L4Re – L4 Runtime Environment

Generated by Doxygen 1.8.8

Contents

Chapter 1

Fiasco.OC & L4 Runtime Environment (L4Re)

1.1 Preface

The intention of this document is to provide a birds eye overview about L4Re and about the environment in which typical applications and servers run. We highlight here the principled functionality of the servers in the environment but do not discuss their specific interfaces. Detailed documentation about these interface is available in the modules section.

The document is meant as a general overview repeating many design concepts of L4-based systems and capability systems in general. We do though assume familiarity with C++ and an idea on the general concepts and terms of L4: threads — as an abstraction for execution —, tasks — holding the capabilities to kernel objects that are accessible by the threads executing in this task —, and IPC over IPC-gates to send messages and to transfer capabilities between tasks.

1.2 General 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.

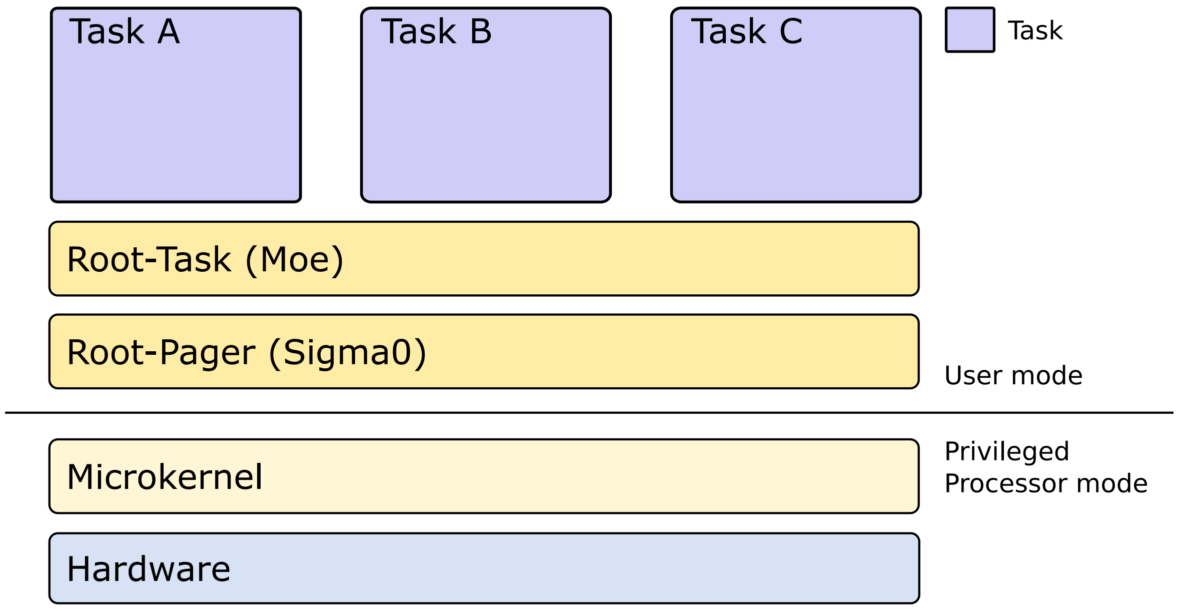


Figure 1.1: Basic Structure of an L4Re based system

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.

In the following sections, we discuss the basic concepts of our microkernel and its runtime environment in more depth.

1.3 The Fiasco.OC Microkernel

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.

1.3.1 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.

1.3.2 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.

1.4 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.

1.5 Introduction to L4Re's concepts

This section introduces basic concepts used by L4Re. Understanding of these concepts is a fundamental requirement to understand the inner workings of L4Re's software components and can dramatically help developers in efficiently developing L4Re-based software.

1.6 Memory management - Data Spaces and the Region Map

1.6.1 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.

1.6.2 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.

1.6.3 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.

1.6.4 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

api_l4re_dataspace and api_l4re_rm.

1.7 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.

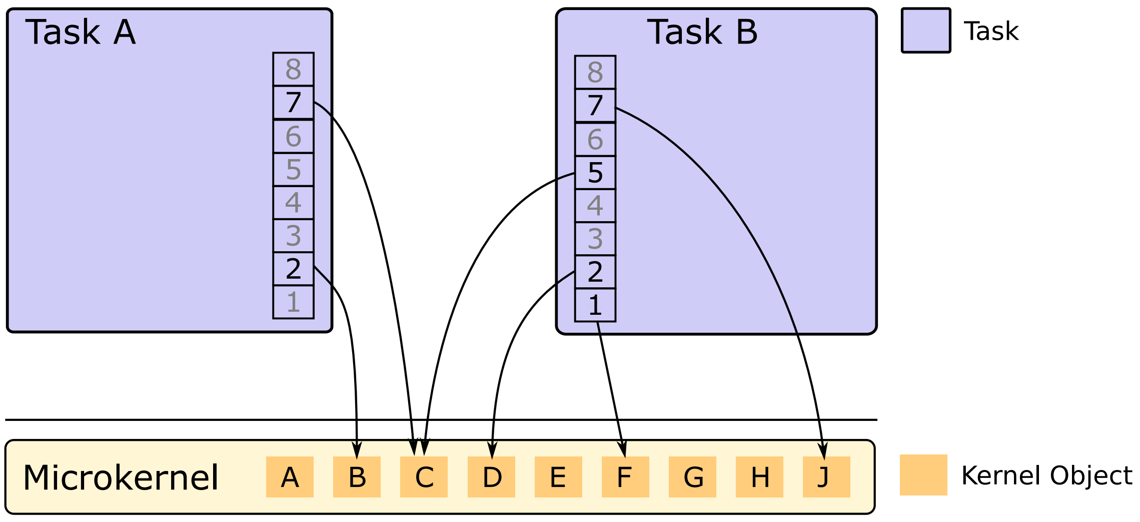


Figure 1.2: Capabilities and Local Naming in L4

So how does an application get access to 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.

1.8 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.

1.8.1 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");
```

1.8.2 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.

1.9 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

1.10 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

> api_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

1.11 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.

1.11.1 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.

1.11.2 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.

Chapter 2

Getting Started

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.

```
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.

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 3

L4Re Servers

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

3.1 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.

3.2 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.

3.3 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.

3.4 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.

3.5 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.

3.6 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.

3.7 Rtc, the Real-Time Clock Server

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

3.8 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 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 (L4Re::Namespace) — 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 (L4Re::Log) — provides tagged log output for applications
- l4re_moe_scheduler (L4::Scheduler) — provides simple scheduler objects for scheduling policy enforcement
- Memory Allocator, Generic Factory (L4Re::Mem_alloc, L4::Factory) — provides allocation of physical RAM as data spaces, as well as allocation of the other L4Re objects provided by Moe

3.8.1 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 gerneic factory interface alllows allocation of the following objects:

- L4Re::Namespace
- L4Re::Dataspace, RAM allocation
- L4Re::Rm, Virtual mamory 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). The granularity for memory allocation is the machine page size (L4_PAGESIZE).

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)

3.8.2 Name-Space Provider

Moe provides a name spaces conforming to the L4Re::Namespace interface (see api_l4re_namespace). 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.

3.8.3 Boot FS

The Boot FS subsystem provides read only access to the files loaded during the platform boot (or available in ROM). This 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.

3.8.4 Log Subsystem

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

3.8.5 Command-Line Options

Moe command-line syntax is:

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

3.8.5.1 --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).

3.8.5.2 --init=<init process>

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

Note

command-line options to the init process are given after the – special option.

3.8.5.3 --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 sam as for --debug=.

3.8.5.4 --ldr-flags=<loader flags>

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

3.9 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.

3.9.1 Lua Bindings for L4Re

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

3.9.1.1 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.

3.9.1.2 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.

3.9.1.3 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
}
```

3.9.1.4 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.

3.10 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.

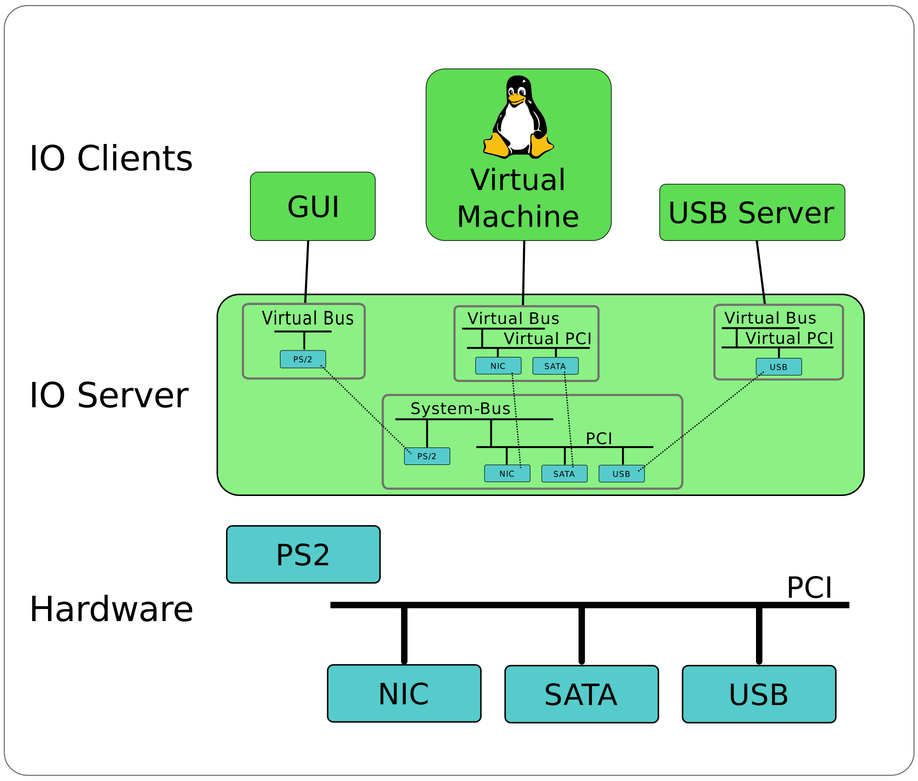


Figure 3.1: IO Service Architecture Overview

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()

  FOODEVICE = Io.Hw.Device(function()
    hid  = "FOODEVICE";
    compatible = {"dev-foo,mmio", "dev-foo"};
    -- note: names for resources are truncated to 4 letters
    int  = Io.Res.irq(17);
    regs = Io.Res.mmio(0x6f000000, 0x6f007fff);
  end);

  BARDEVICE = Io.Hw.Device(function()
    hid  = "BARDEVICE";
    compatible = {"dev-bar,mmio", "dev-bar"};
    -- note: names for resources are truncated to 4 letters
    intA = Io.Res.irq(19);
    intB = Io.Res.irq(20);
    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.

Chapter 4

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_getl4cap(pthread_t t)` to get the capability index of the pthread t.

  For example:

  ```
  pthread_getl4cap(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);
  ```

Chapter 5

Module Index

5.1 Modules

Here is a list of all modules:

Chapter 6

Namespace Index

6.1 Namespace List

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

Chapter 7

Hierarchical Index

7.1 Class Hierarchy

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

Chapter 8

Data Structure Index

8.1 Data Structures

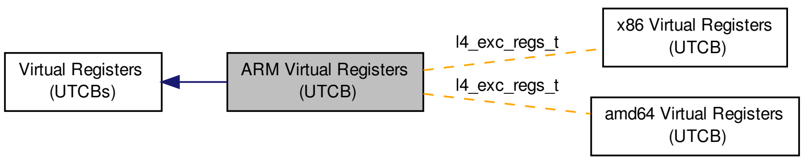
Here are the data structures with brief descriptions:

Chapter 9

Module Documentation

9.1 ARM Virtual Registers (UTCB)

Collaboration diagram for ARM Virtual Registers (UTCB):


Virtual Registers (UTCBs)
ARM Virtual Registers (UTCB)
l4_exc_regs_t
l4_exc_regs_t
x86 Virtual Registers (UTCB)
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

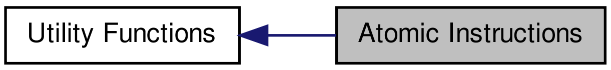
- enum L4_utcb_consts_arm

  UTCB constants for ARM.

9.1.1 Detailed Description

9.2 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)

9.2.1 Detailed Description

9.2.2 Function Documentation

9.2.2.1 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.

9.2.2.2 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.

9.2.2.3 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.

9.2.2.4 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.

9.2.2.5 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

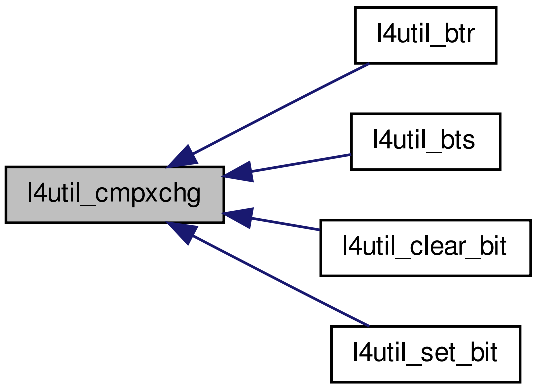
0 if comparison failed, 1 otherwise

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:



9.2.2.6 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.

9.2.2.7 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.

9.2.2.8 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.

9.2.2.9 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.

9.2.2.10 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.

9.2.2.11 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.

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

Parameters

dest	destination operand

Definition at line 311 of file atomic.h.

9.2.2.13 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.

9.2.2.14 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.

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

Atomic increment.

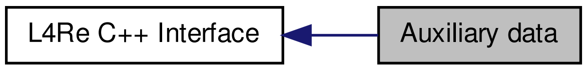
Parameters

dest	destination operand

Definition at line 61 of file atomic_arch.h.

9.3 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

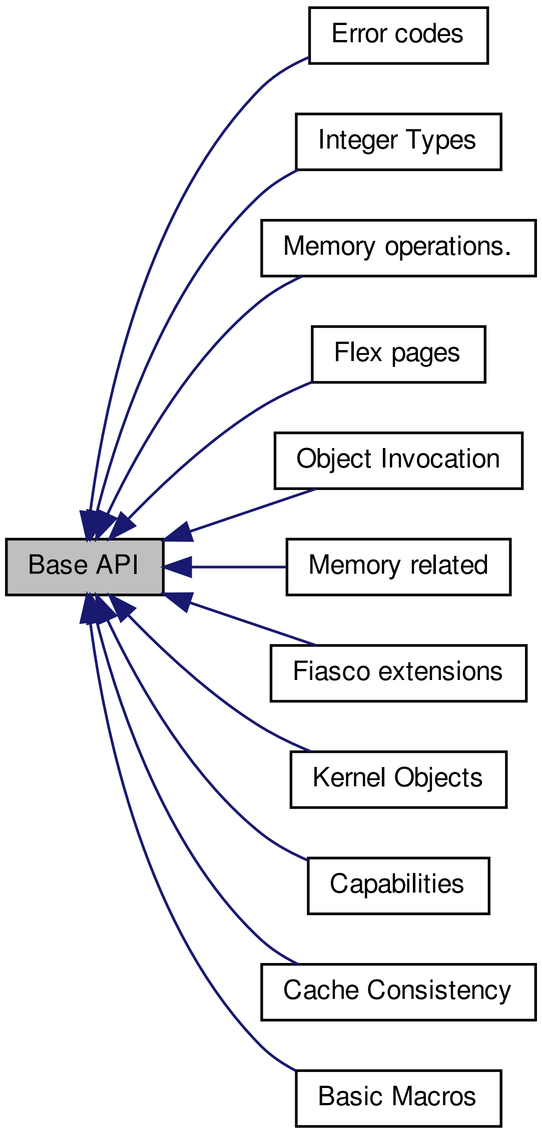
  Flags for program loading.

9.3.1 Detailed Description

9.4 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.
- Cache Consistency

  Various functions for cache consistency.
- Capabilities

Functions and definitions related to 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 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.h

  Interrupt controller.
- file ipc.h

  Common IPC interface.
- file irq.h

  Interrupt functionality.
- file kip.h

  Kernel Info Page access functions.
- file memdesc.h

  Memory description functions.
- 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.

9.4.1 Detailed Description

Interfaces for all kinds of base functionality.

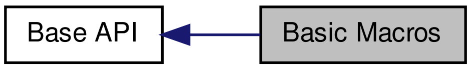
Some notes on Inter Process Communication (IPC)

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.

9.5 Basic Macros

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

Collaboration diagram for Basic Macros:



Macros

- #define L4_DECLARE_CONSTRUCTOR(func, prio)

  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_EXPORT

  Attribute to mark functions, variables, and data types as being exported from a library.
- #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.

9.5.1 Detailed Description

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

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

9.5.2 Macro Definition Documentation

9.5.2.1 #define L4_DECLARE_CONSTRUCTOR(*func, prio*)

L4 Inline function attribute.

Handcoded version of **attribute**((constructor(xx))).

Parameters

func	function declaration (prototype)
prio	the prio must be 65535 - *gcc_prio*

Definition at line 84 of file compiler.h.

9.5.2.2 #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 202 of file compiler.h.

9.5.2.3 #define L4_EXPORT

Attribute to mark functions, variables, and data types as being exported from a library.

All data types, functions, and global variables that shall be exported from a library shall be marked with this attribute. The default may become to hide everything that is not marked as L4_EXPORT from the users of a library and provide the possibility for aggressive optimization of all those internal functionality of a library.

Usage:

```
class L4_EXPORT My_class
{
  ...
};

int L4_EXPORT function(void);

int L4_EXPORT global_data; // global data is not recommended
```

Definition at line 232 of file compiler.h.

9.5.2.4 #define L4_HIDDEN

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

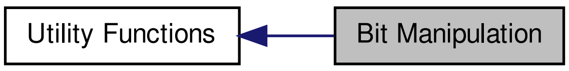
This attribute is intended for functions, data, and data types that shall never be visible outside of a library. In particular, for shared libraries this may result in much faster code within the library and short linking times.

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

int L4_HIDDEN function(void);

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

Definition at line 229 of file compiler.h.

9.6 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.

9.6.1 Detailed Description

9.6.2 Function Documentation

9.6.2.1 void l4util_set_bit (int *b*, volatile **l4_umword_t** ∗ *dest*) `[inline]`

Set bit in memory.

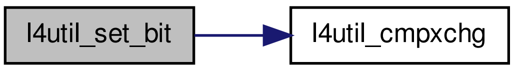
Parameters

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:



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

Clear bit in memory.

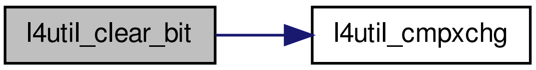
Parameters

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:



9.6.2.3 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.

9.6.2.4 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.

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

Bit test and set.

Parameters

b	bit position
dest	destination operand

Returns

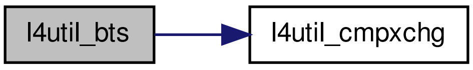
Old value of bit *b*.

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:



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

Bit test and reset.

Parameters

b	bit position
dest	destination operand

Returns

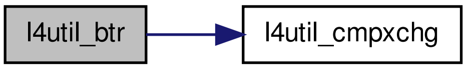
Old value of bit *b*.

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:



9.6.2.7 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.

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

Bit scan reverse.

Parameters

word	value (machine size)

Returns

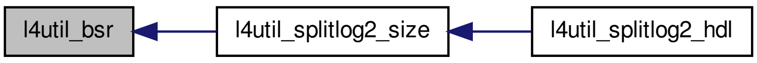
index of most significant set bit in word, -1 if no bit is set (word == 0)

"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:



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

Bit scan forward.

Parameters

word	value (machine size)

Returns

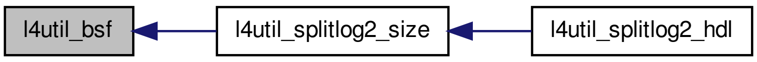
index of least significant bit set in word, -1 if no bit is set (word == 0)

"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:



9.6.2.10 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.

9.6.2.11 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.

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

Find the next power of 2 for a given number.

Parameters

val	initial value

Returns

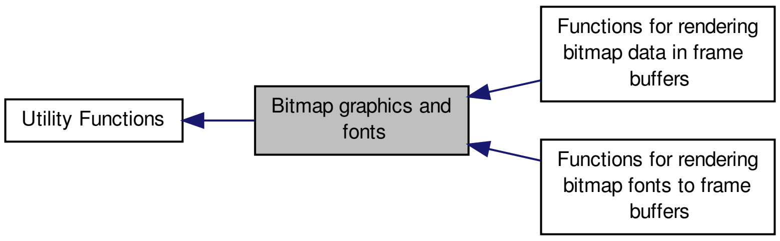
next-highest power of 2

Definition at line 408 of file bitops.h.

9.7 Bitmap graphics and fonts

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

Collaboration diagram for Bitmap graphics and fonts:


Utility Functions
Bitmap graphics and fonts
Functions for rendering bitmap data in frame buffers
Functions for rendering bitmap fonts to frame buffers

Modules

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

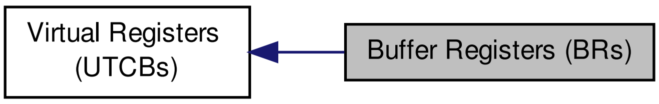
9.7.1 Detailed Description

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.

9.8 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.

9.8.1 Detailed Description

9.8.2 Enumeration Type Documentation

9.8.2.1 enum l4_buffer_desc_consts_t

Constants for buffer descriptors.

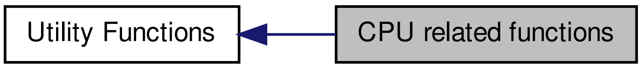
Enumerator

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 225 of file consts.h.

9.9 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.

9.9.1 Detailed Description

9.9.2 Function Documentation

9.9.2.1 int l4util_cpu_has_cpuid (void) [inline]

Check whether the CPU supports the "cpuid" instruction.

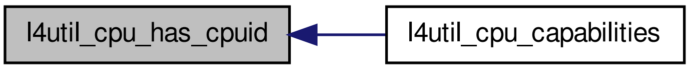
Returns

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:



9.9.2.2 unsigned int l4util_cpu_capabilities (void) [inline]

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

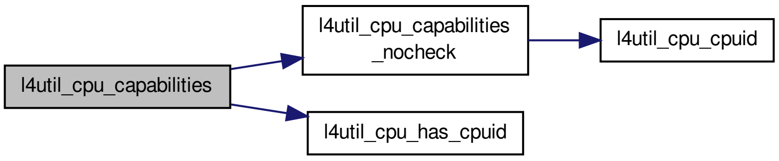
Returns

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:



9.9.2.3 unsigned int l4util_cpu_capabilities_nocheck (void) [inline]

Returns the CPU capabilities.

Returns

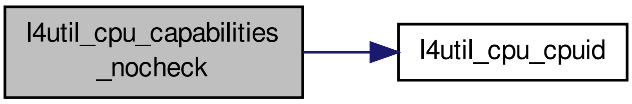
CPU capabilities.

Definition at line 86 of file cpu.h.

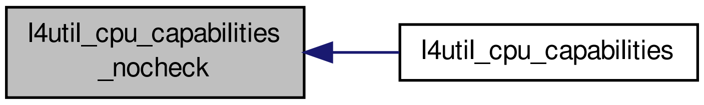
References l4util_cpu_cpuid().

Referenced by l4util_cpu_capabilities().

Here is the call graph for this function:



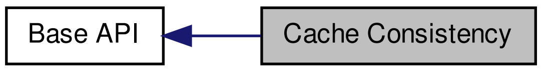
Here is the caller graph for this function:



9.10 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.

9.10.1 Detailed Description

Various functions for cache consistency.

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

9.10.2 Function Documentation

9.10.2.1 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.

9.10.2.2 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.

9.10.2.3 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.

9.10.2.4 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.

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

Make memory coherent for use with external memory.

Parameters

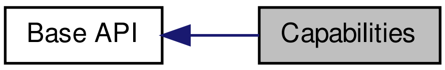
start	Start of range (inclusive)
end	End of range (exclusive)

Definition at line 112 of file cache.h.

9.11 Capabilities

Functions and definitions related to 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 }

    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.

9.11.1 Detailed Description

Functions and definitions related to capabilities.

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

C interface for capabilities:
```
#include <l4/sys/types.h>
```

9.11.2 Typedef Documentation

9.11.2.1 typedef unsigned long l4_cap_idx_t

L4 Capability selector Type.

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

Definition at line 319 of file types.h.

9.11.3 Enumeration Type Documentation

9.11.3.1 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 134 of file consts.h.

9.11.3.2 enum l4_default_caps_t

Default capabilities setup for the initial tasks.

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

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 api_l4re_env for this kind of information.

See also

api_l4re_env 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.

Definition at line 248 of file consts.h.

9.11.4 Function Documentation

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

Test if a capability selector is the invalid capability.

Parameters

c	Capability selector

Returns

Boolean value

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 350 of file types.h.

9.11.4.2 unsigned l4_is_valid_cap (**l4_cap_idx_t** *c*) `[inline]`

Test if a capability selector is a valid selector.

Parameters

c	Capability selector

Returns

Boolean value

Definition at line 354 of file types.h.

9.11.4.3 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

Returns

1 if equal, 0 if not equal

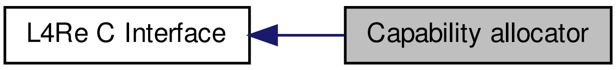
Definition at line 358 of file types.h.

References L4_CAP_SHIFT.

9.12 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.

9.12.1 Detailed Description

Capability allocator C interface.

9.12.2 Function Documentation

9.12.2.1 long l4re_util_cap_last (void)

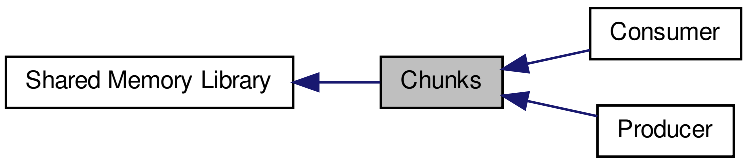
Return last capability index the allocator can return.

Returns

last/biggest capability index the allocator can return

9.13 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.

9.13.1 Detailed Description

9.13.2 Function Documentation

9.13.2.1 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.

Return values

chunk	Chunk structure to fill in.

Returns

0 on success, <0 on error

Examples:

examples/libs/shmc/prodcons.c.

9.13.2.2 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.

Return values

chunk	Chunk data structure to fill.

Returns

0 on success, <0 on error

Examples:

examples/libs/shmc/prodcons.c.

9.13.2.3 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.

Return values

chunk	chunk data structure to fill.

Returns

0 on success, <0 on error

9.13.2.4 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

Returns

<0 on error, 0 if no more chunks, >0 iterator value for next call

9.13.2.5 void∗ l4shmc_chunk_ptr (l4shmc_chunk_t ∗ *chunk*) `[inline]`

Get data pointer to chunk.

Parameters

chunk	Chunk.

Returns

0 on success, <0 on error

Examples:

examples/libs/shmc/prodcons.c.

9.13.2.6 long l4shmc_chunk_capacity (l4shmc_chunk_t ∗ *chunk*) `[inline]`

Get capacity of a chunk.

Parameters

chunk	Chunk.

Returns

0 on success, <0 on error

9.13.2.7 l4shmc_signal_t∗ l4shmc_chunk_signal (l4shmc_chunk_t ∗ *chunk*) `[inline]`

Get the signal of a chunk.

Parameters

chunk	Chunk.

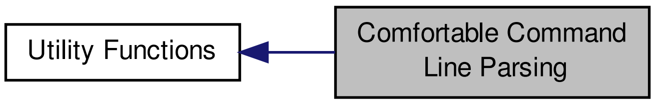
Returns

0 if no signal has been register with this chunk, signal otherwise

9.14 Client/Server IPC Framework

9.15 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.

9.15.1 Detailed Description

9.15.2 Function Documentation

9.15.2.1 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.

`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

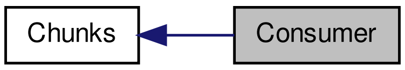
0 if the command-line was successfully parsed, otherwise:

- -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.

9.16 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.

9.16.1 Detailed Description

9.16.2 Function Documentation

9.16.2.1 long l4shmc_enable_chunk (l4shmc_chunk_t ∗ *chunk*)

Enable a signal connected with a chunk.

Parameters

chunk	Chunk to enable.

Returns

0 on success, <0 on 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.

9.16.2.2 long l4shmc_wait_chunk (l4shmc_chunk_t ∗ *chunk*) [inline]

Wait on a specific chunk.

Parameters

chunk	Chunk to wait for.

Returns

0 on success, <0 on error

Examples:

examples/libs/shmc/prodcons.c.

9.16.2.3 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.

Returns

0 on success, <0 on 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.

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

Check whether a specific chunk has an event pending.

Parameters

chunk	Chunk to check.

Returns

0 on success, <0 on 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.

9.16.2.5 long l4shmc_chunk_consumed (l4shmc_chunk_t * *chunk*) [inline]

Mark a chunk as free.

Parameters

chunk	Chunk to mark as free.

Returns

0 on success, <0 on error

Examples:

examples/libs/shmc/prodcons.c.

9.16.2.6 long l4shmc_is_chunk_ready (l4shmc_chunk_t ∗ *chunk*) `[inline]`

Check whether data is available.

Parameters

chunk	Chunk to check.

Returns

0 on success, <0 on error

9.16.2.7 long l4shmc_chunk_size (l4shmc_chunk_t ∗ *chunk*) `[inline]`

Get current size of a chunk.

Parameters

chunk	Chunk.

Returns

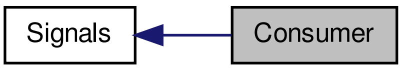
0 on success, <0 on error

Examples:

examples/libs/shmc/prodcons.c.

9.17 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.

9.17.1 Detailed Description

9.17.2 Function Documentation

9.17.2.1 long l4shmc_enable_signal (l4shmc_signal_t ∗ *signal*)

Enable a signal.

Parameters

signal	Signal to enable.

Returns

0 on success, <0 on 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.

9.17.2.2 long l4shmc_wait_any (l4shmc_signal_t ∗∗ *retsignal*) `[inline]`

Wait on any signal.

Return values

retsignal	Signal received.

Returns

0 on success, <0 on error

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

Check whether any waited signal has an event pending.

Return values

retsignal	Signal that has the event pending if any.

Returns

0 on success, <0 on 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.

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

Wait for any signal with timeout.

Parameters

timeout	Timeout.

Return values

retsignal	Signal that has the event pending if any.

Returns

0 on success, <0 on 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.

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

Wait on a specific signal.

Parameters

signal	Signal to wait for.

Returns

0 on success, <0 on error

Examples:

examples/libs/shmc/prodcons.c.

9.17.2.6 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.

Returns

0 on success, <0 on error

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

Check whether a specific signal has an event pending.

Parameters

signal	Signal to check.

Returns

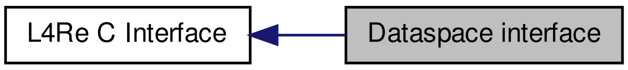
0 on success, <0 on error

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.

9.18 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

  Dataspace type.

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_NO↩ THROW

  Return physical address.

9.18.1 Detailed Description

Dataspace C interface.

9.18.2 Function Documentation

9.18.2.1 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

9.18.2.2 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

9.18.2.3 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

9.18.2.4 long l4re_ds_size (const l4re_ds_t *ds*)

Returns

size of dataspace, <0 on errors

See also

L4Re::Dataspace::size

9.18.2.5 long l4re_ds_flags (const l4re_ds_t *ds*)

See also

L4Re::Dataspace::flags

9.18.2.6 int l4re_ds_info (const l4re_ds_t *ds*, l4re_ds_stats_t ∗ *stats*)

See also

L4Re::Dataspace::info

9.18.2.7 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

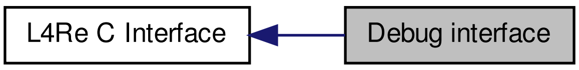
The function returns the physical address of an offset in a dataspace. Use multiple calls of the function to get all physical regions in case of physically non-contiguous dataspaces.

See also

L4Re::Dataspace::phys

9.19 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.

9.19.1 Detailed Description

9.19.2 Function Documentation

9.19.2.1 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

9.20 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.

9.20.1 Detailed Description

9.20.2 Enumeration Type Documentation

9.20.2.1 enum **Libedid_consts**

EDID constants.

Enumerator

Libedid_block_size Size of one EDID block in bytes.

Definition at line 21 of file edid.h.

9.20.3 Function Documentation

9.20.3.1 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

9.20.3.2 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

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

Returns the EDID version number.

Parameters

edid	Pointer to a 128byte EDID block

Returns

Version number

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

Returns the EDID revision number.

Parameters

edid	Pointer to a 128 EDID block

Returns

Revision number

9.20.3.5 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.

9.20.3.6 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.

9.20.3.7 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

9.20.3.8 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

9.20.3.9 void libedid_dump (const unsigned char ∗ *edid*)

Dump raw EDID data to stdout.

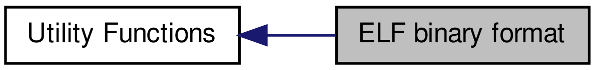
Parameters

edid	Pointer to a 128byte EDID block

9.21 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 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

9.21.1 Detailed Description

Functions and types related to ELF binaries.

9.21.2 Macro Definition Documentation

9.21.2.1 #define EI_CLASS 4

ELF class byte index.

file class

Definition at line 254 of file elf.h.

9.21.2.2 #define EI_CLASS 4

ELF class byte index.

file class

Definition at line 254 of file elf.h.

9.21.2.3 #define ELFCLASSNONE 0

Invalid ELF class.

Invalid class.

Definition at line 270 of file elf.h.

9.21.2.4 #define ELFCLASSNONE 0

Invalid ELF class.

Invalid class.

Definition at line 270 of file elf.h.

9.21.2.5 #define EI_DATA 5

Data encoding byte index.

data encoding

Definition at line 255 of file elf.h.

9.21.2.6 #define EI_DATA 5

Data encoding byte index.

data encoding

Definition at line 255 of file elf.h.

9.21.2.7 #define ELFDATANONE 0

Invalid data encoding.

invalid data encoding

Definition at line 276 of file elf.h.

9.21.2.8 #define ELFDATANONE 0

Invalid data encoding.

invalid data encoding

Definition at line 276 of file elf.h.

9.21.2.9 #define ELFDATA2LSB 1

2's complement, little endian

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

Definition at line 277 of file elf.h.

9.21.2.10 #define ELFDATA2LSB 1

2's complement, little endian

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

Definition at line 277 of file elf.h.

9.21.2.11 #define ELFDATA2MSB 2

2's complement, big endian

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

Definition at line 278 of file elf.h.

9.21.2.12 #define ELFDATA2MSB 2

2's complement, big endian

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

Definition at line 278 of file elf.h.

9.21.2.13 #define EI_VERSION 6

File version byte index.

file version

Value must be EV_CURRENT

Definition at line 256 of file elf.h.

9.21.2.14 #define EI_VERSION 6

File version byte index.

file version

Value must be EV_CURRENT

Definition at line 256 of file elf.h.

9.21.2.15 #define EI_OSABI 7

OS ABI identification.

Operating system / ABI identification.

Definition at line 257 of file elf.h.

9.21.2.16 #define EI_OSABI 7

OS ABI identification.

Operating system / ABI identification.

Definition at line 257 of file elf.h.

9.21.2.17 #define ELFOSABI_SYSV 0

