap_auto

  Helper for Auto_cap and Auto_del_cap.

Functions

- long chksys (long err, char const ∗extra="", long ret=0)

  Generate C++ exception on error.
- long chksys (l4_msgtag_t const &t, char const ∗extra="", l4_utcb_t ∗utcb=l4_utcb(), long ret=0)

  Generate C++ exception on error.
- long chksys (l4_msgtag_t const &t, l4_utcb_t ∗utcb, char const ∗extra="")

  Generate C++ exception on error.

- template<typename T >
  T chkcap (T &&cap, char const *extra="", long err=-L4_ENOMEM)

  Check for valid capability or raise C++ exception.

11.9.1 Detailed Description

L4Re C++ Interfaces.

L4 Runtime Environment.

11.9.2 Function Documentation

11.9.2.1 template<typename T > T L4Re::chkcap (T && *cap*, char const * *extra* = " ", long *err* = −L4_ENOMEM) [inline]

Check for valid capability or raise C++ exception.

Template Parameters

T	Type of object to check, must be capability-like (L4::Cap, L4Re::Util::Auto_cap etc.)

Parameters

cap	Capability value to check.
extra	Optional text for exception.
err	Error value for exception or 0 if the capability value should be used.

This function checks whether the capability is valid. If the capability is invalid an C++ exception is generated, using err if err is not zero, otherwise the capability value is used. A valid capability will just be returned.

Definition at line 139 of file error_helper.

References L4_UNLIKELY.

Referenced by L4virtio::Svr::Dev_config::Dev_config().

Here is the caller graph for this function:

11.9.2.2 long L4Re::chksys (long *err*, char const * *extra* = " ", long *ret* = 0) [inline]

Generate C++ exception on error.

Parameters

err	Error value, if negative exception will be thrown
extra	Optional text for exception (default "")
ret	Optional value for exception, default is error value (err)

This function throws an exception if the err is negative and otherwise returns err.

Definition at line 62 of file error_helper.

References L4_UNLIKELY.

Referenced by L4virtio::Svr::Driver_mem_list_t< DATA >::add(), chksys(), L4virtio::Svr::Dev_config::Dev_config(), L4virtio::Svr::Driver_mem_region_t< DATA >::Driver_mem_region_t(), and L4virtio::Svr::Driver_mem_list_t< DATA >::remove().

Here is the caller graph for this function:

11.9.2.3 long L4Re::chksys (l4_msgtag_t const & *t*, char const * *extra* = " ", l4_utcb_t * *utcb* = l4_utcb () , long *ret* = 0) [inline]

Generate C++ exception on error.

Parameters

t	Message tag.
extra	Optional text for exception (default "")
utcb	Option UTCB
ret	Optional value for exception, default is error value (err)

This function throws an exception if the message tag contains an error or the label in the message tag is negative. Otherwise the label in the message tag is returned.

Definition at line 83 of file error_helper.

References l4_msgtag_t::has_error(), L4_UNLIKELY, and l4_msgtag_t::label().

Here is the call graph for this function:

11.9.2.4 long L4Re::chksys (l4_msgtag_t const & *t*, l4_utcb_t * *utcb*, char const * *extra* = " ") [inline]

Generate C++ exception on error.

Parameters

t	Message tag.
utcb	UTCB.
extra	Optional text for exception (default "")

This function throws an exception if the message tag contains an error or the label in the message tag is negative. Otherwise the label in the message tag is returned.

Definition at line 106 of file error_helper.

References chksys().

Here is the call graph for this function:

11.10 L4Re::Vfs Namespace Reference

Virtual file system for interfaces POSIX libc.

Data Structures

- class Be_file

  Boiler plate class for implementing an open file for L4Re::Vfs.
- class Be_file_system

  Boilerplate class for implementing a L4Re::Vfs::File_system.
- class Directory

  Interface for a POSIX file that is a directory.

- class File

  The basic interface for an open POSIX file.
- class File_system

  Basic interface for an L4Re::Vfs file system.
- class Fs

  POSIX File-system related functionality.
- class Generic_file

  The common interface for an open POSIX file.
- class Mman

  Interface for the POSIX memory management.
- class Ops

  Interface for the POSIX backends for an application.
- class Regular_file

  Interface for a POSIX file that provides regular file semantics.
- class Special_file

  Interface for a POSIX file that provides special file semantics.

Functions

- L4Re::Vfs::Ops *vfs_ops asm ("l4re_env_posix_vfs_ops")

  Reference to the applications L4Re::Vfs::Ops singleton.

11.10.1 Detailed Description

Virtual file system for interfaces POSIX libc.

11.11 L4virtio Namespace Reference

L4-VIRTIO Transport C++ API.

Data Structures

- class Ptr

  Pointer used in virtio descriptors.
- class Virtqueue

  Low-level Virtqueue.

11.11.1 Detailed Description

L4-VIRTIO Transport C++ API.

Chapter 12

Data Structure Documentation

12.1 cxx::Auto_ptr< T > Class Template Reference

Smart pointer with automatic deletion.

Collaboration diagram for cxx::Auto_ptr< T >:

Public Types

- typedef T Ref_type

  The referenced type.

Public Member Functions

- Auto_ptr (T *p=0) throw ()

  Construction by assignment of a normal pointer.
- Auto_ptr (Auto_ptr const &o) throw ()

  Copy construction, releases the original pointer.
- Auto_ptr & operator= (Auto_ptr const &o) throw ()

Assignment from another smart pointer.

- ~Auto_ptr () throw ()

  Destruction, shall delete the object.
- T & operator* () const throw ()

  Dereference the pointer.
- T * operator-> () const throw ()

  Member access for the object.
- T * get () const throw ()

  Get the normal pointer.
- T * release () throw ()

  Release the object and get the normal pointer back.
- void reset (T *p=0) throw ()

  Delete the object and reset the smart pointer to NULL.
- operator Priv_type * () const throw ()

  Operator for `if (!ptr) ...`.

12.1.1 Detailed Description

template<typename T>class cxx::Auto_ptr< T >

Smart pointer with automatic deletion.

Template Parameters

T	The type of the referenced object.

This smart pointer calls the delete operator when the destructor is called. This has the effect that the object the pointer points to will be deleted when the pointer goes out of scope, or a new value gets assigned. The smart pointer provides a release() method to get a normal pointer to the object and set the smart pointer to NULL.

Definition at line 36 of file auto_ptr.

12.1.2 Member Typedef Documentation

12.1.2.1 template<typename T > typedef T cxx::Auto_ptr< T >::Ref_type

The referenced type.

Definition at line 41 of file auto_ptr.

12.1.3 Constructor & Destructor Documentation

12.1.3.1 template<typename T > cxx::Auto_ptr< T >::Auto_ptr (T * *p* = 0) throw) [inline], [explicit]

Construction by assignment of a normal pointer.

Parameters

p	The pointer to the object

Definition at line 51 of file auto_ptr.

12.1.3.2 template<typename T > cxx::Auto_ptr< T >::Auto_ptr (Auto_ptr< T > const & *o*) throw) [inline]

Copy construction, releases the original pointer.

Parameters

o	The smart pointer, which shall be copied and released.

Definition at line 57 of file auto_ptr.

12.1.3.3 template<typename T > cxx::Auto_ptr< T >::~Auto_ptr ( ) throw) [inline]

Destruction, shall delete the object.

Definition at line 76 of file auto_ptr.

12.1.4 Member Function Documentation

12.1.4.1 template<typename T > T* cxx::Auto_ptr< T >::get ( ) const throw) [inline]

Get the normal pointer.

Attention

This function will not release the object, the object will be deleted by the smart pointer.

Definition at line 90 of file auto_ptr.

12.1.4.2 template<typename T > cxx::Auto_ptr< T >::operator Priv_type * ( ) const throw) [inline]

Operator for `if (!ptr) ...`.

Definition at line 110 of file auto_ptr.

12.1.4.3 template<typename T > T& cxx::Auto_ptr< T >::operator* ( ) const throw) [inline]

Dereference the pointer.

Definition at line 80 of file auto_ptr.

12.1.4.4 template<typename T > T* cxx::Auto_ptr< T >::operator-> ( ) const throw) [inline]

Member access for the object.

Definition at line 83 of file auto_ptr.

12.1.4.5 template<typename T > Auto_ptr& cxx::Auto_ptr< T >::operator= (Auto_ptr< T > const & *o*) throw) [inline]

Assignment from another smart pointer.

Parameters

o	The source for the assignment (will be released).

Definition at line 65 of file auto_ptr.

References cxx::Auto_ptr< T >::release().

Here is the call graph for this function:

12.1.4.6 template<typename T > T* cxx::Auto_ptr< T >::release ( ) throw) [inline]

Release the object and get the normal pointer back.

After calling this function the smart pointer will point to NULL and the object will not be deleted by the pointer anymore.

Definition at line 98 of file auto_ptr.

Referenced by cxx::Auto_ptr< T >::operator=().

Here is the caller graph for this function:

The documentation for this class was generated from the following file:

- l4/cxx/auto_ptr

12.2 cxx::Avl_map< KEY_TYPE, DATA_TYPE, COMPARE, ALLOC > Class Template Reference

AVL tree based associative container.

Inheritance diagram for cxx::Avl_map< KEY_TYPE, DATA_TYPE, COMPARE, ALLOC >:

Collaboration diagram for cxx::Avl_map< KEY_TYPE, DATA_TYPE, COMPARE, ALLOC >:

Public Types

- typedef COMPARE< KEY_TYPE > Key_compare

  Type of the comparison functor.
- typedef KEY_TYPE Key_type

  Type of the key values.
- typedef DATA_TYPE Data_type

Type of the data values.

- typedef Base_type::Node Node

  Return type for find.

- typedef Base_type::Node_allocator Node_allocator

  Type of the allocator.

Public Member Functions

- Avl_map (Node_allocator const &alloc=Node_allocator())

  Create an empty AVL-tree based map.

- cxx::Pair< Iterator, int > insert (Key_type const &key, Data_type const &data)

  Insert a <key, data> pair into the map.

- Data_type const & operator[] (Key_type const &key) const

  Get the data for the given key.

- Data_type & operator[] (Key_type const &key)

  Get or insert data for the given key.

12.2.1 Detailed Description

template<typename KEY_TYPE, typename DATA_TYPE, template< typename A > class COMPARE = Lt_functor, template< typename B > class ALLOC = New_allocator>class cxx::Avl_map< KEY_TYPE, DATA_TYPE, COMPARE, ALLOC >

AVL tree based associative container.

Template Parameters

KEY_TYPE	Type of the key values.
DATA_TYPE	Type of the data values.
COMPARE	Type comparison functor for the key values.
ALLOC	Type of the allocator used for the nodes.

Definition at line 56 of file avl_map.

12.2.2 Constructor & Destructor Documentation

12.2.2.1 template<typename KEY_TYPE, typename DATA_TYPE, template< typename A > class COMPARE = Lt_functor, template< typename B > class ALLOC = New_allocator> cxx::Avl_map< KEY_TYPE, DATA_TYPE, COMPARE, ALLOC >::Avl_map (Node_allocator const & *alloc* = Node_allocator ()) [inline]

Create an empty AVL-tree based map.

Parameters

alloc	The node allocator.

Definition at line 91 of file avl_map.

12.2.3 Member Function Documentation

12.2.3.1 template<typename KEY_TYPE, typename DATA_TYPE, template< typename A > class COMPARE = Lt_functor, template< typename B > class ALLOC = New_allocator> cxx::Pair<Iterator, int> cxx::Avl_map< KEY_TYPE, DATA_TYPE, COMPARE, ALLOC >::insert (Key_type const & *key*, Data_type const & *data*) [inline]

Insert a <key, data> pair into the map.

Parameters

key	The key value.
data	The data value to insert.

Returns

A pair of iterator (`first`) and return value (`second`). `second` will be 0 if the element was inserted into the set and `-E_exist` if an element is already in the set. In both cases, `first` contains an iterator that points to the element. `second` may also be `-E_nomem` when memory for the new node could not be allocated. `first` is then invalid.

Definition at line 109 of file avl_map.

Referenced by cxx::Avl_map< Region, Hdlr, cxx::Lt_functor, Alloc >::operator[]().

Here is the caller graph for this function:

12.2.3.2 template<typename KEY_TYPE, typename DATA_TYPE, template< typename A > class COMPARE = Lt_functor, template< typename B > class ALLOC = New_allocator> Data_type const& cxx::Avl_map< KEY_TYPE, DATA_TYPE, COMPARE, ALLOC >::operator[] (Key_type const & *key*) const [inline]

Get the data for the given key.

Parameters

key	The key value to use for lookup.

Precondition

A <key, data> pair for the given key value must exist.

Definition at line 117 of file avl_map.

12.2.3.3 template<typename KEY_TYPE, typename DATA_TYPE, template< typename A > class COMPARE = Lt_functor, template< typename B > class ALLOC = New_allocator> Data_type& cxx::Avl_map< KEY_TYPE, DATA_TYPE, COMPARE, ALLOC >::operator[] (Key_type const & *key*) [inline]

Get or insert data for the given key.

Parameters

key	The key value to use for lookup.

Returns

If the item already exists, a reference to the data item. Otherwise a new data item is default-constructed and inserted under the given key before a reference is returned.

Definition at line 129 of file avl_map.

The documentation for this class was generated from the following file:

- l4/cxx/avl_map

12.3 cxx::Avl_set< ITEM_TYPE, COMPARE, ALLOC > Class Template Reference

AVL set for simple compareable items.

Inheritance diagram for cxx::Avl_set< ITEM_TYPE, COMPARE, ALLOC >:

Collaboration diagram for cxx::Avl_set< ITEM_TYPE, COMPARE, ALLOC >:

Additional Inherited Members

12.3.1 Detailed Description

template<typename ITEM_TYPE, class COMPARE = Lt_functor<ITEM_TYPE>, template< typename A > class ALLOC = New↩_allocator>class cxx::Avl_set< ITEM_TYPE, COMPARE, ALLOC >

AVL set for simple compareable items.

The AVL set can store any kind of items where a partial order is defined. The default relation is defined by the '<' operator.

Template Parameters

ITEM_TYPE	The type of the items to be stored in the set.
COMPARE	The relation to define the partial order, default is to use operator '<'.
ALLOC	The allocator to use for the nodes of the AVL set.

Definition at line 419 of file avl_set.

The documentation for this class was generated from the following file:

- l4/cxx/avl_set

12.4 cxx::Avl_tree< Node, Get_key, Compare > Class Template Reference

A generic AVL tree.

Inheritance diagram for cxx::Avl_tree< Node, Get_key, Compare >:

Collaboration diagram for cxx::Avl_tree< Node, Get_key, Compare >:

Public Types

- typedef Bst::Iterator Iterator

  Forward iterator for the tree.
- typedef Bst::Const_iterator Const_iterator

  Constant forward iterator for the tree.
- typedef Bst::Rev_iterator Rev_iterator

Backward iterator for the tree.

- typedef Bst::Const_rev_iterator Const_rev_iterator

  Constant backward iterator for the tree.

Public Member Functions

- Pair< Node *, bool > insert (Node *new_node)

  Insert a new node into this AVL tree.
- Node * remove (Key_param_type key)

  Remove the node with key from the tree.
- Node * erase (Key_param_type key)

  An alias for remove().
- Avl_tree ()

  Create an empty AVL tree.
- ~Avl_tree () noexcept

  Destroy the tree.

Additional Inherited Members

12.4.1 Detailed Description

template<typename Node, typename Get_key, typename Compare = Lt_functor<typename Get_key::Key_type>>class cxx::Avl_tree< Node, Get_key, Compare >

A generic AVL tree.

Template Parameters

Node	The data type of the nodes (must inherit from Avl_tree_node).
Get_key	The meta function to get the key value from a node. The implementation uses `Get_key::key_of(ptr_to_node)`. The type of the key values must be defined in `Get_key::Key_type`.
Compare	Binary relation to establish a total order for the nodes of the tree. `Compare()(l, r)` must return true if the key *l* is smaller than the key *r*.

This implementation does not provide any memory management. It is the responsibility of the caller to allocate nodes before inserting them and to free them when they are removed or when the tree is destroyed. Conversely, the caller must also ensure that nodes are removed from the tree before they are destroyed.

Examples:

tmpfs/lib/src/fs.cc.

Definition at line 109 of file avl_tree.

12.4.2 Member Function Documentation

12.4.2.1 template<typename Node, typename Get_key , class Compare > Pair< Node *, bool > cxx::Avl_tree< Node, Get_key, Compare >::insert (Node * *new_node*)

Insert a new node into this AVL tree.

Parameters

new_node	A pointer to the new node.

Returns

A pair, with *second* set to `true` and *first* pointing to *new_node*, on success. If there is already a node with the same key then *first* points to this node and *second* is 'false'.

Definition at line 229 of file avl_tree.

Referenced by cxx::Bits::Base_avl_set< ITEM_TYPE, COMPARE, ALLOC, GET_KEY >::insert().

Here is the caller graph for this function:

12.4.2.2 template<typename Node , typename Get_key , class Compare > Node * cxx::Avl_tree< Node, Get_key, Compare >::remove (Key_param_type *key*) [inline]

Remove the node with *key* from the tree.

Parameters

key	The key to the node to remove.

Returns

The pointer to the removed node on success, or 0 if no node with the *key* exists.

Definition at line 291 of file avl_tree.

Referenced by cxx::Bits::Base_avl_set< Pair< KEY_TYPE, DATA_TYPE >, COMPARE< KEY_TYPE >, ALLOC, Bits::Avl_map_get_key< KEY_TYPE > >::remove().

Here is the caller graph for this function:

The documentation for this class was generated from the following file:

- l4/cxx/avl_tree

12.5 cxx::Avl_tree_node Class Reference

Node of an AVL tree.

Inheritance diagram for cxx::Avl_tree_node:

Collaboration diagram for cxx::Avl_tree_node:

Protected Member Functions

- Avl_tree_node ()

  Create an uninitialized node, this is what you should do.

Additional Inherited Members

12.5.1 Detailed Description

Node of an AVL tree.

Examples:

tmpfs/lib/src/fs.cc.

Definition at line 38 of file avl_tree.

The documentation for this class was generated from the following file:

- l4/cxx/avl_tree

12.6 cxx::Base_slab< Obj_size, Slab_size, Max_free, Alloc > Class Template Reference

Basic slab allocator.

Inheritance diagram for cxx::Base_slab< Obj_size, Slab_size, Max_free, Alloc >:

Collaboration diagram for cxx::Base_slab< Obj_size, Slab_size, Max_free, Alloc >:

Public Types

- enum { object_size = Obj_size, slab_size = Slab_size, objects_per_slab = (Slab_size - sizeof(Slab_head)) / object_size, max_free_slabs = Max_free }
- typedef Alloc< Slab_i > Slab_alloc

  Type of the allocator for the slab caches.

Public Member Functions

- unsigned total_objects () const throw ()

  Get the total number of objects managed by the slab allocator.
- unsigned free_objects () const throw ()

  Get the total number of objects managed by the slab allocator.

12.6.1 Detailed Description

template< int Obj_size, int Slab_size = L4_PAGESIZE, int Max_free = 2, template< typename A > class Alloc = New_allocator >class cxx::Base_slab< Obj_size, Slab_size, Max_free, Alloc >

Basic slab allocator.

Parameters

Obj_size	The size of the objects managed by the allocator (in bytes).
Slab_size	The size of a slab cache (in bytes).
Max_free	The maximum number of free slab caches. When this limit is reached slab caches are freed.
Alloc	The allocator that is used to allocate the slab caches.

Definition at line 40 of file slab_alloc.

12.6.2 Member Function Documentation

12.6.2.1 template<int Obj_size, int Slab_size = L4_PAGESIZE, int Max_free = 2, template< typename A > class Alloc = New_allocator> unsigned cxx::Base_slab< Obj_size, Slab_size, Max_free, Alloc >::free_objects ( ) const throw) [inline]

Get the total number of objects managed by the slab allocator.

Returns

The number of objects managed by the allocator (including the free objects).

Definition at line 271 of file slab_alloc.

Referenced by cxx::Base_slab_static< sizeof(Type), Slab_size, Max_free, Alloc >::free_objects().

Here is the caller graph for this function:

12.6.2.2 template<int Obj_size, int Slab_size = L4_PAGESIZE, int Max_free = 2, template< typename A > class Alloc = New_allocator> unsigned cxx::Base_slab< Obj_size, Slab_size, Max_free, Alloc >::total_objects ( ) const throw) [inline]

Get the total number of objects managed by the slab allocator.

Returns

The number of objects managed by the allocator (including the free objects).

Definition at line 263 of file slab_alloc.

Referenced by cxx::Base_slab_static< sizeof(Type), Slab_size, Max_free, Alloc >::total_objects().

Here is the caller graph for this function:

The documentation for this class was generated from the following file:

- l4/cxx/slab_alloc

12.7 cxx::Base_slab_static< Obj_size, Slab_size, Max_free, Alloc > Class Template Reference

Merged slab allocator (allocators for objects of the same size are merged together).

Inheritance diagram for cxx::Base_slab_static< Obj_size, Slab_size, Max_free, Alloc >:

Collaboration diagram for cxx::Base_slab_static< Obj_size, Slab_size, Max_free, Alloc >:

Public Types

- enum { object_size = Obj_size, slab_size = Slab_size, objects_per_slab = _A::objects_per_slab, max_free↩ _slabs = Max_free }

Public Member Functions

- void ∗ alloc () throw ()

  Allocate an object.
- void free (void ∗p) throw ()

  Free the given object (p).
- unsigned total_objects () const throw ()

  Get the total number of objects managed by the slab allocator.
- unsigned free_objects () const throw ()

  Get the number of free objects in the slab allocator.

12.7.1 Detailed Description

template<int Obj_size, int Slab_size = L4_PAGESIZE, int Max_free = 2, template< typename A > class Alloc = New_↩ allocator>class cxx::Base_slab_static< Obj_size, Slab_size, Max_free, Alloc >

Merged slab allocator (allocators for objects of the same size are merged together).

Parameters

Obj_size	The size of an object managed by the slab allocator.
Slab_size	The size of a slab cache.
Max_free	The maximum number of free slab caches.
Alloc	The allocator for the slab caches.

This slab allocator class is useful for merging slab allocators with the same parameters (equal *Obj_size*, *Slab_size*, *Max_free*, and *Alloc* parameters) together and share the overhead for the slab caches among all equal-sized objects.

Definition at line 348 of file slab_alloc.

12.7.2 Member Function Documentation

12.7.2.1 template<int Obj_size, int Slab_size = L4_PAGESIZE, int Max_free = 2, template< typename A > class Alloc = New_allocator> void* cxx::Base_slab_static< Obj_size, Slab_size, Max_free, Alloc >::alloc ( ) throw) [inline]

Allocate an object.

Definition at line 365 of file slab_alloc.

12.7.2.2 template<int Obj_size, int Slab_size = L4_PAGESIZE, int Max_free = 2, template< typename A > class Alloc = New_allocator> void cxx::Base_slab_static< Obj_size, Slab_size, Max_free, Alloc >::free (void * *p*) throw) [inline]

Free the given object (*p*).

Parameters

p	The pointer to the object to free.

Precondition

p must be a pointer to an object allocated by this allocator.

Definition at line 371 of file slab_alloc.

12.7.2.3 template<int Obj_size, int Slab_size = L4_PAGESIZE, int Max_free = 2, template< typename A > class Alloc = New_allocator> unsigned cxx::Base_slab_static< Obj_size, Slab_size, Max_free, Alloc >::free_objects ( ) const throw) [inline]

Get the number of free objects in the slab allocator.

Returns

The number of free objects in all free and partially used slab caches managed by this allocator.

Note

The value is the merged value for all equal parameterized Base_slab_static instances.

Definition at line 389 of file slab_alloc.

12.7.2.4 template<int Obj_size, int Slab_size = L4_PAGESIZE, int Max_free = 2, template< typename A > class Alloc = New_allocator> unsigned cxx::Base_slab_static< Obj_size, Slab_size, Max_free, Alloc >::total_objects ( ) const throw) [inline]

Get the total number of objects managed by the slab allocator.

Returns

The number of objects managed by the allocator (including the free objects).

Note

The value is the merged value for all equal parameterized Base_slab_static instances.

Definition at line 380 of file slab_alloc.

The documentation for this class was generated from the following file:

- l4/cxx/slab_alloc

12.8 cxx::Bitfield< T, LSB, MSB > Class Template Reference

Definition for a member (part) of a bit field.

Collaboration diagram for cxx::Bitfield< T, LSB, MSB >:

12.8.1 Detailed Description

template<typename T, unsigned LSB, unsigned MSB>class cxx::Bitfield< T, LSB, MSB >

Definition for a member (part) of a bit field.

Parameters

T	the underlying type of the bit field.
LSB	the least significant bit of our bits.
MSB	the mos significant bit if our bits.

Definition at line 34 of file bitfield.

The documentation for this class was generated from the following file:

- l4/cxx/bitfield

12.9 cxx::Bitmap< BITS > Class Template Reference

A static bit map.

Inheritance diagram for cxx::Bitmap< BITS >:

Collaboration diagram for cxx::Bitmap< BITS >:

Public Member Functions

- Bitmap () throw ()

  Create a bitmap with BITS bits.
- long scan_zero (long start_bit=0) const throw ()

  Scan for the first zero bit.

Additional Inherited Members

12.9.1 Detailed Description

template<int BITS>class cxx::Bitmap< BITS >

A static bit map.

Parameters

BITS	the number of bits that shall be in the bitmap.

Definition at line 180 of file bitmap.

12.9.2 Constructor & Destructor Documentation

12.9.2.1 template<int BITS> cxx::Bitmap< BITS >::Bitmap ( ) throw) [inline]

Create a bitmap with *BITS* bits.

Definition at line 187 of file bitmap.

12.9.3 Member Function Documentation

12.9.3.1 template<int BITS> long cxx::Bitmap< BITS >::scan_zero (long *start_bit* = 0) const throw) [inline]

Scan for the first zero bit.

Parameters

start_bit	Hint at the number of the first bit to look at. Zero bits below `start_bit` may or may not be taken into account by the implementation.

Return values

>=	0 Number of first zero bit found.
-1	All bits at `start_bit` or higher are set.

Compared to Bitmap_base::scan_zero(), the upper bound is set to BITS.

Definition at line 285 of file bitmap.

References cxx::Bitmap_base::scan_zero().

Here is the call graph for this function:

The documentation for this class was generated from the following file:

- l4/cxx/bitmap

12.10 cxx::Bitmap_base Class Reference

Basic bitmap abstraction.

Inheritance diagram for cxx::Bitmap_base:

Collaboration diagram for cxx::Bitmap_base:

Data Structures

- class Bit

  A writeable bit in a bitmap.
- class Char

  Helper abstraction for a byte contained in the bitmap.
- class Word

  Helper abstraction for a word contained in the bitmap.

Public Member Functions

- void bit (long bit, bool on) throw ()

  Set the value of bit bit to on.
- void clear_bit (long bit) throw ()

  Clear bit bit.
- void set_bit (long bit) throw ()

  Set bit bit.
- word_type bit (long bit) const throw ()

  Get the truth value of a bit.
- word_type operator[] (long bit) const throw ()

  Get the bit at index bit.
- Bit operator[] (long bit) throw ()

  Get the lvalue for the bit at index bit.
- long scan_zero (long max_bit, long start_bit=0) const throw ()

  Scan for the first zero bit.

Static Public Member Functions

- static long words (long bits) throw ()

  Get the number of Words that are used for the bitmap.
- static long chars (long bits) throw ()

  Get the number of chars that are used for the bitmap.

Protected Types

- enum { W_bits = sizeof(word_type) ∗ 8, C_bits = 8 }
- typedef unsigned long word_type

  Data type for each element of the bit buffer.

Static Protected Member Functions

- static unsigned word_index (unsigned bit)

  Get the word index for the given bit.
- static unsigned bit_index (unsigned bit)

  Get the bit index within word_type for the given bit.

Protected Attributes

- word_type ∗ _bits

  Pointer to the buffer storing the bits.

12.10.1 Detailed Description

Basic bitmap abstraction.

This abstraction keeps a pointer to a memory area that is used as bitmap.

Definition at line 30 of file bitmap.

12.10.2 Member Function Documentation

12.10.2.1 void cxx::Bitmap_base::bit (long *bit*, bool *on*) throw) [inline]

Set the value of bit *bit* to *on*.

Parameters

bit	the number of the bit
on	the boolean value that shall be assigned to the bit.

Definition at line 211 of file bitmap.

References bit_index(), and word_index().

Referenced by operator[]().

Here is the call graph for this function:

Here is the caller graph for this function:

12.10.2.2 unsigned long cxx::Bitmap_base::bit (long *bit*) const throw) [inline]

Get the truth value of a bit.

Parameters

bit	the number of the bit to read.

Returns

0 if *bit* is not set, != 0 if *bit* is set.

Definition at line 238 of file bitmap.

References bit_index(), and word_index().

Here is the call graph for this function:

12.10.2.3 static unsigned cxx::Bitmap_base::bit_index (unsigned *bit*) [inline], [static], [protected]

Get the bit index within word_type for the given bit.

Parameters

bit	the bit index in the bitmap.

Returns

the bit index within word_type (bit % W_bits).

Definition at line 61 of file bitmap.

References W_bits.

Referenced by bit(), clear_bit(), and set_bit().

Here is the caller graph for this function:

12.10.2.4 static long cxx::Bitmap_base::chars (long *bits*) throw) [inline], [static]

Get the number of chars that are used for the bitmap.

Definition at line 98 of file bitmap.

References C_bits.

12.10.2.5 void cxx::Bitmap_base::clear_bit (long *bit*) throw) [inline]

Clear bit *bit*.

Parameters

bit	the number of the bit to clear.

Definition at line 220 of file bitmap.

References bit_index(), and word_index().

Here is the call graph for this function:

12.10.2.6 word_type cxx::Bitmap_base::operator[] (long *bit*) const throw) [inline]

Get the bit at index *bit*.

Parameters

bit	the number of the bit to read.

Returns

0 if *bit* is not set, != 0 if *bit* is set.

Definition at line 143 of file bitmap.

References bit().

Here is the call graph for this function:

12.10.2.7 Bit cxx::Bitmap_base::operator[] (long *bit*) throw) [inline]

Get the lvalue for the bit at index *bit*.

Parameters

bit	the number.

Returns

lvalue for *bit*

Definition at line 151 of file bitmap.

12.10.2.8 long cxx::Bitmap_base::scan_zero (long *max_bit*, long *start_bit* = 0) const throw) [inline]

Scan for the first zero bit.

Parameters

max_bit	Upper bound (exclusive) for the scanning operation.
start_bit	Hint at the number of the first bit to look at. Zero bits below `start_bit` may or may not be taken into account by the implementation.

Return values

>=	0 Number of first zero bit found.
-1	All bits between `start_bit` and `max_bit` are set.

Definition at line 259 of file bitmap.

References W_bits, and word_index().

Referenced by cxx::Bitmap< BITS >::scan_zero().

Here is the call graph for this function:

Here is the caller graph for this function:

12.10.2.9 void cxx::Bitmap_base::set_bit (long *bit*) throw) [inline]

Set bit *bit*.

Parameters

bit	the number of the bit to set,

Definition at line 229 of file bitmap.

References bit_index(), and word_index().

Here is the call graph for this function:

12.10.2.10 static unsigned cxx::Bitmap_base::word_index (unsigned *bit*) [inline], [static], [protected]

Get the word index for the given bit.

Parameters

bit	the index of the bit in question.

Returns

the index in Bitmap_base::_bits for the given bit (bit / W_bits).

Definition at line 54 of file bitmap.

References W_bits.

Referenced by bit(), clear_bit(), scan_zero(), and set_bit().

Here is the caller graph for this function:

12.10.2.11 static long cxx::Bitmap_base::words (long *bits*) throw) [inline], [static]

Get the number of *Words* that are used for the bitmap.

Definition at line 81 of file bitmap.

References W_bits.

The documentation for this class was generated from the following file:

- l4/cxx/bitmap

12.11 cxx::Bitmap_base::Bit Class Reference

A writeable bit in a bitmap.

Collaboration diagram for cxx::Bitmap_base::Bit:

12.11.1 Detailed Description

A writeable bit in a bitmap.

Definition at line 66 of file bitmap.

The documentation for this class was generated from the following file:

- l4/cxx/bitmap

12.12 cxx::Bitmap_base::Char< BITS > Class Template Reference

Helper abstraction for a byte contained in the bitmap.

Collaboration diagram for cxx::Bitmap_base::Char< BITS >:

12.12.1 Detailed Description

template<long BITS>class cxx::Bitmap_base::Char< BITS >

Helper abstraction for a byte contained in the bitmap.

Definition at line 103 of file bitmap.

The documentation for this class was generated from the following file:

- l4/cxx/bitmap

12.13 cxx::Bitmap_base::Word< BITS > Class Template Reference

Helper abstraction for a word contained in the bitmap.

Inheritance diagram for cxx::Bitmap_base::Word< BITS >:

Collaboration diagram for cxx::Bitmap_base::Word< BITS >:

12.13.1 Detailed Description

template<long BITS>class cxx::Bitmap_base::Word< BITS >

Helper abstraction for a word contained in the bitmap.

Definition at line 87 of file bitmap.

The documentation for this class was generated from the following file:

- l4/cxx/bitmap

12.14 cxx::Bits::Avl_map_get_key< KEY_TYPE > Struct Template Reference

Key-getter for Avl_map.

Collaboration diagram for cxx::Bits::Avl_map_get_key< KEY_TYPE >:

12.14.1 Detailed Description

template<typename KEY_TYPE>struct cxx::Bits::Avl_map_get_key< KEY_TYPE >

Key-getter for Avl_map.

Definition at line 36 of file avl_map.

The documentation for this struct was generated from the following file:

- l4/cxx/avl_map

12.15 cxx::Bits::Avl_set_get_key< KEY_TYPE > Struct Template Reference

Internal, key-getter for Avl_set nodes.

Collaboration diagram for cxx::Bits::Avl_set_get_key< KEY_TYPE >:

12.15.1 Detailed Description

template<typename KEY_TYPE>struct cxx::Bits::Avl_set_get_key< KEY_TYPE >

Internal, key-getter for Avl_set nodes.

Definition at line 93 of file avl_set.

The documentation for this struct was generated from the following file:

- l4/cxx/avl_set

12.16 cxx::Bits::Base_avl_set< ITEM_TYPE, COMPARE, ALLOC, GET_KEY > Class Template Reference

Internal: AVL set with internally managed nodes.

Inheritance diagram for cxx::Bits::Base_avl_set< ITEM_TYPE, COMPARE, ALLOC, GET_KEY >:

Collaboration diagram for cxx::Bits::Base_avl_set< ITEM_TYPE, COMPARE, ALLOC, GET_KEY >:

Data Structures

- class Node

  A smart pointer to a tree item.

Public Types

- typedef ITEM_TYPE Item_type

  Type for the items store in the set.
- typedef GET_KEY Get_key

  Key-getter type to derive the sort key of an internal node.
- typedef GET_KEY::Key_type Key_type

  Type of the sort key used for the items.
- typedef Type_traits< Item_type > ::Const_type Const_item_type

  Type used for const items within the set.
- typedef COMPARE Item_compare

  Type for the comparison functor.
- typedef ALLOC< _Node > Node_allocator

  Type for the node allocator.
- typedef Avl_set_iter< _Node, Item_type, Fwd > Iterator

  Forward iterator for the set.
- typedef Avl_set_iter< _Node, Const_item_type, Fwd > Const_iterator

  Constant forward iterator for the set.
- typedef Avl_set_iter< _Node, Item_type, Rev > Rev_iterator

Backward iterator for the set.

- typedef Avl_set_iter< _Node, Const_item_type, Rev > Const_rev_iterator

  Constant backward iterator for the set.

Public Member Functions

- Base_avl_set (Node_allocator const &alloc=Node_allocator())

  Create a AVL-tree based set.
- Base_avl_set (Base_avl_set const &o)

  Create a copy of an AVL-tree based set.
- cxx::Pair< Iterator, int > insert (Item_type const &item)

  Insert an item into the set.
- int remove (Key_type const &item)

  Remove an item from the set.
- int erase (Key_type const &item)

  Erase the item with the given key.
- Node find_node (Key_type const &item) const

  Lookup a node equal to `item`.
- Node lower_bound_node (Key_type const &key) const

  Find the first node greater or equal to `key`.
- Const_iterator begin () const

  Get the constant forward iterator for the first element in the set.
- Const_iterator end () const

  Get the end marker for the constant forward iterator.
- Iterator begin ()

  Get the mutable forward iterator for the first element of the set.
- Iterator end ()

  Get the end marker for the mutable forward iterator.
- Const_rev_iterator rbegin () const

  Get the constant backward iterator for the last element in the set.
- Const_rev_iterator rend () const

  Get the end marker for the constant backward iterator.
- Rev_iterator rbegin ()

  Get the mutable backward iterator for the last element of the set.
- Rev_iterator rend ()

  Get the end marker for the mutable backward iterator.

12.16.1 Detailed Description

template<typename ITEM_TYPE, class COMPARE, template< typename A > class ALLOC, typename GET_KEY>class cxx::↩ Bits::Base_avl_set< ITEM_TYPE, COMPARE, ALLOC, GET_KEY >

Internal: AVL set with internally managed nodes.

Use Avl_set, Avl_map, or Avl_tree in applications.

Template Parameters

ITEM_TYPE	The type of the items to be stored in the set.
COMPARE	The relation to define the partial order, default is to use operator '<'.
ALLOC	The allocator to use for the nodes of the AVL set.
GET_KEY	Sort-key getter (must provide the `Key_type` and sort-key for an item (of `ITEM_TYPE`).

Definition at line 117 of file avl_set.

12.16.2 Constructor & Destructor Documentation

12.16.2.1 template<typename ITEM_TYPE, class COMPARE, template< typename A > class ALLOC, typename GET_KEY> cxx::Bits::Base_avl_set< ITEM_TYPE, COMPARE, ALLOC, GET_KEY >::Base_avl_set (Node_allocator const & *alloc* = Node_allocator ()) [inline], [explicit]

Create a AVL-tree based set.

Parameters

alloc	Node allocator.

Create an empty set (AVL-tree based).

Definition at line 229 of file avl_set.

12.16.2.2 template<typename Item , class Compare , template< typename A > class Alloc, typename KEY_TYPE > cxx::Bits::Base_avl_set< Item, Compare, Alloc, KEY_TYPE >::Base_avl_set (Base_avl_set< ITEM_TYPE, COMPARE, ALLOC, GET_KEY > const & *o*) [inline]

Create a copy of an AVL-tree based set.

Parameters

o	The set to copy.

Creates a deep copy of the set with all its items.

Definition at line 380 of file avl_set.

References cxx::Bits::Base_avl_set< ITEM_TYPE, COMPARE, ALLOC, GET_KEY >::begin(), cxx::Bits::Base_avl_set< ITEM_TYPE, COMPARE, ALLOC, GET_KEY >::end(), and cxx::Bits::Base_avl_set< ITEM_TYPE, COMPARE, ALLOC, GET_KEY >::insert().

Here is the call graph for this function:

12.16.3 Member Function Documentation

12.16.3.1 template<typename ITEM_TYPE, class COMPARE, template< typename A > class ALLOC, typename GET_KEY> Const_iterator cxx::Bits::Base_avl_set< ITEM_TYPE, COMPARE, ALLOC, GET_KEY >::begin ( ) const [inline]

Get the constant forward iterator for the first element in the set.

Returns

Constant forward iterator for the first element in the set.

Definition at line 330 of file avl_set.

Referenced by cxx::Bits::Base_avl_set< ITEM_TYPE, COMPARE, ALLOC, GET_KEY >::Base_avl_set().

Here is the caller graph for this function:

12.16.3.2 template<typename ITEM_TYPE, class COMPARE, template< typename A > class ALLOC, typename GET_KEY> Iterator cxx::Bits::Base_avl_set< ITEM_TYPE, COMPARE, ALLOC, GET_KEY >::begin ( ) [inline]

Get the mutable forward iterator for the first element of the set.

Returns

The mutable forward iterator for the first element of the set.

Definition at line 341 of file avl_set.

12.16.3.3 template<typename ITEM_TYPE, class COMPARE, template< typename A > class ALLOC, typename GET_KEY> Const_iterator cxx::Bits::Base_avl_set< ITEM_TYPE, COMPARE, ALLOC, GET_KEY >::end ( ) const [inline]

Get the end marker for the constant forward iterator.

Returns

The end marker for the constant forward iterator.

Definition at line 335 of file avl_set.

Referenced by cxx::Bits::Base_avl_set< ITEM_TYPE, COMPARE, ALLOC, GET_KEY >::Base_avl_set().

Here is the caller graph for this function:

12.16.3.4 template<typename ITEM_TYPE, class COMPARE, template< typename A > class ALLOC, typename GET_KEY> Iterator cxx::Bits::Base_avl_set< ITEM_TYPE, COMPARE, ALLOC, GET_KEY >::end ( ) [inline]

Get the end marker for the mutable forward iterator.

Returns

The end marker for mutable forward iterator.

Definition at line 346 of file avl_set.

12.16.3.5 template<typename ITEM_TYPE, class COMPARE, template< typename A > class ALLOC, typename GET_KEY> int cxx::Bits::Base_avl_set< ITEM_TYPE, COMPARE, ALLOC, GET_KEY >::erase (Key_type const & *item*) [inline]

Erase the item with the given key.

Parameters

item	The key of the item to remove.

Definition at line 298 of file avl_set.

12.16.3.6 template<typename ITEM_TYPE, class COMPARE, template< typename A > class ALLOC, typename GET_KEY> Node cxx::Bits::Base_avl_set< ITEM_TYPE, COMPARE, ALLOC, GET_KEY >::find_node (Key_type const & *item*) const [inline]

Lookup a node equal to `item`.

Parameters

item	The value to search for.

Returns

A smart pointer to the element found. If no element was found the smart pointer will be invalid.

Definition at line 309 of file avl_set.

12.16.3.7 template<typename ITEM_TYPE, class COMPARE, template< typename A > class ALLOC, typename GET_KEY> Pair< typename Base_avl_set< Item, Compare, Alloc, KEY_TYPE >::Iterator, int > cxx::Bits::Base_avl_set< Item, Compare, Alloc, KEY_TYPE >::insert (Item_type const & *item*)

Insert an item into the set.

Parameters

item	The item to insert.

Returns

A pair of iterator (`first`) and return value (`second`). `second` will be 0 if the element was inserted into the set and `-E_exist` if an element is already in the set. In both cases, `first` contains an iterator that points to the element. `second` may also be `-E_nomem` in when memory for the node could not be allocated. `first` is then invalid.

Insert a new item into the set, each item can only be once in the set.

Definition at line 390 of file avl_set.

References cxx::Pair< First, Second >::first, cxx::Avl_tree< Node, Get_key, Compare >::insert(), and cxx::Pair< First, Second >::second.

Referenced by cxx::Bits::Base_avl_set< ITEM_TYPE, COMPARE, ALLOC, GET_KEY >::Base_avl_set(), and cxx::Avl_map< Region, Hdlr, cxx::Lt_functor, Alloc >::insert().

Here is the call graph for this function:

Here is the caller graph for this function:

12.16.3.8 template<typename ITEM_TYPE, class COMPARE, template< typename A > class ALLOC, typename GET_KEY> Node cxx::Bits::Base_avl_set< ITEM_TYPE, COMPARE, ALLOC, GET_KEY >::lower_bound_node (Key_type const & *key*) const [inline]

Find the first node greater or equal to `key`.

Parameters

key	Minimum key to look for.

Returns

Smart pointer to the first node greater or equal to `key`. Will be invalid if no such element was found.

Definition at line 320 of file avl_set.

12.16.3.9 template<typename ITEM_TYPE, class COMPARE, template< typename A > class ALLOC, typename GET_KEY> Const_rev_iterator cxx::Bits::Base_avl_set< ITEM_TYPE, COMPARE, ALLOC, GET_KEY >::rbegin ( ) const [inline]

Get the constant backward iterator for the last element in the set.

Returns

The constant backward iterator for the last element in the set.

Definition at line 352 of file avl_set.

12.16.3.10 template<typename ITEM_TYPE, class COMPARE, template< typename A > class ALLOC, typename GET_KEY> Rev_iterator cxx::Bits::Base_avl_set< ITEM_TYPE, COMPARE, ALLOC, GET_KEY >::rbegin ( ) [inline]

Get the mutable backward iterator for the last element of the set.

Returns

The mutable backward iterator for the last element of the set.

Definition at line 363 of file avl_set.

12.16.3.11 template<typename ITEM_TYPE, class COMPARE, template< typename A > class ALLOC, typename GET_KEY> int cxx::Bits::Base_avl_set< ITEM_TYPE, COMPARE, ALLOC, GET_KEY >::remove (Key_type const & *item*) [inline]

Remove an item from the set.

Parameters

item	The item to remove.

Return values

0	Success
-E_noent	Item does not exist
-E_inval	Internal error

Definition at line 278 of file avl_set.

12.16.3.12 template<typename ITEM_TYPE, class COMPARE, template< typename A > class ALLOC, typename GET_KEY> Const_rev_iterator cxx::Bits::Base_avl_set< ITEM_TYPE, COMPARE, ALLOC, GET_KEY >::rend ( ) const [inline]

Get the end marker for the constant backward iterator.

Returns

The end marker for the constant backward iterator.

Definition at line 357 of file avl_set.

12.16.3.13 template<typename ITEM_TYPE, class COMPARE, template< typename A > class ALLOC, typename GET_KEY> Rev_iterator cxx::Bits::Base_avl_set< ITEM_TYPE, COMPARE, ALLOC, GET_KEY >::rend ( ) [inline]

Get the end marker for the mutable backward iterator.

Returns

The end marker for mutable backward iterator.

Definition at line 368 of file avl_set.

The documentation for this class was generated from the following file:

- l4/cxx/avl_set

12.17 cxx::Bits::Base_avl_set< ITEM_TYPE, COMPARE, ALLOC, GET_KEY >::Node Class Reference

A smart pointer to a tree item.

Collaboration diagram for cxx::Bits::Base_avl_set< ITEM_TYPE, COMPARE, ALLOC, GET_KEY >::Node:

Public Member Functions

- Node ()

  Default construction for NIL pointer.
- Node & operator= (Node const &o)

  Default assignment.

- Item_type const & operator∗ ()

  Dereference the pointer.
- Item_type const ∗ operator-> ()

  Dereferenced member access.
- bool valid () const

  Validity check.
- operator Item_type const ∗ ()

  Cast to a real item pointer.

12.17.1 Detailed Description

template<typename ITEM_TYPE, class COMPARE, template< typename A > class ALLOC, typename GET_KEY>class cxx::Bits::Base_avl_set< ITEM_TYPE, COMPARE, ALLOC, GET_KEY >::Node

A smart pointer to a tree item.

Definition at line 155 of file avl_set.

12.17.2 Member Function Documentation

12.17.2.1 template<typename ITEM_TYPE, class COMPARE, template< typename A > class ALLOC, typename GET_KEY> Item_type const& cxx::Bits::Base_avl_set< ITEM_TYPE, COMPARE, ALLOC, GET_KEY >::Node::operator∗ ( ) [inline]

Dereference the pointer.

Precondition

Node is valid.

Definition at line 175 of file avl_set.

12.17.2.2 template<typename ITEM_TYPE, class COMPARE, template< typename A > class ALLOC, typename GET_KEY> Item_type const∗ cxx::Bits::Base_avl_set< ITEM_TYPE, COMPARE, ALLOC, GET_KEY >::Node::operator-> ( ) [inline]

Dereferenced member access.

Precondition

Node is valid.

Definition at line 181 of file avl_set.

12.17.2.3 template<typename ITEM_TYPE, class COMPARE, template< typename A > class ALLOC, typename GET_KEY> bool cxx::Bits::Base_avl_set< ITEM_TYPE, COMPARE, ALLOC, GET_KEY >::Node::valid ( ) const [inline]

Validity check.

Returns

false if the pointer is NIL, true if valid.

Definition at line 187 of file avl_set.

The documentation for this class was generated from the following file:

- l4/cxx/avl_set

12.18 cxx::Bits::Basic_list< POLICY > Class Template Reference

Internal: Common functions for all head-based list implementations.

`#include <list_basics.h>`

Inheritance diagram for cxx::Bits::Basic_list< POLICY >:

Collaboration diagram for cxx::Bits::Basic_list< POLICY >:

Public Member Functions

- bool empty () const

  Check if the list is empty.
- Value_type front () const

  Return the first element in the list.
- void clear ()

  Remove all elements from the list.
- Iterator begin ()

  Return an iterator to the beginning of the list.
- Const_iterator begin () const

  Return a const iterator to the beginning of the list.
- Const_iterator end () const

  Return a const iterator to the end of the list.
- Iterator end ()

  Return an iterator to the end of the list.

Static Public Member Functions

- static Const_iterator iter (Const_value_type c)

  Return a const iterator that begins at the given element.

12.18.1 Detailed Description

template<typename POLICY>class cxx::Bits::Basic_list< POLICY >

Internal: Common functions for all head-based list implementations.

Definition at line 50 of file list_basics.h.

12.18.2 Member Function Documentation

12.18.2.1 template<typename POLICY> void cxx::Bits::Basic_list< POLICY >::clear ( ) [inline]

Remove all elements from the list.

After the operation the state of the elements is undefined.

Definition at line 135 of file list_basics.h.

12.18.2.2 template<typename POLICY> static Const_iterator cxx::Bits::Basic_list< POLICY >::iter (Const_value_type *c*) [inline],[static]

Return a const iterator that begins at the given element.

Parameters

c	Element where the iterator should start.

Precondition

The element c must already be in a list.

Definition at line 148 of file list_basics.h.

The documentation for this class was generated from the following file:

- l4/cxx/bits/list_basics.h

12.19 cxx::Bits::Bst< Node, Get_key, Compare > Class Template Reference

Basic binary search tree (BST).

```
#include <bst.h>
```

Inheritance diagram for cxx::Bits::Bst< Node, Get_key, Compare >:

Collaboration diagram for cxx::Bits::Bst< Node, Get_key, Compare >:

Public Types

- typedef Get_key::Key_type Key_type

  The type of key values used to generate the total order of the elements.

- typedef Type_traits< Key_type >
  ::Param_type Key_param_type

  The type for key parameters.

- typedef Fwd Fwd_iter_ops

  Helper for building forward iterators for different wrapper classes.
- typedef Rev Rev_iter_ops

  Helper for building reverse iterators for different wrapper classes.

Iterators

- typedef __Bst_iter< Node, Node, Fwd > Iterator

  Forward iterator.
- typedef __Bst_iter< Node, Node const, Fwd > Const_iterator

  Constant forward iterator.
- typedef __Bst_iter< Node, Node, Rev > Rev_iterator

  Backward iterator.
- typedef __Bst_iter< Node, Node const, Rev > Const_rev_iterator

  Constant backward.

Public Member Functions

Get default iterators for the ordered tree.

- Const_iterator begin () const

  Get the constant forward iterator for the first element in the set.
- Const_iterator end () const

  Get the end marker for the constant forward iterator.
- Iterator begin ()

  Get the mutable forward iterator for the first element of the set.
- Iterator end ()

  Get the end marker for the mutable forward iterator.
- Const_rev_iterator rbegin () const

  Get the constant backward iterator for the last element in the set.
- Const_rev_iterator rend () const

  Get the end marker for the constant backward iterator.
- Rev_iterator rbegin ()

  Get the mutable backward iterator for the last element of the set.
- Rev_iterator rend ()

  Get the end marker for the mutable backward iterator.

Lookup functions.

- Node * find_node (Key_param_type key) const

  find the node with the given key.
- Node * lower_bound_node (Key_param_type key) const

  find the first node with a key not less than the given key.
- Const_iterator find (Key_param_type key) const

  find the node with the given key.
- template<typename FUNC >
  void remove_all (FUNC &&callback)

  Clear the tree.

Static Protected Member Functions

- template<typename FUNC >
  static void remove_tree (Bst_node *head, FUNC &&callback)

  Remove all elements in the subtree of head.

Interior access for descendants.

As this class is an intended base class we provide protected access to our interior, use 'using' to make this private in concrete implementations.

- Bst_node * _head

  The head pointer of the tree.
- Bst ()

  Create an empty tree.
- Node * head () const

  Access the head node as object of type Node.
- static Key_type k (Bst_node const *n)

  Get the key value of n.
- static Dir dir (Key_param_type l, Key_param_type r)

  Get the direction to go from `l` to search for `r`.
- static Dir dir (Key_param_type l, Bst_node const *r)

  Get the direction to go from `l` to search for `r`.
- static bool greater (Key_param_type l, Key_param_type r)

  Is l greater than r.
- static bool greater (Key_param_type l, Bst_node const *r)

  Is l greater than r.
- static bool greater (Bst_node const *l, Bst_node const *r)

  Is l greater than r.

12.19.1 Detailed Description

template<typename Node, typename Get_key, typename Compare>class cxx::Bits::Bst< Node, Get_key, Compare >

Basic binary search tree (BST).

This class is intended as a base class for concrete binary search trees, such as an AVL tree. This class already provides the basic lookup methods and iterator definitions for a BST.

Definition at line 40 of file bst.h.

12.19.2 Member Function Documentation

12.19.2.1 template<typename Node, typename Get_key, typename Compare> Const_iterator cxx::Bits::Bst< Node, Get_key, Compare >::begin ( ) const [inline]

Get the constant forward iterator for the first element in the set.

Returns

Constant forward iterator for the first element in the set.

Definition at line 179 of file bst.h.

Referenced by cxx::Bits::Base_avl_set< Pair< KEY_TYPE, DATA_TYPE >, COMPARE< KEY_TYPE >, ALLOC, Bits::Avl_map_get_key< KEY_TYPE > >::begin().

Here is the caller graph for this function:

12.19.2.2 template<typename Node, typename Get_key, typename Compare> Iterator cxx::Bits::Bst< Node, Get_key, Compare >::begin ( ) [inline]

Get the mutable forward iterator for the first element of the set.

Returns

The mutable forward iterator for the first element of the set.

Definition at line 190 of file bst.h.

12.19.2.3 template<typename Node, typename Get_key, typename Compare> static Dir cxx::Bits::Bst< Node, Get_key, Compare >::dir (Key_param_type *l*, Key_param_type *r*) [inline], [static], [protected]

Get the direction to go from `l` to search for `r`.

Parameters

l	is the key to look for.
r	is the key at the current position.

Return values

Direction::L	for left
Direction::R	for right
Direction::N	if `l` is equal to `r`.

Definition at line 118 of file bst.h.

Referenced by cxx::Bits::Bst< _Node, GET_KEY, COMPARE >::dir().

Here is the caller graph for this function:

12.19.2.4 template<typename Node, typename Get_key, typename Compare> static Dir cxx::Bits::Bst< Node, Get_key, Compare >::dir (Key_param_type *l*, Bst_node const * *r*) [inline], [static], [protected]

Get the direction to go from `l` to search for `r`.

Parameters

l	is the key to look for.
r	is the node at the current position.

Return values

Direction::L	For left.
Direction::R	For right.
Direction::N	If `l` is equal to `r`.

Definition at line 135 of file bst.h.

12.19.2.5 template<typename Node, typename Get_key, typename Compare> Const_iterator cxx::Bits::Bst< Node, Get_key, Compare >::end ( ) const [inline]

Get the end marker for the constant forward iterator.

Returns

The end marker for the constant forward iterator.

Definition at line 184 of file bst.h.

Referenced by cxx::Bits::Base_avl_set< Pair< KEY_TYPE, DATA_TYPE >, COMPARE< KEY_TYPE >, ALLOC, Bits::Avl_map_get_key< KEY_TYPE > >::end().

Here is the caller graph for this function:

12.19.2.6 template<typename Node, typename Get_key, typename Compare> Iterator cxx::Bits::Bst< Node, Get_key, Compare >::end ( ) [inline]

Get the end marker for the mutable forward iterator.

Returns

The end marker for mutable forward iterator.

Definition at line 195 of file bst.h.

12.19.2.7 template<typename Node , typename Get_key , class Compare > Bst< Node, Get_key, Compare >::Const_iterator cxx::Bits::Bst< Node, Get_key, Compare >::find (Key_param_type *key*) const [inline]

find the node with the given *key*.

Parameters

key	The key value of the element to search.

Returns

A valid iterator for the node with the given *key*, or an invalid iterator if *key* was not found.

Definition at line 312 of file bst.h.

References cxx::Bits::Bst_node::next().

Here is the call graph for this function:

12.19.2.8 template<typename Node , typename Get_key , class Compare > Node * cxx::Bits::Bst< Node, Get_key, Compare >::find_node (Key_param_type *key*) const [inline]

find the node with the given *key*.

Parameters

key	The key value of the element to search.

Returns

A pointer to the node with the given *key*, or NULL if *key* was not found.

Definition at line 276 of file bst.h.

References cxx::Bits::Bst_node::next().

Referenced by cxx::Bits::Base_avl_set< Pair< KEY_TYPE, DATA_TYPE >, COMPARE< KEY_TYPE >, ALLOC, Bits::Avl_map_get_key< KEY_TYPE > >::find_node().

Here is the call graph for this function:

Here is the caller graph for this function:

12.19.2.9 template<typename Node , typename Get_key , class Compare > Node * cxx::Bits::Bst< Node, Get_key, Compare >::lower_bound_node (Key_param_type *key*) const [inline]

find the first node with a key not less than the given *key*.

Parameters

key	The key value of the element to search.

Returns

A pointer to the node with the given *key*, or NULL if *key* was not found.

Definition at line 292 of file bst.h.

References cxx::Bits::Bst_node::next().

Referenced by cxx::Bits::Base_avl_set< Pair< KEY_TYPE, DATA_TYPE >, COMPARE< KEY_TYPE >, ALLOC, Bits::Avl_map_get_key< KEY_TYPE > >::lower_bound_node().

Here is the call graph for this function:

Here is the caller graph for this function:

12.19.2.10 template<typename Node, typename Get_key, typename Compare> Const_rev_iterator cxx::Bits::Bst< Node, Get_key, Compare >::rbegin ( ) const [inline]

Get the constant backward iterator for the last element in the set.

Returns

The constant backward iterator for the last element in the set.

Definition at line 201 of file bst.h.

Referenced by cxx::Bits::Base_avl_set< Pair< KEY_TYPE, DATA_TYPE >, COMPARE< KEY_TYPE >, ALLOC, Bits::Avl_map_get_key< KEY_TYPE > >::rbegin().

Here is the caller graph for this function:

12.19.2.11 template<typename Node, typename Get_key, typename Compare> Rev_iterator cxx::Bits::Bst< Node, Get_key, Compare >::rbegin ( ) [inline]

Get the mutable backward iterator for the last element of the set.

Returns

The mutable backward iterator for the last element of the set.

Definition at line 212 of file bst.h.

12.19.2.12 template<typename Node, typename Get_key, typename Compare> template<typename FUNC > void cxx::Bits::Bst< Node, Get_key, Compare >::remove_all (FUNC && *callback*) [inline]

Clear the tree.

Parameters

callback	Optional function to be called on each removed element.

The callback may delete the elements. The function guarantees that the elements are no longer used after the callback has been called.

Definition at line 258 of file bst.h.

Referenced by cxx::Avl_tree< _Node, GET_KEY, COMPARE >::~Avl_tree().

Here is the caller graph for this function:

12.19.2.13 template<typename Node, typename Get_key, typename Compare> template<typename FUNC > static void cxx::Bits::Bst< Node, Get_key, Compare >::remove_tree (Bst_node * *head*, FUNC && *callback*) [inline], [static], [protected]

Remove all elements in the subtree of head.

Parameters

head	Head of the the subtree to remove
callback	Optional function called on each removed element.

Definition at line 158 of file bst.h.

Referenced by cxx::Bits::Bst< _Node, GET_KEY, COMPARE >::remove_all(), and cxx::Bits::Bst< _Node, GET_KEY, COMPARE >::remove_tree().

Here is the caller graph for this function:

12.19.2.14 template<typename Node, typename Get_key, typename Compare> Const_rev_iterator cxx::Bits::Bst< Node, Get_key, Compare >::rend ( ) const [inline]

Get the end marker for the constant backward iterator.

Returns

The end marker for the constant backward iterator.

Definition at line 206 of file bst.h.

Referenced by cxx::Bits::Base_avl_set< Pair< KEY_TYPE, DATA_TYPE >, COMPARE< KEY_TYPE >, ALLOC, Bits::Avl_map_get_key< KEY_TYPE > >::rend().

Here is the caller graph for this function:

12.19.2.15 template<typename Node, typename Get_key, typename Compare> Rev_iterator cxx::Bits::Bst< Node, Get_key, Compare >::rend ( ) [inline]

Get the end marker for the mutable backward iterator.

Returns

The end marker for mutable backward iterator.

Definition at line 217 of file bst.h.

The documentation for this class was generated from the following file:

- l4/cxx/bits/bst.h

12.20 cxx::Bits::Bst_node Class Reference

Basic type of a node in a binary search tree (BST).

```
#include <bst_base.h>
```

Inheritance diagram for cxx::Bits::Bst_node:

Collaboration diagram for cxx::Bits::Bst_node:

Protected Member Functions

- Bst_node ()

  Create uninitialized node.

- Bst_node (bool)

  Create initialized node.

Static Protected Member Functions

Access to BST linkage.

Provide access to the tree linkage to inherited classes Inherited nodes, such as AVL nodes should make these methods private via 'using'

- static Bst_node * next (Bst_node const *p, Direction d)

  Get next node in direction d.

- static void next (Bst_node *p, Direction d, Bst_node *n)

  Set next node of p in direction d to n.

- static Bst_node ** next_p (Bst_node *p, Direction d)

  Get pointer to link in direction d.

- template<typename Node >
  static Node * next (Bst_node const *p, Direction d)

  Get next node in direction d as type Node.

- static void rotate (Bst_node **t, Direction idir)

  Rotate subtree t in the opposite direction of idir.

12.20.1 Detailed Description

Basic type of a node in a binary search tree (BST).

Definition at line 77 of file bst_base.h.

The documentation for this class was generated from the following file:

- l4/cxx/bits/bst_base.h

12.21 cxx::Bits::Direction Struct Reference

The direction to go in a binary search tree.

```
#include <bst_base.h>
```

Collaboration diagram for cxx::Bits::Direction:

Public Types

- enum Direction_e { L = 0, R = 1, N = 2 }

  The literal direction values.

Public Member Functions

- Direction ()

  Uninitialized direction.
- Direction (Direction_e d)

  Convert a literal direction (L, R, N) to an object.
- Direction (bool b)

  Convert a boolean to a direction (false == L, true == R)
- Direction operator! () const

  Negate the direction.

Comparison operators (equality and inequality)

- bool **operator==** (Direction_e o) const
- bool **operator!=** (Direction_e o) const
- bool **operator==** (Direction o) const
- bool **operator!=** (Direction o) const

12.21.1 Detailed Description

The direction to go in a binary search tree.

Definition at line 39 of file bst_base.h.

12.21.2 Member Enumeration Documentation

12.21.2.1 enum cxx::Bits::Direction::Direction_e

The literal direction values.

Enumerator

L Go to the left child.

R Go to the right child.

N Stop.

Definition at line 42 of file bst_base.h.

12.21.3 Member Function Documentation

12.21.3.1 Direction cxx::Bits::Direction::operator! ( ) const [inline]

Negate the direction.

Note

This is only defined for a current value of L or R

Definition at line 63 of file bst_base.h.

References Direction().

Here is the call graph for this function:

The documentation for this struct was generated from the following file:

- l4/cxx/bits/bst_base.h

12.22 cxx::H_list< T, POLICY > Class Template Reference

General double-linked list of unspecified cxx::H_list_item elements.

Inheritance diagram for cxx::H_list< T, POLICY >:

Collaboration diagram for cxx::H_list< T, POLICY >:

Public Member Functions

- void add (T *e)

  Add element to the front of the list.
- void push_front (T *e)

  Add element to the front of the list.
- T * pop_front ()

  Remove and return the head element of the list.
- Iterator insert (T *e, Iterator const &pred)

  Insert an element at the iterator position.

Static Public Member Functions

- static Iterator iter (T *c)

  Return an iterator for an arbitrary list element.
- static bool in_list (T const *e)

  Check if the given element is currently part of a list.
- static Iterator insert_after (T *e, Iterator const &pred)

  Insert an element after the iterator position.
- static void insert_before (T *e, Iterator const &succ)

  Insert an element before the iterator position.
- static void replace (T *p, T *e)

  Replace an element in a list with a new element.
- static void remove (T *e)

  Remove the given element from its list.
- static Iterator erase (Iterator const &e)

  Remove the element at the given iterator position.

12.22.1 Detailed Description

template<typename T, typename POLICY = Bits::Basic_list_policy< T, H_list_item>>class cxx::H_list< T, POLICY >

General double-linked list of unspecified cxx::H_list_item elements.

Most of the time, you want to use H_list_t.

Definition at line 79 of file hlist.

12.22.2 Member Function Documentation

12.22.2.1 template<typename T, typename POLICY = Bits::Basic_list_policy< T, H_list_item>> static Iterator cxx::H_list< T, POLICY >::erase (Iterator const & *e*) [inline], [static]

Remove the element at the given iterator position.

Parameters

e	Iterator pointing to the element to be removed. Must not point to end().

Returns

New iterator pointing to the element after the removed one.

Note

The hlist implementation guarantees that the original iterator is still valid after the element has been removed. In fact, the iterator returned is the same as the one supplied in the e parameter.

Definition at line 244 of file hlist.

Referenced by L4::Ipc_svr::Timeout_queue::handle_expired_timeouts().

Here is the caller graph for this function:

12.22.2.2 template<typename T, typename POLICY = Bits::Basic_list_policy< T, H_list_item>> Iterator cxx::H_list< T, POLICY >::insert (T * *e*, Iterator const & *pred*) [inline]

Insert an element at the iterator position.

Parameters

e	New Element to be inserted
pred	Iterator pointing to the element after which the element will be inserted. If end() is given, the element will be inserted at the beginning of the queue.

Returns

Iterator pointing to the newly inserted element.

Definition at line 143 of file hlist.

12.22.2.3 template<typename T, typename POLICY = Bits::Basic_list_policy< T, H_list_item>> static Iterator cxx::H_list< T, POLICY >::insert_after (T * *e*, Iterator const & *pred*) [inline], [static]

Insert an element after the iterator position.

Parameters

e	New element to be inserted.
pred	Iterator pointing to the element after which the element will be inserted. Must not be end().

Returns

Iterator pointing to the newly inserted element.

Precondition

The list must not be empty.

Definition at line 170 of file hlist.

12.22.2.4 template<typename T, typename POLICY = Bits::Basic_list_policy< T, H_list_item>> static void cxx::H_list< T, POLICY >::insert_before (T * *e*, Iterator const & *succ*) [inline], [static]

Insert an element before the iterator position.

Parameters

e	New element to be inserted.
succ	Iterator pointing to the element before which the element will be inserted. Must not be end().

Definition at line 190 of file hlist.

Referenced by L4::Ipc_svr::Timeout_queue::add().

Here is the caller graph for this function:

12.22.2.5 template<typename T, typename POLICY = Bits::Basic_list_policy< T, H_list_item>> static Iterator cxx::H_list< T, POLICY >::iter (T * c) [inline], [static]

Return an iterator for an arbitrary list element.

Parameters

c	List element to start the iteration.

Returns

A mutable forward iterator.

Precondition

The element must be in a list.

Definition at line 103 of file hlist.

Referenced by cxx::H_list< Weak_ref_base, Bits::Basic_list_policy< Weak_ref_base, H_list_item_t< Weak_ref_base > > >::insert(), and cxx::H_list< Weak_ref_base, Bits::Basic_list_policy< Weak_ref_base, H_list_item_t< Weak_ref_base > > >::insert_after().

Here is the caller graph for this function:

12.22.2.6 template<typename T, typename POLICY = Bits::Basic_list_policy< T, H_list_item>> T* cxx::H_list< T, POLICY >::pop_front ( ) [inline]

Remove and return the head element of the list.

Precondition

The list must not be empty or the behaviour will be undefined.

Definition at line 126 of file hlist.

12.22.2.7 template<typename T, typename POLICY = Bits::Basic_list_policy< T, H_list_item>> static void cxx::H_list< T, POLICY >::remove (T * *e*) [inline], [static]

Remove the given element from its list.

Parameters

e	Element to be removed. Must be in a list.

Definition at line 228 of file hlist.

Referenced by L4::Ipc_svr::Timeout_queue::remove().

Here is the caller graph for this function:

12.22.2.8 template<typename T, typename POLICY = Bits::Basic_list_policy< T, H_list_item>> static void cxx::H_list< T, POLICY >::replace (T * *p*, T * *e*) [inline], [static]

Replace an element in a list with a new element.

Parameters

p	Element in list to be replaced.
e	Replacement element, must not yet be in a list.

After the operation the p element is no longer in the list and may be reused.

Definition at line 212 of file hlist.

The documentation for this class was generated from the following file:

- l4/cxx/hlist

12.23 cxx::H_list_item_t< ELEM_TYPE > Class Template Reference

Basic element type for a double-linked H_list.

Inheritance diagram for cxx::H_list_item_t< ELEM_TYPE >:

Collaboration diagram for cxx::H_list_item_t< ELEM_TYPE >:

Public Member Functions

- H_list_item_t ()

  Constructor.
- ~H_list_item_t () noexcept

  Destructor.

12.23.1 Detailed Description

template<typename ELEM_TYPE>class cxx::H_list_item_t< ELEM_TYPE >

Basic element type for a double-linked H_list.

Template Parameters

ELEM_TYPE	Base class of the list element.

Definition at line 33 of file hlist.

12.23.2 Constructor & Destructor Documentation

12.23.2.1 template<typename ELEM_TYPE> cxx::H_list_item_t< ELEM_TYPE >::H_list_item_t ( ) [inline]

Constructor.

Creates an element that is not in any list.

Definition at line 41 of file hlist.

12.23.2.2 template<typename ELEM_TYPE> cxx::H_list_item_t< ELEM_TYPE >::~H_list_item_t ( ) [inline], [noexcept]

Destructor.

Automatically removes the element from any list it still might be enchained in.

Definition at line 48 of file hlist.

The documentation for this class was generated from the following file:

- l4/cxx/hlist

12.24 cxx::H_list_t< T > Struct Template Reference

Double-linked list of typed H_list_item_t elements.

Inheritance diagram for cxx::H_list_t< T >:

Collaboration diagram for cxx::H_list_t< T >:

Additional Inherited Members

12.24.1 Detailed Description

template<typename T>struct cxx::H_list_t< T >

Double-linked list of typed H_list_item_t elements.

Note

H_lists are not self-cleaning. Elements that are still chained during destruction are not removed and will therefore be in an undefined state after the destruction.

Definition at line 256 of file hlist.

The documentation for this struct was generated from the following file:

- l4/cxx/hlist

12.25 cxx::List< D, Alloc > Class Template Reference

Doubly linked list, with internal allocation.

Collaboration diagram for cxx::List< D, Alloc >:

Data Structures

- class Iter

  Iterator.

Public Member Functions

- void push_back (D const &d) throw ()

  Add element at the end of the list.
- void push_front (D const &d) throw ()

  Add element at the beginning of the list.
- void remove (Iter const &i) throw ()

  Remove element pointed to by the iterator.
- unsigned long size () const throw ()

  Get the length of the list.
- D const & operator[] (unsigned long idx) const throw ()

  Random access.
- D & operator[] (unsigned long idx) throw ()

Random access.

- Iter items () throw ()

  Get iterator for the list elements.

12.25.1 Detailed Description

template<typename D, template< typename A > class Alloc = New_allocator>class cxx::List< D, Alloc >

Doubly linked list, with internal allocation.

Container for items of type D, implemented by a doubly linked list. Alloc defines the allocator policy.

Definition at line 334 of file list.

12.25.2 Member Function Documentation

12.25.2.1 template<typename D , template< typename A > class Alloc = New_allocator> Iter cxx::List< D, Alloc >::items ( ) throw) [inline]

Get iterator for the list elements.

Definition at line 412 of file list.

12.25.2.2 template<typename D , template< typename A > class Alloc = New_allocator> D const& cxx::List< D, Alloc >::operator[] (unsigned long *idx*) const throw) [inline]

Random access.

Complexity is O(n).

Definition at line 404 of file list.

12.25.2.3 template<typename D , template< typename A > class Alloc = New_allocator> D& cxx::List< D, Alloc >::operator[] (unsigned long *idx*) throw) [inline]

Random access.

Complexity is O(n).

Definition at line 408 of file list.

12.25.2.4 template<typename D , template< typename A > class Alloc = New_allocator> void cxx::List< D, Alloc >::push_back (D const & *d*) throw) [inline]

Add element at the end of the list.

Definition at line 379 of file list.

12.25.2.5 template<typename D , template< typename A > class Alloc = New_allocator> void cxx::List< D, Alloc >::push_front (D const & *d*) throw) [inline]

Add element at the beginning of the list.

Definition at line 388 of file list.

12.25.2.6 template<typename D , template< typename A > class Alloc = New_allocator> void cxx::List< D, Alloc >::remove (Iter const & *i*) throw) [inline]

Remove element pointed to by the iterator.

Definition at line 397 of file list.

12.25.2.7 template<typename D , template< typename A > class Alloc = New_allocator> unsigned long cxx::List< D, Alloc >::size ( ) const throw) [inline]

Get the length of the list.

Definition at line 401 of file list.

The documentation for this class was generated from the following file:

- l4/cxx/list

12.26 cxx::List< D, Alloc >::Iter Class Reference

Iterator.

Collaboration diagram for cxx::List< D, Alloc >::Iter:

Public Member Functions

- operator E * () const throw ()

  operator for testing validity (syntactically equal to pointers)

12.26.1 Detailed Description

template<typename D, template< typename A > class Alloc = New_allocator>class cxx::List< D, Alloc >::Iter

Iterator.

Forward and backward iteratable.

Definition at line 354 of file list.

The documentation for this class was generated from the following file:

- l4/cxx/list

12.27 cxx::List_alloc Class Reference

Standard list-based allocator.

Collaboration diagram for cxx::List_alloc:

Public Member Functions

- List_alloc ()

  Initializes an empty list allocator.
- void free (void ∗block, unsigned long size, bool initial_free=false)

  Return a free memory block to the allocator.
- void ∗ alloc (unsigned long size, unsigned align)

  Alloc a memory block.
- void ∗ alloc_max (unsigned long min, unsigned long ∗max, unsigned align, unsigned granularity)

  Allocate a memory block of `min` *<= size <=* `max`*.*
- unsigned long avail ()

  Get the amount of available memory.

12.27.1 Detailed Description

Standard list-based allocator.

Definition at line 31 of file list_alloc.

12.27.2 Constructor & Destructor Documentation

12.27.2.1 cxx::List_alloc::List_alloc ( ) [inline]

Initializes an empty list allocator.

Note

To initialize the allocator with available memory use the free() function.

Definition at line 56 of file list_alloc.

12.27.3 Member Function Documentation

12.27.3.1 void * cxx::List_alloc::alloc (unsigned long *size*, unsigned *align*) [inline]

Alloc a memory block.

Parameters

size	Size of the memory block
align	Alignment constraint

Returns

Pointer to memory block

Definition at line 354 of file list_alloc.

12.27.3.2 void * cxx::List_alloc::alloc_max (unsigned long *min*, unsigned long * *max*, unsigned *align*, unsigned *granularity*) [inline]

Allocate a memory block of `min` <= size <=`max`.

Parameters

	min	Minimal size to allocate.
in,out	*max*	Maximum size to allocate. The actual allocated size is returned here.
	align	Alignment constraint.
	granularity	Granularity to use for the allocation.

Returns

Pointer to memory block

Definition at line 257 of file list_alloc.

References l4_round_size(), l4_trunc_size(), and cxx::max().

Here is the call graph for this function:

12.27.3.3 unsigned long cxx::List_alloc::avail ( ) [inline]

Get the amount of available memory.

Returns

Available memory in bytes

Definition at line 425 of file list_alloc.

12.27.3.4 void cxx::List_alloc::free (void * *block*, unsigned long *size*, bool *initial_free* = false) [inline]

Return a free memory block to the allocator.

Parameters

block	pointer to memory block
size	size of memory block
initial_free	Set to true for putting fresh memory to the allocator. This will enforce alignment on that memory.

Definition at line 218 of file list_alloc.

The documentation for this class was generated from the following file:

- l4/cxx/list_alloc

12.28 cxx::List_item Class Reference

Basic list item.

Inherited by cxx::T_list_item< T >.

Collaboration diagram for cxx::List_item:

Data Structures

- class Iter

  Iterator for a list of ListItem-s.
- class T_iter

  Iterator for derived classes from ListItem.

Public Member Functions

- List_item ∗ get_prev_item () const throw ()

  Get previous item.
- List_item ∗ get_next_item () const throw ()

  Get next item.
- void insert_prev_item (List_item ∗p) throw ()

  Insert item p before this item.
- void insert_next_item (List_item ∗p) throw ()

  Insert item p after this item.
- void remove_me () throw ()

  Remove this item from the list.

Static Public Member Functions

- template<typename C , typename N >
  static C ∗ push_back (C ∗head, N ∗p) throw ()

  Append item to a list.
- template<typename C , typename N >
  static C ∗ push_front (C ∗head, N ∗p) throw ()

  Prepend item to a list.

- template<typename C , typename N >
  static C * remove (C *head, N *p) throw ()

  Remove item from a list.

12.28.1 Detailed Description

Basic list item.

Basic item that can be member of a doubly linked, cyclic list.

Definition at line 37 of file list.

12.28.2 Member Function Documentation

12.28.2.1 List_item* cxx::List_item::get_next_item ( ) const throw) [inline]

Get next item.

Definition at line 176 of file list.

12.28.2.2 List_item* cxx::List_item::get_prev_item ( ) const throw) [inline]

Get previous item.

Definition at line 173 of file list.

12.28.2.3 void cxx::List_item::insert_next_item (List_item * *p*) throw) [inline]

Insert item p after this item.

Definition at line 189 of file list.

12.28.2.4 void cxx::List_item::insert_prev_item (List_item * *p*) throw) [inline]

Insert item p before this item.

Definition at line 179 of file list.

12.28.2.5 template<typename C , typename N > C * cxx::List_item::push_back (C * *head*, N * *p*) throw) [inline], [static]

Append item to a list.

Convenience function for empty-head corner case.

Parameters

head	Pointer to the current list head.
p	Pointer to new item.

Returns

the pointer to the new head.

Definition at line 248 of file list.

12.28.2.6 template<typename C , typename N > C * cxx::List_item::push_front (C * *head*, N * *p*) throw) [inline], [static]

Prepend item to a list.

Convenience function for empty-head corner case.

Parameters

head	pointer to the current list head.
p	pointer to new item.

Returns

the pointer to the new head.

Definition at line 259 of file list.

12.28.2.7 template<typename C , typename N > C * cxx::List_item::remove (C * *head*, N * *p*) throw) [inline], [static]

Remove item from a list.

Convenience function for remove-head corner case.

Parameters

head	pointer to the current list head.
p	pointer to the item to remove.

Returns

the pointer to the new head.

Definition at line 269 of file list.

12.28.2.8 void cxx::List_item::remove_me ( ) throw) [inline]

Remove this item from the list.

Definition at line 198 of file list.

Referenced by cxx::List_item::Iter::remove_me().

Here is the caller graph for this function:

The documentation for this class was generated from the following file:

- l4/cxx/list

12.29 cxx::List_item::Iter Class Reference

Iterator for a list of ListItem-s.

Inheritance diagram for cxx::List_item::Iter:

Collaboration diagram for cxx::List_item::Iter:

Public Member Functions

- List_item ∗ remove_me () throw ()

  Remove item pointed to by iterator, and return pointer to element.

12.29.1 Detailed Description

Iterator for a list of ListItem-s.

The Iterator iterates till it finds the first element again.

Definition at line 45 of file list.

12.29.2 Member Function Documentation

12.29.2.1 List_item∗ cxx::List_item::Iter::remove_me ( ) throw) [inline]

Remove item pointed to by iterator, and return pointer to element.

Definition at line 86 of file list.

References cxx::List_item::remove_me().

Here is the call graph for this function:

The documentation for this class was generated from the following file:

- l4/cxx/list

12.30 cxx::List_item::T_iter< T, Poly > Class Template Reference

Iterator for derived classes from ListItem.

Inheritance diagram for cxx::List_item::T_iter< T, Poly >:

Collaboration diagram for cxx::List_item::T_iter< T, Poly >:

Additional Inherited Members

12.30.1 Detailed Description

template<typename T, bool Poly = false>class cxx::List_item::T_iter< T, Poly >

Iterator for derived classes from ListItem.

Allows direct access to derived classes by ∗ operator.

Example: class Foo : public ListItem { public: typedef T_iter<Foo> Iter; ... };

Definition at line 119 of file list.

The documentation for this class was generated from the following file:

- l4/cxx/list

12.31 cxx::Lt_functor< Obj > Struct Template Reference

Generic comparator class that defaults to the less-than operator.

Collaboration diagram for cxx::Lt_functor< Obj >:

12.31.1 Detailed Description

template<typename Obj>struct cxx::Lt_functor< Obj >

Generic comparator class that defaults to the less-than operator.

Definition at line 29 of file std_ops.

The documentation for this struct was generated from the following file:

- l4/cxx/std_ops

12.32 cxx::New_allocator< _Type > Class Template Reference

Standard allocator based on `operator new ()` .

Collaboration diagram for cxx::New_allocator< _Type >:

12.32.1 Detailed Description

template<typename _Type>class cxx::New_allocator< _Type >

Standard allocator based on `operator new ()` .

This allocator is the default allocator used for the *cxx Containers*, such as cxx::Avl_set and cxx::Avl_map, to allocate the internal data structures.

Definition at line 60 of file std_alloc.

The documentation for this class was generated from the following file:

- l4/cxx/std_alloc

12.33 cxx::Nothrow Class Reference

Helper type to distinguish the `oeprator new` version that does not throw exceptions.

Collaboration diagram for cxx::Nothrow:

12.33.1 Detailed Description

Helper type to distinguish the `oeprator new` version that does not throw exceptions.

Definition at line 30 of file std_alloc.

The documentation for this class was generated from the following file:

- l4/cxx/std_alloc

12.34 cxx::Pair< First, Second > Struct Template Reference

Pair of two values.

Collaboration diagram for cxx::Pair< First, Second >:

Public Types

- typedef First First_type

  Type of first value.
- typedef Second Second_type

  Type of second value.

Public Member Functions

- template<typename A1 , typename A2 >
  Pair (A1 &&first, A2 &&second)

  Create a pair from the two values.
- Pair ()

  Default construction.

Data Fields

- First first

  First value.
- Second second

  Second value.

12.34.1 Detailed Description

template<typename First, typename Second>struct cxx::Pair< First, Second >

Pair of two values.

Standard container for a pair of values.

Parameters

First	Type of the first value.

Second	Type of the second value.

Definition at line 36 of file pair.

12.34.2 Constructor & Destructor Documentation

12.34.2.1 template<typename First, typename Second> template<typename A1 , typename A2 > cxx::Pair< First, Second >::Pair (A1 && *first*, A2 && *second*) [inline]

Create a pair from the two values.

Parameters

first	The first value.
second	The second value.

Definition at line 54 of file pair.

The documentation for this struct was generated from the following file:

- l4/cxx/pair

12.35 cxx::Pair_first_compare< Cmp, Typ > Class Template Reference

Comparison functor for Pair.

Collaboration diagram for cxx::Pair_first_compare< Cmp, Typ >:

Public Member Functions

- Pair_first_compare (Cmp const &cmp=Cmp())

  Construction.
- bool operator() (Typ const &l, Typ const &r) const

  Do the comaprison based on the first value.

12.35.1 Detailed Description

template<typename Cmp, typename Typ>class cxx::Pair_first_compare< Cmp, Typ >

Comparison functor for Pair.

Parameters

Cmp	Comparison functor for the first value of the pair.
Typ	The pair type.

This functor can be used to compare Pair values with respect to the first value.

Definition at line 75 of file pair.

12.35.2 Constructor & Destructor Documentation

12.35.2.1 template<typename Cmp , typename Typ > cxx::Pair_first_compare< Cmp, Typ >::Pair_first_compare (Cmp const & *cmp* = `Cmp()`) `[inline]`

Construction.

Parameters

cmp	The comparison functor used for the first value.

Definition at line 85 of file pair.

12.35.3 Member Function Documentation

12.35.3.1 template<typename Cmp , typename Typ > bool cxx::Pair_first_compare< Cmp, Typ >::operator() (Typ const & *l*, Typ const & *r*) const `[inline]`

Do the comaprison based on the first value.

Parameters

l	The lefthand value.
r	The righthand value.

Definition at line 92 of file pair.

The documentation for this class was generated from the following file:

- l4/cxx/pair

12.36 cxx::Ref_ptr< T, CNT > Class Template Reference

A reference-counting pointer with automatic cleanup.

Inheritance diagram for cxx::Ref_ptr< T, CNT >:

Collaboration diagram for cxx::Ref_ptr< T, CNT >:

Public Member Functions

- Ref_ptr () throw ()

  Default constructor creates a pointer with no managed object.
- Ref_ptr (Wp const &o) throw ()

  Create a shared pointer from a weak pointer.
- Ref_ptr (T ∗o) throw ()

  Create a shared pointer from a raw pointer.
- Ref_ptr (T ∗o, bool d) throw ()

  Create a shared pointer from a raw pointer without creating a new reference.
- T ∗ get () const throw ()

  Return a raw pointer to the object this shared pointer points to.
- T ∗ ptr () const throw ()

  Return a raw pointer to the object this shared pointer points to.
- T ∗ release () throw ()

  Release the shared pointer without removing the reference.

12.36.1 Detailed Description

template<typename T = void, template< typename X > class CNT = Default_ref_counter>class cxx::Ref_ptr< T, CNT >

A reference-counting pointer with automatic cleanup.

Template Parameters

T	Type of object the pointer points to.

CNT	Type of management class that manages the life time of the object.

This pointer is similar to the standard C++-11 shared_ptr but it does the reference counting directly in the object being pointed to, so that no additional management structures need to be allocated from the heap.

Classes that use this pointer type must implement two functions:

```
int remove_ref()
```

is called when a reference is removed and must return 0 when there are no further references to the object.

```
void add_ref()
```

is called when another ref_ptr to the object is created.

Ref_obj provides a simple implementation of this interface from which classes may inherit.

Examples:

tmpfs/lib/src/fs.cc.

Definition at line 80 of file ref_ptr.

12.36.2 Constructor & Destructor Documentation

12.36.2.1 template<typename T = void, template< typename X > class CNT = Default_ref_counter> cxx::Ref_ptr< T, CNT >::Ref_ptr (Wp const & *o*) throw) [inline]

Create a shared pointer from a weak pointer.

Increases references.

Definition at line 101 of file ref_ptr.

12.36.2.2 template<typename T = void, template< typename X > class CNT = Default_ref_counter> cxx::Ref_ptr< T, CNT >::Ref_ptr (T * *o*) throw) [inline], [explicit]

Create a shared pointer from a raw pointer.

In contrast to C++11 shared_ptr it is safe to use this constructor multiple times and have the same reference counter.

Definition at line 110 of file ref_ptr.

12.36.2.3 template<typename T = void, template< typename X > class CNT = Default_ref_counter> cxx::Ref_ptr< T, CNT >::Ref_ptr (T * *o*, bool *d*) throw) [inline]

Create a shared pointer from a raw pointer without creating a new reference.

Parameters

o	Pointer to the object.
d	Dummy parameter to select this constructor at compile time. The value may be true or false.

This is the counterpart to release().

Definition at line 123 of file ref_ptr.

12.36.3 Member Function Documentation

12.36.3.1 template<typename T = void, template< typename X > class CNT = Default_ref_counter> T* cxx::Ref_ptr< T, CNT >::get ( ) const throw) [inline]

Return a raw pointer to the object this shared pointer points to.

This does not release the pointer or decrease the reference count.

Definition at line 130 of file ref_ptr.

12.36.3.2 template<typename T = void, template< typename X > class CNT = Default_ref_counter> T* cxx::Ref_ptr< T, CNT >::ptr ( ) const throw) [inline]

Return a raw pointer to the object this shared pointer points to.

This does not release the pointer or decrease the reference count.

Definition at line 136 of file ref_ptr.

12.36.3.3 template<typename T = void, template< typename X > class CNT = Default_ref_counter> T* cxx::Ref_ptr< T, CNT >::release ( ) throw) [inline]

Release the shared pointer without removing the reference.

Returns

A raw pointer to the managed object.

Definition at line 147 of file ref_ptr.

The documentation for this class was generated from the following file:

- l4/cxx/ref_ptr

12.37 cxx::S_list< T, POLICY > Class Template Reference

Simple single-linked list.

Inheritance diagram for cxx::S_list< T, POLICY >:

Collaboration diagram for cxx::S_list< T, POLICY >:

Public Member Functions

- void add (T *e)

  Add an element to the front of the list.
- void push_front (T *e)

  Add an element to the front of the list.
- T * pop_front ()

  Remove and return the head element of the list.

Additional Inherited Members

12.37.1 Detailed Description

template<typename T, typename POLICY = Bits::Basic_list_policy< T, S_list_item >>class cxx::S_list< T, POLICY >

Simple single-linked list.

Template Parameters

T	Type of elements saved in the list. Must inherit from cxx::S_list_item

Definition at line 50 of file slist.

12.37.2 Member Function Documentation

12.37.2.1 template<typename T, typename POLICY = Bits::Basic_list_policy< T, S_list_item >> T* cxx::S_list< T, POLICY >::pop_front ( ) [inline]

Remove and return the head element of the list.

Precondition

The list must not be empty or the behaviour will be undefined.

Definition at line 91 of file slist.

The documentation for this class was generated from the following file:

- l4/cxx/slist

12.38 cxx::Slab< Type, Slab_size, Max_free, Alloc > Class Template Reference

Slab allocator for object of type *Type*.

Inheritance diagram for cxx::Slab< Type, Slab_size, Max_free, Alloc >:

Collaboration diagram for cxx::Slab< Type, Slab_size, Max_free, Alloc >:

Public Member Functions

- Type ∗ alloc () throw ()

  Allocate an object of type Type.
- void free (Type ∗o) throw ()

  Free the object addressed by o.

Additional Inherited Members

12.38.1 Detailed Description

template<typename Type, int Slab_size = L4_PAGESIZE, int Max_free = 2, template< typename A > class Alloc = New_allocator>class cxx::Slab< Type, Slab_size, Max_free, Alloc >

Slab allocator for object of type *Type*.

Parameters

Type	the type of the objects to manage.
Slab_size	size of a slab cache.
Max_free	the maximum number of free slab caches.
Alloc	the allocator for the slab caches.

Definition at line 297 of file slab_alloc.

12.38.2 Member Function Documentation

12.38.2.1 template<typename Type , int Slab_size = L4_PAGESIZE, int Max_free = 2, template< typename A > class Alloc = New_allocator> Type* cxx::Slab< Type, Slab_size, Max_free, Alloc >::alloc ( ) throw) [inline]

Allocate an object of type *Type*.

Returns

A pointer to the object just allocated, or 0 on failure.

Definition at line 314 of file slab_alloc.

12.38.2.2 template<typename Type , int Slab_size = L4_PAGESIZE, int Max_free = 2, template< typename A > class Alloc = New_allocator> void cxx::Slab< Type, Slab_size, Max_free, Alloc >::free (Type * *o*) throw) [inline]

Free the object addressed by *o*.

Parameters

o	The pointer to the object to free.

Precondition

The object must have been allocated with this allocator.

Definition at line 325 of file slab_alloc.

The documentation for this class was generated from the following file:

- l4/cxx/slab_alloc

12.39 cxx::Slab_static< Type, Slab_size, Max_free, Alloc > Class Template Reference

Merged slab allocator (allocators for objects of the same size are merged together).

Inheritance diagram for cxx::Slab_static< Type, Slab_size, Max_free, Alloc >:

Collaboration diagram for cxx::Slab_static< Type, Slab_size, Max_free, Alloc >:

Public Member Functions

- Type * alloc () throw ()

  Allocate an object of type Type.

12.39.1 Detailed Description

template<typename Type, int Slab_size = L4_PAGESIZE, int Max_free = 2, template< typename A > class Alloc = New_allocator>class cxx::Slab_static< Type, Slab_size, Max_free, Alloc >

Merged slab allocator (allocators for objects of the same size are merged together).

Parameters

Type	The type of the objects to manage.
Slab_size	The size of a slab cache.
Max_free	The maximum number of free slab caches.
Alloc	The allocator for the slab caches.

This slab allocator class is useful for merging slab allocators with the same parameters (equal *sizeof(Type)*, *Slab↩_size*, *Max_free*, and *Alloc* parameters) together and share the overhead for the slab caches among all equal-sized objects.

Definition at line 415 of file slab_alloc.

12.39.2 Member Function Documentation

12.39.2.1 template<typename Type , int Slab_size = L4_PAGESIZE, int Max_free = 2, template< typename A > class Alloc = New_allocator> Type* cxx::Slab_static< Type, Slab_size, Max_free, Alloc >::alloc ( ) throw) [inline]

Allocate an object of type *Type*.

Returns

A pointer to the just allocated object, or 0 of failure.

Definition at line 425 of file slab_alloc.

The documentation for this class was generated from the following file:

- l4/cxx/slab_alloc

12.40 cxx::String Class Reference

This class is used to group characters of a string which belong to one syntactical token types number, identifier, string, whitespace or another single character.

Inheritance diagram for cxx::String:

Collaboration diagram for cxx::String:

12.40.1 Detailed Description

This class is used to group characters of a string which belong to one syntactical token types number, identifier, string, whitespace or another single character.

Examples:

tmpfs/lib/src/fs.cc.

Definition at line 33 of file string.

The documentation for this class was generated from the following file:

- l4/cxx/string

12.41 cxx::Weak_ref< T > Class Template Reference

Typed weak reference to an object of type `T`.

Inheritance diagram for cxx::Weak_ref< T >:

Collaboration diagram for cxx::Weak_ref< T >:

Additional Inherited Members

12.41.1 Detailed Description

template<typename T>class cxx::Weak_ref< T >

Typed weak reference to an object of type `T`.

Template Parameters

T	The type of the referenced object.

A weak reference is a reference that is invalidated when the referenced object is about to be deleted. All weak references to an object are kept in a linked list and all the weak references are iterated and reset by the Weak_↩ ref_base::List destructor or Weak_ref_base::reset().

The type `T` must provide two methods that handle the housekeeping of weak references: remove_weak_ref(Weak↩ _ref_base ∗) and add_weak_ref(Weak_ref_base ∗). These functions must handle the insertion and removal of the weak reference into the respective Weak_ref_base::List object. For convenience one can use the cxx::Weak_ref_obj as a base class that handles weak references for you.

Definition at line 67 of file weak_ref.

The documentation for this class was generated from the following file:

- l4/cxx/weak_ref

12.42 cxx::Weak_ref_base Class Reference

Generic (base) weak reference to some object.

Inheritance diagram for cxx::Weak_ref_base:

Collaboration diagram for cxx::Weak_ref_base:

Additional Inherited Members

12.42.1 Detailed Description

Generic (base) weak reference to some object.

A weak reference is a reference that gets reset to NULL when the object shall be deleted. All weak references to the same object are kept in a linked list of weak references.

For typed weak references see `cxx::Weak_ref`.

Definition at line 25 of file weak_ref.

The documentation for this class was generated from the following file:

- l4/cxx/weak_ref

12.43 Elf32_Dyn Struct Reference

ELF32 dynamic entry.

```
#include <elf.h>
```

Collaboration diagram for Elf32_Dyn:

Data Fields

- Elf32_Sword d_tag

  see DT_ values
- Elf32_Word d_val

  integer values with various interpret.
- Elf32_Addr d_ptr

  program virtual addresses

12.43.1 Detailed Description

ELF32 dynamic entry.

Definition at line 460 of file elf.h.

12.43.2 Field Documentation

12.43.2.1 Elf32_Word Elf32_Dyn::d_val

integer values with various interpret.

Definition at line 463 of file elf.h.

The documentation for this struct was generated from the following file:

- l4/util/elf.h

12.44 Elf32_Ehdr Struct Reference

ELF32 header.

```
#include <elf.h>
```

Collaboration diagram for Elf32_Ehdr:

Data Fields

- Elf32_Half e_type

  type of ELF file
- Elf32_Half e_machine

  required architecture
- Elf32_Word e_version

  file version
- Elf32_Addr e_entry

  initial eip
- Elf32_Off e_phoff

  offset of program header table
- Elf32_Off e_shoff

  offset of file header table
- Elf32_Word e_flags

  processor-specific flags
- Elf32_Half e_ehsize

  size of ELF header
- Elf32_Half e_phentsize

  size of program header entry
- Elf32_Half e_phnum

  of entries in prog.
- Elf32_Half e_shentsize

  size of section header entry
- Elf32_Half e_shnum

of entries in sect.

- Elf32_Half e_shstrndx

  sect.head.tab.idx of strtab

12.44.1 Detailed Description

ELF32 header.

Definition at line 118 of file elf.h.

12.44.2 Field Documentation

12.44.2.1 Elf32_Half Elf32_Ehdr::e_phnum

of entries in prog.

head. tab.

Definition at line 129 of file elf.h.

12.44.2.2 Elf32_Half Elf32_Ehdr::e_shnum

of entries in sect.

head. tab.

Definition at line 131 of file elf.h.

The documentation for this struct was generated from the following file:

- l4/util/elf.h

12.45 Elf32_Phdr Struct Reference

ELF32 program header.

```
#include <elf.h>
```

Collaboration diagram for Elf32_Phdr:

Data Fields

- Elf32_Word p_type

  type of program section
- Elf32_Off p_offset

  file offset of program section
- Elf32_Addr p_vaddr

  memory address of prog section
- Elf32_Addr p_paddr

  physical address (ignored)
- Elf32_Word p_filesz

  file size of program section
- Elf32_Word p_memsz

  memory size of program section
- Elf32_Word p_flags

  flags
- Elf32_Word p_align

  alignment of section

12.45.1 Detailed Description

ELF32 program header.

Definition at line 379 of file elf.h.

The documentation for this struct was generated from the following file:

- l4/util/elf.h

12.46 Elf32_Shdr Struct Reference

ELF32 section header - figure 1-9, page 1-9.

```
#include <elf.h>
```

Collaboration diagram for Elf32_Shdr:

Data Fields

- Elf32_Word sh_name

  name of sect (idx into strtab)
- Elf32_Word sh_type

  section's type
- Elf32_Word sh_flags

  section's flags
- Elf32_Addr sh_addr

  memory address of section
- Elf32_Off sh_offset

  file offset of section
- Elf32_Word sh_size

  file size of section
- Elf32_Word sh_link

  idx to associated header section
- Elf32_Word sh_info

  extra info of header section
- Elf32_Word sh_addralign

  address alignment constraints
- Elf32_Word sh_entsize

  size of entry if sect is table

12.46.1 Detailed Description

ELF32 section header - figure 1-9, page 1-9.

Definition at line 302 of file elf.h.

The documentation for this struct was generated from the following file:

- l4/util/elf.h

12.47 Elf32_Sym Struct Reference

ELF32 symbol table entry.

```
#include <elf.h>
```

Collaboration diagram for Elf32_Sym:

Data Fields

- Elf32_Word st_name

  name of symbol (idx symstrtab)
- Elf32_Addr st_value

  value of associated symbol
- Elf32_Word st_size

  size of associated symbol
- unsigned char st_info

  type and binding info
- unsigned char st_other

  undefined
- Elf32_Half st_shndx

  associated section header

12.47.1 Detailed Description

ELF32 symbol table entry.

Definition at line 761 of file elf.h.

The documentation for this struct was generated from the following file:

- l4/util/elf.h

12.48 Elf64_Dyn Struct Reference

ELF64 dynamic entry.

`#include <elf.h>`

Collaboration diagram for Elf64_Dyn:

Data Fields

- Elf64_Sxword d_tag

  see DT_ values
- Elf64_Xword d_val

  integer values with various interpret.
- Elf64_Addr d_ptr

  program virtual addresses

12.48.1 Detailed Description

ELF64 dynamic entry.

Definition at line 469 of file elf.h.

12.48.2 Field Documentation

12.48.2.1 Elf64_Xword Elf64_Dyn::d_val

integer values with various interpret.

Definition at line 472 of file elf.h.

The documentation for this struct was generated from the following file:

- l4/util/elf.h

12.49 Elf64_Ehdr Struct Reference

ELF64 header.

```
#include <elf.h>
```

Collaboration diagram for Elf64_Ehdr:

Data Fields

- Elf64_Half e_type

  type of ELF file
- Elf64_Half e_machine

  required architecture
- Elf64_Word e_version

  file version
- Elf64_Addr e_entry

  initial eip
- Elf64_Off e_phoff

  offset of program header table
- Elf64_Off e_shoff

  offset of file header table
- Elf64_Word e_flags

  processor-specific flags
- Elf64_Half e_ehsize

  size of ELF header
- Elf64_Half e_phentsize

  size of program header entry
- Elf64_Half e_phnum

of entries in prog.

- Elf64_Half e_shentsize

  size of section header entry

- Elf64_Half e_shnum

  of entries in sect.

- Elf64_Half e_shstrndx

  sect.head.tab.idx of strtab

12.49.1 Detailed Description

ELF64 header.

Definition at line 138 of file elf.h.

12.49.2 Field Documentation

12.49.2.1 Elf64_Half Elf64_Ehdr::e_phnum

of entries in prog.

head. tab.

Definition at line 149 of file elf.h.

12.49.2.2 Elf64_Half Elf64_Ehdr::e_shnum

of entries in sect.

head. tab.

Definition at line 151 of file elf.h.

The documentation for this struct was generated from the following file:

- l4/util/elf.h

12.50 Elf64_Phdr Struct Reference

ELF64 program header.

```
#include <elf.h>
```

Collaboration diagram for Elf64_Phdr:

Data Fields

- Elf64_Word p_type

  type of program section
- Elf64_Word p_flags

  flags
- Elf64_Off p_offset

  file offset of program section
- Elf64_Addr p_vaddr

  memory address of prog section
- Elf64_Addr p_paddr

  physical address (ignored)
- Elf64_Xword p_filesz

  file size of program section
- Elf64_Xword p_memsz

  memory size of program section
- Elf64_Xword p_align

  alignment of section

12.50.1 Detailed Description

ELF64 program header.

Definition at line 391 of file elf.h.

The documentation for this struct was generated from the following file:

- l4/util/elf.h

12.51 Elf64_Shdr Struct Reference

ELF64 section header.

```
#include <elf.h>
```

Collaboration diagram for Elf64_Shdr:

Data Fields

- Elf64_Word sh_name

  name of sect (idx into strtab)
- Elf64_Word sh_type

  section's type
- Elf64_Xword sh_flags

  section's flags
- Elf64_Addr sh_addr

  memory address of section
- Elf64_Off sh_offset

  file offset of section
- Elf64_Xword sh_size

  file size of section
- Elf64_Word sh_link

  idx to associated header section
- Elf64_Word sh_info

  extra info of header section
- Elf64_Xword sh_addralign

  address alignment constraints
- Elf64_Xword sh_entsize

  size of entry if sect is table

12.51.1 Detailed Description

ELF64 section header.

Definition at line 316 of file elf.h.

The documentation for this struct was generated from the following file:

- l4/util/elf.h

12.52 Elf64_Sym Struct Reference

ELF64 symbol table entry.

```
#include <elf.h>
```

Collaboration diagram for Elf64_Sym:

Data Fields

- Elf64_Word st_name

  name of symbol (idx symstrtab)
- unsigned char st_info

  type and binding info
- unsigned char st_other

  undefined
- Elf64_Half st_shndx

  associated section header
- Elf64_Addr st_value

  value of associated symbol
- Elf64_Xword st_size

  size of associated symbol

12.52.1 Detailed Description

ELF64 symbol table entry.

Definition at line 771 of file elf.h.

The documentation for this struct was generated from the following file:

- l4/util/elf.h

12.53 gfxbitmap_offset Struct Reference

offsets in pmap[] and bmap[]

```
#include <bitmap.h>
```

Collaboration diagram for gfxbitmap_offset:

Data Fields

- l4_uint32_t preskip_x

  skip pixels at beginning of line
- l4_uint32_t preskip_y

  skip lines
- l4_uint32_t endskip_x

  skip pixels at end of line

12.53.1 Detailed Description

offsets in pmap[] and bmap[]

Definition at line 69 of file bitmap.h.

The documentation for this struct was generated from the following file:

- l4/libgfxbitmap/bitmap.h

12.54 L4::Alloc_list Class Reference

A simple list-based allocator.

```
#include <alloc.h>
```

Collaboration diagram for L4::Alloc_list:

12.54.1 Detailed Description

A simple list-based allocator.

Definition at line 33 of file alloc.h.

The documentation for this class was generated from the following file:

- l4/cxx/alloc.h

12.55 L4::Base_exception Class Reference

Base class for all exceptions, thrown by the L4Re framework.

`#include <l4/cxx/exceptions>`

Inheritance diagram for L4::Base_exception:

Collaboration diagram for L4::Base_exception:

Public Member Functions

- virtual char const ∗ str () const =0 throw ()

  Return a human readable string for the exception.
- virtual ∼Base_exception () throw ()

  Destruction.

Protected Member Functions

- Base_exception () throw ()

  Create a base exception.

12.55.1 Detailed Description

Base class for all exceptions, thrown by the L4Re framework.

This is the abstract base of all exceptions thrown within the L4Re framework. It is basically also a good idea to use it as base of all user defined exceptions.

Definition at line 116 of file exceptions.

The documentation for this class was generated from the following file:

- l4/cxx/exceptions

12.56 L4::Basic_registry Class Reference

This registry returns the corresponding server object based on the label of an Ipc_gate.

Inheritance diagram for L4::Basic_registry:

Collaboration diagram for L4::Basic_registry:

Static Public Member Functions

- static Value ∗ find (l4_umword_t label)

  Get the server object for an lpc_gate label.

- static l4_msgtag_t dispatch (l4_msgtag_t tag, l4_umword_t label, l4_utcb_t ∗utcb)

  The dispatch function called by the server loop.

12.56.1 Detailed Description

This registry returns the corresponding server object based on the label of an Ipc_gate.

Definition at line 359 of file ipc_epiface.

12.56.2 Member Function Documentation

12.56.2.1 static l4_msgtag_t L4::Basic_registry::dispatch (l4_msgtag_t *tag*, l4_umword_t *label*, l4_utcb_t * *utcb*) [inline],[static]

The dispatch function called by the server loop.

This function forwards the message to the server object identified by the given *label*.

Parameters

tag	The message tag used for the invocation.
label	The label used to find the object including the rights bits of the invoked capability.
utcb	The UTCB used for the invocation.

Returns

The return code from the object's dispatch function or -L4_ENOENT if the object does not exist.

Definition at line 384 of file ipc_epiface.

References L4_ENOENT, and l4_msgtag().

Here is the call graph for this function:

12.56.2.2 static Value* L4::Basic_registry::find (l4_umword_t *label*) [inline],[static]

Get the server object for an Ipc_gate label.

Parameters

label	The label usually stored in an Ipc_gate.

Returns

A pointer to the Epiface identified by the given label.

Definition at line 368 of file ipc_epiface.

The documentation for this class was generated from the following file:

- l4/sys/cxx/ipc_epiface

12.57 L4::Bounds_error Class Reference

Access out of bounds.

Inheritance diagram for L4::Bounds_error:

Collaboration diagram for L4::Bounds_error:

Additional Inherited Members

12.57.1 Detailed Description

Access out of bounds.

Definition at line 290 of file exceptions.

The documentation for this class was generated from the following file:

- l4/cxx/exceptions

12.58 L4::Cap< T > Class Template Reference

C++ interface for capabilities.

`#include <capability.h>`

Inheritance diagram for L4::Cap< T >:

Collaboration diagram for L4::Cap< T >:

Public Member Functions

- template<typename O >
  Cap (Cap< O > const &o) throw ()

  Create a copy from o, supporting implicit type casting.
- Cap (Cap_type cap) throw ()

  Constructor to create an invalid capability selector.
- Cap (l4_default_caps_t cap) throw ()

  Initialize capability with one of the default capability selectors.
- Cap (l4_cap_idx_t idx=L4_INVALID_CAP) throw ()

  Initialize capability, defaults to the invalid capability selector.
- Cap (No_init_type) throw ()

  Create an uninitialized cap selector.
- Cap move (Cap const &src) const

  Move a capability to this cap slot.
- Cap copy (Cap const &src) const

  Copy a capability to this cap slot.

- T * operator-> () const throw ()

  Member access of a `T`.

Friends

- class **L4::Kobject**

12.58.1 Detailed Description

template<typename T>class L4::Cap< T >

C++ interface for capabilities.

Template Parameters

T	Type of the object the capability points to.

The C++ version of a capability is comparable to a pointer, in fact it is a kind of smart pointer for our kernel objects and the objects derived from the kernel objects (L4::Kobject).

Add

```
#include <l4/sys/capability>
```

to your code to use the capability interface.

Examples:

examples/clntsrv/client.cc, examples/libs/l4re/c++/shared_ds/ds_clnt.cc, and examples/libs/l4re/streammap/client.cc.

Definition at line 13 of file capability.h.

12.58.2 Constructor & Destructor Documentation

12.58.2.1 template<typename T> template<typename O > L4::Cap< T >::Cap (Cap< O > const & *o*) throw) [inline]

Create a copy from `o`, supporting implicit type casting.

Parameters

o	The source selector that shall be copied (and casted).

Definition at line 221 of file capability.h.

12.58.2.2 template<typename T> L4::Cap< T >::Cap (Cap_type *cap*) throw) [inline]

Constructor to create an invalid capability selector.

Parameters

cap	Capability selector.

Definition at line 228 of file capability.h.

12.58.2.3 template<typename T> L4::Cap< T >::Cap (l4_default_caps_t *cap*) throw) [inline]

Initialize capability with one of the default capability selectors.

Parameters

cap	Capability selector.

Definition at line 234 of file capability.h.

12.58.2.4 template<typename T> L4::Cap< T >::Cap (l4_cap_idx_t *idx* = L4_INVALID_CAP) throw) [inline],[explicit]

Initialize capability, defaults to the invalid capability selector.

Parameters

idx	Capability selector.

Definition at line 240 of file capability.h.

12.58.3 Member Function Documentation

12.58.3.1 template<typename T> Cap L4::Cap< T >::copy (Cap< T > const & *src*) const [inline]

Copy a capability to this cap slot.

Parameters

src	the source capability slot.

Definition at line 263 of file capability.h.

12.58.3.2 template<typename T> Cap L4::Cap< T >::move (Cap< T > const & *src*) const [inline]

Move a capability to this cap slot.

Parameters

src	the source capability slot.

After this operation the source slot is no longer valid.

Definition at line 253 of file capability.h.

The documentation for this class was generated from the following file:

- l4/sys/cxx/capability.h

12.59 L4::Com_error Class Reference

Error conditions during IPC.

```
#include <l4/cxx/exceptions>
```

Inheritance diagram for L4::Com_error:

Collaboration diagram for L4::Com_error:

Public Member Functions

- Com_error (long err) throw ()

  Create a Com_error for the givel L4 IPC error code.

Additional Inherited Members

12.59.1 Detailed Description

Error conditions during IPC.

This exception encapsulates all IPC error conditions of L4 IPC.

Definition at line 275 of file exceptions.

12.59.2 Constructor & Destructor Documentation

12.59.2.1 L4::Com_error::Com_error (long *err*) throw) [inline], [explicit]

Create a Com_error for the givel L4 IPC error code.

Parameters

err	The L4 IPC error code (l4_ipc... return value).

Definition at line 282 of file exceptions.

The documentation for this class was generated from the following file:

- l4/cxx/exceptions

12.60 L4::Debugger Class Reference

C++ debugger interface.

Inheritance diagram for L4::Debugger:

Collaboration diagram for L4::Debugger:

Public Member Functions

- l4_msgtag_t set_object_name (const char ∗name, l4_utcb_t ∗utcb=l4_utcb()) throw ()

  Set the name of a kernel object.
- unsigned long global_id (l4_utcb_t ∗utcb=l4_utcb()) throw ()

  Get the globally unique ID of the object behind a capability.
- unsigned long kobj_to_id (l4_addr_t kobjp, l4_utcb_t ∗utcb=l4_utcb()) throw ()

  Get the globally unique ID of the object behind the kobject pointer.
- int query_log_typeid (const char ∗name, unsigned idx, l4_utcb_t ∗utcb=l4_utcb()) throw ()
- int query_log_name (unsigned idx, char ∗name, unsigned namelen, char ∗shortname, unsigned shortnamelen, l4_utcb_t ∗utcb=l4_utcb()) throw ()
- l4_msgtag_t switch_log (const char ∗name, unsigned on_off, l4_utcb_t ∗utcb=l4_utcb()) throw ()
- l4_msgtag_t get_object_name (unsigned id, char ∗name, unsigned size, l4_utcb_t ∗utcb=l4_utcb()) throw ()

  Get name of object with Id `id`.

Additional Inherited Members

12.60.1 Detailed Description

C++ debugger interface.

Attention

This API is subject to change! Do not rely on it in production code.

Include File

```
#include <l4/sys/debugger>
```

Definition at line 39 of file debugger.

12.60.2 Member Function Documentation

12.60.2.1 l4_msgtag_t L4::Debugger::get_object_name (unsigned *id*, char * *name*, unsigned *size*, l4_utcb_t * *utcb* = l4_utcb ()) throw) [inline]

Get name of object with Id `id`.

Parameters

	id	Id of the object whose name is asked.
`out`	*name*	Buffer to copy the name into. The buffer must be allocated by the caller.
	size	Length of the `name` buffer.
	utcb	The UTCB to use for the operation.

Returns

Syscall return tag

Definition at line 140 of file debugger.

References L4::Kobject::cap().

Here is the call graph for this function:

12.60.2.2 unsigned long L4::Debugger::global_id (l4_utcb_t * *utcb* = l4_utcb ()) throw) [inline]

Get the globally unique ID of the object behind a capability.

Parameters

utcb	The UTCB to use for the operation.

Return values

~0UL	The capability is invalid.
>=0	The global debugger id.

Definition at line 66 of file debugger.

References L4::Kobject::cap().

Here is the call graph for this function:

12.60.2.3 unsigned long L4::Debugger::kobj_to_id (l4_addr_t *kobjp*, l4_utcb_t * *utcb* = l4_utcb ()) throw) [inline]

Get the globally unique ID of the object behind the kobject pointer.

Parameters

kobjp	Kobject pointer
utcb	The UTCB to use for the operation.

Return values

~0UL	The capability or the Kobject pointer are invalid.
>=0	The globally unique id.

Definition at line 77 of file debugger.

References L4::Kobject::cap().

Here is the call graph for this function:

12.60.2.4 int L4::Debugger::query_log_name (unsigned *idx*, char * *name*, unsigned *namelen*, char * *shortname*, unsigned *shortnamelen*, l4_utcb_t * *utcb* = l4_utcb ()) throw) [inline]

Parameters

	idx	ID to query.
out	*name*	Buffer to copy name to. The buffer must be allocated by the caller.
	namelen	Buffer length of name.
out	*shortname*	Buffer to copy `shortname` to. The buffer must be allocated by the caller.
	shortnamelen	Buffer length of `shortname`.
	utcb	The UTCB to use for the operation.

Return values

0	Success
<0	Error

Definition at line 108 of file debugger.

References L4::Kobject::cap().

Here is the call graph for this function:

12.60.2.5 int L4::Debugger::query_log_typeid (const char * *name*, unsigned *idx*, l4_utcb_t * *utcb* = l4_utcb ()) throw) [inline]

Parameters

name	Name to query for.
idx	Idx to start searching, start with 0
utcb	The UTCB to use for the operation.

Return values

>=0	Id
<0	Error

Definition at line 90 of file debugger.

References L4::Kobject::cap().

Here is the call graph for this function:

12.60.2.6 l4_msgtag_t L4::Debugger::set_object_name (const char * *name*, l4_utcb_t * *utcb* = l4_utcb ()) throw) [inline]

Set the name of a kernel object.

Parameters

name	Name
utcb	The UTCB to use for the operation.

Returns

System call return tag.

Definition at line 55 of file debugger.

References L4::Kobject::cap().

Here is the call graph for this function:

12.60.2.7 l4_msgtag_t L4::Debugger::switch_log (const char * *name*, unsigned *on_off*, l4_utcb_t * *utcb* = l4_utcb ()) throw) [inline]

Parameters

name	Name of the log type.
on_off	1: turn log on, 0: turn log off
utcb	The UTCB to use for the operation.

Returns

Syscall return tag

Definition at line 125 of file debugger.

References L4::Kobject::cap().

Here is the call graph for this function:

The documentation for this class was generated from the following file:

- l4/sys/debugger

12.61 L4::Element_already_exists Class Reference

Exception for duplicate element insertions.

```
#include <l4/cxx/exceptions>
```

Inheritance diagram for L4::Element_already_exists:

Collaboration diagram for L4::Element_already_exists:

Additional Inherited Members

12.61.1 Detailed Description

Exception for duplicate element insertions.

Definition at line 203 of file exceptions.

The documentation for this class was generated from the following file:

- l4/cxx/exceptions

12.62 L4::Element_not_found Class Reference

Exception for a failed lookup (element not found).

`#include <l4/cxx/exceptions>`

Inheritance diagram for L4::Element_not_found:

Collaboration diagram for L4::Element_not_found:

Additional Inherited Members

12.62.1 Detailed Description

Exception for a failed lookup (element not found).

Definition at line 232 of file exceptions.

The documentation for this class was generated from the following file:

- l4/cxx/exceptions

12.63 L4::Exception_tracer Class Reference

Back-trace support for exceptions.

```
#include <l4/cxx/exceptions>
```

Inheritance diagram for L4::Exception_tracer:

Collaboration diagram for L4::Exception_tracer:

Public Member Functions

- void const ∗const ∗ pc_array () const throw ()

  Get the array containing the call trace.

- int frame_count () const throw ()

  Get the number of entries that are valid in the call trace.

Protected Member Functions

- Exception_tracer () throw ()

  Create a back trace.

12.63.1 Detailed Description

Back-trace support for exceptions.

This class holds an array of at most L4_CXX_EXCEPTION_BACKTRACE instruction pointers containing the call trace at the instant when an exception was thrown.

Definition at line 63 of file exceptions.

The documentation for this class was generated from the following file:

- l4/cxx/exceptions

12.64 L4::Factory Class Reference

C++ Factory interface to create kernel objects.

Inheritance diagram for L4::Factory:

Collaboration diagram for L4::Factory:

Data Structures

- struct Lstr

  Special type to add a pascal string into the factory create stream.
- struct Nil

  Special type to add a void argument into the factory create stream.
- class S

  Stream class for the create() argument stream.

Public Member Functions

- S create (Cap< void > target, long obj, l4_utcb_t *utcb=l4_utcb()) throw ()

  Generic create call to the factory.

- l4_msgtag_t create_task (Cap< Task > const &target_cap, l4_fpage_t const &utcb_area, l4_utcb_t *utcb=l4_utcb()) throw ()

  Create a new task.
- l4_msgtag_t create_thread (Cap< Thread > const &target_cap, l4_utcb_t *utcb=l4_utcb()) throw ()

  Create a new thread.
- l4_msgtag_t create_factory (Cap< Factory > const &target_cap, unsigned long limit, l4_utcb_t *utcb=l4_utcb()) throw ()

  Create a new factory.
- l4_msgtag_t create_gate (Cap< Kobject > const &target_cap, Cap< Thread > const &thread_cap, l4_umword_t label, l4_utcb_t *utcb=l4_utcb()) throw ()

  Create a new IPC gate.
- l4_msgtag_t create_irq (Cap< Irq >const &target_cap, l4_utcb_t *utcb=l4_utcb()) throw ()

  Create a new IRQ.
- l4_msgtag_t create_vm (Cap< Vm >const &target_cap, l4_utcb_t *utcb=l4_utcb()) throw ()

  Create a new virtual machine.

Additional Inherited Members

12.64.1 Detailed Description

C++ Factory interface to create kernel objects.

A factory is used to create all kinds of kernel objects:

- L4::Task
- L4::Thread
- L4::Factory
- L4::Ipc_gate
- L4::Irq
- L4::Vm

The factory is equipped with a limit that limits the amount of kernel memory available to that factory.

Note

The limit does not give any guarantee for the amount of available kernel memory.

Include File

```
#include <l4/sys/factory>
```

For the C interface refer to Factory.

Definition at line 54 of file factory.

12.64.2 Member Function Documentation

12.64.2.1 S L4::Factory::create (Cap< void > *target*, long *obj*, l4_utcb_t * *utcb* = l4_utcb ()) throw) [inline]

Generic create call to the factory.

Parameters

out	*target*	Capability selector for the new object. The caller must allocate the capability slot. The kernel stores the new objects's capability into this slot.
	obj	The protocol ID that specifies which kind of object shall be created.
	utcb	The UTCB to use for the operation.

Returns

A create stream that allows adding additional arguments to the create() call.

This method does currently not directly invoke the factory. It returns a stream that shall invoke the factory after adding all additional arguments.

Usage:

```
L4::Cap<L4Re::Namespace> ns = L4Re::Util::cap_alloc.
      alloc<L4Re::Namespace>();
factory->create(ns, L4Re::Namespace::Protocol) << "Argument text";
```

Definition at line 234 of file factory.

References L4::Kobject::cap().

Referenced by L4Re::Mem_alloc::alloc().

Here is the call graph for this function:

Here is the caller graph for this function:

12.64.2.2 l4_msgtag_t L4::Factory::create_factory (Cap< Factory > const & *target_cap*, unsigned long *limit*, l4_utcb_t * *utcb* = l4_utcb ()) throw) [inline]

Create a new factory.

Parameters

out	*target_cap*	Capability selector for the root capability of the new factory. The caller must allocate the capability slot. The kernel stores the new factory's capability into this slot.
	limit	Limit for the new factory in bytes.
	utcb	The UTCB to use for the operation.

Returns

Syscall return tag

Note

The limit of the new factory is subtracted from the available amount of the factory used for creation.

Definition at line 306 of file factory.

References L4::Kobject::cap(), and l4_factory_create_factory_u().

Here is the call graph for this function:

12.64.2.3 l4_msgtag_t L4::Factory::create_gate (Cap< Kobject > const & *target_cap*, Cap< Thread > const & *thread_cap*, l4_umword_t *label*, l4_utcb_t * *utcb* = l4_utcb ()) throw) [inline]

Create a new IPC gate.

Parameters

out	*target_cap*	Capability selector for the root capability of the new IPC gate. The caller must allocate the capability slot. The kernel stores the new IPC gate's capability into this slot.
	thread_cap	Capability selector of the thread to bind the gate to.
	label	Label of the gate.
	utcb	The UTCB to use for the operation.

Returns

Syscall return tag

See also

L4::Ipc_gate

Definition at line 327 of file factory.

References L4::Kobject::cap(), and l4_factory_create_gate_u().

Here is the call graph for this function:

12.64.2.4 l4_msgtag_t L4::Factory::create_irq (Cap< Irq >const & *target_cap,* l4_utcb_t * *utcb* = l4_utcb ()) throw) [inline]

Create a new IRQ.

Parameters

out	*target_cap*	Capability selector for the root capability of the new IRQ. The caller must allocate the capability slot. The kernel stores the new IRQ's capability into this slot.
	utcb	The UTCB to use for the operation.

Returns

Syscall return tag

See also

L4::Irq

Definition at line 345 of file factory.

References L4::Kobject::cap(), and l4_factory_create_irq_u().

Here is the call graph for this function:

12.64.2.5 l4_msgtag_t L4::Factory::create_task (Cap< Task > const & *target_cap*, l4_fpage_t const & *utcb_area*, l4_utcb_t * *utcb* = l4_utcb ()) throw) [inline]

Create a new task.

Parameters

out	*target_cap*	Capability selector for the root capability of the new task. The caller must allocate the capability slot. The kernel stores the new task's capability into this slot.
	utcb_area	Flexpage that describes the area used for UTCBs and vCPU state-save-areas of the new task.
	utcb	The UTCB to use for the operation.

Returns

Syscall return tag

Note

The size of the UTCB area specifies indirectly the number of UTCBs available for this task. Refer to L4::Task::add_ku_mem / l4_task_add_ku_mem() for adding more of this type of memory.

See also

L4::Task

Definition at line 269 of file factory.

References L4::Kobject::cap(), and l4_factory_create_task_u().

Here is the call graph for this function:

12.64.2.6 l4_msgtag_t L4::Factory::create_thread (Cap< Thread > const & *target_cap*, l4_utcb_t * *utcb* = l4_utcb ()) throw) [inline]

Create a new thread.

Parameters

out	*target_cap*	Capability selector for the root capability of the new thread. The caller must allocate the capability slot. The kernel stores the new thread's capability into this slot.

	utcb	The UTCB to use for the operation.

Returns

Syscall return tag

See also

L4::Thread

Definition at line 287 of file factory.

References L4::Kobject::cap(), and l4_factory_create_thread_u().

Here is the call graph for this function:

12.64.2.7 l4_msgtag_t L4::Factory::create_vm (Cap< Vm >const & *target_cap*, l4_utcb_t * *utcb* = l4_utcb ()) throw) [inline]

Create a new virtual machine.

Parameters

out	*target_cap*	Capability selector for the root capability of the new VM. The caller must allocate the capability slot. The kernel stores the new VM's capability into this slot.
	utcb	The UTCB to use for the operation.

Returns

Syscall return tag

See also

L4::Vm

Definition at line 362 of file factory.

References L4::Kobject::cap(), and l4_factory_create_vm_u().

Here is the call graph for this function:

The documentation for this class was generated from the following file:

- l4/sys/factory

12.65 L4::Factory::Lstr Struct Reference

Special type to add a pascal string into the factory create stream.

Collaboration diagram for L4::Factory::Lstr:

Data Fields

- char const * s

  The character buffer.

- int len

  The number of characters in the buffer.

12.65.1 Detailed Description

Special type to add a pascal string into the factory create stream.

This encapsulates a string that has an explicit length.

Definition at line 70 of file factory.

The documentation for this struct was generated from the following file:

- l4/sys/factory

12.66 L4::Factory::Nil Struct Reference

Special type to add a void argument into the factory create stream.

Collaboration diagram for L4::Factory::Nil:

12.66.1 Detailed Description

Special type to add a void argument into the factory create stream.

Definition at line 63 of file factory.

The documentation for this struct was generated from the following file:

- l4/sys/factory

12.67 L4::Factory::S Class Reference

Stream class for the create() argument stream.

Collaboration diagram for L4::Factory::S:

Public Member Functions

- S (S const &o)

  Create a copy.
- S (l4_cap_idx_t f, long obj, L4::Cap< void > target, l4_utcb_t *utcb) throw ()

  Create a stream for a specific create() call.
- ~S ()

  Commit the operation in the destructor to have a cool syntax for create().
- operator l4_msgtag_t ()

  Explicitly commits the operation and returns the result.
- S & operator<< (l4_mword_t i)

  Put a single l4_mword_t as next argument.
- S & operator<< (l4_umword_t i)

  Put a single l4_umword_t as next argument.
- S & operator<< (char const *s)

  Add a zero-terminated string as next argument.
- S & operator<< (Lstr const &s)

  Add a pascal string as next argument.
- S & operator<< (Nil)

  Add an empty argument.
- S & operator<< (l4_fpage_t d)

  Add a flex page as next argument.

12.67.1 Detailed Description

Stream class for the create() argument stream.

This stream allows a variable number of arguments to be added to a create() call.

Definition at line 91 of file factory.

12.67.2 Constructor & Destructor Documentation

12.67.2.1 L4::Factory::S::S (l4_cap_idx_t *f*, long *obj*, L4::Cap< void > *target*, l4_utcb_t * *utcb*) throw) [inline]

Create a stream for a specific create() call.

Parameters

	f	The capability for the factory object (L4::Factory).
	obj	The protocol ID to describe the type of the object that shall be created.
out	*target*	The capability selector for the new object. The caller must allocate the capability slot. The kernel stores the new object's capability into this slot.
	utcb	The UTCB to use for the operation.

Definition at line 117 of file factory.

12.67.3 Member Function Documentation

12.67.3.1 L4::Factory::S::operator l4_msgtag_t ( ) [inline]

Explicitly commits the operation and returns the result.

Returns

The result of the create() operation.

Definition at line 137 of file factory.

References l4_msgtag_t::raw.

12.67.3.2 S& L4::Factory::S::operator<< (l4_mword_t *i*) [inline]

Put a single l4_mword_t as next argument.

Parameters

i	The value to add as next argument.

Definition at line 149 of file factory.

12.67.3.3 S& L4::Factory::S::operator<< (l4_umword_t *i*) [inline]

Put a single l4_umword_t as next argument.

Parameters

i	The value to add as next argument.

Definition at line 160 of file factory.

12.67.3.4 S& L4::Factory::S::operator<< (char const * *s*) [inline]

Add a zero-terminated string as next argument.

Parameters

s	The string to add as next argument.

Definition at line 171 of file factory.

12.67.3.5 S& L4::Factory::S::operator<< (Lstr const & *s*) [inline]

Add a pascal string as next argument.

Parameters

s	The string to add as next argument.

Definition at line 182 of file factory.

References L4::Factory::Lstr::len, and L4::Factory::Lstr::s.

12.67.3.6 S& L4::Factory::S::operator<< (l4_fpage_t *d*) [inline]

Add a flex page as next argument.

Parameters

d	The flex page to add (there will be no map operation).

Definition at line 202 of file factory.

The documentation for this class was generated from the following file:

- l4/sys/factory

12.68 L4::Icu Class Reference

C++ Icu interface.

Inheritance diagram for L4::Icu:

Collaboration diagram for L4::Icu:

Data Structures

- class Info

  This class encapsulates information about an ICU.

Public Member Functions

- l4_msgtag_t bind (unsigned irqnum, L4::Cap< Triggerable > irq, l4_utcb_t *utcb=l4_utcb()) throw ()

  Bind an interrupt line of an interrupt controller to an interrupt object.
- l4_msgtag_t unbind (unsigned irqnum, L4::Cap< Triggerable > irq, l4_utcb_t *utcb=l4_utcb()) throw ()

  Remove binding of an interrupt line from the interrupt controller object.
- l4_msgtag_t info (l4_icu_info_t *info, l4_utcb_t *utcb=l4_utcb()) throw ()

Get information about the capabilites of the ICU.

- l4_msgtag_t msi_info (l4_umword_t irqnum, l4_uint64_t source, l4_icu_msi_info_t *msi_info)

  Get MSI info about IRQ.

- l4_msgtag_t mask (unsigned irqnum, l4_umword_t *label=0, l4_timeout_t to=L4_IPC_NEVER, l4_utcb_t *utcb=l4_utcb()) throw ()

  Mask an IRQ line.

- l4_msgtag_t set_mode (unsigned irqnum, l4_umword_t mode, l4_utcb_t *utcb=l4_utcb()) throw ()

  Set interrupt mode.

Additional Inherited Members

12.68.1 Detailed Description

C++ Icu interface.

To setup an IRQ line the following steps are required:

1. set_mode() (optional if IRQ has a default mode)
2. Irq::attach() to attach the IRQ capability to a thread
3. bind()
4. unmask() to receive the first IRQ

Include File

```
#include <l4/sys/icu>
```

Definition at line 221 of file irq.

12.68.2 Member Function Documentation

12.68.2.1 l4_msgtag_t L4::Icu::bind (unsigned *irqnum*, L4::Cap< Triggerable > *irq*, l4_utcb_t * *utcb* = l4_utcb ()) throw) [inline]

Bind an interrupt line of an interrupt controller to an interrupt object.

Parameters

irqnum	IRQ line at the ICU.
irq	IRQ object for the given IRQ line to bind to this ICU.
utcb	UTCB to be used for this operation, usually the UTCB of the calling thread.

Returns

Syscall return tag

Definition at line 263 of file irq.

References L4::Kobject::cap(), and l4_icu_bind_u().

Referenced by L4Re::Util::Event_t< PAYLOAD >::init().

Here is the call graph for this function:

Here is the caller graph for this function:

12.68.2.2 l4_msgtag_t L4::Icu::info (l4_icu_info_t * *info*, l4_utcb_t * *utcb* = l4_utcb ()) throw) [inline]

Get information about the capabilites of the ICU.

Parameters

out	*info*	Info structure to be filled with information.
	utcb	UTCB to be used for this operation, usually the UTCB of the calling thread.

Returns

Syscall return tag

Definition at line 294 of file irq.

References L4::Kobject::cap(), and l4_icu_info_u().

Here is the call graph for this function:

12.68.2.3 l4_msgtag_t L4::Icu::mask (unsigned *irqnum*, l4_umword_t * *label* = 0, l4_timeout_t *to* = L4_IPC_NEVER, l4_utcb_t * *utcb* = l4_utcb ()) throw) [inline]

Mask an IRQ line.

Parameters

irqnum	IRQ line at the ICU.
label	If NULL this function is a send-only message to the ICU. If not NULL this function will enter an open wait after sending the mask message.
to	The timeout-pair (send and receive) that shall be used for this operation. The receive timeout is used with a non-NULL `label` only.
utcb	UTCB to be used for this operation, usually the UTCB of the calling thread.

Returns

Syscall return tag

Definition at line 337 of file irq.

References L4::Kobject::cap(), and l4_icu_mask_u().

Here is the call graph for this function:

12.68.2.4 l4_msgtag_t L4::Icu::msi_info (l4_umword_t *irqnum*, l4_uint64_t *source*, l4_icu_msi_info_t * *msi_info*)

Get MSI info about IRQ.

Parameters

irqnum	IRQ line at the ICU.
source	Platform dependent requester ID for MSIs. On IA32 we use a 20bit source filter value as described in the Intel IRQ remapping specification.
msi_info	A l4_icu_msi_info_t structure receiving the address and the data value to trigger this MSI.

Returns

Syscall return tag

12.68.2.5 l4_msgtag_t L4::Icu::set_mode (unsigned *irqnum*, l4_umword_t *mode*, l4_utcb_t * *utcb* = l4_utcb ()) throw) [inline]

Set interrupt mode.

Parameters

irqnum	IRQ line at the ICU.
mode	Mode, see L4_irq_mode.
utcb	UTCB to be used for this operation, usually the UTCB of the calling thread.

Returns

Syscall return tag

Definition at line 359 of file irq.

References L4::Kobject::cap(), and l4_icu_set_mode_u().

Here is the call graph for this function:

12.68.2.6 l4_msgtag_t L4::Icu::unbind (unsigned *irqnum*, L4::Cap< Triggerable > *irq*, l4_utcb_t * *utcb* = l4_utcb ()) throw) [inline]

Remove binding of an interrupt line from the interrupt controller object.

Parameters

irqnum	IRQ line at the ICU.
irq	IRQ object to remove from the ICU.
utcb	UTCB to be used for this operation, usually the UTCB of the calling thread.

Returns

Syscall return tag

Definition at line 279 of file irq.

References L4::Kobject::cap(), and l4_icu_unbind_u().

Here is the call graph for this function:

The documentation for this class was generated from the following file:

- l4/sys/irq

12.69 L4::Icu::Info Class Reference

This class encapsulates information about an ICU.

Inheritance diagram for L4::Icu::Info:

Collaboration diagram for L4::Icu::Info:

Additional Inherited Members

12.69.1 Detailed Description

This class encapsulates information about an ICU.

Definition at line 248 of file irq.

The documentation for this class was generated from the following file:

- l4/sys/irq

12.70 L4::Invalid_capability Class Reference

Indicates that an invalid object was invoked.

```
#include <l4/cxx/exceptions>
```

Inheritance diagram for L4::Invalid_capability:

Collaboration diagram for L4::Invalid_capability:

Public Member Functions

- Invalid_capability (Cap< void > const &o) throw ()

  Create an Invalid_obejct exception for the Object o.
- char const * str () const throw ()

  Return a human readable string for the exception.
- Cap< void > const & cap () const throw ()

  Get the object that caused the error.

Additional Inherited Members

12.70.1 Detailed Description

Indicates that an invalid object was invoked.

An Object is invalid if it has L4_INVALID_ID as server L4 UID, or if the server does not know the object ID.

Definition at line 246 of file exceptions.

12.70.2 Constructor & Destructor Documentation

12.70.2.1 L4::Invalid_capability::Invalid_capability (Cap< void > const & *o*) throw) [inline], [explicit]

Create an Invalid_obejct exception for the Object o.

Parameters

o	The object that caused the server side error.

Definition at line 256 of file exceptions.

12.70.3 Member Function Documentation

12.70.3.1 Cap<void> const& L4::Invalid_capability::cap ( ) const throw) [inline]

Get the object that caused the error.

Returns

The object that caused the error on invocation.

Definition at line 265 of file exceptions.

The documentation for this class was generated from the following file:

- l4/cxx/exceptions

12.71 L4::Iommu Class Reference

Interface for IO-MMUs used for DMA remapping.

Inheritance diagram for L4::Iommu:

Collaboration diagram for L4::Iommu:

Public Member Functions

- l4_msgtag_t bind (l4_uint64_t src_id, Ipc::Cap< Task > dma_space)

  Associate `dma_space` with the set of device(s) specified by `src_id`.
- l4_msgtag_t unbind (l4_uint64_t src_id, Ipc::Cap< Task > dma_space)

  Remove the association of the given DMA address space from the device(s) specified by `src_id`.

12.71.1 Detailed Description

Interface for IO-MMUs used for DMA remapping.

This interface allows to associate a DMA address space with a platform dependent set of devices.

Definition at line 14 of file iommu.

12.71.2 Member Function Documentation

12.71.2.1 l4_msgtag_t L4::Iommu::bind (l4_uint64_t *src_id,* Ipc::Cap< Task > *dma_space*)

Associate `dma_space` with the set of device(s) specified by `src_id`.

Parameters

src_id	Platform dependent source ID specifying the set of devices that shall use `dma_space` for DMA remapping.
dma_space	The DMA space (L4::Task created with L4_PROTO_DMA_SPACE) providing the mappings that shall be used for the device(s).

12.71.2.2 l4_msgtag_t L4::Iommu::unbind (l4_uint64_t *src_id*, Ipc::Cap< Task > *dma_space*)

Remove the association of the given DMA address space from the device(s) specified by `src_id`.

Parameters

src_id	Platform dependent source ID specifying the set of devices that shall no longer use `dma_space` for DMA remapping.
dma_space	The DMA space formerly associated with associate().

The documentation for this class was generated from the following file:

- l4/sys/iommu

12.72 L4::IOModifier Class Reference

Modifier class for the IO stream.

Collaboration diagram for L4::IOModifier:

12.72.1 Detailed Description

Modifier class for the IO stream.

An IO Modifier can be used to change properties of an IO stream for example the number format.

Definition at line 33 of file basic_ostream.

The documentation for this class was generated from the following file:

- l4/cxx/basic_ostream

12.73 L4::Ipc::Array< ELEM_TYPE, LEN_TYPE > Struct Template Reference

Array data type for dynamically sized arrays in RPCs.

Inheritance diagram for L4::Ipc::Array< ELEM_TYPE, LEN_TYPE >:

Collaboration diagram for L4::Ipc::Array< ELEM_TYPE, LEN_TYPE >:

Public Member Functions

- Array ()

  Make array.
- Array (LEN_TYPE length, ELEM_TYPE *data)

  Make array from length and data pointer.
- Array (typename Non_const< ELEM_TYPE >::type const &other)

  Make a const array from a non-const array.

12.73.1 Detailed Description

template<typename ELEM_TYPE, typename LEN_TYPE = Array_len_default>struct L4::Ipc::Array< ELEM_TYPE, LEN_TYPE >

Array data type for dynamically sized arrays in RPCs.

Template Parameters

ELEM_TYPE	The data type of an array element, should be 'const' when used as input.
LEN_TYPE	Data type used to store the number of elements in the array.

An Array generally encapsulates a data pointer and a length (number of elements). Array does *not* provide any storage for the data itself. The storage is either provided by a client-side caller or in the case of Array_ref is the message itself.

Arrays can be used as input or as output arguments, when used as input ELEM_TYPE should be qualified *const*, when used as output a reference to an array must be used and the ELEM_TYPE must *not* be qualified *const*. It is the caller's responsibility to provide an array buffer of sufficient length. If a message from the server is too large it will be silently truncated.

If backward compatibility with Ipc::Stream is required, then LEN_TYPE must be `unsigned long`.

Definition at line 85 of file ipc_array.

The documentation for this struct was generated from the following file:

- l4/sys/cxx/ipc_array

12.74 L4::Ipc::Array_in_buf< ELEM_TYPE, LEN_TYPE, MAX > Struct Template Reference

Server-side copy in buffer for Array.

Collaboration diagram for L4::Ipc::Array_in_buf< ELEM_TYPE, LEN_TYPE, MAX >:

Public Member Functions

- void copy_in (const_array a)

  copy in data from a source array
- Array_in_buf (const_array a)

  Make Array_in_buf from a const array.
- Array_in_buf (array a)

  Make Array_in_buf from a non-const array.

Data Fields

- ELEM_TYPE data [MAX]

  The data elements.
- LEN_TYPE length

  The length of the array.

12.74.1 Detailed Description

template<typename ELEM_TYPE, typename LEN_TYPE = Array_len_default, LEN_TYPE MAX = (L4_UTCB_GENERIC_DATA_SIZE ∗ sizeof(l4_umword_t)) / sizeof(ELEM_TYPE)>struct L4::Ipc::Array_in_buf< ELEM_TYPE, LEN_TYPE, MAX >

Server-side copy in buffer for Array.

Template Parameters

ELEM_TYPE	Data type of an array element.
LEN_TYPE	Data type for the number of elements in the array.
MAX	The maximum number of elements in the buffer. If the actual message is longer than the buffer, it will be silently truncated.

This type is assignment compatible to Array_ref<ELEM_TYPE, LEN_TYPE> and provides a transparent server-side copy-in mechanism for array parameters. The Array_in_buf provides the storage for the array data and receives a copy of the data passed to the server-function.

Definition at line 123 of file ipc_array.

The documentation for this struct was generated from the following file:

- l4/sys/cxx/ipc_array

12.75 L4::Ipc::Array_ref< ELEM_TYPE, LEN_TYPE > Struct Template Reference

Array reference data type for arrays located in the message.

Inheritance diagram for L4::Ipc::Array_ref< ELEM_TYPE, LEN_TYPE >:

Collaboration diagram for L4::Ipc::Array_ref< ELEM_TYPE, LEN_TYPE >:

12.75.1 Detailed Description

template<typename ELEM_TYPE, typename LEN_TYPE = Array_len_default>struct L4::Ipc::Array_ref< ELEM_TYPE, LEN_TYPE >

Array reference data type for arrays located in the message.

Note

Use Array for normal RPC interfaces, Array_ref is usually used as server-side argument, see Array.

Template Parameters

ELEM_TYPE	The data type of an array element, should be 'const' when used as input.
LEN_TYPE	Data type used to store the number of elements in the array.

Definition at line 39 of file ipc_array.

The documentation for this struct was generated from the following file:

- l4/sys/cxx/ipc_array

12.76 L4::Ipc::As_value< T > Struct Template Reference

Pass the argument as plain data value.

Collaboration diagram for L4::Ipc::As_value< T >:

12.76.1 Detailed Description

template<typename T>struct L4::Ipc::As_value< T >

Pass the argument as plain data value.

Template Parameters

T	The type of the original argument.

As_value<T> is used when *T* would be otherwise interpreted specially, for example as flex page. When using As_value<> then the argument is transmitted as plain data element.

Definition at line 127 of file ipc_types.

The documentation for this struct was generated from the following file:

- l4/sys/cxx/ipc_types

12.77 L4::Ipc::Buf_item Class Reference

RPC warpper for a receive item.

Collaboration diagram for L4::Ipc::Buf_item:

12.77.1 Detailed Description

RPC warpper for a receive item.

Definition at line 305 of file ipc_types.

The documentation for this class was generated from the following file:

- l4/sys/cxx/ipc_types

12.78 L4::Ipc::Call Struct Reference

RPC attribute for a standard RPC call.

Inheritance diagram for L4::Ipc::Call:

Collaboration diagram for L4::Ipc::Call:

12.78.1 Detailed Description

RPC attribute for a standard RPC call.

This is the default for the *FLAGS* parameter for L4::Ipc::Msg::Rpc_call L4::Ipc::Msg::Rpc_inline_call templates and declares the RPC to have default call semantics and timeouts.

Examples:

```
L4_RPC(long, send, (unsigned value), L4::Ipc::Call);
```

which is equvalent to:

```
L4_RPC(long, send, (unsigned value));
```

Definition at line 215 of file ipc_iface.

The documentation for this struct was generated from the following file:

- l4/sys/cxx/ipc_iface

12.79 L4::Ipc::Call_t< RIGHTS > Struct Template Reference

RPC attribute for an RPC call with required rights.

Inheritance diagram for L4::Ipc::Call_t< RIGHTS >:

Collaboration diagram for L4::Ipc::Call_t< RIGHTS >:

12.79.1 Detailed Description

template<unsigned RIGHTS>struct L4::Ipc::Call_t< RIGHTS >

RPC attribute for an RPC call with required rights.

Template Parameters

RIGHTS	The capability rights required for this call. L4_CAP_FPAGE_W and L4_CAP_FPAGE_S are checked within the server (and -L4_EPERM shall be returned if the caller has insufficient rights). L4_CAP_FPAGE_R is always on but might be specified for documentation purposes. Other rights cannot be used in this context, because they cannot be checked at the server side.

Examples:

```
L4_RPC(long, func, (unsigned value), L4::Ipc::Call_t<L4_CAP_FPAGE_RW>);
```

Definition at line 246 of file ipc_iface.

The documentation for this struct was generated from the following file:

- l4/sys/cxx/ipc_iface

12.80 L4::Ipc::Call_zero_send_timeout Struct Reference

RPC attribute for an RPC call, with zero send timeout.

Inheritance diagram for L4::Ipc::Call_zero_send_timeout:

Collaboration diagram for L4::Ipc::Call_zero_send_timeout:

12.80.1 Detailed Description

RPC attribute for an RPC call, with zero send timeout.

Definition at line 225 of file ipc_iface.

The documentation for this struct was generated from the following file:

- l4/sys/cxx/ipc_iface

12.81 L4::Ipc::Cap< T > Class Template Reference

Capability type for RPC interfaces (see `L4::Cap<T>`).

Collaboration diagram for L4::Ipc::Cap< T >:

Public Types

- enum { Rights_mask = 0xff, Cap_mask = L4_CAP_MASK }

Public Member Functions

- template<typename O >
  Cap (Cap< O > const &o)

  Make copy with conversion.
- Cap (L4::Cap< T > cap)

  Make a Cap from L4::Cap<T>, with minimal rights.
- template<typename O >
  Cap (L4::Cap< O > cap)

  Make IPC Cap from L4::Cap with conversion (and minimal rights).
- Cap ()

  Make an invalid cap.
- Cap (L4::Cap< T > cap, unsigned char rights)

  Make c Cap from L4::Cap<T>, with the given rights.
- L4::Cap< T > cap () const

  Return the L4::Cap<T> of this Cap.
- unsigned rights () const

  Return the rights bits stored in this IPC cap.
- L4::Ipc::Snd_fpage fpage () const

  Return the send flexpage for this Cap (see l4_fpage_t)
- bool is_valid () const throw ()

  Return true if this Cap is valid.

Static Public Member Functions

- static Cap from_ci (l4_cap_idx_t c)

  Create an IPC capability from a C capability index plus rights.

12.81.1 Detailed Description

template<typename T>class L4::Ipc::Cap< T >

Capability type for RPC interfaces (see `L4::Cap<T>`).

Template Parameters

T	type of the interface referenced by the capability.

In contrast to `L4::Cap<T>` this type additionally stores a rights mask that shall be used when the capability is transferred to the receiver. This allows to apply restrictions to the transferred capability in the form of a subset of the rights possessed by the sender.

See also

L4::Ipc::make_cap()

Definition at line 533 of file ipc_types.

12.81.2 Member Enumeration Documentation

12.81.2.1 template<typename T> anonymous enum

Enumerator

Rights_mask Mask for rights bits stored internally. L4_FPAGE_RIGHTS_MASK | 0xf0 (extended cap rights: USR1, USR2, USR3, and L4_FPAGE_C_NO_REF_CNT).

Cap_mask Mask for significant capability bits. (incl. the invalid bit to support invalid caps)

Definition at line 539 of file ipc_types.

12.81.3 Member Function Documentation

12.81.3.1 template<typename T> static Cap L4::Ipc::Cap< T >::from_ci (l4_cap_idx_t *c*) [inline], [static]

Create an IPC capability from a C capability index plus rights.

Parameters

c	C capability index with the lowest 8 bits used as rights for the map operation (see L4_fpage↩_rights).

Definition at line 581 of file ipc_types.

References L4::Ipc::Cap< T >::Cap(), L4::Ipc::Cap< T >::Cap_mask, and L4::Ipc::Cap< T >::Rights_mask.

Here is the call graph for this function:

The documentation for this class was generated from the following file:

- l4/sys/cxx/ipc_types

12.82 L4::Ipc::Gen_fpage< T > Class Template Reference

Generic RPC wrapper for L4 flex-pages.

Inherits T.

Collaboration diagram for L4::Ipc::Gen_fpage< T >:

Public Types

- enum Type

Type of mapping object, see L4_fpage_type.

- enum Map_type

  Kind of mapping.

- enum Cacheopt

  Caching options, see l4_fpage_cacheability_opt_t.

Public Member Functions

- bool is_valid () const

  Check if the capability is valid.

- bool cap_received () const

  Check if the capability has been mapped.

- bool id_received () const

  Check if a label was received instead of a mapping.

- bool local_id_received () const

  Check if a local capability id has been received.

- l4_umword_t data () const

  Return the raw flex page descriptor.

- l4_umword_t base_x () const

  Return the raw base descriptor.

12.82.1 Detailed Description

template<typename T>class L4::Ipc::Gen_fpage< T >

Generic RPC wrapper for L4 flex-pages.

Template Parameters

T	Underlying specific flexpage type.

Definition at line 321 of file ipc_types.

12.82.2 Member Function Documentation

12.82.2.1 template<typename T > bool L4::Ipc::Gen_fpage< T >::cap_received ( ) const [inline]

Check if the capability has been mapped.

The capability itself can then be retrieved from the cap slot that has been provided in the receive operation.

Definition at line 438 of file ipc_types.

12.82.2.2 template<typename T > bool L4::Ipc::Gen_fpage< T >::id_received ( ) const [inline]

Check if a label was received instead of a mapping.

For IPC gates, if the L4_RCV_ITEM_LOCAL_ID has been set, then only the label of the IPC gate will be provided if the gate is local to the receiver, i.e. the target thread of the IPC gate is in the same task as the receiving thread.

The label can be retrieved with Gen_fpage::data().

Definition at line 450 of file ipc_types.

12.82.2.3 template<typename T > bool L4::Ipc::Gen_fpage< T >::local_id_received ( ) const [inline]

Check if a local capability id has been received.

If the L4_RCV_ITEM_LOCAL_ID flag has been set by the receiver, and sender and receiver are in the same task, then only the capability index is transferred.

The capability can be retrieved with Gen_fpage::data().

Definition at line 460 of file ipc_types.

The documentation for this class was generated from the following file:

- l4/sys/cxx/ipc_types

12.83 L4::Ipc::In_out< T > Struct Template Reference

Mark an argument as in-out argument.

Collaboration diagram for L4::Ipc::In_out< T >:

12.83.1 Detailed Description

template<typename T>struct L4::Ipc::In_out< T >

Mark an argument as in-out argument.

Template Parameters

T	The original argument type, usually a pointer or a reference.

In_out<> is used when an otherwise output-only value shall also be used as input value.

Definition at line 52 of file ipc_types.

The documentation for this struct was generated from the following file:

- l4/sys/cxx/ipc_types

12.84 L4::Ipc::Iostream Class Reference

Input/Output stream for IPC [un]marshalling.

Inheritance diagram for L4::Ipc::Iostream:

Collaboration diagram for L4::Ipc::Iostream:

Public Member Functions

- Iostream (l4_utcb_t *utcb)

  Create an IPC IO stream with a single message buffer.
- void reset ()

  Reset the stream to its initial state.

IPC operations.

- l4_msgtag_t call (l4_cap_idx_t dst, l4_timeout_t timeout, long proto=0)
  Do an IPC call using the message in the output stream and receiving to the input stream.
- l4_msgtag_t **call** (l4_cap_idx_t dst, long proto=0)
- l4_msgtag_t reply_and_wait (l4_umword_t *src_dst, long proto=0)
  Do an IPC reply and wait.
- l4_msgtag_t **send_and_wait** (l4_cap_idx_t dest, l4_umword_t *src, long proto=0)
- l4_msgtag_t reply_and_wait (l4_umword_t *src_dst, l4_timeout_t timeout, long proto=0)
  Do an IPC reply and wait.
- l4_msgtag_t **send_and_wait** (l4_cap_idx_t dest, l4_umword_t *src, l4_timeout_t timeout, long proto=0)
- l4_msgtag_t **reply** (l4_timeout_t timeout, long proto=0)
- l4_msgtag_t **reply** (long proto=0)

12.84.1 Detailed Description

Input/Output stream for IPC [un]marshalling.

The lpc::Iostream is part of the AW Env IPC framework as well as lpc::Istream and lpc::Ostream. In particular an lpc::Iostream is a combination of an lpc::Istream and an lpc::Ostream. It can use either a single message buffer for receiving and sending messages or a pair of a receive and a send buffer. The stream also supports combined IPC operations such as call() and reply_and_wait(), which can be used to implement RPC functionality.

Examples:

examples/libs/l4re/c++/shared_ds/ds_srv.cc, examples/libs/l4re/streammap/client.cc, and examples/libs/l4re/streammap/server.cc.

Definition at line 787 of file ipc_stream.

12.84.2 Constructor & Destructor Documentation

12.84.2.1 L4::Ipc::Iostream::Iostream (l4_utcb_t * *utcb*) [inline], [explicit]

Create an IPC IO stream with a single message buffer.

Parameters

utcb	The message buffer used as backing store.

The created IO stream uses the same message buffer for sending and receiving IPC messages.

Definition at line 799 of file ipc_stream.

12.84.3 Member Function Documentation

12.84.3.1 l4_msgtag_t L4::Ipc::Iostream::call (l4_cap_idx_t *dst*, l4_timeout_t *timeout*, long *proto* = 0) [inline]

Do an IPC call using the message in the output stream and receiving to the input stream.

Parameters

dst	The destination L4 UID (thread) to call.
timeout	The IPC timeout for the call.
proto	The protocol value to use in the message tag.

Returns

The result dope of the IPC operation.

This is a combined IPC operation consisting of a send and a receive to/from the given destination `dst`.

A call is usually used by clients for RPCs to a server.

Examples:

examples/libs/l4re/streammap/client.cc.

Definition at line 958 of file ipc_stream.

References l4_ipc_call(), and L4::Ipc::Istream::tag().

Here is the call graph for this function:

12.84.3.2 l4_msgtag_t L4::Ipc::Iostream::reply_and_wait (l4_umword_t * *src_dst*, long *proto* = 0) [inline]

Do an IPC reply and wait.

Parameters

in,out	*src_dst*	Input: the destination for the send operation. Output: the source of the received message.
	proto	Protocol to use.

Returns

the result dope of the IPC operation.

This is a combined IPC operation consisting of a send operation and an open wait for any message.

A reply and wait is usually used by servers that reply to a client and wait for the next request by any other client.

Definition at line 872 of file ipc_stream.

References L4_IPC_SEND_TIMEOUT_0.

12.84.3.3 l4_msgtag_t L4::Ipc::Iostream::reply_and_wait (l4_umword_t * *src_dst*, l4_timeout_t *timeout*, long *proto* = 0) [inline]

Do an IPC reply and wait.

Parameters

in,out	*src_dst*	Input: the destination for the send operation. Output: the source of the received message.
	timeout	Timeout used for IPC.
	proto	Protocol to use.

Returns

the result dope of the IPC operation.

This is a combined IPC operation consisting of a send operation and an open wait for any message.

A reply and wait is usually used by servers that reply to a client and wait for the next request by any other client.

Definition at line 973 of file ipc_stream.

References l4_ipc_reply_and_wait(), and L4::Ipc::Istream::tag().

Here is the call graph for this function:

12.84.3.4 void L4::Ipc::Iostream::reset ( ) [inline]

Reset the stream to its initial state.

Input as well as the output stream are reset.

Definition at line 813 of file ipc_stream.

References L4::Ipc::Istream::reset(), and L4::Ipc::Ostream::reset().

Here is the call graph for this function:

The documentation for this class was generated from the following file:

- l4/cxx/ipc_stream

12.85 L4::Ipc::Istream Class Reference

Input stream for IPC unmarshalling.

Inheritance diagram for L4::Ipc::Istream:

Collaboration diagram for L4::Ipc::Istream:

Public Member Functions

- Istream (l4_utcb_t *utcb)

  Create an input stream for the given message buffer.

- void reset ()

  Reset the stream to empty, and ready for receive()/wait().

- template<typename T >
  bool has_more ()
  Check whether a value of type T can be obtained from the stream.
- l4_utcb_t * utcb () const
  Return utcb pointer.

Get/Put Functions.

These functions are basically used to implement the extraction operators (>>) and should not be called directly. See Istream::operator>>()

- template<typename T >
  unsigned long get (T *buf, unsigned long elems)
  Copy out an array of type `T` with `size` elements.
- template<typename T >
  void skip (unsigned long size)
  Skip size elements of type T in the stream.
- template<typename T >
  unsigned long get (Msg_ptr< T > const &buf, unsigned long elems=1)
  Read one size elements of type T from the stream and return a pointer.
- template<typename T >
  void get (T &v)
  Extract a single element of type T from the stream.
- bool **get** (Ipc::Varg *va)
- l4_msgtag_t tag () const
  Get the message tag of a received IPC.
- l4_msgtag_t & tag ()
  Get the message tag of a received IPC.

IPC operations.

- l4_msgtag_t wait (l4_umword_t *src)
  Wait for an incoming message from any sender.
- l4_msgtag_t wait (l4_umword_t *src, l4_timeout_t timeout)
  Wait for an incoming message from any sender.
- l4_msgtag_t receive (l4_cap_idx_t src)
  Wait for a message from the specified sender.
- l4_msgtag_t **receive** (l4_cap_idx_t src, l4_timeout_t timeout)

12.85.1 Detailed Description

Input stream for IPC unmarshalling.

Ipc::Istream is part of the dynamic IPC marshalling infrastructure, as well as Ipc::Ostream and Ipc::Iostream.

Ipc::Istream is an input stream supporting extraction of values from an IPC message buffer. A received IPC message can be unmarshalled using the usual extraction operator (>>).

There exist some special wrapper classes to extract arrays (see Ipc_buf_cp_in and Ipc_buf_in) and indirect strings (see Msg_in_buffer and Msg_io_buffer).

Definition at line 345 of file ipc_stream.

12.85.2 Constructor & Destructor Documentation

12.85.2.1 L4::Ipc::Istream::Istream (l4_utcb_t * *utcb*) [inline]

Create an input stream for the given message buffer.

The given message buffer is used for IPC operations wait()/receive() and received data can be extracted using the >> operator afterwards. In the case of indirect message parts a buffer of type Msg_in_buffer must be inserted into the stream before the IPC operation and contains received data afterwards.

Parameters

utcb	The message buffer to receive IPC messages.

Definition at line 359 of file ipc_stream.

12.85.3 Member Function Documentation

12.85.3.1 template<typename T > unsigned long L4::Ipc::Istream::get (T * *buf*, unsigned long *elems*) [inline]

Copy out an array of type `T` with `size` elements.

Parameters

buf	Pointer to a buffer for size elements of type T.
elems	Number of elements of type T to copy out.

See Istream::operator>>()

Definition at line 404 of file ipc_stream.

12.85.3.2 template<typename T > unsigned long L4::Ipc::Istream::get (Msg_ptr< T > const & *buf*, unsigned long *elems* = 1) [inline]

Read one size elements of type T from the stream and return a pointer.

Parameters

buf	A Msg_ptr that is actually set to point to the element in the stream.
elems	Number of elements to extract (default is 1).

In contrast to a normal get, this version does actually not copy the data but returns a pointer to the data.

See Istream::operator>>()

Definition at line 444 of file ipc_stream.

12.85.3.3 template<typename T > void L4::Ipc::Istream::get (T & *v*) [inline]

Extract a single element of type T from the stream.

Parameters

out	*v*	The element.

See Istream::operator>>()

Definition at line 465 of file ipc_stream.

12.85.3.4 l4_msgtag_t L4::Ipc::Istream::receive (l4_cap_idx_t *src*) [inline]

Wait for a message from the specified sender.

Parameters

src	The sender id to receive from.

Returns

The IPC result dope (l4_msgtag_t).

This is commonly known as 'closed wait'.

Definition at line 571 of file ipc_stream.

References L4_IPC_NEVER.

12.85.3.5 void L4::Ipc::Istream::reset ( ) [inline]

Reset the stream to empty, and ready for receive()/wait().

The stream is reset to the same state as on its creation.

Definition at line 369 of file ipc_stream.

References l4_msg_regs_t::mr.

Referenced by L4::Ipc::Iostream::reset().

Here is the caller graph for this function:

12.85.3.6 template<typename T > void L4::Ipc::Istream::skip (unsigned long *size*) [inline]

Skip size elements of type T in the stream.

Parameters

size	Number of elements to skip.

Definition at line 422 of file ipc_stream.

12.85.3.7 l4_msgtag_t L4::Ipc::Istream::tag ( ) const [inline]

Get the message tag of a received IPC.

Returns

The L4 message tag for the received IPC.

This is in particular useful for handling page faults or exceptions.

See Istream::operator>>()

Definition at line 504 of file ipc_stream.

Referenced by L4::Ipc::Iostream::call(), L4::Ipc::Iostream::reply_and_wait(), and wait().

Here is the caller graph for this function:

12.85.3.8 l4_msgtag_t& L4::Ipc::Istream::tag ( ) [inline]

Get the message tag of a received IPC.

Returns

A reference to the L4 message tag for the received IPC.

This is in particular useful for handling page faults or exceptions.

See Istream::operator>>()

Definition at line 516 of file ipc_stream.

12.85.3.9 l4_msgtag_t L4::Ipc::Istream::wait (l4_umword_t * *src*) [inline]

Wait for an incoming message from any sender.

Parameters

out	*src*	Contains the sender after a successful IPC operation.

Returns

Syscall return tag.

This wait is actually known as 'open wait'.

Definition at line 547 of file ipc_stream.

References L4_IPC_NEVER.

12.85.3.10 l4_msgtag_t L4::Ipc::Istream::wait (l4_umword_t * *src*, l4_timeout_t *timeout*) [inline]

Wait for an incoming message from any sender.

Parameters

out	*src*	Contains the sender after a successful IPC operation.
	timeout	Timeout used for IPC.

Returns

The IPC result dope (l4_msgtag_t).

This wait is actually known as 'open wait'.

Definition at line 1005 of file ipc_stream.

References l4_ipc_wait(), and tag().

Here is the call graph for this function:

The documentation for this class was generated from the following file:

- l4/cxx/ipc_stream

12.86 L4::Ipc::Msg::Clnt_val_ops< MTYPE, DIR, CLASS > Struct Template Reference

Defines client-side handling of 'MTYPE' as RPC argument.

Inheritance diagram for L4::Ipc::Msg::Clnt_val_ops< MTYPE, DIR, CLASS >:

Collaboration diagram for L4::Ipc::Msg::Clnt_val_ops< MTYPE, DIR, CLASS >:

12.86.1 Detailed Description

template<typename MTYPE, typename DIR, typename CLASS>struct L4::Ipc::Msg::Clnt_val_ops< MTYPE, DIR, CLASS >

Defines client-side handling of 'MTYPE' as RPC argument.

Template Parameters

MTYPE	Elem<T>::arg_type (where T is the type used in the RPC definition)
DIR	Dir_in (client -> server), or Dir_out (server -> client)
CLASS	Cls_data, Cls_item, or Cls_buffer

Definition at line 221 of file ipc_basics.

The documentation for this struct was generated from the following file:

- l4/sys/cxx/ipc_basics

12.87 L4::Ipc::Msg::Cls_buffer Struct Reference

Marker type for receive buffer values.

Inheritance diagram for L4::Ipc::Msg::Cls_buffer:

Collaboration diagram for L4::Ipc::Msg::Cls_buffer:

12.87.1 Detailed Description

Marker type for receive buffer values.

Definition at line 165 of file ipc_basics.

The documentation for this struct was generated from the following file:

- l4/sys/cxx/ipc_basics

12.88 L4::Ipc::Msg::Cls_data Struct Reference

Marker type for data values.

Inheritance diagram for L4::Ipc::Msg::Cls_data:

Collaboration diagram for L4::Ipc::Msg::Cls_data:

12.88.1 Detailed Description

Marker type for data values.

Definition at line 161 of file ipc_basics.

The documentation for this struct was generated from the following file:

- l4/sys/cxx/ipc_basics

12.89 L4::Ipc::Msg::Cls_item Struct Reference

Marker type for item values.

Inheritance diagram for L4::Ipc::Msg::Cls_item:

Collaboration diagram for L4::Ipc::Msg::Cls_item:

12.89.1 Detailed Description

Marker type for item values.

Definition at line 163 of file ipc_basics.

The documentation for this struct was generated from the following file:

- l4/sys/cxx/ipc_basics

12.90 L4::Ipc::Msg::Dir_in Struct Reference

Marker type for input values.

Inheritance diagram for L4::Ipc::Msg::Dir_in:

Collaboration diagram for L4::Ipc::Msg::Dir_in:

12.90.1 Detailed Description

Marker type for input values.

Definition at line 156 of file ipc_basics.

The documentation for this struct was generated from the following file:

- l4/sys/cxx/ipc_basics

12.91 L4::Ipc::Msg::Dir_out Struct Reference

Marker type for output values.

Inheritance diagram for L4::Ipc::Msg::Dir_out:

Collaboration diagram for L4::Ipc::Msg::Dir_out:

12.91.1 Detailed Description

Marker type for output values.

Definition at line 158 of file ipc_basics.

The documentation for this struct was generated from the following file:

- l4/sys/cxx/ipc_basics

12.92 L4::Ipc::Msg::Do_in_data Struct Reference

Marker for Input data.

Inheritance diagram for L4::Ipc::Msg::Do_in_data:

Collaboration diagram for L4::Ipc::Msg::Do_in_data:

12.92.1 Detailed Description

Marker for Input data.

Definition at line 169 of file ipc_basics.

The documentation for this struct was generated from the following file:

- l4/sys/cxx/ipc_basics

12.93 L4::Ipc::Msg::Do_in_items Struct Reference

Marker for Input items.

Inheritance diagram for L4::Ipc::Msg::Do_in_items:

Collaboration diagram for L4::Ipc::Msg::Do_in_items:

12.93.1 Detailed Description

Marker for Input items.

Definition at line 173 of file ipc_basics.

The documentation for this struct was generated from the following file:

- l4/sys/cxx/ipc_basics

12.94 L4::Ipc::Msg::Do_out_data Struct Reference

Marker for Output data.

Inheritance diagram for L4::Ipc::Msg::Do_out_data:

Collaboration diagram for L4::Ipc::Msg::Do_out_data:

12.94.1 Detailed Description

Marker for Output data.

Definition at line 171 of file ipc_basics.

The documentation for this struct was generated from the following file:

- l4/sys/cxx/ipc_basics

12.95 L4::Ipc::Msg::Do_out_items Struct Reference

Marker for Output items.

Inheritance diagram for L4::Ipc::Msg::Do_out_items:

Collaboration diagram for L4::Ipc::Msg::Do_out_items:

12.95.1 Detailed Description

Marker for Output items.

Definition at line 175 of file ipc_basics.

The documentation for this struct was generated from the following file:

- l4/sys/cxx/ipc_basics

12.96 L4::Ipc::Msg::Do_rcv_buffers Struct Reference

Marker for receive buffers.

Inheritance diagram for L4::Ipc::Msg::Do_rcv_buffers:

Collaboration diagram for L4::Ipc::Msg::Do_rcv_buffers:

12.96.1 Detailed Description

Marker for receive buffers.

Definition at line 177 of file ipc_basics.

The documentation for this struct was generated from the following file:

- l4/sys/cxx/ipc_basics

12.97 L4::Ipc::Msg::Elem< Array< A, LEN > & > Struct Template Reference

Array as output argument.

Collaboration diagram for L4::Ipc::Msg::Elem< Array< A, LEN > & >:

Public Types

- typedef Array< A, LEN > & arg_type

  Array<> & at the interface.

- typedef Array_ref< A, LEN > svr_type

  Array_ref<> as server storage type.

- typedef svr_type & svr_arg_type

  Array_ref<> & at the server side.

12.97.1 Detailed Description

template<typename A, typename LEN>struct L4::Ipc::Msg::Elem< Array< A, LEN > & >

Array as output argument.

Definition at line 167 of file ipc_array.

The documentation for this struct was generated from the following file:

- l4/sys/cxx/ipc_array

12.98 L4::Ipc::Msg::Elem< Array< A, LEN > > Struct Template Reference

Array as input arguments.

Collaboration diagram for L4::Ipc::Msg::Elem< Array< A, LEN > >:

Public Types

- typedef Array< A, LEN > arg_type

  Array<> as argument at the interface.
- typedef Array_ref< A, LEN > svr_type

  Array_ref<> at the server side.

12.98.1 Detailed Description

template<typename A, typename LEN>struct L4::Ipc::Msg::Elem< Array< A, LEN > >

Array as input arguments.

Definition at line 155 of file ipc_array.

The documentation for this struct was generated from the following file:

- l4/sys/cxx/ipc_array

12.99 L4::Ipc::Msg::Elem< Array_ref< A, LEN > & > Struct Template Reference

Array_ref as output argument.

Collaboration diagram for L4::Ipc::Msg::Elem< Array_ref< A, LEN > & >:

Public Types

- typedef Array_ref< A, LEN > & arg_type

  Array_ref<> at the interface.

- typedef Array_ref< typename L4::Types::Remove_const< A > ::type, LEN > svr_type

  Array_ref<> as server storage.

- typedef svr_type & svr_arg_type

  Array_ref<> & as server argument.

12.99.1 Detailed Description

template<typename A, typename LEN>struct L4::Ipc::Msg::Elem< Array_ref< A, LEN > & >

Array_ref as output argument.

Definition at line 180 of file ipc_array.

The documentation for this struct was generated from the following file:

- l4/sys/cxx/ipc_array

12.100 L4::Ipc::Msg::Is_valid_rpc_type< T > Struct Template Reference

Type trait defining a valid RPC parameter type.

Inheritance diagram for L4::Ipc::Msg::Is_valid_rpc_type< T >:

Collaboration diagram for L4::Ipc::Msg::Is_valid_rpc_type< T >:

Additional Inherited Members

12.100.1 Detailed Description

template<typename T>struct L4::Ipc::Msg::Is_valid_rpc_type< T >

Type trait defining a valid RPC parameter type.

Definition at line 350 of file ipc_basics.

The documentation for this struct was generated from the following file:

- l4/sys/cxx/ipc_basics

12.101 L4::Ipc::Msg::Svr_arg_pack< IPC_TYPE > Struct Template Reference

Server-side RPC arguments data structure used to provide arguments to the server-side implementation of an RPC function.

Collaboration diagram for L4::Ipc::Msg::Svr_arg_pack< IPC_TYPE >:

12.101.1 Detailed Description

template<typename IPC_TYPE>struct L4::Ipc::Msg::Svr_arg_pack< IPC_TYPE >

Server-side RPC arguments data structure used to provide arguments to the server-side implementation of an RPC function.

Definition at line 155 of file ipc_server.

The documentation for this struct was generated from the following file:

- l4/sys/cxx/ipc_server

12.102 L4::Ipc::Msg::Svr_val_ops< MTYPE, DIR, CLASS > Struct Template Reference

Defines server-side handling for `MTYPE` server arguments.

Inheritance diagram for L4::Ipc::Msg::Svr_val_ops< MTYPE, DIR, CLASS >:

Collaboration diagram for L4::Ipc::Msg::Svr_val_ops< MTYPE, DIR, CLASS >:

12.102.1 Detailed Description

template<typename MTYPE, typename DIR, typename CLASS>struct L4::Ipc::Msg::Svr_val_ops< MTYPE, DIR, CLASS >

Defines server-side handling for `MTYPE` server arguments.

Template Parameters

MTYPE	Elem<T>::svr_type (where T is the type used in the RPC definition)
DIR	Dir_in (client -> server), or Dir_out (server -> client)
CLASS	Cls_data, Cls_item, or Cls_buffer

Definition at line 275 of file ipc_basics.

The documentation for this struct was generated from the following file:

- l4/sys/cxx/ipc_basics

12.103 L4::Ipc::Msg_ptr< T > Class Template Reference

Pointer to an element of type T in an Ipc::Istream.

Collaboration diagram for L4::Ipc::Msg_ptr< T >:

Public Member Functions

- Msg_ptr (T *&p)

Create a Msg_ptr object that set pointer p to point into the message buffer.

12.103.1 Detailed Description

template<typename T>class L4::Ipc::Msg_ptr< T >

Pointer to an element of type T in an Ipc::Istream.

This wrapper can be used to extract an element of type T from an Ipc::Istream, whereas the data is not copied out, but a pointer into the message buffer itself is returned. With is mechanism it is possible to avoid an extra copy of large data structures from a received IPC message, instead the returned pointer gives direct access to the data in the message.

See msg_ptr().

Definition at line 241 of file ipc_stream.

12.103.2 Constructor & Destructor Documentation

12.103.2.1 template<typename T> L4::Ipc::Msg_ptr< T >::Msg_ptr (T *& p) [inline],[explicit]

Create a Msg_ptr object that set pointer p to point into the message buffer.

Parameters

p	The pointer that is adjusted to point into the message buffer.

Definition at line 252 of file ipc_stream.

The documentation for this class was generated from the following file:

- l4/cxx/ipc_stream

12.104 L4::Ipc::Opt< T > Struct Template Reference

Attribute for defining an optional RPC argument.

Collaboration diagram for L4::lpc::Opt< T >:

Public Member Functions

- Opt ()

  Make an absent optional argument.
- Opt (T value)

  Make a present optional argument with the given value.
- Opt & operator= (T value)

  Assign a value to the optional argument (makes the argument present)
- void set_valid (bool valid=true)

  Set the argument to present or absent.
- T ∗ operator-> ()

  Get the pointer to the value.
- T const ∗ operator-> () const

  Get the const pointer to the value.
- T value () const

  Get the value.
- T & value ()

  Get the value.
- bool is_valid () const

  Get true if present, false if not.

Data Fields

- T _value

  The value.
- bool _valid

  True if the optional argument is present, false else.

12.104.1 Detailed Description

template<typename T>struct L4::Ipc::Opt< T >

Attribute for defining an optional RPC argument.

Definition at line 147 of file ipc_types.

The documentation for this struct was generated from the following file:

- l4/sys/cxx/ipc_types

12.105 L4::Ipc::Ostream Class Reference

Output stream for IPC marshalling.

Inheritance diagram for L4::Ipc::Ostream:

Collaboration diagram for L4::Ipc::Ostream:

Public Member Functions

- Ostream (l4_utcb_t *utcb)

  Create an IPC output stream using the given message buffer `utcb`.
- void reset ()

  Reset the stream to empty, same state as a newly created stream.
- l4_utcb_t * utcb () const

Return utcb pointer.

Get/Put functions.

These functions are basically used to implement the insertion operators (<<) and should not be called directly.

- template<typename T >
  bool put (T *buf, unsigned long size)
  Put an array with `size` elements of type `T` into the stream.
- template<typename T >
  bool put (T const &v)
  Insert an element of type `T` into the stream.
- int **put** (Varg const &va)
- template<typename T >
  int **put** (Varg_t< T > const &va)
- l4_msgtag_t tag () const
  Extract the L4 message tag from the stream.
- l4_msgtag_t & tag ()
  Extract a reference to the L4 message tag from the stream.

IPC operations.

- l4_msgtag_t send (l4_cap_idx_t dst, long proto=0, unsigned flags=0)
  Send the message via IPC to the given receiver.

12.105.1 Detailed Description

Output stream for IPC marshalling.

Ipc::Ostream is part of the dynamic IPC marshalling infrastructure, as well as Ipc::Istream and Ipc::Iostream.

Ipc::Ostream is an output stream supporting insertion of values into an IPC message buffer. A IPC message can be marshalled using the usual insertion operator <<, see IPC stream operators .

There exist some special wrapper classes to insert arrays (see Ipc::Buf_cp_out) and indirect strings (see Msg_↩out_buffer and Msg_io_buffer).

Definition at line 619 of file ipc_stream.

12.105.2 Member Function Documentation

12.105.2.1 template<typename T > bool L4::Ipc::Ostream::put (T * *buf*, unsigned long *size*) [inline]

Put an array with `size` elements of type `T` into the stream.

Parameters

buf	A pointer to the array to insert into the buffer.
size	The number of elements in the array.

Definition at line 656 of file ipc_stream.

12.105.2.2 template<typename T > bool L4::Ipc::Ostream::put (T const & *v*) [inline]

Insert an element of type `T` into the stream.

Parameters

v	The element to insert.

Definition at line 674 of file ipc_stream.

12.105.2.3 l4_msgtag_t L4::Ipc::Ostream::send (l4_cap_idx_t *dst*, long *proto* = 0, unsigned *flags* = 0) [inline]

Send the message via IPC to the given receiver.

Parameters

dst	The destination for the message.
proto	Protocol to use.
flags	Flags to use.

Returns

Syscall return tag.

Definition at line 951 of file ipc_stream.

References L4_IPC_NEVER, l4_ipc_send(), and L4_MSGTAG_FLAGS.

Here is the call graph for this function:

12.105.2.4 l4_msgtag_t L4::Ipc::Ostream::tag ( ) const [inline]

Extract the L4 message tag from the stream.

Returns

the extracted L4 message tag.

Definition at line 702 of file ipc_stream.

12.105.2.5 l4_msgtag_t& L4::Ipc::Ostream::tag ( ) [inline]

Extract a reference to the L4 message tag from the stream.

Returns

A reference to the L4 message tag.

Definition at line 709 of file ipc_stream.

The documentation for this class was generated from the following file:

- l4/cxx/ipc_stream

12.106 L4::Ipc::Out< T > Struct Template Reference

Mark an argument as a output value in an RPC signature.

Collaboration diagram for L4::Ipc::Out< T >:

12.106.1 Detailed Description

template<typename T>struct L4::Ipc::Out< T >

Mark an argument as a output value in an RPC signature.

Template Parameters

T	The original type of the argument.

Note

The use of Out<> is usually not needed, because typical out-put data types in C++ (pointers to non-const objects or non-const references are interpreted as output values anyway. However, there are some data types, such as returned capabilities that can be marked as such by using Out<>.

Definition at line 42 of file ipc_types.

The documentation for this struct was generated from the following file:

- l4/sys/cxx/ipc_types

12.107 L4::Ipc::Ret_array< T > Struct Template Reference

Dynamically sized output array of type T.

Collaboration diagram for L4::Ipc::Ret_array< T >:

12.107.1 Detailed Description

template<typename T>struct L4::Ipc::Ret_array< T >

Dynamically sized output array of type T.

Template Parameters

T	The data-type of each array element.

Ret_array<> is a special dynamically sized output array where the number of transmitted elements is passed in the return value of the call (if positive)

Definition at line 34 of file ipc_ret_array.

The documentation for this struct was generated from the following file:

- l4/sys/cxx/ipc_ret_array

12.108 L4::Ipc::Send_only Struct Reference

RPC attribute for a send-only RPC.

Collaboration diagram for L4::Ipc::Send_only:

12.108.1 Detailed Description

RPC attribute for a send-only RPC.

This class can be used as FLAGS parameter to L4::Ipc::Msg::Rpc_call and L4::Ipc::Msg::Rpc_inline_call templates and declares the RPC to use send-only semantics and timeouts.

Examples:

```
L4_RPC(long, send, (unsigned value), L4::Ipc::Send_only);
```

Definition at line 263 of file ipc_iface.

The documentation for this struct was generated from the following file:

- l4/sys/cxx/ipc_iface

12.109 L4::Ipc::Small_buf Class Reference

A receive item for receiving a single capability.

Collaboration diagram for L4::Ipc::Small_buf:

Public Member Functions

- Small_buf (L4::Cap< void > cap, unsigned long flags=0)

  Create a receive item from a C++ cap.
- Small_buf (l4_cap_idx_t cap, unsigned long flags=0)

  Create a receive item from a C cap.

12.109.1 Detailed Description

A receive item for receiving a single capability.

This class is the main abstraction for receiving capabilities via Ipc::Istream. To receive a capability an instance of Small_buf that refers to an empty capability slot must be inserted into the Ipc::Istream before the receive operation.

Definition at line 268 of file ipc_types.

12.109.2 Constructor & Destructor Documentation

12.109.2.1 L4::Ipc::Small_buf::Small_buf (L4::Cap< void > *cap*, unsigned long *flags* = 0) [inline],[explicit]

Create a receive item from a C++ cap.

Parameters

cap	Capability slot where to save the capability.
flags	Receive buffer flags, see l4_msg_item_consts_t. L4_RCV_ITEM_SINGLE_CAP will always be set.

Definition at line 278 of file ipc_types.

12.109.2.2 L4::Ipc::Small_buf::Small_buf (l4_cap_idx_t *cap*, unsigned long *flags* = 0) [inline],[explicit]

Create a receive item from a C cap.

Parameters

cap	Capability slot where to save the capability.
flags	Receive buffer flags, see l4_msg_item_consts_t. L4_RCV_ITEM_SINGLE_CAP will always be set.

Definition at line 285 of file ipc_types.

The documentation for this class was generated from the following file:

- l4/sys/cxx/ipc_types

12.110 L4::Ipc::Snd_item Class Reference

RPC wrapper for a send item.

Collaboration diagram for L4::Ipc::Snd_item:

12.110.1 Detailed Description

RPC wrapper for a send item.

Definition at line 294 of file ipc_types.

The documentation for this class was generated from the following file:

- l4/sys/cxx/ipc_types

12.111 L4::Ipc::Str_cp_in< T > Class Template Reference

Abstraction for extracting a zero-termintaed string from an Ipc::Istream.

Collaboration diagram for L4::Ipc::Str_cp_in< T >:

Public Member Functions

- Str_cp_in (T *v, unsigned long &size)

  Create a buffer for extracting an array from an Ipc::Istream.

12.111.1 Detailed Description

template<typename T>class L4::Ipc::Str_cp_in< T >

Abstraction for extracting a zero-termintaed string from an Ipc::Istream.

An instance of Str_cp_in can be used to extract a zero-terminated string an Ipc::Istream. The data from the received message is thereby copied to the given buffer and size is set to the number of characters found in the stream. The string is zero terminated in any circumstances. When the given buffer is smaller than the received string the last byte in the buffer will be the zero terminator. In the case the received string is shorter than the given buffer the zero termination will be placed behind the received data. This provides a zero-terminated result even in cases where the sender did not provide proper termination or in cases of too small receiver buffers.

See also

str_cp_in().

Definition at line 191 of file ipc_stream.

12.111.2 Constructor & Destructor Documentation

12.111.2.1 template<typename T> L4::Ipc::Str_cp_in< T >::Str_cp_in (T * *v*, unsigned long & *size*) [inline]

Create a buffer for extracting an array from an Ipc::Istream.

Parameters

	v	The buffer for string.
in,out	*size*	Input: The number of bytes available in v Output: The number of bytes received (including the terminator).

Definition at line 202 of file ipc_stream.

The documentation for this class was generated from the following file:

- l4/cxx/ipc_stream

12.112 L4::Ipc::Varg Class Reference

Variably sized RPC argument.

Inherited by L4::Ipc::Varg_t< T >.

Collaboration diagram for L4::Ipc::Varg:

Public Types

- typedef l4_umword_t Tag

  The data type for the tag.

Public Member Functions

- L4_varg_type type () const
- int length () const
- Tag tag () const
- void tag (Tag tag)

  Set Varg tag (usually from message)
- void data (char const ∗d)

  Set Varg to indirect data value (usually in UTCB)
- char const ∗ data () const
- Varg ()=default

  Make uninitialized Varg.
- Varg (L4_varg_type t, void const ∗v, int len)

  Make an indirect varg.
- template<typename V >
  Va_type< V >::Ret_value value () const
- template<typename T >
  bool is_of () const
- bool is_nil () const

- bool is_of_int () const
- template<typename T >
  bool get_value (typename Va_type< T >::Value *v) const

  Get the value of the Varg as type T.
- template<typename T >
  void set_value (void const *d)

  Set to indirect value of type T.
- template<typename T >
  void set_direct_value (T val, typename L4::Types::Enable_if< sizeof(T)<=sizeof(char const *), bool >↩ ::type=true)

  Set to directly stored value of type T.
- template<typename T >
  Varg (T const *data)

  Make Varg from indirect value (pointer)
- Varg (char const *data)

  Make Varg from null-terminated string.
- template<typename T >
  Varg (T data, typename L4::Types::Enable_if< sizeof(T)<=sizeof(char const *), bool >::type=true)

  Make Varg from direct value.

12.112.1 Detailed Description

Variably sized RPC argument.

Definition at line 96 of file ipc_varg.

12.112.2 Member Function Documentation

12.112.2.1 char const* L4::Ipc::Varg::data ( ) const [inline]

Returns

pointer to the data, also safe for direct data

Definition at line 120 of file ipc_varg.

Referenced by Varg().

Here is the caller graph for this function:

12.112.2.2 template<typename T > bool L4::Ipc::Varg::get_value (typename Va_type< T >::Value * *v*) const [inline]

Get the value of the Varg as type T.

Template Parameters

T	The expected type of the Varg.

Parameters

v	Pointer to store the value

Returns

true when the Varg is of type T, false if not

Definition at line 181 of file ipc_varg.

12.112.2.3 bool L4::Ipc::Varg::is_nil ( ) const [inline]

Returns

true if the Varg is of nil type.

Definition at line 168 of file ipc_varg.

12.112.2.4 template<typename T > bool L4::Ipc::Varg::is_of ( ) const [inline]

Returns

true if the Varg is of type T

Definition at line 165 of file ipc_varg.

References type().

Here is the call graph for this function:

12.112.2.5 bool L4::Ipc::Varg::is_of_int ( ) const [inline]

Returns

true if the Varg is an integer type (signed or unsigned).

Definition at line 171 of file ipc_varg.

References type().

Here is the call graph for this function:

12.112.2.6 int L4::Ipc::Varg::length ( ) const [inline]

Returns

the length field of the tag

Definition at line 111 of file ipc_varg.

Referenced by L4::Ipc::Varg_list_ref::next().

Here is the caller graph for this function:

12.112.2.7 Tag L4::Ipc::Varg::tag ( ) const [inline]

Returns

the tag value (the Direct_data bit masked)

Definition at line 113 of file ipc_varg.

Referenced by L4::Ipc::Varg_list_ref::next().

Here is the caller graph for this function:

12.112.2.8 L4_varg_type L4::Ipc::Varg::type ( ) const [inline]

Returns

the type field of the tag

Definition at line 109 of file ipc_varg.

Referenced by is_of(), and is_of_int().

Here is the caller graph for this function:

12.112.2.9 template<typename V > Va_type<V>::Ret_value L4::Ipc::Varg::value ( ) const [inline]

Template Parameters

V	The data type of the value to retrieve.

Precondition

The Varg must be of type *V* (otherwise the result is unpredictable).

Returns

The value of the Varg as type V.

Definition at line 151 of file ipc_varg.

The documentation for this class was generated from the following file:

- l4/sys/cxx/ipc_varg

12.113 L4::Ipc::Varg_list< MAX > Class Template Reference

Self-contained list of variable-sized RPC parameters.

Inheritance diagram for L4::Ipc::Varg_list< MAX >:

Collaboration diagram for L4::Ipc::Varg_list< MAX >:

Public Member Functions

- Varg_list (Varg_list_ref const &r)

  Create a parameter list as a copy from a referencing list.

12.113.1 Detailed Description

template<unsigned MAX>class L4::Ipc::Varg_list< MAX >

Self-contained list of variable-sized RPC parameters.

Works like Varg_list_ref but contains a full copy of the data. Use this as a parameter in server functions, if the handler function needs to use the UTCB (e.g. while sending further IPC).

Definition at line 236 of file ipc_varg.