Alias.

UNIX System V ABI (this specification)

Definition at line 282 of file elf.h.

9.21.2.18 #define ELFOSABI_SYSV 0

Alias.

UNIX System V ABI (this specification)

Definition at line 282 of file elf.h.

9.21.2.19 #define ELFOSABI_HPUX 1

HP-UX.

HP-UX operating system.

Definition at line 283 of file elf.h.

9.21.2.20 #define ELFOSABI_HPUX 1

HP-UX.

HP-UX operating system.

Definition at line 283 of file elf.h.

9.21.2.21 #define ELFOSABI_NETBSD 2

NetBSD.

Definition at line 174 of file elf.h.

9.21.2.22 #define ELFOSABI_LINUX 3

Linux.

Definition at line 175 of file elf.h.

9.21.2.23 #define ELFOSABI_SOLARIS 6

Sun Solaris.

Definition at line 176 of file elf.h.

9.21.2.24 #define ELFOSABI_AIX 7

IBM AIX.

Definition at line 177 of file elf.h.

9.21.2.25 #define ELFOSABI_IRIX 8

SGI Irix.

Definition at line 178 of file elf.h.

9.21.2.26 #define ELFOSABI_FREEBSD 9

FreeBSD.

Definition at line 179 of file elf.h.

9.21.2.27 #define ELFOSABI_TRU64 10

Compaq TRU64 UNIX.

Definition at line 180 of file elf.h.

9.21.2.28 #define ELFOSABI_MODESTO 11

Novell Modesto.

Definition at line 181 of file elf.h.

9.21.2.29 #define ELFOSABI_OPENBSD 12

OpenBSD.

Definition at line 182 of file elf.h.

9.21.2.30 #define EI_PAD 9

Byte index of padding bytes.

start of padding bytes

Definition at line 259 of file elf.h.

9.21.2.31 #define EI_PAD 9

Byte index of padding bytes.

start of padding bytes

Definition at line 259 of file elf.h.

9.21.2.32 #define EM_ARC 45

Argonaut RISC Core, Argonaut Techn Inc.

Definition at line 226 of file elf.h.

9.21.2.33 #define SHT_NUM 19

Number of defined types.

Definition at line 348 of file elf.h.

9.21.2.34 #define SHF_GROUP 0x200

Section is member of a group.

Definition at line 368 of file elf.h.

9.21.2.35 #define SHF_TLS 0x400

Section hold thread-local data.

Definition at line 369 of file elf.h.

9.21.2.36 #define SHF_MASKOS 0x0ff00000

OS-specific.

Definition at line 370 of file elf.h.

9.21.2.37 #define PT_LOOS 0x60000000

os spec.

Definition at line 413 of file elf.h.

9.21.2.38 #define PT_HIOS 0x6fffffff

os spec.

Definition at line 414 of file elf.h.

9.21.2.39 #define PT_LOPROC 0x70000000

processor spec.

Definition at line 415 of file elf.h.

9.21.2.40 #define PT_HIPROC 0x7fffffff

processor spec.

Definition at line 416 of file elf.h.

9.21.2.41 #define PT_GNU_EH_FRAME (PT_LOOS + 0x474e550)

EH frame information.

Definition at line 418 of file elf.h.

9.21.2.42 #define PT_GNU_STACK (PT_LOOS + 0x474e551)

Flags for stack.

Definition at line 419 of file elf.h.

9.21.2.43 #define PT_GNU_RELRO (PT_LOOS + 0x474e552)

Read only after reloc.

Definition at line 420 of file elf.h.

9.21.2.44 #define PT_L4_STACK (PT_LOOS + 0x12)

Address of the stack.

Definition at line 422 of file elf.h.

9.21.2.45 #define PT_L4_KIP (PT_LOOS + 0x13)

Address of the KIP.

Definition at line 423 of file elf.h.

9.21.2.46 #define PT_L4_AUX (PT_LOOS + 0x14)

Address of the AUX strcutures.

Definition at line 424 of file elf.h.

9.21.2.47 #define NT_VERSION 1

Contains a version string.

Definition at line 455 of file elf.h.

9.21.2.48 #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.

9.21.2.49 #define DT_LOPROC 0x70000000

processor spec.

Definition at line 516 of file elf.h.

9.21.2.50 #define DT_HIPROC 0x7fffffff

processor spec.

Definition at line 517 of file elf.h.

9.21.2.51 #define DF_1_NOW 0x00000001

Set RTLD_NOW for this object.

Definition at line 528 of file elf.h.

9.21.2.52 #define DF_1_GLOBAL 0x00000002

Set RTLD_GLOBAL for this object.

Definition at line 529 of file elf.h.

9.21.2.53 #define DF_1_GROUP 0x00000004

Set RTLD_GROUP for this object.

Definition at line 530 of file elf.h.

9.21.2.54 #define DF_1_NODELETE 0x00000008

Set RTLD_NODELETE for this object.

Definition at line 531 of file elf.h.

9.21.2.55 #define DF_1_LOADFLTR 0x00000010

Trigger filtee loading at runtime.

Definition at line 532 of file elf.h.

9.21.2.56 #define DF_1_NOOPEN 0x00000040

Set RTLD_NOOPEN for this object.

Definition at line 534 of file elf.h.

9.21.2.57 #define DF_1_ORIGIN 0x00000080

$ORIGIN must be handled.

Definition at line 535 of file elf.h.

9.21.2.58 #define DF_1_DIRECT 0x00000100

Direct binding enabled.

Definition at line 536 of file elf.h.

9.21.2.59 #define DF_1_INTERPOSE 0x00000400

Object is used to interpose.

Definition at line 538 of file elf.h.

9.21.2.60 #define DF_1_NODEFLIB 0x00000800

Ignore default lib search path.

Definition at line 539 of file elf.h.

9.21.2.61 #define DF_1_NODUMP 0x00001000

Object can't be dldump'ed.

Definition at line 540 of file elf.h.

9.21.2.62 #define DF_1_CONFALT 0x00002000

Configuration alternative created.

Definition at line 541 of file elf.h.

9.21.2.63 #define DF_1_ENDFILTEE 0x00004000

Filtee terminates filters search.

Definition at line 542 of file elf.h.

9.21.2.64 #define DF_1_DISPRELDNE 0x00008000

Disp reloc applied at build time.

Definition at line 543 of file elf.h.

9.21.2.65 #define DF_1_DISPRELPND 0x00010000

Disp reloc applied at run-time.

Definition at line 544 of file elf.h.

9.21.2.66 #define DF_P1_LAZYLOAD 0x00000001

Lazyload following object.

Definition at line 551 of file elf.h.

9.21.2.67 #define DF_P1_GROUPPERM 0x00000002

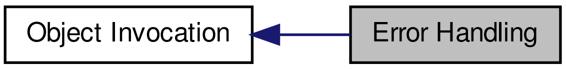
Symbols from next object are not generally available.

Definition at line 552 of file elf.h.

9.22 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_R↩
  ERCVPFTO = 0x0d,
  L4_IPC_SERCVPFTO = 0x0c, L4_IPC_REABORTED = 0x0f, L4_IPC_SEABORTED = 0x0e, L4_IPC_RE↩
  MSGCUT = 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.

9.22.1 Detailed Description

Error handling for L4 object invocation.

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

9.22.2 Enumeration Type Documentation

9.22.2.1 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.

9.22.3 Function Documentation

9.22.3.1 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).

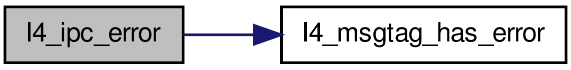
Examples:

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

Definition at line 430 of file ipc.h.

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

Here is the call graph for this function:



9.22.3.2 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

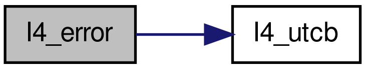
Examples:

examples/clntsrv/client.cc, 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 447 of file ipc.h.

References l4_utcb().

Here is the call graph for this function:



9.22.3.3 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 453 of file ipc.h.

References l4_thread_regs_t::error.

9.22.3.4 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 456 of file ipc.h.

References l4_thread_regs_t::error.

9.22.3.5 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.

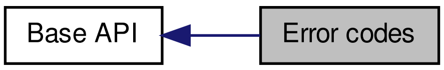
Definition at line 459 of file ipc.h.

References l4_thread_regs_t::error, and L4_IPC_ERROR_MASK.

9.23 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_EIPC_LO = 2000, L4_EIPC_HI = 2000 + 0x1f }

  L4 error codes.

9.23.1 Detailed Description

Common error codes.

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

9.23.2 Enumeration Type Documentation

9.23.2.1 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.

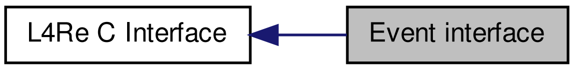
L4_ERANGE	Range error.
L4_ENAMETOOLONG	Name too long.
L4_ENOSYS	No sys.
L4_EBADPROTO	Unsupported protocol.
L4_EADDRNOTAVAIL	Address not available.
L4_ERRNOMAX	Maximum error value.
L4_ENOREPLY	No reply.
L4_EIPC_LO	Communication error-range low.
L4_EIPC_HI	Communication error-range high.

Definition at line 41 of file err.h.

9.24 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 ∗axis, l4re_event_absinfo_t ∗info) L4_NOTHROW

  Get Axis information for a stream.

9.24.1 Detailed Description

Event C interface.

9.24.2 Function Documentation

9.24.2.1 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

9.24.2.2 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

9.24.2.3 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

9.24.2.4 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

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

Get Axis information for a stream.

Parameters

server	Server to talk to.
naxes	Number of axes.

Return values

axis	Number of axes.
info	Information buffer.

Returns

0 for success, <0 on error

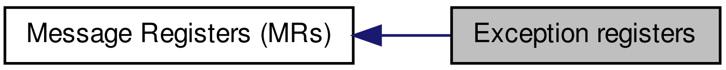
See also

L4Re::Event::get_axis_info

9.25 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.

9.25.1 Detailed Description

Overly definition of the MRs for exception messages.

9.25.2 Function Documentation

9.25.2.1 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`.

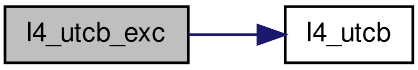
Examples:

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:



9.25.2.2 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.

9.25.2.3 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.

9.25.2.4 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

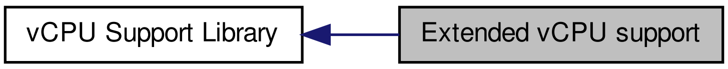
Function to check whether an exception IPC is a page fault, also applies to I/O pagefaults.

Definition at line 105 of file utcb.h.

9.26 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.

9.26.1 Detailed Description

extended vCPU handling functionality.

9.26.2 Function Documentation

9.26.2.1 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.

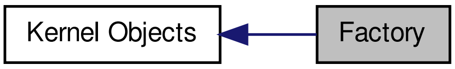
Returns

0 for success, error code otherwise

9.27 Factory

A factory is used to create all kinds of 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_thread (l4_cap_idx_t factory, l4_cap_idx_t target_cap) 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_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_irq (l4_cap_idx_t factory, l4_cap_idx_t target_cap) L4_NOTHROW

    Create a new IRQ.
- 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.

9.27.1 Detailed Description

A factory is used to create all kinds of kernel objects.

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

A factory provides the means to create all kinds of kernel objects. 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.

9.27.2 Function Documentation

9.27.2.1 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.
target_cap	Capability selector for the root capability of the new task.
utcb_area	Flexpage that describes the area for the UTCBs of the new task

Note

The size of the UTCB area specifies indirectly the maximum number of UTCBs available for this task and cannot be changed afterwards.

Returns

Syscall return tag

See also

Task

Definition at line 306 of file factory.h.

References l4_utcb().

Here is the call graph for this function:

l4_factory_create_task
l4_utcb

9.27.2.2 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.
target_cap	Capability selector for the root capability of the new thread.

Returns

Syscall return tag

See also

Thread

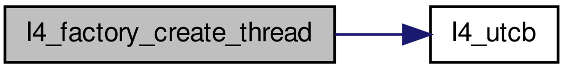
Examples:

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 313 of file factory.h.

References l4_utcb().

Here is the call graph for this function:



9.27.2.3 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.
target_cap	Capability selector for the root capability of the new factory.
limit	Limit for the new factory in bytes

Note

The limit of the new factory is subtracted from the available amount of the factory used for creation.

Returns

Syscall return tag

Definition at line 320 of file factory.h.

References l4_utcb().

Here is the call graph for this function:

l4_factory_create_factory
l4_utcb

9.27.2.4 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.
target_cap	Capability selector for the root capability of the new IPC gate.
thread_cap	Thread to bind the gate to
label	Label of the gate

Returns

Syscall return tag

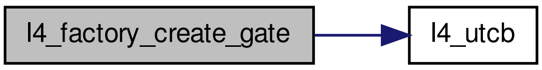
See also

IPC-Gate API

Definition at line 328 of file factory.h.

References l4_utcb().

Here is the call graph for this function:



9.27.2.5 l4_msgtag_t l4_factory_create_irq (l4_cap_idx_t *factory,* l4_cap_idx_t *target_cap*) [inline]

Create a new IRQ.

Parameters

factory	Capability selector for factory to use for creation.
target_cap	Capability selector for the root capability of the new IRQ.

Returns

Syscall return tag

See also

IRQs

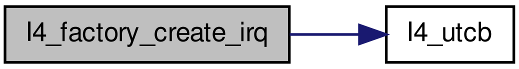
Examples:

examples/sys/isr/main.c.

Definition at line 336 of file factory.h.

References l4_utcb().

Here is the call graph for this function:



9.27.2.6 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.
target_cap	Capability selector for the root capability of the new VM.

Returns

Syscall return tag

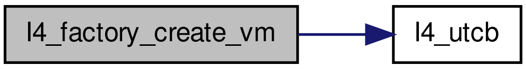
See also

Virtual Machines

Definition at line 343 of file factory.h.

References l4_utcb().

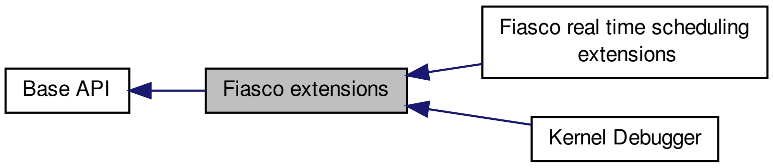
Here is the call graph for this function:



9.28 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_t

    Trace buffer status.
- struct l4_tracebuffer_status_window_t

    Trace-buffer status window descriptor.

Macros

- #define LOG_EVENT_CONTEXT_SWITCH 0

    Event: context switch

    .
- #define LOG_EVENT_IPC_SHORTCUT 1

    Event: IPC shortcut

    .
- #define LOG_EVENT_IRQ_RAISED 2

    Event: IRQ occurred

    .
- #define LOG_EVENT_TIMER_IRQ 3

Event: Timer IRQ occurred

.

- #define LOG_EVENT_THREAD_EX_REGS 4

  Event: thread_ex_regs

  .

- #define LOG_EVENT_MAX_EVENTS 16

  Maximum number of events

  .

- #define LOG_EVENT_CONTEXT_SWITCH 0

  Event: context switch

  .

- #define LOG_EVENT_IPC_SHORTCUT 1

  Event: IPC shortcut

  .

- #define LOG_EVENT_IRQ_RAISED 2

  Event: IRQ occurred

  .

- #define LOG_EVENT_TIMER_IRQ 3

  Event: Timer IRQ occurred

  .

- #define LOG_EVENT_THREAD_EX_REGS 4

  Event: thread_ex_regs

  .

- #define LOG_EVENT_MAX_EVENTS 16

  Maximum number of events

  .

Enumerations

- enum

  Log event types.

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 size, 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.

9.28.1 Detailed Description

Kernel debugger extensions of the Fiasco L4 implementation.

9.28.2 Function Documentation

9.28.2.1 l4_tracebuffer_status_t ∗ fiasco_tbuf_get_status (void) [inline]

Return trace buffer status.

Return trace-buffer status.

Return tracebuffer status.

Returns

Pointer to trace buffer status struct.
Pointer to tracebuffer status struct.
Pointer to trace-buffer status struct.

Definition at line 183 of file ktrace.h.

9.28.2.2 l4_addr_t fiasco_tbuf_get_status_phys (void) [inline]

Return the physical address of the trace buffer status struct.

Return the physical address of the trace-buffer status struct.

Return the physical address of the tracebuffer status struct.

Returns

physical address of status struct.

Definition at line 190 of file ktrace.h.

9.28.2.3 l4_umword_t fiasco_tbuf_log (const char * *text*) [inline]

Create new trace buffer entry with describing <text>.

Create new trace-buffer entry with describing <text>.

Create new tracebuffer entry with describing <text>.

Parameters

text	Logging text

Returns

Pointer to trace buffer entry

Parameters

text	Logging text

Returns

Pointer to tracebuffer entry

Parameters

text	Logging text

Returns

Pointer to trace-buffer entry

Definition at line 197 of file ktrace.h.

9.28.2.4 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.

Create new trace-buffer entry with describing <text> and three additional values.

Create new tracebuffer 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

Parameters

text	Logging text
v1	first value
v2	second value
v3	third value

Returns

Pointer to tracebuffer entry

Parameters

text	Logging text
v1	first value
v2	second value
v3	third value

Returns

Pointer to trace-buffer entry

Definition at line 203 of file ktrace.h.

9.28.2.5 l4_umword_t fiasco_tbuf_log_binary (const unsigned char ∗ *data*) [inline]

Create new trace buffer entry with binary data.

Create new trace-buffer entry with binary data.

Create new tracebuffer entry with binary data.

Parameters

data	binary data

Returns

Pointer to trace buffer entry

Parameters

data	binary data

Returns

Pointer to tracebuffer entry

Parameters

data	binary data

Returns

Pointer to trace-buffer entry

Definition at line 233 of file ktrace.h.

9.28.2.6 void fiasco_tbuf_clear (void) [inline]

Clear trace buffer.

Clear trace-buffer.

Clear tracebuffer.

Definition at line 209 of file ktrace.h.

9.28.2.7 void fiasco_tbuf_dump (void) [inline]

Dump trace buffer to kernel console.

Dump trace-buffer to kernel console.

Dump tracebuffer to kernel console.

Definition at line 215 of file ktrace.h.

9.28.2.8 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.

9.28.2.9 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.

9.28.2.10 long fiasco_ldt_set (l4_cap_idx_t *task*, void ∗ *ldt*, unsigned int *size*, unsigned int *entry_number_start*, l4_utcb_t ∗ *utcb*) [inline]

Set LDT segments descriptors.

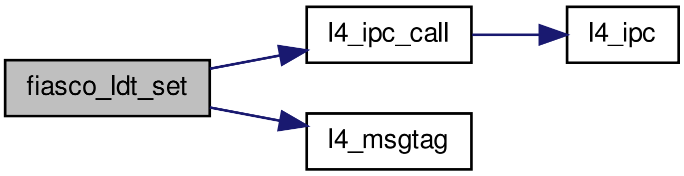
Parameters

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, and l4_msg_regs_t::mr.

Here is the call graph for this function:



9.28.2.11 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.

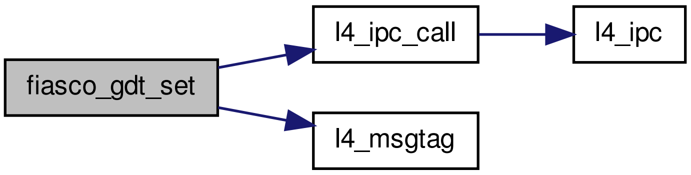
Returns

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:



9.28.2.12 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.

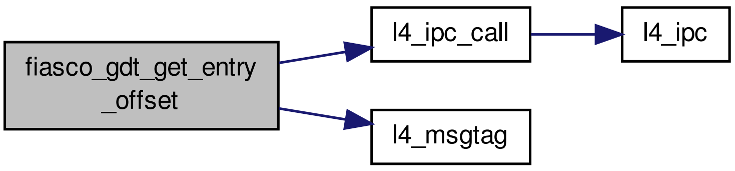
Parameters

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.