The documentation for this class was generated from the following file:

- l4/sys/cxx/ipc_varg

12.114 L4::Ipc::Varg_list_ref Class Reference

List of variable-sized RPC parameters as received by the server.

Inheritance diagram for L4::Ipc::Varg_list_ref:

Collaboration diagram for L4::Ipc::Varg_list_ref:

Public Member Functions

- Varg_list_ref ()

  Create an empty parameter list.
- Varg_list_ref (void const ∗start, void const ∗end)

  Create a parameter list over a given memory region.
- Varg next ()

  Get the next parameter in the list.

12.114.1 Detailed Description

List of variable-sized RPC parameters as received by the server.

The list can be traversed exactly once using *next()*.

This is a reference list, where the returned Varg point to data in the underlying storage, conventionally the UTCB. This type should only be used in server functions when the implementation can ensure that all content is read before the UTCB is reused (e.g. for IPC), otherwise use Varg_list.

Definition at line 249 of file ipc_varg.

12.114.2 Constructor & Destructor Documentation

12.114.2.1 L4::Ipc::Varg_list_ref::Varg_list_ref (void const * *start*, void const * *end*) [inline]

Create a parameter list over a given memory region.

Parameters

start	Pointer to start of the parameter list.
end	Pointer to end of the list (inclusive).

Definition at line 266 of file ipc_varg.

The documentation for this class was generated from the following file:

- l4/sys/cxx/ipc_varg

12.115 L4::Ipc_gate Class Reference

The C++ IPC gate interface.

Inheritance diagram for L4::Ipc_gate:

Collaboration diagram for L4::Ipc_gate:

Public Member Functions

- l4_msgtag_t bind_thread (Ipc::Cap< Thread > t, l4_umword_t label)

  Bind a thread to an IPC gate.
- l4_msgtag_t get_infos (l4_umword_t *label)

  Get information about the IPC-gate.

Additional Inherited Members

12.115.1 Detailed Description

The C++ IPC gate interface.

IPC gates are used to create secure communication channels between protection domains. An IPC gate can be created using the L4::Factory interface. L4::Ipc_gate::bind_thread() binds an L4::Thread as the receiver of all messages to an IPC gate.

The bind_thread() call allows to assign each IPC gate a kernel protected, machine-word sized payload called a *label*. It securely identifies the gate. The two least significant bits of the *label* are ORed with the L4_CAP_FPA↩

GE_S and L4_CAP_FPAGE_W bits stored in the capability when transfered to the receiver. This means the *label* should usually have its two least significant bits set to zero. The *label* is only visible in the L4::Task which is running the thread the IPC gate was bound to and cannot be altered by the sender.

Include File

```
#include <l4/sys/ipc_gate>
```

For the C interface refer to the C IPC-Gate API.

Definition at line 55 of file ipc_gate.

12.115.2 Member Function Documentation

12.115.2.1 l4_msgtag_t L4::Ipc_gate::bind_thread (Ipc::Cap< Thread > *t*, l4_umword_t *label*)

Bind a thread to an IPC gate.

Parameters

t	Thread object that shall be bound to this IPC gate.
label	Label to assign to `this` IPC gate. The two least significant bits should usually be set to zero.

Returns

System call return tag.

12.115.2.2 l4_msgtag_t L4::Ipc_gate::get_infos (l4_umword_t * *label*)

Get information about the IPC-gate.

Parameters

`out`	*label*	The label of the IPC gate is returned here.

Returns

System call return tag.

The documentation for this class was generated from the following file:

- l4/sys/ipc_gate

12.116 L4::Ipc_svr::Br_manager_no_buffers Class Reference

Empty implementation of Server_iface.

Inheritance diagram for L4::Ipc_svr::Br_manager_no_buffers:

Collaboration diagram for L4::lpc_svr::Br_manager_no_buffers:

Public Member Functions

- int alloc_buffer_demand (Demand const &demand)

  Tells the server to allocate buffers for the given demand.
- L4::Cap< void > get_rcv_cap (int) const

  Returns L4::Cap<void>::Invalid, we have no buffer management.
- int realloc_rcv_cap (int)

  Returns -L4_ENOMEM, we have no buffer management.
- int add_timeout (Timeout *, l4_kernel_clock_t)

  Returns -L4_ENOSYS, we have no timeout queue.
- int remove_timeout (Timeout *)

  Returns -L4_ENOSYS, we have no timeout queue.

Protected Member Functions

- unsigned first_free_br () const

  Returns 1 as first free buffer.

- void setup_wait (l4_utcb_t *utcb, L4::Ipc_svr::Reply_mode)

  Setup wait function for the server loop (Server<>).

Additional Inherited Members

12.116.1 Detailed Description

Empty implementation of Server_iface.

This implementation of Server_iface provides no buffer or timeout management at all it just returns errors for all calls that express other than empty demands. However, this may be useful for very simple servers that serve simple server objects only.

Definition at line 216 of file ipc_server_loop.

12.116.2 Member Function Documentation

12.116.2.1 int L4::Ipc_svr::Br_manager_no_buffers::alloc_buffer_demand (Demand const & *demand*) [inline], [virtual]

Tells the server to allocate buffers for the given demand.

Parameters

demand	The total server-side demand of receive buffers needed for a given interface, see Demand.

This function is not called by user applications directly. Usually the server implementation or the registry implementation calls this function whenever a new object is registered at the server.

Returns

success (0) if demand is empty, -L4_ENOMEM else.

Implements L4::Ipc_svr::Server_iface.

Definition at line 223 of file ipc_server_loop.

References L4_ENOMEM, L4_EOK, and L4::Type_info::Demand::no_demand().

Here is the call graph for this function:

The documentation for this class was generated from the following file:

- l4/sys/cxx/ipc_server_loop

12.117 L4::Ipc_svr::Compound_reply Struct Reference

Mix in for LOOP_HOOKS to always use compound reply and wait.

Inheritance diagram for L4::Ipc_svr::Compound_reply:

Collaboration diagram for L4::Ipc_svr::Compound_reply:

12.117.1 Detailed Description

Mix in for LOOP_HOOKS to always use compound reply and wait.

Definition at line 77 of file ipc_server_loop.

The documentation for this struct was generated from the following file:

- l4/sys/cxx/ipc_server_loop

12.118 L4::Ipc_svr::Default_loop_hooks Struct Reference

Default LOOP_HOOKS.

Inheritance diagram for L4::Ipc_svr::Default_loop_hooks:

Collaboration diagram for L4::Ipc_svr::Default_loop_hooks:

Additional Inherited Members

12.118.1 Detailed Description

Default LOOP_HOOKS.

Combination of Ignore_errors, Default_timeout, Compound_reply, and Br_manager_no_buffers.

Definition at line 268 of file ipc_server_loop.

The documentation for this struct was generated from the following file:

- l4/sys/cxx/ipc_server_loop

12.119 L4::Ipc_svr::Default_setup_wait Struct Reference

Mix in for LOOP_HOOKS for setup_wait no op.

Collaboration diagram for L4::Ipc_svr::Default_setup_wait:

12.119.1 Detailed Description

Mix in for LOOP_HOOKS for setup_wait no op.

Definition at line 88 of file ipc_server_loop.

The documentation for this struct was generated from the following file:

- l4/sys/cxx/ipc_server_loop

12.120 L4::Ipc_svr::Default_timeout Struct Reference

Mix in for LOOP_HOOKS to use a 0 send and a infinite receive timeout.

Inheritance diagram for L4::Ipc_svr::Default_timeout:

Collaboration diagram for L4::Ipc_svr::Default_timeout:

12.120.1 Detailed Description

Mix in for LOOP_HOOKS to use a 0 send and a infinite receive timeout.

Definition at line 69 of file ipc_server_loop.

The documentation for this struct was generated from the following file:

- l4/sys/cxx/ipc_server_loop

12.121 L4::Ipc_svr::Direct_dispatch< R > Struct Template Reference

Direct disptach helper, for forwarding dispatch calls a registry *R*.

Inheritance diagram for L4::Ipc_svr::Direct_dispatch< R >:

Collaboration diagram for L4::Ipc_svr::Direct_dispatch< R >:

Public Member Functions

- Direct_dispatch (R &r)

  Make a direct dispatcher.
- l4_msgtag_t operator() (l4_msgtag_t tag, l4_umword_t obj, l4_utcb_t *utcb)

  call operator forwarding to r.dispatch()

Data Fields

- R & r

  stores a reference to the registry object

12.121.1 Detailed Description

template<typename R>struct L4::Ipc_svr::Direct_dispatch< R >

Direct disptach helper, for forwarding dispatch calls a registry *R*.

Template Parameters

R	Data type of the registry that is used for dispatching to different server objects, usually based on the protected IPC label.

Definition at line 139 of file ipc_server_loop.

The documentation for this struct was generated from the following file:

- l4/sys/cxx/ipc_server_loop

12.122 L4::Ipc_svr::Direct_dispatch< R * > Struct Template Reference

Direct disptach helper, for forwarding dispatch calls via a pointer to a registry *R*.

Collaboration diagram for L4::Ipc_svr::Direct_dispatch< R * >:

Public Member Functions

- Direct_dispatch (R *r)

  Make a direct dispatcher.
- l4_msgtag_t operator() (l4_msgtag_t tag, l4_umword_t obj, l4_utcb_t *utcb)

  call operator forwarding to r->dispatch()

Data Fields

- R * r

  stores a pointer to the registry object

12.122.1 Detailed Description

template<typename R>struct L4::Ipc_svr::Direct_dispatch< R * >

Direct disptach helper, for forwarding dispatch calls via a pointer to a registry *R*.

Template Parameters

R	Data type of the registry that is used for dispatching to different server objects, usually based on the protected IPC label.

Definition at line 160 of file ipc_server_loop.

The documentation for this struct was generated from the following file:

- l4/sys/cxx/ipc_server_loop

12.123 L4::Ipc_svr::Exc_dispatch< R, Exc > Struct Template Reference

Dispatch helper wrapping try {} catch {} around the dispatch call.

Inheritance diagram for L4::Ipc_svr::Exc_dispatch< R, Exc >:

Collaboration diagram for L4::Ipc_svr::Exc_dispatch< R, Exc >:

Public Member Functions

- Exc_dispatch (R r)

  Make en exception handling dispatcher.
- l4_msgtag_t operator() (l4_msgtag_t tag, l4_umword_t obj, l4_utcb_t *utcb)

  Dispatch the call via Direct_dispatch<R>() and handle exceptions.

12.123.1 Detailed Description

template<typename R, typename Exc>struct L4::Ipc_svr::Exc_dispatch< R, Exc >

Dispatch helper wrapping try {} catch {} around the dispatch call.

Template Parameters

R	Data type of the registry used for dispatching to objects.
Exc	Data type of the exceptions that shall be catched. This data type must provide a member err_no() that returns the negative integer (int) error code for the exception.

This dispatcher wraps Direct_dispatch<R> with a try-catch (Exc).

Definition at line 184 of file ipc_server_loop.

The documentation for this struct was generated from the following file:

- l4/sys/cxx/ipc_server_loop

12.124 L4::Ipc_svr::Ignore_errors Struct Reference

Mix in for LOOP_HOOKS to ignore IPC errors.

Inheritance diagram for L4::Ipc_svr::Ignore_errors:

Collaboration diagram for L4::Ipc_svr::Ignore_errors:

12.124.1 Detailed Description

Mix in for LOOP_HOOKS to ignore IPC errors.

Definition at line 61 of file ipc_server_loop.

The documentation for this struct was generated from the following file:

- l4/sys/cxx/ipc_server_loop

12.125 L4::Ipc_svr::Server_iface Class Reference

Interface for server-loop related functions.

Inheritance diagram for L4::Ipc_svr::Server_iface:

Collaboration diagram for L4::lpc_svr::Server_iface:

Public Types

- typedef L4::Type_info::Demand Demand

  Data type expressing server-side demand for receive buffers.

Public Member Functions

- Server_iface ()

  Make a server interface.
- virtual int alloc_buffer_demand (Demand const &demand)=0

  Tells the server to allocate buffers for the given demand.
- virtual L4::Cap< void > get_rcv_cap (int index) const =0

  Get capability slot allocated to the given receive buffer.
- virtual int realloc_rcv_cap (int index)=0

  Allocate a new capability for the given receive buffer.
- virtual int add_timeout (Timeout *timeout, l4_kernel_clock_t time)=0

  Add a timeout to the server internal timeout queue.
- virtual int remove_timeout (Timeout *timeout)=0

  Remove the given timeout from the timer queue.
- template<typename T >
  L4::Cap< T > rcv_cap (int index) const

  Get given receive buffer as typed capability.
- L4::Cap< void > rcv_cap (int index) const

  Get receive cap with the given index as generic (void) type.

12.125.1 Detailed Description

Interface for server-loop related functions.

This interface provides access to high-level server-loop related functions, such as management of receive buffers and timeouts.

Definition at line 45 of file ipc_epiface.

12.125.2 Member Function Documentation

12.125.2.1 virtual int L4::Ipc_svr::Server_iface::add_timeout (Timeout * *timeout*, l4_kernel_clock_t *time*) [pure virtual]

Add a timeout to the server internal timeout queue.

Parameters

timeout	The timeout object to register.
time	The time (absolute) at which the timeout shall expire.

Precondition

timeout must not be in any queue.

Returns

0 on success, 1 if timeout is already expired, < 0 on error.

Implemented in L4::Ipc_svr::Br_manager_no_buffers, L4::Ipc_svr::Timeout_queue_hooks< HOOKS, BR_MAN >, L4::Ipc_svr::Timeout_queue_hooks< Br_manager_timeout_hooks, Br_manager >, and L4Re::Util::Br_manager.

12.125.2.2 virtual int L4::Ipc_svr::Server_iface::alloc_buffer_demand (Demand const & *demand*) [pure virtual]

Tells the server to allocate buffers for the given demand.

Parameters

demand	The total server-side demand of receive buffers needed for a given interface, see Demand.

This function is not called by user applications directly. Usually the server implementation or the registry implementation calls this function whenever a new object is registered at the server.

Implemented in L4::Ipc_svr::Br_manager_no_buffers, and L4Re::Util::Br_manager.

12.125.2.3 virtual L4::Cap<void> L4::Ipc_svr::Server_iface::get_rcv_cap (int *index*) const [pure virtual]

Get capability slot allocated to the given receive buffer.

Parameters

index	The receive buffer index of the expected capability argument (0 <= `index` < `caps` registered with alloc_buffer_demand()).

Precondition

0 <= `index` < `caps` registered with alloc_buffer_demand()

Returns

Capability slot currently allocated to the given receive buffer.

Implemented in L4::Ipc_svr::Br_manager_no_buffers, and L4Re::Util::Br_manager.

Referenced by rcv_cap().

Here is the caller graph for this function:

12.125.2.4 template<typename T > L4::Cap<T> L4::Ipc_svr::Server_iface::rcv_cap (int *index*) const [inline]

Get given receive buffer as typed capability.

See also

get_rcv_cap()

Parameters

index	The receive buffer index of the expected capability argument. (0 <= `index` < `caps` registered with alloc_buffer_demand().)

Precondition

0 <= `index` < `caps` registered with alloc_buffer_demand()

Returns

Capability slot currently allocated to the given receive buffer.

Note

This is a convenience wrapper for get_rcv_cap() to avoid L4::cap_cast<>().

Definition at line 120 of file ipc_epiface.

References L4::cap_cast(), and get_rcv_cap().

Here is the call graph for this function:

12.125.2.5 L4::Cap<void> L4::Ipc_svr::Server_iface::rcv_cap (int *index*) const [inline]

Get receive cap with the given index as generic (void) type.

Parameters

index	The index of the cap receive buffer of the expected capability. (0 <= `index` < `caps` registered with alloc_buffer_demand().)

Returns

Capability slot currently allocated to the given capability buffer.

Note

This is a convenience wrapper for get_rcv_cap().

Definition at line 132 of file ipc_epiface.

References get_rcv_cap().

Here is the call graph for this function:

12.125.2.6 virtual int L4::lpc_svr::Server_iface::realloc_rcv_cap (int *index*) [pure virtual]

Allocate a new capability for the given receive buffer.

Parameters

index	The receive buffer index of the expected capability argument (0 <= `index` < `caps` registered with alloc_buffer_demand()).

Precondition

0 <= `index` < `caps` registered with alloc_buffer_demand()

Returns

0 on success, < 0 on error.

Implemented in L4::Ipc_svr::Br_manager_no_buffers, and L4Re::Util::Br_manager.

12.125.2.7 virtual int L4::Ipc_svr::Server_iface::remove_timeout (Timeout * *timeout*) [pure virtual]

Remove the given timeout from the timer queue.

Parameters

timeout	The timout object to remove.

Returns

0 on success, < 0 on error.

Implemented in L4::Ipc_svr::Br_manager_no_buffers, L4::Ipc_svr::Timeout_queue_hooks< HOOKS, BR_MAN >, L4::Ipc_svr::Timeout_queue_hooks< Br_manager_timeout_hooks, Br_manager >, and L4Re::Util::Br_manager.

The documentation for this class was generated from the following file:

- l4/sys/cxx/ipc_epiface

12.126 L4::Ipc_svr::Timed_work< HOOKS > Class Template Reference

DEPRECATED.

Inherits HOOKS.

Collaboration diagram for L4::Ipc_svr::Timed_work< HOOKS >:

12.126.1 Detailed Description

template<typename HOOKS>class L4::Ipc_svr::Timed_work< HOOKS >

DEPRECATED.

Deprecated Use L4::Ipc_svr::Timeout_queue_hooks

Definition at line 97 of file ipc_server_loop.

The documentation for this class was generated from the following file:

- l4/sys/cxx/ipc_server_loop

12.127 L4::Ipc_svr::Timeout Class Reference

Callback interface for Timeout_queue.

Inheritance diagram for L4::Ipc_svr::Timeout:

Collaboration diagram for L4::Ipc_svr::Timeout:

Public Member Functions

- Timeout ()

  Make a timeout.
- virtual ~Timeout ()=0

  Destroy a timeout.
- virtual void expired ()=0

  callback function to be called when timeout happened
- l4_kernel_clock_t timeout () const

  return absolute timeout of this callback.

12.127.1 Detailed Description

Callback interface for Timeout_queue.

Definition at line 20 of file ipc_timeout_queue.

12.127.2 Member Function Documentation

12.127.2.1 virtual void L4::Ipc_svr::Timeout::expired ( ) [pure virtual]

callback function to be called when timeout happened

Note

The timeout object is already dequeued when this function is called, this means the timeout may be safely again within the expired() function.

Referenced by L4::Ipc_svr::Timeout_queue::handle_expired_timeouts().

Here is the caller graph for this function:

12.127.2.2 l4_kernel_clock_t L4::Ipc_svr::Timeout::timeout ( ) const [inline]

return absolute timeout of this callback.

Returns

absolute timeout for this instance of the timeout.

Precondition

The timeout object must have been in a queue before, otherwise the timeout is not set.

Definition at line 43 of file ipc_timeout_queue.

The documentation for this class was generated from the following file:

- l4/cxx/ipc_timeout_queue

12.128 L4::Ipc_svr::Timeout_queue Class Reference

Timeout queue to be used in l4re server loop.

Collaboration diagram for L4::Ipc_svr::Timeout_queue:

Public Types

- typedef L4::Ipc_svr::Timeout Timeout

  Provide a local definition of Timeout for backward compat.

Public Member Functions

- Timeout_queue ()

  Creates an empty timeout queue.
- bool timeout_expired (l4_kernel_clock_t now) const

  Determine if a timeout has happened.
- l4_kernel_clock_t next_timeout () const

  Get the time for the next timeout.
- void handle_expired_timeouts (l4_kernel_clock_t now)

  run the callbacks of expired timeouts
- void add (Timeout *timeout, l4_kernel_clock_t time)

  Add a timeout to the queue.
- void remove (Timeout *timeout)

  Remove timeout from the queue.

12.128.1 Detailed Description

Timeout queue to be used in l4re server loop.

Definition at line 56 of file ipc_timeout_queue.

12.128.2 Member Function Documentation

12.128.2.1 void L4::Ipc_svr::Timeout_queue::add (Timeout * *timeout*, l4_kernel_clock_t *time*) [inline]

Add a timeout to the queue.

Parameters

timeout	timeout object to add
time	the time when the timeout expires

Precondition

timeout must not be in any queue already

Definition at line 118 of file ipc_timeout_queue.

References cxx::Bits::Basic_list< POLICY >::begin(), cxx::Bits::Basic_list< POLICY >::end(), and cxx::H_list< T, POLICY >::insert_before().

Referenced by L4::Ipc_svr::Timeout_queue_hooks< Br_manager_timeout_hooks, Br_manager >::add_timeout().

Here is the call graph for this function:

Here is the caller graph for this function:

12.128.2.2 void L4::Ipc_svr::Timeout_queue::handle_expired_timeouts (l4_kernel_clock_t *now*) [inline]

run the callbacks of expired timeouts

Parameters

now	the current time.

Definition at line 94 of file ipc_timeout_queue.

References cxx::Bits::Basic_list< POLICY >::begin(), cxx::Bits::Basic_list< POLICY >::empty(), cxx::H_list< T, POLICY >::erase(), and L4::Ipc_svr::Timeout::expired().

Referenced by L4::Ipc_svr::Timeout_queue_hooks< Br_manager_timeout_hooks, Br_manager >::setup_wait().

Here is the call graph for this function:

Here is the caller graph for this function:

12.128.2.3 l4_kernel_clock_t L4::lpc_svr::Timeout_queue::next_timeout ( ) const [inline]

Get the time for the next timeout.

Returns

the time for the next timeout or 0 if there is none

Definition at line 85 of file ipc_timeout_queue.

Referenced by L4::Ipc_svr::Timeout_queue_hooks< Br_manager_timeout_hooks, Br_manager >::timeout().

Here is the caller graph for this function:

12.128.2.4 void L4::Ipc_svr::Timeout_queue::remove (Timeout * *timeout*) [inline]

Remove *timeout* from the queue.

Parameters

timeout	timeout to remove from timeout queue

Precondition

timeout must be in this queue

Definition at line 134 of file ipc_timeout_queue.

References cxx::Bits::Basic_list< POLICY >::begin(), cxx::Bits::Basic_list< POLICY >::empty(), and cxx::H_list< T, POLICY >::remove().

Referenced by L4::Ipc_svr::Timeout_queue_hooks< Br_manager_timeout_hooks, Br_manager >::remove_timeout().

Here is the call graph for this function:

Here is the caller graph for this function:

12.128.2.5 bool L4::Ipc_svr::Timeout_queue::timeout_expired (l4_kernel_clock_t *now*) const [inline]

Determine if a timeout has happened.

Parameters

now	the current time.

Returns

true if there is at least one expired timeout in the queue, false if not.

Definition at line 73 of file ipc_timeout_queue.

References cxx::Bits::Basic_list< POLICY >::empty().

Referenced by L4::Ipc_svr::Timeout_queue_hooks< Br_manager_timeout_hooks, Br_manager >::before_reply(), and L4::Ipc_svr::Timeout_queue_hooks< Br_manager_timeout_hooks, Br_manager >::setup_wait().

Here is the call graph for this function:

Here is the caller graph for this function:

The documentation for this class was generated from the following file:

- l4/cxx/ipc_timeout_queue

12.129 L4::Ipc_svr::Timeout_queue_hooks< HOOKS, BR_MAN > Class Template Reference

Loop hooks mixin for integrating a timeout queue into the server loop.

Inheritance diagram for L4::Ipc_svr::Timeout_queue_hooks< HOOKS, BR_MAN >:

Collaboration diagram for L4::lpc_svr::Timeout_queue_hooks< HOOKS, BR_MAN >:

Public Member Functions

- l4_timeout_t timeout ()

  get the time for the next timeout
- void setup_wait (l4_utcb_t *utcb, L4::lpc_svr::Reply_mode mode)

  setup_wait() for the server loop
- L4::lpc_svr::Reply_mode before_reply (l4_msgtag_t, l4_utcb_t *)

  server loop hook
- int add_timeout (Timeout *timeout, l4_kernel_clock_t time)

  Add a timout to the queue for time time.
- int remove_timeout (Timeout *timeout)

  Remove timeout from the queue.

Data Fields

- Timeout_queue queue

  Use this timeout queue.

Additional Inherited Members

12.129.1 Detailed Description

template<typename HOOKS, typename BR_MAN = Br_manager_no_buffers>class L4::Ipc_svr::Timeout_queue_hooks< HOOKS, BR_MAN >

Loop hooks mixin for integrating a timeout queue into the server loop.

Template Parameters

HOOKS	has to inherit from Timeout_queue_hooks<> and provide the functions now() that has to return the current time.
BR_MAN	This used as a base class for and provides the API for selecting the buffer register (BR) that is used to store the timeout value. This is usually L4Re::Util::Br_manager or L4::Ipc_svr::Br_manager_no_buffers.

Definition at line 166 of file ipc_timeout_queue.

12.129.2 Member Function Documentation

12.129.2.1 template<typename HOOKS, typename BR_MAN = Br_manager_no_buffers> int L4::Ipc_svr::Timeout_queue_hooks< HOOKS, BR_MAN >::add_timeout (Timeout * *timeout*, l4_kernel_clock_t *time*) [inline], [virtual]

Add a timout to the queue for time *time*.

Parameters

timeout	The timeout object to add into the queue (must not be in any queue currently).
time	The time when the timeout shall expire.

Precondition

timeout must not be in any queue.

Note

The timeout is automatically dequeued before the Timeout::expired() function is called

Implements L4::Ipc_svr::Server_iface.

Definition at line 218 of file ipc_timeout_queue.

12.129.2.2 template<typename HOOKS, typename BR_MAN = Br_manager_no_buffers> int L4::Ipc_svr::Timeout_queue_hooks< HOOKS, BR_MAN >::remove_timeout (Timeout * *timeout*) [inline], [virtual]

Remove timeout from the queue.

Parameters

timeout	The timeout object to be removed from the queue.

Note

This function may be safely called even if the timeout is not currently enqueued.
in Timeout::expired() the timeout is already dequeued!

Implements L4::Ipc_svr::Server_iface.

Definition at line 231 of file ipc_timeout_queue.

The documentation for this class was generated from the following file:

- l4/cxx/ipc_timeout_queue

12.130 L4::Irq Class Reference

C++ Irq interface.

Inheritance diagram for L4::Irq:

Collaboration diagram for L4::Irq:

Public Member Functions

- l4_msgtag_t attach (l4_umword_t label, Cap< Thread > const &thread=Cap< Thread >::Invalid, l4_utcb_t *utcb=l4_utcb()) throw ()

  Attach a thread to this interrupt.

- l4_msgtag_t detach (l4_utcb_t *utcb=l4_utcb()) throw ()

  Detach from this interrupt.

- l4_msgtag_t receive (l4_timeout_t timeout=L4_IPC_NEVER, l4_utcb_t *utcb=l4_utcb()) throw ()

  Unmask and wait for this IRQ.
- l4_msgtag_t wait (l4_umword_t *label, l4_timeout_t timeout=L4_IPC_NEVER, l4_utcb_t *utcb=l4_utcb()) throw ()

  Unmask IRQ and (open) wait for any message.
- l4_msgtag_t unmask (l4_utcb_t *utcb=l4_utcb()) throw ()

  Unmask IRQ.

Additional Inherited Members

12.130.1 Detailed Description

C++ Irq interface.

The Irq class provides access to abstract interrupts provided by the microkernel. Interrupts may be

- hardware interrupts provided by the platform interrupt controller,
- virtual device interrupts provided by the microkernel's virtual devices (virtual serial or trace buffer) or
- virtual interrupts that can be triggered by user programs (IRQs)

Irq objects can be created using a factory, see the L4::Factory API (L4::Factory::create()).

Include File

```
#include <l4/sys/irq>
```

For the C interface refer to the IRQs API for an overview.

Examples:

examples/libs/l4re/c++/shared_ds/ds_clnt.cc.

Definition at line 105 of file irq.

12.130.2 Member Function Documentation

12.130.2.1 l4_msgtag_t L4::Irq::attach (l4_umword_t *label*, Cap< Thread > const & *thread* = Cap<Thread>::Invalid, l4_utcb_t * *utcb* = l4_utcb()) throw) [inline]

Attach a thread to this interrupt.

Parameters

label	Identifier of the IRQ (*protected label* used for messages)
thread	Capability of the thread to attach the IRQ to.
utcb	UTCB to be used for this operation, usually the UTCB of the calling thread.

Returns

Syscall return tag

Definition at line 120 of file irq.

References L4::Kobject::cap(), and l4_irq_attach_u().

Referenced by L4Re::Util::Event_buffer_consumer_t< PAYLOAD >::process().

Here is the call graph for this function:

Here is the caller graph for this function:

12.130.2.2 l4_msgtag_t L4::Irq::detach (l4_utcb_t * *utcb* = l4_utcb ()) throw) [inline]

Detach from this interrupt.

Parameters

utcb	UTCB to be used for this operation, usually the UTCB of the calling thread.

Returns

Syscall return tag

Definition at line 132 of file irq.

References L4::Kobject::cap(), and l4_irq_detach_u().

Here is the call graph for this function:

12.130.2.3 l4_msgtag_t L4::Irq::receive (l4_timeout_t *timeout* = L4_IPC_NEVER, l4_utcb_t * *utcb* = l4_utcb ()) throw) [inline]

Unmask and wait for this IRQ.

Parameters

timeout	Timeout.
utcb	UTCB to be used for this operation, usually the UTCB of the calling thread.

Returns

Syscall return tag

Note

If this is the function normally used for your IRQs consider using L4::Semaphore instead of L4::Irq.

Definition at line 145 of file irq.

References L4::Kobject::cap(), and l4_irq_receive_u().

Here is the call graph for this function:

12.130.2.4 l4_msgtag_t L4::Irq::unmask (l4_utcb_t * *utcb* = l4_utcb ()) throw) [inline]

Unmask IRQ.

Parameters

utcb	UTCB to be used for this operation, usually the UTCB of the calling thread.

Returns

Syscall return tag

Deprecated Use L4::Irq_eoi::unmask()

Note

wait() and receive() are doing the unmask already.

Definition at line 172 of file irq.

References L4_IPC_NEVER.

Referenced by wait().

Here is the caller graph for this function:

12.130.2.5 l4_msgtag_t L4::Irq::wait (l4_umword_t * *label*, l4_timeout_t *timeout* = L4_IPC_NEVER, l4_utcb_t * *utcb* = l4_utcb ()) throw) [inline]

Unmask IRQ and (open) wait for any message.

Parameters

label	The *protected label* shall be received here.
timeout	Timeout.
utcb	UTCB to be used for this operation, usually the UTCB of the calling thread.

Returns

Syscall return tag

Definition at line 158 of file irq.

References unmask().

Here is the call graph for this function:

The documentation for this class was generated from the following file:

- l4/sys/irq

12.131 L4::Irq_eoi Class Reference

Interface for sending an acknowledge message to an object.

Inheritance diagram for L4::Irq_eoi:

Collaboration diagram for L4::Irq_eoi:

Public Member Functions

- l4_msgtag_t unmask (unsigned irqnum, l4_umword_t *label=0, l4_timeout_t to=L4_IPC_NEVER, l4_utcb_t *utcb=l4_utcb()) throw ()

  Acknowledge the given interrupt line.

12.131.1 Detailed Description

Interface for sending an acknowledge message to an object.

The object is usually an ICU or an IRQ.

Definition at line 40 of file irq.

12.131.2 Member Function Documentation

12.131.2.1 l4_msgtag_t L4::Irq_eoi::unmask (unsigned *irqnum*, l4_umword_t * *label* = 0, l4_timeout_t *to* = L4_IPC_NEVER, l4_utcb_t * *utcb* = l4_utcb ()) throw) [inline]

Acknowledge the given interrupt line.

Parameters

	irqnum	The interrupt line that shall be acknowledged.
out	*label*	If NULL this is a send-only unmask, if not NULL then this operation enters an open wait and the *protected label* shall be received here.
	to	The timeout-pair (send and receive) that shall be used for this operation. The receive timeout is used with a non-NULL `label` only.
	utcb	UTCB to be used for this operation, usually the UTCB of the calling thread.

Definition at line 55 of file irq.

References L4::Kobject::cap().

Here is the call graph for this function:

The documentation for this class was generated from the following file:

- l4/sys/irq

12.132 L4::Irq_handler_object Struct Reference

Server object base class for handling IRQ messages.

Inheritance diagram for L4::Irq_handler_object:

Collaboration diagram for L4::Irq_handler_object:

Additional Inherited Members

12.132.1 Detailed Description

Server object base class for handling IRQ messages.

This server object base class implements the empty interface (L4::Kobject). The implementation of Server_object↩ ::dispatch() must return -L4_ENOREPLY, because IRQ messages do not handle replies.

Examples:

examples/libs/l4re/c++/shared_ds/ds_srv.cc.

Definition at line 173 of file ipc_server.

The documentation for this struct was generated from the following file:

- l4/cxx/ipc_server

12.133 L4::Irq_mux Struct Reference

IRQ multiplexer for shared IRQs.

Inheritance diagram for L4::Irq_mux:

Collaboration diagram for L4::Irq_mux:

Public Member Functions

- l4_msgtag_t chain (Cap< Triggerable > const &slave, l4_utcb_t *utcb=l4_utcb()) throw ()

  Attach an IRQ to this multiplexer.

Additional Inherited Members

12.133.1 Detailed Description

IRQ multiplexer for shared IRQs.

This interface allows broadcasting of shared IRQs to multiple triggerables. The IRQ multiplexer is responsible for the correct mask and unmask logic for such shared IRQs.

The semantics are that each of the slave IRQs is triggered whenever the multiplexer IRQ is triggered. As shared IRQs are usually level-triggered, the real IRQ source will be masked automatically when an IRQ is delivered and shall be unmasked when all attached slave IRQs are acknowledged.

Definition at line 189 of file irq.

12.133.2 Member Function Documentation

12.133.2.1 l4_msgtag_t L4::Irq_mux::chain (Cap< Triggerable > const & *slave*, l4_utcb_t * *utcb* = l4_utcb ()) throw) [inline]

Attach an IRQ to this multiplexer.

Parameters

slave	The slave that shall be attached to the master.
utcb	UTCB to be used for this operation, usually the UTCB of the calling thread.

Returns

Syscall return tag

The chaining feature of IRQ objects allows to deal with shared IRQs. For chaining IRQs there must be an IRQ multiplexer (Irq_mux) bound to the real IRQ source. This function allows to add slave IRQs to this multiplexer.

Definition at line 204 of file irq.

References L4::Kobject::cap(), and l4_irq_mux_chain_u().

Here is the call graph for this function:

The documentation for this struct was generated from the following file:

- l4/sys/irq

12.134 L4::Kip::Mem_desc Class Reference

Memory descriptors stored in the kernel interface page.

Collaboration diagram for L4::Kip::Mem_desc:

Public Types

- enum Mem_type

  Memory types.

Public Member Functions

- Mem_desc (unsigned long start, unsigned long end, Mem_type t, unsigned char st=0, bool virt=false) throw ()

  Initialize memory descriptor.
- unsigned long start () const throw ()

  Return start address of memory descriptor.
- unsigned long end () const throw ()

  Return end address of memory descriptor.
- unsigned long size () const throw ()

  Return size of region described by the memory descriptor.
- Mem_type type () const throw ()

  Return type of the memory descriptor.
- unsigned char sub_type () const throw ()

  Return sub-type of the memory descriptor.
- unsigned is_virtual () const throw ()

  Return whether the memory descriptor describes a virtual or physical region.
- void set (unsigned long start, unsigned long end, Mem_type t, unsigned char st=0, bool virt=false) throw ()

  Set values of a memory descriptor.

Static Public Member Functions

- static Mem_desc ∗ first (void ∗kip) throw ()

  Get first memory descriptor.

- static unsigned long count (void const ∗kip) throw ()

  Return number of memory descriptors stored in the kernel info page.

- static void count (void ∗kip, unsigned count) throw ()

  Set number of memory descriptors.

12.134.1 Detailed Description

Memory descriptors stored in the kernel interface page.

`#include <l4/sys/kip>`

Definition at line 51 of file kip.

12.134.2 Constructor & Destructor Documentation

12.134.2.1 L4::Kip::Mem_desc::Mem_desc (unsigned long *start*, unsigned long *end*, Mem_type *t*, unsigned char *st* = 0, bool *virt* = false) throw) [inline]

Initialize memory descriptor.

Parameters

start	Start address
end	End address
t	Memory type
st	Memory subtype, defaults to 0
virt	True for virtual memory, false for physical memory, defaults to physical

Definition at line 127 of file kip.

12.134.3 Member Function Documentation

12.134.3.1 static unsigned long L4::Kip::Mem_desc::count (void const ∗ *kip*) throw) [inline], [static]

Return number of memory descriptors stored in the kernel info page.

Parameters

kip	Pointer to the kernel info page

Returns

Number of memory descriptors in the kernel info page.

Definition at line 101 of file kip.

Referenced by count().

Here is the caller graph for this function:

12.134.3.2 static void L4::Kip::Mem_desc::count (void * *kip*, unsigned *count*) throw) [inline], [static]

Set number of memory descriptors.

Parameters

kip	Pointer to the kernel info page
count	Number of memory descriptors

Definition at line 112 of file kip.

References count().

Here is the call graph for this function:

12.134.3.3 unsigned long L4::Kip::Mem_desc::end ( ) const throw) [inline]

Return end address of memory descriptor.

Returns

End address of memory descriptor

Definition at line 143 of file kip.

Referenced by size().

Here is the caller graph for this function:

12.134.3.4 static Mem_desc* L4::Kip::Mem_desc::first (void * *kip*) throw) [inline], [static]

Get first memory descriptor.

Parameters

kip	Pointer to the kernel info page

Returns

First memory descriptor stored in the kernel info page

Definition at line 84 of file kip.

12.134.3.5 unsigned L4::Kip::Mem_desc::is_virtual ( ) const throw) [inline]

Return whether the memory descriptor describes a virtual or physical region.

Returns

True for virtual region, false for physical region.

Definition at line 168 of file kip.

12.134.3.6 void L4::Kip::Mem_desc::set (unsigned long *start*, unsigned long *end*, Mem_type *t*, unsigned char *st* = 0, bool *virt* = false) throw) [inline]

Set values of a memory descriptor.

Parameters

start	Start address
end	End address
t	Memory type
st	Sub-type, defaults to 0
virt	Virtual or physical memory region, defaults to physical

Definition at line 178 of file kip.

12.134.3.7 unsigned long L4::Kip::Mem_desc::size ( ) const throw) [inline]

Return size of region described by the memory descriptor.

Returns

Size of the region described by the memory descriptor

Definition at line 149 of file kip.

References end(), and start().

Here is the call graph for this function:

12.134.3.8 unsigned long L4::Kip::Mem_desc::start ( ) const throw) [inline]

Return start address of memory descriptor.

Returns

Start address of memory descriptor

Definition at line 137 of file kip.

Referenced by size().

Here is the caller graph for this function:

12.134.3.9 unsigned char L4::Kip::Mem_desc::sub_type ( ) const throw) [inline]

Return sub-type of the memory descriptor.

Returns

Sub-type of the memory descriptor

Definition at line 161 of file kip.

12.134.3.10 Mem_type L4::Kip::Mem_desc::type ( ) const throw) [inline]

Return type of the memory descriptor.

Returns

Type of the memory descriptor

Definition at line 155 of file kip.

The documentation for this class was generated from the following file:

- l4/sys/kip

12.135 L4::Kobject Class Reference

Base class for all kinds of kernel objects and remote objects, referenced by capabilities.

Inheritance diagram for L4::Kobject:

Collaboration diagram for L4::Kobject:

Public Member Functions

- l4_msgtag_t dec_refcnt (l4_mword_t diff, l4_utcb_t *utcb=l4_utcb())

  Decrement the in kernel reference counter for the object.

Protected Member Functions

- l4_cap_idx_t cap () const throw ()

  Return capability selector.

12.135.1 Detailed Description

Base class for all kinds of kernel objects and remote objects, referenced by capabilities.

`#include <l4/sys/capability>`

This is the base class for all remote objects accessible using RPC. However, subclasses doe not directly inherit from L4::Kobject but *must* use L4::Kobject_t (L4::Kobject_0t, L4::Kobject_2t, L4::Kobject_3t, or L4::Kobject_x) for inheritance, otherwise these classes cannot be used as RPC interfaces.

Attention

Objects derived from Kobject *must* never add any data to those objects. Kobjects can act only as proxy object for encapsulating object invocations.

Definition at line 43 of file kobject.

12.135.2 Member Function Documentation

12.135.2.1 l4_cap_idx_t L4::Kobject::cap ( ) const throw) [inline], [protected]

Return capability selector.

Returns

Capability selector.

This method is for derived classes to gain access to the actual capability selector.

Definition at line 76 of file kobject.

Referenced by L4::Task::add_ku_mem(), L4::Irq::attach(), L4::Icu::bind(), L4::Task::cap_equal(), L4::Task::cap_has_child(), L4::Task::cap_valid(), L4::Irq_mux::chain(), L4::Thread::control(), L4::Factory::create(), L4::Factory::create_factory(), L4::Factory::create_gate(), L4::Factory::create_irq(), L4::Factory::create_task(), L4::Factory::create_thread(), L4::Factory::create_vm(), L4::Task::delete_obj(), L4::Irq::detach(), L4::Semaphore::down(), L4::Thread::ex_regs(), L4::Vcon::get_attr(), L4::Debugger::get_object_name(), L4::Debugger::global_id(), L4::Icu::info(), L4::Debugger::kobj_to_id(), L4::Task::map(), L4::Icu::mask(), L4::Thread::modify_senders(), L4::Debugger::query_log_name(), L4::Debugger::query_log_typeid(), L4::Vcon::read(), L4::Vcon::read_with_flags(), L4::Irq::receive(), L4::Thread::register_del_irq(), L4::Task::release_cap(), L4::Vcon::send(), L4::Vcon::set_attr(), L4::Icu::set_mode(), L4::Debugger::set_object_name(), L4::Thread::stats_time(), L4::Debugger::switch_log(), L4::Thread::switch_to(), L4::Triggerable::trigger(), L4::Icu::unbind(), L4::Task::unmap(), L4::Task::unmap_batch(), L4::Irq_eoi::unmask(), L4::Thread::vcpu_control(), L4::Thread::vcpu_control_ext(), L4::Thread::vcpu_resume_commit(), and L4::Vcon::write().

12.135.2.2 l4_msgtag_t L4::Kobject::dec_refcnt (l4_mword_t *diff*, l4_utcb_t * *utcb* = l4_utcb ()) [inline]

Decrement the in kernel reference counter for the object.

Parameters

diff	is the delta that shall be subtracted from the reference count.
utcb	is the utcb to use for the invocation.

This function is intended for servers to be able to remove the servers own capability from the counted references. This leads to the semantics that the kernel will delete the object even if the capability of the server is valid. The server can detect the deletion by polling its capabilities or by using the IPC-gate deletion IRQs. And to cleanup if the clients dropped the last reference (capability) to the object.

Definition at line 101 of file kobject.

The documentation for this class was generated from the following file:

- l4/sys/kobject

12.136 L4::Kobject_2t< Derived, Base1, Base2, PROTO, S_DEMAND > Class Template Reference

Helper class to create an L4Re interface class that is derived from two base classes (see `L4::Kobject_t`).

`#include <l4/sys/capability>`

Inheritance diagram for L4::Kobject_2t< Derived, Base1, Base2, PROTO, S_DEMAND >:

Collaboration diagram for L4::Kobject_2t< Derived, Base1, Base2, PROTO, S_DEMAND >:

Protected Types

- typedef Derived Class

  The target interface type (inheriting from Kobject_t)
- typedef Typeid::Iface< PROTO, Derived > __Iface

  The interface description for the derived class.
- typedef Typeid::Merge_list < Typeid::Iface_list< __Iface > , Typeid::Merge_list< typename Base1::__Iface_list, typename Base2::__Iface_list > > __Iface_list

  The list of all RPC interfaces provided directly or through inheritance.

Protected Member Functions

- L4::Cap< Class > c () const

  Get the capability to ourselves.

Static Protected Member Functions

- static void __check_protocols__ ()

12.136.1 Detailed Description

template<typename Derived, typename Base1, typename Base2, long PROTO = PROTO_ANY, typename S_DEMAND = Type_↩info::Demand_t<>>class L4::Kobject_2t< Derived, Base1, Base2, PROTO, S_DEMAND >

Helper class to create an L4Re interface class that is derived from two base classes (see `L4::Kobject_t`).

Template Parameters

Derived	is the name of the new interface.
Base1	is the name of the interface's first base class.
Base2	is the name of the interface's second base class.
PROTO	may be set to the statically assigned protocol number used to communicate with this interface.
S_DEMAND	type defining the demand of server-side resources for this interface, usually a L4::Type_info::Demand_t. This value must describe the server-side resources needed by the interface itself, the resource demand of the base interfaces (Base1 and Base2) are automatically included.

The typical usage pattern is shown in the following code snippet. The semantics of this example is an interface My_iface that is derived from L4::Icu and L4Re::Dataspace.

```
class My_iface : public L4::Kobject_2t<My_iface, L4::Icu, L4Re::Dataspace>
{
  ...
};
```

Definition at line 829 of file __typeinfo.h.

12.136.2 Member Typedef Documentation

12.136.2.1 template<typename Derived, typename Base1, typename Base2, long PROTO = PROTO_ANY, typename S_DEMAND = Type_info::Demand_t>> typedef Typeid::Iface<PROTO, Derived> L4::Kobject_2t< Derived, Base1, Base2, PROTO, S_DEMAND >::__Iface [protected]

The interface description for the derived class.

Definition at line 835 of file __typeinfo.h.

12.136.2.2 template<typename Derived, typename Base1, typename Base2, long PROTO = PROTO_ANY, typename S_DEMAND = Type_info::Demand_t>> typedef Typeid::Merge_list< Typeid::Iface_list<__Iface>, Typeid::Merge_list< typename Base1::__Iface_list, typename Base2::__Iface_list > > L4::Kobject_2t< Derived, Base1, Base2, PROTO, S_DEMAND >::__Iface_list [protected]

The list of all RPC interfaces provided directly or through inheritance.

Definition at line 843 of file __typeinfo.h.

12.136.2.3 template<typename Derived, typename Base1, typename Base2, long PROTO = PROTO_ANY, typename S_DEMAND = Type_info::Demand_t>> typedef Derived L4::Kobject_2t< Derived, Base1, Base2, PROTO, S_DEMAND >::Class [protected]

The target interface type (inheriting from Kobject_t)

Definition at line 833 of file __typeinfo.h.

12.136.3 Member Function Documentation

12.136.3.1 template<typename Derived, typename Base1, typename Base2, long PROTO = PROTO_ANY, typename S_DEMAND = Type_info::Demand_t>> static void L4::Kobject_2t< Derived, Base1, Base2, PROTO, S_DEMAND >::__check_protocols__ ( ) [inline], [static], [protected]

Definition at line 846 of file __typeinfo.h.

12.136.3.2 template<typename Derived, typename Base1, typename Base2, long PROTO = PROTO_ANY, typename S_DEMAND = Type_info::Demand_t<>> L4::Cap<Class> L4::Kobject_2t< Derived, Base1, Base2, PROTO, S_DEMAND >::c ( ) const [inline], [protected]

Get the capability to ourselves.

Definition at line 865 of file __typeinfo.h.

The documentation for this class was generated from the following file:

- l4/sys/__typeinfo.h

12.137 L4::Kobject_3t< Derived, Base1, Base2, Base3, PROTO, S_DEMAND > Struct Template Reference

Helper class to create an L4Re interface class that is derived from three base classes (see `L4::Kobject_t`).

`#include <l4/sys/capability>`

Inheritance diagram for L4::Kobject_3t< Derived, Base1, Base2, Base3, PROTO, S_DEMAND >:

Collaboration diagram for L4::Kobject_3t< Derived, Base1, Base2, Base3, PROTO, S_DEMAND >:

Protected Types

- typedef Derived Class

  The target interface type (inheriting from Kobject_t)
- typedef Typeid::Iface< PROTO, Derived > __Iface

  The interface description for the derived class.
- typedef Typeid::Merge_list < Typeid::Iface_list< __Iface > , Typeid::Merge_list< typename Base1::__Iface_list, Typeid::Merge_list< typename Base2::__Iface_list, typename Base3::__Iface_list > > > __Iface_list

  The list of all RPC interfaces provided directly or through inheritance.

Protected Member Functions

- L4::Cap< Class > c () const

  Get the capability to ourselves.

Static Protected Member Functions

- static void __check_protocols__ ()

12.137.1 Detailed Description

template<typename Derived, typename Base1, typename Base2, typename Base3, long PROTO = PROTO_ANY, typename S_D↩EMAND = Type_info::Demand_t<>>struct L4::Kobject_3t< Derived, Base1, Base2, Base3, PROTO, S_DEMAND >

Helper class to create an L4Re interface class that is derived from three base classes (see `L4::Kobject_t`).

Template Parameters

Derived	is the name of the new interface.
Base1	is the name of the interface's first base class.
Base2	is the name of the interface's second base class.
Base3	is the name of the interfaces third base class.
PROTO	may be set to the statically assigned protocol number used to communicate with this interface.
S_DEMAND	type defining the demand on server-side resources for this interface, usually a L4::Type_info::Demand_t. This value must describe the server-side resources needed by the interface itself, the resource demand of the base interfaces (Base1 and Base2) are automatically included.

See also

L4::Kobject_t, L4::Kobject_2t, L4::Kobject_0t, L4::Kobject_x

Definition at line 929 of file __typeinfo.h.

12.137.2 Member Typedef Documentation

12.137.2.1 template<typename Derived, typename Base1, typename Base2, typename Base3, long PROTO = PROTO_ANY, typename S_DEMAND = Type_info::Demand_t<>> typedef Typeid::Iface<PROTO, Derived> L4::Kobject_3t< Derived, Base1, Base2, Base3, PROTO, S_DEMAND >::__Iface [protected]

The interface description for the derived class.

Definition at line 935 of file __typeinfo.h.

12.137.2.2 template<typename Derived, typename Base1, typename Base2, typename Base3, long PROTO = PROTO_ANY, typename S_DEMAND = Type_info::Demand_t<>> typedef Typeid::Merge_list< Typeid::Iface_list<__Iface>, Typeid::Merge_list< typename Base1::__Iface_list, Typeid::Merge_list< typename Base2::__Iface_list, typename Base3::__Iface_list > > > L4::Kobject_3t< Derived, Base1, Base2, Base3, PROTO, S_DEMAND >::__Iface_list [protected]

The list of all RPC interfaces provided directly or through inheritance.

Definition at line 946 of file __typeinfo.h.

12.137.2.3 template<typename Derived, typename Base1, typename Base2, typename Base3, long PROTO = PROTO_ANY, typename S_DEMAND = Type_info::Demand_t<>> typedef Derived L4::Kobject_3t< Derived, Base1, Base2, Base3, PROTO, S_DEMAND >::Class [protected]

The target interface type (inheriting from Kobject_t)

Definition at line 933 of file __typeinfo.h.

12.137.3 Member Function Documentation

12.137.3.1 template<typename Derived, typename Base1, typename Base2, typename Base3, long PROTO = PROTO_ANY, typename S_DEMAND = Type_info::Demand_t<>> static void L4::Kobject_3t< Derived, Base1, Base2, Base3, PROTO, S_DEMAND >::__check_protocols__ ( ) [inline], [static], [protected]

Definition at line 949 of file __typeinfo.h.

12.137.3.2 template<typename Derived, typename Base1, typename Base2, typename Base3, long PROTO = PROTO_ANY, typename S_DEMAND = Type_info::Demand_t<>> L4::Cap<Class> L4::Kobject_3t< Derived, Base1, Base2, Base3, PROTO, S_DEMAND >::c ( ) const [inline], [protected]

Get the capability to ourselves.

Definition at line 977 of file __typeinfo.h.

The documentation for this struct was generated from the following file:

- l4/sys/__typeinfo.h

12.138 L4::Kobject_demand< T > Struct Template Reference

Get the combined server-side resource requirements for all type T...

```
#include <l4/sys/capability>
```

Collaboration diagram for L4::Kobject_demand< T >:

12.138.1 Detailed Description

template<typename... T>struct L4::Kobject_demand< T >

Get the combined server-side resource requirements for all type T...

Template Parameters

T	List of IPC interface types for which the combined server-side resource requirements shall be calculated.

Definition at line 1027 of file __typeinfo.h.

The documentation for this struct was generated from the following file:

- l4/sys/__typeinfo.h

12.139 L4::Kobject_t< Derived, Base, PROTO, S_DEMAND > Class Template Reference

Helper class to create an L4Re interface class that is derived from a single base class.

```
#include <l4/sys/capability>
```

Inheritance diagram for L4::Kobject_t< Derived, Base, PROTO, S_DEMAND >:

Collaboration diagram for L4::Kobject_t< Derived, Base, PROTO, S_DEMAND >:

Protected Types

- typedef Derived Class

  The target interface type (inheriting from Kobject_t)
- typedef Typeid::Iface< PROTO, Derived > __Iface

  The interface description for the derived class.
- typedef Typeid::Merge_list < Typeid::Iface_list< __Iface > , typename Base::__Iface_list > __Iface_list

  The list of all RPC interfaces provided directly or through inheritance.

Protected Member Functions

- L4::Cap< Class > c () const

  Get the capability to ourselves.

Static Protected Member Functions

- static void __check_protocols__ ()

  Helper to check for protocol conflicts.

12.139.1 Detailed Description

template<typename Derived, typename Base, long PROTO = PROTO_ANY, typename S_DEMAND = Type_info::Demand_t<>>class L4::Kobject_t< Derived, Base, PROTO, S_DEMAND >

Helper class to create an L4Re interface class that is derived from a single base class.

Template Parameters

Derived	is the name of the new interface.
Base	is the name of the interfaces single base class.
PROTO	may be set to the statically assigned protocol number used to communicate with this interface.
S_DEMAND	type defining the demand on server-side resources for this interface, usually a L4::Type_info::Demand_t. This value must describe the server-side resources needed by the interface itself, the resource demand of the base interface *Base* is automatically included.

The typical usage pattern is shown in the following code snippet. The semantics of this example is an interface My_iface that is derived from L4::Kobject.

```
class My_iface : public L4::Kobject_t<My_iface, L4::Kobject>
{
  ...
};
```

Definition at line 753 of file __typeinfo.h.

The documentation for this class was generated from the following file:

- l4/sys/__typeinfo.h

12.140 L4::Kobject_typeid< T > Struct Template Reference

Meta object for handling access to type information of Kobjects.

`#include <__typeinfo.h>`

Collaboration diagram for L4::Kobject_typeid< T >:

Public Types

- typedef T::__Kobject_typeid::Demand Demand

  Data type expressing the static demand of receive buffers in a server.

Static Public Member Functions

- static Type_info const * id ()

  Get a pointer to teh Kobject type information of T.
- static Type_info::Demand demand ()

Get the receive-buffer demand for the server providing the interface T.

- template<typename THIS , typename A1 , typename A2 >
  static int proto_dispatch (THIS *self, long proto, A1 a1, A2 &a2)

  Protocol based server-side dispatch function.

12.140.1 Detailed Description

template<typename T>struct L4::Kobject_typeid< T >

Meta object for handling access to type information of Kobjects.

Template Parameters

T	The data type derived from Kobject, usually using Kobject_t.

Definition at line 620 of file __typeinfo.h.

12.140.2 Member Typedef Documentation

12.140.2.1 template<typename T> typedef T::__Kobject_typeid::Demand L4::Kobject_typeid< T >::Demand

Data type expressing the static demand of receive buffers in a server.

This information is the combined demand of all base interfaces for T and the buffer demand of T itself. The buffer demand of T is usually specified as the S_DEMAND argument of the Kobject_t or Kobject_2t inheritance helpers. S_DEMAND is usually of type L4::Type_info::Demand_t, or L4::Type_info::Demand_union_t.

Definition at line 632 of file __typeinfo.h.

12.140.3 Member Function Documentation

12.140.3.1 template<typename T> static Type_info::Demand L4::Kobject_typeid< T >::demand ( ) [inline], [static]

Get the receive-buffer demand for the server providing the interface T.

Returns

A demand value describing the minimum receive buffers needed for handling server side requests for interface T.

Definition at line 649 of file __typeinfo.h.

12.140.3.2 template<typename T> static Type_info const* L4::Kobject_typeid< T >::id ( ) [inline], [static]

Get a pointer to teh Kobject type information of T.

Returns

a pointer to the Kobject typeinfor of T.

Definition at line 640 of file __typeinfo.h.

Referenced by L4::kobject_typeid().

Here is the caller graph for this function:

12.140.3.3 template<typename T> template<typename THIS , typename A1 , typename A2 > static int L4::Kobject_typeid< T >::proto_dispatch (THIS * *self*, long *proto*, A1 *a1*, A2 & *a2*) [inline], [static]

Protocol based server-side dispatch function.

Template Parameters

THIS	Data type of the server-side object implementing the interface T.
A1	Data type of second argument for p_dispatch()
A2	Data type of third argument for p_dispatch()

Parameters

self	The pointer to the server object
proto	The protocol number used by the caller
a1	The second argument passed to self->p_dispatch()
a2	The third argument passed to self->p_dispatch()

This function forwards the call to the overloaded p_dispatch() function of self. The data type of the first argument for p_dispatch is determined by the given protocol number.

Definition at line 670 of file __typeinfo.h.

Referenced by L4::Server_object_t< Kobject >::proto_dispatch().

Here is the caller graph for this function:

The documentation for this struct was generated from the following file:

- l4/sys/__typeinfo.h

12.141 L4::Kobject_x< Derived, ARGS > Struct Template Reference

Generic Kobject inheritance template.

```
#include <l4/sys/capability>
```

Inheritance diagram for L4::Kobject_x< Derived, ARGS >:

Collaboration diagram for L4::Kobject_x< Derived, ARGS >:

12.141.1 Detailed Description

template<typename Derived, typename... ARGS>struct L4::Kobject_x< Derived, ARGS >

Generic Kobject inheritance template.

Template Parameters

Derived	The class name that derives from Kobject_x.
ARGS	An optional protocol number via L4::Proto_t, followed by an optional server-side requirement passed as L4::Type_info::Demand_t, followed by the list of base classes.

Definition at line 1189 of file __typeinfo.h.

The documentation for this struct was generated from the following file:

- l4/sys/__typeinfo.h

12.142 L4::Out_of_memory Class Reference

Exception signalling insufficient memory.

```
#include <l4/cxx/exceptions>
```

Inheritance diagram for L4::Out_of_memory:

Collaboration diagram for L4::Out_of_memory:

Public Member Functions

- Out_of_memory (char const *extra="") throw ()

  Create an out-of-memory exception.
- ~Out_of_memory () throw ()

  Destruction.

Additional Inherited Members

12.142.1 Detailed Description

Exception signalling insufficient memory.

Definition at line 188 of file exceptions.

The documentation for this class was generated from the following file:

- l4/cxx/exceptions

12.143 L4::Platform_control Class Reference

L4 C++ interface for controlling platform-wide properties.

Inheritance diagram for L4::Platform_control:

Collaboration diagram for L4::Platform_control:

Public Types

- enum Opcode { Suspend = L4_PLATFORM_CTL_SYS_SUSPEND_OP, Shutdown = L4_PLATFORM_CT↩ L_SYS_SHUTDOWN_OP, Cpu_enable = L4_PLATFORM_CTL_CPU_ENABLE_OP, Cpu_disable = L4_P↩ LATFORM_CTL_CPU_DISABLE_OP }

  Opcodes for platform-control object.

Public Member Functions

- l4_msgtag_t system_suspend (l4_umword_t extras)

  Enter suspend to RAM.
- l4_msgtag_t system_shutdown (l4_umword_t reboot)

  Shutdown/Reboot the system.
- l4_msgtag_t cpu_enable (l4_umword_t phys_id)

  Enable an offline CPU.
- l4_msgtag_t cpu_disable (l4_umword_t phys_id)

  Disable an online CPU.

Additional Inherited Members

12.143.1 Detailed Description

L4 C++ interface for controlling platform-wide properties.

Add

```
#include <l4/sys/platform_control>
```

to your code to use the platform control functions. The API allows a client to suspend, reboot or shutdown the system.

For the C interface refer to the Platform Control C API.

Definition at line 46 of file platform_control.

12.143.2 Member Enumeration Documentation

12.143.2.1 enum L4::Platform_control::Opcode

Opcodes for platform-control object.

Enumerator

Suspend Opcode for suspend to RAM.

Shutdown Opcode for shutdown / reboot.

Cpu_enable Opcode to enable a CPU.

Cpu_disable Opcode to disable a CPU.

Definition at line 51 of file platform_control.

12.143.3 Member Function Documentation

12.143.3.1 l4_msgtag_t L4::Platform_control::cpu_disable (l4_umword_t *phys_id*)

Disable an online CPU.

Parameters

phys_id	Physical CPU id of CPU (e.g. local APIC id) to disable.

Returns

System call message tag

12.143.3.2 l4_msgtag_t L4::Platform_control::cpu_enable (l4_umword_t *phys_id*)

Enable an offline CPU.

Parameters

phys_id	Physical CPU id of CPU (e.g. local APIC id) to enable.

Returns

System call message tag

12.143.3.3 l4_msgtag_t L4::Platform_control::system_shutdown (l4_umword_t *reboot*)

Shutdown/Reboot the system.

Parameters

reboot	1 for reboot, 0 for power off

12.143.3.4 l4_msgtag_t L4::Platform_control::system_suspend (l4_umword_t *extras*)

Enter suspend to RAM.

Parameters

extras	some extra platform-specific information needed to enter suspend to RAM.

The documentation for this class was generated from the following file:

- l4/sys/platform_control

12.144 L4::Poll_timeout_kipclock Class Reference

A polling timeout based on the L4Re clock.

Collaboration diagram for L4::Poll_timeout_kipclock:

Public Member Functions

- Poll_timeout_kipclock (unsigned poll_time_us)

  Initialise relative timeout in microseconds.
- void set (unsigned poll_time_us)

  (Re-)Set relative timeout in microseconds
- bool test (bool expression=true)

  Test whether timeout has expired.
- bool timed_out () const

  Query whether timeout has expired.

12.144.1 Detailed Description

A polling timeout based on the L4Re clock.

This class allows to conveniently add a timeout to a polling loop.

The original

```
while (device.read(State) & Busy)
  ;
```

is converted to

```
Poll_timeout_kipclock timeout(10000);
while (timeout.test(device.read(State) & Busy))
  ;
if (timeout.timed_out())
  printf("ERROR: Device does not respond.\n");
```

Definition at line 37 of file poll_timeout_kipclock.

12.144.2 Constructor & Destructor Documentation

12.144.2.1 L4::Poll_timeout_kipclock::Poll_timeout_kipclock (unsigned *poll_time_us*) [inline]

Initialise relative timeout in microseconds.

Parameters

poll_time_us	Polling timeout in microseconds.

Definition at line 44 of file poll_timeout_kipclock.

References set().

Here is the call graph for this function:

12.144.3 Member Function Documentation

12.144.3.1 void L4::Poll_timeout_kipclock::set (unsigned *poll_time_us*) [inline]

(Re-)Set relative timeout in microseconds

Parameters

poll_time_us	Polling timeout in microseconds.

Definition at line 53 of file poll_timeout_kipclock.

References l4_kip_clock(), and l4re_kip().

Referenced by Poll_timeout_kipclock().

Here is the call graph for this function:

Here is the caller graph for this function:

12.144.3.2 bool L4::Poll_timeout_kipclock::test (bool *expression* = `true`) `[inline]`

Test whether timeout has expired.

Parameters

expression	Optional expression.

Return values

false	The timeout has expired or the given expression returned false.
true	The timeout has not expired and the optionally given expression returns true.

Definition at line 67 of file poll_timeout_kipclock.

References l4_kip_clock(), and l4re_kip().

Here is the call graph for this function:

12.144.3.3 bool L4::Poll_timeout_kipclock::timed_out ( ) const [inline]

Query whether timeout has expired.

Returns

Expiry state of timeout

Definition at line 79 of file poll_timeout_kipclock.

The documentation for this class was generated from the following file:

- l4/re/util/poll_timeout_kipclock

12.145 L4::Proto_t< P > Struct Template Reference

Data type for defining protocol numbers.

```
#include <__typeinfo.h>
```

Collaboration diagram for L4::Proto_t< P >:

12.145.1 Detailed Description

template<long P = PROTO_EMPTY>struct L4::Proto_t< P >

Data type for defining protocol numbers.

Template Parameters

P	The protocol number itself

This type must be used when specifying a protocol number with Kobject_x.

Definition at line 1173 of file __typeinfo.h.

The documentation for this struct was generated from the following file:

- l4/sys/__typeinfo.h

12.146 L4::Runtime_error Class Reference

Exception for an abstract runtime error.

```
#include <l4/cxx/exceptions>
```

Inheritance diagram for L4::Runtime_error:

Collaboration diagram for L4::Runtime_error:

Public Member Functions

- Runtime_error (long err_no, char const ∗extra=0) throw ()

  Create a new Runtime_error.
- char const ∗ str () const throw ()

  Return a human readable string for the exception.
- char const ∗ extra_str () const

  Get the description text for this runtime error.
- long err_no () const throw ()

  Get the error value for this runtime error.

Additional Inherited Members

12.146.1 Detailed Description

Exception for an abstract runtime error.

This is the base class for a set of exceptions that cover all errors that have a C error value (see l4_error_code_t).

Definition at line 139 of file exceptions.

12.146.2 Constructor & Destructor Documentation

12.146.2.1 L4::Runtime_error::Runtime_error (long *err_no*, char const * *extra* = 0) throw) [inline], [explicit]

Create a new Runtime_error.

Parameters

err_no	Error value for this runtime error.
extra	Description of what was happening while the error occured.

Definition at line 152 of file exceptions.

12.146.3 Member Function Documentation

12.146.3.1 long L4::Runtime_error::err_no ( ) const throw) [inline]

Get the error value for this runtime error.

Returns

Error value.

Definition at line 181 of file exceptions.

12.146.3.2 char const* L4::Runtime_error::extra_str ( ) const [inline]

Get the description text for this runtime error.

Returns

Pointer to the description string.

Definition at line 173 of file exceptions.

The documentation for this class was generated from the following file:

- l4/cxx/exceptions

12.147 L4::Scheduler Class Reference

Scheduler object.

Inheritance diagram for L4::Scheduler:

Collaboration diagram for L4::Scheduler:

Public Member Functions

- l4_msgtag_t info (l4_umword_t *cpu_max, l4_sched_cpu_set_t *cpus, l4_utcb_t *utcb=l4_utcb()) const throw ()

  ABI function for 'info' call.

- l4_msgtag_t run_thread (Ipc::Cap< Thread > thread, l4_sched_param_t const &sp)

  Run a thread on a Scheduler.

- l4_msgtag_t idle_time (l4_sched_cpu_set_t const &cpus, l4_kernel_clock_t *us)

  Query the idle time (in µs) of a CPU.

- bool is_online (l4_umword_t cpu, l4_utcb_t *utcb=l4_utcb()) const throw ()

  Query if a CPU is online.

Additional Inherited Members

12.147.1 Detailed Description

Scheduler object.

```
#include <l4/sys/scheduler>
```

See also

Scheduler for an overview description.

Definition at line 42 of file scheduler.

The documentation for this class was generated from the following file:

- l4/sys/scheduler

12.148 L4::Semaphore Struct Reference

Kernel-provided semaphore object.

Inheritance diagram for L4::Semaphore:

Collaboration diagram for L4::Semaphore:

Public Member Functions

- l4_msgtag_t up (l4_utcb_t *utcb=l4_utcb()) throw ()

  Semaphore up operation (wrapper for trigger()).

- l4_msgtag_t down (l4_timeout_t timeout=L4_IPC_NEVER, l4_utcb_t *utcb=l4_utcb()) throw ()

  Semaphore down operation.

Additional Inherited Members

12.148.1 Detailed Description

Kernel-provided semaphore object.

This is the interface for kernel-provided semaphore objects. The object provides the classical functions `up()` and `down()` for counting the semaphore and blocking. The semaphore is a Triggerable with respect to the `up()` function, this means that a semaphore can be bound to an interrupt line at an ICU (L4::Icu) and incoming interrupts increment the semaphore counter.

The `down()` method decrements the semaphore counter and blocks if the counter is already zero. Blocking on a semaphore may—as all blocking operations—either return sucessfully, or be aborted due to an expired timeout provided to the `down()` operation, or due to an L4::Thread::ex_regs() operation with the L4_THREAD_EX_REG↩S_CANCEL flag set.

The main reason for using a semaphore instead of an L4::Irq is to ensure that incoming trigger signals do not interfere with any open-wait operations, as used for example in a server loop.

Definition at line 47 of file semaphore.

12.148.2 Member Function Documentation

12.148.2.1 l4_msgtag_t L4::Semaphore::down (l4_timeout_t *timeout* = L4_IPC_NEVER, l4_utcb_t * *utcb* = l4_utcb ()) throw) [inline]

Semaphore down operation.

Parameters

timeout	Timeout for blocking the semaphore down operation. Note: The receive timeout of this timeout-pair is significant for blocking, the send part is usually non-blocking.
utcb	UTCB to be used for this operation, usually the UTCB of the calling thread.

Returns

IPC message return tag (use l4_ipc_error() to check for a timeout or a cancel condition).

This method decrements the semaphore counter by one, or blocks if the counter is already zero, until either a timeout or cancel condition hits or the counter is increased by an `up()` operation.

Definition at line 75 of file semaphore.

References L4::Kobject::cap().

Here is the call graph for this function:

12.148.2.2 l4_msgtag_t L4::Semaphore::up (l4_utcb_t * *utcb* = l4_utcb ()) throw) [inline]

Semaphore up operation (wrapper for trigger()).

Parameters

utcb	UTCB to be used for this operation, usually the UTCB of the calling thread.

Returns

Send-only IPC message return tag.

Increases the semaphore counter by one.

Definition at line 57 of file semaphore.

References L4::Triggerable::trigger().

Here is the call graph for this function:

The documentation for this struct was generated from the following file:

- l4/sys/semaphore

12.149 L4::Server< LOOP_HOOKS > Class Template Reference

Basic server loop for handling client requests.

Inheritance diagram for L4::Server< LOOP_HOOKS >:

Collaboration diagram for L4::Server< LOOP_HOOKS >:

Public Member Functions

- Server (l4_utcb_t *utcb)

  Initializes the server loop.

- template<typename DISPATCH >
  L4_NORETURN void internal_loop (DISPATCH dispatch)

The server loop.

- template<typename R >
  L4_NORETURN void loop_noexc (R r)

  Server loop without exception handling.

- template<typename EXC , typename R >
  L4_NORETURN void loop (R r)

  Server loop with internal exception handling.

Protected Member Functions

- l4_msgtag_t reply_n_wait (l4_msgtag_t reply, l4_umword_t *p)

  Internal implementation for reply and wait.

12.149.1 Detailed Description

template<typename LOOP_HOOKS = lpc_svr::Default_loop_hooks>class L4::Server< LOOP_HOOKS >

Basic server loop for handling client requests.

Parameters

LOOP_HOOKS	the server inherits from LOOP_HOOKS and calls the hooks defined in LOOP_HOOKS in the server loop. See lpc_svr::Default_loop_hooks, lpc_svr::Ignore_errors, lpc_svr::Default_timeout, lpc_svr::Compound_reply, and lpc_svr::Br_manager_no_buffers.

This is basically a simple server loop that uses a single message buffer for receiving requests and sending replies. The dispatcher determines how incoming messages are handled.

Definition at line 290 of file ipc_server_loop.

12.149.2 Constructor & Destructor Documentation

12.149.2.1 template<typename LOOP_HOOKS = lpc_svr::Default_loop_hooks> L4::Server< LOOP_HOOKS >::Server (l4_utcb_t * *utcb*) [inline], [explicit]

Initializes the server loop.

Parameters

utcb	The UTCB of the thread running the server loop.

Definition at line 298 of file ipc_server_loop.

12.149.3 Member Function Documentation

12.149.3.1 template<typename L > template<typename DISPATCH > L4_NORETURN void L4::Server< L >::internal_loop (DISPATCH *dispatch*) [inline]

The server loop.

This function usually never returns, it waits for incoming messages calls the dispatcher, sends a reply and waits again.

Definition at line 360 of file ipc_server_loop.

References l4_msgtag_t::has_error(), L4_ENOREPLY, and l4_msgtag().

Referenced by L4::Server< LOOP_HOOKS >::loop(), and L4::Server< LOOP_HOOKS >::loop_noexc().

Here is the call graph for this function:

Here is the caller graph for this function:

12.149.3.2 template<typename LOOP_HOOKS = Ipc_svr::Default_loop_hooks> template<typename EXC , typename R > L4_NORETURN void L4::Server< LOOP_HOOKS >::loop (R *r*) [inline]

Server loop with internal exception handling.

This server loop translates L4::Runtime_error exceptions into negative error return codes sent to the caller.

Definition at line 323 of file ipc_server_loop.

References L4::Server< LOOP_HOOKS >::internal_loop().

Here is the call graph for this function:

The documentation for this class was generated from the following file:

- l4/sys/cxx/ipc_server_loop

12.150 L4::Server_object Class Reference

Abstract server object to be used with L4::Server and L4::Basic_registry.

Inheritance diagram for L4::Server_object:

Collaboration diagram for L4::Server_object:

Public Member Functions

- virtual int dispatch (unsigned long rights, Ipc::Iostream &ios)=0

  The abstract handler for client requests to the object.

12.150.1 Detailed Description

Abstract server object to be used with L4::Server and L4::Basic_registry.

Note

Usually L4::Server_object_t is used as a base class when writing server objects.

This server object provides an abstract interface that is used by the L4::Registry_dispatcher model. You can derive subclasses from this interface and implement application specific server objects.

Definition at line 50 of file ipc_server.

12.150.2 Member Function Documentation

12.150.2.1 virtual int L4::Server_object::dispatch (unsigned long *rights*, Ipc::Iostream & *ios*) [pure virtual]

The abstract handler for client requests to the object.

Parameters

rights	The rights bits in the invoked capability.
ios	The Ipc::Iostream for reading the request and writing the reply.

Return values

-L4_ENOREPLY	No reply message is send.
<0	Error, reply with error code.
>=0	Success, reply with return value.

This function must be implemented by application specific server objects. The implementation must unmarshall data from the stream (`ios`) and create a reply by marshalling to the stream (`ios`). For details about the IPC stream see IPC stream operators.

Note

You need to extract the complete message from the *ios* stream before inserting any reply data or before doing any function call that may use the UTCB. Otherwise, the incoming message may get lost.

Implemented in L4::Server_object_x< Derived, IFACE, BASE >.

The documentation for this class was generated from the following file:

- l4/cxx/ipc_server

12.151 L4::Server_object_t< IFACE, BASE > Struct Template Reference

Base class (template) for server implementing server objects.

Inheritance diagram for L4::Server_object_t< IFACE, BASE >:

Collaboration diagram for L4::Server_object_t< IFACE, BASE >:

Public Types

- typedef IFACE Interface

  Data type of the IPC interface definition.

Public Member Functions

- BASE::Demand get_buffer_demand () const
- int dispatch_meta_request (L4::Ipc::Iostream &ios)

  Implementation of the meta protocol based on IFACE.

Static Public Member Functions

- template<typename THIS >
  static int proto_dispatch (THIS *self, l4_umword_t rights, L4::Ipc::Iostream &ios)

  Implementation of protocol-based dispatch for this server object.

12.151.1 Detailed Description

template<typename IFACE, typename BASE = L4::Server_object>struct L4::Server_object_t< IFACE, BASE >

Base class (template) for server implementing server objects.

Template Parameters

IFACE	The IPC interface class that defines the interface that shall be implemented.
BASE	The server object base class (usually L4::Server_object).

Examples:

examples/libs/l4re/c++/shared_ds/ds_srv.cc, and examples/libs/l4re/streammap/server.cc.

Definition at line 92 of file ipc_server.

12.151.2 Member Function Documentation

12.151.2.1 template<typename IFACE, typename BASE = L4::Server_object> int L4::Server_object_t< IFACE, BASE >::dispatch_meta_request (L4::Ipc::Iostream & *ios*) [inline]

Implementation of the meta protocol based on *IFACE*.

Parameters

ios	The IO stream used for receiving the message.

This function can be used to handle incoming L4_PROTO_META protcol requests. The implementation uses the L4::Type_info of *IFACE* to handle the requests. Call this function in the implementation of Server_object::dispatch() when the received message tag has protocol L4_PROTO_META (L4::Meta::Protocol).

Definition at line 111 of file ipc_server.

12.151.2.2 template<typename IFACE, typename BASE = L4::Server_object> BASE::Demand L4::Server_object_t< IFACE, BASE >::get_buffer_demand ( ) const [inline]

Returns

the server-side buffer demand based in *IFACE*.

Definition at line 98 of file ipc_server.

12.151.2.3 template<typename IFACE, typename BASE = L4::Server_object> template<typename THIS > static int L4::Server_object_t< IFACE, BASE >::proto_dispatch (THIS * *self*, l4_umword_t *rights*, L4::Ipc::Iostream & *ios*) [inline], [static]

Implementation of protocol-based dispatch for this server object.

Parameters

self	The this pointer for teh object (inhertits from Server_object_t).
rights	The rights from the received IPC (forwarded to p_dispatch()).
ios	The message stream for the incoming and the reply message.

Server objects may call this function from their dispatch() function. This function reads the protocol ID from the message tag and uses the p_dispatch code to dispatch to overloaded p_dispatch functions of self.

Definition at line 126 of file ipc_server.

Referenced by L4::Server_object_x< Derived, IFACE, BASE >::dispatch().

Here is the caller graph for this function:

The documentation for this struct was generated from the following file:

- l4/cxx/ipc_server

12.152 L4::Server_object_x< Derived, IFACE, BASE > Struct Template Reference

Helper class to implement p_dispatch based server objects.

Inheritance diagram for L4::Server_object_x< Derived, IFACE, BASE >:

Collaboration diagram for L4::Server_object_x< Derived, IFACE, BASE >:

Public Member Functions

- int dispatch (l4_umword_t r, L4::Ipc::Iostream &ios)

Implementation forwarding to p_dispatch().

Additional Inherited Members

12.152.1 Detailed Description

template<typename Derived, typename IFACE, typename BASE = L4::Server_object>struct L4::Server_object_x< Derived, IFACE, BASE >

Helper class to implement p_dispatch based server objects.

Template Parameters

Derived	The data type of your server object class.
IFACE	The data type providing the interface definition for the object.
BASE	Optional data-type of the base server object (usually L4::Server_object)

This class implements the standard dispatch() function of L4::Server_object and forwards incoming messages to a set of overloaded p_dispatch() functions. There must be a p_dispatch() function in Derived for each interface provided by IFACE with the signature

```
int p_dispatch(Iface *, unsigned rights, L4::Ipc::Iostream &)
```

that is called for messages with protocol == Iface::Protocol.

Example signature for L4Re::Dataspace is:

```
int p_dispatch(L4Re::Dataspace *, unsigned, L4::Ipc::Iostream &)
```

Definition at line 155 of file ipc_server.

The documentation for this struct was generated from the following file:

- l4/cxx/ipc_server

12.153 L4::Smart_cap< T, SMART > Class Template Reference

Smart capability class.

Inherits L4::Cap_base, and SMART.

Collaboration diagram for L4::Smart_cap< T, SMART >:

Public Member Functions

- template<typename O >
  Smart_cap (Cap< O > const &p) throw ()

  Internal Constructor, use to generate a capability from a this pointer.
- Cap< T > operator-> () const throw ()

  Member access of a T.

12.153.1 Detailed Description

template<typename T, typename SMART>class L4::Smart_cap< T, SMART >

Smart capability class.

Definition at line 36 of file smart_capability.

12.153.2 Constructor & Destructor Documentation

12.153.2.1 template<typename T, typename SMART> template<typename O > L4::Smart_cap< T, SMART >::Smart_cap (Cap< O > const & *p*) throw) [inline]

Internal Constructor, use to generate a capability from a *this* pointer.

Attention

This constructor is only useful to generate a capability from the *this* pointer of an objected that is an L4::Kobject. Do *never* use this constructor for something else!

Parameters

p	The *this* pointer of the Kobject or derived object

Definition at line 69 of file smart_capability.

The documentation for this class was generated from the following file:

- l4/sys/smart_capability

12.154 L4::String Class Reference

A null-terminated string container class.

`#include <string.h>`

Collaboration diagram for L4::String:

12.154.1 Detailed Description

A null-terminated string container class.

Definition at line 35 of file string.h.

The documentation for this class was generated from the following file:

- l4/cxx/string.h

12.155 L4::Task Class Reference

C++ interface of the Task kernel object.

Inheritance diagram for L4::Task:

Collaboration diagram for L4::Task:

Public Member Functions

- l4_msgtag_t map (Cap< Task > const &src_task, l4_fpage_t const &snd_fpage, l4_addr_t snd_base, l4_utcb_t *utcb=l4_utcb()) throw ()

  Map resources available in the source task to a destination task.
- l4_msgtag_t unmap (l4_fpage_t const &fpage, l4_umword_t map_mask, l4_utcb_t *utcb=l4_utcb()) throw ()

  Revoke rights from the task.
- l4_msgtag_t unmap_batch (l4_fpage_t const *fpages, unsigned num_fpages, l4_umword_t map_mask, l4_utcb_t *utcb=l4_utcb()) throw ()

  Revoke rights from a task.
- l4_msgtag_t delete_obj (L4::Cap< void > obj, l4_utcb_t *utcb=l4_utcb()) throw ()

  Release capability and delete object.

- l4_msgtag_t release_cap (L4::Cap< void > cap, l4_utcb_t *utcb=l4_utcb()) throw ()

  Release capability.

- l4_msgtag_t cap_valid (Cap< void > const &cap, l4_utcb_t *utcb=l4_utcb()) throw ()

  Test whether a capability selector points to a valid capability.

- l4_msgtag_t cap_has_child (Cap< void > const &cap, l4_utcb_t *utcb=l4_utcb()) throw ()

  Test whether a capability has child mappings (in another task).

- l4_msgtag_t cap_equal (Cap< void > const &cap_a, Cap< void > const &cap_b, l4_utcb_t *utcb=l4_utcb()) throw ()

  Test whether two capabilities point to the same object with the same rights.

- l4_msgtag_t add_ku_mem (l4_fpage_t const &fpage, l4_utcb_t *utcb=l4_utcb()) throw ()

  Add kernel-user memory.

Additional Inherited Members

12.155.1 Detailed Description

C++ interface of the Task kernel object.

The L4::Task class represents a combination of the address spaces provided by the L4Re micro kernel. A task consists of at least a memory address space and an object address space. On IA32 there is also an IO-port address space associated with an L4::Task.

L4::Task objects are created using the L4::Factory interface.

Include File

```
#include <l4/sys/task>
```

Definition at line 43 of file task.

12.155.2 Member Function Documentation

12.155.2.1 l4_msgtag_t L4::Task::add_ku_mem (l4_fpage_t const & *fpage*, l4_utcb_t * *utcb* = l4_utcb ()) throw) [inline]

Add kernel-user memory.

Parameters

fpage	Flexpage describing the virtual area the memory goes to.
utcb	UTCP pointer of the calling thread.

Returns

Syscall return tag

Definition at line 182 of file task.

References L4::Kobject::cap().

Here is the call graph for this function:

12.155.2.2 l4_msgtag_t L4::Task::cap_equal (Cap< void > const & *cap_a*, Cap< void > const & *cap_b*, l4_utcb_t * *utcb* = l4_utcb ()) throw) [inline]

Test whether two capabilities point to the same object with the same rights.

Parameters

cap_a	First capability selector to compare.
cap_b	Second capability selector to compare.
utcb	Optiona: UTCB pointer of the calling thread.

Returns

If the label is 1 `cap_a` and `cap_b` point to the same object. Otherwise the label is 0.

Definition at line 171 of file task.

References L4::Kobject::cap().

Here is the call graph for this function:

12.155.2.3 l4_msgtag_t L4::Task::cap_has_child (Cap< void > const & *cap*, l4_utcb_t * *utcb* = l4_utcb ()) throw) [inline]

Test whether a capability has child mappings (in another task).

Parameters

cap	Capability selector to look up in the destination task.
utcb	UTCB pointer of the calling thread.

Returns

If the label is 1 `cap` has at least one child mapping in another task. Otherwise the label is 0.

Definition at line 158 of file task.

References L4::Kobject::cap().

Here is the call graph for this function:

12.155.2.4 l4_msgtag_t L4::Task::cap_valid (Cap< void > const & *cap*, l4_utcb_t * *utcb* = l4_utcb ()) throw) [inline]

Test whether a capability selector points to a valid capability.

Parameters

cap	Capability selector to check for validity.
utcb	UTCB pointer of the calling thread.

Returns

If the label is >0 the capability is valid. Otherwise the label is 0.

Definition at line 147 of file task.

References L4::Kobject::cap().

Here is the call graph for this function:

12.155.2.5 l4_msgtag_t L4::Task::delete_obj (L4::Cap< void > *obj*, l4_utcb_t * *utcb* = l4_utcb ()) throw) [inline]

Release capability and delete object.

Parameters

obj	Capability selector of the object to delete.
utcb	UTCB pointer of the calling thread.

Returns

Syscall return tag

The object will be deleted if the `obj` has sufficient rights. No error will be reported if the rights are insufficient, however, the capability is removed in all cases.

Definition at line 124 of file task.

References L4::Kobject::cap().

Here is the call graph for this function:

12.155.2.6 l4_msgtag_t L4::Task::map (Cap< Task > const & *src_task*, l4_fpage_t const & *snd_fpage*, l4_addr_t *snd_base*, l4_utcb_t * *utcb* = l4_utcb ()) throw) [inline]

Map resources available in the source task to a destination task.

Parameters

src_task	Capability selector of the source task.
snd_fpage	Send flexpage that describes an area in the address space or object space of the source task.
snd_base	Send base that describes an offset in the receive window of the destination task.
utcb	UTCB pointer of the calling thread.

Returns

Syscall return tag.

This method allows for asynchronous rights delegation from one task to another. It can be used to share memory as well as to delegate access to objects.

Definition at line 62 of file task.

References L4::Kobject::cap().

Here is the call graph for this function:

12.155.2.7 l4_msgtag_t L4::Task::release_cap (L4::Cap< void > *cap*, l4_utcb_t * *utcb* = l4_utcb ()) throw) [inline]

Release capability.

Parameters

cap	Capability selector to release.
utcb	UTCB pointer of the calling thread.

Returns

Syscall return tag.

This operation unmaps the capability from `this` task.

Definition at line 136 of file task.

References L4::Kobject::cap().

Here is the call graph for this function:

12.155.2.8 l4_msgtag_t L4::Task::unmap (l4_fpage_t const & *fpage*, l4_umword_t *map_mask*, l4_utcb_t * *utcb* = l4_utcb ()) throw) [inline]

Revoke rights from the task.

Parameters

fpage	Flexpage that describes an area in the address space or object space of `this` task
map_mask	Unmap mask, see l4_unmap_flags_t
utcb	UTCB pointer of the calling thread.

Returns

Syscall return tag

This method allows to revoke rights from the destination task and from all the tasks that got the rights delegated from that task (i.e., this operation does a recursive rights revocation).

Note

If the capability posseses delete rights or if it is the last capability pointing to the object, calling this function might destroy the object itself.

Definition at line 84 of file task.

References L4::Kobject::cap().

Here is the call graph for this function:

12.155.2.9 l4_msgtag_t L4::Task::unmap_batch (l4_fpage_t const * *fpages*, unsigned *num_fpages*, l4_umword_t *map_mask*, l4_utcb_t * *utcb* = l4_utcb ()) throw) [inline]

Revoke rights from a task.

Parameters

fpages	An array of flexpages. Each item describes an area in the address or object space of `this` task.
num_fpages	Number of fpages in the `fpages` array.
map_mask	Unmap mask, see l4_unmap_flags_t.
utcb	UTCB pointer of the calling thread.

This method allows to revoke rights from the destination task and from all the tasks that got the rights delegated from that task (i.e., this operation does a recursive rights revocation).

Precondition

The caller needs to take care that `num_fpages` is not bigger than L4_UTCB_GENERIC_DATA_SIZE - 2.

Note

If the capability posseses delete rights or if it is the last capability pointing to the object, calling this function might destroy the object itself.

Definition at line 108 of file task.

References L4::Kobject::cap().

Here is the call graph for this function:

The documentation for this class was generated from the following file:

- l4/sys/task

12.156 L4::Thread Class Reference

C++ L4 kernel thread interface.

Inheritance diagram for L4::Thread:

Collaboration diagram for L4::Thread:

Data Structures

- class Attr

  Thread attributes used for control_commit().
- class Modify_senders

  Wrapper class for modifying senders.

Public Member Functions

- l4_msgtag_t ex_regs (l4_addr_t ip, l4_addr_t sp, l4_umword_t flags, l4_utcb_t *utcb=l4_utcb()) throw ()

  Exchange basic thread registers.

- l4_msgtag_t ex_regs (l4_addr_t *ip, l4_addr_t *sp, l4_umword_t *flags, l4_utcb_t *utcb=l4_utcb()) throw ()
  Exchange basic thread registers and return previous values.
- l4_msgtag_t control (Attr const &attr) throw ()
  Commit the given thread-attributes object.
- l4_msgtag_t switch_to (l4_utcb_t *utcb=l4_utcb()) throw ()
  Switch execution to this thread.
- l4_msgtag_t stats_time (l4_kernel_clock_t *us, l4_utcb_t *utcb=l4_utcb()) throw ()
  Get consumed time of thread in us.
- l4_msgtag_t vcpu_resume_start (l4_utcb_t *utcb=l4_utcb()) throw ()
  vCPU resume, start.
- l4_msgtag_t vcpu_resume_commit (l4_msgtag_t tag, l4_utcb_t *utcb=l4_utcb()) throw ()
  vCPU resume, commit.
- l4_msgtag_t vcpu_control (l4_addr_t vcpu_state, l4_utcb_t *utcb=l4_utcb()) throw ()
  Enable or disable the vCPU feature for the thread.
- l4_msgtag_t vcpu_control_ext (l4_addr_t ext_vcpu_state, l4_utcb_t *utcb=l4_utcb()) throw ()
  Enable or disable the extended vCPU feature for the thread.
- l4_msgtag_t register_del_irq (Cap< Irq > irq, l4_utcb_t *u=l4_utcb()) throw ()
  Register an IRQ that will trigger upon deletion events.
- l4_msgtag_t modify_senders (Modify_senders const &todo) throw ()
  Apply sender modifiction rules.

Additional Inherited Members

12.156.1 Detailed Description

C++ L4 kernel thread interface.

The Thread class defines a thread of execution in the L4 context. Usually user-level and kernel threads are mapped 1:1 to each other. Thread kernel objects are created using a factory, see the L4::Factory API (L4::Factory::create()).

Amongst other things an L4::Thread encapsulates:

- CPU state
  - General-purpose registers
  - Program counter
  - Stack pointer
- FPU state
- Scheduling parameters, see the L4::Scheduler API
- Execution state
  - Blocked, Runnable, Running

Thread objects provide an API for

- Thread configuration and manipulation
- Thread switching.

Include File

```
#include <l4/sys/thread>
```

For the C interface see the Thread API.

Definition at line 58 of file thread.

12.156.2 Member Function Documentation

12.156.2.1 l4_msgtag_t L4::Thread::control (Attr const & *attr*) throw) [inline]

Commit the given thread-attributes object.

Parameters

attr	the attribute object to commit to the thread.

Definition at line 213 of file thread.

References L4::Kobject::cap().

Here is the call graph for this function:

12.156.2.2 l4_msgtag_t L4::Thread::ex_regs (l4_addr_t *ip*, l4_addr_t *sp*, l4_umword_t *flags*, l4_utcb_t * *utcb* = l4_utcb ()) throw) [inline]

Exchange basic thread registers.

Parameters

ip	New instruction pointer, use ~0UL to leave the instruction pointer unchanged.
sp	New stack pointer, use ~0UL to leave the stack pointer unchanged.
flags	Ex-regs flags, see L4_thread_ex_regs_flags.
utcb	UTCB to use for this operation.

Returns

System call return tag

This method allows to manipulate a thread. The basic functionality is to set the instruction pointer and the stack pointer of a thread. Additionally, this method allows also to cancel ongoing IPC operations and to force the thread to raise an artificial exception (see `flags`).

The thread is started using L4::Scheduler::run_thread(). However, if at the time L4::Scheduler::run_thread() is called, the instruction pointer of the thread is invalid, a later call to ex_regs() with a valid instruction pointer might start the thread.

Definition at line 85 of file thread.

References L4::Kobject::cap(), and l4_thread_ex_regs_u().

Here is the call graph for this function:

12.156.2.3 l4_msgtag_t L4::Thread::ex_regs (l4_addr_t * *ip*, l4_addr_t * *sp*, l4_umword_t * *flags*, l4_utcb_t * *utcb* = l4_utcb ()) throw) [inline]

Exchange basic thread registers and return previous values.

Parameters

in,out	*ip*	New instruction pointer, use ~0UL to leave the instruction pointer unchanged, return previous instruction pointer.
in,out	*sp*	New stack pointer, use ~0UL to leave the stack pointer unchanged, returns previous stack pointer.
in,out	*flags*	Ex-regs flags, see L4_thread_ex_regs_flags, return previous CPU flags of the thread.
	utcb	UTCB to use for this operation.

Returns

System call return tag

This method allows to manipulate and start a thread. The basic functionality is to set the instruction pointer and the stack pointer of a thread. Additionally, this method allows also to cancel ongoing IPC operations and to force the thread to raise an artificial exception (see `flags`).

Returned values are valid only if function returns successfully.

Definition at line 112 of file thread.

References L4::Kobject::cap(), and l4_thread_ex_regs_ret_u().

Here is the call graph for this function:

12.156.2.4 l4_msgtag_t L4::Thread::modify_senders (Modify_senders const & *todo*) throw) [inline]

Apply sender modifiction rules.

Parameters

todo	Prepared sender modification rules.

Returns

System call return tag.

Definition at line 361 of file thread.

References L4::Kobject::cap(), l4_ipc_call(), L4_IPC_NEVER, l4_msgtag(), and L4_PROTO_THREAD.

Referenced by L4Re::Util::Object_registry::unregister_obj().

Here is the call graph for this function:

Here is the caller graph for this function:

12.156.2.5 l4_msgtag_t L4::Thread::register_del_irq (Cap< Irq > *irq*, l4_utcb_t * *u* = l4_utcb ()) throw) [inline]

Register an IRQ that will trigger upon deletion events.