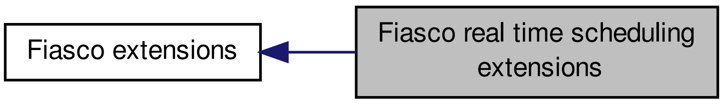
Here is the call graph for this function:



9.29 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.

9.30 Fiasco-UX Virtual devices

Virtual hardware devices, provided by Fiasco-UX.

Collaboration diagram for Fiasco-UX Virtual devices:

L4_kip_api
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.

9.30.1 Detailed Description

Virtual hardware devices, provided by Fiasco-UX.

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

9.30.2 Enumeration Type Documentation

9.30.2.1 enum l4_vhw_entry_type

Type of device.

Enumerator

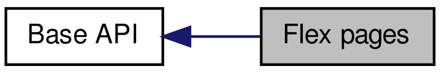
L4_TYPE_VHW_NONE	None entry.
L4_TYPE_VHW_FRAMEBUFFER	Framebuffer device.
L4_TYPE_VHW_INPUT	Input device.
L4_TYPE_VHW_NET	Network device.

Definition at line 44 of file vhw.h.

9.31 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_FPAG↩
  E_ADDR_SHIFT = 12,
  L4_FPAGE_RIGHTS_BITS = 4, L4_FPAGE_TYPE_BITS = 2, L4_FPAGE_SIZE_BITS = 6, L4_FPAGE_A↩
  DDR_BITS = L4_MWORD_BITS - L4_FPAGE_ADDR_SHIFT }

  L4 flexpage structure.
- enum { L4_WHOLE_ADDRESS_SPACE = 63 }

  Constants for flexpages.
- enum L4_fpage_rights { L4_FPAGE_RO = 4, L4_FPAGE_RW = 6 }

  Flex-page rights.
- enum L4_cap_fpage_rights { L4_CAP_FPAGE_R = 0x4, L4_CAP_FPAGE_RO = 0x4, L4_CAP_FPAGE_RW = 0x5 }

  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

  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 L4_DEFAULT_PARAM(0))
  Determine maximum flex page size of a region.

9.31.1 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.

9.31.2 Enumeration Type Documentation

9.31.2.1 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.

Definition at line 55 of file __l4_fpage.h.

9.31.2.2 anonymous enum

Constants for flexpages.

Enumerator

L4_WHOLE_ADDRESS_SPACE Whole address space size.

Definition at line 86 of file __l4_fpage.h.

9.31.2.3 enum L4_fpage_rights

Flex-page rights.

Enumerator

L4_FPAGE_RO Read-only flex page.

L4_FPAGE_RW Read-write flex page.

Definition at line 104 of file __l4_fpage.h.

9.31.2.4 enum L4_cap_fpage_rights

Cap-flex-page rights.

Enumerator

L4_CAP_FPAGE_R Read-only cap.

L4_CAP_FPAGE_RO Read-only cap.

L4_CAP_FPAGE_RW Read-write cap.

Definition at line 117 of file __l4_fpage.h.

9.31.2.5 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 164 of file __l4_fpage.h.

9.31.2.6 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 183 of file __l4_fpage.h.

9.31.3 Function Documentation

9.31.3.1 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 453 of file __l4_fpage.h.

9.31.3.2 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 471 of file __l4_fpage.h.

References L4_WHOLE_ADDRESS_SPACE.

9.31.3.3 l4_fpage_t l4_fpage_invalid (void) `[inline]`

Get an invalid flex page.

Returns

Special *invalid* flex page.

Definition at line 477 of file __l4_fpage.h.

9.31.3.4 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 459 of file __l4_fpage.h.

References L4_FPAGE_ADDR_SHIFT, and L4_FPAGE_RW.

9.31.3.5 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 465 of file __l4_fpage.h.

9.31.3.6 int l4_is_fpage_writable (l4_fpage_t *fp*) [inline]

Test if the flex page is writable.

Parameters

fp	Flex page.

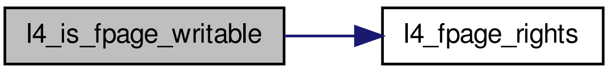
Returns

!= 0 if flex page is writable, 0 if not

Definition at line 484 of file __l4_fpage.h.

References l4_fpage_rights().

Here is the call graph for this function:



9.31.3.7 unsigned l4_fpage_rights (l4_fpage_t *f*) [inline]

Return rights from a flex page.

Parameters

f	Flex page

Returns

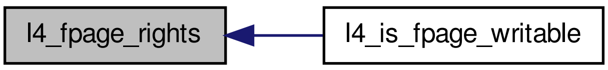
Size part of the given flex page.

Definition at line 403 of file __l4_fpage.h.

References L4_FPAGE_RIGHTS_SHIFT.

Referenced by l4_is_fpage_writable().

Here is the caller graph for this function:



9.31.3.8 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 409 of file __l4_fpage.h.

References L4_FPAGE_TYPE_SHIFT.

9.31.3.9 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 415 of file __l4_fpage.h.

References L4_FPAGE_SIZE_SHIFT.

9.31.3.10 unsigned long l4_fpage_page (l4_fpage_t *f*) `[inline]`

Return page from a flex page.

Parameters

f	Flex page

Returns

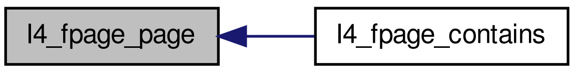
Page part of the given flex page.

Definition at line 421 of file __l4_fpage.h.

References L4_FPAGE_ADDR_SHIFT.

Referenced by l4_fpage_contains().

Here is the caller graph for this function:



9.31.3.11 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 444 of file __l4_fpage.h.

References L4_FPAGE_RIGHTS_SHIFT, and l4_fpage_t::raw.

9.31.3.12 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.

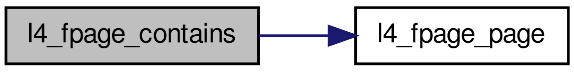
Parameters

fpage	Flex page
addr	Address
size	Size of range in log2.

Definition at line 503 of file __l4_fpage.h.

References L4_FPAGE_ADDR_SHIFT, and l4_fpage_page().

Here is the call graph for this function:



9.31.3.13 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 *L4_DEFAULT_PARAM0*) [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

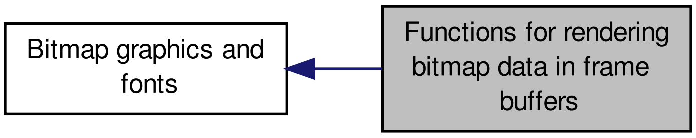
Maximum order (log2-size) possible.

Note

The start address of the flex-page can be determined with l4_trunc_size(addr, returnvalue)

9.32 Functions for rendering bitmap data in frame buffers

Collaboration diagram for Functions for rendering bitmap data in frame buffers:


Bitmap graphics and fonts
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.

9.32.1 Detailed Description

9.32.2 Typedef Documentation

9.32.2.1 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.

9.32.2.2 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.

9.32.3 Function Documentation

9.32.3.1 **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.

9.32.3.2 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.
fbi	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.

9.32.3.3 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.
fbi	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).

9.32.3.4 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.
fbi	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.

9.32.3.5 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*)

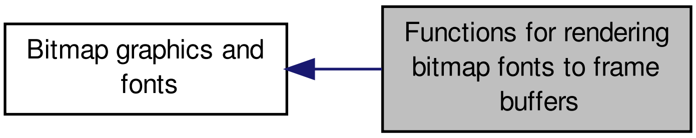
Copy a rectangular area.

Parameters

dest	Destination frame buffer.
src	Source frame buffer.
fbi	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.

9.33 Functions for rendering bitmap fonts to frame buffers

Collaboration diagram for Functions for rendering bitmap fonts to frame buffers:


Bitmap graphics and fonts
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.

9.33.1 Detailed Description

9.33.2 Enumeration Type Documentation

9.33.2.1 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.

9.33.3 Function Documentation

9.33.3.1 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

9.33.3.2 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.

9.33.3.3 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.

9.33.3.4 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.

9.33.3.5 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.

9.33.3.6 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.
fbi	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.

9.33.3.7 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.

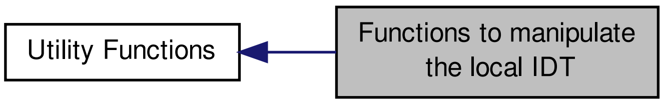
Parameters

fb	Pointer to frame buffer.
fbi	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.

9.34 Functions to manipulate the local IDT

Collaboration diagram for Functions to manipulate the local IDT:



Data Structures

- struct l4util_idt_desc_t

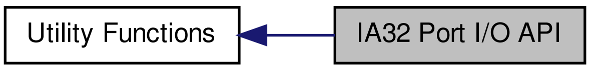
  IDT entry.
- struct l4util_idt_header_t

  Header of an IDT table.

9.34.1 Detailed Description

9.35 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

9.35.1 Detailed Description

9.35.2 Function Documentation

9.35.2.1 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.

9.35.2.2 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.

9.35.2.3 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.

9.35.2.4 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.

9.35.2.5 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.

9.35.2.6 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.

9.35.2.7 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.

9.35.2.8 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.

9.35.2.9 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.

9.35.2.10 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.

9.35.2.11 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.

9.35.2.12 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.

9.36 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 L4_EXPORT l4io_request_iomem (l4_addr_t phys, unsigned long size, int flags, l4_addr_t ∗virt)

  Request an IO memory region.
- long L4_EXPORT 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 L4_EXPORT l4io_release_iomem (l4_addr_t virt, unsigned long size)

  Release an IO memory region.
- long L4_EXPORT 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 L4_EXPORT l4io_request_ioport (unsigned portnum, unsigned len)

  Request an IO port region.
- long L4_EXPORT l4io_release_ioport (unsigned portnum, unsigned len)

  Release an IO port region.
- int L4_EXPORT 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 L4_EXPORT 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 L4_EXPORT l4io_request_resource_iomem (l4io_device_handle_t devhandle, l4io_resource_↵
  handle_t ∗reshandle)

  Request IO memory.
- int L4_EXPORT l4io_has_resource (enum l4io_resource_types_t type, l4vbus_paddr_t start, l4vbus_paddr↵
  _t end)

  Check if a resource is available.

9.36.1 Detailed Description

9.36.2 Typedef Documentation

9.36.2.1 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.

9.36.3 Enumeration Type Documentation

9.36.3.1 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.

9.36.3.2 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.

9.36.3.3 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.

9.36.4 Function Documentation

9.36.4.1 long L4_EXPORT 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.

9.36.4.2 long L4_EXPORT 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.

9.36.4.3 long L4_EXPORT 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

9.36.4.4 long L4_EXPORT 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

9.36.4.5 long **L4_EXPORT** 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

9.36.4.6 long **L4_EXPORT** 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

9.36.4.7 int **L4_EXPORT** 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

9.36.4.8 int L4_EXPORT 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

9.36.4.9 l4_addr_t L4_EXPORT 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

9.36.4.10 int L4_EXPORT 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.

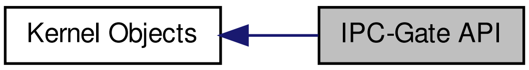
9.37 IPC Messaging Framework

9.38 IPC Streams

9.39 IPC-Gate API

Secure comminication object.

Collaboration diagram for IPC-Gate API:



Enumerations

- enum L4_ipc_gate_ops { L4_IPC_GATE_BIND_OP = 0x10, L4_IPC_GATE_GET_INFO_OP = 0x11 }

  Operations on the IPC-gate.

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 the thread.
- l4_msgtag_t l4_ipc_gate_get_infos (l4_cap_idx_t gate, l4_umword_t ∗label)

  Get information on the IPC-gate.

9.39.1 Detailed Description

Secure comminication 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.

See also

Object Invocation

9.39.2 Enumeration Type Documentation

9.39.2.1 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 75 of file ipc_gate.h.

9.39.3 Function Documentation

9.39.3.1 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 the thread.

Parameters

t	Thread to bind the IPC-gate to
label	Label to use
utcb	UTCB to use.

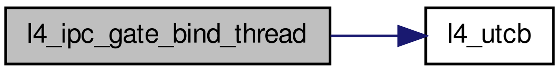
Returns

System call return tag.

Definition at line 117 of file ipc_gate.h.

References l4_utcb().

Here is the call graph for this function:



9.39.3.2 l4_msgtag_t l4_ipc_gate_get_infos (l4_cap_idx_t *gate*, l4_umword_t ∗ *label*) [inline]

Get information on the IPC-gate.

Return values

label	Label of the gate.

Parameters

utcb	UTCb to use.

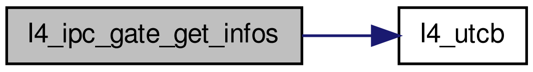
Returns

System call return tag.

Definition at line 124 of file ipc_gate.h.

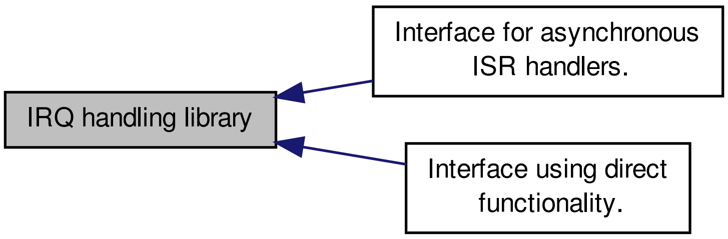
References l4_utcb().

Here is the call graph for this function:



9.40 IRQ handling library

Collaboration diagram for IRQ handling library:


Interface for asynchronous ISR handlers.
IRQ handling library
Interface using direct functionality.

Modules

- Interface for asynchronous ISR handlers.

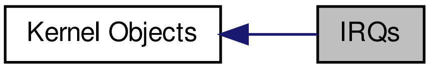
    This interface has just two (main) functions.
- Interface using direct functionality.

9.40.1 Detailed Description

9.41 IRQs

The IRQ and ICU class.

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 to an interrupt source.
- l4_msgtag_t l4_irq_chain (l4_cap_idx_t irq, l4_umword_t label, l4_cap_idx_t slave) L4_NOTHROW

  Chain an IRQ to another master IRQ source.
- l4_msgtag_t l4_irq_detach (l4_cap_idx_t irq) L4_NOTHROW

  Detach from an interrupt source.
- l4_msgtag_t l4_irq_trigger (l4_cap_idx_t irq) L4_NOTHROW

  Trigger an IRQ.
- 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_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_unmask (l4_cap_idx_t irq) L4_NOTHROW

  Unmask IRQ.

9.41.1 Detailed Description

The IRQ and ICU class.

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

The IRQ class provides access to abstract interrupts provided by the micro kernel. Interrupts may be hardware interrupts provided by the platform interrupt controller, virtual device interrupts provided by the micro kernel virtual devices (virtual serial or trace buffer), or IRQs (virtual interrupts that can be triggered by user programs).

IRQ objects can be created using a Factory, see Factory (l4_factory_create_irq()).

9.41.2 Enumeration Type Documentation

9.41.2.1 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 68 of file icu.h.

9.41.3 Function Documentation

9.41.3.1 l4_msgtag_t l4_irq_attach (l4_cap_idx_t *irq*, l4_umword_t *label*, l4_cap_idx_t *thread*) `[inline]`

Attach to an interrupt source.

Parameters

irq	IRQ to attach to.
label	Identifier of the IRQ.
thread	Thread to attach the interrupt to.

Returns

Syscall return tag

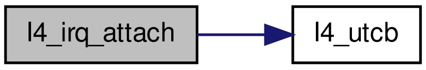
Examples:

examples/sys/isr/main.c.

Definition at line 281 of file irq.h.

References l4_utcb().

Here is the call graph for this function:



9.41.3.2 l4_msgtag_t l4_irq_chain (l4_cap_idx_t *irq*, l4_umword_t *label*, l4_cap_idx_t *slave*) [inline]

Chain an IRQ to another master IRQ source.

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.

Parameters

irq	The master IRQ object.
label	Identifier of the IRQ.
slave	The slave that shall be attached to the master.

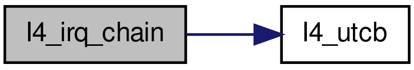
Returns

Syscall return tag

Definition at line 288 of file irq.h.

References l4_utcb().

Here is the call graph for this function:



9.41.3.3 l4_msgtag_t l4_irq_detach (l4_cap_idx_t *irq*) [inline]

Detach from an interrupt source.

Parameters

irq	IRQ to detach from.

Returns

Syscall return tag

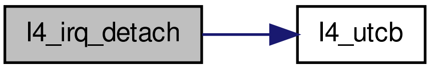
Examples:

examples/sys/isr/main.c.

Definition at line 295 of file irq.h.

References l4_utcb().

Here is the call graph for this function:



9.41.3.4 l4_msgtag_t l4_irq_trigger (l4_cap_idx_t *irq*) [inline]

Trigger an IRQ.

Parameters

irq	IRQ to trigger.

Precondition

irq must be a reference to an IRQ.

Returns

Syscall return tag.

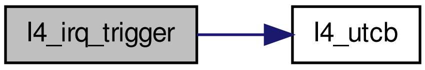
Note that this function is a send only operation, i.e. there is no return value except for a failed send operation. Especially l4_error() will return an error value from the message tag which still contains the IRQ protocol used for the send operation.

Use l4_ipc_error() to check for (send) errors.

Definition at line 301 of file irq.h.

References l4_utcb().

Here is the call graph for this function:



9.41.3.5 l4_msgtag_t l4_irq_receive (l4_cap_idx_t *irq*, l4_timeout_t *to*) [inline]

Unmask and wait for specified IRQ.

Parameters

irq	IRQ to wait for.
to	Timeout.

Returns

Syscall return tag

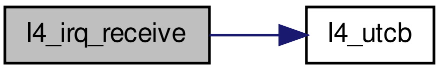
Examples:

examples/sys/isr/main.c.

Definition at line 307 of file irq.h.

References l4_utcb().

Here is the call graph for this function:



9.41.3.6 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	IRQ to wait for.

label	Receive label.
to	Timeout.

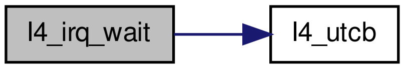
Returns

Syscall return tag

Definition at line 313 of file irq.h.

References l4_utcb().

Here is the call graph for this function:



9.41.3.7 l4_msgtag_t l4_irq_unmask (l4_cap_idx_t *irq*) [inline]

Unmask IRQ.

Parameters

irq	IRQ to unmask.

Returns

Syscall return tag

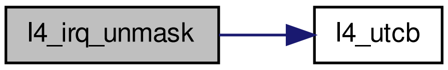
Note

l4_irq_wait and l4_irq_receive are doing the unmask themselves.

Definition at line 320 of file irq.h.

References l4_utcb().

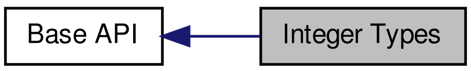
Here is the call graph for this function:



9.42 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

  Signed size type.
- typedef signed int l4_ssize_t

  Unsigned size type.
- typedef unsigned long l4_size_t

  Signed size type.
- typedef signed long l4_ssize_t

  Unsigned size type.
- typedef unsigned int l4_size_t

  Signed size type.
- typedef signed int l4_ssize_t

  Unsigned size type.

9.42.1 Detailed Description