See also

l4_thread_register_del_irq

Definition at line 301 of file thread.

References L4::Kobject::cap().

Here is the call graph for this function:

12.156.2.6 l4_msgtag_t L4::Thread::stats_time (l4_kernel_clock_t * *us*, l4_utcb_t * *utcb* = l4_utcb ()) throw) [inline]

Get consumed time of thread in us.

Parameters

	utcb	UTCB of the current thread.
out	*us*	Consumed time in µs.

Returns

Syscall resturn tag.

Definition at line 232 of file thread.

References L4::Kobject::cap().

Here is the call graph for this function:

12.156.2.7 l4_msgtag_t L4::Thread::switch_to (l4_utcb_t * *utcb* = l4_utcb ()) throw) [inline]

Switch execution to this thread.

Parameters

utcb	the UTCB of the current thread.

Note

The current time slice is inherited to this thread.

Definition at line 222 of file thread.

References L4::Kobject::cap().

Here is the call graph for this function:

12.156.2.8 l4_msgtag_t L4::Thread::vcpu_control (l4_addr_t *vcpu_state*, l4_utcb_t * *utcb* = l4_utcb ()) throw) [inline]

Enable or disable the vCPU feature for the thread.

Parameters

vcpu_state	The virtual address where the kernel shall store the vCPU state in case of vCPU exits. The address must be a valid kernel-user-memory address (see L4::Task::add_ku_mem()).
utcb	UTCB to use for this operation.

Returns

Syscall resturn tag.

This function enables the vCPU feature of `this` thread if `vcpu_state` is set to a valid kernel-user-memory address, or disables the vCPU feature if `vcpu_state` is 0.

Definition at line 268 of file thread.

References L4::Kobject::cap(), and l4_thread_vcpu_control_u().

Here is the call graph for this function:

12.156.2.9 l4_msgtag_t L4::Thread::vcpu_control_ext (l4_addr_t *ext_vcpu_state*, l4_utcb_t * *utcb* = l4_utcb ()) throw) [inline]

Enable or disable the extended vCPU feature for the thread.

Parameters

ext_vcpu_state	The virtual address where the kernel shall store the vCPU state in case of vCPU exits. The address must be a valid kernel-user-memory address (see L4::Task::add_ku_mem()).
utcb	UTCB to use for this operation.

Returns

Syscall return tag.

The extended vCPU feature allows the use of hardware-virtualization features such as Intel's VT or AMD's SVM.

This function enables the extended vCPU feature of `this` thread if `ext_vcpu_state` is set to a valid kernel-user-memory address, or disables the vCPU feature if `ext_vcpu_state` is 0.

Note

The extended vCPU mode includes the normal vCPU mode.

Definition at line 292 of file thread.

References L4::Kobject::cap(), and l4_thread_vcpu_control_ext_u().

Here is the call graph for this function:

12.156.2.10 l4_msgtag_t L4::Thread::vcpu_resume_commit (l4_msgtag_t *tag*, l4_utcb_t * *utcb* = l4_utcb ()) throw) [inline]

vCPU resume, commit.

See also

l4_thread_vcpu_resume_commit

Definition at line 249 of file thread.

References L4::Kobject::cap().

Here is the call graph for this function:

12.156.2.11 l4_msgtag_t L4::Thread::vcpu_resume_start (l4_utcb_t * *utcb* = l4_utcb ()) throw) [inline]

vCPU resume, start.

See also

l4_thread_vcpu_resume_start

Definition at line 241 of file thread.

The documentation for this class was generated from the following file:

- l4/sys/thread

12.157 L4::Thread::Attr Class Reference

Thread attributes used for control_commit().

Collaboration diagram for L4::Thread::Attr:

Public Member Functions

- Attr (l4_utcb_t *utcb=l4_utcb()) throw ()

  Create a thread-attribute object with the given UTCB.
- void pager (Cap< void > const &pager) throw ()

  Set the pager capability selector.
- Cap< void > pager () throw ()

  Get the capability selector used for page-fault messages.
- void exc_handler (Cap< void > const &exc_handler) throw ()

  Set the exception-handler capability selector.
- Cap< void > exc_handler () throw ()

  Get the capability selector used for exception messages.
- void bind (l4_utcb_t *thread_utcb, Cap< Task > const &task) throw ()

  Bind the thread to a task.
- void alien (int on) throw ()

Set the thread to alien mode.

- void ux_host_syscall (int on) throw ()

  Allow host system calls on Fiasco-UX.

Friends

- class **L4::Thread**

12.157.1 Detailed Description

Thread attributes used for control_commit().

This class is responsible for initalizing various attributes of a thread in a UTCB for the control_commit() method.

See also

Thread control for some more details.

Definition at line 126 of file thread.

12.157.2 Constructor & Destructor Documentation

12.157.2.1 L4::Thread::Attr::Attr (l4_utcb_t * *utcb* = l4_utcb ()) throw) [inline], [explicit]

Create a thread-attribute object with the given UTCB.

Parameters

utcb	The UTCB to use for the later L4::Thread::control_commit() function. Usually this is the UTCB of the calling thread.

Definition at line 139 of file thread.

12.157.3 Member Function Documentation

12.157.3.1 void L4::Thread::Attr::bind (l4_utcb_t * *thread_utcb*, Cap< Task > const & *task*) throw) [inline]

Bind the thread to a task.

Parameters

thread_utcb	The UTCB address of the thread within the task specified by `task`.
task	The capability selector for the task the thread shall be bound to.

Binding a thread to a task means that the thread shall afterwards execute in the given task. To actually start execution you need to use L4::Thread::ex_regs().

Definition at line 191 of file thread.

12.157.3.2 void L4::Thread::Attr::exc_handler (Cap< void > const & *exc_handler*) throw) [inline]

Set the exception-handler capability selector.

Parameters

exc_handler	The capability selector that shall be used for exception messages. This capability selector must be valid within the task the thread is bound to.

Definition at line 168 of file thread.

References exc_handler().

Here is the call graph for this function:

12.157.3.3 Cap<void> L4::Thread::Attr::exc_handler ( ) throw) [inline]

Get the capability selector used for exception messages.

Returns

The capability selector used to send exception messages. The selector is valid in the task the thread is bound to.

Definition at line 177 of file thread.

References l4_msg_regs_t::mr.

Referenced by exc_handler().

Here is the caller graph for this function:

12.157.3.4 void L4::Thread::Attr::pager (Cap< void > const & *pager*) throw) [inline]

Set the pager capability selector.

Parameters

pager	The capability selector that shall be used for page-fault messages. This capability selector must be valid within the task the thread is bound to.

Definition at line 149 of file thread.

References pager().

Here is the call graph for this function:

12.157.3.5 Cap<void> L4::Thread::Attr::pager ( ) throw) [inline]

Get the capability selector used for page-fault messages.

Returns

The capability selector used to send page-fault messages. The selector is valid in the task the thread is bound to.

Definition at line 158 of file thread.

References l4_msg_regs_t::mr.

Referenced by pager().

Here is the caller graph for this function:

12.157.3.6 void L4::Thread::Attr::ux_host_syscall (int *on*) throw) [inline]

Allow host system calls on Fiasco-UX.

Precondition

Running on Fiasco-UX.

Definition at line 204 of file thread.

The documentation for this class was generated from the following file:

- l4/sys/thread

12.158 L4::Thread::Modify_senders Class Reference

Wrapper class for modifying senders.

Collaboration diagram for L4::Thread::Modify_senders:

Public Member Functions

- int add (l4_umword_t match_mask, l4_umword_t match, l4_umword_t del_bits, l4_umword_t add_bits) throw ()

  Add a rule.

12.158.1 Detailed Description

Wrapper class for modifying senders.

Use the add() function to add modification rules, and use modify_senders() to commit. Do not use the UTCB inbetween as it is used by add() and modify_senders().

Definition at line 311 of file thread.

12.158.2 Member Function Documentation

12.158.2.1 int L4::Thread::Modify_senders::add (l4_umword_t *match_mask*, l4_umword_t *match*, l4_umword_t *del_bits*, l4_umword_t *add_bits*) throw) [inline]

Add a rule.

Parameters

match_mask	Bitmask of bits to match the label.
match	Bitmask that must be equal to the label after applying match_mask.
del_bits	Bits to be deleted from the label.
add_bits	Bits to be added to the label.

Returns

0 on sucess, <0 on error

Only the first match is applied.

See also

l4_thread_modify_sender_add()

Definition at line 340 of file thread.

References L4_ENOMEM, L4_UTCB_GENERIC_DATA_SIZE, and l4_msg_regs_t::mr.

Referenced by L4Re::Util::Object_registry::unregister_obj().

Here is the caller graph for this function:

The documentation for this class was generated from the following file:

- l4/sys/thread

12.159 L4::Triggerable Struct Reference

Interface that allows an object to be triggered by some source.

Inheritance diagram for L4::Triggerable:

Collaboration diagram for L4::Triggerable:

Public Member Functions

- l4_msgtag_t trigger (l4_utcb_t *utcb=l4_utcb()) throw ()

  Trigger.

Additional Inherited Members

12.159.1 Detailed Description

Interface that allows an object to be triggered by some source.

This interface is usually used in conjunction with L4::Icu.

Definition at line 68 of file irq.

12.159.2 Member Function Documentation

12.159.2.1 l4_msgtag_t L4::Triggerable::trigger (l4_utcb_t * *utcb* = l4_utcb ()) throw) [inline]

Trigger.

Parameters

utcb	UTCB to be used for this operation, usually the UTCB of the calling thread.

Returns

Syscall return tag for a send-only operation, use l4_ipc_error() to check for errors (**do not** use l4_error()).

Note that this function is a send-only operation, this means there is no return value except for a failed send operation. Use l4_ipc_error() to check for errors, **do not** use l4_error(), because l4_error() will always return an error.

Use l4_ipc_error() to check for (send) errors.

Examples:

examples/libs/l4re/c++/shared_ds/ds_clnt.cc.

Definition at line 84 of file irq.

References L4::Kobject::cap(), and l4_irq_trigger_u().

Referenced by L4::Semaphore::up().

Here is the call graph for this function:

Here is the caller graph for this function:

The documentation for this struct was generated from the following file:

- l4/sys/irq

12.160 L4::Type_info Struct Reference

Dynamic Type Information for L4Re Interfaces.

```
#include <l4/sys/capability>
```

Collaboration diagram for L4::Type_info:

Data Structures

- class Demand

  Data type for expressing the needed receive buffers at the server-side of an interface.
- struct Demand_t

  Template type statically describing demand of receive buffers.
- struct Demand_union_t

  Template type statically describing the combination of two Demand object.

12.160.1 Detailed Description

Dynamic Type Information for L4Re Interfaces.

This class represents the runtime-dynamic type information for L4Re interfaces, and is not intended to be used directly by applications.

Note

The interface of is subject to changes.

The main use for this info is to be used by the implementation of the L4::cap_dynamic_cast() function.

Definition at line 509 of file __typeinfo.h.

The documentation for this struct was generated from the following file:

- l4/sys/__typeinfo.h

12.161 L4::Type_info::Demand Class Reference

Data type for expressing the needed receive buffers at the server-side of an interface.

```
#include <l4/sys/capability>
```

Inheritance diagram for L4::Type_info::Demand:

Collaboration diagram for L4::Type_info::Demand:

Public Member Functions

- Demand (unsigned char caps=0, unsigned char flags=0, unsigned char mem=0, unsigned char ports=0)

  Make Demand object.
- bool no_demand () const
- Demand operator| (Demand const &rhs) const

  get the combined demand of this and rhs

Data Fields

- unsigned char caps

  number of capability receive buffers.
- unsigned char flags

  flags, such as the need for timeouts (TBD).
- unsigned char mem

  number of memory receive buffers.
- unsigned char ports

  number of IO-port receive buffers.

12.161.1 Detailed Description

Data type for expressing the needed receive buffers at the server-side of an interface.

Definition at line 516 of file __typeinfo.h.

12.161.2 Constructor & Destructor Documentation

12.161.2.1 L4::Type_info::Demand::Demand (unsigned char *caps* = 0, unsigned char *flags* = 0, unsigned char *mem* = 0, unsigned char *ports* = 0) [inline],[explicit]

Make Demand object.

Parameters

caps	number of capability receive buffers
flags	flags, such as the need for timeouts (TBD).
mem	number of memory receive windows.
ports	number of IO-port receive windows.

Definition at line 537 of file __typeinfo.h.

12.161.3 Member Function Documentation

12.161.3.1 bool L4::Type_info::Demand::no_demand ( ) const [inline]

Returns

true if there is no demand at all

Definition at line 542 of file __typeinfo.h.

Referenced by L4::Ipc_svr::Br_manager_no_buffers::alloc_buffer_demand().

Here is the caller graph for this function:

The documentation for this class was generated from the following file:

- l4/sys/__typeinfo.h

12.162 L4::Type_info::Demand_t< CAPS, FLAGS, MEM, PORTS > Struct Template Reference

Template type statically describing demand of receive buffers.

```
#include <l4/sys/capability>
```

Inheritance diagram for L4::Type_info::Demand_t< CAPS, FLAGS, MEM, PORTS >:

Collaboration diagram for L4::Type_info::Demand_t< CAPS, FLAGS, MEM, PORTS >:

Public Types

- enum { Caps = CAPS, Flags = FLAGS, Mem = MEM, Ports = PORTS }

Additional Inherited Members

12.162.1 Detailed Description

template<unsigned char CAPS = 0, unsigned char FLAGS = 0, unsigned char MEM = 0, unsigned char PORTS = 0>struct L4::↩ Type_info::Demand_t< CAPS, FLAGS, MEM, PORTS >

Template type statically describing demand of receive buffers.

Template Parameters

CAPS	number of capability receive buffers needed.
FLAGS	flags, such as the need for timeouts (TBD).
MEM	number of memory receive windows needed.
PORTS	number of IO-port receive windows needed.

Definition at line 563 of file __typeinfo.h.

The documentation for this struct was generated from the following file:

- l4/sys/__typeinfo.h

12.163 L4::Type_info::Demand_union_t< D1, D2 > Struct Template Reference

Template type statically describing the combination of two Demand object.

```
#include <l4/sys/capability>
```

Inheritance diagram for L4::Type_info::Demand_union_t< D1, D2 >:

Collaboration diagram for L4::Type_info::Demand_union_t< D1, D2 >:

Additional Inherited Members

12.163.1 Detailed Description

template<typename D1, typename D2>struct L4::Type_info::Demand_union_t< D1, D2 >

Template type statically describing the combination of two Demand object.

Template Parameters

D1	first demand object.
D2	second demand object.

Definition at line 583 of file __typeinfo.h.

The documentation for this struct was generated from the following file:

- l4/sys/__typeinfo.h

12.164 L4::Typeid::Detail::_Rpcs< OPCODE, O, X > Struct Template Reference

Empty list of RPCs.

```
#include <__typeinfo.h>
```

Inheritance diagram for L4::Typeid::Detail::_Rpcs< OPCODE, O, X >:

Collaboration diagram for L4::Typeid::Detail::_Rpcs< OPCODE, O, X >:

12.164.1 Detailed Description

template<typename OPCODE, unsigned O, typename... X>struct L4::Typeid::Detail::_Rpcs< OPCODE, O, X >

Empty list of RPCs.

Definition at line 375 of file __typeinfo.h.

The documentation for this struct was generated from the following file:

- l4/sys/__typeinfo.h

12.165 L4::Typeid::Detail::_Rpcs< OPCODE, O, Default_op< R > >::Rpc< Y > Struct Template Reference

Find the given RPC in the list.

```
#include <__typeinfo.h>
```

Inherits L4::Typeid::Detail::Rpc< OP, RPCS >.

Collaboration diagram for L4::Typeid::Detail::_Rpcs< OPCODE, O, Default_op< R > >::Rpc< Y >:

12.165.1 Detailed Description

template<typename OPCODE, unsigned O, typename R>template<typename Y>struct L4::Typeid::Detail::_Rpcs< OPCODE, O, Default_op< R > >::Rpc< Y >

Find the given RPC in the list.

Definition at line 409 of file __typeinfo.h.

The documentation for this struct was generated from the following file:

- l4/sys/__typeinfo.h

12.166 L4::Typeid::Detail::_Rpcs< OPCODE, O, R, X...> Struct Template Reference

Non-empty list of RPCs.

```
#include <__typeinfo.h>
```

Collaboration diagram for L4::Typeid::Detail::_Rpcs< OPCODE, O, R, X...>:

Data Structures

- struct Rpc

Find the given RPC in the list.

Public Types

- enum

  The opcode value to use for this RPC, may be bogus if the opcode_type is void.
- typedef _Rpcs type

  The list element itself.
- typedef OPCODE opcode_type

  The data type for the opcode.
- typedef R rpc

  The RPC type L4::Ipc::Msg::Rpc_call or L4::Ipc::Msg::Rpc_inline_call.
- typedef _Rpcs< OPCODE,
  _Get_opcode< R, O >::value+1,
  X...>::type next

  The next RPC in the list or Rpcs_end if this is the last.

12.166.1 Detailed Description

template<typename OPCODE, unsigned O, typename R, typename... X>struct L4::Typeid::Detail::_Rpcs< OPCODE, O, R, X...>

Non-empty list of RPCs.

Definition at line 379 of file __typeinfo.h.

The documentation for this struct was generated from the following file:

- l4/sys/__typeinfo.h

12.167 L4::Typeid::Detail::_Rpcs< OPCODE, O, R, X...>::Rpc< Y > Struct Template Reference

Find the given RPC in the list.

```
#include <__typeinfo.h>
```

Inherits L4::Typeid::Detail::Rpc< OP, RPCS >.

Collaboration diagram for L4::Typeid::Detail::_Rpcs< OPCODE, O, R, X...>::Rpc< Y >:

12.167.1 Detailed Description

template<typename OPCODE, unsigned O, typename R, typename... X>template<typename Y>struct L4::Typeid::Detail::_Rpcs< OPCODE, O, R, X...>::Rpc< Y >

Find the given RPC in the list.

Definition at line 392 of file __typeinfo.h.

The documentation for this struct was generated from the following file:

- l4/sys/__typeinfo.h

12.168 L4::Typeid::Detail::Rpcs_end Struct Reference

Internal end-of-list marker.

```
#include <__typeinfo.h>
```

Inheritance diagram for L4::Typeid::Detail::Rpcs_end:

Collaboration diagram for L4::Typeid::Detail::Rpcs_end:

12.168.1 Detailed Description

Internal end-of-list marker.

Definition at line 327 of file __typeinfo.h.

The documentation for this struct was generated from the following file:

- l4/sys/__typeinfo.h

12.169 L4::Typeid::P_dispatch< LIST > Struct Template Reference

Use for protocol based dispatch stage.

```
#include <__typeinfo.h>
```

Inherits L4::Typeid::_P_dispatch< LIST >.

Collaboration diagram for L4::Typeid::P_dispatch< LIST >:

12.169.1 Detailed Description

template<typename LIST>struct L4::Typeid::P_dispatch< LIST >

Use for protocol based dispatch stage.

Definition at line 318 of file __typeinfo.h.

The documentation for this struct was generated from the following file:

- l4/sys/__typeinfo.h

12.170 L4::Typeid::Raw_ipc< CLASS > Struct Template Reference

RPCs list for passing raw incoming IPC to the server object.

```
#include <l4/sys/capability>
```

Collaboration diagram for L4::Typeid::Raw_ipc< CLASS >:

12.170.1 Detailed Description

template<typename CLASS>struct L4::Typeid::Raw_ipc< CLASS >

RPCs list for passing raw incoming IPC to the server object.

Template Parameters

CLASS	The type of the interface (e.g., L4::Icu)

This template allows to have fully handcrafted IPC protocols.

Definition at line 422 of file __typeinfo.h.

The documentation for this struct was generated from the following file:

- l4/sys/__typeinfo.h

12.171 L4::Typeid::Rpc_nocode< OPERATION > Struct Template Reference

List of RPCs of an interface using a single operation without an opcode.

```
#include <l4/sys/capability>
```

Inheritance diagram for L4::Typeid::Rpc_nocode< OPERATION >:

Collaboration diagram for L4::Typeid::Rpc_nocode< OPERATION >:

12.171.1 Detailed Description

template<typename OPERATION>struct L4::Typeid::Rpc_nocode< OPERATION >

List of RPCs of an interface using a single operation without an opcode.

Template Parameters

OPERATION	The RPC operation as defined by L4_RPC etc.

Definition at line 464 of file __typeinfo.h.

The documentation for this struct was generated from the following file:

- l4/sys/__typeinfo.h

12.172 L4::Typeid::Rpcs< RPCS > Struct Template Reference

Standard list of RPCs of an interface.

```
#include <l4/sys/capability>
```

Inheritance diagram for L4::Typeid::Rpcs< RPCS >:

Collaboration diagram for L4::Typeid::Rpcs< RPCS >:

12.172.1 Detailed Description

template<typename... RPCS>struct L4::Typeid::Rpcs< RPCS >

Standard list of RPCs of an interface.

Template Parameters

RPCS	list of RPC types as defined by L4_RPC etc.

This is the default list for RPC functions of an interface, it uses L4::Opcode as opcode type and uses opcodes starting from 0.

Definition at line 438 of file __typeinfo.h.

The documentation for this struct was generated from the following file:

- l4/sys/__typeinfo.h

12.173 L4::Typeid::Rpcs_code< OPCODE_TYPE > Struct Template Reference

List of RPCs of an interface using a special opcode type.

```
#include <l4/sys/capability>
```

Collaboration diagram for L4::Typeid::Rpcs_code< OPCODE_TYPE >:

Data Structures

- struct F

12.173.1 Detailed Description

template<typename OPCODE_TYPE>struct L4::Typeid::Rpcs_code< OPCODE_TYPE >

List of RPCs of an interface using a special opcode type.

Template Parameters

OPCODE_TYPE	The data type of the opcode.

List for RPC functions of an interface, using OPCODE_TYPE as data type for the opcode, opcodes starting from 0.

Definition at line 449 of file __typeinfo.h.

The documentation for this struct was generated from the following file:

- l4/sys/__typeinfo.h

12.174 L4::Typeid::Rpcs_code< OPCODE_TYPE >::F< RPCS > Struct Template Reference

```
#include <__typeinfo.h>
```

Inheritance diagram for L4::Typeid::Rpcs_code< OPCODE_TYPE >::F< RPCS >:

Collaboration diagram for L4::Typeid::Rpcs_code< OPCODE_TYPE >::F< RPCS >:

12.174.1 Detailed Description

template<typename OPCODE_TYPE>template<typename... RPCS>struct L4::Typeid::Rpcs_code< OPCODE_TYPE >::F< RPCS >

Template Parameters

RPCS	list of RPC types as defined by L4_RPC etc.

Definition at line 455 of file __typeinfo.h.

The documentation for this struct was generated from the following file:

- l4/sys/__typeinfo.h

12.175 L4::Typeid::Rpcs_sys< ARG > Struct Template Reference

List of RPCs typically used for kernel interfaces.

`#include <l4/sys/capability>`

Inheritance diagram for L4::Typeid::Rpcs_sys< ARG >:

Collaboration diagram for L4::Typeid::Rpcs_sys< ARG >:

12.175.1 Detailed Description

template<typename... ARG>struct L4::Typeid::Rpcs_sys< ARG >

List of RPCs typically used for kernel interfaces.

Template Parameters

RPCS	list of RPC types as defined by L4_RPC etc.

This list of RPC functions uses l4_umword_t as type for the opcode as most kernel protocol do.

Definition at line 475 of file __typeinfo.h.

The documentation for this struct was generated from the following file:

- l4/sys/__typeinfo.h

12.176 L4::Types::Bool< V > Struct Template Reference

Boolean meta type.

Inheritance diagram for L4::Types::Bool< V >:

Collaboration diagram for L4::Types::Bool< V >:

Public Types

- typedef Bool< V > type

  The meta type itself.

12.176.1 Detailed Description

template<bool V>struct L4::Types::Bool< V >

Boolean meta type.

Template Parameters

V	The boolean value

Definition at line 165 of file types.

The documentation for this struct was generated from the following file:

- l4/sys/cxx/types

12.177 L4::Types::False Struct Reference

False meta value.

Inheritance diagram for L4::Types::False:

Collaboration diagram for L4::Types::False:

Additional Inherited Members

12.177.1 Detailed Description

False meta value.

Definition at line 173 of file types.

The documentation for this struct was generated from the following file:

- l4/sys/cxx/types

12.178 L4::Types::Flags< BITS_ENUM, UNDERLYING > Class Template Reference

Template for defining typical Flags bitmaps.

Collaboration diagram for L4::Types::Flags< BITS_ENUM, UNDERLYING >:

Public Types

- enum None_type { None }

  The none type to get an empty bitmap.
- typedef UNDERLYING value_type

  type of the underlying value
- typedef BITS_ENUM bits_enum_type

  enum type defining a name for each bit
- typedef Flags< BITS_ENUM, UNDERLYING > type

  the Flags<> type itself

Public Member Functions

- Flags (None_type)

  Make an empty bitmap.
- Flags ()

  Make default Flags.
- Flags (BITS_ENUM e)

  Make flags from bit name.
- operator Private_bool * () const

  Support for `if (flags)` syntax (test for non-empty flags).
- bool operator! () const

  Support for `if (!flags)` syntax (test for empty flags).
- type & operator|= (type rhs)

  Support |= of two compatible Flags types.

- type & operator|= (bits_enum_type rhs)

  Support |= of Flags type and bit name.
- type & operator&= (type rhs)

  Support &= of two compatible Flags types.
- type & operator&= (bits_enum_type rhs)

  Support &= of Flags type and bit name.
- type operator~ () const

  Support ~ for Flags types.
- type & clear (bits_enum_type flag)

  Clear the given flag.
- value_type as_value () const

  Get the underlying value.

Static Public Member Functions

- static type from_raw (value_type v)

  Make flags from a raw value of value_type.

Friends

- type operator| (type lhs, type rhs)

  Support | of two compatible Flags types.
- type operator| (type lhs, bits_enum_type rhs)

  Support | of Flags type and bit name.
- type operator& (type lhs, type rhs)

  Support & of two compatible Flags types.
- type operator& (type lhs, bits_enum_type rhs)

  Support & of Flags type and bit name.

12.178.1 Detailed Description

template<typename BITS_ENUM, typename UNDERLYING = unsigned long>class L4::Types::Flags< BITS_ENUM, UNDERLYING >

Template for defining typical Flags bitmaps.

Template Parameters

BITS_ENUM	enum type that defines a name for each bit in the bitmap. The values of the enum members must be the number of the bit (*not* a mask).
UNDERLYING	The underlying data type used to represent the bitmap.

The resulting data type provides a type-safe version that allows bitwise `and` and `or` operations with the BITS_E↩NUM members. As well as, test for `0`or `!0`.

Example:

```
enum Test_flag
{
  Do_weak_tests,
  Do_strong_tests
};

typedef L4::Types::Flags<Test_flag> Test_flags;

Test_flags x = Do_weak_tests;

if (x & Do_strong_tests) { ... }
x |= Do_strong_tests;
if (x & Do_strong_tests) { ... }
```

Definition at line 63 of file types.

12.178.2 Member Enumeration Documentation

12.178.2.1 template<typename BITS_ENUM , typename UNDERLYING = unsigned long> enum L4::Types::Flags::None_type

The none type to get an empty bitmap.

Enumerator

None Use this to get an empty bitmap.

Definition at line 80 of file types.

12.178.3 Constructor & Destructor Documentation

12.178.3.1 template<typename BITS_ENUM , typename UNDERLYING = unsigned long> L4::Types::Flags< BITS_ENUM, UNDERLYING >::Flags (None_type) [inline]

Make an empty bitmap.

Usually used for implicit conversion from `Flags::None`.

```
Flags x = Flags::None;
```

Definition at line 90 of file types.

12.178.3.2 template<typename BITS_ENUM , typename UNDERLYING = unsigned long> L4::Types::Flags< BITS_ENUM, UNDERLYING >::Flags (BITS_ENUM *e*) [inline]

Make flags from bit name.

Usually used for implicit conversion for a bit name.

```
Test_flags f = Do_strong_tests;
```

Definition at line 103 of file types.

12.178.4 Member Function Documentation

12.178.4.1 template<typename BITS_ENUM , typename UNDERLYING = unsigned long> type& L4::Types::Flags< BITS_ENUM, UNDERLYING >::clear (bits_enum_type *flag*) [inline]

Clear the given flag.

Parameters

flag	The flag that shall be cleared.

`flags.clear(The_flag)` is a shortcut for `flags &= ~Flags(The_flag)`.

Definition at line 154 of file types.

References L4::Types::Flags< BITS_ENUM, UNDERLYING >::operator&=().

Here is the call graph for this function:

12.178.4.2 template<typename BITS_ENUM , typename UNDERLYING = unsigned long> static type L4::Types::Flags< BITS_ENUM, UNDERLYING >::from_raw (value_type *v*) [inline], [static]

Make flags from a raw value of *value_type*.

This function may be used for example in C wrapper code.

Definition at line 110 of file types.

The documentation for this class was generated from the following file:

- l4/sys/cxx/types

12.179 L4::Types::Same< A, B > Struct Template Reference

Compare two data types for equality.

Inheritance diagram for L4::Types::Same< A, B >:

Collaboration diagram for L4::Types::Same< A, B >:

Additional Inherited Members

12.179.1 Detailed Description

template<typename A, typename B>struct L4::Types::Same< A, B >

Compare two data types for equality.

Template Parameters

A	The first data type
B	The second data type

The result is the boolean True if A and B are the same types.

Definition at line 189 of file types.

The documentation for this struct was generated from the following file:

- l4/sys/cxx/types

12.180 L4::Types::True Struct Reference

True meta value.

Inheritance diagram for L4::Types::True:

Collaboration diagram for L4::Types::True:

Additional Inherited Members

12.180.1 Detailed Description

True meta value.

Definition at line 177 of file types.

The documentation for this struct was generated from the following file:

- l4/sys/cxx/types

12.181 L4::Unknown_error Class Reference

Exception for an unknown condition.

`#include <l4/cxx/exceptions>`

Inheritance diagram for L4::Unknown_error:

Collaboration diagram for L4::Unknown_error:

Public Member Functions

- char const ∗ str () const throw ()

  Return a human readable string for the exception.

Additional Inherited Members

12.181.1 Detailed Description

Exception for an unknown condition.

This error is usually used when a server returns an unknown return state to the client, this may indicate incompatible messages used by the client and the server.

Definition at line 219 of file exceptions.

The documentation for this class was generated from the following file:

- l4/cxx/exceptions

12.182 L4::Vcon Class Reference

C++ L4 Vcon interface.

Inheritance diagram for L4::Vcon:

Collaboration diagram for L4::Vcon:

Public Member Functions

- l4_msgtag_t send (char const *buf, int size, l4_utcb_t *utcb=l4_utcb()) const throw ()

  Send data to `this` virtual console.
- long write (char const *buf, int size, l4_utcb_t *utcb=l4_utcb()) const throw ()

  Write data to `this` virtual console.
- int read (char *buf, int size, l4_utcb_t *utcb=l4_utcb()) const throw ()

Read data from `this` virtual console.

- int read_with_flags (char *buf, int size, l4_utcb_t *utcb=l4_utcb()) const throw ()

  Read data from `this` virtual console which also returns flags.

- l4_msgtag_t set_attr (l4_vcon_attr_t const *attr, l4_utcb_t *utcb=l4_utcb()) const throw ()

  Set the attributes of `this` virtual console.

- l4_msgtag_t get_attr (l4_vcon_attr_t *attr, l4_utcb_t *utcb=l4_utcb()) const throw ()

  Get attributes of `this` virtual console.

Additional Inherited Members

12.182.1 Detailed Description

C++ L4 Vcon interface.

L4::Vcon is a virtual console for simple character-based input and output. The interrupt for read events is provided by the virtual key interrupt.

Include File

```
#include <l4/sys/vcon>
```

See the Virtual Console for the C interface.

Definition at line 43 of file vcon.

12.182.2 Member Function Documentation

12.182.2.1 l4_msgtag_t L4::Vcon::get_attr (l4_vcon_attr_t * *attr*, l4_utcb_t * *utcb* = l4_utcb ()) const throw) [inline]

Get attributes of `this` virtual console.

Parameters

out	*attr*	Attribute structure. Contains the attributes after a successfull call of this function.
	utcb	UTCB pointer of the calling thread.

Returns

Syscall return tag.

Definition at line 130 of file vcon.

References L4::Kobject::cap(), and l4_vcon_get_attr_u().

Here is the call graph for this function:

12.182.2.2 int L4::Vcon::read (char * *buf*, int *size*, l4_utcb_t * *utcb* = l4_utcb ()) const throw) [inline]

Read data from `this` virtual console.

Parameters

out	*buf*	Pointer to data buffer.
	size	Size of the data buffer in bytes.
	utcb	UTCB pointer of the calling thread.

Return values

<0	Error code.
>size	More bytes to read, `size` bytes are in the buffer `buf`.
<=size	Number of bytes read.

Note

Size must not exceed L4_VCON_READ_SIZE.

Definition at line 85 of file vcon.

References L4::Kobject::cap(), and l4_vcon_read_u().

Here is the call graph for this function:

12.182.2.3 int L4::Vcon::read_with_flags (char * *buf*, int *size*, l4_utcb_t * *utcb* = l4_utcb ()) const throw) [inline]

Read data from `this` virtual console which also returns flags.

Parameters

out	*buf*	Pointer to data buffer.
	size	Size of the data buffer in bytes.
	utcb	UTCB pointer of the calling thread.

Return values

<0	Error code.
>size	More bytes to read, `size` bytes are in the buffer `buf`.
<=size	Number of bytes read.

If this function returns a positive value the caller can check the #L4_VCON_READ_STAT_BREAK flag bit for a break condition. The bytes read can be obtained by masking the return value with #L4_VCON_READ_SIZE_MASK.

If a break condition is signaled, it is always the first event in the transmitted content, i.e. all characters supplied by this read call follow the break condition.

Note

Size must not exceed L4_VCON_READ_SIZE.

Definition at line 108 of file vcon.

References L4::Kobject::cap().

Here is the call graph for this function:

12.182.2.4 l4_msgtag_t L4::Vcon::send (char const * *buf*, int *size*, l4_utcb_t * *utcb* = l4_utcb ()) const throw) [inline]

Send data to `this` virtual console.

Parameters

buf	Pointer to the data buffer.
size	Size of the data buffer in bytes.
utcb	UTBC pointer of the calling thread.

Returns

Syscall return tag

Note

Size must not exceed L4_VCON_WRITE_SIZE, a proper value of the `size` parameter is NOT checked.

Definition at line 58 of file vcon.

References L4::Kobject::cap(), and l4_vcon_send_u().

Here is the call graph for this function:

12.182.2.5 l4_msgtag_t L4::Vcon::set_attr (l4_vcon_attr_t const * *attr*, l4_utcb_t * *utcb* = l4_utcb ()) const throw) [inline]

Set the attributes of `this` virtual console.

Parameters

attr	Attribute structure with the attributes for the virtual console.
utcb	UTCB pointer of the calling thread.

Returns

Syscall return tag.

Definition at line 119 of file vcon.

References L4::Kobject::cap(), and l4_vcon_set_attr_u().

Here is the call graph for this function:

12.182.2.6 long L4::Vcon::write (char const * *buf*, int *size*, l4_utcb_t * *utcb* = l4_utcb ()) const throw) [inline]

Write data to `this` virtual console.

Parameters

buf	Pointer to the data buffer.
size	Size of the data buffer in bytes.
utcb	UTCB pointer of the calling thread.

Return values

<0	Error.
>=0	Number of bytes written to the virtual console.

Definition at line 70 of file vcon.

References L4::Kobject::cap(), and l4_vcon_write_u().

Here is the call graph for this function:

The documentation for this class was generated from the following file:

- l4/sys/vcon

12.183 L4::Vm Class Reference

Virtual machine.

Inheritance diagram for L4::Vm:

Collaboration diagram for L4::Vm:

Additional Inherited Members

12.183.1 Detailed Description

Virtual machine.

Definition at line 35 of file __vm.

The documentation for this class was generated from the following file:

- l4/sys/__vm

12.184 l4_buf_regs_t Struct Reference

Encapsulation of the buffer-registers block in the UTCB.

```
#include <l4/sys/utcb.h>
```

Collaboration diagram for l4_buf_regs_t:

Data Fields

- l4_umword_t bdr

  Buffer descriptor.

- l4_umword_t br [L4_UTCB_GENERIC_BUFFERS_SIZE]

  Buffer registers.

12.184.1 Detailed Description

Encapsulation of the buffer-registers block in the UTCB.

Definition at line 96 of file utcb.h.

The documentation for this struct was generated from the following file:

- l4/sys/utcb.h

12.185 l4_exc_regs_t Struct Reference

UTCB structure for exceptions.

```
#include <utcb.h>
```

Collaboration diagram for l4_exc_regs_t:

Data Fields

- l4_umword_t pfa

  page fault address
- l4_umword_t err

  error code
- l4_umword_t tpidruro

  Thread-ID register.
- l4_umword_t r [13]

  registers
- l4_umword_t sp

  stack pointer
- l4_umword_t ulr

  ulr
- l4_umword_t _dummy1

  dummy
- l4_umword_t pc

  pc
- l4_umword_t cpsr

  cpsr
- l4_umword_t r15

  r15
- l4_umword_t r14

  r14
- l4_umword_t r13

  r13
- l4_umword_t r12

  r12
- l4_umword_t r11

r11

- l4_umword_t r10

  r10
- l4_umword_t r9

  r9
- l4_umword_t r8

  r8
- l4_umword_t rdi

  rdi
- l4_umword_t rsi

  rsi
- l4_umword_t rbp

  rbp
- l4_umword_t rbx

  rbx
- l4_umword_t rdx

  rdx
- l4_umword_t rcx

  rcx
- l4_umword_t rax

  rax
- l4_umword_t trapno

  trap number
- l4_umword_t ip

  instruction pointer
- l4_umword_t dummy1

  dummy
- l4_umword_t flags

  rflags
- l4_umword_t ss

  stack segment register
- l4_umword_t gs

  gs register
- l4_umword_t fs

  fs register
- l4_umword_t edi

  edi register
- l4_umword_t esi

  esi register
- l4_umword_t ebp

  ebp register
- l4_umword_t ebx

  ebx register
- l4_umword_t edx

  edx register
- l4_umword_t ecx

  ecx register
- l4_umword_t eax

  eax register

12.185.1 Detailed Description

UTCB structure for exceptions.

Examples:

examples/sys/aliens/main.c, examples/sys/singlestep/main.c, and examples/sys/start-with-exc/main.c.

Definition at line 58 of file utcb.h.

12.185.2 Field Documentation

12.185.2.1 l4_umword_t l4_exc_regs_t::flags

rflags

eflags

Definition at line 80 of file utcb.h.

The documentation for this struct was generated from the following file:

- arm/l4/sys/utcb.h

12.186 l4_fpage_t Union Reference

L4 flexpage type.

```
#include <__l4_fpage.h>
```

Collaboration diagram for l4_fpage_t:

Data Fields

- l4_umword_t fpage

  Raw value.
- l4_umword_t raw

  Raw value.

12.186.1 Detailed Description

L4 flexpage type.

Definition at line 81 of file __l4_fpage.h.

The documentation for this union was generated from the following file:

- l4/sys/__l4_fpage.h

12.187 l4_icu_info_t Struct Reference

Info structure for an ICU.

```
#include <icu.h>
```

Inheritance diagram for l4_icu_info_t:

Collaboration diagram for l4_icu_info_t:

Data Fields

- unsigned features

Feature flags.

- unsigned nr_irqs

  The number of IRQ lines supported by the ICU,.

- unsigned nr_msis

  The number of MSI vectors supported by the ICU,.

12.187.1 Detailed Description

Info structure for an ICU.

This structure contains information about the features of an ICU.

See also

l4_icu_info().

Definition at line 159 of file icu.h.

12.187.2 Field Documentation

12.187.2.1 unsigned l4_icu_info_t::features

Feature flags.

If L4_ICU_FLAG_MSI is set the ICU supports MSIs.

Definition at line 166 of file icu.h.

The documentation for this struct was generated from the following file:

- l4/sys/icu.h

12.188 l4_icu_msi_info_t Struct Reference

Info to use for a specific MSI.

```
#include <icu.h>
```

Collaboration diagram for l4_icu_msi_info_t:

Data Fields

- l4_uint64_t msi_addr

  Value to use as address when sending this MSI.
- l4_uint32_t msi_data

  Value to use as data written to msi_addr, when sending this MSI.

12.188.1 Detailed Description

Info to use for a specific MSI.

Definition at line 180 of file icu.h.

12.188.2 Field Documentation

12.188.2.1 l4_uint32_t l4_icu_msi_info_t::msi_data

Value to use as data written to msi_addr, when sending this MSI.

Definition at line 185 of file icu.h.

The documentation for this struct was generated from the following file:

- l4/sys/icu.h

12.189 l4_kernel_info_mem_desc_t Struct Reference

Memory descriptor data structure.

```
#include <memdesc.h>
```

Collaboration diagram for l4_kernel_info_mem_desc_t:

12.189.1 Detailed Description

Memory descriptor data structure.

Note

This data type is opaque, and must be accessed by the accessor functions defined in this module.

Definition at line 64 of file memdesc.h.

The documentation for this struct was generated from the following file:

- l4/sys/memdesc.h

12.190 l4_msg_regs_t Union Reference

Encapsulation of the message-register block in the UTCB.

```
#include <l4/sys/utcb.h>
```

Collaboration diagram for l4_msg_regs_t:

Data Fields

- l4_umword_t mr [L4_UTCB_GENERIC_DATA_SIZE]

  Message registers.
- l4_uint64_t mr64 [L4_UTCB_GENERIC_DATA_SIZE/(sizeof(l4_uint64_t)/sizeof(l4_umword_t))]

  Message registers 64bit alias.

12.190.1 Detailed Description

Encapsulation of the message-register block in the UTCB.

Examples:

examples/sys/utcb-ipc/main.c.

Definition at line 80 of file utcb.h.

The documentation for this union was generated from the following file:

- l4/sys/utcb.h

12.191 l4_msgtag_t Struct Reference

Message tag data structure.

```
#include <types.h>
```

Collaboration diagram for l4_msgtag_t:

Public Member Functions

- long label () const throw ()

  Get the protocol value.
- void label (long v) throw ()

  Set the protocol value.
- unsigned words () const throw ()

  Get the number of untyped words.
- unsigned items () const throw ()

  Get the number of typed items.
- unsigned flags () const throw ()

  Get the flags value.
- bool is_page_fault () const throw ()

  Test if protocol indicates page-fault protocol.
- bool is_preemption () const throw ()

  Test if protocol indicates preemption protocol.
- bool is_sys_exception () const throw ()

  Test if protocol indicates system-exception protocol.
- bool is_exception () const throw ()

  Test if protocol indicates exception protocol.
- bool is_sigma0 () const throw ()

  Test if protocol indicates sigma0 protocol.
- bool is_io_page_fault () const throw ()

  Test if protocol indicates IO-page-fault protocol.
- unsigned has_error () const throw ()

  Test if flags indicate an error.

Data Fields

- l4_mword_t raw

  raw value

12.191.1 Detailed Description

Message tag data structure.

`#include <l4/sys/types.h>`

Describes the details of an IPC operation, in particular which parts of the UTCB have to be transmitted, and also flags to enable real-time and FPU extensions.

The message tag also contains a user-defined label that could be used to specify a protocol ID. Some negative values are reserved for kernel protocols such as page faults and exceptions.

The type must be treated completely opaque.

Examples:

examples/libs/l4re/c++/shared_ds/ds_srv.cc, examples/libs/l4re/streammap/server.cc, examples/sys/aliens/main.c, examples/sys/ipc/ipc_example.c, examples/sys/isr/main.c, examples/sys/singlestep/main.c, examples/sys/start-with-exc/main.c, and examples/sys/utcb-ipc/main.c.

Definition at line 163 of file types.h.

12.191.2 Member Function Documentation

12.191.2.1 unsigned l4_msgtag_t::flags ( ) const throw) [inline]

Get the flags value.

The flags are a combination of the flags defined by l4_msgtag_flags.

Definition at line 181 of file types.h.

The documentation for this struct was generated from the following file:

- l4/sys/types.h

12.192 l4_sched_cpu_set_t Struct Reference

CPU sets.

`#include <scheduler.h>`

Collaboration diagram for l4_sched_cpu_set_t:

Public Member Functions

- unsigned char granularity () const
- unsigned offset () const
- void set (unsigned char granularity, unsigned offset)

  Set offset and granularity.

Data Fields

- l4_umword_t gran_offset

  Combination of granularity and offset.
- l4_umword_t map

  Bitmap of CPUs.

12.192.1 Detailed Description

CPU sets.

Examples:

examples/sys/migrate/thread_migrate.cc.

Definition at line 40 of file scheduler.h.

12.192.2 Member Function Documentation

12.192.2.1 unsigned char l4_sched_cpu_set_t::granularity ( ) const [inline]

Returns

Get granularity value

Definition at line 62 of file scheduler.h.

12.192.2.2 unsigned l4_sched_cpu_set_t::offset ( ) const [inline]

Returns

Get offset value

Definition at line 64 of file scheduler.h.

12.192.3 Field Documentation

12.192.3.1 l4_umword_t l4_sched_cpu_set_t::gran_offset

Combination of granularity and offset.

The granularity defines how many CPUs each bit in map describes. And the offset is the numer of the first CPU described by the first bit in the bitmap.

Precondition

offset must be a multiple of 2^graularity.

MSB	LSB
8bit granularity	24bit offset ..

Definition at line 53 of file scheduler.h.

Referenced by L4::Scheduler::info(), and l4_sched_cpu_set().

The documentation for this struct was generated from the following file:

- l4/sys/scheduler.h

12.193 l4_sched_param_t Struct Reference

Scheduler parameter set.

```
#include <scheduler.h>
```

Collaboration diagram for l4_sched_param_t:

Data Fields

- l4_sched_cpu_set_t affinity

  CPU affinity.
- l4_umword_t prio

  Priority for scheduling.
- l4_umword_t quantum

  Timeslice in micro seconds.

12.193.1 Detailed Description

Scheduler parameter set.

Examples:

examples/sys/aliens/main.c, examples/sys/migrate/thread_migrate.cc, examples/sys/singlestep/main.c, examples/sys/start-with-exc/main.c, and examples/sys/utcb-ipc/main.c.

Definition at line 106 of file scheduler.h.

The documentation for this struct was generated from the following file:

- l4/sys/scheduler.h

12.194 l4_snd_fpage_t Struct Reference

Send-flex-page types.

```
#include <__l4_fpage.h>
```

Collaboration diagram for l4_snd_fpage_t:

Data Fields

- l4_umword_t snd_base

  Offset in receive window (send base)

- l4_fpage_t fpage

  Source flex-page descriptor.

12.194.1 Detailed Description

Send-flex-page types.

Definition at line 98 of file __l4_fpage.h.

The documentation for this struct was generated from the following file:

- l4/sys/__l4_fpage.h

12.195 l4_thread_regs_t Struct Reference

Encapsulation of the thread-control-register block of the UTCB.

```
#include <l4/sys/utcb.h>
```

Collaboration diagram for l4_thread_regs_t:

Data Fields

- l4_umword_t error

  System call error codes.

- l4_timeout_t xfer

  Message transfer timeout.

- l4_umword_t user [3]

  User values (ignored and preserved by the kernel)

12.195.1 Detailed Description

Encapsulation of the thread-control-register block of the UTCB.

Definition at line 114 of file utcb.h.

The documentation for this struct was generated from the following file:

- l4/sys/utcb.h

12.196 l4_timeout_s Struct Reference

Basic timeout specification.

`#include <__timeout.h>`

Collaboration diagram for l4_timeout_s:

Data Fields

- l4_uint16_t t

  timeout value

12.196.1 Detailed Description

Basic timeout specification.

Basically a floating point number with 10 bits mantissa and 5 bits exponent (t = m∗2^e).

The timeout can also specify an absolute point in time (bit 16 == 1).

Definition at line 45 of file __timeout.h.

The documentation for this struct was generated from the following file:

- l4/sys/__timeout.h

12.197 l4_timeout_t Union Reference

Timeout pair.

`#include <__timeout.h>`

Collaboration diagram for l4_timeout_t:

Data Fields

- l4_uint32_t raw

  raw value
- struct {
  - l4_timeout_s rcv

    receive timeout
  - l4_timeout_s snd

    send timeout

  } p

  combined timeout

12.197.1 Detailed Description

Timeout pair.

For IPC there are usually a send and a receive timeout. So this structure contains a pair of timeouts.

Definition at line 57 of file __timeout.h.

The documentation for this union was generated from the following file:

- l4/sys/__timeout.h

12.198 l4_tracebuffer_status_t Struct Reference

Trace-buffer status.

```
#include <ktrace.h>
```

Collaboration diagram for l4_tracebuffer_status_t:

Data Fields

- volatile l4_tracebuffer_entry_t * current_entry

  Address of the most current event in trace-buffer.
- l4_uint32_t logevents [LOG_EVENT_MAX_EVENTS]

  Available LOG events.
- l4_uint32_t scaler_tsc_to_ns

  Scaler used for translation of CPU cycles to nano seconds.
- l4_uint32_t scaler_tsc_to_us

  Scaler used for translation of CPU cycles to micro seconds.
- l4_uint32_t scaler_ns_to_tsc

  Scaler used for translation of nano seconds to CPU cycles.
- volatile l4_uint32_t cnt_context_switch

  Number of context switches (intra AS or inter AS)
- volatile l4_uint32_t cnt_addr_space_switch

  Number of inter AS context switches.
- volatile l4_uint32_t cnt_shortcut_failed

  How often was the IPC shortcut taken.
- volatile l4_uint32_t cnt_shortcut_success

How often was the IPC shortcut not taken.

- volatile l4_uint32_t cnt_irq

  Number of hardware interrupts (without kernel scheduling interrupt)

- volatile l4_uint32_t cnt_ipc_long

  Number of long IPCs.

- volatile l4_uint32_t cnt_page_fault

  Number of page faults.

- volatile l4_uint32_t cnt_io_fault

  Number of faults (application runs at IOPL 0 and tries to execute cli, sti, in, or out but does not have a sufficient right in the I/O bitmap)

- volatile l4_uint32_t cnt_task_create

  Number of tasks created.

- volatile l4_uint32_t cnt_schedule

  Number of reschedules.

- volatile l4_uint32_t cnt_iobmap_tlb_flush

  Number of flushes of the I/O bitmap.

12.198.1 Detailed Description

Trace-buffer status.

Definition at line 75 of file ktrace.h.

12.198.2 Field Documentation

12.198.2.1 volatile l4_uint32_t l4_tracebuffer_status_t::cnt_iobmap_tlb_flush

Number of flushes of the I/O bitmap.

Increases on context switches between two small address spaces if at least one of the spaces has an I/O bitmap allocated.

Definition at line 114 of file ktrace.h.

The documentation for this struct was generated from the following file:

- l4/sys/ktrace.h

12.199 l4_tracebuffer_status_window_t Struct Reference

Trace-buffer status window descriptor.

```
#include <ktrace.h>
```

Collaboration diagram for l4_tracebuffer_status_window_t:

Data Fields

- l4_tracebuffer_entry_t * tracebuffer

  Address of trace-buffer.

- l4_umword_t size

  Size of trace-buffer.

- volatile l4_uint64_t version

  Version number of trace-buffer (incremented if trace-buffer overruns)

12.199.1 Detailed Description

Trace-buffer status window descriptor.

Definition at line 60 of file ktrace.h.

The documentation for this struct was generated from the following file:

- l4/sys/ktrace.h

12.200 l4_vcon_attr_t Struct Reference

Vcon attribute structure.

```
#include <vcon.h>
```

Collaboration diagram for l4_vcon_attr_t:

Data Fields

- l4_umword_t i_flags

  input flags

- l4_umword_t o_flags

  output flags

- l4_umword_t l_flags

  local flags

12.200.1 Detailed Description

Vcon attribute structure.

Definition at line 173 of file vcon.h.

The documentation for this struct was generated from the following file:

- l4/sys/vcon.h

12.201 l4_vcpu_ipc_regs_t Struct Reference

vCPU message registers.

```
#include <__vcpu-arch.h>
```

Collaboration diagram for l4_vcpu_ipc_regs_t:

12.201.1 Detailed Description

vCPU message registers.

Definition at line 52 of file __vcpu-arch.h.

The documentation for this struct was generated from the following file:

- arm/l4/sys/__vcpu-arch.h

12.202 l4_vcpu_regs_t Struct Reference

vCPU registers.

```
#include <__vcpu-arch.h>
```

Collaboration diagram for l4_vcpu_regs_t:

Data Fields

- l4_umword_t pfa

  page fault address
- l4_umword_t err

  error code
- l4_umword_t sp

  stack pointer
- l4_umword_t ip

  instruction pointer
- l4_umword_t flags

  eflags
- l4_umword_t r15

  r15 register
- l4_umword_t r14

  r14 register
- l4_umword_t r13

  r13 register
- l4_umword_t r12

  r12 register
- l4_umword_t r11

  r11 register
- l4_umword_t r10

  r10 register
- l4_umword_t r9

  r9 register
- l4_umword_t r8

  r8 reigster
- l4_umword_t di

rdi register

- l4_umword_t si

  rsi register
- l4_umword_t bp

  rbp register
- l4_umword_t bx

  rbx register
- l4_umword_t dx

  rdx register
- l4_umword_t cx

  rcx register
- l4_umword_t ax

  rax register
- l4_umword_t trapno

  trap number
- l4_umword_t cs

  dummy
- l4_umword_t ss

  ss register
- l4_umword_t es

  gs register
- l4_umword_t ds

  fs register
- l4_umword_t gs

  gs register
- l4_umword_t fs

  fs register
- l4_umword_t dummy1

  dummy

12.202.1 Detailed Description

vCPU registers.

Definition at line 27 of file __vcpu-arch.h.

12.202.2 Field Documentation

12.202.2.1 l4_umword_t l4_vcpu_regs_t::ax

rax register

eax register

Definition at line 61 of file __vcpu-arch.h.

12.202.2.2 l4_umword_t l4_vcpu_regs_t::bp

rbp register

ebp register

Definition at line 56 of file __vcpu-arch.h.

12.202.2.3 l4_umword_t l4_vcpu_regs_t::bx

rbx register

ebx register

Definition at line 58 of file __vcpu-arch.h.

12.202.2.4 l4_umword_t l4_vcpu_regs_t::cx

rcx register

ecx register

Definition at line 60 of file __vcpu-arch.h.

12.202.2.5 l4_umword_t l4_vcpu_regs_t::di

rdi register

edi register

Definition at line 54 of file __vcpu-arch.h.

12.202.2.6 l4_umword_t l4_vcpu_regs_t::dx

rdx register

edx register

Definition at line 59 of file __vcpu-arch.h.

12.202.2.7 l4_umword_t l4_vcpu_regs_t::si

rsi register

esi register

Definition at line 55 of file __vcpu-arch.h.

The documentation for this struct was generated from the following file:

- arm/l4/sys/__vcpu-arch.h

12.203 l4_vcpu_state_t Struct Reference

State of a vCPU.

```
#include <vcpu.h>
```

Inheritance diagram for l4_vcpu_state_t:

Collaboration diagram for l4_vcpu_state_t:

Data Fields

- l4_vcpu_regs_t r

  Register state.
- l4_vcpu_ipc_regs_t i

  IPC state.
- l4_uint16_t state

Current vCPU state.

- l4_uint16_t saved_state

  Saved vCPU state.

- l4_uint16_t sticky_flags

  Pending flags.

- l4_cap_idx_t user_task

  User task to use.

- l4_umword_t entry_sp

  Stack pointer for entry (when coming from user task)

- l4_umword_t entry_ip

  IP for entry.

12.203.1 Detailed Description

State of a vCPU.

Definition at line 34 of file vcpu.h.

The documentation for this struct was generated from the following file:

- l4/sys/vcpu.h

12.204 l4_vhw_descriptor Struct Reference

Virtual hardware devices description.

```
#include <vhw.h>
```

Collaboration diagram for l4_vhw_descriptor:

Data Fields

- l4_uint32_t magic

  Magic.
- l4_uint8_t version

  Version of the descriptor.
- l4_uint8_t count

  Number of entries.
- l4_uint8_t pad1

  padding
- l4_uint8_t pad2

  padding
- struct l4_vhw_entry descs []

  Array of device descriptions.

12.204.1 Detailed Description

Virtual hardware devices description.

Examples:

examples/sys/ux-vhw/main.c.

Definition at line 70 of file vhw.h.

12.204.2 Field Documentation

12.204.2.1 l4_uint8_t l4_vhw_descriptor::count

Number of entries.

Examples:

examples/sys/ux-vhw/main.c.

Definition at line 73 of file vhw.h.

12.204.2.2 struct l4_vhw_entry l4_vhw_descriptor::descs[]

Array of device descriptions.

Definition at line 77 of file vhw.h.

12.204.2.3 l4_uint32_t l4_vhw_descriptor::magic

Magic.

Examples:

examples/sys/ux-vhw/main.c.

Definition at line 71 of file vhw.h.

12.204.2.4 l4_uint8_t l4_vhw_descriptor::version

Version of the descriptor.

Examples:

examples/sys/ux-vhw/main.c.

Definition at line 72 of file vhw.h.

The documentation for this struct was generated from the following file:

- l4/sys/vhw.h

12.205 l4_vhw_entry Struct Reference

Description of a device.

```
#include <vhw.h>
```

Collaboration diagram for l4_vhw_entry:

Data Fields

- enum l4_vhw_entry_type type

  Type of virtual hardware.
- l4_uint32_t provider_pid

  Host PID of the VHW provider.
- l4_addr_t mem_start

  Start of memory region.
- l4_addr_t mem_size

  Size of memory region.
- l4_uint32_t irq_no

  IRQ number.
- l4_uint32_t fd

  File descriptor.

12.205.1 Detailed Description

Description of a device.

Examples:

examples/sys/ux-vhw/main.c.

Definition at line 55 of file vhw.h.

12.205.2 Field Documentation

12.205.2.1 l4_uint32_t l4_vhw_entry::fd

File descriptor.

Definition at line 63 of file vhw.h.

12.205.2.2 l4_uint32_t l4_vhw_entry::irq_no

IRQ number.

Examples:

examples/sys/ux-vhw/main.c.

Definition at line 62 of file vhw.h.

12.205.2.3 l4_addr_t l4_vhw_entry::mem_size

Size of memory region.

Examples:

examples/sys/ux-vhw/main.c.

Definition at line 60 of file vhw.h.

12.205.2.4 l4_addr_t l4_vhw_entry::mem_start

Start of memory region.

Examples:

examples/sys/ux-vhw/main.c.

Definition at line 59 of file vhw.h.

12.205.2.5 l4_uint32_t l4_vhw_entry::provider_pid

Host PID of the VHW provider.

Examples:

examples/sys/ux-vhw/main.c.

Definition at line 57 of file vhw.h.

12.205.2.6 enum l4_vhw_entry_type l4_vhw_entry::type

Type of virtual hardware.

Examples:

examples/sys/ux-vhw/main.c.

Definition at line 56 of file vhw.h.

The documentation for this struct was generated from the following file:

- l4/sys/vhw.h

12.206 l4_vm_svm_vmcb_control_area Struct Reference

VMCB structure for SVM VMs.

```
#include <__vm-svm.h>
```

Collaboration diagram for l4_vm_svm_vmcb_control_area:

12.206.1 Detailed Description

VMCB structure for SVM VMs.

Definition at line 39 of file __vm-svm.h.

The documentation for this struct was generated from the following file:

- l4/sys/__vm-svm.h

12.207 l4_vm_svm_vmcb_state_save_area Struct Reference

State save area structure for SVM VMs.

```
#include <__vm-svm.h>
```

Collaboration diagram for l4_vm_svm_vmcb_state_save_area:

12.207.1 Detailed Description

State save area structure for SVM VMs.

Definition at line 96 of file __vm-svm.h.

The documentation for this struct was generated from the following file:

- l4/sys/__vm-svm.h

12.208 l4_vm_svm_vmcb_state_save_area_seg Struct Reference

State save area segment selector struct.

```
#include <__vm-svm.h>
```

Collaboration diagram for l4_vm_svm_vmcb_state_save_area_seg:

12.208.1 Detailed Description

State save area segment selector struct.

Definition at line 84 of file __vm-svm.h.

The documentation for this struct was generated from the following file:

- l4/sys/__vm-svm.h

12.209 l4_vm_svm_vmcb_t Struct Reference

Control structure for SVM VMs.

```
#include <__vm-svm.h>
```

Collaboration diagram for l4_vm_svm_vmcb_t:

12.209.1 Detailed Description

Control structure for SVM VMs.

Definition at line 165 of file __vm-svm.h.

The documentation for this struct was generated from the following file:

- l4/sys/__vm-svm.h

12.210 l4_vm_tz_state Struct Reference

state structure for TrustZone VMs

```
#include <vm.h>
```

Collaboration diagram for l4_vm_tz_state:

12.210.1 Detailed Description

state structure for TrustZone VMs

Definition at line 52 of file vm.h.

The documentation for this struct was generated from the following file:

- arm/l4/sys/vm.h

12.211 L4Re::Cap_alloc Class Reference

Capability allocator interface.

Collaboration diagram for L4Re::Cap_alloc:

Public Member Functions

- virtual L4::Cap< void > alloc ()=0 throw ()

  Allocate a capability.
- template<typename T >
  L4::Cap< T > alloc () throw ()

  Allocate a capability.
- virtual void free (L4::Cap< void > cap)=0 throw ()

  Free a capability.
- virtual ~Cap_alloc ()=0

  Destructor.

Static Public Member Functions

- template<typename CAP_ALLOC >
  static L4Re::Cap_alloc * get_cap_alloc (CAP_ALLOC &ca)

  Construct an instance of a capability allocator.

12.211.1 Detailed Description

Capability allocator interface.

Definition at line 40 of file cap_alloc.

12.211.2 Member Function Documentation

12.211.2.1 virtual L4::Cap<void> L4Re::Cap_alloc::alloc ( ) throw) [pure virtual]

Allocate a capability.

Returns

Capability of type void

Referenced by alloc(), and get_cap_alloc().

Here is the caller graph for this function:

12.211.2.2 template<typename T > L4::Cap<T> L4Re::Cap_alloc::alloc ( ) throw) [inline]

Allocate a capability.

Returns

Capability of type T

Definition at line 62 of file cap_alloc.

References alloc(), and L4::cap_cast().

Here is the call graph for this function:

12.211.2.3 virtual void L4Re::Cap_alloc::free (L4::Cap< void > *cap*) throw) [pure virtual]

Free a capability.

Parameters

cap	Capability to free.

Referenced by get_cap_alloc().

Here is the caller graph for this function:

12.211.2.4 template<typename CAP_ALLOC > static L4Re::Cap_alloc* L4Re::Cap_alloc::get_cap_alloc (CAP_ALLOC & *ca*) [inline], [static]

Construct an instance of a capability allocator.

Parameters

ca	Capability allocator

Returns

Instance of a capability allocator.

Definition at line 85 of file cap_alloc.

References alloc(), and free().

Here is the call graph for this function:

The documentation for this class was generated from the following file:

- l4/re/cap_alloc

12.212 L4Re::Console Class Reference

Console class.

Inheritance diagram for L4Re::Console:

Collaboration diagram for L4Re::Console:

Additional Inherited Members

12.212.1 Detailed Description

Console class.

Definition at line 39 of file console.

The documentation for this class was generated from the following file:

- l4/re/console

12.213 L4Re::Dataspace Class Reference

Interface for memory-like objects.

Inheritance diagram for L4Re::Dataspace:

Collaboration diagram for L4Re::Dataspace:

Data Structures

- struct Stats

  Information about the dataspace.

Public Types

- enum Map_flags {
  Map_ro = 0, Map_rw = 1, Map_normal = 0x00, Map_cacheable = Map_normal,
  Map_bufferable = 0x10, Map_uncacheable = 0x20, Map_caching_mask = 0x30, Map_caching_shift = 4 }

  Flags for map operations.

Public Member Functions

- long map (l4_addr_t offset, unsigned long flags, l4_addr_t local_addr, l4_addr_t min_addr, l4_addr_t max_addr) const throw ()

  Request a flex-page mapping from the dataspace.
- long map_region (l4_addr_t offset, unsigned long flags, l4_addr_t min_addr, l4_addr_t max_addr) const throw ()

  Map a part of a dataspace completely.
- long clear (l4_addr_t offset, unsigned long size)

  Clear parts of a dataspace.
- long allocate (l4_addr_t offset, l4_size_t size)

  Allocate a range in the dataspace.
- long copy_in (l4_addr_t dst_offs, L4::Ipc::Cap< Dataspace > src, l4_addr_t src_offs, unsigned long size)

  Copy contents from another dataspace.
- long phys (l4_addr_t offset, l4_addr_t &phys_addr, l4_size_t &phys_size)

  Get the physical addresses of a dataspace.
- long size () const throw ()

  Get size of a dataspace.
- long flags () const throw ()

  Get flags of the dataspace.
- long info (Stats *stats)

  Get information on the dataspace.

Additional Inherited Members

12.213.1 Detailed Description

Interface for memory-like objects.

Dataspaces are a central abstraction provided by L4Re. A dataspace is an abstraction for any thing that is available via usual memory access instructions. A dataspace can be a file, as well as the memory-mapped registers of a device, or anonymous memory, such as a heap.

The dataspace interface defines a set of methods that allow any kind of dataspace to be attached (mapped) to the virtual address space of an L4 task and then be accessed via memory-access instructions. The L4Re::Rm interface can be used to attach a dataspace to a virtual address space of a task paged by a certain instance of a region map.

Include File

```
#include <l4/re/dataspace>
```

Examples:

examples/libs/l4re/c++/mem_alloc/ma+rm.cc, examples/libs/l4re/c++/shared_ds/ds_clnt.cc, and examples/libs/l4re/c++/shared_ds/ds_srv.cc.

Definition at line 59 of file dataspace.

12.213.2 Member Enumeration Documentation

12.213.2.1 enum L4Re::Dataspace::Map_flags

Flags for map operations.

Enumerator

Map_ro Request read-only mapping.

Map_rw Request writable mapping.

Map_normal request normal memory mapping

Map_cacheable request normal memory mapping

Map_bufferable request bufferable (write buffered) mappings

Map_uncacheable request uncacheable memory mappings

Map_caching_mask mask for caching flags

Map_caching_shift shift value for caching flags

Definition at line 68 of file dataspace.

12.213.3 Member Function Documentation

12.213.3.1 long L4Re::Dataspace::allocate (l4_addr_t *offset*, l4_size_t *size*)

Allocate a range in the dataspace.

Parameters

offset	Offset in the dataspace, in bytes.
size	Size of the range, in bytes.

Return values

L4_EOK	Success
-L4_ERANGE	Given range is outside the dataspace. (A dataspace provider may also silently ignore areas outside the dataspace.)
-L4_ENOMEM	Not enough memory available.
<0	IPC errors

On success, at least the given range is guaranteed to be allocated. The dataspace manager may also allocate more memory due to page granularity.

The memory is allocated with the same rights as the dataspace capability.

12.213.3.2 long L4Re::Dataspace::clear (l4_addr_t *offset*, unsigned long *size*)

Clear parts of a dataspace.

Parameters

offset	Offset within dataspace (in bytes).
size	Size of region to clear (in bytes).

Return values

>=0	Success.
-L4_ERANGE	Given range is outside the dataspace. (A dataspace provider may also silently ignore areas outside the dataspace.)
-L4_EACCESS	Dataspace is read-only.
<0	IPC errors

Zeroes out the memory. Depending on the type of memory the memory could also be deallocated and replaced by a shared zero-page.

12.213.3.3 long L4Re::Dataspace::copy_in (l4_addr_t *dst_offs*, L4::Ipc::Cap< Dataspace > *src*, l4_addr_t *src_offs*, unsigned long *size*)

Copy contents from another dataspace.

Parameters

dst_offs	Offset in destination dataspace.
src	Source dataspace to copy from.
src_offs	Offset in the source dataspace.
size	Size to copy (in bytes).

Return values

L4_EOK	Success
-L4_EACCESS	Destination dataspace not writable.
-L4_EINVAL	Invalid parameter supplied.
<0	IPC errors

The copy operation may use copy-on-write mechanisms. The operation may also fail if both dataspaces are not from the same dataspace manager or the dataspace managers do not cooperate.

12.213.3.4 long L4Re::Dataspace::flags ( ) const throw)

Get flags of the dataspace.

Return values

>=0	Flags of the dataspace
<0	IPC errors

See also

L4Re::Dataspace::Map_flags

Definition at line 116 of file dataspace_impl.h.

References L4Re::Dataspace::Stats::flags, and info().

Referenced by L4vbus::Vbus::request_resource().

Here is the call graph for this function:

Here is the caller graph for this function:

12.213.3.5 long L4Re::Dataspace::info (Stats * *stats*)

Get information on the dataspace.

Parameters

out	*stats*	Dataspace information

Return values

0	Success
<0	IPC errors

Referenced by L4virtio::Svr::Driver_mem_region_t< DATA >::Driver_mem_region_t(), flags(), and size().

Here is the caller graph for this function:

12.213.3.6 long L4Re::Dataspace::map (l4_addr_t *offset*, unsigned long *flags*, l4_addr_t *local_addr*, l4_addr_t *min_addr*, l4_addr_t *max_addr*) const throw)

Request a flex-page mapping from the dataspace.

Parameters

offset	Offset to start within dataspace
flags	map flags, see Map_flags.
local_addr	Local address to map to.
min_addr	Defines start of receive window. (Rounded down to page size.)
max_addr	Defines end of receive window. (Rounded up to page size.)

Return values

L4_EOK	Success
-L4_ERANGE	Invalid offset.
-L4_EPERM	Insufficient permission to map with requested rights.
<0	IPC errors

The map call will attempt to map the largest possible flexpage that covers the given local address and still fits into the region defined by `min_addr` and `max_addr`. If the given region is invalid or does not overlap the local address, the smallest valid page size is used.

Definition at line 92 of file dataspace_impl.h.

References l4_fpage_max_order(), L4_LOG2_PAGESIZE, l4_round_page(), and l4_trunc_page().

Here is the call graph for this function:

12.213.3.7 long L4Re::Dataspace::map_region (l4_addr_t *offset*, unsigned long *flags*, l4_addr_t *min_addr*, l4_addr_t *max_addr*) const throw)

Map a part of a dataspace completely.

Parameters

offset	Offset to start within dataspace
flags	map flags, see Map_flags.
min_addr	Defines start of receive window. (Rounded down to page size.)
max_addr	Defines end of receive window. (Rounded up to page size.)

Return values

L4_EOK	Success
-L4_ERANGE	Invalid offset.
-L4_EPERM	Insufficient permission to map with requested rights.
<0	IPC errors

Definition at line 55 of file dataspace_impl.h.

References l4_fpage_max_order(), L4_LOG2_PAGESIZE, l4_round_page(), l4_round_size(), l4_trunc_page(), l4_trunc_size(), and L4_UNLIKELY.

Referenced by L4Re::Rm::attach().

Here is the call graph for this function:

Here is the caller graph for this function:

12.213.3.8 long L4Re::Dataspace::phys (l4_addr_t *offset*, l4_addr_t & *phys_addr*, l4_size_t & *phys_size*)

Get the physical addresses of a dataspace.

Parameters

offset	Offset in dataspace

Return values

phys_addr	Physical address.
phys_size	Size of largest physically contiguous region in the dataspace after `phys_addr` (in bytes).
L4_EOK	Success
-L4_EINVAL	Dataspace is not pinned.
<0	IPC errors

This call will only succeed on pinned memory dataspaces.

Deprecated Use L4Re::Dma_space instead. This function will be removed in future releases.

12.213.3.9 long L4Re::Dataspace::size ( ) const throw)

Get size of a dataspace.

Return values

>0	Size of the dataspace (in bytes).
<0	IPC errors

Examples:

examples/libs/l4re/c++/shared_ds/ds_clnt.cc.

Definition at line 106 of file dataspace_impl.h.

References info(), and L4Re::Dataspace::Stats::size.

Here is the call graph for this function:

The documentation for this class was generated from the following files:

- l4/re/dataspace
- l4/re/impl/dataspace_impl.h

12.214 L4Re::Dataspace::Stats Struct Reference

Information about the dataspace.

Collaboration diagram for L4Re::Dataspace::Stats:

Data Fields

- unsigned long size

  size
- unsigned long flags

  flags

12.214.1 Detailed Description

Information about the dataspace.

Definition at line 85 of file dataspace.

The documentation for this struct was generated from the following file:

- l4/re/dataspace

12.215 L4Re::Debug_obj Class Reference

Debug interface.

Inheritance diagram for L4Re::Debug_obj:

Collaboration diagram for L4Re::Debug_obj:

Public Member Functions

- long debug (unsigned long function)

  Debug call.

Additional Inherited Members

12.215.1 Detailed Description

Debug interface.

See also

Debugging API .

Definition at line 51 of file debug.

12.215.2 Member Function Documentation

12.215.2.1 long L4Re::Debug_obj::debug (unsigned long *function*)

Debug call.

Parameters

function	Function to call.

Returns

- L4_EOK
  - IPC errors

The documentation for this class was generated from the following file:

- l4/re/debug

12.216 L4Re::Dma_space Class Reference

DMA Address Space.

Inherits L4::Kobject_0t< Derived, PROTO, S_DEMAND >.

Collaboration diagram for L4Re::Dma_space:

Public Types

- enum Direction { Bidirectional, To_device, From_device, None }

  Direction of the DMA transfers.
- enum Attribute { No_sync }

  Attributes used for the memory region during the transfer.
- enum Space_attrib { Coherent, Phys_space }

  Attributes assigned to the DMA space when associated with a specific device.
- typedef l4_uint64_t Dma_addr

  Data type for DMA addresses.
- typedef L4::Types::Flags < Attribute > Attributes

  Attributes for DMA mappings.
- typedef L4::Types::Flags < Space_attrib > Space_attribs

  Attributes used when configuring the DMA space.

Public Member Functions

- long map (L4::Ipc::Cap< L4Re::Dataspace > src, l4_addr_t offset, L4::Ipc::In_out< l4_size_t * > size, Attributes attrs, Direction dir, Dma_addr *dma_addr)

  Map the given part of this data space into the DMA address space.
- long unmap (Dma_addr dma_addr, l4_size_t size, Attributes attrs, Direction dir)

  Unmap the given part of this data space from the DMA address space.
- long associate (L4::Ipc::Opt< L4::Ipc::Cap< L4::Task > > dma_task, Space_attribs attr)

  Associate a DMA task for a device to this Dma_space.
- long disassociate ()

  Disassociate the DMA task from this Dma_space.

12.216.1 Detailed Description

DMA Address Space.

A Dma_space represents the DMA address space of a device. Whenever a device needs direct access to parts of an L4Re::Dataspace, that part of the data space must be mapped to the DMA address space that is assigned to that device. After the DMA accesses to the memory are finished the memory must be unmapped from the device's DMA address space.

Mapping to a DMA address space, using map(), makes the given parts of the data space visible to the associated device at the returned DMA address. As long as the memory is mapped into a DMA space it is 'pinned' and cannot be subject to dynamic memory management such as swapping. Additionally, map() is responsible for the necessary syncing operations before the DMA.

unmap() is the reverse operation to map() and unmaps the given data-space part for the DMA address space. unmap() is responsible for the necessary sync operations after the DMA.

Definition at line 61 of file dma_space.

12.216.2 Member Typedef Documentation

12.216.2.1 typedef L4::Types::Flags<Attribute> L4Re::Dma_space::Attributes

Attributes for DMA mappings.

See also

Attribute

Definition at line 106 of file dma_space.

12.216.3 Member Enumeration Documentation

12.216.3.1 enum L4Re::Dma_space::Attribute

Attributes used for the memory region during the transfer.

See also

Attributes

Enumerator

No_sync Do not sync the memory hierarchy. When this flag is *not set* (default) the memory region shall be made coherent to the point-of-coherency of the device associated with this Dma_space. When using this attribute the client is responsible for syncing the memory hierarchy for DMA. This can either be done using

the cache API or by another map() or unmap() operation of the same part of the data space (without the No_sync attribute).

Definition at line 85 of file dma_space.

12.216.3.2 enum L4Re::Dma_space::Direction

Direction of the DMA transfers.

Enumerator

Bidirectional device reads and writes to the memory

To_device device reads the memory

From_device device writes to the memory

None device is coherently connected to the memory

Definition at line 73 of file dma_space.

12.216.3.3 enum L4Re::Dma_space::Space_attrib

Attributes assigned to the DMA space when associated with a specific device.

See also

Space_attribs

Enumerator

Coherent The device is connected coherently with the cache. This means that the map() and unmap() do not need to sync CPU caches before and after DMA.

Phys_space The DMA space has no DMA task assigned and uses the CPUs physical memory.

Definition at line 113 of file dma_space.

12.216.4 Member Function Documentation

12.216.4.1 long L4Re::Dma_space::associate (L4::Ipc::Opt< L4::Ipc::Cap< L4::Task > > *dma_task,* Space_attribs *attr*)

Associate a DMA task for a device to this Dma_space.

Precondition

requires capability rights: {RW}

Parameters

in	*dma_task*	The DMA task used for the device that shall be associated with this DMA space. The dma_task might be an invalid capability when Phys_space is set in attr, in this case the CPUs physical memory is used as DMA address space.

in	*attr*	Attributes for this DMA space. See Space_attrib.

12.216.4.2 long L4Re::Dma_space::disassociate ( )

Disassociate the DMA task from this Dma_space.

Precondition

requires capability rights: {RW}

12.216.4.3 long L4Re::Dma_space::map (L4::Ipc::Cap< L4Re::Dataspace > *src*, l4_addr_t *offset*, L4::Ipc::In_out< l4_size_t * > *size*, Attributes *attrs*, Direction *dir*, Dma_addr * *dma_addr*)

Map the given part of this data space into the DMA address space.

Precondition

requires capability rights: {R}

Parameters

in	*src*	Source data space (that describes the memory). Caller needs write rights to the data space.
in	*offset*	The offset (bytes) with in `src`.
in,out	*size*	The size (bytes) of the of the region to be mapped to for DMA, after successful mapping the size returned is the size mapped for DMA as single block, this size might be smaller than the original input size, in this case the caller might call map() again with a new offset and the remaining size.
in	*attrs*	The attributes used for this DMA mapping (a combination of Dma_space::Attribute values).
in	*dir*	The direction of the DMA transfer issued with this mapping. The same value must later be passed to unmap().
out	*dma_addr*	The DMA address to use for DMA with the associated device.

Returns

0 in the case of success, a negative error code otherwise.

12.216.4.4 long L4Re::Dma_space::unmap (Dma_addr *dma_addr*, l4_size_t *size*, Attributes *attrs*, Direction *dir*)

Unmap the given part of this data space from the DMA address space.

Precondition

requires capability rights: {R}

Parameters

dma_addr	The DMA address (returned by Dma_space::map()).

size	The size (bytes) of the memory region to unmap.
attrs	The attributes for the unmap (currently none).
dir	The direction of the finished DMA operation.

The documentation for this class was generated from the following file:

- l4/re/dma_space

12.217 L4Re::Env Class Reference

Initial Environment (C++ version).

Collaboration diagram for L4Re::Env:

Public Types

- typedef l4re_env_cap_entry_t Cap_entry

  C++ type for an entry in the initial objects array.

Public Member Functions

- L4::Cap< Parent > parent () const throw ()

  Object-capability to the parent.
- L4::Cap< Mem_alloc > mem_alloc () const throw ()

  Object-capability to the memory allocator.
- L4::Cap< L4::Factory > user_factory () const throw ()

  Object-capability to the user-level object factory.
- L4::Cap< Rm > rm () const throw ()

  Object-capability to the region map.
- L4::Cap< Log > log () const throw ()

Object-capability to the logging service.

- L4::Cap< L4::Thread > main_thread () const throw ()

  Object-capability of the first user thread.
- L4::Cap< L4::Task > task () const throw ()

  Object-capability of the user task.
- L4::Cap< L4::Factory > factory () const throw ()

  Object-capability to the factory object available to the task.
- l4_cap_idx_t first_free_cap () const throw ()

  First available capability selector.
- l4_fpage_t utcb_area () const throw ()

  UTCB area of the task.
- l4_addr_t first_free_utcb () const throw ()

  First free UTCB.
- Cap_entry const * initial_caps () const throw ()

  Get a pointer to the first entry in the initial objects array.
- Cap_entry const * get (char const *name, unsigned l) const throw ()

  Get the Cap_entry for the object named name.
- template<typename T >
  L4::Cap< T > get_cap (char const *name, unsigned l) const throw ()

  Get the capability selector for the object named name.
- template<typename T >
  L4::Cap< T > get_cap (char const *name) const throw ()

  Get the capability selector for the object named name.
- void parent (L4::Cap< Parent > const &c) throw ()

  Set parent object-capability.
- void mem_alloc (L4::Cap< Mem_alloc > const &c) throw ()

  Set memory allocator object-capability.
- void rm (L4::Cap< Rm > const &c) throw ()

  Set region map object-capability.
- void log (L4::Cap< Log > const &c) throw ()

  Set log object-capability.
- void main_thread (L4::Cap< L4::Thread > const &c) throw ()

  Set object-capability of first user thread.
- void factory (L4::Cap< L4::Factory > const &c) throw ()

  Set factory object-capability.
- void first_free_cap (l4_cap_idx_t c) throw ()

  Set first available capability selector.
- void utcb_area (l4_fpage_t utcbs) throw ()

  Set UTCB area of the task.
- void first_free_utcb (l4_addr_t u) throw ()

  Set first free UTCB.
- L4::Cap< L4::Scheduler > scheduler () const throw ()

  Get the scheduler capability for the task.
- void scheduler (L4::Cap< L4::Scheduler > const &c) throw ()

  Set the scheduler capability.
- void initial_caps (Cap_entry *first) throw ()

  Set the pointer to the first Cap_entry in the initial objects array.

Static Public Member Functions

- static Env const * env () throw ()

  Returns the initial environment for the current task.

12.217.1 Detailed Description

Initial Environment (C++ version).

This class provides an initial set of capabilities as well as information the first free UTCB and used capability slots.

See also

Initial environment

Definition at line 85 of file env.

12.217.2 Member Function Documentation

12.217.2.1 static Env const* L4Re::Env::env ( ) throw) [inline], [static]

Returns the initial environment for the current task.

Returns

Pointer to the initial environment class.

A typical use of this function is L4Re::Env::env()-><member>()

Examples:

examples/clntsrv/client.cc, examples/libs/l4re/c++/mem_alloc/ma+rm.cc, examples/libs/l4re/c++/shared_ds/ds_clnt.cc, examples/libs/l4re/c++/shared_ds/ds_srv.cc, examples/libs/l4re/streammap/client.cc, and examples/sys/migrate/thread_migrate.cc.

Definition at line 103 of file env.

Referenced by L4virtio::Svr::Dev_config::Dev_config().

Here is the caller graph for this function:

12.217.2.2 L4::Cap<L4::Factory> L4Re::Env::factory ( ) const throw) [inline]

Object-capability to the factory object available to the task.

Returns

Factory object-capability

Definition at line 151 of file env.

References l4re_env_t::factory.

12.217.2.3 void L4Re::Env::factory (L4::Cap< L4::Factory > const & *c*) throw) [inline]

Set factory object-capability.

Parameters

c	Factory object-capability

Definition at line 256 of file env.

References l4re_env_t::factory.

12.217.2.4 l4_cap_idx_t L4Re::Env::first_free_cap ( ) const throw) [inline]

First available capability selector.

Returns

First capability selector.

First capability selector available for use for in the application.

Definition at line 159 of file env.

References l4re_env_t::first_free_cap.

12.217.2.5 void L4Re::Env::first_free_cap (l4_cap_idx_t *c*) throw) [inline]

Set first available capability selector.

Parameters

c	First capability selector available to the application.

Definition at line 262 of file env.

References l4re_env_t::first_free_cap.

12.217.2.6 l4_addr_t L4Re::Env::first_free_utcb ( ) const throw) [inline]

First free UTCB.

Returns

object-capability

First free UTCB within the UTCB area available for the application to use.

Definition at line 174 of file env.

References l4re_env_t::first_free_utcb.

12.217.2.7 void L4Re::Env::first_free_utcb (l4_addr_t *u*) throw) [inline]

Set first free UTCB.

Parameters

u	First UTCB available for the application to use.

Definition at line 274 of file env.

References l4re_env_t::first_free_utcb.

12.217.2.8 Cap_entry const* L4Re::Env::get (char const * *name*, unsigned *l*) const throw) [inline]

Get the Cap_entry for the object named *name*.

Parameters

name	is the name of the object.
l	is the length of the name, thus *name* might not be zero terminated.

Returns

A pointer to the Cap_entry for the object named *name*, or NULL if no such object was found.

Definition at line 192 of file env.

References l4re_env_get_cap_l().

Here is the call graph for this function:

12.217.2.9 template<typename T > L4::Cap<T> L4Re::Env::get_cap (char const * *name*, unsigned *l*) const throw) [inline]

Get the capability selector for the object named *name*.

Parameters

name	is the name of the object.
l	is the length of the name, thus *name* might not be zero terminated.

Returns

A capability selector for the object named *name*, or an invalid capability selector if no such object was found.

Examples:

examples/clntsrv/client.cc, examples/libs/l4re/c++/shared_ds/ds_clnt.cc, and examples/libs/l4re/streammap/client.cc.

Definition at line 204 of file env.

References L4_ENOENT.

12.217.2.10 template<typename T > L4::Cap<T> L4Re::Env::get_cap (char const * *name*) const throw) [inline]

Get the capability selector for the object named *name*.

Parameters

name	is the name of the object (zero terminated).

Returns

A capability selector for the object named *name*, or an invalid capability selector if no such object was found.

Definition at line 219 of file env.

12.217.2.11 Cap_entry const* L4Re::Env::initial_caps ( ) const throw) [inline]

Get a pointer to the first entry in the initial objects array.

Returns

A pointer to the first entry in the initial objects array.

Definition at line 181 of file env.

12.217.2.12 void L4Re::Env::initial_caps (Cap_entry * *first*) throw) [inline]

Set the pointer to the first Cap_entry in the initial objects array.

Parameters

first	is the first element in the array.

Definition at line 297 of file env.

12.217.2.13 L4::Cap<Log> L4Re::Env::log ( ) const throw) [inline]

Object-capability to the logging service.

Returns

Log object-capability

Definition at line 133 of file env.

References l4re_env_t::log.

12.217.2.14 void L4Re::Env::log (L4::Cap< Log > const & *c*) throw) [inline]

Set log object-capability.

Parameters

c	Log object-capability

Definition at line 244 of file env.

References l4re_env_t::log.

12.217.2.15 L4::Cap<L4::Thread> L4Re::Env::main_thread ( ) const throw) [inline]

Object-capability of the first user thread.

Returns

Object-capability of the first user thread.

Definition at line 139 of file env.

References l4re_env_t::main_thread.

12.217.2.16 void L4Re::Env::main_thread (L4::Cap< L4::Thread > const & *c*) throw) [inline]

Set object-capability of first user thread.

Parameters

c	First thread's object-capability

Definition at line 250 of file env.

References l4re_env_t::main_thread.

12.217.2.17 L4::Cap<Mem_alloc> L4Re::Env::mem_alloc ( ) const throw) [inline]

Object-capability to the memory allocator.

Returns

Memory allocator object-capability

Examples:

examples/libs/l4re/c++/shared_ds/ds_srv.cc.

Definition at line 116 of file env.

References l4re_env_t::mem_alloc.

12.217.2.18 void L4Re::Env::mem_alloc (L4::Cap< Mem_alloc > const & *c*) throw) [inline]

Set memory allocator object-capability.

Parameters

c	Memory allocator object-capability

Definition at line 232 of file env.

References l4re_env_t::mem_alloc.

12.217.2.19 L4::Cap<Parent> L4Re::Env::parent ( ) const throw) [inline]

Object-capability to the parent.

Returns

Parent object-capability

Definition at line 110 of file env.

References l4re_env_t::parent.

12.217.2.20 void L4Re::Env::parent (L4::Cap< Parent > const & *c*) throw) [inline]

Set parent object-capability.

Parameters

c	Parent object-capability

Definition at line 226 of file env.

References l4re_env_t::parent.

12.217.2.21 L4::Cap<Rm> L4Re::Env::rm ( ) const throw) [inline]

Object-capability to the region map.

Returns

Region map object-capability

Examples:

examples/libs/l4re/c++/shared_ds/ds_clnt.cc, and examples/libs/l4re/c++/shared_ds/ds_srv.cc.

Definition at line 127 of file env.

References l4re_env_t::rm.

12.217.2.22 void L4Re::Env::rm (L4::Cap< Rm > const & *c*) throw) [inline]

Set region map object-capability.

Parameters

c	Region map object-capability

Definition at line 238 of file env.

References l4re_env_t::rm.

12.217.2.23 L4::Cap<L4::Scheduler> L4Re::Env::scheduler ( ) const throw) [inline]

Get the scheduler capability for the task.

Returns

The capability selector for the default scheduler used for this task.

Examples:

examples/sys/migrate/thread_migrate.cc.

Definition at line 282 of file env.

References l4re_env_t::scheduler.

12.217.2.24 void L4Re::Env::scheduler (L4::Cap< L4::Scheduler > const & *c*) throw) [inline]

Set the scheduler capability.

Parameters

c	is the capability to be set as scheduler.

Definition at line 289 of file env.

References l4re_env_t::scheduler.

12.217.2.25 L4::Cap<L4::Task> L4Re::Env::task ( ) const throw) [inline]

Object-capability of the user task.

Returns

Object-capability of the user task.

Definition at line 145 of file env.

References L4_BASE_TASK_CAP.

12.217.2.26 l4_fpage_t L4Re::Env::utcb_area ( ) const throw) [inline]

UTCB area of the task.

Returns

UTCB area

Definition at line 165 of file env.

References l4re_env_t::utcb_area.

12.217.2.27 void L4Re::Env::utcb_area (l4_fpage_t *utcbs*) throw) [inline]

Set UTCB area of the task.

Parameters

utcbs	UTCB area

Definition at line 268 of file env.

References l4re_env_t::utcb_area.

The documentation for this class was generated from the following file:

- l4/re/env

12.218 L4Re::Event Class Reference

Event class.

Inheritance diagram for L4Re::Event:

Collaboration diagram for L4Re::Event:

Public Member Functions

- long get_buffer (L4::Ipc::Out< L4::Cap< Dataspace > > ds)

  Get event signal buffer.

Additional Inherited Members

12.218.1 Detailed Description

Event class.

Definition at line 133 of file event.

12.218.2 Member Function Documentation

12.218.2.1 long L4Re::Event::get_buffer (L4::Ipc::Out< L4::Cap< Dataspace > > *ds*)

Get event signal buffer.

Return values

ds	Event buffer.

Returns

0 on success, negative error code otherwise.

Referenced by L4Re::Util::Event_t< PAYLOAD >::init(), and L4Re::Util::Event_t< PAYLOAD >::init_poll().

Here is the caller graph for this function:

The documentation for this class was generated from the following file:

- l4/re/event

12.219 L4Re::Event_buffer_t< PAYLOAD > Class Template Reference

Event buffer class.

Inheritance diagram for L4Re::Event_buffer_t< PAYLOAD >:

Collaboration diagram for L4Re::Event_buffer_t< PAYLOAD >:

Data Structures

- struct Event

  Event structure used in buffer.

Public Member Functions

- Event_buffer_t (void *buffer, l4_addr_t size)

  Initialize event buffer.
- Event * next () throw ()

  Next event in buffer.
- bool put (Event const &ev) throw ()

  Put event into buffer at current position.

12.219.1 Detailed Description

template<typename PAYLOAD = Default_event_payload>class L4Re::Event_buffer_t< PAYLOAD >

Event buffer class.

Definition at line 187 of file event.

12.219.2 Constructor & Destructor Documentation

12.219.2.1 template<typename PAYLOAD = Default_event_payload> L4Re::Event_buffer_t< PAYLOAD >::Event_buffer_t (void * *buffer*, l4_addr_t *size*) [inline]

Initialize event buffer.

Parameters

buffer	Pointer to buffer.
size	Size of buffer in bytes.

Definition at line 234 of file event.

12.219.3 Member Function Documentation

12.219.3.1 template<typename PAYLOAD = Default_event_payload> Event* L4Re::Event_buffer_t< PAYLOAD >::next ( ) throw) [inline]

Next event in buffer.

Returns

0 if no event available, event otherwise.

Definition at line 244 of file event.

References L4Re::Event_buffer_t< PAYLOAD >::Event::time.

12.219.3.2 template<typename PAYLOAD = Default_event_payload> bool L4Re::Event_buffer_t< PAYLOAD >::put (Event const & *ev*) throw) [inline]

Put event into buffer at current position.

Parameters

ev	Event to put into the buffer.

Returns

false if buffer is full and entry could not be added.

Definition at line 261 of file event.

References l4_wmb(), and L4Re::Event_buffer_t< PAYLOAD >::Event::time.

Here is the call graph for this function:

The documentation for this class was generated from the following file:

- l4/re/event

12.220 L4Re::Event_buffer_t< PAYLOAD >::Event Struct Reference

Event structure used in buffer.

Collaboration diagram for L4Re::Event_buffer_t< PAYLOAD >::Event:

Public Member Functions

- void free () throw ()

  Free the entry.

Data Fields

- long long time

  Event time stamp.

12.220.1 Detailed Description

template<typename PAYLOAD = Default_event_payload>struct L4Re::Event_buffer_t< PAYLOAD >::Event

Event structure used in buffer.

Definition at line 194 of file event.

The documentation for this struct was generated from the following file:

- l4/re/event

12.221 L4Re::Inhibitor Class Reference

Set of inhibitor locks, which inhibit specific actions when held.

Inheritance diagram for L4Re::Inhibitor:

Collaboration diagram for L4Re::Inhibitor:

Public Types

- enum { Name_max = 20 }

Public Member Functions

- long acquire (l4_umword_t id, L4::Ipc::String<> reason)

  Acquire a specific inhibitor lock.
- long release (l4_umword_t id)

  Release a specific inhibitor lock.
- long next_lock_info (char *name, unsigned len, l4_mword_t current_id=-1, l4_utcb_t *utcb=l4_utcb())

  Get information for the next available inhibitor lock.

Additional Inherited Members

12.221.1 Detailed Description

Set of inhibitor locks, which inhibit specific actions when held.

This interface provides access to a set of inhibitor locks, each determined by an ID that is specific to the Inhibitor object. Each individual lock shall prevent, a specific (implementation defined) action to be executed, as long as the lock is held.

For example there can be an inhibitor lock to prevent a transition to suspend-to-RAM state and a different one to prevent shutdown.

A client shall take an inhibitor lock if it needs to execute code before the action is taken. For example a lock-screen application shall grab an inhibitor lock for the suspend action to be able to lock the screen before the system goes to sleep.

Inhibitor locks are usually closely related to specific events. Usually a server automatically subscribes a client holding a lock to the corresponding event. The server shall send the event to inform the client that an action is pending. Upon reception of the event, the client is supposed to release the corresponding inhibitor lock.

Definition at line 40 of file inhibitor.

12.221.2 Member Enumeration Documentation

12.221.2.1 anonymous enum

Enumerator

Name_max The maximum length of a lock's name.

Definition at line 44 of file inhibitor.

12.221.3 Member Function Documentation

12.221.3.1 long L4Re::Inhibitor::acquire (l4_umword_t *id*, L4::Ipc::String<> *reason*)

Acquire a specific inhibitor lock.

Parameters

id	ID of the inhibitor lock that the client intends to acquire
reason	The reason why you need the lock. Used for informing the user or debugging.

Return values

0	Success
-L4_ENODEV	The specified `id` does not exist.

12.221.3.2 long L4Re::Inhibitor::next_lock_info (char * *name*, unsigned *len*, l4_mword_t *current_id* = -1, l4_utcb_t * *utcb* = l4_utcb()) [inline]

Get information for the next available inhibitor lock.

Parameters

name	A pointer to a buffer for the name of the lock.
len	The length of the available buffer (usually Name_max is used)
current_id	The ID of the last available lock, use -1 to get the first lock.
utcb	The UTCB to use for the message.

Return values

>0	The ID of the next available lock if there is one (in this case `name` shall contain the name of the inhibitor lock).
-L4_ENODEV	There are no more locks.

Definition at line 86 of file inhibitor.

References L4::Kobject_t< Inhibitor, L4::Kobject, L4RE_PROTO_INHIBITOR >::c().

Here is the call graph for this function:

12.221.3.3 long L4Re::Inhibitor::release (l4_umword_t *id*)

Release a specific inhibitor lock.

Parameters

id	The ID of the inhibitor lock to release.

Return values

0	Success
-L4_ENODEV	Lock with the given `id` does not exist.

The documentation for this class was generated from the following file:

- l4/re/inhibitor

12.222 L4Re::Log Class Reference

Log interface class.

Inheritance diagram for L4Re::Log:

Collaboration diagram for L4Re::Log:

Public Member Functions

- void printn (char const ∗string, int len) const throw ()

  Print string with length len, NULL characters don't matter.

- void print (char const ∗string) const throw ()

  Print NULL-terminated string.

Additional Inherited Members

12.222.1 Detailed Description

Log interface class.

Definition at line 44 of file log.

12.222.2 Member Function Documentation

12.222.2.1 void L4Re::Log::print (char const * *string*) const throw)

Print NULL-terminated string.

Parameters

string	string to print

12.222.2.2 void L4Re::Log::printn (char const * *string*, int *len*) const throw)

Print string with length len, NULL characters don't matter.

Parameters

string	string to print
len	length of string

The documentation for this class was generated from the following file:

- l4/re/log

12.223 L4Re::Mem_alloc Class Reference

Memory allocation interface.

Inheritance diagram for L4Re::Mem_alloc:

Collaboration diagram for L4Re::Mem_alloc:

Public Types

- enum Mem_alloc_flags { Continuous = 0x01, Pinned = 0x02, Super_pages = 0x04 }

  Flags for the allocator.

Public Member Functions

- long alloc (long size, L4::Cap< Dataspace > mem, unsigned long flags=0, unsigned long align=0) const throw ()

  Allocate anonymous memory.
- long free (L4::Cap< Dataspace > mem) const throw ()

  Free dataspace.

Additional Inherited Members

12.223.1 Detailed Description

Memory allocation interface.

The memory-allocator API is the basic API to allocate memory from the L4Re subsystem. The memory is allocated in terms of dataspaces (see L4Re::Dataspace). The provided dataspaces have at least the property that data written to such a dataspace is available as long as the dataspace is not freed or the data is not overwritten. In particular, the memory backing a dataspace from an allocator need not be allocated instantly, but may be allocated lazily on demand.

A memory allocator can provide dataspaces with additional properties, such as physically contiguous memory, pre-allocated memory, or pinned memory. To request memory with an additional property the L4Re::Mem_alloc::alloc() method provides a flags parameter. If the concrete implementation of a memory allocator does not support or allow allocation of memory with a certain property, the allocation may be refused.

Definition at line 61 of file mem_alloc.

12.223.2 Member Enumeration Documentation

12.223.2.1 enum L4Re::Mem_alloc::Mem_alloc_flags

Flags for the allocator.

They describe requested properties of the allocated memory. Support of these properties by the dataspace provider is optional.

Enumerator

Continuous Allocate physically contiguous memory.

Pinned Deprecated, use L4Re::Dma_space instead.

Super_pages Allocate super pages.

Definition at line 71 of file mem_alloc.

12.223.3 Member Function Documentation

12.223.3.1 long L4Re::Mem_alloc::alloc (long *size*, L4::Cap< Dataspace > *mem*, unsigned long *flags* = 0, unsigned long *align* = 0) const throw)

Allocate anonymous memory.

Parameters

	size	Size in bytes to be requested (granularity is (super)pages and the size is rounded up to this granularity). If `size` is a negative value and `flags` set the Mem_alloc_flags::Continuous bit the allocator tries to allocate as much memory as possible leaving an amount of at least `-size` bytes within the associated quota.
`out`	*mem*	Capability slot where the capability to the dataspace is received.
	flags	Special dataspace properties, see Mem_alloc_flags
	align	Log2 alignment of dataspace if supported by allocator, will be at least L4_PAGESHIFT, with Super_pages flag set at least L4_SUPERPAGESHIFT

Return values

0	Success
-L4_ERANGE	Given size not supported.
-L4_ENOMEM	Not enough memory available.
<0	IPC error

Definition at line 35 of file mem_alloc_impl.h.

References L4::Factory::create(), and l4_error().

Here is the call graph for this function:

12.223.3.2 long L4Re::Mem_alloc::free (L4::Cap< Dataspace > *mem*) const throw)

Free dataspace.

Parameters

mem	Dataspace to be released.

Return values

0	

Deprecated This function is an empty stub which remains here for backward compatibility only. Use `L4::Task::unmap(mem, L4_FP_DELETE_OBJ)` to remove a dataspace.

Definition at line 47 of file mem_alloc_impl.h.

The documentation for this class was generated from the following files:

- l4/re/mem_alloc
- l4/re/impl/mem_alloc_impl.h

12.224 L4Re::Namespace Class Reference

Name-space interface.

Inheritance diagram for L4Re::Namespace:

Collaboration diagram for L4Re::Namespace:

Public Types

- enum Register_flags {
  Ro = L4_CAP_FPAGE_RO, Rw = L4_CAP_FPAGE_RW, Rs = L4_CAP_FPAGE_RS, Rws = L4_CAP_FP↩AGE_RWS,
  Strong = L4_CAP_FPAGE_S, Trusted = 0x008 , Link = 0x100, Overwrite = 0x200 }

  Flags for registering name spaces.
- enum Query_result_flags { Partly_resolved = 0x020 }

  Flags returned by query IPC, only used internally.

Public Member Functions

- long query (char const ∗name, L4::Cap< void > const &cap, int timeout=To_default, l4_umword_t ∗local_↩id=0, bool iterate=true) const throw ()

  Query the name space for a named object.
- long query (char const ∗name, unsigned len, L4::Cap< void > const &cap, int timeout=To_default, l4_↩umword_t ∗local_id=0, bool iterate=true) const throw ()

Query the name space for a named object.

- long register_obj (char const *name, L4::Ipc::Cap< void > obj, unsigned flags=Rw) const throw ()

  Register an object with a name.

- long unlink (char const *name)

  Remove an entry from the name space.

Additional Inherited Members

12.224.1 Detailed Description

Name-space interface.

All name space objects must provide this interface. However, it is not mandatory that a name space object allows to register new capabilities.

The name lookup is done iteratively, this means the hierarchical names are resolved component wise by the client itself.

Definition at line 60 of file namespace.

12.224.2 Member Function Documentation

12.224.2.1 long L4Re::Namespace::query (char const * *name*, L4::Cap< void > const & *cap*, int *timeout* = `To_default`, l4_umword_t * *local_id* = `0`, bool *iterate* = `true`) const throw)

Query the name space for a named object.

Parameters

in	*name*	String to query (without any leading slashes).
in	*cap*	Capability slot where the received capability will be put.
in	*timeout*	Timeout of query in milliseconds. The client will only wait if a name has already been registered with the server but no object has yet been attached.
out	*local_id*	If given, L4_RCV_ITEM_LOCAL_ID will be set for the IPC from the name space, so that if the capability that was received is a local item, the capability ID will be returned with this parameter.
in	*iterate*	If true, the client will try to resolve names by iteratively calling the name spaces until the name is fully resolved.

Return values

0	Name could be fully resolved.
>0	Name could only be partly resolved. The number of remaining characters is returned.
-L4_ENOENT	Entry could not be found.
-L4_EAGAIN	Entry exists but no object is yet attached. Try again later.
<0	IPC errors, see l4_error_code_t.

Definition at line 118 of file namespace_impl.h.

12.224.2.2 long L4Re::Namespace::query (char const * *name*, unsigned *len*, L4::Cap< void > const & *cap*, int *timeout* = `To_default`, l4_umword_t * *local_id* = `0`, bool *iterate* = `true`) const throw)

Query the name space for a named object.

The query string does not necessarily need to be null-terminated.

Parameters

in	*len*	Length of the string to query without any terminating null characters.
in	*name*	String to query (without any leading slashes).
in	*cap*	Capability slot where the received capability will be put.
in	*timeout*	Timeout of query in milliseconds. The client will only wait if a name has already been registered with the server but no object has yet been attached.
out	*local_id*	If given, L4_RCV_ITEM_LOCAL_ID will be set for the IPC from the name space, so that if the capability that was received is a local item, the capability ID will be returned with this parameter.
in	*iterate*	If true, the client will try to resolve names by iteratively calling the name spaces until the name is fully resolved.

Return values

0	Name could be fully resolved.
>0	Name could only be partly resolved. The number of remaining characters is returned.
-L4_ENOENT	Entry could not be found.
-L4_EAGAIN	Entry exists but no object is yet attached. Try again later.
<0	IPC errors, see l4_error_code_t.

Definition at line 77 of file namespace_impl.h.

References L4_EAGAIN, L4_EINVAL, l4_sleep(), and L4_UNLIKELY.

Here is the call graph for this function:

12.224.2.3 long L4Re::Namespace::register_obj (char const * *name*, L4::Ipc::Cap< void > *obj*, unsigned *flags* = Rw) const throw) [inline]

Register an object with a name.

Parameters

name	Name under which the object should be registered.
obj	Capability to object to register. An invalid capability may be given to only reserve the name for later use.
flags	Flags to assign to the entry, see L4Re::Namespace::Register_flags. Note that the rights that are assigned to a capability are not only determined by the rights given in these flags but also by the rights with which the `obj` capability was mapped to the name space.

Return values

0	Object was successfully registered with *name*.

-L4_EEXIST	Name already registered.
-L4_EPERM	Caller doesn't have necessary permissions.
-L4_ENOMEM	Server has insufficient resources.
-L4_EINVAL	Invalid parameter.
<0	IPC errors, see l4_error_code_t.

Precondition

requires capability rights: {RW}

Definition at line 176 of file namespace.

12.224.2.4 long L4Re::Namespace::unlink (char const * *name*) [inline]

Remove an entry from the name space.

Parameters

name	Name of the entry to remove.

Return values

0	Entry successfully removed.
-L4_ENOENT	Given name does not exist.
-L4_EPERM	Caller does not have write permission.
-L4_EACCESS	Name cannot be removed.
<0	IPC errors, see l4_error_code_t.

Precondition

requires capability rights: {RW}

Definition at line 202 of file namespace.

The documentation for this class was generated from the following files:

- l4/re/namespace
- l4/re/impl/namespace_impl.h

12.225 L4Re::Parent Class Reference

Parent interface.

Inheritance diagram for L4Re::Parent:

Collaboration diagram for L4Re::Parent:

Public Member Functions

- long signal (unsigned long sig, unsigned long val)

  Send a signal to the parent.

Additional Inherited Members

12.225.1 Detailed Description

Parent interface.

See also

Parent API for more details about the purpose.

Definition at line 51 of file parent.

12.225.2 Member Function Documentation

12.225.2.1 long L4Re::Parent::signal (unsigned long *sig*, unsigned long *val*)

Send a signal to the parent.

Parameters

sig	Signal to send
val	Value of the signal

Returns

0 on success, <0 on error

- -L4_ENOREPLY
- IPC errors

The documentation for this class was generated from the following file:

- l4/re/parent

12.226 L4Re::Rm Class Reference

Region map.

```
#include <l4/re/rm>
```

Inheritance diagram for L4Re::Rm:

Collaboration diagram for L4Re::Rm:

Public Types

- enum Detach_result { Detached_ds = 0, Kept_ds = 1, Split_ds = 2 , Detach_again = 4 }

  Result values for detach operation.
- enum Region_flags {
  Read_only = 0x01, Detach_free = 0x02, Pager = 0x04, Reserved = 0x08,
  Caching_shift = 8, Caching_ds_shift = Caching_shift - Dataspace::Map_caching_shift, Caching =

Dataspace::Map_caching_mask << Caching_ds_shift, Cache_normal = Dataspace::Map_normal << Caching_ds_shift,
Cache_buffered = Dataspace::Map_bufferable << Caching_ds_shift, Cache_uncached = Dataspace::Map↩_uncacheable << Caching_ds_shift, Region_flags = Caching | 0x0f }

Flags for regions.

- enum Attach_flags { Search_addr = 0x20, In_area = 0x40, Eager_map = 0x80, Attach_flags = 0xf0 }

  Flags for attach operation.

- enum Detach_flags { Detach_exact = 1, Detach_overlap = 2, Detach_keep = 4 }

  Flags for detach operation.

Public Member Functions

- long reserve_area (l4_addr_t *start, unsigned long size, unsigned flags=0, unsigned char align=L4_PAGE↩SHIFT) const throw ()

  Reserve the given area in the region map.

- template<typename T >
  long reserve_area (T **start, unsigned long size, unsigned flags=0, unsigned char align=L4_PAGESHIFT) const throw ()

  Reserve the given area in the region map.

- long free_area (l4_addr_t addr)

  Free an area from the region map.

- long attach (l4_addr_t *start, unsigned long size, unsigned long flags, L4::Ipc::Cap< Dataspace > mem, l4_addr_t offs=0, unsigned char align=L4_PAGESHIFT) const throw ()

  Attach a data space to a region.

- template<typename T >
  long attach (T **start, unsigned long size, unsigned long flags, L4::Ipc::Cap< Dataspace > mem, l4_addr_t offs=0, unsigned char align=L4_PAGESHIFT) const throw ()

  Attach a data space to a region.

- int detach (l4_addr_t addr, L4::Cap< Dataspace > *mem, L4::Cap< L4::Task > const &task=This_task) const throw ()

  Detach a region from the address space.

- int detach (void *addr, L4::Cap< Dataspace > *mem, L4::Cap< L4::Task > const &task=This_task) const throw ()

  Detach a region from the address space.

- int detach (l4_addr_t start, unsigned long size, L4::Cap< Dataspace > *mem, L4::Cap< L4::Task > const &task) const throw ()

  Detach all regions of the specified interval.

- int find (l4_addr_t *addr, unsigned long *size, l4_addr_t *offset, unsigned *flags, L4::Cap< Dataspace > *m) throw ()

  Find a region given an address and size.

Additional Inherited Members

12.226.1 Detailed Description

Region map.

Definition at line 69 of file rm.

12.226.2 Member Function Documentation

12.226.2.1 long L4Re::Rm::attach (l4_addr_t * *start*, unsigned long *size*, unsigned long *flags*, L4::Ipc::Cap< Dataspace > *mem*, l4_addr_t *offs* = 0, unsigned char *align* = L4_PAGESHIFT) const throw)

Attach a data space to a region.

Parameters

in,out	*start*	Virtual start address where the region manager shall attach the data space. If L4Re::Rm::Search_addr is given this value is used as the start address to search for a free virtual memory region and the resulting address is returned here. If L4Re::Rm::In_area is given the value is used as a selector for the area (see L4Re::Rm::reserve_area) to attach the data space to.
	size	Size of the data space to attach (in bytes)
	flags	Flags, see L4Re::Rm::Attach_flags and L4Re::Rm::Region_flags
	mem	Data space
	offs	Offset into the data space to use
	align	Alignment of the virtual region, log2-size, default: a page (L4_PAGESHIFT). This is only meaningful if the L4Re::Rm::Search_addr flag is used.

Return values

0	Success
-L4_ENOENT	No area could be found (see L4Re::Rm::In_area)
-L4_EPERM	Operation not allowed.
-L4_EINVAL	
-L4_EADDRNOTAVAIL	The given address is not available.
<0	IPC errors

Makes the whole or parts of a data space visible in the virtual memory of the corresponding task. The corresponding region in the virtual address space is backed with the contents of the dataspace.

Note

When searching for a free place in the virtual address space, the space between *start* and the end of the virtual address space is searched.
There is no region object created, instead the region is defined by a virtual address within this range (see L4Re::Rm::find).

Definition at line 43 of file rm_impl.h.

References L4::Ipc::Cap< T >::cap(), L4Re::Dataspace::map_region(), L4Re::Dataspace::Map_ro, and L4Re::Dataspace::Map_rw.

Here is the call graph for this function:

12.226.2.2 template<typename T > long L4Re::Rm::attach (T ** *start*, unsigned long *size*, unsigned long *flags*, L4::Ipc::Cap< Dataspace > *mem*, l4_addr_t *offs* = 0, unsigned char *align* = L4_PAGESHIFT) const throw) [inline]

Attach a data space to a region.

Parameters

in,out	*start*	Virtual start address where the region manager shall attach the data space. If L4Re::Rm::Search_addr is given this value is used as the start address to search for a free virtual memory region and the resulting address is returned here. If L4Re::Rm::In_area is given the value is used as a selector for the area (see L4Re::Rm::reserve_area) to attach the data space to.
	size	Size of the data space to attach (in bytes)
	flags	Flags, see L4Re::Rm::Attach_flags and L4Re::Rm::Region_flags
	mem	Data space
	offs	Offset into the data space to use
	align	Alignment of the virtual region, log2-size, default: a page (L4_PAGESHIFT). This is only meaningful if the L4Re::Rm::Search_addr flag is used.

Return values

0	Success
-L4_ENOENT	No area could be found (see L4Re::Rm::In_area)
-L4_EPERM	Operation not allowed.
-L4_EINVAL	
-L4_EADDRNOTAVAIL	The given address is not available.
<0	IPC errors

Makes the whole or parts of a data space visible in the virtual memory of the corresponding task. The corresponding region in the virtual address space is backed with the contents of the dataspace.

Note

When searching for a free place in the virtual address space, the space between *start* and the end of the virtual address space is searched.
There is no region object created, instead the region is defined by a virtual address within this range (see L4Re::Rm::find).

Definition at line 339 of file rm.

12.226.2.3 int L4Re::Rm::detach (l4_addr_t *addr*, L4::Cap< Dataspace > * *mem*, L4::Cap< L4::Task > const & *task* = This_task) const throw) [inline]

Detach a region from the address space.

Parameters

	addr	Virtual address of region, any address within the region is valid.
out	*mem*	Dataspace that is affected. Give 0 if not interested.
	task	This argument specifies the task where the pages are unmapped. Provide L4::Cap<L4::Task>::Invalid for none. The default is the current task.

Return values

Detach_result	On success.
-L4_ENOENT	No region found.
<0	IPC errors

Frees a region in the virtual address space given by addr (address type). The corresponding part of the address space is now available again.

Definition at line 491 of file rm.

12.226.2.4 int L4Re::Rm::detach (void * *addr*, L4::Cap< Dataspace > * *mem*, L4::Cap< L4::Task > const & *task* = This_task) const throw) [inline]

Detach a region from the address space.

Parameters

	addr	Virtual address of region, any address within the region is valid.
out	*mem*	Dataspace that is affected. Give 0 if not interested.
	task	This argument specifies the task where the pages are unmapped. Provide L4::Cap<L4::Task>::Invalid for none. The default is the current task.

Return values

Detach_result	On success.
-L4_ENOENT	No region found.
<0	IPC errors

Frees a region in the virtual address space given by addr (address type). The corresponding part of the address space is now available again.

Definition at line 496 of file rm.

12.226.2.5 int L4Re::Rm::detach (l4_addr_t *start*, unsigned long *size*, L4::Cap< Dataspace > * *mem*, L4::Cap< L4::Task > const & *task*) const throw) [inline]

Detach all regions of the specified interval.

Parameters

	start	Start of area to detach, must be within region.
	size	Size of of area to detach (in bytes).
out	*mem*	Dataspace that is affected. Give 0 if not interested.
	task	This argument specifies the task where the pages are unmapped. Provide L4::Cap<L4::Task>::Invalid for none. The default is the current task.

Return values

Detach_result	On success.
-L4_ENOENT	No region found.
<0	IPC errors

Frees all regions within the interval given by start and size. If a region overlaps the start or the end of the interval this region is only detached partly. If the interval is within one region the original region is split up into two separate regions.

Definition at line 501 of file rm.

12.226.2.6 int L4Re::Rm::find (l4_addr_t * *addr*, unsigned long * *size*, l4_addr_t * *offset*, unsigned * *flags*, L4::Cap< Dataspace > * *m*) throw) [inline]

Find a region given an address and size.

Parameters

	addr	Address to look for
	size	Size of the area to look for (in bytes).
out	*addr*	Start address of the found region.
out	*size*	Size of the found region (in bytes).
out	*offset*	Offset at the beginning of the region within the associated dataspace.
out	*flags*	Region flags, see Region_flags (and In_area).
out	*m*	Associated dataspace or paging service.

Return values

0	Success
-L4_EPERM	Operation not allowed.
-L4_ENOENT	No region found.
<0	IPC errors

This function returns the properties of the region that contains the area described by the addr and size parameter. If no such region is found but a reserved area, the area is returned and In_area is set in 'flags'. Note, in the case of an area the 'offset' and 'm' return values are invalid.

```
                size-out
              /           \
             /             \
         addr-out           \
            ^_________________\
------------|----------------|-------------------
|           | Region         |       Dataspace |
|           |_______|___|____|                 |
------------|-------|---|-----------------------
 \         /        |\ /
  \       /         | |> size-in
  offset-out        |
                    |> addr-in
```

Note

The value of the size input parameter should be 1 to assure that a region can be determined unambiguously.

Definition at line 455 of file rm.

12.226.2.7 long L4Re::Rm::free_area (l4_addr_t *addr*)

Free an area from the region map.

Parameters

addr	An address within the area to free.

Return values

0	Success
-L4_ENOENT	No area found.
<0	IPC errors

Note

The data spaces that are attached to that area are not detached by this operation.

See also

reserve_area() for more information about areas.

12.226.2.8 long L4Re::Rm::reserve_area (l4_addr_t * *start*, unsigned long *size*, unsigned *flags* = 0, unsigned char *align* = L4_PAGESHIFT) const throw) [inline]

Reserve the given area in the region map.

Parameters

in,out	*start*	The virtual start address of the area to reserve. Returns the start address of the area.
	size	The size of the area to reserve (in bytes).
	flags	Flags for the reserved area (see L4Re::Rm::Region_flags and L4Re::Rm::↩ Attach_flags).
	align	Alignment of area if searched as bits (log2 value).

Return values

0	Success
-L4_EADDRNOTAVAIL	The given area cannot be reserved.
<0	IPC errors

This function reserves an area within the virtual address space implemented by the region map. There are two kinds of areas available:

- Reserved areas (*flags* = Reserved), where no data spaces can be attached
- Special purpose areas (*flags* = 0), where data spaces can be attached to the area via the In_area flag and a start address within the area itself.

Note

When searching for a free place in the virtual address space (with *flags* = Search_addr), the space between *start* and the end of the virtual address space is searched.

Definition at line 236 of file rm.

12.226.2.9 template<typename T > long L4Re::Rm::reserve_area (T ** *start*, unsigned long *size*, unsigned *flags* = 0, unsigned char *align* = L4_PAGESHIFT) const throw) [inline]

Reserve the given area in the region map.

Parameters

in,out	*start*	The virtual start address of the area to reserve. Returns the start address of the area.
	size	The size of the area to reserve (in bytes).
	flags	Flags for the reserved area (see Region_flags and Attach_flags).
	align	Alignment of area if searched as bits (log2 value).

Return values

0	Success
-L4_EADDRNOTAVAIL	The given area cannot be reserved.
<0	IPC errors

For more information, please refer to the analogous function

See also

L4Re::Rm::reserve_area.

Definition at line 262 of file rm.

The documentation for this class was generated from the following files:

- l4/re/rm
- l4/re/impl/rm_impl.h

12.227 L4Re::Smart_cap_auto< Unmap_flags > Class Template Reference

Helper for Auto_cap and Auto_del_cap.

Collaboration diagram for L4Re::Smart_cap_auto< Unmap_flags >:

12.227.1 Detailed Description

template<unsigned long Unmap_flags = L4_FP_ALL_SPACES>class L4Re::Smart_cap_auto< Unmap_flags >

Helper for Auto_cap and Auto_del_cap.

Definition at line 110 of file cap_alloc.

The documentation for this class was generated from the following file:

- l4/re/cap_alloc

12.228 L4Re::Util::Auto_cap< T > Struct Template Reference

Automatic capability that implements automatic free and unmap of the capability selector.

Inheritance diagram for L4Re::Util::Auto_cap< T >:

Collaboration diagram for L4Re::Util::Auto_cap< T >:

12.228.1 Detailed Description

template<typename T>struct L4Re::Util::Auto_cap< T >

Automatic capability that implements automatic free and unmap of the capability selector.

Template Parameters

T	Type of the object that is referred by the capability.

This kind of automatic capability is useful for capabilities that shall have a lifetime that is strictly coupled to one C++ scope.

Usage:

```
{
  L4Re::Util::Auto_cap<L4Re::Dataspace>::Cap
    ds_cap(L4Re::Util::cap_alloc.alloc<L4Re::Dataspace>());

  // use the dataspace cap
  L4Re::chksys(mem_alloc->alloc(4096, ds_cap.get()));

  ...
```

```
  // At the end of the scope ds_cap is unmapped and the capability selector
  // is freed.
}
```

Definition at line 161 of file cap_alloc.

The documentation for this struct was generated from the following file:

- l4/re/util/cap_alloc

12.229 L4Re::Util::Auto_del_cap< T > Struct Template Reference

Automatic capability that implements automatic free and unmap+delete of the capability selector.

Inheritance diagram for L4Re::Util::Auto_del_cap< T >:

Collaboration diagram for L4Re::Util::Auto_del_cap< T >:

12.229.1 Detailed Description

template<typename T>struct L4Re::Util::Auto_del_cap< T >

Automatic capability that implements automatic free and unmap+delete of the capability selector.

Template Parameters

T	Type of the object that is referred by the capability.

This kind of automatic capability is useful for capabilities with that shall have a lifetime that is strictly coupled to one C++ scope. The main difference to Auto_cap is that the unmap is done with the deletion flag enabled and this leads to the deletion of the object if the current task holds appropriate deletion rights.

Usage:

```
{
  L4Re::Util::Auto_del_cap<L4Re::Dataspace>::Cap
    ds_cap(L4Re::Util::cap_alloc.alloc<L4Re::Dataspace>());

  // use the dataspace cap
  L4Re::chksys(mem_alloc->alloc(4096, ds_cap.get()));

  ...

  // At the end of the scope ds_cap is unmapped and the capability selector
  // is freed. Because the deletion flag is set the data space shall be
  // also deleted (even if there are other references to this data space).
}
```

Definition at line 196 of file cap_alloc.

The documentation for this struct was generated from the following file:

- l4/re/util/cap_alloc

12.230 L4Re::Util::Br_manager Class Reference

Buffer-register (BR) manager for L4::Server.

Inheritance diagram for L4Re::Util::Br_manager:

Collaboration diagram for L4Re::Util::Br_manager:

Public Member Functions

- Br_manager ()

  Make a buffer-register (BR) manager.
- int alloc_buffer_demand (Demand const &d)

  Tells the server to allocate buffers for the given demand.
- L4::Cap< void > get_rcv_cap (int i) const

  Get capability slot allocated to the given receive buffer.
- int realloc_rcv_cap (int i)

  Allocate a new capability for the given receive buffer.
- void set_rcv_cap_flags (unsigned long flags)

  Set the receive flags for the buffers.
- int add_timeout (L4::Ipc_svr::Timeout *, l4_kernel_clock_t)

  No timeouts handled by us.
- int remove_timeout (L4::Ipc_svr::Timeout *)

  No timeouts handled by us.

- void setup_wait (l4_utcb_t *utcb, L4::Ipc_svr::Reply_mode)

  setup_wait() used the server loop (L4::Server)

Protected Member Functions

- unsigned first_free_br () const

  Used for assigning BRs for a timeout.

Additional Inherited Members

12.230.1 Detailed Description

Buffer-register (BR) manager for L4::Server.

Implementation of the L4::Ipc_svr::Server_iface API for managing the server-side receive buffers needed for a set of server objects running within a server.

Definition at line 36 of file br_manager.

12.230.2 Member Function Documentation

12.230.2.1 int L4Re::Util::Br_manager::alloc_buffer_demand (Demand const & *demand*) [inline],[virtual]

Tells the server to allocate buffers for the given demand.

Parameters

demand	The total server-side demand of receive buffers needed for a given interface, see Demand.

This function is not called by user applications directly. Usually the server implementation or the registry implementation calls this function whenever a new object is registered at the server.

Implements L4::Ipc_svr::Server_iface.

Definition at line 50 of file br_manager.

References L4Re::Util::Counting_cap_alloc< COUNTERTYPE >::alloc(), L4Re::Util::cap_alloc, L4::Type_info::Demand::caps, L4_EINVAL, L4_ENOMEM, L4_EOK, L4_ERANGE, L4_UTCB_GENERIC_BUFFERS_SIZE, L4::Type_info::Demand::mem, and L4::Type_info::Demand::ports.

Here is the call graph for this function:

12.230.2.2 L4::Cap<void> L4Re::Util::Br_manager::get_rcv_cap (int *index*) const [inline],[virtual]

Get capability slot allocated to the given receive buffer.

Parameters

index	The receive buffer index of the expected capability argument (0 <= `index` < `caps` registered with alloc_buffer_demand()).

Precondition

0 <= `index` < `caps` registered with alloc_buffer_demand()

Returns

Capability slot currently allocated to the given receive buffer.

Implements L4::Ipc_svr::Server_iface.

Definition at line 81 of file br_manager.

References L4_CAP_MASK.

12.230.2.3 int L4Re::Util::Br_manager::realloc_rcv_cap (int *index*) [inline], [virtual]

Allocate a new capability for the given receive buffer.

Parameters

index	The receive buffer index of the expected capability argument (0 <= `index` < `caps` registered with alloc_buffer_demand()).

Precondition

0 <= `index` < `caps` registered with alloc_buffer_demand()

Returns

0 on success, < 0 on error.

Implements L4::Ipc_svr::Server_iface.

Definition at line 89 of file br_manager.

References L4Re::Util::Counting_cap_alloc< COUNTERTYPE >::alloc(), L4Re::Util::cap_alloc, L4_EINVAL, L4_ENOMEM, and L4_EOK.

Here is the call graph for this function:

12.230.2.4 void L4Re::Util::Br_manager::set_rcv_cap_flags (unsigned long *flags*) [inline]

Set the receive flags for the buffers.

Precondition

Must be called before any handlers are registered.

Parameters

flags	New receive capability flags, see l4_msg_item_consts_t.

Definition at line 113 of file br_manager.

The documentation for this class was generated from the following file:

- l4/re/util/br_manager

12.231 L4Re::Util::Br_manager_hooks Struct Reference

Predefined server-loop hooks for a server loop using the Br_manager.

Inheritance diagram for L4Re::Util::Br_manager_hooks:

Collaboration diagram for L4Re::Util::Br_manager_hooks:

Additional Inherited Members

12.231.1 Detailed Description

Predefined server-loop hooks for a server loop using the Br_manager.

This class can be used whenever a server loop including full management of receive buffer resources is needed.

Definition at line 160 of file br_manager.

The documentation for this struct was generated from the following file:

- l4/re/util/br_manager

12.232 L4Re::Util::Br_manager_timeout_hooks Struct Reference

Predefined server-loop hooks for a server with using the Br_manager and a timeout queue.

Inheritance diagram for L4Re::Util::Br_manager_timeout_hooks:

Collaboration diagram for L4Re::Util::Br_manager_timeout_hooks:

Additional Inherited Members

12.232.1 Detailed Description

Predefined server-loop hooks for a server with using the Br_manager and a timeout queue.

This class can be used for server loops that need the full package of buffer-register management and a timeout queue.

Definition at line 174 of file br_manager.

The documentation for this struct was generated from the following file:

- l4/re/util/br_manager

12.233 L4Re::Util::Cap_alloc_base Class Reference

Capability allocator.

Inherited by L4Re::Util::Cap_alloc< Size >.

Collaboration diagram for L4Re::Util::Cap_alloc_base:

Public Member Functions

- template<typename T >
  L4::Cap< T > alloc () throw ()

  Allocate a capability slot.

- template<typename T >
  void free (L4::Cap< T > const &cap, l4_cap_idx_t task=-1UL, l4_umword_t unmap_flags=L4_FP_ALL_SP↩ACES) throw ()

  Free a capability slot.

12.233.1 Detailed Description

Capability allocator.

Definition at line 38 of file bitmap_cap_alloc.

The documentation for this class was generated from the following file:

- l4/re/util/bitmap_cap_alloc

12.234 L4Re::Util::Counting_cap_alloc< COUNTERTYPE > Class Template Reference

Reference-counting cap allocator.

Collaboration diagram for L4Re::Util::Counting_cap_alloc< COUNTERTYPE >:

Public Member Functions

- L4::Cap< void > alloc () throw ()

  Allocated a new capability.
- template<typename T >
  L4::Cap< T > alloc () throw ()

  Allocated a new capability.
- void take (L4::Cap< void > cap) throw ()

  Increase the reference counter for the capability.
- bool free (L4::Cap< void > cap, l4_cap_idx_t task=L4_INVALID_CAP, unsigned unmap_flags=L4_FP_AL↩L_SPACES) throw ()

  Free the capability.
- bool release (L4::Cap< void > cap, l4_cap_idx_t task=L4_INVALID_CAP, unsigned unmap_flags=L4_FP↩_ALL_SPACES) throw ()

  Decrease the reference counter for a capability.
- long last () throw ()

  Return highest capability id managed by this allocator.

Protected Member Functions

- Counting_cap_alloc () throw ()

  Create a new, empty allocator.
- void setup (void *m, long capacity, long bias) throw ()

  Set up the backing memory for the allocator.

12.234.1 Detailed Description

template<typename COUNTERTYPE = L4Re::Util::Counter<unsigned char>>class L4Re::Util::Counting_cap_alloc< COUNTERTYPE >

Reference-counting cap allocator.

Note

The operations in this class are not thread-safe.

Definition at line 56 of file counting_cap_alloc.

12.234.2 Constructor & Destructor Documentation

12.234.2.1 template<typename COUNTERTYPE = L4Re::Util::Counter<unsigned char>> L4Re::Util::Counting_cap_alloc< COUNTERTYPE >::Counting_cap_alloc ( ) throw) [inline], [protected]

Create a new, empty allocator.

Needs to be initialized with setup() before it can be used.

Definition at line 85 of file counting_cap_alloc.

12.234.3 Member Function Documentation

12.234.3.1 template<typename COUNTERTYPE = L4Re::Util::Counter<unsigned char>> L4::Cap<void> L4Re::Util::Counting_cap_alloc< COUNTERTYPE >::alloc ( ) throw) [inline]

Allocated a new capability.

Returns

The newly allocated capability, invalid if the allocator was exhausted.

Examples:

examples/libs/l4re/c++/mem_alloc/ma+rm.cc, examples/libs/l4re/c++/shared_ds/ds_clnt.cc, and examples/libs/l4re/c++/shared_ds/ds_srv.cc.

Definition at line 110 of file counting_cap_alloc.

References L4_CAP_SHIFT.

Referenced by L4Re::Util::Counting_cap_alloc< COUNTERTYPE >::alloc(), L4Re::Util::Br_manager::alloc_buffer_demand(), and L4Re::Util::Br_manager::realloc_rcv_cap().

Here is the caller graph for this function:

12.234.3.2 template<typename COUNTERTYPE = L4Re::Util::Counter<unsigned char>> template<typename T > **L4::Cap<T> L4Re::Util::Counting_cap_alloc**< COUNTERTYPE >::alloc ( ) throw) [inline]

Allocated a new capability.

Returns

The newly allocated capability, invalid if the allocator was exhausted.

Definition at line 131 of file counting_cap_alloc.

References L4Re::Util::Counting_cap_alloc< COUNTERTYPE >::alloc(), and L4::cap_cast().

Here is the call graph for this function:

12.234.3.3 template<typename COUNTERTYPE = L4Re::Util::Counter<unsigned char>> bool **L4Re::Util::Counting_cap_alloc**< COUNTERTYPE >::free (**L4::Cap**< void > *cap*, **l4_cap_idx_t** *task* = **L4_INVALID_CAP**, unsigned *unmap_flags* = **L4_FP_ALL_SPACES**) throw) [inline]

Free the capability.

Parameters

cap	Capability to free.
task	If set, task to unmap the capability from.
unmap_flags	Flags for unmap, see l4_unmap_flags_t.

Precondition

The capability has been allocated. Calling free twice results in undefined behaviour.

Returns

True, if the capability was managed by this allocator.

Examples:

examples/libs/l4re/c++/mem_alloc/ma+rm.cc, examples/libs/l4re/c++/shared_ds/ds_clnt.cc, examples/libs/l4re/c++/shared_ds/ds_srv.cc, and examples/libs/l4re/streammap/client.cc.

Definition at line 172 of file counting_cap_alloc.

References L4_CAP_SHIFT, L4_INVALID_CAP, and l4_task_unmap().

Here is the call graph for this function:

12.234.3.4 template<typename COUNTERTYPE = L4Re::Util::Counter<unsigned char>> bool L4Re::Util::Counting_cap_alloc< COUNTERTYPE >::release (L4::Cap< void > *cap*, l4_cap_idx_t *task* = L4_INVALID_CAP, unsigned *unmap_flags* = L4_FP_ALL_SPACES) throw) [inline]

Decrease the reference counter for a capability.

Parameters

cap	Capability to release.
task	If set, task to unmap the capability from.
unmap_flags	Flags for unmap, see l4_unmap_flags_t.

Precondition

The capability has been allocated. Calling release on a free capability results in undefined behaviour.

Returns

True, if the capability was freed as a result of this operation. If false is returned the capability is either still in use or was already free or invalid.

Definition at line 211 of file counting_cap_alloc.

References L4_CAP_SHIFT, L4_INVALID_CAP, and l4_task_unmap().

Here is the call graph for this function:

12.234.3.5 template<typename COUNTERTYPE = L4Re::Util::Counter<unsigned char>> void L4Re::Util::Counting_cap_alloc< COUNTERTYPE >::setup (void * *m*, long *capacity*, long *bias*) throw) [inline], [protected]

Set up the backing memory for the allocator.

Parameters

m	Pointer to backing memory.
capacity	Number of capabilities that can be stored.
bias	First capability id to use by this allocator.

Definition at line 96 of file counting_cap_alloc.

12.234.3.6 template<typename COUNTERTYPE = L4Re::Util::Counter<unsigned char>> void L4Re::Util::Counting_cap_alloc< COUNTERTYPE >::take (L4::Cap< void > *cap*) throw) [inline]

Increase the reference counter for the capability.

Parameters

cap	Capability, whose reference counter should be increased.

If the capability was still free, it will be automatically allocated. Silently does nothing if the capability is not managed by this allocator.

Definition at line 146 of file counting_cap_alloc.

References L4_CAP_SHIFT.

The documentation for this class was generated from the following file:

- l4/re/util/counting_cap_alloc

12.235 L4Re::Util::Dataspace_svr Class Reference

Dataspace server class.

Collaboration diagram for L4Re::Util::Dataspace_svr:

Public Member Functions

- int map (l4_addr_t offset, l4_addr_t local_addr, unsigned long flags, l4_addr_t min_addr, l4_addr_t max_addr, L4::Ipc::Snd_fpage &memory)

  Map a region of the dataspace.
- virtual int map_hook (l4_addr_t offs, unsigned long flags, l4_addr_t min, l4_addr_t max)

  A hook that is called as the first operation in each map request.
- virtual int phys (l4_addr_t offset, l4_addr_t &phys_addr, l4_size_t &phys_size) throw ()

  Return physical address for a virtual address.
- virtual void take () throw ()

  Take a reference to this dataspace.
- virtual unsigned long release () throw ()

  Release a reference to this dataspace.
- virtual unsigned long copy (l4_addr_t dst_offs, l4_umword_t src_id, l4_addr_t src_offs, unsigned long size) throw ()

  Copy from src dataspace to this destination dataspace.
- virtual long clear (unsigned long offs, unsigned long size) const throw ()

  Clear a region in the dataspace.
- virtual long allocate (l4_addr_t offset, l4_size_t size, unsigned access) throw ()

  Allocate a region within a dataspace.

- virtual unsigned long page_shift () const throw ()

  Define the size of the flexpage to map.
- virtual bool is_static () const throw ()

  Return whether the dataspace is static.

12.235.1 Detailed Description

Dataspace server class.

The default implementation of the interface provides a continuously mapped dataspace.

Definition at line 40 of file dataspace_svr.

12.235.2 Member Function Documentation

12.235.2.1 virtual long L4Re::Util::Dataspace_svr::allocate (l4_addr_t *offset*, l4_size_t *size*, unsigned *access*) throw) [inline], [virtual]

Allocate a region within a dataspace.

Parameters

offset	Offset in the dataspace, in bytes.
size	Size of the range, in bytes.
access	Access mode with which the memory backing the dataspace region should be allocated.

Return values

0	Success
<0	Error

Definition at line 163 of file dataspace_svr.

References L4_ENODEV.

12.235.2.2 virtual long L4Re::Util::Dataspace_svr::clear (unsigned long *offs*, unsigned long *size*) const throw) [virtual]

Clear a region in the dataspace.

Parameters

offs	Start of the region
size	Size of the region

Return values

0	Success
<0	Error

12.235.2.3 virtual unsigned long L4Re::Util::Dataspace_svr::copy (l4_addr_t *dst_offs*, l4_umword_t *src_id*, l4_addr_t *src_offs*, unsigned long *size*) throw) [inline], [virtual]

Copy from src dataspace to this destination dataspace.

Parameters

dst_offs	Offset into the destination dataspace
src_id	Local id of the source dataspace
src_offs	Offset into the source dataspace
size	Number of bytes to copy

Returns

Number of bytes copied

Definition at line 137 of file dataspace_svr.

References L4_ENODEV.

12.235.2.4 virtual bool L4Re::Util::Dataspace_svr::is_static ( ) const throw) [inline],[virtual]

Return whether the dataspace is static.

Returns

True if dataspace is static

Definition at line 179 of file dataspace_svr.

12.235.2.5 int L4Re::Util::Dataspace_svr::map (l4_addr_t *offset*, l4_addr_t *local_addr*, unsigned long *flags*, l4_addr_t *min_addr*, l4_addr_t *max_addr*, L4::Ipc::Snd_fpage & *memory*)

Map a region of the dataspace.

Parameters

	offset	Offset to start within data space
	local_addr	Local address to map to.
	flags	Map flags, see L4Re::Dataspace::Map_flags.
	min_addr	Defines start of receive window.
	max_addr	Defines end of receive window.
out	*memory*	Send fpage to map

Return values

0	Success
<0	Error

12.235.2.6 virtual int L4Re::Util::Dataspace_svr::map_hook (l4_addr_t *offs*, unsigned long *flags*, l4_addr_t *min*, l4_addr_t *max*) [inline],[virtual]

A hook that is called as the first operation in each map request.

Parameters

offs	Offs param to map
flags	Flags param to map

min	Min param to map
max	Max param to map

Return values

<0	Error and the map request will be aborted with that error.
>=0	Success

See also

map

Definition at line 90 of file dataspace_svr.

12.235.2.7 virtual unsigned long L4Re::Util::Dataspace_svr::page_shift ( ) const throw) [inline], [virtual]

Define the size of the flexpage to map.

Returns

flexpage size

Definition at line 171 of file dataspace_svr.

References L4_LOG2_PAGESIZE.

12.235.2.8 virtual int L4Re::Util::Dataspace_svr::phys (l4_addr_t *offset*, l4_addr_t & *phys_addr*, l4_size_t & *phys_size*) throw) [virtual]

Return physical address for a virtual address.

Parameters

	offset	Offset into the dataspace
out	*phys_addr*	Physical address
out	*phys_size*	Size of continious physical region

Return values

0	Success
<0	Error

12.235.2.9 virtual unsigned long L4Re::Util::Dataspace_svr::release ( ) throw) [inline], [virtual]

Release a reference to this dataspace.

Returns

Number of references to the dataspace

Default does nothing and returns always zero.

Definition at line 124 of file dataspace_svr.

12.235.2.10 virtual void L4Re::Util::Dataspace_svr::take ( ) throw) [inline], [virtual]

Take a reference to this dataspace.

Default does nothing.

Definition at line 114 of file dataspace_svr.

The documentation for this class was generated from the following file:

- l4/re/util/dataspace_svr

12.236 L4Re::Util::Event_buffer_consumer_t< PAYLOAD > Class Template Reference

An event buffer consumer.

Inheritance diagram for L4Re::Util::Event_buffer_consumer_t< PAYLOAD >:

Collaboration diagram for L4Re::Util::Event_buffer_consumer_t< PAYLOAD >:

Public Member Functions

- template<typename CB , typename D >
  void foreach_available_event (CB const &cb, D data=D())

  Call function on every available event.
- template<typename CB , typename D >
  void process (L4::Cap< L4::Irq > irq, L4::Cap< L4::Thread > thread, CB const &cb, D data=D())

  Continuously wait for events and process them.

12.236.1 Detailed Description

template<typename PAYLOAD>class L4Re::Util::Event_buffer_consumer_t< PAYLOAD >

An event buffer consumer.

Definition at line 92 of file event_buffer.

12.236.2 Member Function Documentation

12.236.2.1 template<typename PAYLOAD > template<typename CB , typename D > void L4Re::Util::Event_buffer_consumer_t< PAYLOAD >::foreach_available_event (CB const & *cb*, D *data* = D()) [inline]

Call function on every available event.

Parameters

cb	Function callback.
data	Data to pass as an argument to the callback.

Definition at line 103 of file event_buffer.

References L4Re::Event_buffer_t< PAYLOAD >::Event::free().

Referenced by L4Re::Util::Event_buffer_consumer_t< PAYLOAD >::process().

Here is the call graph for this function:

Here is the caller graph for this function:

12.236.2.2 template<typename PAYLOAD > template<typename CB , typename D > void L4Re::Util::Event_buffer_consumer_t< PAYLOAD >::process (L4::Cap< L4::Irq > *irq*, L4::Cap< L4::Thread > *thread*, CB const & *cb*, D *data* = D()) [inline]

Continuously wait for events and process them.

Parameters

irq	Event signal to wait for.
thread	Thread capability of the thread calling this function.
cb	Callback function that is called for each received event.
data	Data to pass as an argument to the processing callback.

Note

This function never returns.

Definition at line 124 of file event_buffer.

References L4::Irq::attach(), L4Re::Util::Event_buffer_consumer_t< PAYLOAD >::foreach_available_event(), l4_error(), l4_ipc_error(), L4_IPC_NEVER, l4_irq_receive(), and l4_utcb().

Here is the call graph for this function:

The documentation for this class was generated from the following file:

- l4/re/util/event_buffer

12.237 L4Re::Util::Event_buffer_t< PAYLOAD > Class Template Reference

Event_buffer utility class.

Inheritance diagram for L4Re::Util::Event_buffer_t< PAYLOAD >:

Collaboration diagram for L4Re::Util::Event_buffer_t< PAYLOAD >:

Public Member Functions

- void * buf () const throw ()

  Return the buffer.
- long attach (L4::Cap< L4Re::Dataspace > ds, L4::Cap< L4Re::Rm > rm) throw ()

  Attach event buffer from address space.
- long detach (L4::Cap< L4Re::Rm > rm) throw ()

  Detach event buffer from address space.

12.237.1 Detailed Description

template<typename PAYLOAD>class L4Re::Util::Event_buffer_t< PAYLOAD >

Event_buffer utility class.

Definition at line 36 of file event_buffer.

12.237.2 Member Function Documentation

12.237.2.1 template<typename PAYLOAD > long L4Re::Util::Event_buffer_t< PAYLOAD >::attach (L4::Cap< L4Re::Dataspace > *ds*, L4::Cap< L4Re::Rm > *rm*) throw) [inline]

Attach event buffer from address space.

Parameters

ds	Dataspace of the event buffer.
rm	Region manager to attach buffer to.

Returns

0 on success, negative error code otherwise.

Definition at line 56 of file event_buffer.

References L4::Ipc::make_cap_rw(), and L4Re::Rm::Search_addr.

Here is the call graph for this function:

12.237.2.2 template<typename PAYLOAD > void* L4Re::Util::Event_buffer_t< PAYLOAD >::buf ( ) const throw) [inline]

Return the buffer.

Returns

Pointer to the event buffer.

Definition at line 46 of file event_buffer.

12.237.2.3 template<typename PAYLOAD > long L4Re::Util::Event_buffer_t< PAYLOAD >::detach (L4::Cap< L4Re::Rm > *rm*) throw) [inline]

Detach event buffer from address space.

Parameters

rm	Region manager to detach buffer from.

Returns

0 on success, negative error code otherwise.

Definition at line 77 of file event_buffer.

The documentation for this class was generated from the following file:

- l4/re/util/event_buffer

12.238 L4Re::Util::Event_t< PAYLOAD > Class Template Reference

Convenience wrapper for getting access to an event object.

Collaboration diagram for L4Re::Util::Event_t< PAYLOAD >:

Public Types

- enum Mode { Mode_irq, Mode_polling }

  Modes of operation.

Public Member Functions

- template<typename IRQ_TYPE >
  int init (L4::Cap< L4Re::Event > event, L4Re::Env const *env=L4Re::Env::env(), L4Re::Cap_alloc *ca=L4Re::Cap_alloc::get_cap_alloc(L4Re::Util::cap_alloc))

  Initialise an event object.
- int init_poll (L4::Cap< L4Re::Event > event, L4Re::Env const *env=L4Re::Env::env(), L4Re::Cap_alloc *ca=L4Re::Cap_alloc::get_cap_alloc(L4Re::Util::cap_alloc))

  Initialise an event object in polling mode.
- L4Re::Event_buffer_t< PAYLOAD > & buffer ()

  Get event buffer.
- L4::Cap< L4::Triggerable > irq () const

  Get event IRQ.

12.238.1 Detailed Description

template<typename PAYLOAD>class L4Re::Util::Event_t< PAYLOAD >

Convenience wrapper for getting access to an event object.

After calling init() the class supplies the event-buffer and the associated IRQ object.

Definition at line 41 of file event.

12.238.2 Member Enumeration Documentation

12.238.2.1 template<typename PAYLOAD > enum L4Re::Util::Event_t::Mode

Modes of operation.

Enumerator

Mode_irq Create an IRQ and attach, to get notifications.

Mode_polling Do not use an IRQ.

Definition at line 47 of file event.

12.238.3 Member Function Documentation

12.238.3.1 template<typename PAYLOAD > L4Re::Event_buffer_t<PAYLOAD>& L4Re::Util::Event_t< PAYLOAD >::buffer ( ) [inline]

Get event buffer.

Returns

Event buffer object.

Definition at line 156 of file event.

12.238.3.2 template<typename PAYLOAD > template<typename IRQ_TYPE > int L4Re::Util::Event_t< PAYLOAD >::init (L4::Cap< L4Re::Event > *event*, L4Re::Env const * *env* = L4Re::Env::env (), L4Re::Cap_alloc * *ca* = L4Re::Cap_alloc::get_cap_alloc (L4Re::Util::cap_alloc)) [inline]

Initialise an event object.

Template Parameters

IRQ_TYPE	Type used for handling notifications from the event provider. This must be derived from L4::Triggerable.

Parameters

event	Capability to event.
env	Pointer to L4Re-Environment
ca	Pointer to capability allocator.

Return values

0	Success
-L4_ENOMEM	No memory to allocate required capabilities.
<0	Other IPC errors.

Definition at line 68 of file event.

References L4::Icu::bind(), L4Re::Event::get_buffer(), L4_ENOMEM, l4_error(), L4::Ipc::make_cap_rw(), and L4Re::Rm::Search_addr.

Here is the call graph for this function:

12.238.3.3 template<typename PAYLOAD > int L4Re::Util::Event_t< PAYLOAD >::init_poll (L4::Cap< L4Re::Event > *event*, L4Re::Env const * *env* = L4Re::Env::env () , L4Re::Cap_alloc * *ca* = L4Re::Cap_alloc::get_cap_alloc (L4Re::Util::cap_alloc)) [inline]

Initialise an event object in polling mode.

Parameters

event	Capability to event.
env	Pointer to L4Re-Environment
ca	Pointer to capability allocator.

Return values

0	Success
-L4_ENOMEM	No memory to allocate required capabilities.
<0	Other IPC errors.

Definition at line 121 of file event.

References L4Re::Event::get_buffer(), L4_ENOMEM, L4::Ipc::make_cap_rw(), and L4Re::Rm::Search_addr.

Here is the call graph for this function:

12.238.3.4 template<typename PAYLOAD > L4::Cap<L4::Triggerable> L4Re::Util::Event_t< PAYLOAD >::irq ( ) const [inline]

Get event IRQ.

Returns

Event IRQ.

Definition at line 163 of file event.

The documentation for this class was generated from the following file:

- l4/re/util/event

12.239 L4Re::Util::Item_alloc_base Class Reference

Item allocator.

Inherited by L4Re::Util::Item_alloc< Bits >.

Collaboration diagram for L4Re::Util::Item_alloc_base:

12.239.1 Detailed Description

Item allocator.

Definition at line 38 of file item_alloc.

The documentation for this class was generated from the following file:

- l4/re/util/item_alloc

12.240 L4Re::Util::Names::Name Class Reference

Name class.

Inheritance diagram for L4Re::Util::Names::Name:

Collaboration diagram for L4Re::Util::Names::Name:

12.240.1 Detailed Description

Name class.

Definition at line 42 of file name_space_svr.

The documentation for this class was generated from the following file:

- l4/re/util/name_space_svr

12.241 L4Re::Util::Names::Name_space Class Reference

Abstract server-side implementation of the L4::Namespace interface.

Collaboration diagram for L4Re::Util::Names::Name_space:

Protected Member Functions

- virtual Entry ∗ alloc_dynamic_entry (Name const &n, unsigned flags)=0

  Allocate a new entry for the given name.
- virtual void free_dynamic_entry (Entry ∗e)=0

  Free an entry previously allocated with alloc_dynamic_entry().
- virtual int get_epiface (l4_umword_t data, bool is_local, L4::Epiface ∗∗lo)=0

  Return a pointer to the epiface assigned to a given label.
- virtual int copy_receive_cap (L4::Cap< void > ∗cap)=0

  Return the receive capability for permanent use.
- virtual void free_capability (L4::Cap< void > cap)=0

  Free a capability previously acquired with copy_receive_cap().
- virtual void free_epiface (L4::Epiface ∗epiface)=0

  Free epiface previously acquired with get_epiface().

12.241.1 Detailed Description

Abstract server-side implementation of the L4::Namespace interface.

Definition at line 184 of file name_space_svr.

12.241.2 Member Function Documentation

12.241.2.1 virtual Entry* L4Re::Util::Names::Name_space::alloc_dynamic_entry (Name const & *n*, unsigned *flags*) [protected], [pure virtual]

Allocate a new entry for the given name.

Parameters

n	Name of the entry, must be copied.
flags	Entry flags, see Obj::Flags.

Returns

A pointer to the newly allocated entry or NULL on error.

12.241.2.2 virtual int L4Re::Util::Names::Name_space::copy_receive_cap (L4::Cap< void > * *cap*) [protected], [pure virtual]

Return the receive capability for permanent use.

Parameters

out	*cap*	Capability slot with the received capability. Must be permanently available until free_capability() is called.

12.241.2.3 virtual void L4Re::Util::Names::Name_space::free_capability (L4::Cap< void > *cap*) [protected], [pure virtual]

Free a capability previously acquired with copy_receive_cap().

Parameters

in	*cap*	Capability to free.

12.241.2.4 virtual void L4Re::Util::Names::Name_space::free_dynamic_entry (Entry * *e*) [protected], [pure virtual]

Free an entry previously allocated with alloc_dynamic_entry().

Parameters

e	Entry to free.

12.241.2.5 virtual void L4Re::Util::Names::Name_space::free_epiface (L4::Epiface * *epiface*) [protected], [pure virtual]

Free epiface previously acquired with get_epiface().

Parameters

in	*epiface*	Epiface to free.

Called when an entry that points to an epiface is deleted allowing implementations that hold resources to free them.

12.241.2.6 virtual int L4Re::Util::Names::Name_space::get_epiface (l4_umword_t *data*, bool *is_local*, L4::Epiface ** *lo*) [protected],[pure virtual]

Return a pointer to the epiface assigned to a given label.

Parameters

in	*data*	Label or in the local case the capability slot of the receiving capability.
in	*is_local*	If true, a truly local capability was supplied.
out	*lo*	Pointer to epiface responsible for the capability.

Returns

L4_OK if a valid interface could be found or an error message otherwise.

The caller must make sure that the epiface remains valid until free_epiface is called. In particular, the capability slot must not be reallocated as long as the namespace server holds a reference to the epiface.

The documentation for this class was generated from the following file:

- l4/re/util/name_space_svr

12.242 L4Re::Util::Object_registry Class Reference

A registry that manages server objects and their attached IPC gates for a single server loop for a specific thread.

Inheritance diagram for L4Re::Util::Object_registry:

Collaboration diagram for L4Re::Util::Object_registry:

Public Member Functions

- Object_registry (L4::Ipc_svr::Server_iface ∗sif)

  Create a registry for the main thread of the task using the default factory.
- Object_registry (L4::Ipc_svr::Server_iface ∗sif, L4::Cap< L4::Thread > server, L4::Cap< L4::Factory > factory)

  Create a registry for arbitrary threads.
- L4::Cap< void > register_obj (L4::Epiface ∗o, char const ∗service)

  Register a new server object to a pre-allocated IPC gate.
- L4::Cap< void > register_obj (L4::Epiface ∗o)

  Register a new server object on a newly allocated capability.
- L4::Cap< L4::Irq > register_irq_obj (L4::Epiface ∗o)

  Register a handler for an interrupt.
- L4::Cap< L4::Irq > register_irq_obj (L4::Epiface ∗o, L4::Cap< L4::Irq > const &irq)

  Register a handler for an already existing interrupt.
- bool unregister_obj (L4::Epiface ∗o)

  Remove a server object from the handler list.

Additional Inherited Members

12.242.1 Detailed Description

A registry that manages server objects and their attached IPC gates for a single server loop for a specific thread.

This class manages most of the setup of a server object. If necessary, an IPC gate is created, the specified thread is bound to the IPC gate. Incoming IPC is dispatched to the server object based on the label of the IPC gate.

The object registry is also able to manage IRQ endpoints. They require a different method for the object creation. Otherwise they are handled in the same way as IPC gates: a server object is responsible to process the incoming interrupts.

Definition at line 51 of file object_registry.

12.242.2 Constructor & Destructor Documentation

12.242.2.1 L4Re::Util::Object_registry::Object_registry (L4::Ipc_svr::Server_iface * *sif*) [inline], [explicit]

Create a registry for the main thread of the task using the default factory.

Parameters

sif	Server loop interface.

Definition at line 75 of file object_registry.

12.242.2.2 L4Re::Util::Object_registry::Object_registry (L4::Ipc_svr::Server_iface * *sif*, L4::Cap< L4::Thread > *server*, L4::Cap< L4::Factory > *factory*) [inline]

Create a registry for arbitrary threads.

Parameters

sif	Server loop interface.
server	Capability to the thread that executes the server objects.
factory	Capability to a factory object capable of creating new IPC gates.

Definition at line 89 of file object_registry.

12.242.3 Member Function Documentation

12.242.3.1 L4::Cap<L4::Irq> L4Re::Util::Object_registry::register_irq_obj (L4::Epiface * *o*) [inline]

Register a handler for an interrupt.

Parameters

o	Server object that handles IRQs.

Returns

The capability the server object was registered with.

The IRQ will be newly allocated using the registry's factory object.

Definition at line 247 of file object_registry.

12.242.3.2 L4::Cap<L4::Irq> L4Re::Util::Object_registry::register_irq_obj (L4::Epiface * *o*, L4::Cap< L4::Irq > const & *irq*) [inline]

Register a handler for an already existing interrupt.

Parameters

o	Server object that handles the interrupts.
irq	Capability to an IRQ object, may be a hardware or software interrupt.

Returns

The capability the server object was registered with.

Definition at line 263 of file object_registry.

12.242.3.3 L4::Cap<void> L4Re::Util::Object_registry::register_obj (L4::Epiface * *o*, char const * *service*) [inline]

Register a new server object to a pre-allocated IPC gate.

Parameters

o	Server object that handles IPC requests.
service	Name of a pre-allocated IPC gate.

Returns

The capability the server object was registered with.

Definition at line 219 of file object_registry.

12.242.3.4 L4::Cap<void> L4Re::Util::Object_registry::register_obj (L4::Epiface * *o*) [inline]

Register a new server object on a newly allocated capability.

Parameters

o	Server object that handles IPC requests.

Returns

The capability the server object was registered with.

The IPC gate will be allocated using the registry's factory.

Definition at line 233 of file object_registry.

12.242.3.5 bool L4Re::Util::Object_registry::unregister_obj (L4::Epiface * *o*) [inline]

Remove a server object from the handler list.

Parameters

o	Server object to unbind.

Returns

True if the server object was bound to a valid capability.

The capability used by the server object will be unmapped.

Definition at line 279 of file object_registry.

References L4::Thread::Modify_senders::add(), L4_FP_ALL_SPACES, and L4::Thread::modify_senders().

Here is the call graph for this function:

The documentation for this class was generated from the following file:

- l4/re/util/object_registry

12.243 L4Re::Util::Ref_cap< T > Struct Template Reference

Automatic capability that implements automatic free and unmap of the capability selector.

Collaboration diagram for L4Re::Util::Ref_cap< T >:

12.243.1 Detailed Description

template<typename T>struct L4Re::Util::Ref_cap< T >

Automatic capability that implements automatic free and unmap of the capability selector.

Template Parameters

T	Type of the object that is referred by the capability.

This kind of automatic capability implements a counted reference to a capability selector. The capability shall be unmapped and freed when the reference count in the allocator goes to zero.

Usage:

```
L4Re::Util::Ref_cap<L4Re::Dataspace>::Cap global_ds_cap;

{
  L4Re::Util::Ref_cap<L4Re::Dataspace>::Cap
    ds_cap(L4Re::Util::cap_alloc.alloc<L4Re::Dataspace>());
  // reference count for the allocated cap selector is now 1
```

```
  // use the dataspace cap
  L4Re::chksys(mem_alloc->alloc(4096, ds_cap.get()));

  global_ds_cap = ds_cap;
  // reference count is now 2
  ...
}
// reference count dropped to 1 (ds_cap is no longer existing).
```

Definition at line 231 of file cap_alloc.

The documentation for this struct was generated from the following file:

- l4/re/util/cap_alloc

12.244 L4Re::Util::Ref_del_cap< T > Struct Template Reference

Automatic capability that implements automatic free and unmap+delete of the capability selector.

Collaboration diagram for L4Re::Util::Ref_del_cap< T >:

12.244.1 Detailed Description

template<typename T>struct L4Re::Util::Ref_del_cap< T >

Automatic capability that implements automatic free and unmap+delete of the capability selector.

Template Parameters

T	Type of the object that is referred by the capability.

This kind of automatic capability implements a counted reference to a capability selector. The capability shall be unmapped and freed when the reference count in the allocator goes to zero. The main difference to Ref_cap is that the unmap is done with the deletion flag enabled and this leads to the deletion of the object if the current task holds appropriate deletion rights.

Usage:

```
L4Re::Util::Ref_del_cap<L4Re::Dataspace>::Cap global_ds_cap;

{
  L4Re::Util::Ref_del_cap<L4Re::Dataspace>::Cap
    ds_cap(L4Re::Util::cap_alloc.alloc<L4Re::Dataspace>());
  // reference count for the allocated cap selector is now 1

  // use the dataspace cap
  L4Re::chksys(mem_alloc->alloc(4096, ds_cap.get()));

  global_ds_cap = ds_cap;
  // reference count is now 2
```

```
  ...
}
// reference count dropped to 1 (ds_cap is no longer existing).
...
global_ds_cap = L4_INVALID_CAP;
// reference count dropped to 0 (data space shall be deleted).
```

Definition at line 272 of file cap_alloc.

The documentation for this struct was generated from the following file:

- l4/re/util/cap_alloc

12.245 L4Re::Util::Registry_server< LOOP_HOOKS > Class Template Reference

A server loop object which has a Object_registry included.

Inheritance diagram for L4Re::Util::Registry_server< LOOP_HOOKS >:

Collaboration diagram for L4Re::Util::Registry_server< LOOP_HOOKS >:

Public Member Functions

- Registry_server ()

  Create a new server loop object for the main thread of the task.
- Registry_server (l4_utcb_t *utcb, L4::Cap< L4::Thread > server, L4::Cap< L4::Factory > factory)

  Create a new server loop object for an arbitrary thread and factory.
- Object_registry const * registry () const

  Return registry of this server loop.
- Object_registry * registry ()

  Return registry of this server loop.
- void L4_NORETURN loop ()

  Start the server loop.

Additional Inherited Members

12.245.1 Detailed Description

template<typename LOOP_HOOKS = L4::Ipc_svr::Default_loop_hooks>class L4Re::Util::Registry_server< LOOP_HOOKS >

A server loop object which has a Object_registry included.

Examples:

examples/clntsrv/server.cc, examples/libs/l4re/c++/shared_ds/ds_srv.cc, and examples/libs/l4re/streammap/server.cc.

Definition at line 301 of file object_registry.

12.245.2 Constructor & Destructor Documentation

12.245.2.1 template<typename LOOP_HOOKS = L4::Ipc_svr::Default_loop_hooks> L4Re::Util::Registry_server< LOOP_HOOKS >::Registry_server ( ) [inline]

Create a new server loop object for the main thread of the task.

Precondition

Must be called from the main thread or behaviour is undefined.

Definition at line 313 of file object_registry.

12.245.2.2 template<typename LOOP_HOOKS = L4::Ipc_svr::Default_loop_hooks> L4Re::Util::Registry_server< LOOP_HOOKS >::Registry_server (l4_utcb_t * *utcb*, L4::Cap< L4::Thread > *server*, L4::Cap< L4::Factory > *factory*) [inline]

Create a new server loop object for an arbitrary thread and factory.

Parameters

utcb	The UTCB of the thread running the server loop.
server	Capability to thread running the server loop.
factory	Capability to factory object used to create new IPC gates.

Definition at line 323 of file object_registry.

12.245.3 Member Function Documentation

12.245.3.1 template<typename LOOP_HOOKS = L4::Ipc_svr::Default_loop_hooks> Object_registry const* L4Re::Util::Registry_server< LOOP_HOOKS >::registry ( ) const [inline]

Return registry of this server loop.

Definition at line 329 of file object_registry.

12.245.3.2 template<typename LOOP_HOOKS = L4::Ipc_svr::Default_loop_hooks> Object_registry* L4Re::Util::Registry_server< LOOP_HOOKS >::registry ( ) [inline]

Return registry of this server loop.

Definition at line 331 of file object_registry.

The documentation for this class was generated from the following file:

- l4/re/util/object_registry

12.246 L4Re::Util::Smart_cap_auto< Unmap_flags > Class Template Reference

Helper for Auto_cap and Auto_del_cap.

Collaboration diagram for L4Re::Util::Smart_cap_auto< Unmap_flags >:

Static Public Member Functions

- static void free (L4::Cap_base &c)

  Free operation for L4::Smart_cap.

- static void invalidate (L4::Cap_base &c)

  Invalidate operation for L4::Smart_cap.

- static L4::Cap_base copy (L4::Cap_base const &src)

  Copy operation for L4::Smart_cap.

12.246.1 Detailed Description

template<unsigned long Unmap_flags = L4_FP_ALL_SPACES>class L4Re::Util::Smart_cap_auto< Unmap_flags >

Helper for Auto_cap and Auto_del_cap.

Definition at line 59 of file cap_alloc.

The documentation for this class was generated from the following file:

- l4/re/util/cap_alloc

12.247 L4Re::Util::Smart_count_cap< Unmap_flags > Class Template Reference

Helper for Ref_cap and Ref_del_cap.

Collaboration diagram for L4Re::Util::Smart_count_cap< Unmap_flags >:

Static Public Member Functions

- static void free (L4::Cap_base &c) throw ()

  Free operation for L4::Smart_cap (decrement ref count and delete if 0).

- static void invalidate (L4::Cap_base &c) throw ()

  Invalidate operation for L4::Smart_cap.

- static L4::Cap_base copy (L4::Cap_base const &src)

  Copy operation for L4::Smart_cap (increment ref count).

12.247.1 Detailed Description

template<unsigned long Unmap_flags = L4_FP_ALL_SPACES>class L4Re::Util::Smart_count_cap< Unmap_flags >

Helper for Ref_cap and Ref_del_cap.

Definition at line 99 of file cap_alloc.

The documentation for this class was generated from the following file:

- l4/re/util/cap_alloc

12.248 L4Re::Util::Vcon_svr< SVR > Class Template Reference

Console server template class.

Collaboration diagram for L4Re::Util::Vcon_svr< SVR >:

12.248.1 Detailed Description

template<typename SVR>class L4Re::Util::Vcon_svr< SVR >

Console server template class.

This template uses vcon_write() and vcon_read() to get and deliver data from the implementor.

vcon_read() needs to update the status argument with the L4_vcon_read_stat flags, especially the L4_VCON_R↩EAD_STAT_DONE flag to indicate that there's nothing more to read for the other end.

vcon_write() gets the live data from the UTCB. Make sure to copy out the data before using the UTCB again.

The size parameter of both functions is given in bytes.

Definition at line 46 of file vcon_svr.

The documentation for this class was generated from the following file:

- l4/re/util/vcon_svr

12.249 L4Re::Util::Video::Goos_svr Class Reference

Goos server class.

Collaboration diagram for L4Re::Util::Video::Goos_svr:

Public Member Functions

- L4::Cap< L4Re::Dataspace > get_fb () const

  Return framebuffer memory dataspace.
- L4Re::Video::Goos::Info const * screen_info () const

  Goos information structure.
- L4Re::Video::View::Info const * view_info () const

  View information structure.
- virtual int refresh (int x, int y, int w, int h)

  Refresh area of the framebuffer.
- void init_infos ()

  Initialize the view information structure of this object.
- virtual ~Goos_svr ()

  Destructor.

Protected Attributes

- L4::Cap< L4Re::Dataspace > _fb_ds

  Goos memory dataspace.
- L4Re::Video::Goos::Info _screen_info

  Goos information.
- L4Re::Video::View::Info _view_info

  View information.

12.249.1 Detailed Description

Goos server class.

Definition at line 36 of file goos_svr.

12.249.2 Member Function Documentation

12.249.2.1 L4::Cap<L4Re::Dataspace> L4Re::Util::Video::Goos_svr::get_fb ( ) const [inline]

Return framebuffer memory dataspace.

Returns

Goos memory dataspace

Definition at line 53 of file goos_svr.

References _fb_ds.

12.249.2.2 void L4Re::Util::Video::Goos_svr::init_infos ( ) [inline]

Initialize the view information structure of this object.

This function initializes the view info structure of this goos object based on the information in the goos information, i.e. the width, height and pixel_info of the goos information has to contain valid values before calling init_info().

Definition at line 89 of file goos_svr.

References L4Re::Video::View::Info::buffer_index, L4Re::Video::View::Info::flags, L4Re::Video::View::Info::height, L4Re::Video::Goos::Info::height, L4Re::Video::View::Info::pixel_info, L4Re::Video::Goos::Info::pixel_info, L4Re::Video::View::Info::view_index, L4Re::Video::View::Info::width, L4Re::Video::Goos::Info::width, L4Re::Video::View::Info::xpos, and L4Re::Video::View::Info::ypos.

12.249.2.3 virtual int L4Re::Util::Video::Goos_svr::refresh (int *x*, int *y*, int *w*, int *h*) [inline], [virtual]

Refresh area of the framebuffer.

Parameters

x	X coordinate (pixels)
y	Y coordinate (pixels)
w	Width of area in pixels
h	Height of area in pixels

Returns

0 on success, negative error code otherwise

Definition at line 77 of file goos_svr.

References L4_ENOSYS.

12.249.2.4 L4Re::Video::Goos::Info const* L4Re::Util::Video::Goos_svr::screen_info ( ) const [inline]

Goos information structure.

Returns

Return goos information structure.

Definition at line 59 of file goos_svr.

References _screen_info.

12.249.2.5 L4Re::Video::View::Info const* L4Re::Util::Video::Goos_svr::view_info ( ) const [inline]

View information structure.

Returns

Return view information structure.

Definition at line 65 of file goos_svr.

References _view_info.

The documentation for this class was generated from the following file:

- l4/re/util/video/goos_svr

12.250 L4Re::Vfs::Be_file Class Reference

Boiler plate class for implementing an open file for L4Re::Vfs.

Inheritance diagram for L4Re::Vfs::Be_file:

Collaboration diagram for L4Re::Vfs::Be_file:

Public Member Functions

- int unlock_all_locks () throw ()

  Unlock all locks on the file.
- L4::Cap< L4Re::Dataspace > data_space () const throw ()

  Get an L4Re::Dataspace object for the file.
- ssize_t readv (const struct iovec *, int) throw ()

  Default backend for POSIX read and readv functions.
- ssize_t writev (const struct iovec *, int) throw ()

  Default backend for POSIX write and writev functions.
- ssize_t pwritev (const struct iovec *, int, off64_t) throw ()

  Default backend for POSIX pwrite and pwritev functions.
- ssize_t preadv (const struct iovec *, int, off64_t) throw ()

  Default backend for POSIX pread and preadv functions.
- off64_t lseek64 (off64_t, int) throw ()

  Default backend for POSIX seek and lseek functions.
- int ftruncate64 (off64_t) throw ()

  Default backend for the POSIX truncate, ftruncate and similar functions.
- int fsync () const throw ()

  Default backend for POSIX fsync.

- int fdatasync () const throw ()

  Default backend for POSIX fdatasync.
- int ioctl (unsigned long, va_list) throw ()

  Default backend for POSIX ioctl.
- int fstat64 (struct stat64 ∗) const throw ()

  Get status information for the file.
- int fchmod (mode_t) throw ()

  Default backend for POSIX chmod and fchmod.
- int get_status_flags () const throw ()

  Default backend for POSIX fcntl subfunctions.
- int set_status_flags (long) throw ()

  Default backend for POSIX fcntl subfunctions.
- int get_lock (struct flock64 ∗) throw ()

  Default backend for POSIX fcntl subfunctions.
- int set_lock (struct flock64 ∗, bool) throw ()

  Default backend for POSIX fcntl subfunctions.
- int faccessat (const char ∗, int, int) throw ()

  Default backend for POSIX access and faccessat functions.
- int utime (const struct utimbuf ∗) throw ()

  Default backend for POSIX utime.
- int utimes (const struct timeval[2]) throw ()

  Default backend for POSIX utimes.
- int mkdir (const char ∗, mode_t) throw ()

  Default backend for POSIX mkdir and mkdirat.
- int unlink (const char ∗) throw ()

  Default backend for POSIX unlink, unlinkat.
- int rename (const char ∗, const char ∗) throw ()

  Default backend for POSIX rename, renameat.
- int link (const char ∗, const char ∗) throw ()

  Default backend for POSIX link, linkat.
- int symlink (const char ∗, const char ∗) throw ()

  Default backend for POSIX symlink, symlinkat.
- int rmdir (const char ∗) throw ()

  Default backend for POSIX rmdir, rmdirat.
- ssize_t readlink (char ∗, size_t)

  Default backend for POSIX readlink, readlinkat.

12.250.1 Detailed Description

Boiler plate class for implementing an open file for L4Re::Vfs.

This class may be used as a base class for everything that a POSIX file descriptor may point to. This are things such as regular files, directories, special device files, streams, pipes, and so on.

Examples:

tmpfs/lib/src/fs.cc.

Definition at line 39 of file backend.

12.250.2 Member Function Documentation

12.250.2.1 L4::Cap<L4Re::Dataspace> L4Re::Vfs::Be_file::data_space ( ) const throw) [inline], [virtual]

Get an L4Re::Dataspace object for the file.

This is used as a backend for POSIX mmap and mmap2 functions.

Note

mmap is not possible if the functions returns an invalid capability.

Returns

A capability to an L4Re::Dataspace, that represents the files contents in an L4Re way.

Implements L4Re::Vfs::Regular_file.

Definition at line 47 of file backend.

12.250.2.2 int L4Re::Vfs::Be_file::fstat64 (struct stat64 * *buf*) const throw) [inline], [virtual]

Get status information for the file.

This is the backend for POSIX fstat, stat, fstat64 and friends.

Parameters

out	*buf*	This buffer is filled with the status information.

Returns

0 on success, or <0 on error.

Implements L4Re::Vfs::Generic_file.

Definition at line 86 of file backend.

12.250.2.3 int L4Re::Vfs::Be_file::unlock_all_locks ( ) throw) [inline], [virtual]

Unlock all locks on the file.

Note

All locks means all locks independent by which file the locks were taken.

This method is called by the POSIX close implementation to get the POSIX semantics of releasing all locks taken by this application on a close for any fd referencing the real file.

Returns

0 on success, or <0 on error.

Implements L4Re::Vfs::Generic_file.

Definition at line 43 of file backend.

The documentation for this class was generated from the following file:

- l4/l4re_vfs/backend

12.251 L4Re::Vfs::Be_file_system Class Reference

Boilerplate class for implementing a L4Re::Vfs::File_system.

Inheritance diagram for L4Re::Vfs::Be_file_system:

Collaboration diagram for L4Re::Vfs::Be_file_system:

Public Member Functions

- Be_file_system (char const ∗fstype) throw ()

  Create a file-system object for the given fstype.
- ∼Be_file_system () throw ()

  Destroy a file-system object.
- char const ∗ type () const throw ()

  Return the file-system type.

12.251.1 Detailed Description

Boilerplate class for implementing a L4Re::Vfs::File_system.

This class already takes care of registering and unregistering the file system in the global registry and implements the type() method.

Examples:

tmpfs/lib/src/fs.cc.

Definition at line 290 of file backend.

12.251.2 Constructor & Destructor Documentation

12.251.2.1 L4Re::Vfs::Be_file_system::Be_file_system (char const * *fstype*) throw) [inline], [explicit]

Create a file-system object for the given *fstype*.

Parameters

fstype	The type that type() shall return.

This constructor takes care of registering the file system in the registry of L4Re::Vfs::vfs_ops.

Definition at line 304 of file backend.

12.251.2.2 L4Re::Vfs::Be_file_system::~Be_file_system ( ) throw) [inline]

Destroy a file-system object.

This destructor takes care of removing this file system from the registry of L4Re::Vfs::vfs_ops.

Definition at line 316 of file backend.

12.251.3 Member Function Documentation

12.251.3.1 char const* L4Re::Vfs::Be_file_system::type ( ) const throw) [inline], [virtual]

Return the file-system type.

Returns the file-system type given as *fstype* in the constructor.

Implements L4Re::Vfs::File_system.

Definition at line 326 of file backend.

The documentation for this class was generated from the following file:

- l4/l4re_vfs/backend

12.252 L4Re::Vfs::Directory Class Reference

Interface for a POSIX file that is a directory.

```
#include <vfs.h>
```

Inheritance diagram for L4Re::Vfs::Directory:

Collaboration diagram for L4Re::Vfs::Directory:

Public Member Functions

- virtual int faccessat (const char *path, int mode, int flags)=0 throw ()

  Check access permissions on the given file.
- virtual int mkdir (const char *path, mode_t mode)=0 throw ()

  Create a new subdirectory.
- virtual int unlink (const char *path)=0 throw ()

  Unlink the given file from that directory.
- virtual int rename (const char *src_path, const char *dst_path)=0 throw ()

  Rename the given file.
- virtual int link (const char *src_path, const char *dst_path)=0 throw ()

  Create a hard link (second name) for the given file.
- virtual int symlink (const char *src_path, const char *dst_path)=0 throw ()

  Create a symbolic link for the given file.
- virtual int rmdir (const char *path)=0 throw ()

  Delete an empty directory.

12.252.1 Detailed Description

Interface for a POSIX file that is a directory.

This interface provides functionality for directory files in the L4Re::Vfs. However, real objects use always the combined L4Re::Vfs::File interface.

Definition at line 141 of file vfs.h.

12.252.2 Member Function Documentation

12.252.2.1 virtual int L4Re::Vfs::Directory::faccessat (const char * *path*, int *mode*, int *flags*) throw) [pure virtual]

Check access permissions on the given file.

Backend function for POSIX access and faccessat functions.

Parameters

path	The path relative to this directory. Note: *path* is relative to this directory and may contain subdirectories.
mode	The access mode to check.
flags	The flags as in POSIX faccessat (AT_EACCESS, AT_SYMLINK_NOFOLLOW).

Returns

0 on success, or <0 on error.

Implemented in L4Re::Vfs::Be_file.

12.252.2.2 virtual int L4Re::Vfs::Directory::link (const char * *src_path*, const char * *dst_path*) throw) [pure virtual]

Create a hard link (second name) for the given file.

Backend for the POSIX link and linkat functions.

Parameters

src_path	The old name to the file. Note: *src_path* is relative to this directory and may contain subdirectories.
dst_path	The new (second) name for the file. Note: *dst_path* is relative to this directory and may contain subdirectories.

Returns

0 on success, or <0 on error.

Implemented in L4Re::Vfs::Be_file.

12.252.2.3 virtual int L4Re::Vfs::Directory::mkdir (const char * *path*, mode_t *mode*) throw) [pure virtual]

Create a new subdirectory.

Backend for POSIX mkdir and mkdirat function calls.

Parameters

path	The name of the subdirectory to create. Note: *path* is relative to this directory and may contain subdirectories.
mode	The file mode to use for the new directory.

Returns

0 on success, or <0 on error. -ENOTDIR if this or some component in path is is not a directory.

Implemented in L4Re::Vfs::Be_file.

12.252.2.4 virtual int L4Re::Vfs::Directory::rename (const char * *src_path,* const char * *dst_path*) throw) [pure virtual]

Rename the given file.

Backend for the POSIX rename, renameat functions.

Parameters

src_path	The old name to the file to rename. Note: *src_path* is relative to this directory and may contain subdirectories.
dst_path	The new name for the file. Note: *dst_path* is relative to this directory and may contain subdirectories.

Returns

0 on success, or <0 on error.

Implemented in L4Re::Vfs::Be_file.

12.252.2.5 virtual int L4Re::Vfs::Directory::rmdir (const char * *path*) throw) [pure virtual]

Delete an empty directory.

Backend for POSIX rmdir, rmdirat functions.

Parameters

path	The name of the directory to remove. Note: *path* is relative to this directory and may contain subdirectories.

Returns

0 on success, or <0 on error.

Implemented in L4Re::Vfs::Be_file.

12.252.2.6 virtual int L4Re::Vfs::Directory::symlink (const char * *src_path*, const char * *dst_path*) throw) [pure virtual]

Create a symbolic link for the given file.

Backend for the POSIX symlink and symlinkat functions.

Parameters

src_path	The old name to the file. Note: *src_path* shall be an absolute path.
dst_path	The name for symlink. Note: *dst_path* is relative to this directory and may contain subdirectories.

Returns

0 on success, or <0 on error.

Implemented in L4Re::Vfs::Be_file.

12.252.2.7 virtual int L4Re::Vfs::Directory::unlink (const char * *path*) throw) [pure virtual]

Unlink the given file from that directory.

Backend for the POSIX unlink and unlinkat functions.

Parameters

path	The name to the file to unlink. Note: *path* is relative to this directory and may contain subdirectories.

Returns

0 on success, or <0 on error.

Implemented in L4Re::Vfs::Be_file.

The documentation for this class was generated from the following file:

- l4/l4re_vfs/vfs.h

12.253 L4Re::Vfs::File Class Reference

The basic interface for an open POSIX file.

```
#include <vfs.h>
```

Inheritance diagram for L4Re::Vfs::File:

Collaboration diagram for L4Re::Vfs::File:

Additional Inherited Members

12.253.1 Detailed Description

The basic interface for an open POSIX file.

An open POSIX file can be anything that hides behind a POSIX file descriptor. This means that even a directories are files. An open file can be anything from a directory to a special device file so see Generic_file, Regular_file, Directory, and Special_file for more information.

Note

For implementing a backend for the L4Re::Vfs you may use L4Re::Vfs::Be_file as a base class.

Definition at line 430 of file vfs.h.

The documentation for this class was generated from the following file:

- l4/l4re_vfs/vfs.h

12.254 L4Re::Vfs::File_system Class Reference

Basic interface for an L4Re::Vfs file system.

```
#include <vfs.h>
```

Inheritance diagram for L4Re::Vfs::File_system:

Collaboration diagram for L4Re::Vfs::File_system:

Public Member Functions

- virtual char const ∗ type () const =0 throw ()

  Returns the type of the file system, used in mount as fstype argument.

- virtual int mount (char const ∗source, unsigned long mountflags, void const ∗data, cxx::Ref_ptr< File > ∗dir)=0 throw ()

  Create a directory object dir representing source mounted with this file system.

12.254.1 Detailed Description

Basic interface for an L4Re::Vfs file system.

Note

For implementing a special file system you may use L4Re::Vfs::Be_file_system as a base class.

The may purpose of this interface is that there is a single object for each supported file-system type (e.g., ext2, vfat) exists in your application and is registered at the L4Re::Vfs::Fs singleton available in via L4Re::Vfs::vfs_ops. At the end the POSIX mount function call the File_system::mount method for the given file-system type given in mount.

Definition at line 831 of file vfs.h.

12.254.2 Member Function Documentation

12.254.2.1 virtual int L4Re::Vfs::File_system::mount (char const ∗ *source*, unsigned long *mountflags*, void const ∗ *data*, cxx::Ref_ptr< File > ∗ *dir*) throw) [pure virtual]

Create a directory object *dir* representing *source* mounted with this file system.

Parameters

source	The path to the source device to mount. This may also be some URL or anything file-system specific.
mountflags	The mount flags as specified in the POSIX mount call.
data	The data as specified in the POSIX mount call. The contents are file-system specific.

Return values

dir	A new directory object representing the file-system root directory.

Returns

0 on success, and <0 on error (e.g. -EINVAL).

Referenced by L4Re::Vfs::Fs::mount().

Here is the caller graph for this function:

12.254.2.2 virtual char const* L4Re::Vfs::File_system::type ( ) const throw) [pure virtual]

Returns the type of the file system, used in mount as fstype argument.

Note

This method is already provided by Be_file_system.

Implemented in L4Re::Vfs::Be_file_system.

The documentation for this class was generated from the following file:

- l4/l4re_vfs/vfs.h

12.255 L4Re::Vfs::Fs Class Reference

POSIX File-system related functionality.

`#include <vfs.h>`

Inheritance diagram for L4Re::Vfs::Fs:

Collaboration diagram for L4Re::Vfs::Fs:

Public Member Functions

- virtual cxx::Ref_ptr< File > get_file (int fd)=0 throw ()

  Get the L4Re::Vfs::File for the file descriptor fd.
- virtual cxx::Ref_ptr< File > get_root ()=0 throw ()

  Get the directory object for the applications root directory.
- virtual cxx::Ref_ptr< File > get_cwd () throw ()

  Get the directory object for the applications current working directory.
- virtual void set_cwd (cxx::Ref_ptr< File > const &) throw ()

  Set the current working directory for the application.
- virtual int alloc_fd (cxx::Ref_ptr< File > const &f=cxx::Ref_ptr<>::Nil)=0 throw ()

  Allocate the next free file descriptor.
- virtual cxx::Ref_ptr< File > set_fd (int fd, cxx::Ref_ptr< File > const &f=cxx::Ref_ptr<>::Nil)=0 throw ()

  Set the file object referenced by the file descriptor fd.
- virtual cxx::Ref_ptr< File > free_fd (int fd)=0 throw ()

  Free the file descriptor fd.
- int mount (char const *path, cxx::Ref_ptr< File > const &dir) throw ()

  Mount a given file object at the given global path in the VFS.
- int mount (char const *source, char const *target, char const *fstype, unsigned long mountflags, void const *data) throw ()

  Backend for the POSIX mount call.

12.255.1 Detailed Description

POSIX File-system related functionality.

Note

This class usually exists as a singleton as a superclass of L4Re::Vfs::Ops (

See also

L4Re::Vfs::vfs_ops).

Definition at line 883 of file vfs.h.

12.255.2 Member Function Documentation

12.255.2.1 virtual int L4Re::Vfs::Fs::alloc_fd (cxx::Ref_ptr< File > const & *f* = cxx::Ref_ptr<>::Nil) throw) [pure virtual]

Allocate the next free file descriptor.

Parameters

f	The file to assign to that file descriptor.

Returns

the allocated file descriptor, or -EMFILE on error.

12.255.2.2 virtual cxx::Ref_ptr<File> L4Re::Vfs::Fs::free_fd (int *fd*) throw) [pure virtual]

Free the file descriptor *fd*.

Parameters

fd	The file descriptor to free.

Returns

A pointer to the file object that was assigned to the fd.

12.255.2.3 virtual cxx::Ref_ptr<File> L4Re::Vfs::Fs::get_file (int *fd*) throw) [pure virtual]

Get the L4Re::Vfs::File for the file descriptor *fd*.

Parameters

fd	The POSIX file descriptor number.

Returns

A pointer to the File object, or 0 if *fd* is not open.

12.255.2.4 int L4Re::Vfs::Fs::mount (char const * *path*, cxx::Ref_ptr< File > const & *dir*) throw) [inline]

Mount a given file object at the given global path in the VFS.

Parameters

path	The global path to mount *dir* at.
dir	A pointer to the file/directory object that shall be mounted at *path*.

Returns

0 on success, or <0 on error.

Definition at line 972 of file vfs.h.

12.255.2.5 virtual cxx::Ref_ptr<File> L4Re::Vfs::Fs::set_fd (int *fd*, cxx::Ref_ptr< File > const & *f* = cxx::Ref_ptr<>::Nil) throw) [pure virtual]

Set the file object referenced by the file descriptor *fd*.

Parameters

fd	The file descriptor to set to *f*;
f	The file object to assign.

Returns

A pointer to the file object that was previously assigned to fd.

The documentation for this class was generated from the following file:

- l4/l4re_vfs/vfs.h

12.256 L4Re::Vfs::Generic_file Class Reference

The common interface for an open POSIX file.

```
#include <vfs.h>
```

Inheritance diagram for L4Re::Vfs::Generic_file:

Collaboration diagram for L4Re::Vfs::Generic_file:

Public Member Functions

- virtual int unlock_all_locks ()=0 throw ()

  Unlock all locks on the file.
- virtual int fstat64 (struct stat64 ∗buf) const =0 throw ()

  Get status information for the file.
- virtual int fchmod (mode_t)=0 throw ()

  Change POSIX access rights on that file.
- virtual int get_status_flags () const =0 throw ()

  Get file status flags (fcntl F_GETFL).
- virtual int set_status_flags (long flags)=0 throw ()

  Set file status flags (fcntl F_SETFL).

12.256.1 Detailed Description

The common interface for an open POSIX file.

This interface is common to all kinds of open files, independent of the file type (e.g., directory, regular file etc.). However, in the L4Re::Vfs the interface File is used for every real object.

See also

L4Re::Vfs::File for mor information.

Definition at line 63 of file vfs.h.

12.256.2 Member Function Documentation

12.256.2.1 virtual int L4Re::Vfs::Generic_file::fchmod (mode_t) throw) [pure virtual]

Change POSIX access rights on that file.

Backend for POSIX chmod and fchmod.

Implemented in L4Re::Vfs::Be_file.

12.256.2.2 virtual int L4Re::Vfs::Generic_file::fstat64 (struct stat64 * *buf*) const throw) [pure virtual]

Get status information for the file.

This is the backend for POSIX fstat, stat, fstat64 and friends.

Parameters

out	*buf*	This buffer is filled with the status information.

Returns

0 on success, or <0 on error.

Implemented in L4Re::Vfs::Be_file.

12.256.2.3 virtual int L4Re::Vfs::Generic_file::get_status_flags ( ) const throw) [pure virtual]

Get file status flags (fcntl F_GETFL).

This function is used by the fcntl implementation for the F_GETFL command.

Returns

flags such as O_RDONLY, O_WRONLY, O_RDWR, O_DIRECT, O_ASYNC, O_NOATIME, O_NONBLOCK, or <0 on error.

Implemented in L4Re::Vfs::Be_file.

12.256.2.4 virtual int L4Re::Vfs::Generic_file::set_status_flags (long *flags*) throw) [pure virtual]

Set file status flags (fcntl F_SETFL).

This function is used by the fcntl implementation for the F_SETFL command.

Parameters

flags	The file status flags to set. This must be a combination of O_RDONLY, O_WRONLY, O_RD↩WR, O_APPEND, O_ASYNC, O_DIRECT, O_NOATIME, O_NONBLOCK.