```
#include<l4/sys/l4int.h>
```

9.42.2 Typedef Documentation

9.42.2.1 typedef signed char l4_int8_t

Signed 8bit value.

Definition at line 35 of file l4int.h.

9.42.2.2 typedef unsigned char l4_uint8_t

Unsigned 8bit value.

Definition at line 36 of file l4int.h.

9.42.2.3 typedef signed short int l4_int16_t

Signed 16bit value.

Definition at line 37 of file l4int.h.

9.42.2.4 typedef unsigned short int l4_uint16_t

Unsigned 16bit value.

Definition at line 38 of file l4int.h.

9.42.2.5 typedef signed int l4_int32_t

Signed 32bit value.

Definition at line 39 of file l4int.h.

9.42.2.6 typedef unsigned int l4_uint32_t

Unsigned 32bit value.

Definition at line 40 of file l4int.h.

9.42.2.7 typedef signed long long l4_int64_t

Signed 64bit value.

Definition at line 41 of file l4int.h.

9.42.2.8 typedef unsigned long long l4_uint64_t

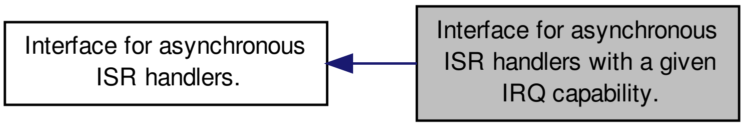
Unsigned 64bit value.

Definition at line 42 of file l4int.h.

9.43 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.:


Interface for asynchronous ISR handlers.
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.

9.43.1 Detailed Description

This group is just an enhanced version to l4irq_request() which takes a capability object instead of a plain number.

9.43.2 Function Documentation

9.43.2.1 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

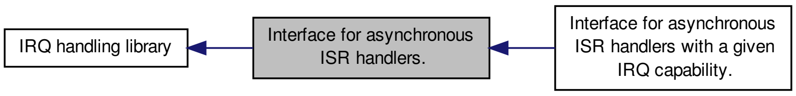
Returns

Pointer to l4irq_t structure, 0 on error

9.44 Interface for asynchronous ISR handlers.

This interface has just two (main) functions.

Collaboration diagram for Interface for asynchronous ISR handlers.:


IRQ handling library
Interface for asynchronous ISR handlers.
Interface for asynchronous ISR handlers with a given IRQ capability.

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.

9.44.1 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.

9.44.2 Function Documentation

9.44.2.1 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.

9.44.2.2 long l4irq_release (l4irq_t ∗ *irq*)

Release asynchronous ISR handler and free resources.

Parameters

irq	IRQ data structure

Returns

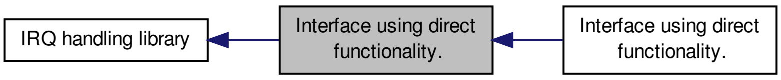
0 sucess, != 0 failure

Examples:

examples/libs/libirq/async_isr.c.

9.45 Interface using direct functionality.

Collaboration diagram for Interface using direct functionality.:


IRQ handling library
Interface using direct functionality.
Interface using direct functionality.

Modules

- 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.

9.45.1 Detailed Description

9.45.2 Function Documentation

9.45.2.1 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.

9.45.2.2 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.

9.45.2.3 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.

9.45.2.4 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.

9.45.2.5 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.

9.45.2.6 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

9.45.2.7 long l4irq_wait_any (l4irq_t ∗∗ *irq*)

Wait for any attached IRQ.

Return values

irq	Received interrupt.

Returns

0 on success, != 0 on error

9.45.2.8 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.

9.45.2.9 long l4irq_detach (l4irq_t ∗ *irq*)

Detach from IRQ.

Parameters

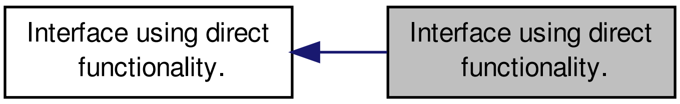
irq	IRQ data structure

Returns

0 on success, != 0 on error

9.46 Interface using direct functionality.

Collaboration diagram for Interface using direct functionality.:


Interface using direct functionality.
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.

9.46.1 Detailed Description

9.46.2 Function Documentation

9.46.2.1 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.

9.46.2.2 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.

9.46.2.3 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.

9.46.2.4 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

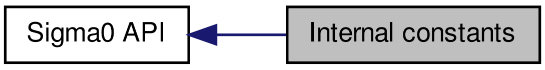
Pointer to l4irq_t structure, 0 on error

The pointer to the IRQ structure is used as a label in the IRQ object.

9.47 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))

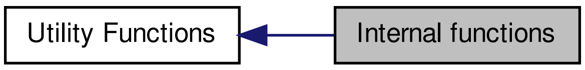
  New client.

9.47.1 Detailed Description

Internal sigma0 definitions.

9.48 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)

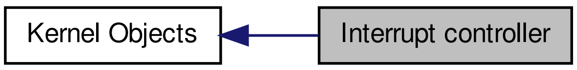
  decode base-64-encoded string infile

9.48.1 Detailed Description

9.49 Interrupt controller

The ICU class.

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 vector 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 vector 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 mode of interrupt.
- l4_msgtag_t l4_icu_info (l4_cap_idx_t icu, l4_icu_info_t ∗info) L4_NOTHROW

  Get info about capabilites of ICU.
- l4_msgtag_t l4_icu_msi_info (l4_cap_idx_t icu, unsigned irqnum, l4_umword_t ∗msg) 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 vector.
- l4_msgtag_t l4_icu_mask (l4_cap_idx_t icu, unsigned irqnum, l4_umword_t ∗label, l4_timeout_t to) L4_NO↩
THROW

  Mask an IRQ vector.

9.49.1 Detailed Description

The ICU class.

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

To setup an IRQ vector 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

9.49.2 Typedef Documentation

9.49.2.1 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().

9.49.3 Enumeration Type Documentation

9.49.3.1 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 51 of file icu.h.

9.49.4 Function Documentation

9.49.4.1 l4_msgtag_t l4_icu_bind (l4_cap_idx_t *icu*, unsigned *irqnum*, l4_cap_idx_t *irq*) `[inline]`

Bind an interrupt vector of an interrupt controller to an interrupt object.

Parameters

icu	ICU to use.
irqnum	IRQ vector at the ICU.
irq	IRQ capability to bind the IRQ to.

Returns

Syscall return tag

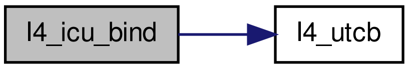
Examples:

examples/sys/isr/main.c.

Definition at line 425 of file icu.h.

References l4_utcb().

Here is the call graph for this function:



9.49.4.2 l4_msgtag_t l4_icu_unbind (l4_cap_idx_t *icu,* unsigned *irqnum,* l4_cap_idx_t *irq*) [inline]

Remove binding of an interrupt vector from the interrupt controller object.

Parameters

icu	ICU to use.
irqnum	IRQ vector at the ICU.
irq	IRQ object to remove from the ICU.

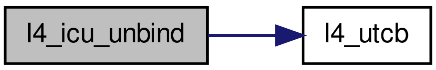
Returns

Syscall return tag

Definition at line 429 of file icu.h.

References l4_utcb().

Here is the call graph for this function:



9.49.4.3 l4_msgtag_t l4_icu_set_mode (l4_cap_idx_t *icu,* unsigned *irqnum,* l4_umword_t *mode*) [inline]

Set mode of interrupt.

Parameters

icu	ICU to use.

irqnum	IRQ vector at the ICU.
mode	Mode, see L4_irq_mode.

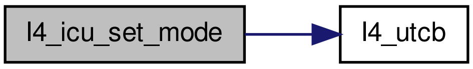
Returns

Syscall return tag

Definition at line 451 of file icu.h.

References l4_utcb().

Here is the call graph for this function:



9.49.4.4 l4_msgtag_t l4_icu_info (l4_cap_idx_t *icu*, l4_icu_info_t ∗ *info*) [inline]

Get info about capabilites of ICU.

Parameters

icu	ICU to use.
info	Pointer to an info structure to be filled with information.

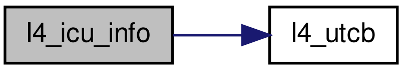
Returns

Syscall return tag

Definition at line 433 of file icu.h.

References l4_utcb().

Here is the call graph for this function:



9.49.4.5 l4_msgtag_t l4_icu_msi_info (l4_cap_idx_t *icu*, unsigned *irqnum*, l4_umword_t ∗ *msg*) [inline]

Get MSI info about IRQ.

Parameters

icu	ICU to use.
irqnum	IRQ vector at the ICU.
msg	Pointer to a word to receive the message that must be used for the PCI devices MSI message.

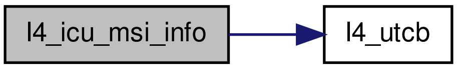
Returns

Syscall return tag

Definition at line 437 of file icu.h.

References l4_utcb().

Here is the call graph for this function:



9.49.4.6 **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 vector.

Parameters

icu	ICU to use.
irqnum	IRQ vector 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.

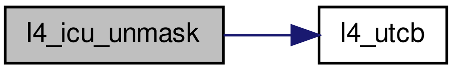
Returns

Syscall return tag, the error values therein are undefined because l4_icu_unmask() is a sender-only IPC

Definition at line 441 of file icu.h.

References l4_utcb().

Here is the call graph for this function:



9.49.4.7 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 vector.

Parameters

icu	ICU to use.
irqnum	IRQ vector 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.

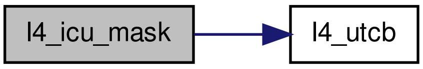
Returns

Syscall return tag

Definition at line 446 of file icu.h.

References l4_utcb().

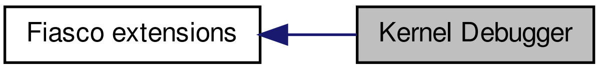
Here is the call graph for this function:



9.50 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

  The string name of 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.

9.50.1 Detailed Description

Kernel debugger related functionality.

Attention

This API is subject to change!

This is a debugging factility, any call to any function might be invalid. Do not rely on it in any real code.

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

9.50.2 Macro Definition Documentation

9.50.2.1 #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.

9.50.2.2 #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.

9.50.2.3 #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.

9.50.2.4 #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.

9.50.2.5 #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.

9.50.2.6 #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.

9.50.2.7 #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.

9.50.2.8 #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.

9.50.3 Function Documentation

9.50.3.1 l4_msgtag_t l4_debugger_set_object_name (l4_cap_idx_t *cap*, const char ∗ *name*) [inline]

The string name of kernel object.

Parameters

cap	Capability
name	Name

This is a debugging factility, the call might be invalid.

Examples:

examples/sys/aliens/main.c.

Definition at line 290 of file debugger.h.

References l4_utcb().

Here is the call graph for this function:

l4_debugger_set_object_name
l4_utcb

9.50.3.2 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

Returns

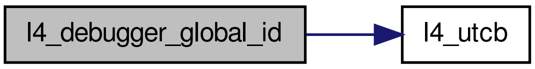
~0UL on non-valid capability, ID otherwise

This is a debugging factility, the call might be invalid.

Definition at line 297 of file debugger.h.

References l4_utcb().

Here is the call graph for this function:



9.50.3.3 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

Returns

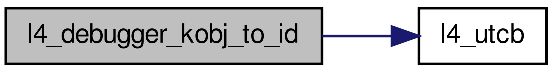
∼0UL on non-valid capability or invalid kobject pointer, ID otherwise

This is a debugging factility, the call might be invalid.

Definition at line 303 of file debugger.h.

References l4_utcb().

Here is the call graph for this function:



9.50.3.4 void outchar (char *c*) `[inline]`

Print character.

Parameters

c	Character

Definition at line 263 of file kdebug.h.

9.50.3.5 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.

9.50.3.6 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.

9.50.3.7 void outhex32 (int *number*) `[inline]`

Print 32 bit number (hexadecimal)

Parameters

number	32 bit number

Definition at line 303 of file kdebug.h.

9.50.3.8 void outhex20 (int *number*) `[inline]`

Print 20 bit number (hexadecimal)

Parameters

number	20 bit number

Definition at line 314 of file kdebug.h.

9.50.3.9 void outhex16 (int *number*) `[inline]`

Print 16 bit number (hexadecimal)

Parameters

number	16 bit number

Definition at line 325 of file kdebug.h.

9.50.3.10 void outhex12 (int *number*) `[inline]`

Print 12 bit number (hexadecimal)

Parameters

number	12 bit number

Definition at line 336 of file kdebug.h.

9.50.3.11 void outhex8 (int *number*) `[inline]`

Print 8 bit number (hexadecimal)

Parameters

number	8 bit number

Definition at line 347 of file kdebug.h.

9.50.3.12 void outdec (int *number*) [inline]

Print number (decimal)

Parameters

number	Number

Definition at line 358 of file kdebug.h.

9.50.3.13 char l4kd_inchar (void) [inline]

Read character from console, non blocking.

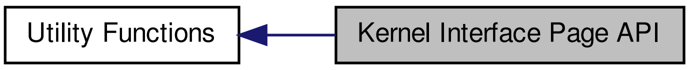
Returns

Input character, -1 if no character to read

Definition at line 369 of file kdebug.h.

9.51 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.

9.51.1 Detailed Description

9.51.2 Macro Definition Documentation

9.51.2.1 #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.

9.51.3 Function Documentation

9.51.3.1 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.

9.51.3.2 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.

9.51.3.3 unsigned long l4util_kip_kernel_abi_version (**l4_kernel_info_t** ∗)

Return kernel ABI version.

Returns

Kernel ABI version.

9.51.3.4 **l4_addr_t** l4util_memdesc_vm_high (**l4_kernel_info_t** ∗ *kinfo*)

Return end of virtual memory.

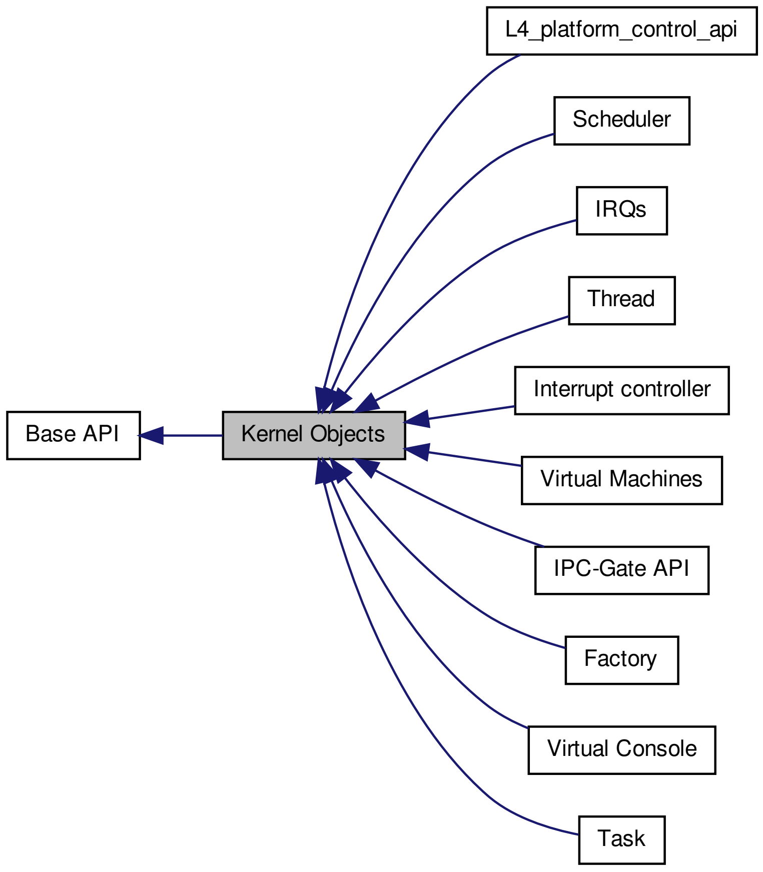
Returns

0 if memory descriptor could not be found, last address of address space otherwise

9.52 Kernel Objects

API of kernel objects.

Collaboration diagram for Kernel Objects:



Modules

- Factory

    A factory is used to create all kinds of kernel objects.
- IPC-Gate API

    Secure comminication object.
- IRQs

    The IRQ and ICU class.
- Interrupt controller

    The ICU class.
- L4_platform_control_api

    Class definition for the platform-control object.
- Scheduler

    Scheduler object.
- Task

Class definition of the Task kernel object.

- Thread

  Thread object.
- Virtual Console

  Virtual console for simple character based input and output.
- Virtual Machines

  Virtual Machine API.

9.52.1 Detailed Description

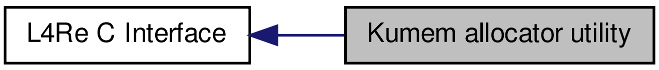
API of kernel objects.

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

9.53 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.

9.53.1 Detailed Description

Kumem allocator utility C interface.

9.53.2 Function Documentation

9.53.2.1 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

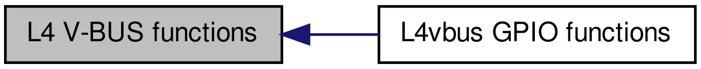
task	Task to use for allocation.
regmgr	Region manager to use for allocation.

Returns

0 for success, error code otherwise

9.54 L4 V-BUS functions