Note

Creation flags such as O_CREAT, O_EXCL, O_NOCTTY, O_TRUNC are ignored.

Returns

0 on success, or <0 on error.

Implemented in L4Re::Vfs::Be_file.

12.256.2.5 virtual int L4Re::Vfs::Generic_file::unlock_all_locks ( ) throw) [pure virtual]

Unlock all locks on the file.

Note

All locks means all locks independent by which file the locks were taken.

This method is called by the POSIX close implementation to get the POSIX semantics of releasing all locks taken by this application on a close for any fd referencing the real file.

Returns

0 on success, or <0 on error.

Implemented in L4Re::Vfs::Be_file.

The documentation for this class was generated from the following file:

- l4/l4re_vfs/vfs.h

12.257 L4Re::Vfs::Mman Class Reference

Interface for the POSIX memory management.

`#include <vfs.h>`

Inheritance diagram for L4Re::Vfs::Mman:

Collaboration diagram for L4Re::Vfs::Mman:

Public Member Functions

- virtual int mmap2 (void ∗start, size_t len, int prot, int flags, int fd, off_t offset, void ∗∗ptr)=0 throw ()

  Backend for the mmap2 system call.
- virtual int munmap (void ∗start, size_t len)=0 throw ()

  Backend for the munmap system call.
- virtual int mremap (void ∗old, size_t old_sz, size_t new_sz, int flags, void ∗∗new_addr)=0 throw ()

  Backend for the mremap system call.
- virtual int mprotect (const void ∗a, size_t sz, int prot)=0 throw ()

  Backend for the mprotect system call.
- virtual int msync (void ∗addr, size_t len, int flags)=0 throw ()

  Backend for the msync system call.
- virtual int madvise (void ∗addr, size_t len, int advice)=0 throw ()

  Backend for the madvice system call.

12.257.1 Detailed Description

Interface for the POSIX memory management.

Note

This interface exists usually as a singleton as superclass of L4Re::Vfs::Ops.

An implementation for this interface is in l4/l4re_vfs/impl/vfs_impl.h and used by the l4re_vfs library or by the VFS implementation in ldso.

Definition at line 789 of file vfs.h.

The documentation for this class was generated from the following file:

- l4/l4re_vfs/vfs.h

12.258 L4Re::Vfs::Ops Class Reference

Interface for the POSIX backends for an application.

```
#include <vfs.h>
```

Inheritance diagram for L4Re::Vfs::Ops:

Collaboration diagram for L4Re::Vfs::Ops:

Additional Inherited Members

12.258.1 Detailed Description

Interface for the POSIX backends for an application.

Note

There usually exists a singe instance of this interface available via L4Re::Vfs::vfs_ops that is used for all kinds of C-Library functions.

Definition at line 1020 of file vfs.h.

The documentation for this class was generated from the following file:

- l4/l4re_vfs/vfs.h

12.259 L4Re::Vfs::Regular_file Class Reference

Interface for a POSIX file that provides regular file semantics.

```
#include <vfs.h>
```

Inheritance diagram for L4Re::Vfs::Regular_file:

Collaboration diagram for L4Re::Vfs::Regular_file:

Public Member Functions

- virtual L4::Cap< L4Re::Dataspace > data_space () const =0 throw ()

  Get an L4Re::Dataspace object for the file.
- virtual ssize_t readv (const struct iovec *, int iovcnt)=0 throw ()

  Read one or more blocks of data from the file.
- virtual ssize_t writev (const struct iovec *, int iovcnt)=0 throw ()

  Write one or more blocks of data to the file.
- virtual off64_t lseek64 (off64_t, int)=0 throw ()

  Change the file pointer.
- virtual int ftruncate64 (off64_t pos)=0 throw ()

  Truncate the file at the given position.
- virtual int fsync () const =0 throw ()

  Sync the data and meta data to persistent storage.
- virtual int fdatasync () const =0 throw ()

  Sync the data to persistent storage.
- virtual int get_lock (struct flock64 *lock)=0 throw ()

  Test if the given lock can be placed in the file.
- virtual int set_lock (struct flock64 *lock, bool wait)=0 throw ()

  Acquire or release the given lock on the file.

12.259.1 Detailed Description

Interface for a POSIX file that provides regular file semantics.

Real objects use always the combined L4Re::Vfs::File interface.

Definition at line 262 of file vfs.h.

12.259.2 Member Function Documentation

12.259.2.1 virtual L4::Cap<L4Re::Dataspace> L4Re::Vfs::Regular_file::data_space ( ) const throw) [pure virtual]

Get an L4Re::Dataspace object for the file.

This is used as a backend for POSIX mmap and mmap2 functions.

Note

mmap is not possible if the functions returns an invalid capability.

Returns

A capability to an L4Re::Dataspace, that represents the files contents in an L4Re way.

Implemented in L4Re::Vfs::Be_file.

12.259.2.2 virtual int L4Re::Vfs::Regular_file::fdatasync ( ) const throw) [pure virtual]

Sync the data to persistent storage.

This is the backend for POSIX fdatasync.

Implemented in L4Re::Vfs::Be_file.

12.259.2.3 virtual int L4Re::Vfs::Regular_file::fsync ( ) const throw) [pure virtual]

Sync the data and meta data to persistent storage.

This is the backend for POSIX fsync.

Implemented in L4Re::Vfs::Be_file.

12.259.2.4 virtual int L4Re::Vfs::Regular_file::ftruncate64 (off64_t *pos*) throw) [pure virtual]

Truncate the file at the given position.

This function is the backend for truncate and friends.

Parameters

pos	The offset at which the file shall be truncated.

Returns

0 on success, or <0 on error.

Implemented in L4Re::Vfs::Be_file.

12.259.2.5 virtual int L4Re::Vfs::Regular_file::get_lock (struct flock64 * *lock*) throw) [pure virtual]

Test if the given lock can be placed in the file.

This function is used as backend for fcntl F_GETLK commands.

Parameters

lock	The lock that shall be placed on the file. The *l_type* member will contain `F_UNLCK` if the lock could be placed.

Returns

0 on success, <0 on error.

Implemented in L4Re::Vfs::Be_file.

12.259.2.6 virtual off64_t L4Re::Vfs::Regular_file::lseek64 (off64_t , int) throw) [pure virtual]

Change the file pointer.

This is the backend for POSIX seek, lseek and friends.

Returns

The new file position, or <0 on error.

Implemented in L4Re::Vfs::Be_file.

12.259.2.7 virtual ssize_t L4Re::Vfs::Regular_file::readv (const struct iovec * , int *iovcnt*) throw) [pure virtual]

Read one or more blocks of data from the file.

This function acts as backend for POSIX read and readv calls and reads data starting for the f_pos pointer of that open file. The file pointer is advanced according to the number of red bytes.

Returns

The number of bytes red from the file. or <0 on error-

Implemented in L4Re::Vfs::Be_file.

12.259.2.8 virtual int L4Re::Vfs::Regular_file::set_lock (struct flock64 * *lock*, bool *wait*) throw) [pure virtual]

Acquire or release the given lock on the file.

This function is used as backend for fcntl F_SETLK and F_SETLKW commands.

Parameters

lock	The lock that shall be placed on the file.
wait	If true, then block if there is a conflicting lock on the file.

Returns

0 on success, <0 on error.

Implemented in L4Re::Vfs::Be_file.

12.259.2.9 virtual ssize_t L4Re::Vfs::Regular_file::writev (const struct iovec * , int *iovcnt*) throw) [pure virtual]

Write one or more blocks of data to the file.

This function acts as backend for POSIX write and writev calls. The data is written starting at the current file pointer and the file pointer must be advanced according to the number of written bytes.

Returns

The number of bytes written to the file, or <0 on error.

Implemented in L4Re::Vfs::Be_file.

The documentation for this class was generated from the following file:

- l4/l4re_vfs/vfs.h

12.260 L4Re::Vfs::Special_file Class Reference

Interface for a POSIX file that provides special file semantics.

```
#include <vfs.h>
```

Inheritance diagram for L4Re::Vfs::Special_file:

Collaboration diagram for L4Re::Vfs::Special_file:

Public Member Functions

- virtual int ioctl (unsigned long cmd, va_list args)=0 throw ()

  The famous IO control.

12.260.1 Detailed Description

Interface for a POSIX file that provides special file semantics.

Real objects use always the combined L4Re::Vfs::File interface.

Definition at line 395 of file vfs.h.

12.260.2 Member Function Documentation

12.260.2.1 virtual int L4Re::Vfs::Special_file::ioctl (unsigned long *cmd*, va_list *args*) throw) [pure virtual]

The famous IO control.

Backend for POSIX generic object invocation ioctl.

Parameters

cmd	The ioctl command.
args	The arguments for the ioctl, usually some kind of pointer.

Returns

>=0 on success, or <0 on error.

Implemented in L4Re::Vfs::Be_file.

The documentation for this class was generated from the following file:

- l4/l4re_vfs/vfs.h

12.261 L4Re::Video::Color_component Class Reference

A color component.

Collaboration diagram for L4Re::Video::Color_component:

Public Member Functions

- Color_component ()

  Constructor.
- Color_component (unsigned char bits, unsigned char shift)

  Constructor.
- unsigned char size () const

  Return the number of bits used by the component.
- unsigned char shift () const

  Return the position of the component in the pixel.
- bool operator== (Color_component const &o) const

  Compare for equality.
- int get (unsigned long v) const

  Get component from value (normalized to 16bits).
- long unsigned set (int v) const

  Transform 16bit normalized value to the component in the color space.
- template<typename OUT >
  void dump (OUT &s) const

  Dump information on the view information to a stream.

12.261.1 Detailed Description

A color component.

Definition at line 31 of file colors.

12.261.2 Constructor & Destructor Documentation

12.261.2.1 L4Re::Video::Color_component::Color_component (unsigned char *bits*, unsigned char *shift*) [inline]

Constructor.

Parameters

bits	Number of bits used by the component
shift	Position in bits of the component in the pixel

Definition at line 46 of file colors.

12.261.3 Member Function Documentation

12.261.3.1 template<typename OUT > void L4Re::Video::Color_component::dump (OUT & *s*) const [inline]

Dump information on the view information to a stream.

Parameters

s	Stream

Returns

The stream

Definition at line 93 of file colors.

12.261.3.2 int L4Re::Video::Color_component::get (unsigned long *v*) const [inline]

Get component from value (normalized to 16bits).

Parameters

v	Value

Returns

Converted value

Definition at line 73 of file colors.

12.261.3.3 bool L4Re::Video::Color_component::operator== (Color_component const & *o*) const [inline]

Compare for equality.

Returns

True if the same components are described, false if not.

Definition at line 65 of file colors.

12.261.3.4 long unsigned L4Re::Video::Color_component::set (int *v*) const [inline]

Transform 16bit normalized value to the component in the color space.

Parameters

v	Value return Converted value.

Definition at line 84 of file colors.

12.261.3.5 unsigned char L4Re::Video::Color_component::shift ( ) const [inline]

Return the position of the component in the pixel.

Returns

Position in bits of the component in the pixel

Definition at line 59 of file colors.

12.261.3.6 unsigned char L4Re::Video::Color_component::size ( ) const [inline]

Return the number of bits used by the component.

Returns

Number of bits used by the component

Definition at line 53 of file colors.

Referenced by L4Re::Video::Pixel_info::bits_per_pixel(), and L4Re::Video::Pixel_info::has_alpha().

Here is the caller graph for this function:

The documentation for this class was generated from the following file:

- l4/re/video/colors

12.262 L4Re::Video::Goos Class Reference

A goos.

Inheritance diagram for L4Re::Video::Goos:

Collaboration diagram for L4Re::Video::Goos:

Data Structures

- struct Info

  Information structure of a goos.

Public Types

- enum Flags { F_auto_refresh = 0x01, F_pointer = 0x02, F_dynamic_views = 0x04, F_dynamic_buffers = 0x08 }

Flags for a goos.

Public Member Functions

- long info (Info *info)

  Return the goos information of the goos.
- long get_static_buffer (unsigned idx, L4::Ipc::Out< L4::Cap< L4Re::Dataspace > > rbuf)

  Return a static buffer of a goos.
- long create_buffer (unsigned long size, L4::Ipc::Out< L4::Cap< L4Re::Dataspace > > rbuf)

  Create a buffer.
- long delete_buffer (unsigned idx)

  Delete a buffer.
- int create_view (View *view, l4_utcb_t *utcb=l4_utcb()) const throw ()

  Create a view.
- int delete_view (View const &v, l4_utcb_t *utcb=l4_utcb()) const throw ()

  Delete a view.
- View view (unsigned index) const throw ()

  Return a view.
- long refresh (int x, int y, int w, int h)

  Trigger refreshing of the given area on the virtual screen.

Additional Inherited Members

12.262.1 Detailed Description

A goos.

Definition at line 207 of file goos.

12.262.2 Member Enumeration Documentation

12.262.2.1 enum L4Re::Video::Goos::Flags

Flags for a goos.

Enumerator

F_auto_refresh The graphics display is automatically refreshed.

F_pointer We have a mouse pointer.

F_dynamic_views Supports dynamically allocated views.

F_dynamic_buffers Supports dynamically allocated buffers.

Definition at line 212 of file goos.

12.262.3 Member Function Documentation

12.262.3.1 long L4Re::Video::Goos::create_buffer (unsigned long *size*, L4::Ipc::Out< L4::Cap< L4Re::Dataspace > > *rbuf*)

Create a buffer.

Parameters

size	Size of buffer in bytes.
rbuf	Capability slot to point the buffer dataspace to.

Return values

>=0	Success, the value returned is the buffer index.
<0	Error

12.262.3.2 int L4Re::Video::Goos::create_view (View * *view*, l4_utcb_t * *utcb* = l4_utcb ()) const throw) [inline]

Create a view.

Parameters

out	*view*	A view object.
	utcb	UTCB of the caller. This is a default parameter.

Return values

>=0	Success, the value returned is the view index.
<0	Error

Definition at line 296 of file goos.

12.262.3.3 long L4Re::Video::Goos::delete_buffer (unsigned *idx*)

Delete a buffer.

Parameters

idx	Buffer to delete.

Return values

0	Success
<0	Error

12.262.3.4 int L4Re::Video::Goos::delete_view (View const & *v*, l4_utcb_t * *utcb* = l4_utcb ()) const throw) [inline]

Delete a view.

Parameters

v	The view object to delete.
utcb	UTCB of the caller. This is a default parameter.

Return values

0	Success
<0	Error

Definition at line 316 of file goos.

12.262.3.5 long L4Re::Video::Goos::get_static_buffer (unsigned *idx*, L4::Ipc::Out< L4::Cap< L4Re::Dataspace > > *rbuf*)

Return a static buffer of a goos.

Parameters

idx	Index of the static buffer.
rbuf	Capability slot to point the buffer dataspace to.

Return values

0	Success
<0	Error

12.262.3.6 long L4Re::Video::Goos::info (Info * *info*)

Return the goos information of the goos.

Parameters

out	*info*	Goos information structure pointer.

Return values

0	Success
<0	Error

12.262.3.7 View L4Re::Video::Goos::view (unsigned *index*) const throw) [inline]

Return a view.

Parameters

index	Index of the view to return.

Returns

The view.

Definition at line 347 of file goos.

The documentation for this class was generated from the following file:

- l4/re/video/goos

12.263 L4Re::Video::Goos::Info Struct Reference

Information structure of a goos.

Collaboration diagram for L4Re::Video::Goos::Info:

Public Member Functions

- bool auto_refresh () const

  Return whether this goos does auto refreshing or the view refresh functions must be used to make changes visible.
- bool has_pointer () const

  Return whether a pointer is used by the provider of the goos.
- bool has_dynamic_views () const

  Return whether dynamic view are supported.
- bool has_dynamic_buffers () const

  Return whether dynamic buffers are supported.

Data Fields

- unsigned long width

  Width.

- unsigned long height

  Height.
- unsigned flags

  Flags, see Flags.
- unsigned num_static_views

  Number of static view.
- unsigned num_static_buffers

  Number of static buffers.
- Pixel_info pixel_info

  Pixel information.

12.263.1 Detailed Description

Information structure of a goos.

Definition at line 221 of file goos.

12.263.2 Member Function Documentation

12.263.2.1 bool L4Re::Video::Goos::Info::auto_refresh ( ) const [inline]

Return whether this goos does auto refreshing or the view refresh functions must be used to make changes visible.

Definition at line 232 of file goos.

The documentation for this struct was generated from the following file:

- l4/re/video/goos

12.264 L4Re::Video::Pixel_info Class Reference

Pixel information.

Collaboration diagram for L4Re::Video::Pixel_info:

Public Member Functions

- Color_component const & r () const

  Return the red color compoment of the pixel.
- Color_component const & g () const

  Return the green color compoment of the pixel.
- Color_component const & b () const

  Return the blue color compoment of the pixel.
- Color_component const & a () const

  Return the alpha color compoment of the pixel.
- unsigned char bytes_per_pixel () const

  Query size of pixel in bytes.
- unsigned char bits_per_pixel () const

  Number of bits of the pixel.
- bool has_alpha () const

  Return whether the pixel has an alpha channel.
- void r (Color_component const &c)

  Set the red color component of the pixel.
- void g (Color_component const &c)

  Set the green color component of the pixel.
- void b (Color_component const &c)

  Set the blue color component of the pixel.
- void a (Color_component const &c)

  Set the alpha color component of the pixel.
- void bytes_per_pixel (unsigned char bpp)

  Set the size of the pixel in bytes.
- Pixel_info ()

  Constructor.
- Pixel_info (unsigned char bpp, char r, char rs, char g, char gs, char b, char bs, char a=0, char as=0)

  Constructor.
- template<typename VBI >
  Pixel_info (VBI const *vbi)

  Convenience constructor.
- bool operator== (Pixel_info const &o) const

  Compare for complete equzality of the color sapce.
- template<typename OUT >
  void dump (OUT &s) const

  Dump information on the pixel to a stream.

12.264.1 Detailed Description

Pixel information.

This class wraps the information on a pixel, such as the size and position of each color component in the pixel.

Definition at line 106 of file colors.

12.264.2 Constructor & Destructor Documentation

12.264.2.1 L4Re::Video::Pixel_info::Pixel_info (unsigned char *bpp*, char *r*, char *rs*, char *g*, char *gs*, char *b*, char *bs*, char *a* = 0, char *as* = 0) [inline]

Constructor.

Parameters

bpp	Size of pixel in bytes.
r	Red component size.
rs	Red component shift.
g	Green component size.
gs	Green component shift.
b	Blue component size.
bs	Blue component shift.
a	Alpha component size, defaults to 0.
as	Alpha component shift, defaults to 0.

Definition at line 203 of file colors.

12.264.2.2 template<typename VBI > L4Re::Video::Pixel_info::Pixel_info (VBI const * *vbi*) [inline], [explicit]

Convenience constructor.

Parameters

vbi	Suitable information structure. Convenience constructor to create the pixel info from a VESA Framebuffer Info.

Definition at line 215 of file colors.

12.264.3 Member Function Documentation

12.264.3.1 Color_component const& L4Re::Video::Pixel_info::a ( ) const [inline]

Return the alpha color compoment of the pixel.

Returns

Alpha color component.

Definition at line 135 of file colors.

12.264.3.2 void L4Re::Video::Pixel_info::a (Color_component const & *c*) [inline]

Set the alpha color component of the pixel.

Parameters

c	Alpha color component.

Definition at line 178 of file colors.

12.264.3.3 Color_component const& L4Re::Video::Pixel_info::b ( ) const [inline]

Return the blue color compoment of the pixel.

Returns

Blue color component.

Definition at line 129 of file colors.

12.264.3.4 void L4Re::Video::Pixel_info::b (Color_component const & *c*) [inline]

Set the blue color component of the pixel.

Parameters

c	Blue color component.

Definition at line 172 of file colors.

12.264.3.5 unsigned char L4Re::Video::Pixel_info::bits_per_pixel ( ) const [inline]

Number of bits of the pixel.

Returns

Number of bits used by the pixel.

Definition at line 147 of file colors.

References L4Re::Video::Color_component::size().

Here is the call graph for this function:

12.264.3.6 unsigned char L4Re::Video::Pixel_info::bytes_per_pixel ( ) const [inline]

Query size of pixel in bytes.

Returns

Size of pixel in bytes.

Definition at line 141 of file colors.

12.264.3.7 void L4Re::Video::Pixel_info::bytes_per_pixel (unsigned char *bpp*) [inline]

Set the size of the pixel in bytes.

Parameters

bpp	Size of pixel in bytes.

Definition at line 184 of file colors.

12.264.3.8 template<typename OUT > void L4Re::Video::Pixel_info::dump (OUT & *s*) const [inline]

Dump information on the pixel to a stream.

Parameters

s	Stream

Returns

The stream

Definition at line 238 of file colors.

Referenced by L4Re::Video::View::Info::dump().

Here is the caller graph for this function:

12.264.3.9 Color_component const& L4Re::Video::Pixel_info::g ( ) const [inline]

Return the green color compoment of the pixel.

Returns

Green color component.

Definition at line 123 of file colors.

12.264.3.10 void L4Re::Video::Pixel_info::g (Color_component const & *c*) [inline]

Set the green color component of the pixel.

Parameters

c	Green color component.

Definition at line 166 of file colors.

12.264.3.11 bool L4Re::Video::Pixel_info::has_alpha ( ) const [inline]

Return whether the pixel has an alpha channel.

Returns

True if the pixel has an alpha channel, false if not.

Definition at line 154 of file colors.

References L4Re::Video::Color_component::size().

Here is the call graph for this function:

12.264.3.12 bool L4Re::Video::Pixel_info::operator== (Pixel_info const & *o*) const [inline]

Compare for complete equzality of the color sapce.

Parameters

o	A Pixel_info to compare to.

Returns

true if the both Pixel_info's are equal, false if not.

Definition at line 227 of file colors.

12.264.3.13 Color_component const& L4Re::Video::Pixel_info::r ( ) const [inline]

Return the red color compoment of the pixel.

Returns

Red color component.

Definition at line 117 of file colors.

12.264.3.14 void L4Re::Video::Pixel_info::r (Color_component const & *c*) [inline]

Set the red color component of the pixel.

Parameters

c	Red color component.

Definition at line 160 of file colors.

The documentation for this class was generated from the following file:

- l4/re/video/colors

12.265 L4Re::Video::View Class Reference

View.

Collaboration diagram for L4Re::Video::View:

Data Structures

- struct Info

  Information structure of a view.

Public Types

- enum Flags {
  F_none = 0x00, F_set_buffer = 0x01, F_set_buffer_offset = 0x02, F_set_bytes_per_line = 0x04, F_set_pixel = 0x08, F_set_position = 0x10, F_dyn_allocated = 0x20, F_set_background = 0x40, F_set_flags = 0x80, F_fully_dynamic }

  Flags on a view.
- enum V_flags { F_above = 0x1000, F_flags_mask = 0xff000 }

  Property flags of a view.

Public Member Functions

- int info (Info *info) const throw ()

  Return the view information of the view.
- int set_info (Info const &info) const throw ()

  Set the information structure for this view.
- int set_viewport (int scr_x, int scr_y, int w, int h, unsigned long buf_offset) const throw ()

  Set the position of the view in the goos.
- int stack (View const &pivot, bool behind=true) const throw ()

  Move this view in the view stack.
- int push_top () const throw ()

  Make this view the top-most view.
- int push_bottom () const throw ()

  Push this view the back.
- int refresh (int x, int y, int w, int h) const throw ()

Refresh/Redraw the view.

- bool valid () const

  Return whether this view is valid.

12.265.1 Detailed Description

View.

Definition at line 34 of file goos.

12.265.2 Member Enumeration Documentation

12.265.2.1 enum L4Re::Video::View::Flags

Flags on a view.

Enumerator

F_none everything for this view is static (the VESA-FB case)

F_set_buffer buffer object for this view can be changed

F_set_buffer_offset buffer offset can be set

F_set_bytes_per_line bytes per line can be set

F_set_pixel pixel type can be set

F_set_position position on screen can be set

F_dyn_allocated View is dynamically allocated.

F_set_background Set view as background for session.

F_set_flags Set view flags (.

See also

V_flags)

F_fully_dynamic Flags for a fully dynamic view.

Definition at line 54 of file goos.

12.265.2.2 enum L4Re::Video::View::V_flags

Property flags of a view.

Such flags can be set or deleted with the F_set_flags operation using the `set_info()` method.

Enumerator

F_above Flag the view as stay on top.

F_flags_mask Mask containing all possible property flags.

Definition at line 77 of file goos.

12.265.3 Member Function Documentation

12.265.3.1 int L4Re::Video::View::info (Info * *info*) const throw) [inline]

Return the view information of the view.

Parameters

out	*info*	Information structure pointer.

Return values

0	Success
<0	Error

Definition at line 351 of file goos.

12.265.3.2 int L4Re::Video::View::refresh (int *x*, int *y*, int *w*, int *h*) const throw) [inline]

Refresh/Redraw the view.

Parameters

x	X position.
y	Y position.
w	Width.
h	Height.

Return values

0	Success
<0	Error

Definition at line 363 of file goos.

12.265.3.3 int L4Re::Video::View::set_info (Info const & *info*) const throw) [inline]

Set the information structure for this view.

Parameters

info	Information structure.

Return values

0	Success
<0	Error

The function will also set the view port according to the values given in the information structure.

Definition at line 355 of file goos.

12.265.3.4 int L4Re::Video::View::set_viewport (int *scr_x*, int *scr_y*, int *w*, int *h*, unsigned long *buf_offset*) const throw) [inline]

Set the position of the view in the goos.

Parameters

scr_x	X position
scr_y	Y position
w	Width
h	Height
buf_offset	Offset in the buffer in bytes

Return values

0	Success
<0	Error

Definition at line 367 of file goos.

References L4Re::Video::View::Info::buffer_offset, L4Re::Video::View::Info::flags, L4Re::Video::View::Info::height, L4Re::Video::View::Info::width, L4Re::Video::View::Info::xpos, and L4Re::Video::View::Info::ypos.

12.265.3.5 int L4Re::Video::View::stack (View const & *pivot*, bool *behind* = `true`) const throw) [inline]

Move this view in the view stack.

Parameters

pivot	View to move relative to
behind	When true move the view behind the pivot view, if false move the view before the pivot view.

Return values

0	Success
<0	Error

Definition at line 359 of file goos.

The documentation for this class was generated from the following file:

- l4/re/video/goos

12.266 L4Re::Video::View::Info Struct Reference

Information structure of a view.

Collaboration diagram for L4Re::Video::View::Info:

Public Member Functions

- bool has_static_buffer () const

  Return whether the view has a static buffer.
- bool has_static_buffer_offset () const

  Return whether the static buffer offset is available.
- bool has_set_buffer () const

  Return whether a buffer is set.
- bool has_set_buffer_offset () const

Return whether the given buffer offset is valid.

- bool has_set_bytes_per_line () const

  Return whether the given bytes-per-line value is valid.

- bool has_set_pixel () const

  Return whether the given pixel information is valid.

- bool has_set_position () const

  Return whether the position information given is valid.

- template<typename OUT >
  void dump (OUT &s) const

  Dump information on the view information to a stream.

Data Fields

- unsigned flags

  Flags, see Flags and V_flags.

- unsigned view_index

  Index of the view.

- unsigned long xpos

  X position in pixels of the view in the goos.

- unsigned long ypos

  Y position in pixels of the view in the goos.

- unsigned long width

  Width of the view in pixels.

- unsigned long height

  Height of the view in pixels.

- unsigned long buffer_offset

  Offset in the memory buffer in bytes.

- unsigned long bytes_per_line

  Bytes per line.

- Pixel_info pixel_info

  Pixel information.

- unsigned buffer_index

  Number of the buffer used for this view.

12.266.1 Detailed Description

Information structure of a view.

Definition at line 86 of file goos.

The documentation for this struct was generated from the following file:

- l4/re/video/goos

12.267 l4re_aux_t Struct Reference

Auxiliary descriptor.

```
#include <l4aux.h>
```

Collaboration diagram for l4re_aux_t:

Data Fields

- char const * binary

  Binary name.

- l4_cap_idx_t kip_ds

  Data space of the KIP.

- l4_umword_t dbg_lvl

  Debug levels for l4re.

- l4_umword_t ldr_flags

  Flags for l4re, see l4re_aux_ldr_flags_t.

12.267.1 Detailed Description

Auxiliary descriptor.

Definition at line 51 of file l4aux.h.

The documentation for this struct was generated from the following file:

- l4/re/l4aux.h

12.268 l4re_ds_stats_t Struct Reference

Information about the data space.

```
#include <dataspace.h>
```

Collaboration diagram for l4re_ds_stats_t:

Data Fields

- unsigned long size

  size
- unsigned long flags

  flags

12.268.1 Detailed Description

Information about the data space.

Definition at line 45 of file dataspace.h.

The documentation for this struct was generated from the following file:

- l4/re/c/dataspace.h

12.269 l4re_elf_aux_mword_t Struct Reference

Auxiliary vector element for a single unsigned data word.

`#include <elf_aux.h>`

Collaboration diagram for l4re_elf_aux_mword_t:

12.269.1 Detailed Description

Auxiliary vector element for a single unsigned data word.

Definition at line 124 of file elf_aux.h.

The documentation for this struct was generated from the following file:

- l4/re/elf_aux.h

12.270 l4re_elf_aux_t Struct Reference

Generic header for each auxiliary vector element.

```
#include <elf_aux.h>
```

Collaboration diagram for l4re_elf_aux_t:

12.270.1 Detailed Description

Generic header for each auxiliary vector element.

Definition at line 104 of file elf_aux.h.

The documentation for this struct was generated from the following file:

- l4/re/elf_aux.h

12.271 l4re_elf_aux_vma_t Struct Reference

Auxiliary vector element for a reserved virtual memory area.

```
#include <elf_aux.h>
```

Collaboration diagram for l4re_elf_aux_vma_t:

12.271.1 Detailed Description

Auxiliary vector element for a reserved virtual memory area.

Definition at line 113 of file elf_aux.h.

The documentation for this struct was generated from the following file:

- l4/re/elf_aux.h

12.272 l4re_env_cap_entry_t Struct Reference

Entry in the L4Re environment array for the named inital objects.

```
#include <env.h>
```

Collaboration diagram for l4re_env_cap_entry_t:

Public Member Functions

- l4re_env_cap_entry_t ()

Create an invalid entry.

- l4re_env_cap_entry_t (char const *n, l4_cap_idx_t c, l4_umword_t f=0)

  Create an entry with the name n, capability c, and flags f.

Data Fields

- l4_cap_idx_t cap

  The capability selector for the obeject.

- l4_umword_t flags

  Some flags for the object.

- char name [16]

  The name of the object.

12.272.1 Detailed Description

Entry in the L4Re environment array for the named inital objects.

Definition at line 35 of file env.h.

12.272.2 Constructor & Destructor Documentation

12.272.2.1 l4re_env_cap_entry_t::l4re_env_cap_entry_t (char const * *n*, l4_cap_idx_t *c*, l4_umword_t *f* = 0) [inline]

Create an entry with the name *n*, capability *c*, and flags *f*.

Parameters

n	is the name of the initial object.
c	is the capability selector that refers the initial object.
f	are the additional flags for the object.

Definition at line 67 of file env.h.

References name.

12.272.3 Field Documentation

12.272.3.1 l4_umword_t l4re_env_cap_entry_t::flags

Some flags for the object.

Note

Currently unused.

Definition at line 46 of file env.h.

Referenced by l4re_env_get_cap_l().

The documentation for this struct was generated from the following file:

- l4/re/env.h

12.273 l4re_env_t Struct Reference

Initial Environment structure (C version)

```
#include <env.h>
```

Collaboration diagram for l4re_env_t:

Data Fields

- l4_cap_idx_t parent

  Parent object-capability.
- l4_cap_idx_t rm

  Region map object-capability.
- l4_cap_idx_t mem_alloc

  Memory allocator object-capability.
- l4_cap_idx_t log

  Logging object-capability.
- l4_cap_idx_t main_thread

  Object-capability of the first user thread.
- l4_cap_idx_t factory

  Object-capability of the factory available to the task.
- l4_cap_idx_t scheduler

Object capability for the scheduler set to use.

- l4_cap_idx_t first_free_cap

  First capability index available to the application.
- l4_fpage_t utcb_area

  UTCB area of the task.
- l4_addr_t first_free_utcb

  First UTCB within the UTCB area available to the application.

12.273.1 Detailed Description

Initial Environment structure (C version)

See also

Initial environment

Definition at line 96 of file env.h.

The documentation for this struct was generated from the following file:

- l4/re/env.h

12.274 l4re_event_t Struct Reference

Event structure used in buffer.

```
#include <event.h>
```

Collaboration diagram for l4re_event_t:

Data Fields

- long long time

  Time stamp of the event.
- unsigned short type

  Type of the event.
- unsigned short code

  Code of the event.

- int value

  Value of the event.
- l4_umword_t stream_id

  Stream ID.

12.274.1 Detailed Description

Event structure used in buffer.

Definition at line 40 of file event.h.

The documentation for this struct was generated from the following file:

- l4/re/c/event.h

12.275 l4re_video_color_component_t Struct Reference

Color component structure.

```
#include <colors.h>
```

Collaboration diagram for l4re_video_color_component_t:

Data Fields

- unsigned char size

  Size in bits.
- unsigned char shift

  offset in pixel

12.275.1 Detailed Description

Color component structure.

Definition at line 31 of file colors.h.

The documentation for this struct was generated from the following file:

- l4/re/c/video/colors.h

12.276 l4re_video_goos_info_t Struct Reference

Goos information structure.

```
#include <goos.h>
```

Collaboration diagram for l4re_video_goos_info_t:

Data Fields

- unsigned long width

  Width of the goos.
- unsigned long height

  Height of the goos.
- unsigned flags

  Flags of the framebuffer, see l4re_video_goos_info_flags_t.
- unsigned num_static_views

  Number of static views.
- unsigned num_static_buffers

Number of static buffers.

- l4re_video_pixel_info_t pixel_info

  Pixel layout of the goos.

12.276.1 Detailed Description

Goos information structure.

Definition at line 51 of file goos.h.

The documentation for this struct was generated from the following file:

- l4/re/c/video/goos.h

12.277 l4re_video_pixel_info_t Struct Reference

Pixel_info structure.

```
#include <colors.h>
```

Collaboration diagram for l4re_video_pixel_info_t:

Data Fields

- l4re_video_color_component_t a

  Colors.

- unsigned char bytes_per_pixel

  Bytes per pixel.

12.277.1 Detailed Description

Pixel_info structure.

Definition at line 41 of file colors.h.

The documentation for this struct was generated from the following file:

- l4/re/c/video/colors.h

12.278 l4re_video_view_info_t Struct Reference

View information structure.

```
#include <view.h>
```

Collaboration diagram for l4re_video_view_info_t:

Data Fields

- unsigned flags

  Flags.
- unsigned view_index

  Number of view in the goos.
- unsigned long height

  Position in goos and size of view.
- unsigned long buffer_offset

  Memory offset in goos buffer.
- unsigned long bytes_per_line

  Size of line in view.
- l4re_video_pixel_info_t pixel_info

  Pixel info.
- unsigned buffer_index

  Number of buffer of goos.

12.278.1 Detailed Description

View information structure.

Definition at line 59 of file view.h.

The documentation for this struct was generated from the following file:

- l4/re/c/video/view.h

12.279 l4re_video_view_t Struct Reference

C representation of a goos view.

```
#include <view.h>
```

Collaboration diagram for l4re_video_view_t:

12.279.1 Detailed Description

C representation of a goos view.

A view is a visible rectangle that provides a view to the contents of a buffer (frame buffer) memory object and is placed on a real screen.

Definition at line 78 of file view.h.

The documentation for this struct was generated from the following file:

- l4/re/c/video/view.h

12.280 l4util_idt_desc_t Struct Reference

IDT entry.

```
#include <idt.h>
```

Collaboration diagram for l4util_idt_desc_t:

Data Fields

- l4_uint64_t b

  see Intel doc
- l4_uint32_t b

  see Intel doc

12.280.1 Detailed Description

IDT entry.

Definition at line 33 of file idt.h.

The documentation for this struct was generated from the following file:

- amd64/l4/util/idt.h

12.281 l4util_idt_header_t Struct Reference

Header of an IDT table.

```
#include <idt.h>
```

Collaboration diagram for l4util_idt_header_t:

Data Fields

- l4_uint16_t limit

  limit field (see Intel doc)

- void * base

  idt base (see Intel doc)

12.281.1 Detailed Description

Header of an IDT table.

Definition at line 40 of file idt.h.

The documentation for this struct was generated from the following file:

- amd64/l4/util/idt.h

12.282 l4util_mb_addr_range_t Struct Reference

INT-15, AX=E820 style "AddressRangeDescriptor" ...with a "size" parameter on the front which is the structure size - 4, pointing to the next one, up until the full buffer length of the memory map has been reached.

```
#include <mb_info.h>
```

Collaboration diagram for l4util_mb_addr_range_t:

Data Fields

- l4_uint64_t addr

  <Size of structure

- l4_uint64_t size

  <Start address

- l4_uint32_t type

  <Size of memory range

12.282.1 Detailed Description

INT-15, AX=E820 style "AddressRangeDescriptor" ...with a "size" parameter on the front which is the structure size - 4, pointing to the next one, up until the full buffer length of the memory map has been reached.

Definition at line 43 of file mb_info.h.

The documentation for this struct was generated from the following file:

- l4/util/mb_info.h

12.283 l4util_mb_apm_t Struct Reference

APM BIOS info.

```
#include <mb_info.h>
```

Collaboration diagram for l4util_mb_apm_t:

12.283.1 Detailed Description

APM BIOS info.

Definition at line 91 of file mb_info.h.

The documentation for this struct was generated from the following file:

- l4/util/mb_info.h

12.284 l4util_mb_drive_t Struct Reference

Drive Info structure.

```
#include <mb_info.h>
```

Collaboration diagram for l4util_mb_drive_t:

Data Fields

- l4_uint8_t drive_number

  <The size of this structure.
- l4_uint8_t drive_mode

  <The BIOS drive number.
- l4_uint16_t drive_cylinders

  <The access mode (see below).
- l4_uint8_t drive_heads

  <number of cylinders
- l4_uint8_t drive_sectors

  <number of heads
- l4_uint16_t drive_ports [0]

  <number of sectors per track

12.284.1 Detailed Description

Drive Info structure.

Definition at line 74 of file mb_info.h.

12.284.2 Field Documentation

12.284.2.1 l4_uint16_t l4util_mb_drive_t::drive_cylinders

<The access mode (see below).

Definition at line 79 of file mb_info.h.

12.284.2.2 l4_uint8_t l4util_mb_drive_t::drive_mode

<The BIOS drive number.

Definition at line 78 of file mb_info.h.

12.284.2.3 l4_uint8_t l4util_mb_drive_t::drive_number

<The size of this structure.

Definition at line 77 of file mb_info.h.

The documentation for this struct was generated from the following file:

- l4/util/mb_info.h

12.285 l4util_mb_info_t Struct Reference

```
#include <mb_info.h>
```

Collaboration diagram for l4util_mb_info_t:

Data Fields

- l4_uint32_t flags

  MultiBoot info version number.
- l4_uint32_t mem_lower

  available memory below 1MB
- l4_uint32_t mem_upper

  available memory starting from 1MB [kB]
- l4_uint32_t boot_device

  "root" partition
- l4_uint32_t cmdline

  Kernel command line.
- l4_uint32_t mods_count

  number of modules
- l4_uint32_t mods_addr

  module list
- l4_uint32_t mmap_length

  size of memory mapping buffer
- l4_uint32_t mmap_addr

  address of memory mapping buffer
- l4_uint32_t drives_length

  size of drive info buffer
- l4_uint32_t drives_addr

  address of driver info buffer
- l4_uint32_t config_table

  ROM configuration table.
- l4_uint32_t boot_loader_name

  Boot Loader Name.
- l4_uint32_t apm_table

APM table.

- l4_uint32_t vbe_ctrl_info

  VESA video contoller info.

- l4_uint32_t vbe_mode_info

  VESA video mode info.

- l4_uint16_t vbe_mode

  VESA video mode number.

- l4_uint16_t vbe_interface_seg

  VESA segment of prot BIOS interface.

- l4_uint16_t vbe_interface_off

  VESA offset of prot BIOS interface.

- l4_uint16_t vbe_interface_len

  VESA lenght of prot BIOS interface.

- l4_uint32_t tabsize

  (a.out) Kernel symbol table info

- l4_uint32_t num

  (ELF) Kernel section header table

12.285.1 Detailed Description

MultiBoot Info description

This is the struct passed to the boot image. This is done by placing its address in the EAX register.

Definition at line 203 of file mb_info.h.

The documentation for this struct was generated from the following file:

- l4/util/mb_info.h

12.286 l4util_mb_mod_t Struct Reference

```
#include <mb_info.h>
```

Collaboration diagram for l4util_mb_mod_t:

Data Fields

- l4_uint32_t mod_start

  Starting address of module in memory.

- l4_uint32_t mod_end

  End address of module in memory.

- l4_uint32_t cmdline

  Module command line.

- l4_uint32_t pad

  padding to take it to 16 bytes

12.286.1 Detailed Description

The structure type "mod_list" is used by the multiboot_info structure.

Definition at line 27 of file mb_info.h.

12.286.2 Field Documentation

12.286.2.1 l4_uint32_t l4util_mb_mod_t::mod_end

End address of module in memory.

Definition at line 30 of file mb_info.h.

12.286.2.2 l4_uint32_t l4util_mb_mod_t::mod_start

Starting address of module in memory.

Definition at line 29 of file mb_info.h.

The documentation for this struct was generated from the following file:

- l4/util/mb_info.h

12.287 l4util_mb_vbe_ctrl_t Struct Reference

VBE controller information.

```
#include <mb_info.h>
```

Collaboration diagram for l4util_mb_vbe_ctrl_t:

12.287.1 Detailed Description

VBE controller information.

Definition at line 105 of file mb_info.h.

The documentation for this struct was generated from the following file:

- l4/util/mb_info.h

12.288 l4util_mb_vbe_mode_t Struct Reference

VBE mode information.

```
#include <mb_info.h>
```

Collaboration diagram for l4util_mb_vbe_mode_t:

Data Fields

all VESA versions

- l4_uint16_t **mode_attributes**
- l4_uint8_t **win_a_attributes**
- l4_uint8_t **win_b_attributes**
- l4_uint16_t **win_granularity**

- l4_uint16_t **win_size**
- l4_uint16_t **win_a_segment**
- l4_uint16_t **win_b_segment**
- l4_uint32_t **win_func**
- l4_uint16_t **bytes_per_scanline**

>= VESA version 1.2

- l4_uint16_t **x_resolution**
- l4_uint16_t **y_resolution**
- l4_uint8_t **x_char_size**
- l4_uint8_t **y_char_size**
- l4_uint8_t **number_of_planes**
- l4_uint8_t **bits_per_pixel**
- l4_uint8_t **number_of_banks**
- l4_uint8_t **memory_model**
- l4_uint8_t **bank_size**
- l4_uint8_t **number_of_image_pages**
- l4_uint8_t **reserved0**

direct color

- l4_uint8_t **red_mask_size**
- l4_uint8_t **red_field_position**
- l4_uint8_t **green_mask_size**
- l4_uint8_t **green_field_position**
- l4_uint8_t **blue_mask_size**
- l4_uint8_t **blue_field_position**
- l4_uint8_t **reserved_mask_size**
- l4_uint8_t **reserved_field_position**
- l4_uint8_t **direct_color_mode_info**

>= VESA version 2.0

- l4_uint32_t **phys_base**
- l4_uint32_t **reserved1**
- l4_uint16_t **reversed2**

>= VESA version 3.0

- l4_uint16_t **linear_bytes_per_scanline**
- l4_uint8_t **banked_number_of_image_pages**
- l4_uint8_t **linear_number_of_image_pages**
- l4_uint8_t **linear_red_mask_size**
- l4_uint8_t **linear_red_field_position**
- l4_uint8_t **linear_green_mask_size**
- l4_uint8_t **linear_green_field_position**
- l4_uint8_t **linear_blue_mask_size**
- l4_uint8_t **linear_blue_field_position**
- l4_uint8_t **linear_reserved_mask_size**
- l4_uint8_t **linear_reserved_field_position**
- l4_uint32_t **max_pixel_clock**
- l4_uint8_t **reserved3** [189+1]

12.288.1 Detailed Description

VBE mode information.

Definition at line 123 of file mb_info.h.

The documentation for this struct was generated from the following file:

- l4/util/mb_info.h

12.289 L4vbus::Device Class Reference

Device on a virtual bus (V-BUS)

Inheritance diagram for L4vbus::Device:

Collaboration diagram for L4vbus::Device:

Public Member Functions

- L4::Cap< Vbus > bus_cap () const

  access the V-BUS capability of the underlying virtual bus.

- l4vbus_device_handle_t dev_handle () const

  access the device handle of this device.

- int device_by_hid (Device *child, char const *hid, int depth=L4VBUS_MAX_DEPTH, l4vbus_device_t *devinfo=0) const

  Find a device by the HID.

- int next_device (Device *child, int depth=L4VBUS_MAX_DEPTH, l4vbus_device_t *devinfo=0) const

  Find next child following child.

- int get_resource (int res_idx, l4vbus_resource_t *res) const

  Iterate over the resources of a device.

- int is_compatible (char const *cid) const

  Check if the given device has a compatibility ID (CID) or HID that matches cid.

- bool operator== (Device const &o) const

  Test if two devices are the same V-BUS device.

- bool operator!= (Device const &o) const

  Test if two devices are not the same.

Protected Attributes

- L4::Cap< Vbus > _bus
- l4vbus_device_handle_t _dev

  The device handle for this device.

12.289.1 Detailed Description

Device on a virtual bus (V-BUS)

Definition at line 56 of file vbus.

12.289.2 Member Function Documentation

12.289.2.1 L4::Cap<Vbus> L4vbus::Device::bus_cap ( ) const [inline]

access the V-BUS capability of the underlying virtual bus.

Returns

the capability to the underlying V-BUS.

Definition at line 68 of file vbus.

References _bus.

Referenced by L4vbus::Pci_host_bridge::cfg_read(), L4vbus::Pci_dev::cfg_read(), L4vbus::Pci_host_bridge::cfg_write(), L4vbus::Pci_dev::cfg_write(), L4vbus::Pci_host_bridge::irq_enable(), and L4vbus::Pci_dev::irq_enable().

Here is the caller graph for this function:

12.289.2.2 l4vbus_device_handle_t L4vbus::Device::dev_handle ( ) const [inline]

access the device handle of this device.

Returns

the device handle for this device.

The device handle is used to directly address the device on its virtual bus.

Definition at line 77 of file vbus.

References _dev.

12.289.2.3 int L4vbus::Device::device_by_hid (Device * *child*, char const * *hid*, int *depth* = L4VBUS_MAX_DEPTH, l4vbus_device_t * *devinfo* = 0) const [inline]

Find a device by the HID.

This function searches the vbus for a device with the given HID and returns a handle to the first matching device. The HID usually conforms to an ACPI HID or a Linux device tree compatible ID.

It is possible to have multiple devices with the same HID on a vbus. In order to find all matching devices this function has to be called repeatedly with `child` pointing to the device found in the previous iteration. The iteration starts at `child` that might be any device node in the tree.

Parameters

in,out	*child*	Handle of the device from where in the device tree the search should start. To start searching from the beginning `child` must be initialized using the default (#L4VBUS_NULL). If a matching device is found its handle is returned through this parameter.
	hid	HID of the device
	depth	Maximum depth for the recursive lookup
out	*devinfo*	Device information structure (might be NULL)

Return values

>=	0 A device with the given HID was found.
-L4_ENODEV	No device with the given HID could be found on the vbus.
-L4_ENOENT	Invalid or no HID provided.

Definition at line 108 of file vbus.

References _bus, _dev, and l4vbus_get_device_by_hid().

Here is the call graph for this function:

12.289.2.4 int L4vbus::Device::get_resource (int *res_idx*, l4vbus_resource_t * *res*) const [inline]

Iterate over the resources of a device.

Parameters

out	*res_idx*	Index of the resource, the number of resources is available in the devinfo from get device functions.
out	*res*	Descriptor of the resource

Returns

0 on success, else failure

Definition at line 143 of file vbus.

References _bus, _dev, and l4vbus_get_resource().

Here is the call graph for this function:

12.289.2.5 int L4vbus::Device::is_compatible (char const * *cid*) const [inline]

Check if the given device has a compatibility ID (CID) or HID that matches *cid*.

Parameters

cid	the compatibility ID to test

Returns

1 when the given ID (*cid*) matches this device, 0 when the given ID does not match, <0 on error.

Definition at line 157 of file vbus.

References _bus, _dev, and l4vbus_is_compatible().

Here is the call graph for this function:

12.289.2.6 int L4vbus::Device::next_device (Device * *child*, int *depth* = L4VBUS_MAX_DEPTH, l4vbus_device_t * *devinfo* = 0) const [inline]

Find next child following *child*.

Parameters

	child	Device to start looking for child devices
	depth	Depth to look for
out	*devinfo*	device information (might be NULL)

Returns

0 on success, else failure

Definition at line 126 of file vbus.

References _bus, _dev, and l4vbus_get_next_device().

Here is the call graph for this function:

12.289.2.7 bool L4vbus::Device::operator!= (Device const & *o*) const [inline]

Test if two devices are not the same.

Returns

true if the two devices are different, false else.

Definition at line 173 of file vbus.

References _bus, and _dev.

12.289.2.8 bool L4vbus::Device::operator== (Device const & *o*) const [inline]

Test if two devices are the same V-BUS device.

Returns

true if the two devices are the same, false else.

Definition at line 164 of file vbus.

References _bus, and _dev.

12.289.3 Field Documentation

12.289.3.1 L4::Cap<Vbus> L4vbus::Device::_bus [protected]

The V-BUS capability (where this device is located on).

Definition at line 179 of file vbus.

Referenced by bus_cap(), L4vbus::Gpio_pin::config_get(), L4vbus::Gpio_pin::config_pad(), L4vbus::Gpio_module↩ ::config_pad(), L4vbus::Gpio_pin::config_pull(), device_by_hid(), L4vbus::Gpio_pin::get(), L4vbus::Gpio_module↩ ::get(), get_resource(), is_compatible(), next_device(), operator!=(), operator==(), L4vbus::Gpio_pin::set(), L4vbus↩ ::Gpio_module::set(), L4vbus::Gpio_pin::setup(), L4vbus::Gpio_module::setup(), L4vbus::Gpio_pin::to_irq(), and L4vbus::Icu::vicu().

The documentation for this class was generated from the following file:

- l4/vbus/vbus

12.290 L4vbus::Gpio_module Class Reference

A Gpio_module groups multiple GPIO pins together.

Inheritance diagram for L4vbus::Gpio_module:

Collaboration diagram for L4vbus::Gpio_module:

Data Structures

- struct Pin_slice

  A slice of the pins provided by this module.

Public Member Functions

- int setup (Pin_slice const &mask, unsigned mode, unsigned value) const

  Configure function of multiple GPIO pins at once.
- int config_pad (Pin_slice const &mask, unsigned func, unsigned value) const

  Hardware specific configuration function for multiple GPIO pins.
- int get (unsigned offset, unsigned ∗data) const

  Read values of multiple GPIO pins at once.
- int set (Pin_slice const &mask, unsigned data)

  Set multiple GPIO output pins at once.
- Gpio_pin pin (unsigned pin) const

  Get Gpio_pin for a specific pin of this Gpio_module.

Additional Inherited Members

12.290.1 Detailed Description

A Gpio_module groups multiple GPIO pins together.

Definition at line 129 of file vbus_gpio.

12.290.2 Member Function Documentation

12.290.2.1 int L4vbus::Gpio_module::config_pad (Pin_slice const & *mask,* unsigned *func,* unsigned *value*) const [inline]

Hardware specific configuration function for multiple GPIO pins.

Parameters

mask	Mask of GPIO pins to configure. A bit set to 1 configures this pin. A maximum of 32 pins can be configured at once. The real number depends on the hardware and the driver implementation.
func	Hardware specific configuration register, usually offset to the GPIO chip's base address.
value	Value which is written into the hardware specific configuration register for the specified pins

Returns

0 if OK, error code otherwise

Definition at line 181 of file vbus_gpio.

References L4vbus::Device::_bus, L4vbus::Device::_dev, and l4vbus_gpio_multi_config_pad().

Here is the call graph for this function:

12.290.2.2 int L4vbus::Gpio_module::get (unsigned *offset*, unsigned * *data*) const [inline]

Read values of multiple GPIO pins at once.

Parameters

	offset	Pin corresponding to the LSB in *data*. Note: allowed may be hardware specific.
`out`	*data*	Each bit returns the value (0 or 1) for the corresponding GPIO pin. The value of pins that are not accessible is undefined.

Returns

0 if OK, error code otherwise

Definition at line 197 of file vbus_gpio.

References L4vbus::Device::_bus, L4vbus::Device::_dev, and l4vbus_gpio_multi_get().

Here is the call graph for this function:

12.290.2.3 Gpio_pin L4vbus::Gpio_module::pin (unsigned *pin*) const `[inline]`

Get Gpio_pin for a specific pin of this Gpio_module.

Parameters

pin	GPIO pin number to get Gpio_pin for.

Returns

Gpio_pin

Definition at line 225 of file vbus_gpio.

12.290.2.4 int L4vbus::Gpio_module::set (Pin_slice const & *mask*, unsigned *data*) `[inline]`

Set multiple GPIO output pins at once.

Parameters

mask	Mask of GPIO pins to set. A bit set to 1 selects this pin. A maximum of 32 pins can be set at once. The real number depends on the hardware and the driver implementation.
data	Each bit corresponds to the GPIO pin in *mask*. The value of each bit is written to the GPIO pin if its bit in *mask* is set.

Returns

0 if OK, error code otherwise

Definition at line 213 of file vbus_gpio.

References L4vbus::Device::_bus, L4vbus::Device::_dev, and l4vbus_gpio_multi_set().

Here is the call graph for this function:

12.290.2.5 int L4vbus::Gpio_module::setup (Pin_slice const & *mask*, unsigned *mode*, unsigned *value*) const [inline]

Configure function of multiple GPIO pins at once.

Parameters

mask	Mask of GPIO pins to configure. A bit set to 1 configures this pin. A maximum of 32 pins can be configured at once. The real number depends on the hardware and the driver implementation.
mode	GPIO function, see L4vbus_gpio_generic_func for generic functions. Hardware specific functions must be provided in the lower 8 bits.
value	Optional value to set the GPIO pins to if they are configured as output pins

Returns

0 if OK, error code otherwise

Definition at line 162 of file vbus_gpio.

References L4vbus::Device::_bus, L4vbus::Device::_dev, and l4vbus_gpio_multi_setup().

Here is the call graph for this function:

The documentation for this class was generated from the following file:

- l4/vbus/vbus_gpio

12.291 L4vbus::Gpio_module::Pin_slice Struct Reference

A slice of the pins provided by this module.

Collaboration diagram for L4vbus::Gpio_module::Pin_slice:

12.291.1 Detailed Description

A slice of the pins provided by this module.

Data type to specify a selection of pins for the 'multi' methods.

Definition at line 142 of file vbus_gpio.

The documentation for this struct was generated from the following file:

- l4/vbus/vbus_gpio

12.292 L4vbus::Gpio_pin Class Reference

A GPIO pin.

Inheritance diagram for L4vbus::Gpio_pin:

Collaboration diagram for L4vbus::Gpio_pin:

Public Member Functions

- int get () const

  Read value of GPIO input pin.
- int set (int value) const

  Set GPIO output pin.
- int setup (unsigned mode, unsigned value) const

Configure the function of a GPIO pin.

- int config_pull (unsigned mode) const

  Generic function to set pull up/down mode.
- int config_pad (unsigned func, unsigned value) const

  Hardware specific configuration function.
- int config_get (unsigned func, unsigned ∗value) const

  Read hardware specific configuration.
- int to_irq () const

  Create IRQ for GPIO pin.
- unsigned pin () const

  Get pin number.

Additional Inherited Members

12.292.1 Detailed Description

A GPIO pin.

Definition at line 22 of file vbus_gpio.

12.292.2 Member Function Documentation

12.292.2.1 int L4vbus::Gpio_pin::config_get (unsigned *func,* unsigned ∗ *value*) const [inline]

Read hardware specific configuration.

Parameters

	func	Hardware specific configuration register to read from. Usually this is an offset to the GPIO chip's base address.
out	*value*	The configuration value.

Returns

0 if OK, error code otherwise

Definition at line 98 of file vbus_gpio.

References L4vbus::Device::_bus, L4vbus::Device::_dev, and l4vbus_gpio_config_get().

Here is the call graph for this function:

12.292.2.2 int L4vbus::Gpio_pin::config_pad (unsigned *func,* unsigned *value*) const [inline]

Hardware specific configuration function.

Parameters

func	Hardware specific configuration register, usually offset to the GPIO chip's base address
value	Value which is written into the hardware specific configuration register for the specified pin

Returns

0 if OK, error code otherwise

Definition at line 85 of file vbus_gpio.

References L4vbus::Device::_bus, L4vbus::Device::_dev, and l4vbus_gpio_config_pad().

Here is the call graph for this function:

12.292.2.3 int L4vbus::Gpio_pin::config_pull (unsigned *mode*) const [inline]

Generic function to set pull up/down mode.

Parameters

mode	mode for pull up/down resistors, see L4vbus_gpio_pull_modes

Returns

0 if OK, error code otherwise

Definition at line 71 of file vbus_gpio.

References L4vbus::Device::_bus, L4vbus::Device::_dev, and l4vbus_gpio_config_pull().

Here is the call graph for this function:

12.292.2.4 int L4vbus::Gpio_pin::get ( ) const [inline]

Read value of GPIO input pin.

Returns

Value of GPIO pin (usually 0 or 1), negative error code otherwise.

Definition at line 34 of file vbus_gpio.

References L4vbus::Device::_bus, L4vbus::Device::_dev, and l4vbus_gpio_get().

Here is the call graph for this function:

12.292.2.5 unsigned L4vbus::Gpio_pin::pin ( ) const [inline]

Get pin number.

Returns

GPIO pin number

Definition at line 118 of file vbus_gpio.

12.292.2.6 int L4vbus::Gpio_pin::set (int *value*) const [inline]

Set GPIO output pin.

Parameters

value	Value to write to the GPIO pin (usually 0 or 1)

Returns

0 if OK, error code otherwise

Definition at line 45 of file vbus_gpio.

References L4vbus::Device::_bus, L4vbus::Device::_dev, and l4vbus_gpio_set().

Here is the call graph for this function:

12.292.2.7 int L4vbus::Gpio_pin::setup (unsigned *mode,* unsigned *value*) const [inline]

Configure the function of a GPIO pin.

Parameters

mode	GPIO function, see L4vbus_gpio_generic_func for generic functions. Hardware specific functions must be provided in the lower 8 bits.
value	Optional value to set the GPIO pin to if it is configured as an output pin

Returns

0 if OK, error code otherwise

Definition at line 60 of file vbus_gpio.

References L4vbus::Device::_bus, L4vbus::Device::_dev, and l4vbus_gpio_setup().

Here is the call graph for this function:

12.292.2.8 int L4vbus::Gpio_pin::to_irq ( ) const [inline]

Create IRQ for GPIO pin.

Returns

IRQ number if OK, negative error code otherwise

Definition at line 108 of file vbus_gpio.

References L4vbus::Device::_bus, L4vbus::Device::_dev, and l4vbus_gpio_to_irq().

Here is the call graph for this function:

The documentation for this class was generated from the following file:

- l4/vbus/vbus_gpio

12.293 L4vbus::Icu Class Reference

V-BUS Interrupt controller API (ICU)

Inheritance diagram for L4vbus::Icu:

Collaboration diagram for L4vbus::Icu:

Public Member Functions

- int vicu (L4::Cap< L4::Icu > icu) const

  Request the L4Re::Icu capability for this V-BUS ICU.

Additional Inherited Members

12.293.1 Detailed Description

V-BUS Interrupt controller API (ICU)

Allows to access the underlying L4Re::Icu capability managing IRQs for the V-BUS.

Definition at line 191 of file vbus.

The documentation for this class was generated from the following file:

- l4/vbus/vbus

12.294 L4vbus::Pci_dev Class Reference

A PCI device.

Inheritance diagram for L4vbus::Pci_dev:

Collaboration diagram for L4vbus::Pci_dev:

Public Member Functions

- int cfg_read (l4_uint32_t reg, l4_uint32_t *value, l4_uint32_t width) const

  Read from the device's vPCI configuration space.
- int cfg_write (l4_uint32_t reg, l4_uint32_t value, l4_uint32_t width) const

  Write to the device's vPCI configuration space.
- int irq_enable (unsigned char *trigger, unsigned char *polarity) const

Enable the device's PCI interrupt.

Additional Inherited Members

12.294.1 Detailed Description

A PCI device.

Definition at line 87 of file vbus_pci.

12.294.2 Member Function Documentation

12.294.2.1 int L4vbus::Pci_dev::cfg_read (l4_uint32_t *reg*, l4_uint32_t * *value*, l4_uint32_t *width*) const [inline]

Read from the device's vPCI configuration space.

Parameters

	reg	Register in configuration space to read
out	*value*	Value that has been read
	width	Width to read in bits (e.g. 8, 16, 32)

Returns

0 on success, else failure

Definition at line 99 of file vbus_pci.

References L4vbus::Device::_dev, L4vbus::Device::bus_cap(), and l4vbus_pcidev_cfg_read().

Here is the call graph for this function:

12.294.2.2 int L4vbus::Pci_dev::cfg_write (l4_uint32_t *reg*, l4_uint32_t *value*, l4_uint32_t *width*) const [inline]

Write to the device's vPCI configuration space.

Parameters

reg	Register in configuration space to write

value	Value to write
width	Width to write in bits (e.g. 8, 16, 32)

Returns

0 on success, else failure

Definition at line 115 of file vbus_pci.

References L4vbus::Device::_dev, L4vbus::Device::bus_cap(), and l4vbus_pcidev_cfg_write().

Here is the call graph for this function:

12.294.2.3 int L4vbus::Pci_dev::irq_enable (unsigned char * *trigger*, unsigned char * *polarity*) const [inline]

Enable the device's PCI interrupt.

Parameters

out	*trigger*	False if interrupt is level-triggered
out	*polarity*	True if interrupt is of low polarity

Returns

On success: Interrupt line to be used, else failure

Definition at line 131 of file vbus_pci.

References L4vbus::Device::_dev, L4vbus::Device::bus_cap(), and l4vbus_pcidev_irq_enable().

Here is the call graph for this function:

The documentation for this class was generated from the following file:

- l4/vbus/vbus_pci

12.295 L4vbus::Pci_host_bridge Class Reference

A Pci host bridge.

Inheritance diagram for L4vbus::Pci_host_bridge:

Collaboration diagram for L4vbus::Pci_host_bridge:

Public Member Functions

- int cfg_read (l4_uint32_t bus, l4_uint32_t devfn, l4_uint32_t reg, l4_uint32_t *value, l4_uint32_t width) const

  Read from the vPCI configuration space using the PCI root bridge.
- int cfg_write (l4_uint32_t bus, l4_uint32_t devfn, l4_uint32_t reg, l4_uint32_t value, l4_uint32_t width) const

  Write to the vPCI configuration space using the PCI root bridge.

- int irq_enable (l4_uint32_t bus, l4_uint32_t devfn, int pin, unsigned char *trigger, unsigned char *polarity) const

  Enable PCI interrupt for a specific device using the PCI root bridge.

Additional Inherited Members

12.295.1 Detailed Description

A Pci host bridge.

Definition at line 20 of file vbus_pci.

12.295.2 Member Function Documentation

12.295.2.1 int L4vbus::Pci_host_bridge::cfg_read (l4_uint32_t *bus*, l4_uint32_t *devfn*, l4_uint32_t *reg*, l4_uint32_t * *value*, l4_uint32_t *width*) const [inline]

Read from the vPCI configuration space using the PCI root bridge.

Parameters

	bus	Bus number
	devfn	Device id (upper 16bit) and function (lower 16bit)
	reg	Register in configuration space to read
out	*value*	Value that has been read
	width	Width to read in bits (e.g. 8, 16, 32)

Returns

0 on success, else failure

Definition at line 34 of file vbus_pci.

References L4vbus::Device::_dev, L4vbus::Device::bus_cap(), and l4vbus_pci_cfg_read().

Here is the call graph for this function:

12.295.2.2 int L4vbus::Pci_host_bridge::cfg_write (l4_uint32_t *bus*, l4_uint32_t *devfn*, l4_uint32_t *reg*, l4_uint32_t *value*, l4_uint32_t *width*) const [inline]

Write to the vPCI configuration space using the PCI root bridge.

Parameters

bus	Bus number
devfn	Device id (upper 16bit) and function (lower 16bit)
reg	Register in configuration space to write
value	Value to write
width	Width to write in bits (e.g. 8, 16, 32)

Returns

0 on success, else failure

Definition at line 53 of file vbus_pci.

References L4vbus::Device::_dev, L4vbus::Device::bus_cap(), and l4vbus_pci_cfg_write().

Here is the call graph for this function:

12.295.2.3 int L4vbus::Pci_host_bridge::irq_enable (l4_uint32_t *bus*, l4_uint32_t *devfn*, int *pin*, unsigned char * *trigger*, unsigned char * *polarity*) const [inline]

Enable PCI interrupt for a specific device using the PCI root bridge.

Parameters

	bus	Bus number
	devfn	Device id (upper 16bit) and function (lower 16bit)
	pin	Interrupt pin (normally as reported in configuration register INTR)
out	*trigger*	False if interrupt is level-triggered
out	*polarity*	True if interrupt is of low polarity

Returns

On success: Interrupt line to be used, else failure

Definition at line 74 of file vbus_pci.

References L4vbus::Device::_dev, L4vbus::Device::bus_cap(), and l4vbus_pci_irq_enable().

Here is the call graph for this function:

The documentation for this class was generated from the following file:

- l4/vbus/vbus_pci

12.296 L4vbus::Pm< DEC > Class Template Reference

Power-management API mixin.

Inheritance diagram for L4vbus::Pm< DEC >:

Collaboration diagram for L4vbus::Pm< DEC >:

Public Member Functions

- int pm_suspend () const

  Suspend the module.
- int pm_resume () const

  Resume the module.

12.296.1 Detailed Description

template<typename DEC>class L4vbus::Pm< DEC >

Power-management API mixin.

Definition at line 32 of file vbus.

The documentation for this class was generated from the following file:

- l4/vbus/vbus

12.297 L4vbus::Vbus Class Reference

The virtual BUS.

Inheritance diagram for L4vbus::Vbus:

Collaboration diagram for L4vbus::Vbus:

Public Member Functions

- int request_resource (l4vbus_resource_t *res, int flags=0) const

  Request the given resource from the bus.
- int release_resource (l4vbus_resource_t *res) const

  Release the given resource from the bus.
- Device root () const

Get the root device of the device tree of this bus.

Additional Inherited Members

12.297.1 Detailed Description

The virtual BUS.

Definition at line 207 of file vbus.

12.297.2 Member Function Documentation

12.297.2.1 int L4vbus::Vbus::release_resource (l4vbus_resource_t * *res*) const [inline]

Release the given resource from the bus.

Parameters

res	The resource that shall be requested from the bus.

Returns

>=0 on success, <0 on error.

Definition at line 227 of file vbus.

References l4vbus_release_resource().

Here is the call graph for this function:

12.297.2.2 int L4vbus::Vbus::request_resource (l4vbus_resource_t * *res*, int *flags* = 0) const [inline]

Request the given resource from the bus.

Parameters

res	The resource that shall be requested from the bus.
flags	The flags for the request.

Returns

>=0 on success, <0 on error.

Definition at line 217 of file vbus.

References L4Re::Dataspace::flags(), and l4vbus_request_resource().

Here is the call graph for this function:

12.297.2.3 Device L4vbus::Vbus::root ( ) const [inline]

Get the root device of the device tree of this bus.

Returns

A V-BUS device representing the root of the device tree.

Definition at line 236 of file vbus.

The documentation for this class was generated from the following file:

- l4/vbus/vbus

12.298 l4vbus_device_t Struct Reference

Detailed information about a vbus device.

```
#include <vbus_types.h>
```

Collaboration diagram for l4vbus_device_t:

Data Fields

- l4_uint32_t type

  Bitfield of supported sub-interfaces, see l4vbus_iface_type_t.
- char name [L4VBUS_DEV_NAME_LEN]

Name.

- unsigned num_resources

  Number of resources for this device.
- unsigned flags

  Flags, see l4vbus_device_flags_t.

12.298.1 Detailed Description

Detailed information about a vbus device.

Definition at line 56 of file vbus_types.h.

The documentation for this struct was generated from the following file:

- l4/vbus/vbus_types.h

12.299 l4vbus_resource_t Struct Reference

Description of a single vbus resource.

```
#include <vbus_types.h>
```

Collaboration diagram for l4vbus_resource_t:

Data Fields

- l4_uint16_t type

  Resource type, see l4vbus_resource_type_t.
- l4_uint16_t flags

  Flags.
- l4vbus_paddr_t start

  Start of resource range.
- l4vbus_paddr_t end

  End of resource range (inclusive)
- l4vbus_device_handle_t provider

  Device handle of the provider of the resource.
- l4_uint32_t id

  Resource ID (4 bytes), usually a 4 letter ASCII name is used.

12.299.1 Detailed Description

Description of a single vbus resource.

Definition at line 23 of file vbus_types.h.

The documentation for this struct was generated from the following file:

- l4/vbus/vbus_types.h

12.300 L4vcpu::State Class Reference

C++ implementation of state word in the vCPU area.

Collaboration diagram for L4vcpu::State:

Public Member Functions

- State (unsigned v)

  Initialize state.
- void add (unsigned bits) throw ()

  Add flags.
- void clear (unsigned bits) throw ()

  Clear flags.
- void set (unsigned v) throw ()

  Set flags.

12.300.1 Detailed Description

C++ implementation of state word in the vCPU area.

Definition at line 31 of file vcpu.

12.300.2 Constructor & Destructor Documentation

12.300.2.1 L4vcpu::State::State (unsigned *v*) [inline],[explicit]

Initialize state.

Parameters

v	Initial state.

Definition at line 41 of file vcpu.

12.300.3 Member Function Documentation

12.300.3.1 void L4vcpu::State::add (unsigned *bits*) throw) [inline]

Add flags.

Parameters

bits	Bits to add to the word.

Definition at line 48 of file vcpu.

12.300.3.2 void L4vcpu::State::clear (unsigned *bits*) throw) [inline]

Clear flags.

Parameters

bits	Bits to clear in the word.

Definition at line 55 of file vcpu.

12.300.3.3 void L4vcpu::State::set (unsigned *v*) throw) [inline]

Set flags.

Parameters

v	Set the word to the value of v.

Definition at line 62 of file vcpu.

The documentation for this class was generated from the following file:

- l4/vcpu/vcpu

12.301 L4vcpu::Vcpu Class Reference

C++ implementation of the vCPU save state area.

Inheritance diagram for L4vcpu::Vcpu:

Collaboration diagram for L4vcpu::Vcpu:

Public Member Functions

- void irq_disable () throw ()

  Disable the vCPU for event delivery.
- unsigned irq_disable_save () throw ()

  Disable the vCPU for event delivery and return previous state.
- State ∗ state () throw ()

Get state word.

- State state () const throw ()

  Get state word.

- State * saved_state () throw ()

  Get saved_state word.

- State saved_state () const throw ()

  Get saved_state word.

- l4_uint16_t sticky_flags () const throw ()

  Get sticky flags.

- void irq_enable (l4_utcb_t *utcb, l4vcpu_event_hndl_t do_event_work_cb, l4vcpu_setup_ipc_t setup_ipc) throw ()

  Enable the vCPU for event delivery.

- void irq_restore (unsigned s, l4_utcb_t *utcb, l4vcpu_event_hndl_t do_event_work_cb, l4vcpu_setup_ipc_t setup_ipc) throw ()

  Restore a previously saved IRQ/event state.

- void wait_for_event (l4_utcb_t *utcb, l4vcpu_event_hndl_t do_event_work_cb, l4vcpu_setup_ipc_t setup_ipc) throw ()

  Wait for event.

- void task (L4::Cap< L4::Task > const task=L4::Cap< L4::Task >::Invalid) throw ()

  Set the task of the vCPU.

- int is_page_fault_entry () const

  Return whether the entry reason was a page fault.

- int is_irq_entry () const

  Return whether the entry reason was an IRQ/IPC message.

- l4_vcpu_regs_t * r () throw ()

  Return pointer to register state.

- l4_vcpu_regs_t const * r () const throw ()

  Return pointer to register state.

- l4_vcpu_ipc_regs_t * i () throw ()

  Return pointer to IPC state.

- l4_vcpu_ipc_regs_t const * i () const throw ()

  Return pointer to IPC state.

- void entry_sp (l4_umword_t sp)

  Set vCPU entry stack pointer.

- void entry_ip (l4_umword_t ip)

  Set vCPU entry instruction pointer.

- void print_state (const char *prefix="") const throw ()

  Print the state of the vCPU.

Static Public Member Functions

- static int ext_alloc (Vcpu **vcpu, l4_addr_t *ext_state, L4::Cap< L4::Task > task=L4Re::Env::env() ->task(), L4::Cap< L4Re::Rm > rm=L4Re::Env::env() ->rm()) throw ()

  Allocate state area for an extended vCPU.

- static Vcpu * cast (void *x) throw ()

  Cast a void pointer to a class pointer.

- static Vcpu * cast (l4_addr_t x) throw ()

  Cast an address to a class pointer.

12.301.1 Detailed Description

C++ implementation of the vCPU save state area.

Definition at line 72 of file vcpu.

12.301.2 Member Function Documentation

12.301.2.1 static Vcpu* L4vcpu::Vcpu::cast (void * *x*) throw) [inline], [static]

Cast a void pointer to a class pointer.

Parameters

x	Pointer.

Returns

Pointer to Vcpu class.

Definition at line 265 of file vcpu.

12.301.2.2 static Vcpu* L4vcpu::Vcpu::cast (l4_addr_t *x*) throw) [inline], [static]

Cast an address to a class pointer.

Parameters

x	Pointer.

Returns

Pointer to Vcpu class.

Definition at line 275 of file vcpu.

12.301.2.3 void L4vcpu::Vcpu::entry_ip (l4_umword_t *ip*) [inline]

Set vCPU entry instruction pointer.

Parameters

ip	Instruction pointer address to set.

Definition at line 239 of file vcpu.

References l4_vcpu_state_t::entry_ip.

12.301.2.4 void L4vcpu::Vcpu::entry_sp (l4_umword_t *sp*) [inline]

Set vCPU entry stack pointer.

Parameters

sp	Stack pointer address to set.

Note

The value is only used when entering from a user-task.

Definition at line 232 of file vcpu.

References l4_vcpu_state_t::entry_sp.

12.301.2.5 static int L4vcpu::Vcpu::ext_alloc (Vcpu ** *vcpu*, l4_addr_t * *ext_state*, L4::Cap< L4::Task > *task* = L4Re::Env::env () ->task (), L4::Cap< L4Re::Rm > *rm* = L4Re::Env::env () ->rm ()) throw) [static]

Allocate state area for an extended vCPU.

Return values

vcpu	Allocated vcpu-state area.
ext_state	Allocated extended vcpu-state area.

Parameters

task	Task to use for allocation, defaults to own task.
rm	Region manager to use for allocation defaults to standard region manager.

Returns

0 for success, error code otherwise

12.301.2.6 l4_vcpu_ipc_regs_t* L4vcpu::Vcpu::i ( ) throw) [inline]

Return pointer to IPC state.

Returns

Pointer to IPC state.

Definition at line 216 of file vcpu.

References l4_vcpu_state_t::i.

12.301.2.7 l4_vcpu_ipc_regs_t const* L4vcpu::Vcpu::i ( ) const throw) [inline]

Return pointer to IPC state.

Returns

Pointer to IPC state.

Definition at line 223 of file vcpu.

References l4_vcpu_state_t::i.

12.301.2.8 unsigned L4vcpu::Vcpu::irq_disable_save ( ) throw) [inline]

Disable the vCPU for event delivery and return previous state.

Returns

IRQ state before disabling IRQs.

Definition at line 85 of file vcpu.

References l4vcpu_irq_disable_save().

Here is the call graph for this function:

12.301.2.9 void L4vcpu::Vcpu::irq_enable (l4_utcb_t * *utcb*, l4vcpu_event_hndl_t *do_event_work_cb*, l4vcpu_setup_ipc_t *setup_ipc*) throw) [inline]

Enable the vCPU for event delivery.

Parameters

utcb	The UTCB to use.
do_event_work_cb	Call-back function that is called in case an event (such as an interrupt) is pending.
setup_ipc	Call-back function that is called before an IPC operation is called, and before event delivery is enabled.

Definition at line 142 of file vcpu.

References l4vcpu_irq_enable().

Here is the call graph for this function:

12.301.2.10 void L4vcpu::Vcpu::irq_restore (unsigned *s*, l4_utcb_t * *utcb*, l4vcpu_event_hndl_t *do_event_work_cb*, l4vcpu_setup_ipc_t *setup_ipc*) throw) [inline]

Restore a previously saved IRQ/event state.

Parameters

s	IRQ state to be restored.
utcb	The UTCB to use.
do_event_work_cb	Call-back function that is called in case an event (such as an interrupt) is pending.

setup_ipc	Call-back function that is called before an IPC operation is called, and before event delivery is enabled.

Definition at line 157 of file vcpu.

References l4vcpu_irq_restore().

Here is the call graph for this function:

12.301.2.11 int L4vcpu::Vcpu::is_irq_entry ( ) const [inline]

Return whether the entry reason was an IRQ/IPC message.

return 0 if not, !=0 otherwise.

Definition at line 195 of file vcpu.

References l4vcpu_is_irq_entry().

Here is the call graph for this function:

12.301.2.12 int L4vcpu::Vcpu::is_page_fault_entry ( ) const [inline]

Return whether the entry reason was a page fault.

return 0 if not, !=0 otherwise.

Definition at line 188 of file vcpu.

References l4vcpu_is_page_fault_entry().

Here is the call graph for this function:

12.301.2.13 l4_vcpu_regs_t* L4vcpu::Vcpu::r ( ) throw) [inline]

Return pointer to register state.

Returns

Pointer to register state.

Definition at line 202 of file vcpu.

References l4_vcpu_state_t::r.

12.301.2.14 l4_vcpu_regs_t const* L4vcpu::Vcpu::r ( ) const throw) [inline]

Return pointer to register state.

Returns

Pointer to register state.

Definition at line 209 of file vcpu.

References l4_vcpu_state_t::r.

12.301.2.15 State* L4vcpu::Vcpu::saved_state ( ) throw) [inline]

Get saved_state word.

Returns

Pointer to saved_state word in the vCPU

Definition at line 113 of file vcpu.

References l4_vcpu_state_t::saved_state.

12.301.2.16 State L4vcpu::Vcpu::saved_state ( ) const throw) [inline]

Get saved_state word.

Returns

Pointer to saved_state word in the vCPU

Definition at line 123 of file vcpu.

References l4_vcpu_state_t::saved_state.

12.301.2.17 State* L4vcpu::Vcpu::state ( ) throw) [inline]

Get state word.

Returns

Pointer to state word in the vCPU

Definition at line 95 of file vcpu.

References l4_vcpu_state_t::state.

12.301.2.18 State L4vcpu::Vcpu::state ( ) const throw) [inline]

Get state word.

Returns

Pointer to state word in the vCPU

Definition at line 106 of file vcpu.

References l4_vcpu_state_t::state.

12.301.2.19 void L4vcpu::Vcpu::task (L4::Cap< L4::Task > const *task* = L4::Cap<L4::Task>::Invalid) throw) [inline]

Set the task of the vCPU.

Parameters

task	Task to set, defaults to invalid task.

Definition at line 181 of file vcpu.

12.301.2.20 void L4vcpu::Vcpu::wait_for_event (l4_utcb_t * *utcb*, l4vcpu_event_hndl_t *do_event_work_cb*, l4vcpu_setup_ipc_t *setup_ipc*) throw) [inline]

Wait for event.

Parameters

utcb	The UTCB to use.
do_event_work_cb	Call-back function that is called in case an event (such as an interrupt) is pending.
setup_ipc	Call-back function that is called before an IPC operation is called.

Note that event delivery remains disabled after this function returns.

Definition at line 173 of file vcpu.

References l4vcpu_wait_for_event().

Here is the call graph for this function:

The documentation for this class was generated from the following file:

- l4/vcpu/vcpu

12.302 L4virtio::Driver::Virtqueue Class Reference

Driver-side implementation of a Virtqueue.

Inheritance diagram for L4virtio::Driver::Virtqueue:

Collaboration diagram for L4virtio::Driver::Virtqueue:

Public Member Functions

- void init_queue (unsigned num, l4_uint32_t align, void *base)

  Initialise this virtqueue.
- l4_uint16_t alloc_descriptor ()

  Allocate and return an unused descriptor from the descriptor table.
- void enqueue_descriptor (l4_uint16_t descno)

  Enqueue a descriptor in the available ring.
- Desc & desc (l4_uint16_t descno)

  Return a reference to a descriptor in the descriptor table.
- l4_uint16_t find_next_used ()

Return the next finished block.

- void free_descriptor (l4_uint16_t head, l4_uint16_t tail)

  Free a chained list of descriptors in the descriptor queue.

Additional Inherited Members

12.302.1 Detailed Description

Driver-side implementation of a Virtqueue.

Adds function for managing the descriptor list, enqueueing new and dequeueing finished requests.

Definition at line 333 of file virtqueue.

12.302.2 Member Function Documentation

12.302.2.1 l4_uint16_t L4virtio::Driver::Virtqueue::alloc_descriptor ( ) [inline]

Allocate and return an unused descriptor from the descriptor table.

The descriptor will be removed from the free list, the content should be considered undefined. After use, it needs to be freed using free_descriptor().

Returns

The index of the reserved descriptor or Virtqueue::Eoq if no free descriptor is available.

Note: the implementation uses (2^16 -1) as the end of queue marker. That means that the final entry in the queue can not be allocated iff the queue size is 2^16.

Definition at line 387 of file virtqueue.

References L4virtio::Virtqueue::_desc, and L4virtio::Virtqueue::Desc::next.

12.302.2.2 Desc& L4virtio::Driver::Virtqueue::desc (l4_uint16_t *descno*) [inline]

Return a reference to a descriptor in the descriptor table.

Parameters

descno	Index of the descriptor, expected to be in correct range.

Definition at line 419 of file virtqueue.

References L4virtio::Virtqueue::_desc, and L4virtio::Virtqueue::_idx_mask.

12.302.2.3 void L4virtio::Driver::Virtqueue::enqueue_descriptor (l4_uint16_t *descno*) [inline]

Enqueue a descriptor in the available ring.

Parameters

descno	Index of the head descriptor to enqueue.

Definition at line 404 of file virtqueue.

References L4virtio::Virtqueue::_avail, L4virtio::Virtqueue::_idx_mask, L4virtio::Virtqueue::Avail::idx, and L4virtio::Virtqueue::Avail::ring.

12.302.2.4 l4_uint16_t L4virtio::Driver::Virtqueue::find_next_used ( ) [inline]

Return the next finished block.

Returns

Index of the head or Virtqueue::Eoq if no used element is currently available.

Definition at line 433 of file virtqueue.

References L4virtio::Virtqueue::_current_avail, L4virtio::Virtqueue::_idx_mask, L4virtio::Virtqueue::_used, L4virtio::Virtqueue::Used_elem::id, L4virtio::Virtqueue::Used::idx, and L4virtio::Virtqueue::Used::ring.

12.302.2.5 void L4virtio::Driver::Virtqueue::free_descriptor (l4_uint16_t *head*, l4_uint16_t *tail*) [inline]

Free a chained list of descriptors in the descriptor queue.

Parameters

head	Index of the first element in the descriptor chain.
tail	Index of the last element in the descriptor chain.

Simply takes the descriptor chain and prepends it to the beginning of the free list. Assumes that the list has been correctly chained.

Definition at line 451 of file virtqueue.

References L4virtio::Virtqueue::_desc, L4virtio::Virtqueue::_idx_mask, and L4virtio::Virtqueue::Desc::next.

12.302.2.6 void L4virtio::Driver::Virtqueue::init_queue (unsigned *num*, l4_uint32_t *align*, void * *base*) [inline]

Initialise this virtqueue.

Parameters

num	The number of entries in the descriptor table, the available ring, and the used ring (must be a power of 2).
align	The alignment for the used ring (must be a power of 2).
base	The base address for the queue data structure.

This function sets up the memory and initialises the freelist.

Definition at line 358 of file virtqueue.

References L4virtio::Virtqueue::_avail, L4virtio::Virtqueue::_desc, L4virtio::Virtqueue::_used, L4virtio::Virtqueue::↩ Avail::idx, L4virtio::Virtqueue::Used::idx, L4virtio::Virtqueue::Desc::next, and L4virtio::Virtqueue::setup().

Here is the call graph for this function:

The documentation for this class was generated from the following file:

- l4/l4virtio/virtqueue

12.303 L4virtio::Ptr< T > Class Template Reference

Pointer used in virtio descriptors.

Inheritance diagram for L4virtio::Ptr< T >:

Collaboration diagram for L4virtio::Ptr< T >:

Public Types

- enum Invalid_type { Invalid }

Type for making an invalid (NULL) Ptr.

Public Member Functions

- Ptr (Invalid_type)

  Make and invalid Ptr.
- Ptr (l4_uint64_t vm_addr)

  Make a Ptr from a raw 64bit address.
- l4_uint64_t get () const
- bool is_valid () const

12.303.1 Detailed Description

template<typename T>class L4virtio::Ptr< T >

Pointer used in virtio descriptors.

As the descriptor contain guest addresses these pointers cannot be dereferenced directly.

Definition at line 46 of file virtqueue.

12.303.2 Member Enumeration Documentation

12.303.2.1 template<typename T> enum L4virtio::Ptr::Invalid_type

Type for making an invalid (NULL) Ptr.

Enumerator

Invalid Use to set a Ptr to invalid (NULL)

Definition at line 50 of file virtqueue.

12.303.3 Member Function Documentation

12.303.3.1 template<typename T> l4_uint64_t L4virtio::Ptr< T >::get ( ) const [inline]

Returns

The raw 64bit address of the pointer.

Definition at line 61 of file virtqueue.

Referenced by L4virtio::Virtqueue::Desc::dump(), L4virtio::Svr::Driver_mem_list_t< DATA >::load_desc(), and L4virtio::Svr::Driver_mem_region_t< DATA >::local().

Here is the caller graph for this function:

12.303.3.2 template<typename T> bool L4virtio::Ptr< T >::is_valid ( ) const [inline]

Returns

true if the pointer is invalid (NULL).

Definition at line 64 of file virtqueue.

The documentation for this class was generated from the following file:

- l4/l4virtio/virtqueue

12.304 L4virtio::Svr::Bad_descriptor Struct Reference

Exception used by Queue to indicate descriptor errors.

Collaboration diagram for L4virtio::Svr::Bad_descriptor:

Public Types

- enum Error {
  Bad_address, Bad_rights, Bad_flags, Bad_next,
  Bad_size }

  The error code.

Public Member Functions

- Bad_descriptor (Request_processor const ∗proc, Error e)

  Make a bad descriptor exception.

Data Fields

- Request_processor const ∗ proc

  The processor that triggered the exception.

12.304.1 Detailed Description

Exception used by Queue to indicate descriptor errors.

Definition at line 308 of file virtio.

12.304.2 Constructor & Destructor Documentation

12.304.2.1 L4virtio::Svr::Bad_descriptor::Bad_descriptor (Request_processor const * *proc,* Error *e*) [inline]

Make a bad descriptor exception.

Parameters

proc	The request processor causing the exception
e	The error code.

Definition at line 331 of file virtio.

The documentation for this struct was generated from the following file:

- l4/l4virtio/server/virtio

12.305 L4virtio::Svr::Data_buffer Struct Reference

Abstract data buffer.

Collaboration diagram for L4virtio::Svr::Data_buffer:

Public Member Functions

- template<typename T >
  Data_buffer (T *p)

  Create buffer for object p.
- template<typename T >
  void set (T *p)

  Set buffer for object p.
- l4_uint32_t copy_to (Data_buffer *dst)

  Copy contents from this buffer to the destination buffer.
- l4_uint32_t skip (l4_uint32_t bytes)

  Skip given number of bytes in this buffer.
- bool done () const

  Check if there are no more bytes left in the buffer.

Data Fields

- char * pos

Current buffer position.

- l4_uint32_t left

  Bytes left in buffer.

12.305.1 Detailed Description

Abstract data buffer.

Definition at line 224 of file virtio.

12.305.2 Constructor & Destructor Documentation

12.305.2.1 template<typename T > L4virtio::Svr::Data_buffer::Data_buffer (T * *p*) [inline],[explicit]

Create buffer for object *p*.

Template Parameters

T	type of object (implicit)

Parameters

p	pointer to object.

The buffer shall point to the start of the object *p* and the size left is sizeof(T).

Definition at line 240 of file virtio.

12.305.3 Member Function Documentation

12.305.3.1 l4_uint32_t L4virtio::Svr::Data_buffer::copy_to (Data_buffer * *dst*) [inline]

Copy contents from this buffer to the destination buffer.

Parameters

dst	Destination buffer.

Returns

the number of bytes copied.

This function copies the maximum number of bytes from this to *dst*.

Definition at line 267 of file virtio.

References left, cxx::min(), and pos.

Here is the call graph for this function:

12.305.3.2 bool L4virtio::Svr::Data_buffer::done ( ) const [inline]

Check if there are no more bytes left in the buffer.

Returns

true if there are no more bytes left in the buffer.

Definition at line 299 of file virtio.

12.305.3.3 template<typename T > void L4virtio::Svr::Data_buffer::set (T * *p*) [inline]

Set buffer for object *p*.

Template Parameters

T	type of object (implicit)

Parameters

p	pointer to object.

The buffer shall point to the start of the object *p* and the size left is sizeof(T).

Definition at line 253 of file virtio.

12.305.3.4 l4_uint32_t L4virtio::Svr::Data_buffer::skip (l4_uint32_t *bytes*) [inline]

Skip given number of bytes in this buffer.

Parameters

bytes	Number of bytes that shall be skipped.

Returns

The number of bytes skipped.

Try to skip the given number of bytes in this buffer, if there are less bytes left in the buffer that given then at most left bytes are skipped and the amount is returned.

Definition at line 287 of file virtio.

References cxx::min().

Here is the call graph for this function:

The documentation for this struct was generated from the following file:

- l4/l4virtio/server/virtio

12.306 L4virtio::Svr::Dev_config Class Reference

Abstraction for L4-Virtio device config memory.

Inherited by L4virtio::Svr::Dev_config_t< PRIV_CONFIG >.

Collaboration diagram for L4virtio::Svr::Dev_config:

Public Member Functions

- Dev_config (l4_uint32_t vendor, l4_uint32_t device, unsigned cfg_size, l4_uint32_t num_queues=0)

  Setup/Create a L4-Virtio config data space.
- Status status () const

  Get current device status (trusted).
- void set_status (Status status)

  Set device status register.
- void set_failed ()

  Set device status failed bit.
- bool change_queue_config (l4_uint32_t num_queues)

  Setup new queue configuration.
- l4virtio_config_queue_t volatile const * qconfig (unsigned index) const

  Get queue read-only config data for queue with the given index.
- void reset_hdr (bool inc_generation=false) const

  Reset the config header to the initial contents.
- bool reset_queue (unsigned index, unsigned num_max, bool inc_generation=false) const

  Reset queue config for the given queue.
- l4virtio_config_hdr_t const volatile * hdr () const

  Get a read-only pointer to the config header.

- L4::Cap< L4Re::Dataspace > ds () const

  Get data-space capability for the shared config data space.

12.306.1 Detailed Description

Abstraction for L4-Virtio device config memory.

Virtio defines a device configuration mechanism, L4-Virtio implements this mechanism based on shared memory a set_status() and a config_queue() call. This class provides an abstraction for L4-Virtio host implementations to establish such a shared memory data space and providing the necessary contents and access functions.

Definition at line 53 of file l4virtio.

12.306.2 Constructor & Destructor Documentation

12.306.2.1 L4virtio::Svr::Dev_config::Dev_config (l4_uint32_t *vendor*, l4_uint32_t *device*, unsigned *cfg_size*, l4_uint32_t *num_queues* = 0) [inline]

Setup/Create a L4-Virtio config data space.

Parameters

vendor	The vendor ID to store in config header.
device	The device ID to store in config header.
cfg_size	The size of the device-specific config data in bytes.
num_queues	The number of queues provided by the device.

This constructor allocates a data space used for L4-virtio config attaches the data space to the local address space and writes the initial contents to the config header.

Definition at line 93 of file l4virtio.

References L4Re::Util::cap_alloc, L4Re::chkcap(), L4Re::chksys(), L4Re::Env::env(), L4_PAGESIZE, L4::Ipc↩::make_cap_rw(), reset_hdr(), and L4Re::Rm::Search_addr.

Here is the call graph for this function:

12.306.3 Member Function Documentation

12.306.3.1 bool L4virtio::Svr::Dev_config::change_queue_config (l4_uint32_t *num_queues*) [inline]

Setup new queue configuration.

Parameters

num_queues	The number of queues provided by the device.

Definition at line 171 of file l4virtio.

References L4_PAGESIZE, and reset_hdr().

Here is the call graph for this function:

12.306.3.2 L4::Cap<L4Re::Dataspace> L4virtio::Svr::Dev_config::ds ( ) const [inline]

Get data-space capability for the shared config data space.

Returns

Capability for the shared config data space.

Definition at line 262 of file l4virtio.

12.306.3.3 l4virtio_config_hdr_t const volatile* L4virtio::Svr::Dev_config::hdr ( ) const [inline]

Get a read-only pointer to the config header.

Returns

Read-only pointer to the shared config header.

Definition at line 255 of file l4virtio.

Referenced by set_failed(), set_status(), and L4virtio::Svr::Device_t< DATA >::setup_queue().

Here is the caller graph for this function:

12.306.3.4 l4virtio_config_queue_t volatile const* L4virtio::Svr::Dev_config::qconfig (unsigned *index*) const [inline]

Get queue read-only config data for queue with the given *index*.

Parameters

index	The index of the queue.