Collaboration diagram for L4 V-BUS functions:



Modules

- L4vbus GPIO functions

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 ACPI conforming or L4Io static name.
- 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 ∗res, int flags)

  Request a resource of a specific type.
- int l4vbus_release_resource (l4_cap_idx_t vbus, l4vbus_resource_t ∗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.

- 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.
- 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.
- 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.

9.54.1 Detailed Description

9.54.2 Function Documentation

9.54.2.1 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 ACPI conforming or L4Io static name.

Parameters

vbus	Capability of the system bus
parent	Handle to the parent to start the search

Return values

child	Handle to the found device

Parameters

hid	HID name of the device
depth	Depth to look for

Return values

devinfo	Device information structure (might be NULL)

Returns

0 on success, else failure

9.54.2.2 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

Return values

devinfo	device information (might be NULL)

Returns

0 on success, else failure

9.54.2.3 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

Return values

res_idx	Index of the resource, the number of resources is available in the devinfo from get device functions.

res	Descriptor of the resource

Returns

0 on success, else failure

9.54.2.4 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	V-BUS capability
dev	device handle for which the CID shall be tested
cid	the compatibility ID to test

Returns

1 when the given ID (*cid*) matches the given device (*dev*), 0 when the given ID does not match, <0 on error.

9.54.2.5 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

9.54.2.6 int l4vbus_request_resource (l4_cap_idx_t *vbus,* l4vbus_resource_t ∗ *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

9.54.2.7 int l4vbus_release_resource (l4_cap_idx_t *vbus*, l4vbus_resource_t ∗ *res*)

Release a previously requested resource.

Parameters

vbus	Capability of the system bus.
res	Descriptor of the resource.

Returns

0 on success, else failure

9.54.2.8 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

9.54.2.9 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.

Parameters

vbus	Capability of the system bus
handle	Device handle for the PCI root bridge
bus	Bus number
devfn	Device id (upper 16bit) and function (lower 16bit)
reg	Register in configuration space to read

Return values

value	Value that has been read

Parameters

width	Width to read in bits (e.g. 8, 16, 32)

Returns

0 on succes, else failure

9.54.2.10 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.

Parameters

vbus	Capability of the system bus
handle	Device handle of 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)

9.54.2.11 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.

Parameters

vbus	Capability of the system bus
handle	Device handle of 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)

Return values

trigger	True if interrupt is level-triggered
polarity	False if interrupt is of low polarity

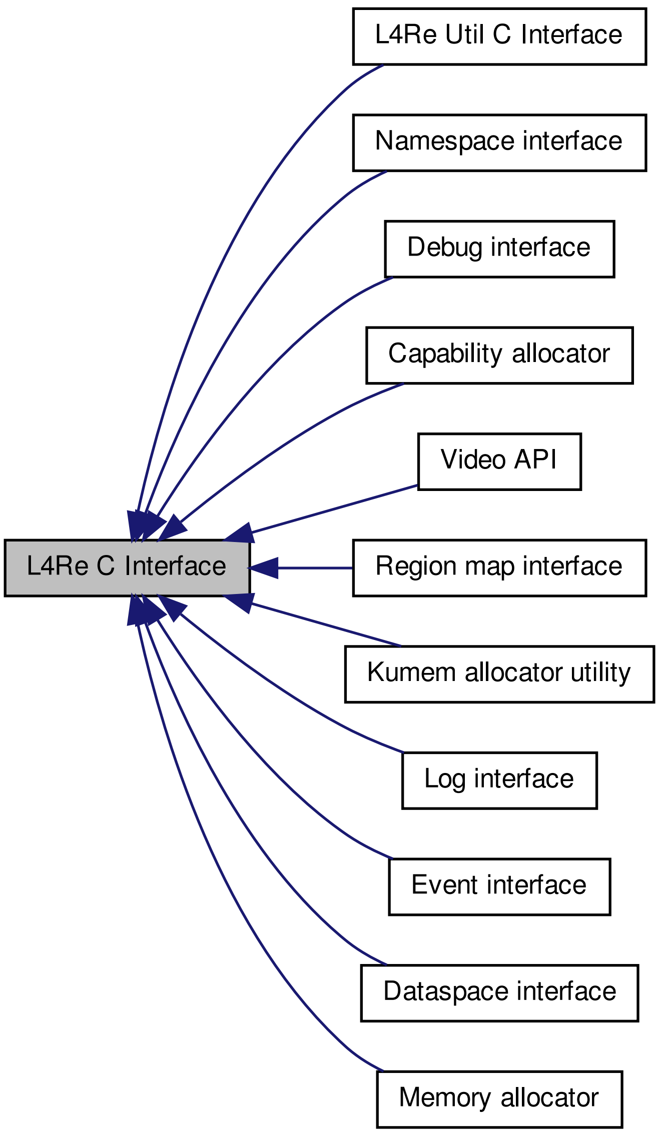
Returns

On success: Interrupt line to be used, else failure

9.55 L4Re C Interface

Documentation for the L4Re C Interface.

Collaboration diagram for L4Re C Interface:



Modules

- Capability allocator

    Capability allocator 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 L4_EXPORT l4re_inhibitor_acquire (l4_cap_idx_t cap, l4_umword_t id, char const ∗reason)

  (c) 2014 Steffen Liebergeld `steffen.liebergeld@kernkonzept.com`

9.55.1 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.

9.55.2 Function Documentation

9.55.2.1 long L4_EXPORT l4re_inhibitor_acquire (l4_cap_idx_t *cap,* l4_umword_t *id,* char const ∗ *reason*)

(c) 2014 Steffen Liebergeld `steffen.liebergeld@kernkonzept.com`

This file is licensed under the terms of the GNU Lesser General Public License 2.1. See the file COPYING-LGPL-2.1 for details.

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

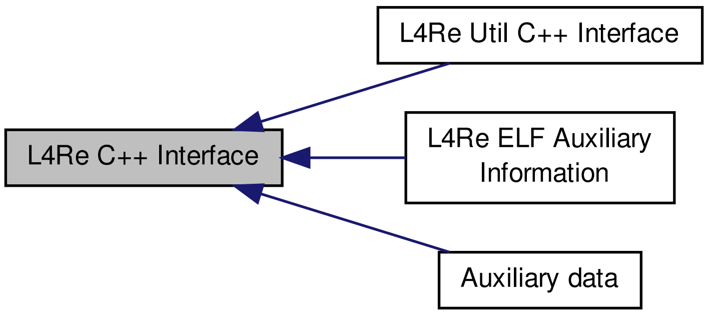
See also

L4Re::Inhibitor::acquire()

9.56 L4Re C++ Interface

Documentation of the L4 Runtime Environment C++ API.

Collaboration diagram for L4Re C++ Interface:



Modules

- Auxiliary data
- L4Re ELF Auxiliary Information

  API for embedding auxiliary information into binary programs.
- L4Re Util C++ Interface

  Documentation of the L4 Runtime Environment utility functionality in C++.

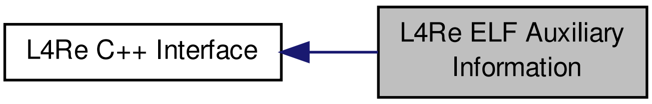
9.56.1 Detailed Description

Documentation of the L4 Runtime Environment C++ API.

9.57 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 }

9.57.1 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.

9.57.2 Macro Definition Documentation

9.57.2.1 #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.

9.57.2.2 #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.

9.57.3 Enumeration Type Documentation

9.57.3.1 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.

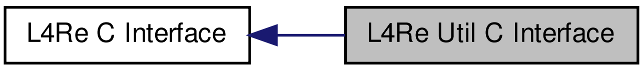
L4RE_ELF_AUX_T_KIP_ADDR Tag for descriptor that defines the KIP address for the binaries address space.

Definition at line 70 of file elf_aux.h.

9.58 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.

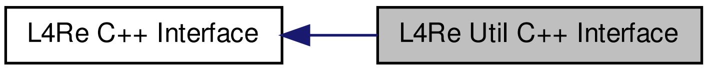
The interface functions closely align with the C++ functions and add no further functionalities.

For new programs it is advised to use the C++ interface.

9.59 L4Re Util C++ Interface

Documentation of the L4 Runtime Environment utility functionality in C++.

Collaboration diagram for L4Re Util C++ Interface:

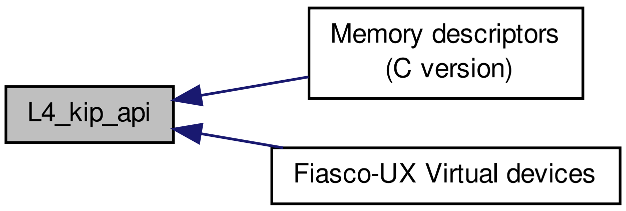


Documentation of the L4 Runtime Environment utility functionality in C++.

9.60 L4_kip_api

C interface for the Kernel Interface Page:
`#include <l4/sys/kip.h>`

Collaboration diagram for L4_kip_api:



Modules

- Fiasco-UX Virtual devices

  Virtual hardware devices, provided by Fiasco-UX.
- Memory descriptors (C version)

  C Interface for KIP memory descriptors.

Data Structures

- struct l4_kernel_info_t

  L4 Kernel Interface Page.

Macros

- #define L4_KERNEL_INFO_MAGIC (0x4BE6344CL) /∗ "L4µK" ∗/

  Kernel Info Page identifier ("L4µK").

Typedefs

- typedef struct l4_kernel_info_t l4_kernel_info_t

  L4 Kernel Interface Page.
- typedef struct l4_kernel_info_t l4_kernel_info_t

  L4 Kernel Interface Page.

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.

9.60.1 Detailed Description

C interface for the Kernel Interface Page:
`#include <l4/sys/kip.h>`

9.60.2 Function Documentation

9.60.2.1 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.

9.60.2.2 const char ∗ l4_kip_version_string (l4_kernel_info_t ∗ *kip*) `[inline]`

Get the kernel version string.

Parameters

kip	Kernel Info Page.

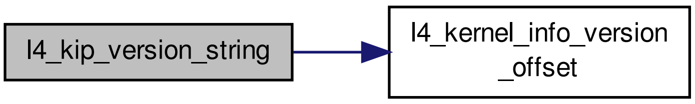
Returns

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:



9.60.2.3 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).

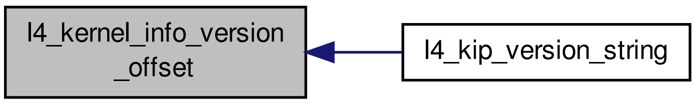
Returns

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:



9.60.2.4 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).

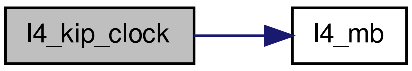
Returns

Value of the clock field in the KIP.

Definition at line 134 of file kip.h.

References l4_mb().

Here is the call graph for this function:



9.60.2.5 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).

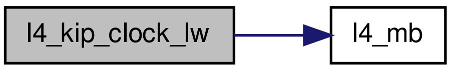
Returns

Lower machine word of clock value from the KIP.

Definition at line 155 of file kip.h.

References l4_mb().

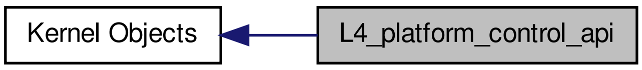
Here is the call graph for this function:



9.61 L4_platform_control_api

Class definition for the platform-control object.

Collaboration diagram for L4_platform_control_api:

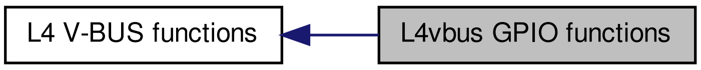


Class definition for the platform-control object.

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

9.62 L4vbus GPIO functions

Collaboration diagram for L4vbus GPIO functions:



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 outvalue)

    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.

9.62.1 Detailed Description

9.62.2 Enumeration Type Documentation

9.62.2.1 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.

9.62.2.2 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.

9.62.3 Function Documentation

9.62.3.1 int l4vbus_gpio_setup (**l4_cap_idx_t** *vbus,* l4vbus_device_handle_t *handle,* unsigned *pin,* unsigned *mode,* int *outvalue*)

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.
outvalue	Optional value to set the GPIO pin to if it is configured as an output pin

Returns

0 if OK, error code otherwise

9.62.3.2 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

9.62.3.3 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

9.62.3.4 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.

Return values

value	The configuration value.

Returns

0 if OK, error code otherwise

9.62.3.5 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.

9.62.3.6 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

9.62.3.7 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.
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

9.62.3.8 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.
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

9.62.3.9 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.

Return values

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

9.62.3.10 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

9.62.3.11 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.

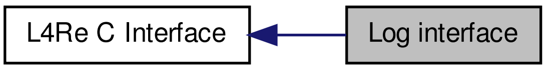
Returns

IRQ number if OK, negative error code otherwise

9.63 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.

9.63.1 Detailed Description

Log C interface.

9.63.2 Function Documentation

9.63.2.1 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

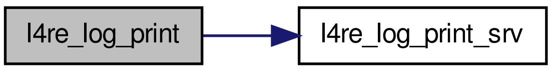
See also

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:



9.63.2.2 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

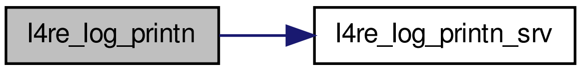
See also

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:



9.63.2.3 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

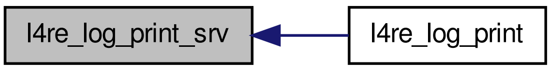
0 for success, <0 on error

See also

L4Re::Log::print

Referenced by l4re_log_print().

Here is the caller graph for this function:



9.63.2.4 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

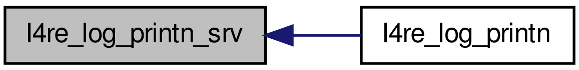
0 for success, <0 on error

See also

L4Re::Log::printn

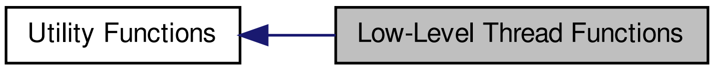
Referenced by l4re_log_printn().

Here is the caller graph for this function:



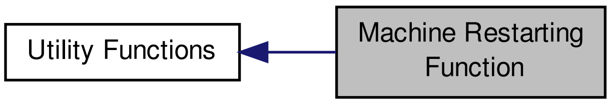
9.64 Low-Level Thread Functions

Collaboration diagram for Low-Level Thread Functions:



9.65 Machine Restarting Function

Collaboration diagram for Machine Restarting Function:



Functions

- void l4util_reboot (void))

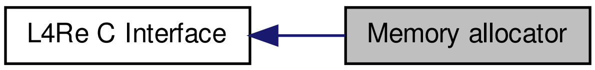
  Machine reboot.

9.65.1 Detailed Description

9.66 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 (unsigned long size, l4re_ds_t const mem, unsigned long flags) L4_NOTHROW

    Allocate memory.
- long l4re_ma_alloc_align (unsigned long size, l4re_ds_t const mem, unsigned long flags, unsigned long align) L4_NOTHROW

    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, unsigned 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.

9.66.1 Detailed Description

Memory allocator C interface.

9.66.2 Enumeration Type Documentation

9.66.2.1 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.

9.66.3 Function Documentation

9.66.3.1 long l4re_ma_alloc (unsigned 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.

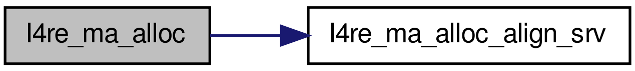
Examples:

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:



9.66.3.2 long l4re_ma_alloc_align (unsigned 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.

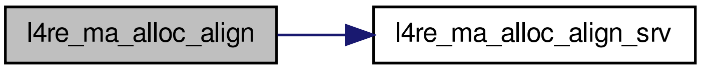
Note

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:



9.66.3.3 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.

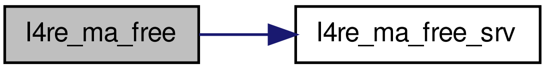
Examples:

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:



9.66.3.4 long l4re_ma_alloc_align_srv (l4_cap_idx_t *srv*, unsigned 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

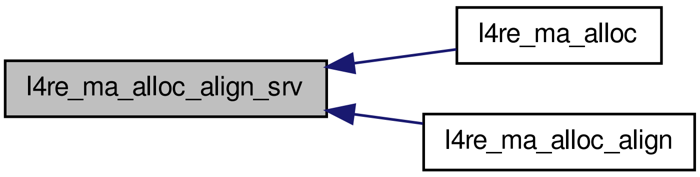
See also

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:



9.66.3.5 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

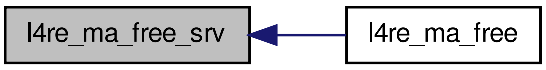
0 on success, <0 on error

See also

L4Re::Mem_alloc::free

Referenced by l4re_ma_free().

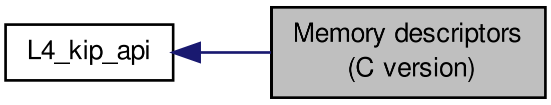
Here is the caller graph for this function:



9.67 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.

9.67.1 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).

9.67.2 Typedef Documentation

9.67.2.1 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.

9.67.3 Enumeration Type Documentation

9.67.3.1 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.

9.67.4 Function Documentation

9.67.4.1 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.

9.67.4.2 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.

9.67.4.3 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.

9.67.4.4 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.

9.67.4.5 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.

9.67.4.6 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.

9.67.4.7 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

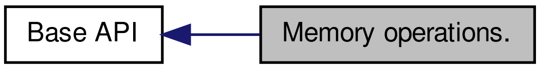
1 if region is virtual memory, 0 if region is physical memory

Definition at line 228 of file memdesc.h.

9.68 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_WID↩ TH_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.

9.68.1 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>
```