Returns

Read-only pointer to the config of the queue with the given *index*, or NULL if *index* is out of range.

Definition at line 188 of file l4virtio.

References L4_UNLIKELY.

Referenced by reset_queue(), and L4virtio::Svr::Device_t< DATA >::setup_queue().

Here is the caller graph for this function:

12.306.3.5 bool L4virtio::Svr::Dev_config::reset_queue (unsigned *index*, unsigned *num_max*, bool *inc_generation* = `false`) const `[inline]`

Reset queue config for the given queue.

Parameters

index	The index of the queue to reset.
num_max	The maximum number of descriptor supported by this queue.
inc_generation	The config generation will be incremented when this is true.

Returns

true on success, or false when *index* is out of range.

Definition at line 231 of file l4virtio.

References l4virtio_config_queue_t::align, L4_UNLIKELY, l4virtio_config_queue_t::num, l4virtio_config_queue_t↩ ::num_max, l4virtio_config_queue_t::pfn, and qconfig().

Referenced by L4virtio::Svr::Device_t< DATA >::reset_queue_config().

Here is the call graph for this function:

Here is the caller graph for this function:

12.306.3.6 void L4virtio::Svr::Dev_config::set_failed ( ) [inline]

Set device status failed bit.

This function sets the internal status register and also the status register in the shared memory to *status*.

Definition at line 161 of file l4virtio.

References L4virtio::Svr::Dev_status::failed(), hdr(), L4virtio::Svr::Dev_status::raw, and l4virtio_config_hdr_t↩ ::status.

Referenced by L4virtio::Svr::Device_t< DATA >::device_error().

Here is the call graph for this function:

Here is the caller graph for this function:

12.306.3.7 void L4virtio::Svr::Dev_config::set_status (Status *status*) [inline]

Set device status register.

Parameters

status	The new value for the device status register.

This function sets the internal status register and also the status register in the shared memory to *status*.

Definition at line 149 of file l4virtio.

References hdr(), L4virtio::Svr::Dev_status::raw, l4virtio_config_hdr_t::status, and status().

Here is the call graph for this function:

12.306.3.8 Status L4virtio::Svr::Dev_config::status ( ) const [inline]

Get current device status (trusted).

Returns

Current device status register (trusted).

The status returned by this function is value stored internally and cannot be written by the guest (i.e., the value can be taken as trusted.)

Definition at line 140 of file l4virtio.

Referenced by set_status().

Here is the caller graph for this function:

The documentation for this class was generated from the following file:

- l4/l4virtio/server/l4virtio

12.307 L4virtio::Svr::Dev_features Struct Reference

Type for device feature bitmap.

Collaboration diagram for L4virtio::Svr::Dev_features:

Public Member Functions

- Dev_features (l4_uint32_t v)

  Make Features from a raw bitmap.

Data Fields

- l4_uint32_t raw

  The raw value of the features bitmap.

- typedef cxx::Bitfield < decltype(raw), 28, 28 > ring_indirect_desc_bfm_t

  Type to access the ring_indirect_desc bits (28 to 28) of raw .
- ring_indirect_desc_bfm_t::Val ring_indirect_desc () const

  Get the ring_indirect_desc bits (28 to 28) of raw .
- ring_indirect_desc_bfm_t::Ref ring_indirect_desc ()

  Get a reference to the ring_indirect_desc bits (28 to 28) of raw .

- typedef cxx::Bitfield < decltype(raw), 29, 29 > ring_event_idx_bfm_t

  Type to access the ring_event_idx bits (29 to 29) of raw .
- ring_event_idx_bfm_t::Val ring_event_idx () const

  Get the ring_event_idx bits (29 to 29) of raw .
- ring_event_idx_bfm_t::Ref ring_event_idx ()

  Get a reference to the ring_event_idx bits (29 to 29) of raw .

12.307.1 Detailed Description

Type for device feature bitmap.

Definition at line 73 of file virtio.

12.307.2 Member Typedef Documentation

12.307.2.1 typedef cxx::Bitfield<decltype(raw), 29 , 29 > L4virtio::Svr::Dev_features::ring_event_idx_bfm_t

Type to access the *ring_event_idx* bits (29 to 29) of *raw* .

Definition at line 81 of file virtio.

12.307.2.2 typedef cxx::Bitfield<decltype(raw), 28 , 28 > L4virtio::Svr::Dev_features::ring_indirect_desc_bfm↩_t

Type to access the *ring_indirect_desc* bits (28 to 28) of *raw* .

Definition at line 81 of file virtio.

12.307.3 Member Function Documentation

12.307.3.1 ring_event_idx_bfm_t::Val L4virtio::Svr::Dev_features::ring_event_idx ( ) const [inline]

Get the *ring_event_idx* bits (29 to 29) of *raw* .

Definition at line 82 of file virtio.

12.307.3.2 ring_event_idx_bfm_t::Ref L4virtio::Svr::Dev_features::ring_event_idx ( ) [inline]

Get a reference to the *ring_event_idx* bits (29 to 29) of *raw* .

Definition at line 82 of file virtio.

12.307.3.3 ring_indirect_desc_bfm_t::Val L4virtio::Svr::Dev_features::ring_indirect_desc ( ) const [inline]

Get the *ring_indirect_desc* bits (28 to 28) of *raw* .

Definition at line 81 of file virtio.

12.307.3.4 ring_indirect_desc_bfm_t::Ref L4virtio::Svr::Dev_features::ring_indirect_desc ( ) [inline]

Get a reference to the *ring_indirect_desc* bits (28 to 28) of *raw* .

Definition at line 81 of file virtio.

The documentation for this struct was generated from the following file:

- l4/l4virtio/server/virtio

12.308 L4virtio::Svr::Dev_status Struct Reference

Type of the device status register.

Collaboration diagram for L4virtio::Svr::Dev_status:

Public Member Functions

- Dev_status (l4_uint32_t v)

  Make Status from raw value.
- bool running () const

  Check if the device is in running state.

Data Fields

- unsigned char raw

  Raw value of the VIRTIO device status register.

- typedef cxx::Bitfield < decltype(raw), 0, 0 > acked_bfm_t

  Type to access the acked bits (0 to 0) of raw .
- acked_bfm_t::Val acked () const

  Get the acked bits (0 to 0) of raw .
- acked_bfm_t::Ref acked ()

  Get a reference to the acked bits (0 to 0) of raw .

- typedef cxx::Bitfield < decltype(raw), 1, 1 > driver_bfm_t

Type to access the driver bits (1 to 1) of raw .

- driver_bfm_t::Val driver () const

  Get the driver bits (1 to 1) of raw .
- driver_bfm_t::Ref driver ()

  Get a reference to the driver bits (1 to 1) of raw .

- typedef cxx::Bitfield < decltype(raw), 2, 2 > driver_ok_bfm_t

  Type to access the driver_ok bits (2 to 2) of raw .
- driver_ok_bfm_t::Val driver_ok () const

  Get the driver_ok bits (2 to 2) of raw .
- driver_ok_bfm_t::Ref driver_ok ()

  Get a reference to the driver_ok bits (2 to 2) of raw .

- typedef cxx::Bitfield < decltype(raw), 4, 4 > feature_ok_bfm_t

  Type to access the feature_ok bits (4 to 4) of raw .
- feature_ok_bfm_t::Val feature_ok () const

  Get the feature_ok bits (4 to 4) of raw .
- feature_ok_bfm_t::Ref feature_ok ()

  Get a reference to the feature_ok bits (4 to 4) of raw .

- typedef cxx::Bitfield < decltype(raw), 7, 7 > failed_bfm_t

  Type to access the failed bits (7 to 7) of raw .
- failed_bfm_t::Val failed () const

  Get the failed bits (7 to 7) of raw .
- failed_bfm_t::Ref failed ()

  Get a reference to the failed bits (7 to 7) of raw .

12.308.1 Detailed Description

Type of the device status register.

Definition at line 43 of file virtio.

12.308.2 Member Typedef Documentation

12.308.2.1 typedef cxx::Bitfield<decltype(raw), 0 , 0 > L4virtio::Svr::Dev_status::acked_bfm_t

Type to access the *acked* bits (0 to 0) of *raw* .

Definition at line 51 of file virtio.

12.308.2.2 typedef cxx::Bitfield<decltype(raw), 1 , 1 > L4virtio::Svr::Dev_status::driver_bfm_t

Type to access the *driver* bits (1 to 1) of *raw* .

Definition at line 51 of file virtio.

12.308.2.3 typedef cxx::Bitfield<decltype(raw), 2 , 2 > L4virtio::Svr::Dev_status::driver_ok_bfm_t

Type to access the *driver_ok* bits (2 to 2) of *raw* .

Definition at line 52 of file virtio.

12.308.2.4 typedef cxx::Bitfield<decltype(raw), 7 , 7 > L4virtio::Svr::Dev_status::failed_bfm_t

Type to access the *failed* bits (7 to 7) of *raw* .

Definition at line 54 of file virtio.

12.308.2.5 typedef cxx::Bitfield<decltype(raw), 4 , 4 > L4virtio::Svr::Dev_status::feature_ok_bfm_t

Type to access the *feature_ok* bits (4 to 4) of *raw* .

Definition at line 53 of file virtio.

12.308.3 Member Function Documentation

12.308.3.1 acked_bfm_t::Val L4virtio::Svr::Dev_status::acked ( ) const [inline]

Get the *acked* bits (0 to 0) of *raw* .

Definition at line 51 of file virtio.

12.308.3.2 acked_bfm_t::Ref L4virtio::Svr::Dev_status::acked ( ) [inline]

Get a reference to the *acked* bits (0 to 0) of *raw* .

Definition at line 51 of file virtio.

12.308.3.3 driver_bfm_t::Ref L4virtio::Svr::Dev_status::driver ( ) [inline]

Get a reference to the *driver* bits (1 to 1) of *raw* .

Definition at line 52 of file virtio.

12.308.3.4 driver_bfm_t::Val L4virtio::Svr::Dev_status::driver ( ) const [inline]

Get the *driver* bits (1 to 1) of *raw* .

Definition at line 52 of file virtio.

12.308.3.5 driver_ok_bfm_t::Val L4virtio::Svr::Dev_status::driver_ok ( ) const [inline]

Get the *driver_ok* bits (2 to 2) of *raw* .

Definition at line 53 of file virtio.

12.308.3.6 driver_ok_bfm_t::Ref L4virtio::Svr::Dev_status::driver_ok ( ) [inline]

Get a reference to the *driver_ok* bits (2 to 2) of *raw* .

Definition at line 53 of file virtio.

12.308.3.7 failed_bfm_t::Val L4virtio::Svr::Dev_status::failed ( ) const [inline]

Get the *failed* bits (7 to 7) of *raw* .

Definition at line 55 of file virtio.

Referenced by L4virtio::Svr::Dev_config::set_failed().

Here is the caller graph for this function:

12.308.3.8 failed_bfm_t::Ref L4virtio::Svr::Dev_status::failed ( ) [inline]

Get a reference to the *failed* bits (7 to 7) of *raw* .

Definition at line 55 of file virtio.

12.308.3.9 feature_ok_bfm_t::Ref L4virtio::Svr::Dev_status::feature_ok ( ) [inline]

Get a reference to the *feature_ok* bits (4 to 4) of *raw* .

Definition at line 54 of file virtio.

12.308.3.10 feature_ok_bfm_t::Val L4virtio::Svr::Dev_status::feature_ok ( ) const [inline]

Get the *feature_ok* bits (4 to 4) of *raw* .

Definition at line 54 of file virtio.

12.308.3.11 bool L4virtio::Svr::Dev_status::running ( ) const [inline]

Check if the device is in running state.

Returns

true if the device is in running state.

The device is in running state when acked(), driver(), and driver_ok() return true, and failed() returns false.

Definition at line 64 of file virtio.

The documentation for this struct was generated from the following file:

- l4/l4virtio/server/virtio

12.309 L4virtio::Svr::Device_t< DATA > Class Template Reference

Server-side L4-VIRTIO device stub.

Collaboration diagram for L4virtio::Svr::Device_t< DATA >:

Public Member Functions

- virtual void reset ()=0

  reset callback, called for doing a device reset
- virtual bool check_queues ()=0

  callback for checking if the queues at DRIVER_OK transition
- virtual int reconfig_queue (unsigned idx)=0

  callback for client queue-config request
- virtual void register_single_driver_irq ()=0

  callback for registering a singe guest IRQ for all queues
- virtual L4::Cap< L4::Irq > device_notify_irq () const =0

  callback to gather the device notification IRQ
- Device_t (Dev_config *dev_config)

  Make a device for the given config.
- Mem_list const * mem_info () const

  Get the memory region list used for this device.
- void reset_queue_config (unsigned idx, unsigned num_max, bool inc_generation=false)

  Trigger reset for the configuration space for queue idx.
- void init_mem_info (unsigned num)

  Initialize the memory region list to the given maximum.
- void device_error ()

  Transition device into failed state.
- bool setup_queue (Virtqueue *q, unsigned qn, unsigned num_max)

  Enable/disable the specified queue.

Protected Attributes

- Mem_list _mem_info

  Memory region list.

12.309.1 Detailed Description

template<typename DATA>class L4virtio::Svr::Device_t< DATA >

Server-side L4-VIRTIO device stub.

Definition at line 626 of file l4virtio.

12.309.2 Member Function Documentation

12.309.2.1 template<typename DATA > void L4virtio::Svr::Device_t< DATA >::device_error ( ) [inline]

Transition device into failed state.

Note

Callers should trigger a guest config IRQ after calling this function.

This function does a full reset, (calls reset()) and sets the failed bit in the device status register.

Definition at line 751 of file l4virtio.

References L4virtio::Svr::Device_t< DATA >::reset(), and L4virtio::Svr::Dev_config::set_failed().

Here is the call graph for this function:

12.309.2.2 template<typename DATA > void L4virtio::Svr::Device_t< DATA >::init_mem_info (unsigned *num*) [inline]

Initialize the memory region list to the given maximum.

Parameters

num	Maximum number of memory regions that can be managed.

Definition at line 738 of file l4virtio.

References L4virtio::Svr::Driver_mem_list_t< DATA >::init().

Here is the call graph for this function:

12.309.2.3 template<typename DATA > void L4virtio::Svr::Device_t< DATA >::reset_queue_config (unsigned *idx*, unsigned *num_max*, bool *inc_generation* = `false`) [inline]

Trigger reset for the configuration space for queue *idx*.

Parameters

idx	The queue index to reset.
num_max	Maximum number of entries in this queue.
inc_generation	The config generation will be incremented when this is true.

This function resets the driver-readable configuration space for the queue with the given index. The queue configuration is reset to all 0, name num_max to the given value.

Definition at line 728 of file l4virtio.

References L4virtio::Svr::Dev_config::reset_queue().

Here is the call graph for this function:

12.309.2.4 template<typename DATA > bool L4virtio::Svr::Device_t< DATA >::setup_queue (Virtqueue * *q*, unsigned *qn*, unsigned *num_max*) [inline]

Enable/disable the specified queue.

Parameters

q	Pointer to the ring that represents the virtqueue internally.
qn	Index of the queue.
num_max	Maximum number of supported entries in this queue.

Returns

true for success.

- This function calculates the parameters of the virtqueue from the clients configuration space values, checks the accessibility of the queue data structures and initializes *q* to ready state when all checks succeeded.

Definition at line 770 of file l4virtio.

References l4virtio_config_queue_t::align, L4virtio::Virtqueue::disable(), L4virtio::Svr::Driver_mem_list_t< DATA >::find(), l4virtio_config_hdr_t::guest_page_size, L4virtio::Svr::Dev_config::hdr(), L4_UNLIKELY, l4virtio_config_queue_t::num, l4virtio_config_queue_t::pfn, L4virtio::Svr::Dev_config::qconfig(), L4virtio::Virtqueue::setup(), and L4virtio::Virtqueue::total_size().

Here is the call graph for this function:

The documentation for this class was generated from the following file:

- l4/l4virtio/server/l4virtio

12.310 L4virtio::Svr::Driver_mem_list_t< DATA > Class Template Reference

List of driver memory regions assigned to a single L4-VIRTIO transport instance.

Collaboration diagram for L4virtio::Svr::Driver_mem_list_t< DATA >:

Public Types

- typedef L4Re::Util::Auto_cap < L4Re::Dataspace >::Cap Ds_cap

  type for storing a data-space capability internally

Public Member Functions

- Driver_mem_list_t ()

  Make an empty, zero capacity list.
- void init (unsigned max)

  Make a fresh list with capacity max.
- bool full () const
- Mem_region const ∗ add (l4_uint64_t drv_base, l4_uint32_t size, l4_addr_t offset, Ds_cap &&ds)

  Add a new region to the list.
- void remove (Mem_region const ∗r)

  Remove the given region from the list.
- Mem_region const ∗ find (l4_uint64_t base, l4_uint32_t size) const

  Find memory region containing the given driver address region.
- Mem_region ∗ find (l4_uint64_t base, l4_uint32_t size)

  Find memory region containing the given driver address region.
- void load_desc (Virtqueue::Desc const &desc, Request_processor const ∗p, Virtqueue::Desc const ∗∗table) const

  Default implementation for loading an indirect descriptor.
- void load_desc (Virtqueue::Desc const &desc, Request_processor const ∗p, Mem_region const ∗∗data) const

  Default implementation returning the Driver_mem_region.
- template<typename ARG >
  void load_desc (Virtqueue::Desc const &desc, Request_processor const ∗p, ARG ∗data) const

  Default implementation returning generic information.

12.310.1 Detailed Description

template<typename DATA>class L4virtio::Svr::Driver_mem_list_t< DATA >

List of driver memory regions assigned to a single L4-VIRTIO transport instance.

Note

The regions added to this list *must* never overlap.

Definition at line 465 of file l4virtio.

12.310.2 Member Function Documentation

12.310.2.1 template<typename DATA > Mem_region const* L4virtio::Svr::Driver_mem_list_t< DATA >::add (l4_uint64_t *drv_base*, l4_uint32_t *size*, l4_addr_t *offset*, Ds_cap && *ds*) [inline]

Add a new region to the list.

Parameters

drv_base	Driver base address of the region.
size	Size of the region in bytes.
offset	Offset within the data space attached to drv_base.
ds	Data space backing the driver memory.

Returns

A pointer to the new region.

Definition at line 505 of file l4virtio.

References L4Re::chksys(), L4virtio::Svr::Driver_mem_list_t< DATA >::full(), and L4_ENOMEM.

Here is the call graph for this function:

12.310.2.2 template<typename DATA > Mem_region const* L4virtio::Svr::Driver_mem_list_t< DATA >::find (l4_uint64_t *base*, l4_uint32_t *size*) const [inline]

Find memory region containing the given driver address region.

Parameters

base	Driver base address.
size	Size of the region.

Returns

Pointer to the region containing the given region, NULL if none is found.

Definition at line 539 of file l4virtio.

Referenced by L4virtio::Svr::Driver_mem_list_t< DATA >::load_desc(), and L4virtio::Svr::Device_t< DATA >↩ ::setup_queue().

Here is the caller graph for this function:

12.310.2.3 template<typename DATA > Mem_region* L4virtio::Svr::Driver_mem_list_t< DATA >::find (l4_uint64_t *base*, l4_uint32_t *size*) [inline]

Find memory region containing the given driver address region.

Parameters

base	Driver base address.
size	Size of the region.

Returns

Pointer to the region containing the given region, NULL if none is found.

Definition at line 547 of file l4virtio.

12.310.2.4 template<typename DATA > bool L4virtio::Svr::Driver_mem_list_t< DATA >::full ( ) const [inline]

Returns

True if the remaining capacity is 0.

Definition at line 494 of file l4virtio.

Referenced by L4virtio::Svr::Driver_mem_list_t< DATA >::add().

Here is the caller graph for this function:

12.310.2.5 template<typename DATA > void L4virtio::Svr::Driver_mem_list_t< DATA >::init (unsigned *max*) [inline]

Make a fresh list with capacity *max*.

Parameters

max	The capacity of this vector.

Definition at line 486 of file l4virtio.

References cxx::max().

Referenced by L4virtio::Svr::Device_t< DATA >::init_mem_info().

Here is the call graph for this function:

Here is the caller graph for this function:

12.310.2.6 template<typename DATA > void L4virtio::Svr::Driver_mem_list_t< DATA >::load_desc (Virtqueue::Desc const & *desc,* Request_processor const * *p,* Virtqueue::Desc const ** *table*) const [inline]

Default implementation for loading an indirect descriptor.

Parameters

	desc	The descriptor to load
	p	The request processor calling us
out	*table*	Shall be set to the loaded descriptor table

Definition at line 558 of file l4virtio.

References L4virtio::Virtqueue::Desc::addr, L4virtio::Svr::Bad_descriptor::Bad_address, L4virtio::Svr::Driver_mem_list_t< DATA >::find(), L4virtio::Ptr< T >::get(), L4_UNLIKELY, L4virtio::Virtqueue::Desc::len, and L4virtio::Svr::Driver_mem_region_t< DATA >::local().

Here is the call graph for this function:

12.310.2.7 template<typename DATA > void L4virtio::Svr::Driver_mem_list_t< DATA >::load_desc (Virtqueue::Desc const & *desc*, Request_processor const * *p*, Mem_region const ** *data*) const [inline]

Default implementation returning the Driver_mem_region.

Parameters

	desc	The descriptor to load
	p	The request processor calling us
out	*data*	Shall be set to a pointer to the Driver_mem_region that covers the descriptor.

Definition at line 575 of file l4virtio.

References L4virtio::Virtqueue::Desc::addr, L4virtio::Svr::Bad_descriptor::Bad_address, L4virtio::Svr::Driver_mem_list_t< DATA >::find(), L4virtio::Ptr< T >::get(), L4_UNLIKELY, and L4virtio::Virtqueue::Desc::len.

Here is the call graph for this function:

12.310.2.8 template<typename DATA > template<typename ARG > void L4virtio::Svr::Driver_mem_list_t< DATA >::load_desc (Virtqueue::Desc const & *desc*, Request_processor const * *p*, ARG * *data*) const [inline]

Default implementation returning generic information.

Template Parameters

ARG	Abstract argument type used with Request_processor::start() and Request_processor::next() to deliver the result of loading a descriptor. This type must provide a constructor taking three arguments: (1) pointer to a Driver_mem_region, (2) the Virtqueue::Desc descriptor, and (3) a pointer to the calling Request_processor.

Parameters

	desc	The descriptor to load
	p	The request processor calling us
out	*data*	Shall be assigned to ARG(mem, desc, p)

Definition at line 598 of file l4virtio.

References L4virtio::Virtqueue::Desc::addr, L4virtio::Svr::Bad_descriptor::Bad_address, L4virtio::Svr::Driver_mem_list_t< DATA >::find(), L4virtio::Ptr< T >::get(), L4_UNLIKELY, and L4virtio::Virtqueue::Desc::len.

Here is the call graph for this function:

12.310.2.9 template<typename DATA > void L4virtio::Svr::Driver_mem_list_t< DATA >::remove (Mem_region const * *r*) [inline]

Remove the given region from the list.

Parameters

r	The region to remove (result from add(), or find()).

Definition at line 519 of file l4virtio.

References L4Re::chksys(), and L4_ERANGE.

Here is the call graph for this function:

The documentation for this class was generated from the following file:

- l4/l4virtio/server/l4virtio

12.311 L4virtio::Svr::Driver_mem_region_t< DATA > Class Template Reference

Region of driver memory, that shall be managed locally.

Inherits DATA.

Collaboration diagram for L4virtio::Svr::Driver_mem_region_t< DATA >:

Public Member Functions

- Driver_mem_region_t ()

  Make default empty memroy region.
- Driver_mem_region_t (l4_uint64_t drv_base, l4_uint32_t size, l4_addr_t offset, Ds_cap &&ds)

  Make a local memory region for the given driver values.

- bool is_writable () const
- Flags flags () const
- bool empty () const
- l4_uint64_t drv_base () const
- void * local_base () const
- l4_uint32_t size () const
- l4_addr_t ds_offset () const
- L4::Cap< L4Re::Dataspace > ds () const
- bool contains (l4_uint64_t base, l4_uint32_t size) const

  Test if the given driver address range is within this region.

- template<typename T >
  T * local (Ptr< T > p) const

  Get the local address for driver address p.

12.311.1 Detailed Description

template<typename DATA>class L4virtio::Svr::Driver_mem_region_t< DATA >

Region of driver memory, that shall be managed locally.

Template Parameters

DATA	Class defining additional information

Definition at line 312 of file l4virtio.

12.311.2 Constructor & Destructor Documentation

12.311.2.1 template<typename DATA > L4virtio::Svr::Driver_mem_region_t< DATA >::Driver_mem_region_t (l4_uint64_t *drv_base*, l4_uint32_t *size*, l4_addr_t *offset*, Ds_cap && *ds*) [inline]

Make a local memory region for the given driver values.

Parameters

drv_base	Base address of the memory region used by the driver.
size	Size of the memory region.
offset	Offset within the data space that is mapped to *drv_base* within the driver.
ds	Data space capability backing the memory.

This constructor attaches the region of given data space to the local address space and stores the corresponding data for later reference.

Definition at line 357 of file l4virtio.

References L4Re::chksys(), L4virtio::Svr::Driver_mem_region_t< DATA >::ds(), L4Re::Dataspace::Stats::flags, L4Re::Dataspace::info(), L4_CAP_FPAGE_RO, L4_CAP_FPAGE_RW, L4_EINVAL, L4_ERANGE, L4_PAGE↩SIZE, l4_round_page(), L4::Ipc::make_cap(), L4Re::Dataspace::Map_rw, L4Re::Rm::Read_only, L4Re::Rm::↩Search_addr, L4Re::Dataspace::Stats::size, and L4virtio::Svr::Driver_mem_region_t< DATA >::size().

Here is the call graph for this function:

12.311.3 Member Function Documentation

12.311.3.1 template<typename DATA > bool L4virtio::Svr::Driver_mem_region_t< DATA >::contains (l4_uint64_t *base*, l4_uint32_t *size*) const [inline]

Test if the given driver address range is within this region.

Parameters

base	The driver base address.
size	The size of the region to lookup.

Returns

true if the given driver address region is contained in this region, false else.

Definition at line 428 of file l4virtio.

12.311.3.2 template<typename DATA > l4_uint64_t L4virtio::Svr::Driver_mem_region_t< DATA >::drv_base ( ) const [inline]

Returns

The base address used by the driver.

Definition at line 407 of file l4virtio.

12.311.3.3 template<typename DATA > L4::Cap<L4Re::Dataspace> L4virtio::Svr::Driver_mem_region_t< DATA >::ds ( ) const [inline]

Returns

The data space capability for this region.

Definition at line 419 of file l4virtio.

Referenced by L4virtio::Svr::Driver_mem_region_t< DATA >::Driver_mem_region_t().

Here is the caller graph for this function:

12.311.3.4 template<typename DATA > l4_addr_t L4virtio::Svr::Driver_mem_region_t< DATA >::ds_offset ( ) const [inline]

Returns

The offset within the data space.

Definition at line 416 of file l4virtio.

12.311.3.5 template<typename DATA > bool L4virtio::Svr::Driver_mem_region_t< DATA >::empty ( ) const [inline]

Returns

True if the region is empty (size == 0), false else.

Definition at line 403 of file l4virtio.

12.311.3.6 template<typename DATA > Flags L4virtio::Svr::Driver_mem_region_t< DATA >::flags ( ) const [inline]

Returns

The flags for this region.

Definition at line 400 of file l4virtio.

12.311.3.7 template<typename DATA > bool L4virtio::Svr::Driver_mem_region_t< DATA >::is_writable ( ) const [inline]

Returns

True if the region is writable, false else.

Definition at line 397 of file l4virtio.

12.311.3.8 template<typename DATA > template<typename T > T* L4virtio::Svr::Driver_mem_region_t< DATA >::local (Ptr< T > *p*) const [inline]

Get the local address for driver address *p*.

Parameters

p	Driver address to translate.

Precondition

p must be contained in this region.

Returns

Local address for the given driver address *p*.

Definition at line 452 of file l4virtio.

References L4virtio::Ptr< T >::get().

Referenced by L4virtio::Svr::Driver_mem_list_t< DATA >::load_desc().

Here is the call graph for this function:

Here is the caller graph for this function:

12.311.3.9 template<typename DATA > void* L4virtio::Svr::Driver_mem_region_t< DATA >::local_base ( ) const [inline]

Returns

The local base address.

Definition at line 410 of file l4virtio.

12.311.3.10 template<typename DATA > l4_uint32_t L4virtio::Svr::Driver_mem_region_t< DATA >::size ( ) const [inline]

Returns

The size of the region in bytes.

Definition at line 413 of file l4virtio.

Referenced by L4virtio::Svr::Driver_mem_region_t< DATA >::Driver_mem_region_t().

Here is the caller graph for this function:

The documentation for this class was generated from the following file:

- l4/l4virtio/server/l4virtio

12.312 L4virtio::Svr::Request_processor Class Reference

Encapsulate the state for processing a VIRTIO request.

Collaboration diagram for L4virtio::Svr::Request_processor:

Public Member Functions

- template<typename DESC_MAN , typename... ARGS>
  void start (DESC_MAN *dm, Virtqueue *ring, Virtqueue::Head_desc const &request, ARGS...args)

Start processing a new request.

- template<typename DESC_MAN , typename... ARGS>
  Virtqueue::Request const & start (DESC_MAN *dm, Virtqueue::Request const &request, ARGS...args)

  Start processing a new request.
- Virtqueue::Desc::Flags current_flags () const

  Get the flags of the currently processed descriptor.
- bool has_more () const

  Are there more chained descriptors ?
- template<typename DESC_MAN , typename... ARGS>
  bool next (DESC_MAN *dm, ARGS...args)

  Switch to the next descriptor in a descriptor chain.

12.312.1 Detailed Description

Encapsulate the state for processing a VIRTIO request.

A VIRTIO request is a possibly chained list of descriptors retrieved from the available ring of a virtqueue, using Virtqueue::next_avail().

The descriptor processing depends on helper (DESC_MAN) for interpreting the descriptors in the context of the device implementation.

DESC_MAN has to provide the functionality to safely dereference a descriptor from a descriptor list.

The following methods must be provided by DESC_MAN:

- ```
  DESC_MAN::load_desc(Virtqueue::Desc const &desc,
                      Request_processor const *proc,
                      Virtqueue::Desc const **table)
  ```

  This function is used to dereference *desc* as an indirect descriptor table, and must return a pointer to an indirect descriptor table.
- ```
  DESC_MAN::load_desc(Virtqueue::Desc const &desc,
                      Request_processor const *proc, ...)
  ```

  This function is used to dereference a descriptor as a normal data buffer, and '...' are the arguments that are passed to start() and next().

Definition at line 360 of file virtio.

12.312.2 Member Function Documentation

12.312.2.1 Virtqueue::Desc::Flags L4virtio::Svr::Request_processor::current_flags ( ) const [inline]

Get the flags of the currently processed descriptor.

Returns

The flags of the currently processed descriptor.

Definition at line 424 of file virtio.

References L4virtio::Virtqueue::Desc::flags.

12.312.2.2 bool L4virtio::Svr::Request_processor::has_more ( ) const [inline]

Are there more chained descriptors ?

Returns

true if there are more chained descriptors in the current request.

Definition at line 431 of file virtio.

References L4virtio::Virtqueue::Desc::flags, and L4virtio::Virtqueue::Desc::Flags::next().

Here is the call graph for this function:

12.312.2.3 template<typename DESC_MAN , typename... ARGS> bool L4virtio::Svr::Request_processor::next (DESC_MAN * *dm,* ARGS... *args*) [inline]

Switch to the next descriptor in a descriptor chain.

Template Parameters

DESCM_MAN	Type of descriptor manager (implicit).

Parameters

dm	Descriptor manager that is used to translate VIRTIO descriptor addresses.
args	Extra arguments passed to dm->load_desc()

Returns

true if a descriptor is available, false if not.

Definition at line 442 of file virtio.

References L4virtio::Svr::Bad_descriptor::Bad_flags, L4virtio::Svr::Bad_descriptor::Bad_next, L4virtio::Virtqueue::Desc::flags, L4virtio::Virtqueue::Desc::Flags::indirect(), L4_UNLIKELY, L4virtio::Virtqueue::Desc::Flags::next(), and L4virtio::Virtqueue::Desc::next.

Here is the call graph for this function:

12.312.2.4 template<typename DESC_MAN , typename... ARGS> void L4virtio::Svr::Request_processor::start (DESC_MAN * *dm,* Virtqueue * *ring,* Virtqueue::Head_desc const & *request,* ARGS... *args*) [inline]

Start processing a new request.

Template Parameters

DESCM_MAN	Type of descriptor manager (implicit).

Parameters

dm	Descriptor manager that is used to translate VIRTIO descriptor addresses.
ring	VIRTIO ring of the request.
request	VIRTIO request from Virtqueue::next_avail()
args	Extra arguments passed to dm->load_desc()

Precondition

The given request must be valid.

Definition at line 383 of file virtio.

References L4virtio::Svr::Bad_descriptor::Bad_size, L4virtio::Svr::Virtqueue::Head_desc::desc(), L4virtio::Svr::Virtqueue::desc(), L4virtio::Virtqueue::Desc::flags, L4virtio::Virtqueue::Desc::Flags::indirect(), L4_UNLIKELY, L4virtio::Virtqueue::Desc::len, and L4virtio::Virtqueue::num().

Referenced by start().

Here is the call graph for this function:

Here is the caller graph for this function:

12.312.2.5 template<typename DESC_MAN , typename... ARGS> Virtqueue::Request const& L4virtio::Svr::↩ Request_processor::start (DESC_MAN * *dm,* Virtqueue::Request const & *request,* ARGS... *args*) [inline]

Start processing a new request.

Template Parameters

DESCM_MAN	Type of descriptor manager (implicit).

Parameters

dm	Descriptor manager that is used to translate VIRTIO descriptor addresses.
request	VIRTIO request from Virtqueue::next_avail()
args	Extra arguments passed to dm->load_desc()

Precondition

The given request must be valid.

Definition at line 414 of file virtio.

References start().

Here is the call graph for this function:

The documentation for this class was generated from the following file:

- l4/l4virtio/server/virtio

12.313 L4virtio::Svr::Virtqueue Class Reference

Virtqueue implementation for the device.

Inheritance diagram for L4virtio::Svr::Virtqueue:

Collaboration diagram for L4virtio::Svr::Virtqueue:

Data Structures

- class Head_desc

  VIRTIO request, essentially a descriptor from the available ring.

Public Member Functions

- Request next_avail ()

  Get the next available descriptor from the available ring.
- bool desc_avail () const

  Test for available descriptors.

- void consumed (Head_desc const &r, l4_uint32_t len=0)

  Put the given descriptor into the used ring.
- void disable_notify ()

  Set the 'no notify' flag for this queue.
- void enable_notify ()

  Clear the 'no notify' flag for this queue.
- Desc const ∗ desc (unsigned idx) const

  Get a descriptor from the descriptor list.

Additional Inherited Members

12.313.1 Detailed Description

Virtqueue implementation for the device.

This class represents a single virtqueue, with a local running available index.

Definition at line 92 of file virtio.

12.313.2 Member Function Documentation

12.313.2.1 void L4virtio::Svr::Virtqueue::consumed (Head_desc const & *r*, l4_uint32_t *len* = 0) [inline]

Put the given descriptor into the used ring.

Parameters

r	request that shall be marked as finished.
len	the total number of bytes written.

Precondition

queue must be in working state.
r must be a valid request from this queue.

Definition at line 171 of file virtio.

References L4virtio::Virtqueue::_desc, L4virtio::Virtqueue::_idx_mask, L4virtio::Virtqueue::_used, L4virtio::Virtqueue::Used::idx, and L4virtio::Virtqueue::Used::ring.

12.313.2.2 Desc const∗ L4virtio::Svr::Virtqueue::desc (unsigned *idx*) const [inline]

Get a descriptor from the descriptor list.

Parameters

idx	the index of the descriptor.

Precondition

$idx < num$
queue must be in working state

Definition at line 216 of file virtio.

References L4virtio::Virtqueue::_desc.

Referenced by L4virtio::Svr::Request_processor::start().

Here is the caller graph for this function:

12.313.2.3 bool L4virtio::Svr::Virtqueue::desc_avail ( ) const [inline]

Test for available descriptors.

Returns

true if there are descriptors availabale, false if not.

Precondition

The queue must be in working state.

Definition at line 159 of file virtio.

References L4virtio::Virtqueue::_avail, L4virtio::Virtqueue::_current_avail, and L4virtio::Virtqueue::Avail::idx.

12.313.2.4 void L4virtio::Svr::Virtqueue::disable_notify ( ) [inline]

Set the 'no notify' flag for this queue.

This function may be called on a disabled queue.

Definition at line 193 of file virtio.

References L4virtio::Virtqueue::_used, L4virtio::Virtqueue::Used::flags, L4_LIKELY, L4virtio::Virtqueue::Used::↩ Flags::no_notify(), and L4virtio::Virtqueue::ready().

Here is the call graph for this function:

12.313.2.5 void L4virtio::Svr::Virtqueue::enable_notify ( ) [inline]

Clear the 'no notify' flag for this queue.

This function may be called on a disabled queue.

Definition at line 204 of file virtio.

References L4virtio::Virtqueue::_used, L4virtio::Virtqueue::Used::flags, L4_LIKELY, L4virtio::Virtqueue::Used::↩ Flags::no_notify(), and L4virtio::Virtqueue::ready().

Here is the call graph for this function:

12.313.2.6 Request L4virtio::Svr::Virtqueue::next_avail ( ) [inline]

Get the next available descriptor from the available ring.

Precondition

The queue must be in working state.

Returns

A Request for the next available descriptor, the Request is invalid if there are no descriptors in the available ring.

Note

The return value must be checked even when a previous desc_avail() returned true.

Definition at line 142 of file virtio.

References L4virtio::Virtqueue::_avail, L4virtio::Virtqueue::_current_avail, L4virtio::Virtqueue::_idx_mask, L4virtio::Virtqueue::Avail::idx, L4_LIKELY, and L4virtio::Virtqueue::Avail::ring.

The documentation for this class was generated from the following file:

- l4/l4virtio/server/virtio

12.314 L4virtio::Svr::Virtqueue::Head_desc Class Reference

VIRTIO request, essentially a descriptor from the available ring.

Inherited by L4virtio::Svr::Virtqueue::Request.

Collaboration diagram for L4virtio::Svr::Virtqueue::Head_desc:

Public Member Functions

- Head_desc ()

  Make invalid (NULL) request.
- bool valid () const
- operator Null_ptr_check const ∗ () const
- Desc const ∗ desc () const

12.314.1 Detailed Description

VIRTIO request, essentially a descriptor from the available ring.

Definition at line 98 of file virtio.

12.314.2 Member Function Documentation

12.314.2.1 Desc const∗ L4virtio::Svr::Virtqueue::Head_desc::desc ( ) const [inline]

Returns

Pointer to the head descriptor of the request.

Definition at line 119 of file virtio.

Referenced by L4virtio::Svr::Request_processor::start().

Here is the caller graph for this function:

12.314.2.2 L4virtio::Svr::Virtqueue::Head_desc::operator Null_ptr_check const * ( ) const [inline]

Returns

True if the request is valid (not NULL).

Definition at line 115 of file virtio.

12.314.2.3 bool L4virtio::Svr::Virtqueue::Head_desc::valid ( ) const [inline]

Returns

True if the request is valid (not NULL).

Definition at line 112 of file virtio.

The documentation for this class was generated from the following file:

- l4/l4virtio/server/virtio

12.315 L4virtio::Virtqueue Class Reference

Low-level Virtqueue.

Inheritance diagram for L4virtio::Virtqueue:

Collaboration diagram for L4virtio::Virtqueue:

Data Structures

- class Avail

  Type of available ring, this is read-only for the host.
- class Desc

  Descriptor in the descriptor table.
- class Used

  Used ring.
- struct Used_elem

  Type of an element of the used ring.

Public Member Functions

- void disable ()

  Completely disable the queue.
- unsigned long total_size (unsigned num, l4_uint32_t align) const

  Calculate the total size for a virtqueue of the given dimensions.

- unsigned long total_size () const

  Calculate the total size of this virtqueue.
- void setup (unsigned num, l4_uint32_t align, void *base)

  Enable this queue.
- void dump (Desc const *d) const

  Dump descriptors for this queue.
- bool ready () const

  Test if this queue is in working state.
- unsigned num () const
- bool no_notify_guest () const

  Get the no IRQ flag of this queue.
- bool no_notify_host () const

  Get the no notify flag of this queue.
- void no_notify_host (bool value)

  Set the no-notify flag for this queue.

Protected Member Functions

- Virtqueue ()

  Create a disabled virtqueue.

Protected Attributes

- Desc * _desc

  pointer to descriptor table, NULL if queue is off.
- Avail * _avail

  pointer to available ring.
- Used * _used

  pointer to used ring.
- l4_uint16_t _current_avail

  The life counter for the queue.
- l4_uint16_t _idx_mask

  mask used for indexing into the descriptor table and the rings.

12.315.1 Detailed Description

Low-level Virtqueue.

This class represents a single virtqueue, with a local running available index.

Definition at line 77 of file virtqueue.

12.315.2 Member Function Documentation

12.315.2.1 void L4virtio::Virtqueue::disable ( ) [inline]

Completely disable the queue.

setup() must be used to enable the queue again.

Definition at line 217 of file virtqueue.

Referenced by L4virtio::Svr::Device_t< DATA >::setup_queue().

Here is the caller graph for this function:

12.315.2.2 void L4virtio::Virtqueue::dump (Desc const * *d*) const [inline]

Dump descriptors for this queue.

Precondition

the queue must be in working state.

Definition at line 280 of file virtqueue.

References L4virtio::Virtqueue::Desc::dump().

Here is the call graph for this function:

12.315.2.3 bool L4virtio::Virtqueue::no_notify_guest ( ) const [inline]

Get the no IRQ flag of this queue.

Precondition

queue must be in working state.

Returns

true if the guest does not want to get IRQs (currently).

Definition at line 299 of file virtqueue.

References L4virtio::Virtqueue::Avail::flags, and L4virtio::Virtqueue::Avail::Flags::no_irq().

Here is the call graph for this function:

12.315.2.4 bool L4virtio::Virtqueue::no_notify_host ( ) const [inline]

Get the no notify flag of this queue.

Precondition

queue must be in working state.

Returns

true if the host does not want to get IRQs (currently).

Definition at line 309 of file virtqueue.

References L4virtio::Virtqueue::Used::flags, and L4virtio::Virtqueue::Used::Flags::no_notify().

Here is the call graph for this function:

12.315.2.5 void L4virtio::Virtqueue::no_notify_host (bool *value*) [inline]

Set the no-notify flag for this queue.

Precondition

Queue must be in a working state.

Definition at line 318 of file virtqueue.

References L4virtio::Virtqueue::Used::flags, and L4virtio::Virtqueue::Used::Flags::no_notify().

Here is the call graph for this function:

12.315.2.6 unsigned L4virtio::Virtqueue::num ( ) const [inline]

Returns

The number of entries in the ring.

Definition at line 291 of file virtqueue.

Referenced by setup(), L4virtio::Svr::Request_processor::start(), and total_size().

Here is the caller graph for this function:

12.315.2.7 bool L4virtio::Virtqueue::ready ( ) const [inline]

Test if this queue is in working state.

Returns

true when the queue is in working state, false else.

Definition at line 287 of file virtqueue.

References L4_LIKELY.

Referenced by L4virtio::Svr::Virtqueue::disable_notify(), and L4virtio::Svr::Virtqueue::enable_notify().

Here is the caller graph for this function:

12.315.2.8 void L4virtio::Virtqueue::setup (unsigned *num*, l4_uint32_t *align*, void * *base*) [inline]

Enable this queue.

Parameters

num	The number of entries in the descriptor table, the available ring, and the used ring (must be q power of 2).
align	The alignment for the used ring (must be a power of 2).
base	The base address for the queue data structure.

Due to the data type of the descriptors, the queue can have a maximum size of 2^16.

Definition at line 259 of file virtqueue.

References L4_EINVAL, num(), and L4virtio::Virtqueue::Avail::ring.

Referenced by L4virtio::Driver::Virtqueue::init_queue(), and L4virtio::Svr::Device_t< DATA >::setup_queue().

Here is the call graph for this function:

Here is the caller graph for this function:

12.315.2.9 unsigned long L4virtio::Virtqueue::total_size (unsigned *num*, l4_uint32_t *align*) const [inline]

Calculate the total size for a virtqueue of the given dimensions.

Parameters

num	The number of entries in the descriptor table, the available ring, and the used ring (must be q power of 2).
align	The alignment for the used ring (must be a power of 2).

Returns

the total size in bytes of the queue data structures.

Definition at line 227 of file virtqueue.

Referenced by L4virtio::Svr::Device_t< DATA >::setup_queue().

Here is the caller graph for this function:

12.315.2.10 unsigned long L4virtio::Virtqueue::total_size ( ) const [inline]

Calculate the total size of this virtqueue.

Precondition

The queue has been set up.

Definition at line 243 of file virtqueue.

References num().

Here is the call graph for this function:

The documentation for this class was generated from the following file:

- l4/l4virtio/virtqueue

12.316 L4virtio::Virtqueue::Avail Class Reference

Type of available ring, this is read-only for the host.

Collaboration diagram for L4virtio::Virtqueue::Avail:

Data Structures

- struct Flags

  Flags of the available ring.

Data Fields

- Flags flags

  flags of available ring
- l4_uint16_t idx

  available index written by guest
- l4_uint16_t ring []

  array of available descriptor indexes.

12.316.1 Detailed Description

Type of available ring, this is read-only for the host.

Definition at line 124 of file virtqueue.

The documentation for this class was generated from the following file:

- l4/l4virtio/virtqueue

12.317 L4virtio::Virtqueue::Avail::Flags Struct Reference

Flags of the available ring.

Collaboration diagram for L4virtio::Virtqueue::Avail::Flags:

Public Member Functions

- Flags (l4_uint16_t v)

  Make Flags from the raw value.

Data Fields

- l4_uint16_t raw

  raw 16bit flags value of the available ring.

- typedef cxx::Bitfield
  < decltype(raw), 0, 0 > no_irq_bfm_t

  Guest does not want to receive interrupts when requests are finished.
- no_irq_bfm_t::Val no_irq () const

  Get the no_irq bits (0 to 0) of raw .
- no_irq_bfm_t::Ref no_irq ()

  Get a reference to the no_irq bits (0 to 0) of raw .

12.317.1 Detailed Description

Flags of the available ring.

Definition at line 130 of file virtqueue.

12.317.2 Member Typedef Documentation

12.317.2.1 typedef cxx::Bitfield<decltype(raw), 0 , 0 > L4virtio::Virtqueue::Avail::Flags::no_irq_bfm_t

Guest does not want to receive interrupts when requests are finished.

Type to access the *no_irq* bits (0 to 0) of *raw* .

Definition at line 139 of file virtqueue.

12.317.3 Member Function Documentation

12.317.3.1 no_irq_bfm_t::Val L4virtio::Virtqueue::Avail::Flags::no_irq ( ) const [inline]

Get the *no_irq* bits (0 to 0) of *raw* .

Definition at line 139 of file virtqueue.

Referenced by L4virtio::Virtqueue::no_notify_guest().

Here is the caller graph for this function:

12.317.3.2 no_irq_bfm_t::Ref L4virtio::Virtqueue::Avail::Flags::no_irq ( ) [inline]

Get a reference to the *no_irq* bits (0 to 0) of *raw* .

Definition at line 139 of file virtqueue.

The documentation for this struct was generated from the following file:

- l4/l4virtio/virtqueue

12.318 L4virtio::Virtqueue::Desc Class Reference

Descriptor in the descriptor table.

Collaboration diagram for L4virtio::Virtqueue::Desc:

Data Structures

- struct Flags

  Type for descriptor flags.

Public Member Functions

- void dump (unsigned idx) const

  Dump a single descriptor.

Data Fields

- Ptr< void > addr

  Address stored in descriptor.

- l4_uint32_t len

  Length of described buffer.

- Flags flags

  Descriptor flags.

- l4_uint16_t next

  Index of the next chained descriptor.

12.318.1 Detailed Description

Descriptor in the descriptor table.

Definition at line 83 of file virtqueue.

The documentation for this class was generated from the following file:

- l4/l4virtio/virtqueue

12.319 L4virtio::Virtqueue::Desc::Flags Struct Reference

Type for descriptor flags.

Collaboration diagram for L4virtio::Virtqueue::Desc::Flags:

Public Member Functions

- Flags (l4_uint16_t v)

  Make Flags from raw 16bit value.

Data Fields

- l4_uint16_t raw

  raw flags value of a virtio descriptor.

- typedef cxx::Bitfield < decltype(raw), 0, 0 > next_bfm_t

  Part of a descriptor chain which is continued with the next field.
- next_bfm_t::Val next () const

  Get the next bits (0 to 0) of raw .
- next_bfm_t::Ref next ()

  Get a reference to the next bits (0 to 0) of raw .

- typedef cxx::Bitfield < decltype(raw), 1, 1 > write_bfm_t

  Block described by this descriptor is writeable.
- write_bfm_t::Val write () const

  Get the write bits (1 to 1) of raw .
- write_bfm_t::Ref write ()

  Get a reference to the write bits (1 to 1) of raw .

- typedef cxx::Bitfield < decltype(raw), 2, 2 > indirect_bfm_t

Indirect descriptor, block contains a list of descriptors.

- indirect_bfm_t::Val indirect () const

  Get the indirect bits (2 to 2) of raw .

- indirect_bfm_t::Ref indirect ()

  Get a reference to the indirect bits (2 to 2) of raw .

12.319.1 Detailed Description

Type for descriptor flags.

Definition at line 89 of file virtqueue.

12.319.2 Member Typedef Documentation

12.319.2.1 typedef cxx::Bitfield<decltype(raw), 2 , 2 > L4virtio::Virtqueue::Desc::Flags::indirect_bfm_t

Indirect descriptor, block contains a list of descriptors.

Type to access the *indirect* bits (2 to 2) of *raw* .

Definition at line 100 of file virtqueue.

12.319.2.2 typedef cxx::Bitfield<decltype(raw), 0 , 0 > L4virtio::Virtqueue::Desc::Flags::next_bfm_t

Part of a descriptor chain which is continued with the next field.

Type to access the *next* bits (0 to 0) of *raw* .

Definition at line 98 of file virtqueue.

12.319.2.3 typedef cxx::Bitfield<decltype(raw), 1 , 1 > L4virtio::Virtqueue::Desc::Flags::write_bfm_t

Block described by this descriptor is writeable.

Type to access the *write* bits (1 to 1) of *raw* .

Definition at line 98 of file virtqueue.

12.319.3 Member Function Documentation

12.319.3.1 indirect_bfm_t::Val L4virtio::Virtqueue::Desc::Flags::indirect ( ) const [inline]

Get the *indirect* bits (2 to 2) of *raw* .

Definition at line 102 of file virtqueue.

Referenced by L4virtio::Svr::Request_processor::next(), and L4virtio::Svr::Request_processor::start().

Here is the caller graph for this function:

12.319.3.2 indirect_bfm_t::Ref L4virtio::Virtqueue::Desc::Flags::indirect ( ) [inline]

Get a reference to the *indirect* bits (2 to 2) of *raw* .

Definition at line 102 of file virtqueue.

12.319.3.3 next_bfm_t::Ref L4virtio::Virtqueue::Desc::Flags::next ( ) [inline]

Get a reference to the *next* bits (0 to 0) of *raw* .

Definition at line 98 of file virtqueue.

12.319.3.4 next_bfm_t::Val L4virtio::Virtqueue::Desc::Flags::next ( ) const [inline]

Get the *next* bits (0 to 0) of *raw* .

Definition at line 98 of file virtqueue.

Referenced by L4virtio::Svr::Request_processor::has_more(), and L4virtio::Svr::Request_processor::next().

Here is the caller graph for this function:

12.319.3.5 write_bfm_t::Val L4virtio::Virtqueue::Desc::Flags::write ( ) const [inline]

Get the *write* bits (1 to 1) of *raw* .

Definition at line 100 of file virtqueue.

12.319.3.6 write_bfm_t::Ref L4virtio::Virtqueue::Desc::Flags::write ( ) [inline]

Get a reference to the *write* bits (1 to 1) of *raw* .

Definition at line 100 of file virtqueue.

The documentation for this struct was generated from the following file:

- l4/l4virtio/virtqueue

12.320 L4virtio::Virtqueue::Used Class Reference

Used ring.

Collaboration diagram for L4virtio::Virtqueue::Used:

Data Structures

- struct Flags

  flags for the used ring.

Data Fields

- Flags flags

  flags of the used ring.
- l4_uint16_t idx

  index of the last entry in the ring.
- Used_elem ring []

array of used descriptors.

12.320.1 Detailed Description

Used ring.

Definition at line 169 of file virtqueue.

The documentation for this class was generated from the following file:

- l4/l4virtio/virtqueue

12.321 L4virtio::Virtqueue::Used::Flags Struct Reference

flags for the used ring.

Collaboration diagram for L4virtio::Virtqueue::Used::Flags:

Public Member Functions

- Flags (l4_uint16_t v)

  make Flags from raw value

Data Fields

- l4_uint16_t raw

  raw flags value as specified by virtio.

- typedef cxx::Bitfield < decltype(raw), 0, 0 > no_notify_bfm_t

  host does not want to be notified when new requests have been queued.

- no_notify_bfm_t::Val no_notify () const

  Get the no_notify bits (0 to 0) of raw .

- no_notify_bfm_t::Ref no_notify ()

  Get a reference to the no_notify bits (0 to 0) of raw .

12.321.1 Detailed Description

flags for the used ring.

Definition at line 175 of file virtqueue.

12.321.2 Member Typedef Documentation

12.321.2.1 typedef cxx::Bitfield<decltype(raw), 0 , 0 > L4virtio::Virtqueue::Used::Flags::no_notify_bfm_t

host does not want to be notified when new requests have been queued.

Type to access the *no_notify* bits (0 to 0) of *raw* .

Definition at line 184 of file virtqueue.

12.321.3 Member Function Documentation

12.321.3.1 no_notify_bfm_t::Val L4virtio::Virtqueue::Used::Flags::no_notify ( ) const [inline]

Get the *no_notify* bits (0 to 0) of *raw* .

Definition at line 184 of file virtqueue.

Referenced by L4virtio::Svr::Virtqueue::disable_notify(), L4virtio::Svr::Virtqueue::enable_notify(), and L4virtio::Virtqueue::no_notify_host().

Here is the caller graph for this function:

12.321.3.2 no_notify_bfm_t::Ref L4virtio::Virtqueue::Used::Flags::no_notify ( ) [inline]

Get a reference to the *no_notify* bits (0 to 0) of *raw* .

Definition at line 184 of file virtqueue.

The documentation for this struct was generated from the following file:

- l4/l4virtio/virtqueue

12.322 L4virtio::Virtqueue::Used_elem Struct Reference

Type of an element of the used ring.

Collaboration diagram for L4virtio::Virtqueue::Used_elem:

Public Member Functions

- Used_elem (l4_uint16_t id, l4_uint32_t len)

  Initialize a used ring element.

Data Fields

- l4_uint32_t id

  descriptor index
- l4_uint32_t len

  length field

12.322.1 Detailed Description

Type of an element of the used ring.

Definition at line 150 of file virtqueue.

12.322.2 Constructor & Destructor Documentation

12.322.2.1 L4virtio::Virtqueue::Used_elem::Used_elem (l4_uint16_t *id*, l4_uint32_t *len*) [inline]

Initialize a used ring element.

Parameters

id	The index of the descriptor to be marked as used.
len	The total bytes written into the buffer of the descriptor chain.

Definition at line 161 of file virtqueue.

The documentation for this struct was generated from the following file:

- l4/l4virtio/virtqueue

12.323 l4virtio_block_config_t Struct Reference

Device configuration for block devices.

```
#include <virtio_block.h>
```

Collaboration diagram for l4virtio_block_config_t:

Data Fields

- l4_uint64_t capacity

  Capacity of device in 512-byte sectors.
- l4_uint32_t size_max

  Maximum size of a single segment.
- l4_uint32_t seg_max

  Maximum number of segments per request.
- l4_uint32_t blk_size

  Block size of underlying disk.

12.323.1 Detailed Description

Device configuration for block devices.

Definition at line 62 of file virtio_block.h.

12.323.2 Field Documentation

12.323.2.1 l4_uint32_t l4virtio_block_config_t::blk_size

Block size of underlying disk.

Definition at line 73 of file virtio_block.h.

The documentation for this struct was generated from the following file:

- l4/l4virtio/virtio_block.h

12.324 l4virtio_block_header_t Struct Reference

Header structure of a request for a block device.

`#include <virtio_block.h>`

Collaboration diagram for l4virtio_block_header_t:

Data Fields

- l4_uint32_t type

  Kind of request, see L4virtio_block_operations.

- l4_uint32_t ioprio

  Priority (unused)

- l4_uint64_t sector

  First sector to read/write.

12.324.1 Detailed Description

Header structure of a request for a block device.

Definition at line 51 of file virtio_block.h.

The documentation for this struct was generated from the following file:

- l4/l4virtio/virtio_block.h

12.325 l4virtio_config_hdr_t Struct Reference

L4-VIRTIO config header, provided in shared data space.

`#include <virtio.h>`

Inherited by L4virtio::Device::Config_hdr.

Collaboration diagram for l4virtio_config_hdr_t:

Data Fields

- l4_uint32_t magic

  magic value (must be 'virt').
- l4_uint32_t version

  VIRTIO version.
- l4_uint32_t device

  device ID
- l4_uint32_t vendor

  vendor ID
- l4_uint32_t num_queues

  number of virtqueues
- l4_uint32_t queues_offset

  offset of virtqueue config array
- l4_uint32_t dev_cfg_offset

  offset of device specific config
- l4_uint32_t generation

  generation counter
- l4_uint32_t status

  Device status register (read-only).
- l4_uint32_t irq_status

  IRQ status (currently unused).
- l4_uint32_t host_features [8]

  host feature bitmap
- l4_uint32_t guest_features [8]

  guest feature bitmap
- l4_uint32_t guest_page_size

  Guest page size (used for PFN to address translation).

12.325.1 Detailed Description

L4-VIRTIO config header, provided in shared data space.

Definition at line 98 of file virtio.h.

12.325.2 Field Documentation

12.325.2.1 l4_uint32_t l4virtio_config_hdr_t::status

Device status register (read-only).

The register must be written using l4virtio_set_status().

Definition at line 113 of file virtio.h.

Referenced by L4virtio::Svr::Dev_config::set_failed(), and L4virtio::Svr::Dev_config::set_status().

The documentation for this struct was generated from the following file:

- l4/l4virtio/virtio.h

12.326 l4virtio_config_queue_t Struct Reference

Queue configuration entry.

```
#include <virtio.h>
```

Collaboration diagram for l4virtio_config_queue_t:

Data Fields

- l4_uint16_t num_max

  maximum number of descriptors supported by this queue (read)
- l4_uint16_t num

  number of descriptors configured for this queue (read-write)
- l4_uint32_t align

  alignment of the used ring (read-write)
- l4_uint32_t pfn

  start page frame of the queue (read-write)

12.326.1 Detailed Description

Queue configuration entry.

An array of such entries is available at the l4virtio_config_hdr_t::queues_offset in the config data space.

Definition at line 128 of file virtio.h.

The documentation for this struct was generated from the following file:

- l4/l4virtio/virtio.h

Chapter 13

Example Documentation

13.1 examples/clntsrv/client.cc

Client/Server example using C++ infrastructure – Client implementation.

```
/*
 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
 *               Alexander Warg <warg@os.inf.tu-dresden.de>
 *     economic rights: Technische Universität Dresden (Germany)
 *
 * This file is part of TUD:OS and distributed under the terms of the
 * GNU General Public License 2.
 * Please see the COPYING-GPL-2 file for details.
 */
#include <l4/sys/err.h>
#include <l4/sys/types.h>
#include <l4/re/env>
#include <l4/re/util/cap_alloc>

#include <stdio.h>
#include "shared.h"

int
main()
{
  L4::Cap<Calc> server = L4Re::Env::env()->get_cap<Calc>("calc_server");
  if (!server.is_valid())
    {
      printf("Could not get server capability!\n");
      return 1;
    }

  l4_uint32_t val1 = 8;
  l4_uint32_t val2 = 5;

  printf("Asking for %d - %d\n", val1, val2);

  if (server->sub(val1, val2, &val1))
    {
      printf("Error talking to server\n");
      return 1;
    }
  printf("Result of substract call: %d\n", val1);
  printf("Asking for -%d\n", val1);
  if (server->neg(val1, &val1))
    {
      printf("Error talking to server\n");
      return 1;
    }
  printf("Result of negate call: %d\n", val1);

  return 0;
}
```

13.2 examples/clntsrv/clntsrv.cfg

Sample configuration file for the client/server example.

```
-- vim:set ft=lua:

-- Include L4 functionality
local L4 = require("L4");

-- Some shortcut for less typing
local ld = L4.default_loader;

-- Channel for the two programs to talk to each other.
local calc_server = ld:new_channel();

-- The server program, getting the channel in server mode.
ld:start({ caps = { calc_server = calc_server:svr() },
           log = { "server", "blue" } },
         "rom/ex_clntsrv-server");

-- The client program, getting the 'calc_server' channel to be able to talk
-- to the server. The client will be started with a green log output.
ld:start({ caps = { calc_server = calc_server },
           log = { "client", "green" } },
         "rom/ex_clntsrv-client");
```

13.3 examples/clntsrv/server.cc

Client/Server example using C++ infrastructure – Server implementation.

```
/*
 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
 *               Alexander Warg <warg@os.inf.tu-dresden.de>
 *     economic rights: Technische Universität Dresden (Germany)
 *
 * This file is part of TUD:OS and distributed under the terms of the
 * GNU General Public License 2.
 * Please see the COPYING-GPL-2 file for details.
 */
#include <stdio.h>
#include <l4/re/env>
#include <l4/re/util/cap_alloc>
#include <l4/re/util/object_registry>
#include <l4/re/util/br_manager>
#include <l4/sys/cxx/ipc_epiface>

#include "shared.h"

static L4Re::Util::Registry_server<L4Re::Util::Br_manager_hooks>
       server;

class Calculation_server : public L4::Epiface_t<Calculation_server, Calc>
{
public:
  int op_sub(Calc::Rights, l4_uint32_t a, l4_uint32_t b,
      l4_uint32_t &res)
  {
    res = a - b;
    return 0;
  }

  int op_neg(Calc::Rights, l4_uint32_t a, l4_uint32_t &res)
  {
    res = -a;
    return 0;
  }
};


int
main()
{
  static Calculation_server calc;

  // Register calculation server
  if (!server.registry()->register_obj(&calc, "calc_server").is_valid())
    {
      printf("Could not register my service, is there a 'calc_server' in the caps table?\n");
      return 1;
    }

  printf("Welcome to the calculation server!\n"
         "I can do substractions and negations.\n");

  // Wait for client requests
  server.loop();
```

```
  return 0;
}
```

13.4 examples/libs/l4re/c++/mem_alloc/ma+rm.cc

Coarse grained memory allocation, in C++.

```
/*
 * (c) 2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>
 *     economic rights: Technische Universität Dresden (Germany)
 *
 * This file is part of TUD:OS and distributed under the terms of the
 * GNU General Public License 2.
 * Please see the COPYING-GPL-2 file for details.
 */

#include <l4/re/mem_alloc>
#include <l4/re/rm>
#include <l4/re/env>
#include <l4/re/dataspace>
#include <l4/re/util/cap_alloc>
#include <l4/sys/err.h>
#include <cstdio>
#include <cstring>

static int allocate_mem(unsigned long size_in_bytes, unsigned long flags,
                        void **virt_addr)
{
  int r;
  L4::Cap<L4Re::Dataspace> d;

  /* Allocate a free capability index for our data space */
  d = L4Re::Util::cap_alloc.alloc<L4Re::Dataspace>();
  if (!d.is_valid())
    return -L4_ENOMEM;

  size_in_bytes = l4_trunc_page(size_in_bytes);

  /* Allocate memory via a dataspace */
  if ((r = L4Re::Env::env()->mem_alloc()->alloc(size_in_bytes, d, flags)))
    return r;

  /* Make the dataspace visible in our address space */
  *virt_addr = 0;
  if ((r = L4Re::Env::env()->rm()->attach(virt_addr, size_in_bytes,
                                          L4Re::Rm::Search_addr,
                                          L4::Ipc::make_cap_rw(d), 0,
                                          flags & L4Re::Mem_alloc::Super_pages
                                            ? L4_SUPERPAGESHIFT :
      L4_PAGESHIFT)))
    return r;

  /* Done, virtual address is in virt_addr */
  return 0;
}

static int free_mem(void *virt_addr)
{
  int r;
  L4::Cap<L4Re::Dataspace> ds;

  /* Detach memory from our address space */
  if ((r = L4Re::Env::env()->rm()->detach(virt_addr, &ds)))
    return r;

  /* Free memory at our memory allocator, this is optional */
  if ((r = L4Re::Env::env()->mem_alloc()->free(ds)))
    return r;

  /* Release and return capability slot to allocator */
  L4Re::Util::cap_alloc.free(ds, L4Re::Env::env()->task().cap());

  /* All went ok */
  return 0;
}

int main(void)
{
  void *virt;
```

```
  /* Allocate memory: 16k Bytes (usually) */
  if (allocate_mem(4 * L4_PAGESIZE, 0, &virt))
    return 1;

  printf("Allocated memory.\n");

  /* Do something with the memory */
  memset(virt, 0x12, 4 * L4_PAGESIZE);

  printf("Touched memory.\n");

  /* Free memory */
  if (free_mem(virt))
    return 2;

  printf("Freed and done. Bye.\n");

  return 0;
}
```

13.5 examples/libs/l4re/c++/shared_ds/ds_clnt.cc

Sharing memory between applications, client side.

```
/*
 * (c) 2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
 *          Alexander Warg <warg@os.inf.tu-dresden.de>
 *     economic rights: Technische Universität Dresden (Germany)
 *
 * This file is part of TUD:OS and distributed under the terms of the
 * GNU General Public License 2.
 * Please see the COPYING-GPL-2 file for details.
 */

#include <l4/re/util/cap_alloc> // L4::Cap
#include <l4/re/dataspace>      // L4Re::Dataspace
#include <l4/re/rm>             // L4::Rm
#include <l4/re/env>            // L4::Env
#include <l4/sys/cache.h>

#include <cstring>
#include <cstdio>
#include <unistd.h>

#include "interface.h"

int main()
{
  /*
   * Try to get server interface cap.
   */

  L4::Cap<My_interface> svr = L4Re::Env::env()->
      get_cap<My_interface>("shm");
  if (!svr.is_valid())
    {
      printf("Could not get the server capability\n");
      return 1;
    }

  /*
   * Alloc data space cap slot
   */
  L4::Cap<L4Re::Dataspace> ds = L4Re::Util::cap_alloc.
      alloc<L4Re::Dataspace>();
  if (!ds.is_valid())
    {
      printf("Could not get capability slot!\n");
      return 1;
    }

  /*
   * Alloc server notifier IRQ cap slot
   */
  L4::Cap<L4::Irq> irq = L4Re::Util::cap_alloc.
      alloc<L4::Irq>();
  if (!irq.is_valid())
    {
      printf("Could not get capability slot!\n");
      return 1;
    }
```

```
  /*
   * Request shared data-space cap.
   */
  if (svr->get_shared_buffer(ds, irq))
    {
      printf("Could not get shared memory dataspace!\n");
      return 1;
    }

  /*
   * Attach to arbitrary region
   */
  char *addr = 0;
  int err = L4Re::Env::env()->rm()->attach(&addr, ds->size(),
                                           L4Re::Rm::Search_addr,
                                           L4::Ipc::make_cap_rw(ds));
  if (err < 0)
    {
      printf("Error attaching data space: %s\n", l4sys_errtostr(err));
      return 1;
    }

  printf("Content: %s\n", addr);

  // wait a bit for the demo effect
  printf("Sleeping a bit...\n");
  sleep(1);

  /*
   * Fill in new stuff
   */
  memset(addr, 0, ds->size());
  char const * const msg = "Hello from client, too!";
  printf("Setting new content in shared memory\n");
  snprintf(addr, strlen(msg)+1, msg);
  l4_cache_clean_data((unsigned long)addr,
                      (unsigned long)addr + strlen(msg) + 1);

  // notify the server
  irq->trigger();

  /*
   * Detach region containing addr, result should be Detached_ds (other results
   * only apply if we split regions etc.).
   */
  err = L4Re::Env::env()->rm()->detach(addr, 0);
  if (err)
    printf("Failed to detach region\n");

  /* Free objects and capabilties, just for completeness. */
  L4Re::Util::cap_alloc.free(ds, L4Re::This_task);
  L4Re::Util::cap_alloc.free(irq, L4Re::This_task);

  return 0;
}
```

13.6 examples/libs/l4re/c++/shared_ds/ds_srv.cc

Sharing memory between applications, server/creator side.

```
/*
 * (c) 2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
 *          Alexander Warg <warg@os.inf.tu-dresden.de>
 *     economic rights: Technische Universität Dresden (Germany)
 *
 * This file is part of TUD:OS and distributed under the terms of the
 * GNU General Public License 2.
 * Please see the COPYING-GPL-2 file for details.
 */

#include <l4/re/env>
#include <l4/re/namespace>
#include <l4/re/util/cap_alloc>
#include <l4/re/util/object_registry>
#include <l4/re/dataspace>
#include <l4/cxx/ipc_server>

#include <l4/sys/typeinfo_svr>

#include <cstring>
#include <cstdio>
#include <unistd.h>
```

```
#include "interface.h"

class My_server_obj : public L4::Server_object_t<L4::Kobject>
{
private:
  L4::Cap<L4Re::Dataspace> _shm;
  L4::Cap<L4::Irq> _irq;

public:
  explicit My_server_obj(L4::Cap<L4Re::Dataspace> shm,
      L4::Cap<L4::Irq> irq)
  : _shm(shm), _irq(irq)
  {}

  int dispatch(l4_umword_t obj, L4::Ipc::Iostream &ios);
};


int My_server_obj::dispatch(l4_umword_t obj, L4::Ipc::Iostream &ios)
{
  // we don't care about the original object reference, however
  // we could read out the access rights from the lowest 2 bits
  (void) obj;

  l4_msgtag_t t;
  ios >> t; // extract the tag

  switch (t.label())
    {
    case L4::Meta::Protocol:
      // handle the meta protocol requests, implementing the
      // runtime dynamic type system for L4 objects.
      return L4::Util::handle_meta_request<My_interface>(ios);
    case 0:
      // since we have just one operation we have no opcode dispatch,
      // and just return the data-space and the notifier IRQ capabilities
      ios << _shm << _irq;
      return 0;
    default:
      // every other protocol is not supported.
      return -L4_EBADPROTO;
    }
}


class Shm_observer : public L4::Irq_handler_object
{
private:
  char *_shm;

public:
  explicit Shm_observer(char *shm)
  : _shm(shm)
  {}

  int dispatch(l4_umword_t obj, L4::Ipc::Iostream &ios);
};

int Shm_observer::dispatch(l4_umword_t obj, L4::Ipc::Iostream &ios)
{
  // We don't care about the original object reference, however
  // we could read out the access rights from the lowest 2 bits
  (void)obj;

  // Since we end up here in this function, we got a 'message' from the IRQ
  // that is bound to us. The 'ios' stream won't contain any valuable info.
  (void)ios;

  printf("Client sent us: %s\n", _shm);

  return 0;
}

static L4Re::Util::Registry_server<> server;

enum
{
  DS_SIZE = 4 << 12,
};

static char *get_ds(L4::Cap<L4Re::Dataspace> *_ds)
{
  *_ds = L4Re::Util::cap_alloc.alloc<L4Re::Dataspace>();
  if (!(*_ds).is_valid())
    {
      printf("Dataspace allocation failed.\n");
```

```
      return 0;
    }

  int err =  L4Re::Env::env()->mem_alloc()->alloc(DS_SIZE, *_ds, 0);
  if (err < 0)
    {
      printf("mem_alloc->alloc() failed.\n");
      L4Re::Util::cap_alloc.free(*_ds);
      return 0;
    }

  /*
   * Attach DS to local address space
   */
  char *_addr = 0;
  err =  L4Re::Env::env()->rm()->attach(&_addr, (*_ds)->size(),
                                        L4Re::Rm::Search_addr,
                                        L4::Ipc::make_cap_rw(*_ds));
  if (err < 0)
    {
      printf("Error attaching data space: %s\n", l4sys_errtostr(err));
      L4Re::Util::cap_alloc.free(*_ds);
      return 0;
    }

  /*
   * Success! Write something to DS.
   */
  printf("Attached DS\n");
  static char const * const msg = "[DS] Hello from server!";
  snprintf(_addr, strlen(msg) + 1, msg);

  return _addr;
}

int main()
{
  L4::Cap<L4Re::Dataspace> ds;
  char *addr;

  if (!(addr = get_ds(&ds)))
    return 2;

  // First the IRQ handler, because we need it in the My_server_obj object
  Shm_observer observer(addr);

  // Registering the observer as an IRQ handler, this allocates an
  // IRQ object using the factory of our server.
  L4::Cap<L4::Irq> irq = server.registry()->register_irq_obj(&observer);

  // Now the initial server object shared with the client via our parent.
  // it provides the data-space and the IRQ capabilities to a client.
  My_server_obj server_obj(ds, irq);

  // Registering the server object to the capability 'shm' in our the L4Re::Env.
  // This capability must be provided by the parent. (see the shared_ds.lua)
  server.registry()->register_obj(&server_obj, "shm");

  // Run our server loop.
  server.loop();
  return 0;
}
```

13.7 examples/libs/l4re/c++/shared_ds/shared_ds.cfg

Sharing memory between applications, configuration file.

```

-- Include L4 functionality
local L4 = require("L4");

-- Create a channel from the client to the server
local channel = L4.default_loader:new_channel();

-- Start the server, giving the channel with full server rights.
-- The server will have a yellow log output.
L4.default_loader:start(
  {
    caps = { shm = channel:svr() },
    log  = { "server", "yellow" }
  },
  "rom/ex_l4re_ds_srv"
```

```
00016 );
00017
00018 -- Start the client, giving it the channel with read only rights. The
00019 -- log output will be green.
00020 L4.default_loader:start(
00021   {
00022     caps = { shm = channel },
00023     log  = { "client", "green"  },
00024     l4re_dbg = L4.Dbg.Warn
00025   },
00026   "rom/ex_l4re_ds_clnt"
00027 );
```

13.8 examples/libs/l4re/c/ma+rm.c

Coarse grained memory allocation, in C.

```
/*
 * (c) 2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>
 *     economic rights: Technische Universität Dresden (Germany)
 *
 * This file is part of TUD:OS and distributed under the terms of the
 * GNU General Public License 2.
 * Please see the COPYING-GPL-2 file for details.
 */

#include <l4/re/c/mem_alloc.h>
#include <l4/re/c/rm.h>
#include <l4/re/c/util/cap_alloc.h>
#include <l4/sys/err.h>
#include <stdio.h>
#include <string.h>

static int allocate_mem(unsigned long size_in_bytes, unsigned long flags,
                        void **virt_addr)
{
  int r;
  l4re_ds_t ds;

  /* Allocate a free capability index for our data space */
  ds = l4re_util_cap_alloc();
  if (l4_is_invalid_cap(ds))
    return -L4_ENOMEM;

  size_in_bytes = l4_trunc_page(size_in_bytes);

  /* Allocate memory via a dataspace */
  if ((r = l4re_ma_alloc(size_in_bytes, ds, flags)))
    return r;

  /* Make the dataspace visible in our address space */
  *virt_addr = 0;
  if ((r = l4re_rm_attach(virt_addr, size_in_bytes,
                          L4RE_RM_SEARCH_ADDR, ds, 0,
                          flags & L4RE_MA_SUPER_PAGES
                             ? L4_SUPERPAGESHIFT : L4_PAGESHIFT)))
    return r;

  /* Done, virtual address is in virt_addr */
  return 0;
}

static int free_mem(void *virt_addr)
{
  int r;
  l4re_ds_t ds;

  /* Detach memory from our address space */
  if ((r = l4re_rm_detach_ds(virt_addr, &ds)))
    return r;

  /* Free memory at our memory allocator */
  if ((r = l4re_ma_free(ds)))
    return r;

  l4re_util_cap_free(ds);

  /* All went ok */
  return 0;
}
```

```
int main(void)
{
  void *virt;

  /* Allocate memory: 16k Bytes (usually) */
  if (allocate_mem(4 * L4_PAGESIZE, 0, &virt))
    return 1;

  printf("Allocated memory.\n");

  /* Do something with the memory */
  memset(virt, 0x12, 4 * L4_PAGESIZE);

  printf("Touched memory.\n");

  /* Free memory */
  if (free_mem(virt))
    return 2;

  printf("Freed and done. Bye.\n");

  return 0;
}
```

13.9 examples/libs/l4re/streammap/client.cc

Client/Server example showing how to map a page to another task – Client implementation. Note that there's also a shared memory library that supplies this functionality in more convenient way.

```
/*
 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
 *               Alexander Warg <warg@os.inf.tu-dresden.de>
 *     economic rights: Technische Universität Dresden (Germany)
 *
 * This file is part of TUD:OS and distributed under the terms of the
 * GNU General Public License 2.
 * Please see the COPYING-GPL-2 file for details.
 */
#include <l4/sys/err.h>
#include <l4/sys/types.h>
#include <l4/re/env>
#include <l4/re/util/cap_alloc>
#include <l4/cxx/ipc_stream>

#include <stdio.h>

#include "shared.h"

static int
func_smap_call(L4::Cap<void> const &server)
{
  L4::Ipc::Iostream s(l4_utcb());
  l4_addr_t addr = 0;
  int err;

  if ((err = L4Re::Env::env()->rm()->reserve_area(&addr,
      L4_PAGESIZE,
                                                  L4Re::Rm::Search_addr)))
    {
      printf("The reservation of one page within our virtual memory failed with %d\n", err);
      return 1;
    }

  s << L4::Opcode(Mapper::Do_map)
    << (l4_addr_t)addr;
  s << L4::Ipc::Rcv_fpage::mem((l4_addr_t)addr, L4_PAGESHIFT, 0);
  int r = l4_error(s.call(server.cap(), Mapper::Protocol));
  if (r)
    return r; // failure

  printf("String sent by server: %s\n", (char *)addr);

  return 0; // ok
}

int
main()
{

  L4::Cap<void> server = L4Re::Env::env()->get_cap<void>("smap");
  if (!server.is_valid())
```

```
    {
      printf("Could not get capability slot!\n");
      return 1;
    }


  printf("Asking for page from server\n");

  if (func_smap_call(server))
    {
      printf("Error talking to server\n");
      return 1;
    }
  printf("It worked!\n");

  L4Re::Util::cap_alloc.free(server, L4Re::This_task);

  return 0;
}
```

13.10 examples/libs/l4re/streammap/server.cc

Client/Server example showing how to map a page to another task – Server implementation. Note that there's also a shared memory library that supplies this functionality in more convenient way.

```
/*
 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
 *               Alexander Warg <warg@os.inf.tu-dresden.de>
 *     economic rights: Technische Universität Dresden (Germany)
 *
 * This file is part of TUD:OS and distributed under the terms of the
 * GNU General Public License 2.
 * Please see the COPYING-GPL-2 file for details.
 */
#include <stdio.h>
#include <l4/re/env>
#include <l4/re/util/cap_alloc>
#include <l4/re/util/object_registry>
#include <l4/cxx/ipc_server>

#include "shared.h"

static char page_to_map[L4_PAGESIZE] __attribute__((aligned(
      L4_PAGESIZE)));

static L4Re::Util::Registry_server<> server;

class Smap_server : public L4::Server_object_t<Mapper>
{
public:
  int dispatch(l4_umword_t obj, L4::Ipc::Iostream &ios);
};

int
Smap_server::dispatch(l4_umword_t, L4::Ipc::Iostream &ios)
{
  l4_msgtag_t t;
  ios >> t;

  // We're only talking the Map_example protocol
  if (t.label() != Mapper::Protocol)
    return -L4_EBADPROTO;

  L4::Opcode opcode;
  ios >> opcode;

  switch (opcode)
    {
    case Mapper::Do_map:
      l4_addr_t snd_base;
      ios >> snd_base;
      // put something into the page to read it out at the other side
      snprintf(page_to_map, sizeof(page_to_map), "Hello from the server!");
      printf("Sending to client\n");
      // send page
      ios << L4::Ipc::Snd_fpage::mem((l4_addr_t)page_to_map,
      L4_PAGESHIFT,
                                     L4_FPAGE_RO, snd_base);
      return L4_EOK;
    default:
      return -L4_ENOSYS;
```

```
    }
}

int
main()
{
  static Smap_server smap;

  // Register server
  if (!server.registry()->register_obj(&smap, "smap").is_valid())
    {
      printf("Could not register my service, read-only namespace?\n");
      return 1;
    }

  printf("Welcome to the memory map example server!\n");

  // Wait for client requests
  server.loop();

  return 0;
}
```

13.11 examples/libs/l4re/streammap/streammap.cfg

Sample configuration file for the client/server map example.

```
00001 -- vim:set ft=lua:
00002
00003 -- Include L4 functionality
00004 local L4 = require("L4");
00005
00006 -- Channel for the communication between the server and the client.
00007 local smap_channel = L4.default_loader:new_channel();
00008
00009 -- The server program, using the 'smap' channel in server
00010 -- mode. The log prefix will be 'server', colored yellow.
00011 L4.default_loader:start({ caps = { smap = smap_channel:svr() },
00012                           log = { "server", "yellow" }},
00013                         "rom/ex_smap-server");
00014
00015
00016 -- The client program.
00017 -- It is given the 'smap' channel to be able to talk to the server.
00018 -- The log prefix will be 'client', colored green.
00019 L4.default_loader:start({ caps = { smap = smap_channel },
00020                           log = { "client", "green" } },
00021                         "rom/ex_smap-client");
```

13.12 examples/libs/libirq/async_isr.c

libirq usage example using asychronous ISR handler functionality.

```
/*
 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>
 *     economic rights: Technische Universität Dresden (Germany)
 *
 * This file is part of TUD:OS and distributed under the terms of the
 * GNU General Public License 2.
 * Please see the COPYING-GPL-2 file for details.
 */
/*
 * This example shall show how to use the libirq.
 */

#include <l4/irq/irq.h>
#include <l4/util/util.h>

#include <stdio.h>

enum { IRQ_NO = 17 };

static void isr_handler(void *data)
{
  (void)data;
  printf("Got IRQ %d\n", IRQ_NO);
```

```
}

int main(void)
{
  const int seconds = 5;
  l4irq_t *irqdesc;

  if (!(irqdesc = l4irq_request(IRQ_NO, isr_handler, 0, 0xff, 0)))
    {
      printf("Requesting IRQ %d failed\n", IRQ_NO);
      return 1;
    }

  printf("Attached to key IRQ %d\nPress keys now, will terminate in %d seconds\n",
         IRQ_NO, seconds);

  l4_sleep(seconds * 1000);

  if (l4irq_release(irqdesc))
    {
      printf("Failed to release IRQ\n");
      return 1;
    }

  printf("Bye\n");
  return 0;
}
```

13.13 examples/libs/libirq/loop.c

libirq usage example using a self-created thread.

```
/*
 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>
 *     economic rights: Technische Universität Dresden (Germany)
 *
 * This file is part of TUD:OS and distributed under the terms of the
 * GNU General Public License 2.
 * Please see the COPYING-GPL-2 file for details.
 */

#include <l4/irq/irq.h>
#include <l4/util/util.h>
#include <stdio.h>
#include <pthread.h>

enum { IRQ_NO = 17 };

static void isr_handler(void)
{
  printf("Got IRQ %d\n", IRQ_NO);
}

static void *isr_thread(void *data)
{
  l4irq_t *irq;
  (void)data;

  if (!(irq = l4irq_attach(IRQ_NO)))
    return NULL;

  while (1)
    {
      if (l4irq_wait(irq))
        continue;
      isr_handler();
    }

  return NULL;
}


int main(void)
{
  pthread_t thread;

  if (pthread_create(&thread, NULL, isr_thread, NULL))
    return 1;

  l4_sleep_forever();
  return 0;
}
```

13.14 examples/libs/shmc/prodcons.c

Simple shared memory example.

```
/*
 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>
 *     economic rights: Technische Universität Dresden (Germany)
 *
 * This file is part of TUD:OS and distributed under the terms of the
 * GNU General Public License 2.
 * Please see the COPYING-GPL-2 file for details.
 */

/*
 * This example uses shared memory between two threads, one producer, one
 * consumer.
 */

#include <l4/shmc/shmc.h>

#include <l4/util/util.h>

#include <stdio.h>
#include <string.h>
#include <pthread-l4.h>

#include <l4/sys/thread.h>

// a small helper
#define CHK(func) if (func) { printf("failure: %d\n", __LINE__); return (void *)-1; }

static const char some_data[] = "Hi consumer!";

static void *thread_producer(void *d)
{
  (void)d;
  l4shmc_chunk_t p_one;
  l4shmc_signal_t s_one, s_done;
  l4shmc_area_t shmarea;

  // attach this thread to the shm object
  CHK(l4shmc_attach("testshm", &shmarea));

  // add a chunk
  CHK(l4shmc_add_chunk(&shmarea, "one", 1024, &p_one));

  // add a signal
  CHK(l4shmc_add_signal(&shmarea, "prod", &s_one));

  CHK(l4shmc_attach_signal_to(&shmarea, "done",
                              pthread_l4_cap(pthread_self()), 10000, &s_done));

  // connect chunk and signal
  CHK(l4shmc_connect_chunk_signal(&p_one, &s_one));

  printf("PRODUCER: ready\n");

  while (1)
    {
      while (l4shmc_chunk_try_to_take(&p_one))
        printf("Uh, should not happen!\n"); //l4_thread_yield();

      memcpy(l4shmc_chunk_ptr(&p_one), some_data, sizeof(some_data));

      CHK(l4shmc_chunk_ready_sig(&p_one, sizeof(some_data)));

      printf("PRODUCER: Sent data\n");

      CHK(l4shmc_wait_signal(&s_done));
    }

  l4_sleep_forever();
  return NULL;
}


static void *thread_consume(void *d)
{
  (void)d;
  l4shmc_area_t shmarea;
  l4shmc_chunk_t p_one;
  l4shmc_signal_t s_one, s_done;

  // attach to shared memory area
  CHK(l4shmc_attach("testshm", &shmarea));
```

```
  // get chunk 'one'
  CHK(l4shmc_get_chunk(&shmarea, "one", &p_one));

  // add a signal
  CHK(l4shmc_add_signal(&shmarea, "done", &s_done));

  // attach signal to this thread
  CHK(l4shmc_attach_signal_to(&shmarea, "prod",
                              pthread_l4_cap(pthread_self()), 10000, &s_one));

  // connect chunk and signal
  CHK(l4shmc_connect_chunk_signal(&p_one, &s_one));

  while (1)
    {
      CHK(l4shmc_wait_chunk(&p_one));

      printf("CONSUMER: Received from chunk one: %s\n",
             (char *)l4shmc_chunk_ptr(&p_one));
      memset(l4shmc_chunk_ptr(&p_one), 0, l4shmc_chunk_size(&p_one));

      CHK(l4shmc_chunk_consumed(&p_one));
      CHK(l4shmc_trigger(&s_done));
    }

  return NULL;
}


int main(void)
{
  pthread_t one, two;

  // create new shared memory area, 8K in size
  if (l4shmc_create("testshm", 8192))
    return 1;

  // create two threads, one for producer, one for consumer
  pthread_create(&one, 0, thread_producer, 0);
  pthread_create(&two, 0, thread_consume, 0);

  // now sleep, the two threads are doing the work
  l4_sleep_forever();

  return 0;
}
```

13.15 examples/sys/aliens/main.c

This example shows how system call tracing can be done.

```
/*
 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
 *               Alexander Warg <warg@os.inf.tu-dresden.de>,
 *               Björn Döbel <doebel@os.inf.tu-dresden.de>
 *     economic rights: Technische Universität Dresden (Germany)
 *
 * This file is part of TUD:OS and distributed under the terms of the
 * GNU General Public License 2.
 * Please see the COPYING-GPL-2 file for details.
 */
/*
 * Example to show syscall tracing.
 */
//#define MEASURE

#include <l4/sys/ipc.h>
#include <l4/sys/thread.h>
#include <l4/sys/factory.h>
#include <l4/sys/utcb.h>
#include <l4/sys/kdebug.h>
#include <l4/util/util.h>
#include <l4/util/rdtsc.h>
#include <l4/re/env.h>
#include <l4/re/c/util/cap_alloc.h>
#include <l4/sys/debugger.h>

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
```

```
static char alien_thread_stack[8 << 10];
static l4_cap_idx_t alien;

static void alien_thread(void)
{
  volatile l4_msgtag_t x;
  while (1) {
    x = l4_ipc_call(0x1234 << L4_CAP_SHIFT, l4_utcb(),
      l4_msgtag(0, 0, 0, 0), L4_IPC_NEVER);
#ifdef MEASURE
    l4_sleep(0);
#else
    l4_sleep(1000);
    outstring("An int3 -- you should see this\n");
    outnstring("345", 3);
#endif
  }

}

int main(void)
{
  l4_msgtag_t tag;
#ifdef MEASURE
  l4_cpu_time_t s, e;
#endif
  l4_utcb_t *u = l4_utcb();
  l4_exc_regs_t exc;
  l4_umword_t mr0, mr1;

  printf("Alien feature testing\n");

  l4_debugger_set_object_name(l4re_env()->main_thread, "alientest");

  /* Start alien thread */
  if (l4_is_invalid_cap(alien = l4re_util_cap_alloc()))
    return 1;

  l4_touch_rw(alien_thread_stack, sizeof(alien_thread_stack));

  tag = l4_factory_create_thread(l4re_env()->factory, alien);
  if (l4_error(tag))
    return 1;

  l4_debugger_set_object_name(alien, "alienth");

  l4_thread_control_start();
  l4_thread_control_pager(l4re_env()->main_thread);
  l4_thread_control_exc_handler(l4re_env()->main_thread);
  l4_thread_control_bind((l4_utcb_t *)l4re_env()->first_free_utcb,
      L4RE_THIS_TASK_CAP);
  l4_thread_control_alien(1);
  tag = l4_thread_control_commit(alien);
  if (l4_error(tag))
    return 2;

  tag = l4_thread_ex_regs(alien,
                          (l4_umword_t)alien_thread,
                          (l4_umword_t)alien_thread_stack + sizeof(alien_thread_stack),
                          0);
  if (l4_error(tag))
    return 3;

  l4_sched_param_t sp = l4_sched_param(1, 0);
  tag = l4_scheduler_run_thread(l4re_env()->scheduler, alien, &sp);
  if (l4_error(tag))
    return 4;

#ifdef MEASURE
  l4_calibrate_tsc(l4re_kip());
#endif

  /* Pager/Exception loop */
  if (l4_msgtag_has_error(tag = l4_ipc_receive(alien, u,
      L4_IPC_NEVER)))
    {
      printf("l4_ipc_receive failed");
      return 1;
    }

  memcpy(&exc, l4_utcb_exc(), sizeof(exc));
  mr0 = l4_utcb_mr()->mr[0];
  mr1 = l4_utcb_mr()->mr[1];

  for (;;)
    {
```

```
#ifdef MEASURE
      s = l4_rdtsc();
#endif

      if (l4_msgtag_is_exception(tag))
        {
#ifndef MEASURE
          printf("PC=%08lx SP=%08lx Err=%08lx Trap=%lx, %s syscall, SC-Nr: %lx\n",
                 l4_utcb_exc_pc(&exc), exc.sp, exc.err,
                 exc.trapno, (exc.err & 4) ? " after" : "before",
                 exc.err >> 3);
#endif
          tag = l4_msgtag((exc.err & 4) ? 0 : L4_PROTO_ALLOW_SYSCALL,
                          L4_UTCB_EXCEPTION_REGS_SIZE, 0, 0);
        }
      else
        printf("Umm, non-handled request (like PF): %lx %lx\n", mr0, mr1);

      memcpy(l4_utcb_exc(), &exc, sizeof(exc));

      /* Reply and wait */
      if (l4_msgtag_has_error(tag = l4_ipc_call(alien, u, tag,
      L4_IPC_NEVER)))
        {
          printf("l4_ipc_call failed\n");
          return 1;
        }
      memcpy(&exc, l4_utcb_exc(), sizeof(exc));
      mr0 = l4_utcb_mr()->mr[0];
      mr1 = l4_utcb_mr()->mr[1];
#ifdef MEASURE
      e = l4_rdtsc();
      printf("time %lld\n", l4_tsc_to_ns(e - s));
#endif
    }

  return 0;
}
```

13.16 examples/sys/ipc/ipc.cfg

Sample configuration file for the IPC example.

```
00001 # vim:se ft=lua:
00002
00003 local L4 = require("L4");
00004
00005 L4.default_loader:start({}, "rom/ex_ipc1");
```