9.68.2 Enumeration Type Documentation

9.68.2.1 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.

9.68.3 Function Documentation

9.68.3.1 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.

9.68.3.2 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.

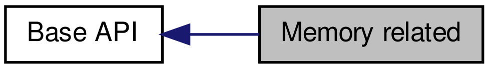
Upon an given invalid address or invalid width value the function does nothing.

Definition at line 147 of file mem_op.h.

9.69 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_addr_t address, unsigned char bits) L4_NOTHROW

    Round address up to the next flex page with bits size.

9.69.1 Detailed Description

Memory related constants, data types and functions.

9.69.2 Macro Definition Documentation

9.69.2.1 #define L4_PAGEMASK

Mask for the page number.

Note

The most significant bits are set.

Definition at line 285 of file consts.h.

Referenced by l4_round_page(), and l4_trunc_page().

9.69.2.2 #define L4_LOG2_PAGESIZE

Number of bits used for page offset.

Size of page in log2.

Definition at line 294 of file consts.h.

9.69.2.3 #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 303 of file consts.h.

9.69.2.4 #define L4_SUPERPAGEMASK

Mask for the number of a large page.

Note

The most significant bits are set.

Definition at line 312 of file consts.h.

9.69.2.5 #define L4_LOG2_SUPERPAGESIZE

Number of bits used as offset for a large page.

Size of large page in log2

Definition at line 320 of file consts.h.

9.69.3 Enumeration Type Documentation

9.69.3.1 enum l4_addr_consts_t

Address related constants.

Enumerator

L4_INVALID_ADDR Invalid address.

Definition at line 368 of file consts.h.

9.69.4 Function Documentation

9.69.4.1 l4_addr_t l4_trunc_page (l4_addr_t *address*) [inline]

Round an address down to the next lower page boundary.

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.

9.69.4.2 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.

9.69.4.3 l4_addr_t l4_round_page (l4_addr_t *address*) [inline]

Round address up to the next page.

Parameters

address	The address to round up.

Definition at line 350 of file consts.h.

References L4_PAGEMASK, and L4_PAGESIZE.

9.69.4.4 l4_addr_t l4_round_size (l4_addr_t *address*, unsigned char *bits*) [inline]

Round address up to the next flex page with *bits* size.

Parameters

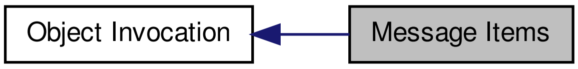
address	The address to round up to the next flex page.
bits	The size of the flex page (log2).

Definition at line 361 of file consts.h.

9.70 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.

9.70.1 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.

9.70.2 Enumeration Type Documentation

9.70.2.1 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 193 of file consts.h.

9.70.3 Function Documentation

9.70.3.1 **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

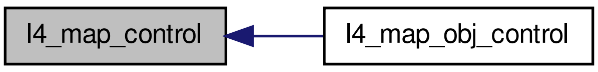
The value to be used as first word in a map item for memory.

Definition at line 490 of file __l4_fpage.h.

References L4_ITEM_MAP.

Referenced by l4_map_obj_control().

Here is the caller graph for this function:



9.70.3.2 **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.

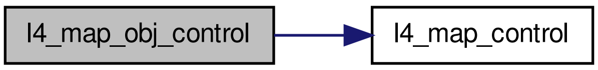
Returns

The value to be used as first word in a map item for kernel objects or IO-ports.

Definition at line 497 of file __l4_fpage.h.

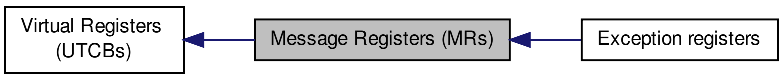
References l4_map_control().

Here is the call graph for this function:



9.71 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

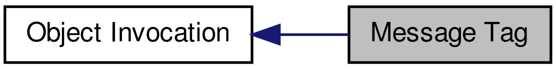
    Encapsulation of the message-register block in the UTCB.

9.71.1 Detailed Description

9.72 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_msgtag_protocol {
  L4_PROTO_NONE = 0, L4_PROTO_ALLOW_SYSCALL = 1, L4_PROTO_PF_EXCEPTION = 1, L4_PRO↩
  TO_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_PR↩
  OTO_TASK = -11L,
  L4_PROTO_THREAD = -12L, L4_PROTO_LOG = -13L, L4_PROTO_SCHEDULER = -14L, L4_PROTO_↩
  FACTORY = -15L,
  L4_PROTO_VM = -16L , L4_PROTO_META = -21L }

  Message tag for IPC operations.
- enum l4_msgtag_flags {
  L4_MSGTAG_ERROR, L4_MSGTAG_XCPU, L4_MSGTAG_TRANSFER_FPU, L4_MSGTAG_SCHEDUL↩
  E,
  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.

9.72.1 Detailed Description

API related to the message tag data type.

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

9.72.2 Typedef Documentation

9.72.2.1 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.

9.72.3 Enumeration Type Documentation

9.72.3.1 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_META Meta information protocol.

Definition at line 49 of file types.h.

9.72.3.2 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 89 of file types.h.

9.72.4 Function Documentation

9.72.4.1 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

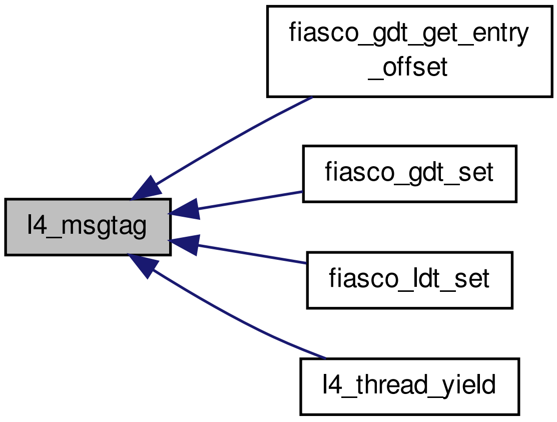
Examples:

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 366 of file types.h.

Referenced by fiasco_gdt_get_entry_offset(), fiasco_gdt_set(), fiasco_ldt_set(), and l4_thread_yield().

Here is the caller graph for this function:



9.72.4.2 long l4_msgtag_label (l4_msgtag_t *t*) [inline]

Get the protocol of tag.

Parameters

t	The tag

Returns

Label

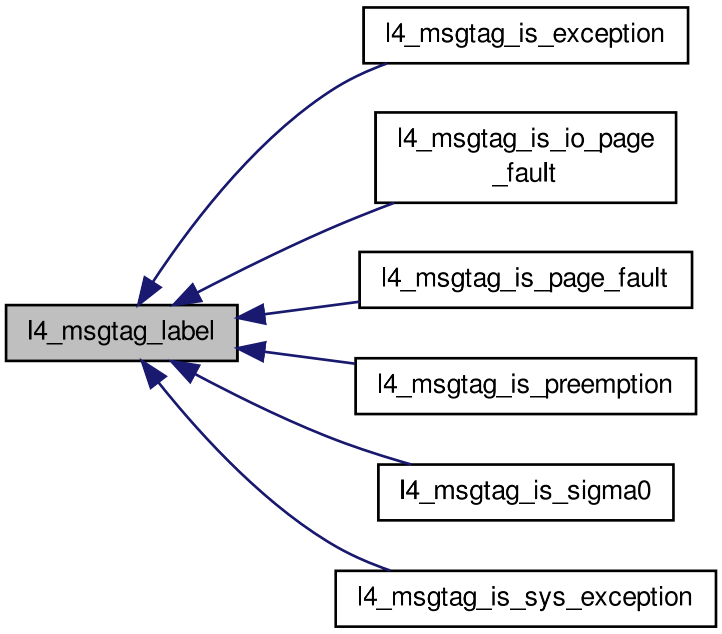
Examples:

examples/sys/singlestep/main.c, and examples/sys/start-with-exc/main.c.

Definition at line 377 of file types.h.

Referenced by 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:



9.72.4.3 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 381 of file types.h.

9.72.4.4 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 385 of file types.h.

9.72.4.5 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 389 of file types.h.

9.72.4.6 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:

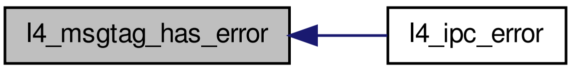
examples/sys/aliens/main.c, examples/sys/singlestep/main.c, and examples/sys/utcb-ipc/main.c.

Definition at line 394 of file types.h.

References L4_MSGTAG_ERROR.

Referenced by l4_ipc_error().

Here is the caller graph for this function:



9.72.4.7 unsigned l4_msgtag_is_page_fault (l4_msgtag_t *t*) [inline]

Test for page-fault protocol.

Parameters

t	the tag

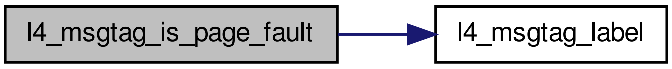
Returns

Boolean value

Definition at line 399 of file types.h.

References l4_msgtag_label(), and L4_PROTO_PAGE_FAULT.

Here is the call graph for this function:



9.72.4.8 unsigned l4_msgtag_is_preemption (l4_msgtag_t *t*) [inline]

Test for preemption protocol.

Parameters

t	the tag

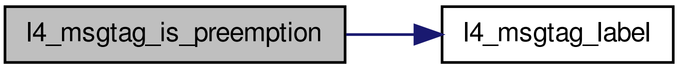
Returns

Boolean value

Definition at line 402 of file types.h.

References l4_msgtag_label(), and L4_PROTO_PREEMPTION.

Here is the call graph for this function:



9.72.4.9 unsigned l4_msgtag_is_sys_exception (l4_msgtag_t *t*) [inline]

Test for system-exception protocol.

Parameters

t	the tag

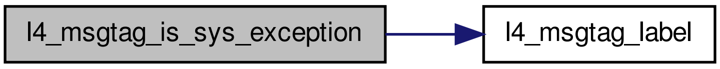
Returns

Boolean value

Definition at line 405 of file types.h.

References l4_msgtag_label(), and L4_PROTO_SYS_EXCEPTION.

Here is the call graph for this function:



9.72.4.10 unsigned l4_msgtag_is_exception (l4_msgtag_t *t*) [inline]

Test for exception protocol.

Parameters

t	the tag

Returns

Boolean value

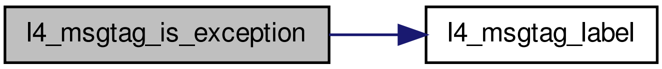
Examples:

examples/sys/aliens/main.c, examples/sys/singlestep/main.c, and examples/sys/start-with-exc/main.c.

Definition at line 408 of file types.h.

References l4_msgtag_label(), and L4_PROTO_EXCEPTION.

Here is the call graph for this function:



9.72.4.11 unsigned l4_msgtag_is_sigma0 (l4_msgtag_t *t*) [inline]

Test for sigma0 protocol.

Parameters

t	the tag

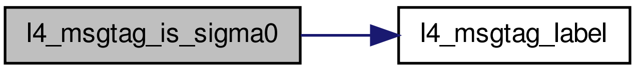
Returns

Boolean value

Definition at line 411 of file types.h.

References l4_msgtag_label(), and L4_PROTO_SIGMA0.

Here is the call graph for this function:



9.72.4.12 unsigned l4_msgtag_is_io_page_fault (**l4_msgtag_t** *t*) [inline]

Test for IO-page-fault protocol.

Parameters

t	the tag

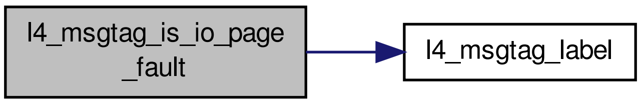
Returns

Boolean value

Definition at line 414 of file types.h.

References l4_msgtag_label(), and L4_PROTO_IO_PAGE_FAULT.

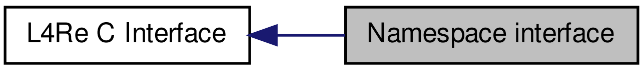
Here is the call graph for this function:



9.73 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
- long l4re_ns_register_obj_srv (l4re_namespace_t srv, char const ∗name, l4_cap_idx_t const obj, unsigned flags) L4_NOTHROW

9.73.1 Detailed Description

Namespace C interface.

9.73.2 Enumeration Type Documentation

9.73.2.1 enum **l4re_ns_register_flags**

Namespace register flags.

See also

L4Re::Namespace::Register_flags

Definition at line 39 of file namespace.h.

9.73.3 Function Documentation

9.73.3.1 long l4re_ns_query_to_srv (**l4re_namespace_t** *srv,* char const ∗ *name,* **l4_cap_idx_t** const *cap,* int *timeout*)

Returns

0 on success, <0 on error

See also

L4Re::Namespace::query

9.73.3.2 long l4re_ns_register_obj_srv (l4re_namespace_t *srv,* char const ∗ *name,* l4_cap_idx_t const *obj,* unsigned *flags*)

Returns

0 on success, <0 on error

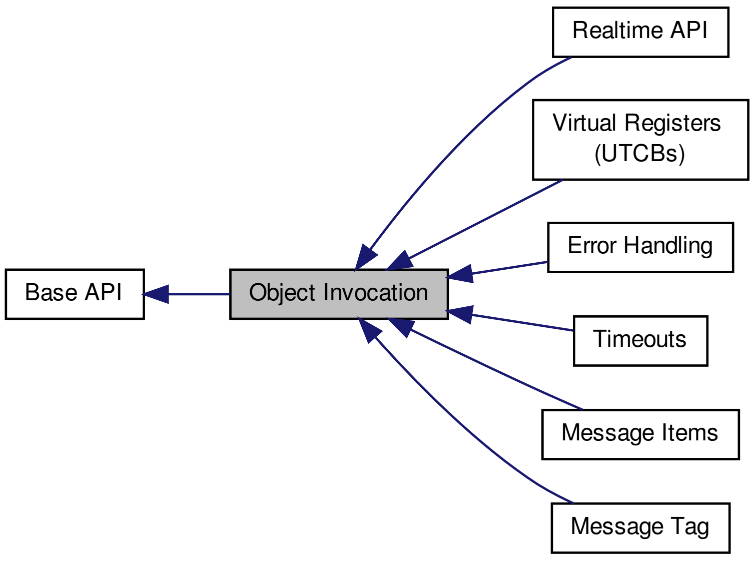
See also

L4Re::Namespace::register_obj

9.74 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_N↩
OTHROW

  *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_N↩
OTHROW

  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_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_NOTH↩
ROW

  Add a flex-page to be sent to the UTCB.

9.74.1 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

9.74.2 Enumeration Type Documentation

9.74.2.1 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_REC↩V) 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 45 of file consts.h.

9.74.3 Function Documentation

9.74.3.1 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().

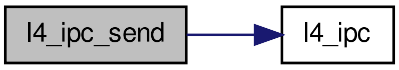
Examples:

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

Definition at line 93 of file ipc.h.

References l4_ipc(), L4_SYSF_SEND, and l4_msgtag_t::raw.

Here is the call graph for this function:



9.74.3.2 **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().

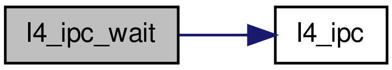
Examples:

examples/sys/ipc/ipc_example.c.

Definition at line 101 of file ipc.h.

References L4_INVALID_CAP, l4_ipc(), L4_SYSF_WAIT, and l4_msgtag_t::raw.