13.17 examples/sys/ipc/ipc_example.c

This example shows how two threads can exchange data using the L4 IPC mechanism. One thread is sending an integer to the other thread which is returning the square of the integer. Both values are printed.

```
/*
 * (c) 2008-2009 Author(s)
 *     economic rights: Technische Universität Dresden (Germany)
 *
 * This file is part of TUD:OS and distributed under the terms of the
 * GNU General Public License 2.
 * Please see the COPYING-GPL-2 file for details.
 */
#include <l4/sys/ipc.h>

#include <pthread-l4.h>
#include <unistd.h>
#include <stdio.h>

static pthread_t t2;

/* Thread1 is the initiator thread, i.e. it initiates the IPC calls. In
 * other words, it takes the client role. It uses L4 IPC mechanisms to send
 * an integer value to thread2 and received a calculation result back. */
static void *thread1_fn(void *arg)
{
```

```
  l4_msgtag_t tag;
  int ipc_error;
  unsigned long value = 1;
  (void)arg;

  while (1)
    {
      printf("Sending:  %ld\n", value);

      /* Store the value which we want to have squared in the first message
       * register of our UTCB. */
      l4_utcb_mr()->mr[0] = value;

      /* To an L4 IPC call, i.e. send a message to thread2 and wait for a
       * reply from thread2. The '1' in the msgtag denotes that we want to
       * transfer one word of our message registers (i.e. MR0). No timeout. */
      tag = l4_ipc_call(pthread_l4_cap(t2), l4_utcb(),
                        l4_msgtag(0, 1, 0, 0), L4_IPC_NEVER);
      /* Check for IPC error, if yes, print out the IPC error code, if not,
       * print the received result. */
      ipc_error = l4_ipc_error(tag, l4_utcb());
      if (ipc_error)
        fprintf(stderr, "thread1: IPC error: %x\n", ipc_error);
      else
        printf("Received: %ld\n", l4_utcb_mr()->mr[0]);

      /* Wait some time and increment our value. */
      sleep(1);
      value++;
    }
  return NULL;
}

/* Thread2 is in the server role, i.e. it waits for requests from others and
 * sends back the calculation results. */
static void *thread2_fn(void *arg)
{
  l4_msgtag_t tag;
  l4_umword_t label;
  int ipc_error;
  (void)arg;

  /* Wait for requests from any thread. No timeout, i.e. wait forever. */
  tag = l4_ipc_wait(l4_utcb(), &label, L4_IPC_NEVER);
  while (1)
    {
      /* Check if we had any IPC failure, if yes, print the error code
       * and just wait again. */
      ipc_error = l4_ipc_error(tag, l4_utcb());
      if (ipc_error)
        {
          fprintf(stderr, "thread2: IPC error: %x\n", ipc_error);
          tag = l4_ipc_wait(l4_utcb(), &label, L4_IPC_NEVER);
          continue;
        }

      /* So, the IPC was ok, now take the value out of message register 0
       * of the UTCB and store the square of it back to it. */
      l4_utcb_mr()->mr[0] = l4_utcb_mr()->mr[0] *
      l4_utcb_mr()->mr[0];

      /* Send the reply and wait again for new messages.
       * The '1' in the msgtag indicated that we want to transfer 1 word in
       * the message registers (i.e. MR0) */
      tag = l4_ipc_reply_and_wait(l4_utcb(),
      l4_msgtag(0, 1, 0, 0),
                                  &label, L4_IPC_NEVER);
    }
  return NULL;
}

int main(void)
{
  // We will have two threads, one is already running the main function, the
  // other (thread2) will be created using pthread_create.

  if (pthread_create(&t2, NULL, thread2_fn, NULL))
    {
      fprintf(stderr, "Thread creation failed\n");
      return 1;
    }

  // Just run thread1 in the main thread
  thread1_fn(NULL);
  return 0;
}
```

13.18 examples/sys/isr/main.c

Example of an interrupt service routine.

```
/*
 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
 *               Alexander Warg <warg@os.inf.tu-dresden.de>,
 *               Björn Döbel <doebel@os.inf.tu-dresden.de>
 *     economic rights: Technische Universität Dresden (Germany)
 *
 * This file is part of TUD:OS and distributed under the terms of the
 * GNU General Public License 2.
 * Please see the COPYING-GPL-2 file for details.
 */
/*
 * This example shall show how to connect to an interrupt, receive interrupt
 * events and detach again. As the interrupt source we'll use the virtual
 * key interrupt. The interrupt number of the virtual key interrupt can be
 * found in the kernel info page.
 */

#include <l4/re/c/util/cap_alloc.h>
#include <l4/re/c/namespace.h>
#include <l4/sys/utcb.h>
#include <l4/sys/irq.h>
#include <l4/sys/factory.h>
#include <l4/sys/icu.h>

#include <stdio.h>

int main(void)
{
  int irqno = 1;
  l4_cap_idx_t irqcap, icucap;
  l4_msgtag_t tag;
  int err;
  icucap = l4re_env_get_cap("icu");

  /* Get a free capability slot for the ICU capability */
  if (l4_is_invalid_cap(icucap))
    {
      printf("Did not find an ICU\n");
      return 1;
    }

  /* Get another free capaiblity slot for the corresponding IRQ object*/
  if (l4_is_invalid_cap(irqcap = l4re_util_cap_alloc()))
    return 1;
  /* Create IRQ object */
  if (l4_error(tag = l4_factory_create_irq(l4re_global_env->
      factory, irqcap)))
    {
      printf("Could not create IRQ object: %lx\n", l4_error(tag));
      return 1;
    }

  /*
   * Bind the recently allocated IRQ object to the IRQ number irqno
   * as provided by the ICU.
   */
  if (l4_error(l4_icu_bind(icucap, irqno, irqcap)))
    {
      printf("Binding IRQ%d to the ICU failed\n", irqno);
      return 1;
    }

  /* Attach ourselves to the IRQ */
  tag = l4_irq_attach(irqcap, 0xDEAD, l4re_env()->main_thread);
  if ((err = l4_error(tag)))
    {
      printf("Error attaching to IRQ %d: %d\n", irqno, err);
      return 1;
    }

  printf("Attached to key IRQ %d\nPress keys now, Shift-Q to exit\n", irqno);

  /* IRQ receive loop */
  while (1)
    {
      unsigned long label = 0;
      /* Wait for the interrupt to happen */
      tag = l4_irq_receive(irqcap, L4_IPC_NEVER);
      if ((err = l4_ipc_error(tag, l4_utcb())))
        printf("Error on IRQ receive: %d\n", err);
      else
```

```
        {
          /* Process the interrupt -- may do a 'break' */
          printf("Got IRQ with label 0x%lX\n", label);
        }
    }

  /* We're done, detach from the interrupt. */
  tag = l4_irq_detach(irqcap);
  if ((err = l4_error(tag)))
    printf("Error detach from IRQ: %d\n", err);

  return 0;
}
```

13.19 examples/sys/migrate/thread_migrate.cc

Thread migration example.

```
/*
 * (c) 2008-2009 Author(s)
 *     economic rights: Technische Universität Dresden (Germany)
 *
 * This file is part of TUD:OS and distributed under the terms of the
 * GNU General Public License 2.
 * Please see the COPYING-GPL-2 file for details.
 */
#include <l4/sys/scheduler>
#include <l4/re/env>
#include <l4/re/util/cap_alloc>

#include <pthread-l4.h>
#include <unistd.h>
#include <stdio.h>
#include <string.h>

enum { NR_THREADS = 12 };
static L4::Cap<L4::Thread> threads[NR_THREADS];
static l4_umword_t         cpu_map, cpu_nrs;

/* Function for the threads. The content is not really relevant, so lets
 * just sleep around a bit. */
static void *thread_fn(void *)
{
  while (1)
    sleep(1);

  return 0;
}

/* Check how many CPUs we have available.
 */
static int check_cpus(void)
{
  l4_sched_cpu_set_t cs = l4_sched_cpu_set(0, 0);

  if (l4_error(L4Re::Env::env()->scheduler()->info(&cpu_nrs, &cs)) < 0)
    return 1;

  cpu_map = cs.map;

  printf("%ld maximal supported CPUs.\n", cpu_nrs);
  if (cpu_nrs >= L4_MWORD_BITS)
    {
      printf("Will only handle %ld CPUs.\n", cpu_nrs);
      cpu_nrs = L4_MWORD_BITS;
    }
  else if (cpu_nrs == 1)
    printf("Only found 1 CPU.\n");

  return cpu_nrs < 2;
}

/* Create a couple of threads and store their capabilities in an array */
static int create_threads(void)
{
  unsigned i;

  for (i = 0; i < NR_THREADS; ++i)
    {
      pthread_t t;
```

```
      if (pthread_create(&t, NULL, thread_fn, NULL))
        return 1;

      threads[i] = L4::Cap<L4::Thread>(pthread_l4_cap(t));
    }
  printf("Created %d threads.\n", NR_THREADS);
  return 0;
}

/* Helper function to get the next CPU */
static unsigned get_next_cpu(unsigned c)
{
  unsigned x = c;
  for (;;)
    {
      x = (x + 1) % cpu_nrs;
      if (L4Re::Env::env()->scheduler()->is_online(x))
        return x;
      if (x == c)
        return c;
    }
}

/* Function that shuffles the threads on the available CPUs */
static void shuffle(void)
{
  unsigned start = 0;
  while (1)
    {
      unsigned t;
      unsigned c = start;
      for (t = 0; t < NR_THREADS; ++t)
        {
          l4_sched_param_t sp = l4_sched_param(20);
          c = get_next_cpu(c);
          sp.affinity = l4_sched_cpu_set(c, 0);
          if (l4_error(L4Re::Env::env()->scheduler()->run_thread(threads[t], sp)))
            printf("Error migrating thread%02d to CPU%02d\n", t, c);
          printf("Migrated Thread%02d -> CPU%02d\n", t, c);
        }

      start++;
      if (start == cpu_nrs)
        start = 0;
      sleep(1);
    }
}

int main(void)
{
  if (check_cpus())
    return 1;

  if (create_threads())
    return 1;

  shuffle();

  return 0;
}
```

13.20 examples/sys/migrate/thread_migrate.cfg

Sample configuration file for the thread migration example.

```
-- vim:set ft=lua:

local L4 = require("L4");

-- The log prefix will be 'migrate', colored green.
L4.default_loader:start({ log = { "migrate", "green" } },
                        "rom/ex_thread_migrate");
```

13.21 examples/sys/singlestep/main.c

This example shows how a thread can be single stepped on the x86 architecture.

```
/*
 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
 *               Alexander Warg <warg@os.inf.tu-dresden.de>,
 *               Björn Döbel <doebel@os.inf.tu-dresden.de>
 *     economic rights: Technische Universität Dresden (Germany)
 *
 * This file is part of TUD:OS and distributed under the terms of the
 * GNU General Public License 2.
 * Please see the COPYING-GPL-2 file for details.
 */
/*
 * Single stepping example for the x86-32 architecture.
 */
#include <l4/sys/ipc.h>
#include <l4/sys/factory.h>
#include <l4/sys/thread.h>
#include <l4/sys/utcb.h>
#include <l4/sys/kdebug.h>

#include <l4/util/util.h>
#include <l4/re/env.h>
#include <l4/re/c/util/cap_alloc.h>

#include <stdlib.h>
#include <stdio.h>
#include <string.h>

static char thread_stack[8 << 10];

static void thread_func(void)
{
  while (1)
    {
      unsigned long d = 0;

      /* Enable single stepping  */
      asm volatile("pushf; pop %0; or $256,%0; push %0; popf\n"
                   : "=r" (d) : "r" (d));

      /* Some instructions */
      asm volatile("nop");
      asm volatile("nop");
      asm volatile("nop");
      asm volatile("mov $0x12345000, %%edx" : : : "edx"); // a non-existent cap
      asm volatile("int $0x30\n");
      asm volatile("nop");
      asm volatile("nop");
      asm volatile("nop");

      /* Disabled single stepping */
      asm volatile("pushf; pop %0; and $~256,%0; push %0; popf\n"
                   : "=r" (d) : "r" (d));

      /* You won't see those */
      asm volatile("nop");
      asm volatile("nop");
      asm volatile("nop");
    }
}

int main(void)
{
  l4_msgtag_t tag;
  int ipc_stat = 0;
  l4_cap_idx_t th = l4re_util_cap_alloc();
  l4_exc_regs_t exc;
  l4_umword_t mr0, mr1;
  l4_utcb_t *u = l4_utcb();

  printf("Singlestep testing\n");

  if (l4_is_invalid_cap(th))
    return 1;

  l4_touch_rw(thread_stack, sizeof(thread_stack));
  l4_touch_ro(thread_func, 1);

  tag = l4_factory_create_thread(l4re_env()->factory, th);
  if (l4_error(tag))
    return 1;

  l4_thread_control_start();
  l4_thread_control_pager(l4re_env()->main_thread);
  l4_thread_control_exc_handler(l4re_env()->main_thread);
  l4_thread_control_bind((l4_utcb_t *)l4re_env()->first_free_utcb,
                         L4RE_THIS_TASK_CAP);
  l4_thread_control_alien(1);
```

```
  tag = l4_thread_control_commit(th);
  if (l4_error(tag))
    return 2;

  tag = l4_thread_ex_regs(th, (l4_umword_t)thread_func,
                          (l4_umword_t)thread_stack + sizeof(thread_stack),
                          0);
  if (l4_error(tag))
    return 3;

  l4_sched_param_t sp = l4_sched_param(1, 0);
  tag = l4_scheduler_run_thread(l4re_env()->scheduler, th, &sp);
  if (l4_error(tag))
    return 4;

  /* Pager/Exception loop */
  if (l4_msgtag_has_error(tag = l4_ipc_receive(th, u,
      L4_IPC_NEVER)))
    {
      printf("l4_ipc_receive failed");
      return 5;
    }
  memcpy(&exc, l4_utcb_exc(), sizeof(exc));
  mr0 = l4_utcb_mr()->mr[0];
  mr1 = l4_utcb_mr()->mr[1];

  for (;;)
    {
      if (l4_msgtag_is_exception(tag))
        {
          printf("PC = %08lx Trap = %08lx Err = %08lx, SP = %08lx SC-Nr: %lx\n",
                 l4_utcb_exc_pc(&exc), exc.trapno, exc.err,
                 exc.sp, exc.err >> 3);
          if (exc.err >> 3)
            {
              if (!(exc.err & 4))
                {
                  tag = l4_msgtag(L4_PROTO_ALLOW_SYSCALL,
                                  L4_UTCB_EXCEPTION_REGS_SIZE, 0, 0);
                  if (ipc_stat)
                    enter_kdebug("Should not be 1");
                }
              else
                {
                  tag = l4_msgtag(L4_PROTO_NONE,
                                  L4_UTCB_EXCEPTION_REGS_SIZE, 0, 0);
                  if (!ipc_stat)
                    enter_kdebug("Should not be 0");
                }
              ipc_stat = !ipc_stat;
            }
          l4_sleep(100);
        }
      else
        printf("Umm, non-handled request: %ld, %08lx %08lx\n",
               l4_msgtag_label(tag), mr0, mr1);

      memcpy(l4_utcb_exc(), &exc, sizeof(exc));

      /* Reply and wait */
      if (l4_msgtag_has_error(tag = l4_ipc_call(th, u, tag,
      L4_IPC_NEVER)))
        {
          printf("l4_ipc_call failed\n");
          return 5;
        }
      memcpy(&exc, l4_utcb_exc(), sizeof(exc));
      mr0 = l4_utcb_mr()->mr[0];
      mr1 = l4_utcb_mr()->mr[1];
    }

  return 0;
}
```

13.22 examples/sys/start-with-exc/main.c

This example shows how to start a newly created thread with a defined set of CPU registers.

```
/*
 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
 *               Alexander Warg <warg@os.inf.tu-dresden.de>,
 *               Björn Döbel <doebel@os.inf.tu-dresden.de>,
```

```
 *               Frank Mehnert <fm3@os.inf.tu-dresden.de>
 *     economic rights: Technische Universität Dresden (Germany)
 *
 * This file is part of TUD:OS and distributed under the terms of the
 * GNU General Public License 2.
 * Please see the COPYING-GPL-2 file for details.
 */
/*
 * Start a thread with an exception reply. This example does only work on
 * the x86-32 and ARM architectures.
 */

#include <l4/sys/thread.h>
#include <l4/sys/factory.h>
#include <l4/sys/ipc.h>
#include <l4/sys/utcb.h>
#include <l4/util/util.h>
#include <l4/re/env.h>
#include <l4/re/c/util/cap_alloc.h>

#include <stdlib.h>
#include <stdio.h>

/* Stack for the thread to be created. 8kB are enough. */
static char thread_stack[8 << 10];

/* The thread to be created. For illustration it will print out its
 * register set.
 */
static void L4_STICKY(thread_func(l4_umword_t *d))
{
  while (1)
    {
      printf("hey, I'm a thread\n");
      printf("got register values: %ld %ld %ld %ld %ld %ld\n",
             d[7], d[6], d[5], d[4], d[2], d[1], d[0]);
      l4_sleep(800);
    }
}

/* Startup trick for this example. Put all the CPU registers on the stack so
 * that the C function above can get it on the stack. */
asm(
".global thread     \n\t"
"thread:      \n\t"
#ifdef ARCH_x86
" pusha     \n\t"
" push %esp   \n\t"
" call thread_func  \n\t"
#endif
#ifdef ARCH_arm
"       push {r0-r7}            \n\t"
"       mov r0, sp              \n\t"
"       bl thread_func          \n\t"
#endif
);
extern void thread(void);

/* Our main function */
int main(void)
{
  /* Get a capability slot for our new thread. */
  l4_cap_idx_t t1 = l4re_util_cap_alloc();
  l4_utcb_t *u = l4_utcb();
  l4_exc_regs_t *e = l4_utcb_exc_u(u);
  l4_msgtag_t tag;
  int err;

  printf("Example showing how to start a thread with an exception.\n");
  /* We do not want to implement a pager here, take the shortcut. */
  printf("Make sure to start this program with ldr-flags=eager_map\n");

  if (l4_is_invalid_cap(t1))
    return 1;

  /* Create the thread using our default factory */
  tag = l4_factory_create_thread(l4re_env()->factory, t1);
  if (l4_error(tag))
    return 1;

  /* Setup the thread by setting the pager and task. */
  l4_thread_control_start();
  l4_thread_control_pager(l4re_env()->main_thread);
  l4_thread_control_exc_handler(l4re_env()->main_thread);
  l4_thread_control_bind((l4_utcb_t *)l4re_env()->first_free_utcb,
                         L4RE_THIS_TASK_CAP);
  tag = l4_thread_control_commit(t1);
```

```
  if (l4_error(tag))
    return 2;

  /* Start the thread by finally setting instruction and stack pointer */
  tag = l4_thread_ex_regs(t1,
                          (l4_umword_t)thread,
                          (l4_umword_t)thread_stack + sizeof(thread_stack),
                          L4_THREAD_EX_REGS_TRIGGER_EXCEPTION);
  if (l4_error(tag))
    return 3;

  l4_sched_param_t sp = l4_sched_param(1, 0);
  tag = l4_scheduler_run_thread(l4re_env()->scheduler, t1, &sp);
  if (l4_error(tag))
    return 4;


  /* Receive initial exception from just started thread */
  tag = l4_ipc_receive(t1, u, L4_IPC_NEVER);
  if ((err = l4_ipc_error(tag, u)))
    {
      printf("Umm, ipc error: %x\n", err);
      return 1;
    }
  /* We expect an exception IPC */
  if (!l4_msgtag_is_exception(tag))
    {
      printf("PF?: %lx %lx (not prepared to handle this) %ld\n",
       l4_utcb_mr_u(u)->mr[0], l4_utcb_mr_u(u)->mr[1], l4_msgtag_label(tag));
      return 1;
    }

  /* Fill out the complete register set of the new thread */
  e->sp = (l4_umword_t)(thread_stack + sizeof(thread_stack));
#ifdef ARCH_x86
  e->ip = (l4_umword_t)thread;
  e->eax = 1;
  e->ebx = 4;
  e->ecx = 2;
  e->edx = 3;
  e->esi = 6;
  e->edi = 7;
  e->ebp = 5;
#endif
#ifdef ARCH_arm
  e->pc = (l4_umword_t)thread;
  e->r[0] = 0;
  e->r[1] = 1;
  e->r[2] = 2;
  e->r[3] = 3;
  e->r[4] = 4;
  e->r[5] = 5;
  e->r[6] = 6;
  e->r[7] = 7;
#endif
  /* Send a complete exception */
  tag = l4_msgtag(0, L4_UTCB_EXCEPTION_REGS_SIZE, 0, 0);

  /* Send reply and start the thread with the defined CPU register set */
  tag = l4_ipc_send(t1, u, tag, L4_IPC_NEVER);
  if ((err = l4_ipc_error(tag, u)))
    printf("Error sending IPC: %x\n", err);

  /* Idle around */
  while (1)
    l4_sleep(10000);

  return 0;
}
```

13.23 examples/sys/utcb-ipc/main.c

This example shows how to send IPC using the UTCB to store payload.

```
/*
 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
 *               Alexander Warg <warg@os.inf.tu-dresden.de>,
 *               Björn Döbel <doebel@os.inf.tu-dresden.de>
 *     economic rights: Technische Universität Dresden (Germany)
 *
 * This file is part of TUD:OS and distributed under the terms of the
```

```
 * GNU General Public License 2.
 * Please see the COPYING-GPL-2 file for details.
 */
#include <l4/sys/ipc.h>
#include <l4/sys/thread.h>
#include <l4/sys/factory.h>
#include <l4/sys/utcb.h>
#include <l4/re/env.h>
#include <l4/re/c/util/cap_alloc.h>

#include <stdio.h>
#include <string.h>

static unsigned char stack2[8 << 10];
static l4_cap_idx_t thread1_cap, thread2_cap;

static void thread1(void)
{
  l4_msg_regs_t *mr = l4_utcb_mr();
  l4_msgtag_t tag;
  int i, j;

  printf("Thread1 up (%p)\n", l4_utcb());

  for (i = 0; i < 10; i++)
    {
      for (j = 0; j < L4_UTCB_GENERIC_DATA_SIZE; j++)
        mr->mr[j] = 'A' + (i + j) % ('~' - 'A' + 1);
      tag = l4_msgtag(0, L4_UTCB_GENERIC_DATA_SIZE, 0, 0);
      if (l4_msgtag_has_error(l4_ipc_send(thread2_cap,
      l4_utcb(), tag, L4_IPC_NEVER)))
  printf("IPC-send error\n");
    }
}

static void thread2(void)
{
  l4_msgtag_t tag;
  l4_msg_regs_t mr;
  unsigned i;

  printf("Thread2 up (%p)\n", l4_utcb());

  while (1)
    {
      if (l4_msgtag_has_error(tag = l4_ipc_receive(thread1_cap,
      l4_utcb(), L4_IPC_NEVER)))
        printf("IPC receive error\n");
      memcpy(&mr, l4_utcb_mr(), sizeof(mr));
      printf("Thread2 receive (%d): ", l4_msgtag_words(tag));
      for (i = 0; i < l4_msgtag_words(tag); i++)
        printf("%c", (char)mr.mr[i]);
      printf("\n");
    }
}

int main(void)
{
  l4_msgtag_t tag;

  thread1_cap = l4re_env()->main_thread;
  thread2_cap = l4re_util_cap_alloc();

  if (l4_is_invalid_cap(thread2_cap))
    return 1;

  tag = l4_factory_create_thread(l4re_env()->factory, thread2_cap);
  if (l4_error(tag))
    return 1;

  l4_thread_control_start();
  l4_thread_control_pager(l4re_env()->rm);
  l4_thread_control_exc_handler(l4re_env()->rm);
  l4_thread_control_bind((l4_utcb_t *)l4re_env()->first_free_utcb,
                         L4RE_THIS_TASK_CAP);
  tag = l4_thread_control_commit(thread2_cap);
  if (l4_error(tag))
    return 2;

  tag = l4_thread_ex_regs(thread2_cap,
                          (l4_umword_t)thread2,
                          (l4_umword_t)(stack2 + sizeof(stack2)), 0);
  if (l4_error(tag))
    return 3;

  l4_sched_param_t sp = l4_sched_param(1, 0);
  tag = l4_scheduler_run_thread(l4re_env()->scheduler, thread2_cap, &sp);
```

```
  if (l4_error(tag))
    return 4;

  thread1();

  return 0;
}
```

13.24 examples/sys/ux-vhw/main.c

This example shows how to iterate the virtual hardware descriptors under Fiasco-UX.

```
/*
 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
 *               Alexander Warg <warg@os.inf.tu-dresden.de>
 *     economic rights: Technische Universität Dresden (Germany)
 *
 * This file is part of TUD:OS and distributed under the terms of the
 * GNU General Public License 2.
 * Please see the COPYING-GPL-2 file for details.
 */
#include <l4/sys/ipc.h>
#include <l4/sys/vhw.h>
#include <l4/util/util.h>
#include <l4/util/kip.h>
#include <l4/re/env.h>

#include <stdlib.h>
#include <stdio.h>

static void print_entry(struct l4_vhw_entry *e)
{
  printf("type: %d  mem start: %08lx  end: %08lx\n"
         "irq: %d pid %d\n",
   e->type, e->mem_start, e->mem_size,
   e->irq_no, e->provider_pid);
}

int main(void)
{
  l4_kernel_info_t *kip = l4re_kip();
  struct l4_vhw_descriptor *vhw;
  int i;

  if (!kip)
    {
      printf("KIP not available!\n");
      return 1;
    }

  if (!l4util_kip_kernel_is_ux(kip))
    {
      printf("This example is for Fiasco-UX only.\n");
      return 1;
    }

  vhw = l4_vhw_get(kip);

  printf("kip at %p, vhw at %p\n", kip, vhw);
  printf("magic: %08x, version: %08x, count: %02d\n",
         vhw->magic, vhw->version, vhw->count);

  for (i = 0; i < vhw->count; i++)
    print_entry(l4_vhw_get_entry(vhw, i));

  return 0;
}
```

13.25 hello/server/src/main.c

This is the famous "Hello World!" program.

```
/*
 * (c) 2008-2009 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
 *               Frank Mehnert <fm3@os.inf.tu-dresden.de>,
 *               Lukas Grützmacher <lg2@os.inf.tu-dresden.de>
```

```
 *     economic rights: Technische Universität Dresden (Germany)
 *
 * This file is part of TUD:OS and distributed under the terms of the
 * GNU General Public License 2.
 * Please see the COPYING-GPL-2 file for details.
 */
#include <stdio.h>
#include <unistd.h>

int
main(void)
{
  for (;;)
    {
      puts("Hello World!");
      sleep(1);
    }
}
```

13.26 tmpfs/lib/src/fs.cc

Example file system for L4Re::Vfs.

```
/*
 * (c) 2010 Adam Lackorzynski <adam@os.inf.tu-dresden.de>,
 *          Alexander Warg <warg@os.inf.tu-dresden.de>
 *     economic rights: Technische Universität Dresden (Germany)
 * This file is part of TUD:OS and distributed under the terms of the
 * GNU Lesser General Public License 2.1.
 * Please see the COPYING-LGPL-2.1 file for details.
 */

#include <l4/l4re_vfs/backend>
#include <l4/cxx/string>
#include <l4/cxx/avl_tree>

#include <sys/stat.h>
#include <sys/ioctl.h>
#include <dirent.h>

#include <cstdio>

namespace {

using namespace L4Re::Vfs;
using cxx::Ref_ptr;

class File_data
{
public:
  File_data() : _buf(0), _size(0) {}

  unsigned long put(unsigned long offset,
                    unsigned long bufsize, void *srcbuf);
  unsigned long get(unsigned long offset,
                    unsigned long bufsize, void *dstbuf);

  unsigned long size(unsigned long offset);
  unsigned long size() const { return _size; }

  ~File_data() throw() { free(_buf); }

private:
  void *_buf;
  unsigned long _size;
};

unsigned long
File_data::put(unsigned long offset, unsigned long bufsize, void *srcbuf)
{
  if (offset + bufsize > _size)
    size(offset + bufsize);

  if (!_buf)
    return 0;

  memcpy((char *)_buf + offset, srcbuf, bufsize);
  return bufsize;
}

unsigned long
```

```
File_data::get(unsigned long offset, unsigned long bufsize, void *dstbuf)
{
  unsigned long s = bufsize;

  if (offset > _size)
    return 0;

  if (offset + bufsize > _size)
    s = _size - offset;

  memcpy(dstbuf, (char *)_buf + offset, s);
  return s;
}

unsigned long
File_data::size(unsigned long offset)
{
  if (offset != _size)
    {
      _size = offset;
      _buf = realloc(_buf, _size);
    }

  if (_buf)
    return 0;
  return -ENOSPC;
}


class Node : public cxx::Avl_tree_node
{
public:
  Node(const char *path, mode_t mode)
    : _ref_cnt(0), _path(strdup(path))
  {
    memset(&_info, 0, sizeof(_info));
    _info.st_mode = mode;
  }

  const char *path() const { return _path; }
  struct stat64 *info() { return &_info; }

  void add_ref() throw() { ++_ref_cnt; }
  int remove_ref() throw() { return --_ref_cnt; }

  bool is_dir() const { return S_ISDIR(_info.st_mode); }

  virtual ~Node() { free(_path); }

private:
  int           _ref_cnt;
  char         *_path;
  struct stat64 _info;
};

struct Node_get_key
{
  typedef cxx::String Key_type;
  static Key_type key_of(Node const *n)
  { return n->path(); }
};

struct Path_avl_tree_compare
{
  bool operator () (const char *l, const char *r) const
  { return strcmp(l, r) < 0; }
  bool operator () (const cxx::String l, const cxx::String r) const
  {
    int v = strncmp(l.start(), r.start(), cxx::min(l.len(), r.len()));
    return v < 0 || (v == 0 && l.len() < r.len());
  }
};

class Pers_file : public Node
{
public:
  Pers_file(const char *name, mode_t mode)
    : Node(name, (mode & 0777) | __S_IFREG) {}
  File_data const &data() const { return _data; }
  File_data &data() { return _data; }
private:
  File_data     _data;
};

class Pers_dir : public Node
{
private:
```

```
  typedef cxx::Avl_tree<Node, Node_get_key, Path_avl_tree_compare>
      Tree;
  Tree _tree;

public:
  Pers_dir(const char *name, mode_t mode)
    : Node(name, (mode & 0777) | __S_IFDIR) {}
  Ref_ptr<Node> find_path(cxx::String);
  bool add_node(Ref_ptr<Node> const &);

  typedef Tree::Const_iterator Const_iterator;
  Const_iterator begin() const { return _tree.begin(); }
  Const_iterator end() const { return _tree.end(); }
};

Ref_ptr<Node> Pers_dir::find_path(cxx::String path)
{
  return cxx::ref_ptr(_tree.find_node(path));
}

bool Pers_dir::add_node(Ref_ptr<Node> const &n)
{
  bool e = _tree.insert(n.ptr()).second;
  if (e)
    n->add_ref();
  return e;
}

class Tmpfs_dir : public Be_file
{
public:
  explicit Tmpfs_dir(Ref_ptr<Pers_dir> const &d) throw()
    : _dir(d), _getdents_state(false) {}
  int get_entry(const char *, int, mode_t, Ref_ptr<File> *) throw();
  ssize_t getdents(char *, size_t) throw();
  int fstat64(struct stat64 *buf) const throw();
  int utime(const struct utimbuf *) throw();
  int fchmod(mode_t) throw();
  int mkdir(const char *, mode_t) throw();
  int unlink(const char *) throw();
  int rename(const char *, const char *) throw();
  int faccessat(const char *, int, int) throw();

private:
  int walk_path(cxx::String const &_s,
                Ref_ptr<Node> *ret, cxx::String *remaining = 0);

  Ref_ptr<Pers_dir> _dir;
  bool _getdents_state;
  Pers_dir::Const_iterator _getdents_iter;
};

class Tmpfs_file : public Be_file_pos
{
public:
  explicit Tmpfs_file(Ref_ptr<Pers_file> const &f) throw()
    : Be_file_pos(), _file(f) {}

  off64_t size() const throw();
  int fstat64(struct stat64 *buf) const throw();
  int ftruncate64(off64_t p) throw();
  int ioctl(unsigned long, va_list) throw();
  int utime(const struct utimbuf *) throw();
  int fchmod(mode_t) throw();

private:
  ssize_t preadv(const struct iovec *v, int iovcnt, off64_t p) throw();
  ssize_t pwritev(const struct iovec *v, int iovcnt, off64_t p) throw();
  Ref_ptr<Pers_file> _file;
};

ssize_t Tmpfs_file::preadv(const struct iovec *v, int iovcnt, off64_t p) throw()
{
  if (iovcnt < 0)
    return -EINVAL;

  ssize_t sum = 0;
  for (int i = 0; i < iovcnt; ++i)
    {
      sum  += _file->data().get(p, v[i].iov_len, v[i].iov_base);
      p += v[i].iov_len;
    }
  return sum;
}

ssize_t Tmpfs_file::pwritev(const struct iovec *v, int iovcnt, off64_t p) throw()
{
```

```
  if (iovcnt < 0)
    return -EINVAL;

  ssize_t sum = 0;
  for (int i = 0; i < iovcnt; ++i)
    {
      sum  += _file->data().put(p, v[i].iov_len, v[i].iov_base);
      p += v[i].iov_len;
    }
  return sum;
}

int Tmpfs_file::fstat64(struct stat64 *buf) const throw()
{
  _file->info()->st_size = _file->data().size();
  memcpy(buf, _file->info(), sizeof(*buf));
  return 0;
}

int Tmpfs_file::ftruncate64(off64_t p) throw()
{
  if (p < 0)
      return -EINVAL;

  if (_file->data().size(p) == 0)
      return 0;

  return -EIO; // most likely ENOSPC, but can't report that
}

off64_t Tmpfs_file::size() const throw()
{ return _file->data().size(); }

int
Tmpfs_file::ioctl(unsigned long v, va_list args) throw()
{
  switch (v)
    {
    case FIONREAD: // return amount of data still available
      int *available = va_arg(args, int *);
      *available = _file->data().size() - pos();
      return 0;
    };
  return -EINVAL;
}

int
Tmpfs_file::utime(const struct utimbuf *times) throw()
{
  _file->info()->st_atime = times->actime;
  _file->info()->st_mtime = times->modtime;
  return 0;
}

int
Tmpfs_file::fchmod(mode_t m) throw()
{
  _file->info()->st_mode = m;
  return 0;
}

int
Tmpfs_dir::faccessat(const char *path, int mode, int) throw()
{
  Ref_ptr<Node> node;
  cxx::String name = path;

  int err = walk_path(name, &node, &name);
  if (err < 0)
    return err;

  if (mode == F_OK) // existence check
    return 0;

  struct stat64 *stats = node->info();

  if ((mode & R_OK) && !(stats->st_mode & S_IRUSR))
    return -EACCES;

  if ((mode & W_OK) && !(stats->st_mode & S_IWUSR))
    return -EACCES;

  if ((mode & X_OK) && !(stats->st_mode & S_IXUSR))
    return -EACCES;

  return 0;
}
```

```
int
Tmpfs_dir::get_entry(const char *name, int flags, mode_t mode,
                     Ref_ptr<File> *file) throw()
{
  Ref_ptr<Node> path;
  if (!*name)
    {
      *file = cxx::ref_ptr(this);
      return 0;
    }

  cxx::String n = name;

  int e = walk_path(n, &path, &n);

  if (e == -ENOTDIR)
    return e;

  if (!(flags & O_CREAT) && e < 0)
    return e;

  if ((flags & O_CREAT) && e == -ENOENT)
    {
      Ref_ptr<Node> node(new Pers_file(n.start(), mode));
      // when ENOENT is return, path is always a directory
      bool e = cxx::ref_ptr_static_cast<Pers_dir>(path)->add_node(node);
      if (!e)
        return -ENOMEM;
      path = node;
    }

  if (path->is_dir())
    *file = cxx::ref_ptr(new Tmpfs_dir(cxx::ref_ptr_static_cast<Pers_dir>(path)));
  else
    *file = cxx::ref_ptr(new Tmpfs_file(cxx::ref_ptr_static_cast<Pers_file>(path)));

  if (!*file)
    return -ENOMEM;


  return 0;
}

ssize_t
Tmpfs_dir::getdents(char *buf, size_t sz) throw()
{
  struct dirent64 *d = (struct dirent64 *)buf;
  ssize_t ret = 0;

  if (!_getdents_state)
    {
      _getdents_iter = _dir->begin();
      _getdents_state = true;
    }
  else if (_getdents_iter == _dir->end())
    {
      _getdents_state = false;
      return 0;
    }

  for (; _getdents_iter != _dir->end(); ++_getdents_iter)
    {
      unsigned l = strlen(_getdents_iter->path()) + 1;
      if (l > sizeof(d->d_name))
        l = sizeof(d->d_name);

      unsigned n = offsetof (struct dirent64, d_name) + l;
      n = (n + sizeof(long) - 1) & ~(sizeof(long) - 1);

      if (n > sz)
        break;

      d->d_ino = 1;
      d->d_off = 0;
      memcpy(d->d_name, _getdents_iter->path(), l);
      d->d_reclen = n;
      d->d_type   = DT_REG;
      ret += n;
      sz  -= n;
      d    = (struct dirent64 *)((unsigned long)d + n);
    }

  return ret;
}

int
```

```
Tmpfs_dir::fstat64(struct stat64 *buf) const throw()
{
  memcpy(buf, _dir->info(), sizeof(*buf));
  return 0;
}

int
Tmpfs_dir::utime(const struct utimbuf *times) throw()
{
  _dir->info()->st_atime = times->actime;
  _dir->info()->st_mtime = times->modtime;
  return 0;
}

int
Tmpfs_dir::fchmod(mode_t m) throw()
{
  _dir->info()->st_mode = m;
  return 0;
}

int
Tmpfs_dir::walk_path(cxx::String const &_s,
                     Ref_ptr<Node> *ret, cxx::String *remaining)
{
  Ref_ptr<Pers_dir> p = _dir;
  cxx::String s = _s;
  Ref_ptr<Node> n;

  while (1)
    {
      if (s.len() == 0)
        {
          *ret = p;
          return 0;
        }

      cxx::String::Index sep = s.find("/");

      if (sep - s.start() == 1 && *s.start() == '.')
        {
          s = s.substr(s.start() + 2);
          continue;
        }

      n = p->find_path(s.head(sep - s.start()));

      if (!n)
        {
          *ret = p;
          if (remaining)
            *remaining = s.head(sep - s.start());
          return -ENOENT;
        }


      if (sep == s.end())
        {
          *ret = n;
          return 0;
        }

      if (!n->is_dir())
        return -ENOTDIR;

      s = s.substr(sep + 1);

      p = cxx::ref_ptr_static_cast<Pers_dir>(n);
    }

  *ret = n;

  return 0;
}

int
Tmpfs_dir::mkdir(const char *name, mode_t mode) throw()
{
  Ref_ptr<Node> node = _dir;
  cxx::String p = cxx::String(name);
  cxx::String path, last = p;
  cxx::String::Index s = p.rfind("/");

  // trim /'s at the end
  while (p.len() && s == p.end() - 1)
    {
      p.len(p.len() - 1);
```

```
      s = p.rfind("/");
    }

  //printf("MKDIR '%s' p=%p %p\n", name, p.start(), s);

  if (s != p.end())
    {
      path = p.head(s);
      last = p.substr(s + 1, p.end() - s);

      int e = walk_path(path, &node);
      if (e < 0)
        return e;
    }

  if (!node->is_dir())
    return -ENOTDIR;

  // due to path walking we can end up with an empty name
  if (p.len() == 0 || p == cxx::String("."))
    return 0;

  Ref_ptr<Pers_dir> dnode = cxx::ref_ptr_static_cast<Pers_dir>(node);

  Ref_ptr<Pers_dir> dir(new Pers_dir(last.start(), mode));
  return dnode->add_node(dir) ? 0 : -EEXIST;
}

int
Tmpfs_dir::unlink(const char *name) throw()
{
  cxx::Ref_ptr<Node> n;

  int e = walk_path(name, &n);
  if (e < 0)
    return -ENOENT;

  printf("Unimplemented (if file exists): %s(%s)\n", __func__, name);
  return -ENOMEM;
}

int
Tmpfs_dir::rename(const char *old, const char *newn) throw()
{
  printf("Unimplemented: %s(%s, %s)\n", __func__, old, newn);
  return -ENOMEM;
}



class Tmpfs_fs : public Be_file_system
{
public:
  Tmpfs_fs() : Be_file_system("tmpfs") {}
  int mount(char const *source, unsigned long mountflags,
            void const *data, cxx::Ref_ptr<File> *dir) throw()
  {
    (void)mountflags;
    (void)source;
    (void)data;
    *dir = cxx::ref_ptr(new Tmpfs_dir(cxx::ref_ptr(new Pers_dir("root", 0777))));
    if (!*dir)
      return -ENOMEM;
    return 0;
  }
};

static Tmpfs_fs _tmpfs L4RE_VFS_FILE_SYSTEM_ATTRIBUTE;

}
```

Index

Atomic Instructions, 64
Attach_flags
    Region map API, 399
Auxiliary data, 71

Bad_address
    L4 VIRTIO Transport Layer, 297
Bad_flags
    L4 VIRTIO Transport Layer, 297
Bad_next
    L4 VIRTIO Transport Layer, 297
Bad_rights
    L4 VIRTIO Transport Layer, 297
Bad_size
    L4 VIRTIO Transport Layer, 297
Basic Macros, 75
Bidirectional
    L4Re::Dma_space, 1055
Bit Manipulation, 77
Bitmap graphics and fonts, 83
Br_bytes
    L4::Ipc::Msg, 554
Buffer Registers (BRs)
    L4_BDR_IO_SHIFT, 84
    L4_BDR_MEM_SHIFT, 84
    L4_BDR_OBJ_SHIFT, 84

C++ Exceptions, 85
C_bits
    Small C++ Template Library, 430
Cache Consistency, 90
Cache_buffered
    Region map API, 400
Cache_normal
    Region map API, 400
Cache_uncached
    Region map API, 400
Caching
    Region map API, 400
Caching_ds_shift
    Region map API, 400
Caching_shift
    Region map API, 400
Cap_mask
    L4::Ipc::Cap, 764
Capabilities, 93
    L4_BASE_DEBUGGER_CAP, 94
    L4_BASE_FACTORY_CAP, 94
    L4_BASE_ICU_CAP, 94
    L4_BASE_IOMMU_CAP, 94
    L4_BASE_LOG_CAP, 94
    L4_BASE_PAGER_CAP, 94
    L4_BASE_SCHEDULER_CAP, 94
    L4_BASE_TASK_CAP, 94
    L4_BASE_THREAD_CAP, 94
    L4_CAP_MASK, 94
    L4_CAP_SHIFT, 94
    L4_CAP_SIZE, 94
    L4_INVALID_CAP, 94
Capability allocator, 97
Caps
    L4 kernel object type information, 302
Chunks, 98
Coherent
    L4Re::Dma_space, 1055
Comfortable Command Line Parsing, 103
Consumer, 106, 110
Continuous
    L4Re::Mem_alloc, 1083
Cpu_disable
    L4::Platform_control, 895
Cpu_enable
    L4::Platform_control, 895
cxx, 537
cxx::Bits::Direction
    L, 635
    N, 635
    R, 635

Dataspace interface, 117
    L4RE_DS_MAP_BUFFERABLE, 118
    L4RE_DS_MAP_CACHEABLE, 118
    L4RE_DS_MAP_CACHING_MASK, 118
    L4RE_DS_MAP_CACHING_SHIFT, 118
    L4RE_DS_MAP_NORMAL, 118
    L4RE_DS_MAP_UNCACHEABLE, 118
Debug interface, 120
Detach_again
    Region map API, 399
Detach_exact
    Region map API, 399
Detach_free
    Region map API, 400
Detach_keep
    Region map API, 399
Detach_overlap
    Region map API, 399
Detached_ds
    Region map API, 399

EDID parsing functionality
Libedid_block_size, 122
Eager_map
Region map API, 399
Error codes, 149
L4_EACCESS, 149
L4_EADDRNOTAVAIL, 150
L4_EAGAIN, 149
L4_EBADPROTO, 150
L4_EBUSY, 149
L4_EEXIST, 149
L4_EINVAL, 149
L4_EIO, 149
L4_EIPC_HI, 150
L4_EIPC_LO, 150
L4_EMSGMISSARG, 150
L4_EMSGTOOLONG, 150
L4_EMSGTOOSHORT, 150
L4_ENAMETOOLONG, 150
L4_ENODEV, 149
L4_ENOENT, 149
L4_ENOMEM, 149
L4_ENOREPLY, 150
L4_ENOSYS, 150
L4_EOK, 149
L4_EPERM, 149
L4_ERANGE, 150
L4_ERRNOMAX, 150
Error Handling, 144
L4_IPC_ENOT_EXISTENT, 145
L4_IPC_ERROR_MASK, 145
L4_IPC_REABORTED, 145
L4_IPC_RECANCELED, 145
L4_IPC_REMAPFAILED, 145
L4_IPC_REMSGCUT, 145
L4_IPC_RERCVPFTO, 145
L4_IPC_RESNDPFTO, 145
L4_IPC_RETIMEOUT, 145
L4_IPC_SEABORTED, 145
L4_IPC_SECANCELED, 145
L4_IPC_SEMAPFAILED, 145
L4_IPC_SEMSGCUT, 145
L4_IPC_SERCVPFTO, 145
L4_IPC_SESNDPFTO, 145
L4_IPC_SETIMEOUT, 145
L4_IPC_SND_ERR_MASK, 145
Event interface, 152
Exception registers, 155

F_above
L4Re::Video::View, 1208
F_auto_refresh
L4Re::Video::Goos, 1196
F_dyn_allocated
L4Re::Video::View, 1208
F_dynamic_buffers
L4Re::Video::Goos, 1196
F_dynamic_views
L4Re::Video::Goos, 1196
F_flags_mask
L4Re::Video::View, 1208
F_fully_dynamic
L4Re::Video::View, 1208
F_l4re_video_goos_auto_refresh
Video API, 503
F_l4re_video_goos_dynamic_buffers
Video API, 504
F_l4re_video_goos_dynamic_views
Video API, 504
F_l4re_video_goos_pointer
Video API, 503
F_l4re_video_view_above
Video API, 504
F_l4re_video_view_dyn_allocated
Video API, 504
F_l4re_video_view_flags_mask
Video API, 504
F_l4re_video_view_none
Video API, 504
F_l4re_video_view_set_background
Video API, 504
F_l4re_video_view_set_buffer
Video API, 504
F_l4re_video_view_set_buffer_offset
Video API, 504
F_l4re_video_view_set_bytes_per_line
Video API, 504
F_l4re_video_view_set_flags
Video API, 504
F_l4re_video_view_set_pixel
Video API, 504
F_l4re_video_view_set_position
Video API, 504
F_none
L4Re::Video::View, 1208
F_pointer
L4Re::Video::Goos, 1196
F_set_background
L4Re::Video::View, 1208
F_set_buffer
L4Re::Video::View, 1208
F_set_buffer_offset
L4Re::Video::View, 1208
F_set_bytes_per_line
L4Re::Video::View, 1208
F_set_flags
L4Re::Video::View, 1208
F_set_pixel
L4Re::Video::View, 1208
F_set_position
L4Re::Video::View, 1208
Factory, 158
Fiasco extensions, 172
Fiasco real time scheduling extensions, 178
Fiasco-UX Virtual devices
L4_TYPE_VHW_FRAMEBUFFER, 179
L4_TYPE_VHW_INPUT, 179

L4_TYPE_VHW_NET, 179
L4_TYPE_VHW_NONE, 179
Flags
L4 kernel object type information, 302
Flex pages, 180
L4_CAP_FPAGE_D, 182
L4_CAP_FPAGE_R, 182
L4_CAP_FPAGE_RO, 182
L4_CAP_FPAGE_RS, 183
L4_CAP_FPAGE_RSD, 183
L4_CAP_FPAGE_RW, 182
L4_CAP_FPAGE_RWD, 183
L4_CAP_FPAGE_RWS, 183
L4_CAP_FPAGE_RWSD, 183
L4_CAP_FPAGE_S, 182
L4_CAP_FPAGE_W, 182
L4_FPAGE_ADDR_BITS, 183
L4_FPAGE_ADDR_SHIFT, 183
L4_FPAGE_BUFFERABLE, 183
L4_FPAGE_C_NO_REF_CNT, 184
L4_FPAGE_CACHE_OPT, 183
L4_FPAGE_CACHEABLE, 183
L4_FPAGE_RIGHTS_BITS, 183
L4_FPAGE_RIGHTS_MASK, 183
L4_FPAGE_RIGHTS_SHIFT, 183
L4_FPAGE_RO, 184
L4_FPAGE_RW, 184
L4_FPAGE_RWX, 184
L4_FPAGE_RX, 184
L4_FPAGE_SIZE_BITS, 183
L4_FPAGE_SIZE_SHIFT, 183
L4_FPAGE_TYPE_BITS, 183
L4_FPAGE_TYPE_SHIFT, 183
L4_FPAGE_UNCACHEABLE, 183
L4_FPAGE_W, 184
L4_FPAGE_X, 184
L4_IOPORT_MAX, 182
L4_WHOLE_ADDRESS_SPACE, 182
L4_WHOLE_IOADDRESS_SPACE, 182
From_device
L4Re::Dma_space, 1055
Functions for rendering bitmap data in frame buffers, 192
Functions for rendering bitmap fonts to frame buffers, 195

IO interface
L4IO_DEVICE_ANY, 204
L4IO_DEVICE_INVALID, 204
L4IO_DEVICE_OTHER, 204
L4IO_DEVICE_PCI, 204
L4IO_DEVICE_USB, 204
L4IO_MEM_CACHED, 204
L4IO_MEM_EAGER_MAP, 204
L4IO_MEM_NONCACHED, 204
L4IO_MEM_USE_MTRR, 204
L4IO_MEM_USE_RESERVED_AREA, 204
L4IO_RESOURCE_ANY, 204
L4IO_RESOURCE_INVALID, 204
L4IO_RESOURCE_IRQ, 204
L4IO_RESOURCE_MEM, 204
L4IO_RESOURCE_PORT, 204
IRQs
L4_IRQ_F_BOTH, 215
L4_IRQ_F_BOTH_EDGE, 215
L4_IRQ_F_CLEAR_WAKEUP, 215
L4_IRQ_F_EDGE, 215
L4_IRQ_F_LEVEL, 215
L4_IRQ_F_LEVEL_HIGH, 215
L4_IRQ_F_LEVEL_LOW, 215
L4_IRQ_F_MASK, 215
L4_IRQ_F_NEG, 215
L4_IRQ_F_NEG_EDGE, 215
L4_IRQ_F_NONE, 215
L4_IRQ_F_POS, 215
L4_IRQ_F_POS_EDGE, 215
L4_IRQ_F_SET_WAKEUP, 215
In_area
Region map API, 399
info
Scheduler, 412
Initial Environment, 229
Integer Types, 234
Interface using direct functionality., 240, 244
Internal constants, 247
Internal functions, 249
Internal Helpers, 246
Interrupt controller, 250
L4_ICU_FLAG_MSI, 252
Invalid
L4virtio::Ptr, 1294
Item_bytes
L4::Ipc::Msg, 554
Item_words
L4::Ipc::Msg, 554

Kept_ds
Region map API, 399
Kernel Debugger, 263
ko, 266
outchar, 268
outdec, 269
outhex12, 269
outhex16, 269
outhex20, 269
outhex32, 269
outhex8, 269
outnstring, 270
outstring, 270
Kernel Interface Page, 271
Kernel Objects, 277
ko
Kernel Debugger, 266
Kumem allocator utility, 279
Kumem utiltities, 280

L
cxx::Bits::Direction, 635
L4, 539

PROTO_ANY, 542
PROTO_EMPTY, 542
L4 IPC Opcodes
L4_ICU_OP_BIND, 281
L4_ICU_OP_INFO, 281
L4_ICU_OP_MASK, 282
L4_ICU_OP_MSI_INFO, 282
L4_ICU_OP_SET_MODE, 282
L4_ICU_OP_UNBIND, 281
L4_ICU_OP_UNMASK, 282
L4_IPC_GATE_BIND_OP, 282
L4_IPC_GATE_GET_INFO_OP, 282
L4_PLATFORM_CTL_CPU_DISABLE_OP, 282
L4_PLATFORM_CTL_CPU_ENABLE_OP, 282
L4_PLATFORM_CTL_SYS_SHUTDOWN_OP, 282
L4_PLATFORM_CTL_SYS_SUSPEND_OP, 282
L4_TASK_ADD_KU_MEM_OP, 283
L4_TASK_CAP_INFO_OP, 282
L4_TASK_LDT_SET_X86_OP, 283
L4_TASK_MAP_OP, 282
L4_TASK_UNMAP_OP, 282
L4_THREAD_AMD64_SET_SEGMENT_BASE_↩ OP, 283
L4_THREAD_ARM_TPIDRURO_OP, 283
L4_THREAD_CONTROL_OP, 283
L4_THREAD_EX_REGS_OP, 283
L4_THREAD_MODIFY_SENDER_OP, 283
L4_THREAD_OPCODE_MASK, 283
L4_THREAD_REGISTER_DELETE_IRQ_OP, 283
L4_THREAD_STATS_OP, 283
L4_THREAD_SWITCH_OP, 283
L4_THREAD_VCPU_CONTROL_OP, 283
L4_THREAD_VCPU_RESUME_OP, 283
L4_THREAD_X86_GDT_OP, 283
L4_VCON_GET_ATTR_OP, 283
L4_VCON_READ_OP, 283
L4_VCON_SET_ATTR_OP, 283
L4_VCON_WRITE_OP, 283
L4 kernel object type information, 301
Caps, 302
Flags, 302
Mem, 302
Ports, 302
L4 V-BUS functions
L4VBUS_DMAD_BIND, 285
L4VBUS_DMAD_KERNEL_DMA_SPACE, 285
L4VBUS_DMAD_L4RE_DMA_SPACE, 285
L4VBUS_DMAD_UNBIND, 285
L4 VIRTIO Block Device
L4VIRTIO_BLOCK_S_IOERR, 293
L4VIRTIO_BLOCK_S_OK, 293
L4VIRTIO_BLOCK_S_UNSUPP, 293
L4VIRTIO_BLOCK_T_FLUSH, 292
L4VIRTIO_BLOCK_T_IN, 292
L4VIRTIO_BLOCK_T_OUT, 292
L4 VIRTIO Transport Layer
Bad_address, 297
Bad_flags, 297
Bad_next, 297
Bad_rights, 297
Bad_size, 297
L4VIRTIO_ID_9P, 298
L4VIRTIO_ID_BALLOON, 298
L4VIRTIO_ID_BLOCK, 298
L4VIRTIO_ID_CAIF, 298
L4VIRTIO_ID_CONSOLE, 298
L4VIRTIO_ID_NET, 298
L4VIRTIO_ID_RNG, 298
L4VIRTIO_ID_RPMSG, 298
L4VIRTIO_ID_RPROC_SERIAL, 298
L4VIRTIO_ID_SCSI, 298
L4VIRTIO_ID_SOCK, 298
L4VIRTIO_IRQ_STATUS_CONFIG, 297
L4VIRTIO_IRQ_STATUS_VRING, 297
L4VIRTIO_OP_CONFIG_QUEUE, 297
L4VIRTIO_OP_REGISTER_DS, 298
L4VIRTIO_OP_REGISTER_IFACE, 297
L4VIRTIO_OP_SET_STATUS, 297
L4VIRTIO_STATUS_ACKNOWLEDGE, 298
L4VIRTIO_STATUS_DRIVER, 298
L4VIRTIO_STATUS_DRIVER_OK, 298
L4VIRTIO_STATUS_FAILED, 298
L4VIRTIO_STATUS_FEATURES_OK, 298
L4::Ipc::Cap
Cap_mask, 764
Rights_mask, 764
L4::Ipc::Msg
Br_bytes, 554
Item_bytes, 554
Item_words, 554
Mr_bytes, 554
Mr_words, 554
Word_bytes, 554
L4::Platform_control
Cpu_disable, 895
Cpu_enable, 895
Shutdown, 895
Suspend, 895
L4::Types::Flags
None, 979
L4_BASE_DEBUGGER_CAP
Capabilities, 94
L4_BASE_FACTORY_CAP
Capabilities, 94
L4_BASE_ICU_CAP
Capabilities, 94
L4_BASE_IOMMU_CAP
Capabilities, 94
L4_BASE_LOG_CAP
Capabilities, 94
L4_BASE_PAGER_CAP
Capabilities, 94
L4_BASE_SCHEDULER_CAP
Capabilities, 94
L4_BASE_TASK_CAP

Capabilities, 94
L4_BASE_THREAD_CAP
Capabilities, 94
L4_BDR_IO_SHIFT
Buffer Registers (BRs), 84
L4_BDR_MEM_SHIFT
Buffer Registers (BRs), 84
L4_BDR_OBJ_SHIFT
Buffer Registers (BRs), 84
L4_CAP_FPAGE_D
Flex pages, 182
L4_CAP_FPAGE_R
Flex pages, 182
L4_CAP_FPAGE_RO
Flex pages, 182
L4_CAP_FPAGE_RS
Flex pages, 183
L4_CAP_FPAGE_RSD
Flex pages, 183
L4_CAP_FPAGE_RW
Flex pages, 182
L4_CAP_FPAGE_RWD
Flex pages, 183
L4_CAP_FPAGE_RWS
Flex pages, 183
L4_CAP_FPAGE_RWSD
Flex pages, 183
L4_CAP_FPAGE_S
Flex pages, 182
L4_CAP_FPAGE_W
Flex pages, 182
L4_CAP_MASK
Capabilities, 94
L4_CAP_SHIFT
Capabilities, 94
L4_CAP_SIZE
Capabilities, 94
L4_EACCESS
Error codes, 149
L4_EADDRNOTAVAIL
Error codes, 150
L4_EAGAIN
Error codes, 149
L4_EBADPROTO
Error codes, 150
L4_EBUSY
Error codes, 149
L4_EEXIST
Error codes, 149
L4_EINVAL
Error codes, 149
L4_EIO
Error codes, 149
L4_EIPC_HI
Error codes, 150
L4_EIPC_LO
Error codes, 150
L4_EMSGMISSARG
Error codes, 150
L4_EMSGTOOLONG
Error codes, 150
L4_EMSGTOOSHORT
Error codes, 150
L4_ENAMETOOLONG
Error codes, 150
L4_ENODEV
Error codes, 149
L4_ENOENT
Error codes, 149
L4_ENOMEM
Error codes, 149
L4_ENOREPLY
Error codes, 150
L4_ENOSYS
Error codes, 150
L4_EOK
Error codes, 149
L4_EPERM
Error codes, 149
L4_ERANGE
Error codes, 150
L4_ERRNOMAX
Error codes, 150
L4_FP_ALL_SPACES
Task, 434
L4_FP_DELETE_OBJ
Task, 434
L4_FP_OTHER_SPACES
Task, 434
L4_FPAGE_ADDR_BITS
Flex pages, 183
L4_FPAGE_ADDR_SHIFT
Flex pages, 183
L4_FPAGE_BUFFERABLE
Flex pages, 183
L4_FPAGE_C_NO_REF_CNT
Flex pages, 184
L4_FPAGE_CACHE_OPT
Flex pages, 183
L4_FPAGE_CACHEABLE
Flex pages, 183
L4_FPAGE_RIGHTS_BITS
Flex pages, 183
L4_FPAGE_RIGHTS_MASK
Flex pages, 183
L4_FPAGE_RIGHTS_SHIFT
Flex pages, 183
L4_FPAGE_RO
Flex pages, 184
L4_FPAGE_RW
Flex pages, 184
L4_FPAGE_RWX
Flex pages, 184
L4_FPAGE_RX
Flex pages, 184
L4_FPAGE_SIZE_BITS

Flex pages, 183
L4_FPAGE_SIZE_SHIFT
Flex pages, 183
L4_FPAGE_TYPE_BITS
Flex pages, 183
L4_FPAGE_TYPE_SHIFT
Flex pages, 183
L4_FPAGE_UNCACHEABLE
Flex pages, 183
L4_FPAGE_W
Flex pages, 184
L4_FPAGE_X
Flex pages, 184
L4_ICU_FLAG_MSI
Interrupt controller, 252
L4_ICU_OP_BIND
L4 IPC Opcodes, 281
L4_ICU_OP_INFO
L4 IPC Opcodes, 281
L4_ICU_OP_MASK
L4 IPC Opcodes, 282
L4_ICU_OP_MSI_INFO
L4 IPC Opcodes, 282
L4_ICU_OP_SET_MODE
L4 IPC Opcodes, 282
L4_ICU_OP_UNBIND
L4 IPC Opcodes, 281
L4_ICU_OP_UNMASK
L4 IPC Opcodes, 282
L4_INVALID_ADDR
Memory related, 352
L4_INVALID_CAP
Capabilities, 94
L4_IOPORT_MAX
Flex pages, 182
L4_IPC_ENOT_EXISTENT
Error Handling, 145
L4_IPC_ERROR_MASK
Error Handling, 145
L4_IPC_GATE_BIND_OP
L4 IPC Opcodes, 282
L4_IPC_GATE_GET_INFO_OP
L4 IPC Opcodes, 282
L4_IPC_REABORTED
Error Handling, 145
L4_IPC_RECANCELED
Error Handling, 145
L4_IPC_REMAPFAILED
Error Handling, 145
L4_IPC_REMSGCUT
Error Handling, 145
L4_IPC_RERCVPFTO
Error Handling, 145
L4_IPC_RESNDPFTO
Error Handling, 145
L4_IPC_RETIMEOUT
Error Handling, 145
L4_IPC_SEABORTED
Error Handling, 145
L4_IPC_SECANCELED
Error Handling, 145
L4_IPC_SEMAPFAILED
Error Handling, 145
L4_IPC_SEMSGCUT
Error Handling, 145
L4_IPC_SERCVPFTO
Error Handling, 145
L4_IPC_SESNDPFTO
Error Handling, 145
L4_IPC_SETIMEOUT
Error Handling, 145
L4_IPC_SND_ERR_MASK
Error Handling, 145
L4_IRQ_F_BOTH
IRQs, 215
L4_IRQ_F_BOTH_EDGE
IRQs, 215
L4_IRQ_F_CLEAR_WAKEUP
IRQs, 215
L4_IRQ_F_EDGE
IRQs, 215
L4_IRQ_F_LEVEL
IRQs, 215
L4_IRQ_F_LEVEL_HIGH
IRQs, 215
L4_IRQ_F_LEVEL_LOW
IRQs, 215
L4_IRQ_F_MASK
IRQs, 215
L4_IRQ_F_NEG
IRQs, 215
L4_IRQ_F_NEG_EDGE
IRQs, 215
L4_IRQ_F_NONE
IRQs, 215
L4_IRQ_F_POS
IRQs, 215
L4_IRQ_F_POS_EDGE
IRQs, 215
L4_IRQ_F_SET_WAKEUP
IRQs, 215
L4_ITEM_CONT
Message Items, 355
L4_ITEM_MAP
Message Items, 355
L4_MAP_ITEM_GRANT
Message Items, 355
L4_MAP_ITEM_MAP
Message Items, 355
L4_MEM_WIDTH_1BYTE
Memory operations., 347
L4_MEM_WIDTH_2BYTE
Memory operations., 347
L4_MEM_WIDTH_4BYTE
Memory operations., 347
L4_MSGTAG_ERROR

Message Tag, 360
L4_MSGTAG_FLAGS
Message Tag, 361
L4_MSGTAG_PROPAGATE
Message Tag, 361
L4_MSGTAG_SCHEDULE
Message Tag, 361
L4_MSGTAG_TRANSFER_FPU
Message Tag, 361
L4_MSGTAG_XCPU
Message Tag, 360
L4_PLATFORM_CTL_CPU_DISABLE_OP
L4 IPC Opcodes, 282
L4_PLATFORM_CTL_CPU_ENABLE_OP
L4 IPC Opcodes, 282
L4_PLATFORM_CTL_SYS_SHUTDOWN_OP
L4 IPC Opcodes, 282
L4_PLATFORM_CTL_SYS_SUSPEND_OP
L4 IPC Opcodes, 282
L4_PROTO_ALLOW_SYSCALL
Message Tag, 361
L4_PROTO_DEBUGGER
Message Tag, 361
L4_PROTO_DMA_SPACE
Message Tag, 361
L4_PROTO_EXCEPTION
Message Tag, 361
L4_PROTO_FACTORY
Message Tag, 361
L4_PROTO_IO_PAGE_FAULT
Message Tag, 361
L4_PROTO_IOMMU
Message Tag, 361
L4_PROTO_IRQ
Message Tag, 361
L4_PROTO_IRQ_MUX
Message Tag, 361
L4_PROTO_IRQ_SENDER
Message Tag, 361
L4_PROTO_KOBJECT
Message Tag, 361
L4_PROTO_LOG
Message Tag, 361
L4_PROTO_META
Message Tag, 361
L4_PROTO_NONE
Message Tag, 361
L4_PROTO_PAGE_FAULT
Message Tag, 361
L4_PROTO_PF_EXCEPTION
Message Tag, 361
L4_PROTO_PLATFORM_CTL
Message Tag, 362
L4_PROTO_PREEMPTION
Message Tag, 361
L4_PROTO_SCHEDULER
Message Tag, 361
L4_PROTO_SEMAPHORE
Message Tag, 361
L4_PROTO_SIGMA0
Message Tag, 361
L4_PROTO_SYS_EXCEPTION
Message Tag, 361
L4_PROTO_TASK
Message Tag, 361
L4_PROTO_THREAD
Message Tag, 361
L4_PROTO_VM
Message Tag, 361
L4_RCV_ITEM_LOCAL_ID
Message Items, 356
L4_RCV_ITEM_SINGLE_CAP
Message Items, 356
L4_SCHEDULER_IDLE_TIME_OP
Scheduler, 412
L4_SCHEDULER_INFO_OP
Scheduler, 412
L4_SCHEDULER_RUN_THREAD_OP
Scheduler, 412
L4_SYSF_CALL
Object Invocation, 377
L4_SYSF_NONE
Object Invocation, 377
L4_SYSF_OPEN_WAIT
Object Invocation, 377
L4_SYSF_RECV
Object Invocation, 377
L4_SYSF_REPLY
Object Invocation, 377
L4_SYSF_REPLY_AND_WAIT
Object Invocation, 377
L4_SYSF_SEND
Object Invocation, 377
L4_SYSF_SEND_AND_WAIT
Object Invocation, 377
L4_SYSF_WAIT
Object Invocation, 377
L4_TASK_ADD_KU_MEM_OP
L4 IPC Opcodes, 283
L4_TASK_CAP_INFO_OP
L4 IPC Opcodes, 282
L4_TASK_LDT_SET_X86_OP
L4 IPC Opcodes, 283
L4_TASK_MAP_OP
L4 IPC Opcodes, 282
L4_TASK_UNMAP_OP
L4 IPC Opcodes, 282
L4_THREAD_AMD64_SET_SEGMENT_BASE_OP
L4 IPC Opcodes, 283
L4_THREAD_ARM_TPIDRURO_OP
L4 IPC Opcodes, 283
L4_THREAD_CONTROL_ALIEN
Thread, 444
L4_THREAD_CONTROL_BIND_TASK
Thread, 444
L4_THREAD_CONTROL_MR_IDX_BIND_TASK

Thread, 444
L4_THREAD_CONTROL_MR_IDX_BIND_UTCB
Thread, 444
L4_THREAD_CONTROL_MR_IDX_EXC_HANDLER
Thread, 444
L4_THREAD_CONTROL_MR_IDX_FLAG_VALS
Thread, 444
L4_THREAD_CONTROL_MR_IDX_FLAGS
Thread, 444
L4_THREAD_CONTROL_MR_IDX_PAGER
Thread, 444
L4_THREAD_CONTROL_OP
L4 IPC Opcodes, 283
L4_THREAD_CONTROL_SET_EXC_HANDLER
Thread, 444
L4_THREAD_CONTROL_SET_PAGER
Thread, 444
L4_THREAD_CONTROL_UX_NATIVE
Thread, 444
L4_THREAD_EX_REGS_CANCEL
Thread, 444
L4_THREAD_EX_REGS_OP
L4 IPC Opcodes, 283
L4_THREAD_EX_REGS_TRIGGER_EXCEPTION
Thread, 444
L4_THREAD_MODIFY_SENDER_OP
L4 IPC Opcodes, 283
L4_THREAD_OPCODE_MASK
L4 IPC Opcodes, 283
L4_THREAD_REGISTER_DELETE_IRQ_OP
L4 IPC Opcodes, 283
L4_THREAD_STATS_OP
L4 IPC Opcodes, 283
L4_THREAD_SWITCH_OP
L4 IPC Opcodes, 283
L4_THREAD_VCPU_CONTROL_OP
L4 IPC Opcodes, 283
L4_THREAD_VCPU_RESUME_OP
L4 IPC Opcodes, 283
L4_THREAD_X86_GDT_OP
L4 IPC Opcodes, 283
L4_TYPE_VHW_FRAMEBUFFER
Fiasco-UX Virtual devices, 179
L4_TYPE_VHW_INPUT
Fiasco-UX Virtual devices, 179
L4_TYPE_VHW_NET
Fiasco-UX Virtual devices, 179
L4_TYPE_VHW_NONE
Fiasco-UX Virtual devices, 179
L4_UTCB_BUF_REGS_OFFSET
x86 Virtual Registers (UTCB), 535
L4_UTCB_EXCEPTION_REGS_SIZE
x86 Virtual Registers (UTCB), 535
L4_UTCB_GENERIC_BUFFERS_SIZE
x86 Virtual Registers (UTCB), 535
L4_UTCB_GENERIC_DATA_SIZE
x86 Virtual Registers (UTCB), 535
L4_UTCB_INHERIT_FPU
x86 Virtual Registers (UTCB), 535
L4_UTCB_MSG_REGS_OFFSET
x86 Virtual Registers (UTCB), 535
L4_UTCB_OFFSET
x86 Virtual Registers (UTCB), 535
L4_UTCB_THREAD_REGS_OFFSET
x86 Virtual Registers (UTCB), 535
L4_VCON_ECHO
Virtual Console, 509
L4_VCON_GET_ATTR_OP
L4 IPC Opcodes, 283
L4_VCON_ICANON
Virtual Console, 509
L4_VCON_ICRNL
Virtual Console, 509
L4_VCON_IGNCR
Virtual Console, 509
L4_VCON_INLCR
Virtual Console, 509
L4_VCON_OCRNL
Virtual Console, 510
L4_VCON_ONLCR
Virtual Console, 510
L4_VCON_ONLRET
Virtual Console, 510
L4_VCON_READ_OP
L4 IPC Opcodes, 283
L4_VCON_READ_SIZE
Virtual Console, 510
L4_VCON_SET_ATTR_OP
L4 IPC Opcodes, 283
L4_VCON_WRITE_OP
L4 IPC Opcodes, 283
L4_VCON_WRITE_SIZE
Virtual Console, 510
L4_VCPU_F_DEBUG_EXC
vCPU API, 527
L4_VCPU_F_EXCEPTIONS
vCPU API, 527
L4_VCPU_F_FPU_ENABLED
vCPU API, 527
L4_VCPU_F_IRQ
vCPU API, 527
L4_VCPU_F_PAGE_FAULTS
vCPU API, 527
L4_VCPU_F_USER_MODE
vCPU API, 527
L4_VCPU_OFFSET_EXT_INFOS
vCPU API, 527
L4_VCPU_OFFSET_EXT_STATE
vCPU API, 527
L4_VCPU_SF_IRQ_PENDING
vCPU API, 527
L4_VM_VMX_BASIC_REG
VM API for VMX, 493
L4_VM_VMX_CR0_FIXED0_REG
VM API for VMX, 493
L4_VM_VMX_CR0_FIXED1_REG

VM API for VMX, 493
L4_VM_VMX_CR4_FIXED0_REG
VM API for VMX, 493
L4_VM_VMX_CR4_FIXED1_REG
VM API for VMX, 493
L4_VM_VMX_ENTRY_CTLS_DFL1_REG
VM API for VMX, 493
L4_VM_VMX_EPT_VPID_CAP_REG
VM API for VMX, 493
L4_VM_VMX_EXIT_CTLS_DFL1_REG
VM API for VMX, 493
L4_VM_VMX_MISC_REG
VM API for VMX, 493
L4_VM_VMX_NUM_CAPS_REGS
VM API for VMX, 493
L4_VM_VMX_NUM_DFL1_REGS
VM API for VMX, 493
L4_VM_VMX_PINBASED_CTLS_DFL1_REG
VM API for VMX, 493
L4_VM_VMX_PROCBASED_CTLS2_REG
VM API for VMX, 493
L4_VM_VMX_PROCBASED_CTLS_DFL1_REG
VM API for VMX, 493
L4_VM_VMX_TRUE_ENTRY_CTLS_REG
VM API for VMX, 493
L4_VM_VMX_TRUE_EXIT_CTLS_REG
VM API for VMX, 493
L4_VM_VMX_TRUE_PINBASED_CTLS_REG
VM API for VMX, 493
L4_VM_VMX_TRUE_PROCBASED_CTLS_REG
VM API for VMX, 493
L4_VM_VMX_VMCS_CR2
VM API for VMX, 492
L4_VM_VMX_VMCS_ENUM_REG
VM API for VMX, 493
L4_VM_VMX_VMCS_MSR_CSTAR
VM API for VMX, 492
L4_VM_VMX_VMCS_MSR_KERNEL_GS_BASE
VM API for VMX, 492
L4_VM_VMX_VMCS_MSR_LSTAR
VM API for VMX, 492
L4_VM_VMX_VMCS_MSR_STAR
VM API for VMX, 492
L4_VM_VMX_VMCS_MSR_SYSCALL_MASK
VM API for VMX, 492
L4_VM_VMX_VMCS_MSR_TSC_AUX
VM API for VMX, 492
L4_VM_VMX_VMCS_XCR0
VM API for VMX, 492
L4_WHOLE_ADDRESS_SPACE
Flex pages, 182
L4_WHOLE_IOADDRESS_SPACE
Flex pages, 182
l4_mem_type_archspecific
Memory descriptors (C version), 344
l4_mem_type_bootloader
Memory descriptors (C version), 344
l4_mem_type_conventional
Memory descriptors (C version), 344
l4_mem_type_dedicated
Memory descriptors (C version), 344
l4_mem_type_reserved
Memory descriptors (C version), 344
l4_mem_type_shared
Memory descriptors (C version), 344
l4_mem_type_undefined
Memory descriptors (C version), 344
L4IO_DEVICE_ANY
IO interface, 204
L4IO_DEVICE_INVALID
IO interface, 204
L4IO_DEVICE_OTHER
IO interface, 204
L4IO_DEVICE_PCI
IO interface, 204
L4IO_DEVICE_USB
IO interface, 204
L4IO_MEM_CACHED
IO interface, 204
L4IO_MEM_EAGER_MAP
IO interface, 204
L4IO_MEM_NONCACHED
IO interface, 204
L4IO_MEM_USE_MTRR
IO interface, 204
L4IO_MEM_USE_RESERVED_AREA
IO interface, 204
L4IO_RESOURCE_ANY
IO interface, 204
L4IO_RESOURCE_INVALID
IO interface, 204
L4IO_RESOURCE_IRQ
IO interface, 204
L4IO_RESOURCE_MEM
IO interface, 204
L4IO_RESOURCE_PORT
IO interface, 204
L4RE_DS_MAP_BUFFERABLE
Dataspace interface, 118
L4RE_DS_MAP_CACHEABLE
Dataspace interface, 118
L4RE_DS_MAP_CACHING_MASK
Dataspace interface, 118
L4RE_DS_MAP_CACHING_SHIFT
Dataspace interface, 118
L4RE_DS_MAP_NORMAL
Dataspace interface, 118
L4RE_DS_MAP_UNCACHEABLE
Dataspace interface, 118
L4RE_ELF_AUX_T_KIP_ADDR
L4Re ELF Auxiliary Information, 312
L4RE_ELF_AUX_T_NONE
L4Re ELF Auxiliary Information, 312
L4RE_ELF_AUX_T_STACK_ADDR
L4Re ELF Auxiliary Information, 312
L4RE_ELF_AUX_T_STACK_SIZE

L4Re ELF Auxiliary Information, 312
L4RE_ELF_AUX_T_VMA
L4Re ELF Auxiliary Information, 312
L4RE_RM_ATTACH_FLAGS
Region map interface, 402
L4RE_RM_EAGER_MAP
Region map interface, 402
L4RE_RM_IN_AREA
Region map interface, 402
L4RE_RM_NO_ALIAS
Region map interface, 402
L4RE_RM_OVERMAP
Region map interface, 402
L4RE_RM_PAGER
Region map interface, 402
L4RE_RM_READ_ONLY
Region map interface, 402
L4RE_RM_REGION_FLAGS
Region map interface, 402
L4RE_RM_RESERVED
Region map interface, 402
L4RE_RM_SEARCH_ADDR
Region map interface, 402
L4Re ELF Auxiliary Information
L4RE_ELF_AUX_T_KIP_ADDR, 312
L4RE_ELF_AUX_T_NONE, 312
L4RE_ELF_AUX_T_STACK_ADDR, 312
L4RE_ELF_AUX_T_STACK_SIZE, 312
L4RE_ELF_AUX_T_VMA, 312
L4Re::Dataspace
Map_bufferable, 1042
Map_cacheable, 1042
Map_caching_mask, 1042
Map_caching_shift, 1042
Map_normal, 1042
Map_ro, 1042
Map_rw, 1042
Map_uncacheable, 1042
L4Re::Dma_space
Bidirectional, 1055
Coherent, 1055
From_device, 1055
No_sync, 1054
None, 1055
Phys_space, 1055
To_device, 1055
L4Re::Inhibitor
Name_max, 1076
L4Re::Mem_alloc
Continuous, 1083
Pinned, 1083
Super_pages, 1083
L4Re::Util::Event_t
Mode_irq, 1133
Mode_polling, 1133
L4Re::Video::Goos
F_auto_refresh, 1196
F_dynamic_buffers, 1196
F_dynamic_views, 1196
F_pointer, 1196
L4Re::Video::View
F_above, 1208
F_dyn_allocated, 1208
F_flags_mask, 1208
F_fully_dynamic, 1208
F_none, 1208
F_set_background, 1208
F_set_buffer, 1208
F_set_buffer_offset, 1208
F_set_bytes_per_line, 1208
F_set_flags, 1208
F_set_pixel, 1208
F_set_position, 1208
L4SIGMA0_IPCERROR
Sigma0 API, 423
L4SIGMA0_NOFPAGE
Sigma0 API, 423
L4SIGMA0_NOTALIGNED
Sigma0 API, 423
L4SIGMA0_OK
Sigma0 API, 423
L4SIGMA0_SMALLERFPAGE
Sigma0 API, 423
L4VBUS_DMAD_BIND
L4 V-BUS functions, 285
L4VBUS_DMAD_KERNEL_DMA_SPACE
L4 V-BUS functions, 285
L4VBUS_DMAD_L4RE_DMA_SPACE
L4 V-BUS functions, 285
L4VBUS_DMAD_UNBIND
L4 V-BUS functions, 285
L4VBUS_GPIO_PIN_PULL_DOWN
L4vbus GPIO functions, 317
L4VBUS_GPIO_PIN_PULL_NONE
L4vbus GPIO functions, 317
L4VBUS_GPIO_PIN_PULL_UP
L4vbus GPIO functions, 317
L4VBUS_GPIO_SETUP_INPUT
L4vbus GPIO functions, 317
L4VBUS_GPIO_SETUP_IRQ
L4vbus GPIO functions, 317
L4VBUS_GPIO_SETUP_OUTPUT
L4vbus GPIO functions, 317
L4VIRTIO_BLOCK_S_IOERR
L4 VIRTIO Block Device, 293
L4VIRTIO_BLOCK_S_OK
L4 VIRTIO Block Device, 293
L4VIRTIO_BLOCK_S_UNSUPP
L4 VIRTIO Block Device, 293
L4VIRTIO_BLOCK_T_FLUSH
L4 VIRTIO Block Device, 292
L4VIRTIO_BLOCK_T_IN
L4 VIRTIO Block Device, 292
L4VIRTIO_BLOCK_T_OUT
L4 VIRTIO Block Device, 292
L4VIRTIO_ID_9P

L4 VIRTIO Transport Layer, 298
L4VIRTIO_ID_BALLOON
L4 VIRTIO Transport Layer, 298
L4VIRTIO_ID_BLOCK
L4 VIRTIO Transport Layer, 298
L4VIRTIO_ID_CAIF
L4 VIRTIO Transport Layer, 298
L4VIRTIO_ID_CONSOLE
L4 VIRTIO Transport Layer, 298
L4VIRTIO_ID_NET
L4 VIRTIO Transport Layer, 298
L4VIRTIO_ID_RNG
L4 VIRTIO Transport Layer, 298
L4VIRTIO_ID_RPMSG
L4 VIRTIO Transport Layer, 298
L4VIRTIO_ID_RPROC_SERIAL
L4 VIRTIO Transport Layer, 298
L4VIRTIO_ID_SCSI
L4 VIRTIO Transport Layer, 298
L4VIRTIO_ID_SOCK
L4 VIRTIO Transport Layer, 298
L4VIRTIO_IRQ_STATUS_CONFIG
L4 VIRTIO Transport Layer, 297
L4VIRTIO_IRQ_STATUS_VRING
L4 VIRTIO Transport Layer, 297
L4VIRTIO_OP_CONFIG_QUEUE
L4 VIRTIO Transport Layer, 297
L4VIRTIO_OP_REGISTER_DS
L4 VIRTIO Transport Layer, 298
L4VIRTIO_OP_REGISTER_IFACE
L4 VIRTIO Transport Layer, 297
L4VIRTIO_OP_SET_STATUS
L4 VIRTIO Transport Layer, 297
L4VIRTIO_STATUS_ACKNOWLEDGE
L4 VIRTIO Transport Layer, 298
L4VIRTIO_STATUS_DRIVER
L4 VIRTIO Transport Layer, 298
L4VIRTIO_STATUS_DRIVER_OK
L4 VIRTIO Transport Layer, 298
L4VIRTIO_STATUS_FAILED
L4 VIRTIO Transport Layer, 298
L4VIRTIO_STATUS_FEATURES_OK
L4 VIRTIO Transport Layer, 298
L4vbus GPIO functions
L4VBUS_GPIO_PIN_PULL_DOWN, 317
L4VBUS_GPIO_PIN_PULL_NONE, 317
L4VBUS_GPIO_PIN_PULL_UP, 317
L4VBUS_GPIO_SETUP_INPUT, 317
L4VBUS_GPIO_SETUP_IRQ, 317
L4VBUS_GPIO_SETUP_OUTPUT, 317
L4virtio, 563
L4virtio::Ptr
Invalid, 1294
Libedid_block_size
EDID parsing functionality, 122
Link
Name-space API, 371
Log interface, 331
Logging interface, 334

Machine Restarting Function, 336
Map_bufferable
L4Re::Dataspace, 1042
Map_cacheable
L4Re::Dataspace, 1042
Map_caching_mask
L4Re::Dataspace, 1042
Map_caching_shift
L4Re::Dataspace, 1042
Map_normal
L4Re::Dataspace, 1042
Map_ro
L4Re::Dataspace, 1042
Map_rw
L4Re::Dataspace, 1042
Map_uncacheable
L4Re::Dataspace, 1042
max
Small C++ Template Library, 431
max_free_slabs
Small C++ Template Library, 430, 431
Mem
L4 kernel object type information, 302
Memory allocator, 337
Memory descriptors (C version)
l4_mem_type_archspecific, 344
l4_mem_type_bootloader, 344
l4_mem_type_conventional, 344
l4_mem_type_dedicated, 344
l4_mem_type_reserved, 344
l4_mem_type_shared, 344
l4_mem_type_undefined, 344
Memory operations., 347
L4_MEM_WIDTH_1BYTE, 347
L4_MEM_WIDTH_2BYTE, 347
L4_MEM_WIDTH_4BYTE, 347
Memory related, 350
L4_INVALID_ADDR, 352
Message Items, 355
L4_ITEM_CONT, 355
L4_ITEM_MAP, 355
L4_MAP_ITEM_GRANT, 355
L4_MAP_ITEM_MAP, 355
L4_RCV_ITEM_LOCAL_ID, 356
L4_RCV_ITEM_SINGLE_CAP, 356
Message Tag, 359
L4_MSGTAG_ERROR, 360
L4_MSGTAG_FLAGS, 361
L4_MSGTAG_PROPAGATE, 361
L4_MSGTAG_SCHEDULE, 361
L4_MSGTAG_TRANSFER_FPU, 361
L4_MSGTAG_XCPU, 360
L4_PROTO_ALLOW_SYSCALL, 361
L4_PROTO_DEBUGGER, 361
L4_PROTO_DMA_SPACE, 361
L4_PROTO_EXCEPTION, 361
L4_PROTO_FACTORY, 361

L4_PROTO_IO_PAGE_FAULT, 361
L4_PROTO_IOMMU, 361
L4_PROTO_IRQ, 361
L4_PROTO_IRQ_MUX, 361
L4_PROTO_IRQ_SENDER, 361
L4_PROTO_KOBJECT, 361
L4_PROTO_LOG, 361
L4_PROTO_META, 361
L4_PROTO_NONE, 361
L4_PROTO_PAGE_FAULT, 361
L4_PROTO_PF_EXCEPTION, 361
L4_PROTO_PLATFORM_CTL, 362
L4_PROTO_PREEMPTION, 361
L4_PROTO_SCHEDULER, 361
L4_PROTO_SEMAPHORE, 361
L4_PROTO_SIGMA0, 361
L4_PROTO_SYS_EXCEPTION, 361
L4_PROTO_TASK, 361
L4_PROTO_THREAD, 361
L4_PROTO_VM, 361
min
Small C++ Template Library, 431
Mode_irq
L4Re::Util::Event_t, 1133
Mode_polling
L4Re::Util::Event_t, 1133
Mr_bytes
L4::Ipc::Msg, 554
Mr_words
L4::Ipc::Msg, 554

N
cxx::Bits::Direction, 635
Name-space API
Link, 371
Overwrite, 371
Partly_resolved, 370
Ro, 371
Rs, 371
Rw, 371
Rws, 371
Strong, 371
Trusted, 371
Name_max
L4Re::Inhibitor, 1076
Namespace interface, 372
No_sync
L4Re::Dma_space, 1054
None
L4::Types::Flags, 979
L4Re::Dma_space, 1055

Object Invocation, 375
L4_SYSF_CALL, 377
L4_SYSF_NONE, 377
L4_SYSF_OPEN_WAIT, 377
L4_SYSF_RECV, 377
L4_SYSF_REPLY, 377
L4_SYSF_REPLY_AND_WAIT, 377
L4_SYSF_SEND, 377
L4_SYSF_SEND_AND_WAIT, 377
L4_SYSF_WAIT, 377
object_size
Small C++ Template Library, 430, 431
objects_per_slab
Small C++ Template Library, 430, 431
operator new
Small C++ Template Library, 432
outchar
Kernel Debugger, 268
outdec
Kernel Debugger, 269
outhex12
Kernel Debugger, 269
outhex16
Kernel Debugger, 269
outhex20
Kernel Debugger, 269
outhex32
Kernel Debugger, 269
outhex8
Kernel Debugger, 269
outnstring
Kernel Debugger, 270
outstring
Kernel Debugger, 270
Overwrite
Name-space API, 371

PROTO_ANY
L4, 542
PROTO_EMPTY
L4, 542
Pager
Region map API, 400
Partly_resolved
Name-space API, 370
Phys_space
L4Re::Dma_space, 1055
Pinned
L4Re::Mem_alloc, 1083
Ports
L4 kernel object type information, 302
Producer, 393, 395

R
cxx::Bits::Direction, 635
Random number support, 396
Read_only
Region map API, 400
Region map API
Attach_flags, 399
Cache_buffered, 400
Cache_normal, 400
Cache_uncached, 400
Caching, 400
Caching_ds_shift, 400
Caching_shift, 400

Detach_again, 399
Detach_exact, 399
Detach_free, 400
Detach_keep, 399
Detach_overlap, 399
Detached_ds, 399
Eager_map, 399
In_area, 399
Kept_ds, 399
Pager, 400
Read_only, 400
Region_flags, 400
Reserved, 400
Search_addr, 399
Split_ds, 399
Region map interface, 401
L4RE_RM_ATTACH_FLAGS, 402
L4RE_RM_EAGER_MAP, 402
L4RE_RM_IN_AREA, 402
L4RE_RM_NO_ALIAS, 402
L4RE_RM_OVERMAP, 402
L4RE_RM_PAGER, 402
L4RE_RM_READ_ONLY, 402
L4RE_RM_REGION_FLAGS, 402
L4RE_RM_RESERVED, 402
L4RE_RM_SEARCH_ADDR, 402
Region_flags
Region map API, 400
Reply_compound
Server-Side IPC framework, 416
Reply_separate
Server-Side IPC framework, 416
Reserved
Region map API, 400
Rights_mask
L4::Ipc::Cap, 764
Ro
Name-space API, 371
Rs
Name-space API, 371
Rw
Name-space API, 371
Rws
Name-space API, 371

Scheduler, 411
info, 412
L4_SCHEDULER_IDLE_TIME_OP, 412
L4_SCHEDULER_INFO_OP, 412
L4_SCHEDULER_RUN_THREAD_OP, 412
Search_addr
Region map API, 399
Server-Side IPC framework
Reply_compound, 416
Reply_separate, 416
Shared Memory Library, 417
Shutdown
L4::Platform_control, 895
Sigma0 API
L4SIGMA0_IPCERROR, 423
L4SIGMA0_NOFPAGE, 423
L4SIGMA0_NOTALIGNED, 423
L4SIGMA0_OK, 423
L4SIGMA0_SMALLERFPAGE, 423
Signals, 426
slab_size
Small C++ Template Library, 430, 431
Small C++ Template Library, 429
C_bits, 430
max, 431
max_free_slabs, 430, 431
min, 431
object_size, 430, 431
objects_per_slab, 430, 431
operator new, 432
slab_size, 430, 431
W_bits, 430
Split_ds
Region map API, 399
Strong
Name-space API, 371
Super_pages
L4Re::Mem_alloc, 1083
Suspend
L4::Platform_control, 895

Task, 433
L4_FP_ALL_SPACES, 434
L4_FP_DELETE_OBJ, 434
L4_FP_OTHER_SPACES, 434
Thread, 442
L4_THREAD_CONTROL_ALIEN, 444
L4_THREAD_CONTROL_BIND_TASK, 444
L4_THREAD_CONTROL_MR_IDX_BIND_TASK, 444
L4_THREAD_CONTROL_MR_IDX_BIND_UTCB, 444
L4_THREAD_CONTROL_MR_IDX_EXC_HANDLER, 444
L4_THREAD_CONTROL_MR_IDX_FLAG_VALS, 444
L4_THREAD_CONTROL_MR_IDX_FLAGS, 444
L4_THREAD_CONTROL_MR_IDX_PAGER, 444
L4_THREAD_CONTROL_SET_EXC_HANDLER, 444
L4_THREAD_CONTROL_SET_PAGER, 444
L4_THREAD_CONTROL_UX_NATIVE, 444
L4_THREAD_EX_REGS_CANCEL, 444
L4_THREAD_EX_REGS_TRIGGER_EXCEPTION, 444
Thread control, 460
Timeouts, 468
Timestamp Counter, 476
To_device
L4Re::Dma_space, 1055
Trusted
Name-space API, 371

Utility Functions, 483

vCPU API
L4_VCPU_F_DEBUG_EXC, 527
L4_VCPU_F_EXCEPTIONS, 527
L4_VCPU_F_FPU_ENABLED, 527
L4_VCPU_F_IRQ, 527
L4_VCPU_F_PAGE_FAULTS, 527
L4_VCPU_F_USER_MODE, 527
L4_VCPU_OFFSET_EXT_INFOS, 527
L4_VCPU_OFFSET_EXT_STATE, 527
L4_VCPU_SF_IRQ_PENDING, 527
VM API for VMX
L4_VM_VMX_BASIC_REG, 493
L4_VM_VMX_CR0_FIXED0_REG, 493
L4_VM_VMX_CR0_FIXED1_REG, 493
L4_VM_VMX_CR4_FIXED0_REG, 493
L4_VM_VMX_CR4_FIXED1_REG, 493
L4_VM_VMX_ENTRY_CTLS_DFL1_REG, 493
L4_VM_VMX_EPT_VPID_CAP_REG, 493
L4_VM_VMX_EXIT_CTLS_DFL1_REG, 493
L4_VM_VMX_MISC_REG, 493
L4_VM_VMX_NUM_CAPS_REGS, 493
L4_VM_VMX_NUM_DFL1_REGS, 493
L4_VM_VMX_PINBASED_CTLS_DFL1_REG, 493
L4_VM_VMX_PROCBASED_CTLS2_REG, 493
L4_VM_VMX_PROCBASED_CTLS_DFL1_REG, 493
L4_VM_VMX_TRUE_ENTRY_CTLS_REG, 493
L4_VM_VMX_TRUE_EXIT_CTLS_REG, 493
L4_VM_VMX_TRUE_PINBASED_CTLS_REG, 493
L4_VM_VMX_TRUE_PROCBASED_CTLS_REG, 493
L4_VM_VMX_VMCS_CR2, 492
L4_VM_VMX_VMCS_ENUM_REG, 493
L4_VM_VMX_VMCS_MSR_CSTAR, 492
L4_VM_VMX_VMCS_MSR_KERNEL_GS_BASE, 492
L4_VM_VMX_VMCS_MSR_LSTAR, 492
L4_VM_VMX_VMCS_MSR_STAR, 492
L4_VM_VMX_VMCS_MSR_SYSCALL_MASK, 492
L4_VM_VMX_VMCS_MSR_TSC_AUX, 492
L4_VM_VMX_VMCS_XCR0, 492
Video API
F_l4re_video_goos_auto_refresh, 503
F_l4re_video_goos_dynamic_buffers, 504
F_l4re_video_goos_dynamic_views, 504
F_l4re_video_goos_pointer, 503
F_l4re_video_view_above, 504
F_l4re_video_view_dyn_allocated, 504
F_l4re_video_view_flags_mask, 504
F_l4re_video_view_none, 504
F_l4re_video_view_set_background, 504
F_l4re_video_view_set_buffer, 504
F_l4re_video_view_set_buffer_offset, 504
F_l4re_video_view_set_bytes_per_line, 504
F_l4re_video_view_set_flags, 504
F_l4re_video_view_set_pixel, 504
F_l4re_video_view_set_position, 504
Virtual Console, 508
L4_VCON_ECHO, 509
L4_VCON_ICANON, 509
L4_VCON_ICRNL, 509
L4_VCON_IGNCR, 509
L4_VCON_INLCR, 509
L4_VCON_OCRNL, 510
L4_VCON_ONLCR, 510
L4_VCON_ONLRET, 510
L4_VCON_READ_SIZE, 510
L4_VCON_WRITE_SIZE, 510
Virtual Machines, 520

W_bits
Small C++ Template Library, 430
Word_bytes
L4::Ipc::Msg, 554

x86 Virtual Registers (UTCB)
L4_UTCB_BUF_REGS_OFFSET, 535
L4_UTCB_EXCEPTION_REGS_SIZE, 535
L4_UTCB_GENERIC_BUFFERS_SIZE, 535
L4_UTCB_GENERIC_DATA_SIZE, 535
L4_UTCB_INHERIT_FPU, 535
L4_UTCB_MSG_REGS_OFFSET, 535
L4_UTCB_OFFSET, 535
L4_UTCB_THREAD_REGS_OFFSET, 535