Here is the call graph for this function:



9.74.3.3 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:

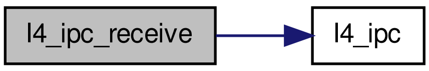
examples/sys/aliens/main.c, examples/sys/singlestep/main.c, examples/sys/start-with-exc/main.c, and examples/sys/utcb-ipc/main.c.

Definition at line 110 of file ipc.h.

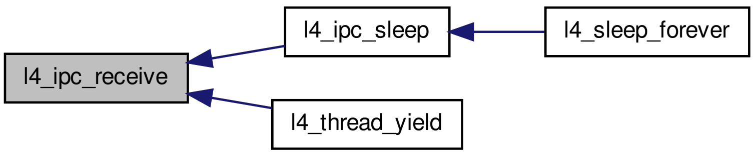
References l4_ipc(), L4_SYSF_RECV, l4_timeout_t::raw, and l4_msgtag_t::raw.

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:



9.74.3.4 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:

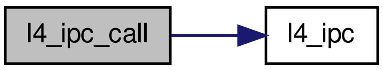
examples/sys/aliens/main.c, examples/sys/ipc/ipc_example.c, and examples/sys/singlestep/main.c.

Definition at line 68 of file ipc.h.

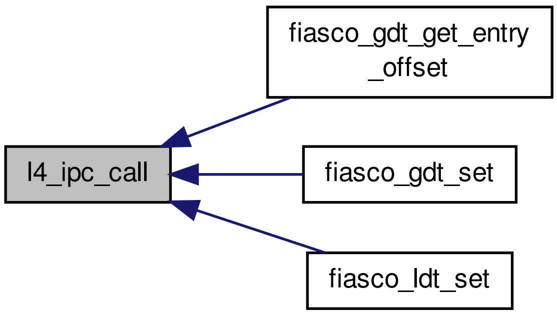
References l4_ipc(), L4_SYSF_CALL, and l4_msgtag_t::raw.

Referenced by fiasco_gdt_get_entry_offset(), fiasco_gdt_set(), and fiasco_ldt_set().

Here is the call graph for this function:



Here is the caller graph for this function:



9.74.3.5 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.

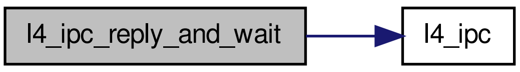
Examples:

examples/sys/ipc/ipc_example.c.

Definition at line 76 of file ipc.h.

References L4_INVALID_CAP, l4_ipc(), L4_SYSF_REPLY_AND_WAIT, and l4_msgtag_t::raw.

Here is the call graph for this function:



9.74.3.6 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.

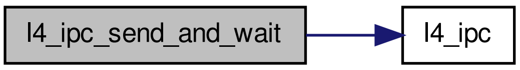
Note

This is a special-purpose operation and shall not be used in general applications.

Definition at line 84 of file ipc.h.

References l4_ipc(), L4_SYSF_SEND_AND_WAIT, and l4_msgtag_t::raw.

Here is the call graph for this function:



9.74.3.7 **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

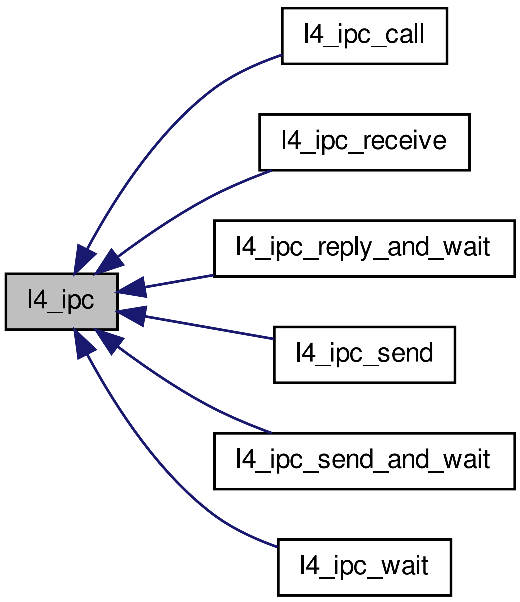
return tag

Definition at line 34 of file ipc.h.

References l4_timeout_t::raw, and l4_msgtag_t::raw.

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:



9.74.3.8 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()).

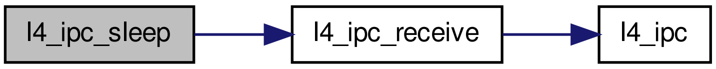
The invoking thread waits until the timeout is expired or the wait was aborted by another thread by l4_thread_ex_↩ regs().

Definition at line 28 of file ipc-impl.h.

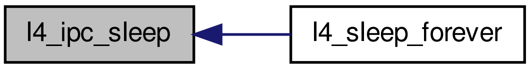
References L4_INVALID_CAP, and l4_ipc_receive().

Referenced by l4_sleep_forever().

Here is the call graph for this function:



Here is the caller graph for this function:



9.74.3.9 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.

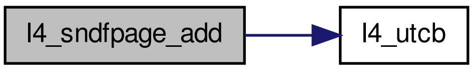
Returns

0 on success, negative error code otherwise

Definition at line 486 of file ipc.h.

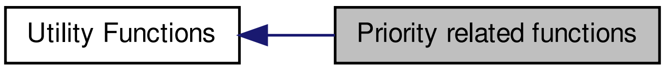
References l4_utcb().

Here is the call graph for this function:



9.75 Priority related functions

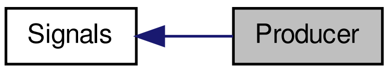
Collaboration diagram for Priority related functions:



9.75.1 Detailed Description

9.76 Producer

Collaboration diagram for Producer:



Functions

- long l4shmc_trigger (l4shmc_signal_t ∗signal)

  Trigger a signal.

9.76.1 Detailed Description

9.76.2 Function Documentation

9.76.2.1 long l4shmc_trigger (l4shmc_signal_t ∗ *signal*) `[inline]`

Trigger a signal.

Parameters

signal	Signal to trigger.

Returns

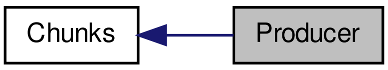
0 on success, <0 on error

Examples:

examples/libs/shmc/prodcons.c.

9.77 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.

9.77.1 Detailed Description

9.77.2 Function Documentation

9.77.2.1 long l4shmc_chunk_try_to_take (l4shmc_chunk_t ∗ *chunk*) [inline]

Try to mark chunk busy.

Parameters

chunk	chunk to mark.

Returns

0 if chunk could be taken, <0 if not (try again then)

Examples:

examples/libs/shmc/prodcons.c.

9.77.2.2 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.

Returns

0 on success, <0 on error

9.77.2.3 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.

Returns

0 on success, <0 on error

Examples:

examples/libs/shmc/prodcons.c.

9.77.2.4 long l4shmc_is_chunk_clear (l4shmc_chunk_t ∗ *chunk*) [inline]

Check whether chunk is free.

Parameters

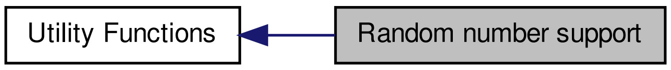
chunk	Chunk to check.

Returns

0 on success, <0 on error

9.78 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.

9.78.1 Detailed Description

9.78.2 Function Documentation

9.78.2.1 l4_uint32_t l4util_rand (void)

Deliver next random number.

Returns

A new random number

9.78.2.2 void l4util_srand (l4_uint32_t *seed*)

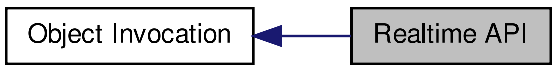
Initialize random number generator.

Parameters

seed	Value to initialize

9.79 Realtime API

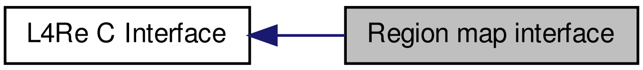
Collaboration diagram for Realtime API:



9.80 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.

9.80.1 Detailed Description

Region map C interface.

9.80.2 Enumeration Type Documentation

9.80.2.1 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.

9.80.3 Function Documentation

9.80.3.1 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

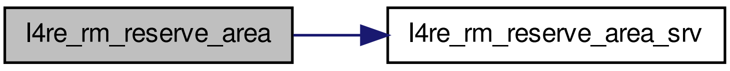
L4Re::Rm::reserve_area

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:



9.80.3.2 int l4re_rm_free_area (l4_addr_t *addr*) [inline]

Returns

0 on success, <0 on error

See also

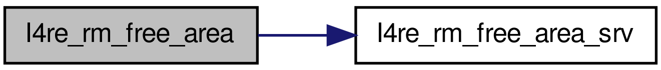
L4Re::Rm::free_area

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:



9.80.3.3 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]

Returns

0 on success, <0 on error

See also

L4Re::Rm::attach

This function is using the L4::Env::env()->rm() service.

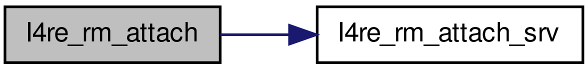
Examples:

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:



9.80.3.4 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

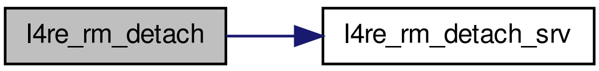
L4Re::Rm::detach

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:



9.80.3.5 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.

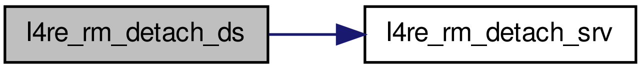
Examples:

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:



9.80.3.6 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

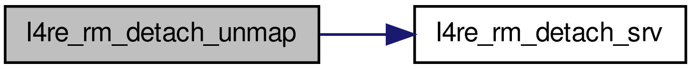
L4Re::Rm::detach

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:



9.80.3.7 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

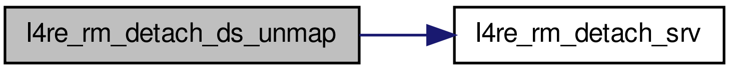
L4Re::Rm::detach

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:



9.80.3.8 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

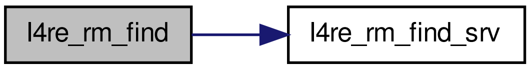
See also

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:



9.80.3.9 void l4re_rm_show_lists (void) [inline]

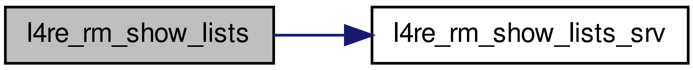
Dump region map internal data structures.

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:



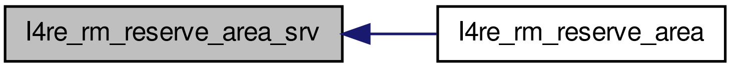
9.80.3.10 int l4re_rm_reserve_area_srv (l4_cap_idx_t *rm*, l4_addr_t ∗ *start*, unsigned long *size*, unsigned *flags*, unsigned char *align*)

See also

L4Re::Rm::reserve_area

Referenced by l4re_rm_reserve_area().

Here is the caller graph for this function:



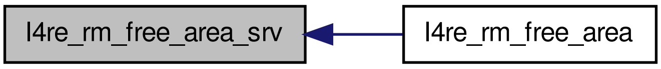
9.80.3.11 int l4re_rm_free_area_srv (**l4_cap_idx_t** *rm,* **l4_addr_t** *addr*)

See also

L4Re::Rm::free_area

Referenced by l4re_rm_free_area().

Here is the caller graph for this function:



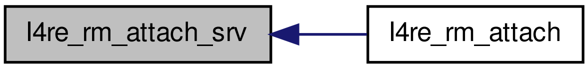
9.80.3.12 int l4re_rm_attach_srv (**l4_cap_idx_t** *rm,* void ∗∗ *start,* unsigned long *size,* unsigned long *flags,* **l4re_ds_t** const *mem,* **l4_addr_t** *offs,* unsigned char *align*)

See also

L4Re::Rm::attach

Referenced by l4re_rm_attach().

Here is the caller graph for this function:



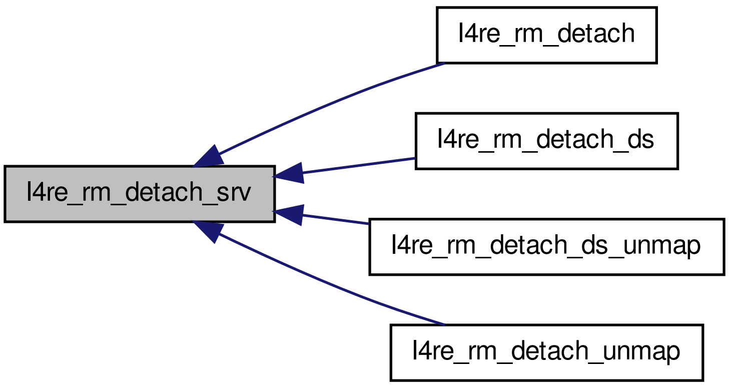
9.80.3.13 int l4re_rm_detach_srv (l4_cap_idx_t *rm,* l4_addr_t *addr,* l4re_ds_t ∗ *ds,* l4_cap_idx_t *task*)

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:



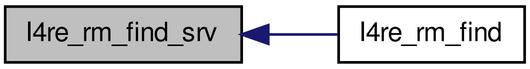
9.80.3.14 int l4re_rm_find_srv (l4_cap_idx_t *rm,* l4_addr_t ∗ *addr,* unsigned long ∗ *size,* l4_addr_t ∗ *offset,* unsigned ∗ *flags,* l4re_ds_t ∗ *m*)

See also

L4Re::Rm::find

Referenced by l4re_rm_find().

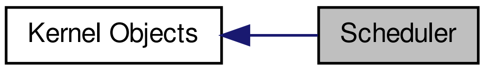
Here is the caller graph for this function:



9.81 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_sched_cpu_set_t l4_sched_cpu_set (l4_umword_t offset, unsigned char granularity, l4_umword_t map L4_DEFAULT_PARAM(1)) L4_NOTHROW
- l4_msgtag_t l4_scheduler_info (l4_cap_idx_t scheduler, l4_umword_t ∗cpu_max, l4_sched_cpu_set_t ∗cpus) L4_NOTHROW

  Get scheduler information.
- l4_sched_param_t l4_sched_param (unsigned prio, l4_cpu_time_t quantum L4_DEFAULT_PARAM(0)) L4↩
_NOTHROW

  Construct scheduler parameter.
- l4_msgtag_t l4_scheduler_run_thread (l4_cap_idx_t scheduler, l4_cap_idx_t thread, l4_sched_param_↩
t const ∗sp) L4_NOTHROW

  Run a thread on a Scheduler.
- l4_msgtag_t l4_scheduler_idle_time (l4_cap_idx_t scheduler, l4_sched_cpu_set_t const ∗cpus) L4_NOTH↩
ROW

  Query idle time of a CPU, in µs.

- int l4_scheduler_is_online (l4_cap_idx_t scheduler, l4_umword_t cpu) L4_NOTHROW

  Query if a CPU is online.

9.81.1 Detailed Description

Scheduler object.

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

9.81.2 Enumeration Type Documentation

9.81.2.1 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 185 of file scheduler.h.

9.81.3 Function Documentation

9.81.3.1 l4_sched_cpu_set_t l4_sched_cpu_set (l4_umword_t *offset*, unsigned char *granularity*, l4_umword_t map *L4_DEFAULT_PARAM1*) [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.

9.81.3.2 l4_msgtag_t l4_scheduler_info (l4_cap_idx_t *scheduler*, l4_umword_t ∗ *cpu_max*, l4_sched_cpu_set_t ∗ *cpus*) [inline]

Get scheduler information.

Parameters

scheduler	Scheduler object.

Return values

cpu_max	maximum number of CPUs ever available.

Parameters

cpus	*cpus.offset* is first CPU of interest. *cpus.granularity* (see l4_sched_cpu_set_t).

Return values

cpus	*cpus.map* Bitmap of online CPUs.

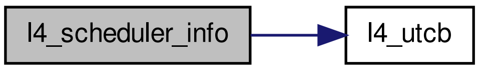
Returns

0 on success, <0 error code otherwise.

Definition at line 284 of file scheduler.h.

References l4_utcb().

Here is the call graph for this function:



9.81.3.3 l4_msgtag_t l4_scheduler_run_thread (l4_cap_idx_t *scheduler,* l4_cap_idx_t *thread,* l4_sched_param_t const ∗ *sp*) [inline]

Run a thread on a Scheduler.

Parameters

scheduler	Scheduler object.
thread	Thread to run.
sp	Scheduling parameters.

Returns

0 on success, <0 error code otherwise.

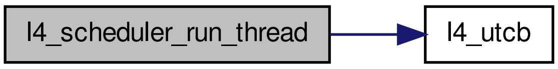
Examples:

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 291 of file scheduler.h.

References l4_utcb().

Here is the call graph for this function:



9.81.3.4 **l4_msgtag_t** l4_scheduler_idle_time (**l4_cap_idx_t** *scheduler,* **l4_sched_cpu_set_t** const ∗ *cpus*) [inline]

Query idle time of a CPU, in µs.

Parameters

scheduler	Scheduler object.
cpus	Set of CPUs to query.

The consumed time is returned as l4_kernel_clock_t at UTCB message register 0.

Definition at line 298 of file scheduler.h.

References l4_utcb().

Here is the call graph for this function:

l4_scheduler_idle_time
l4_utcb

9.81.3.5 int l4_scheduler_is_online (**l4_cap_idx_t** *scheduler,* **l4_umword_t** *cpu*) [inline]

Query if a CPU is online.

Parameters

scheduler	Scheduler object.
cpu	CPU number.

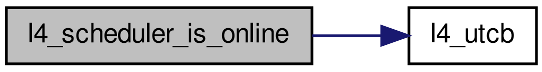
Returns

true if online, false if not (or any other query error).

Definition at line 304 of file scheduler.h.

References l4_utcb().

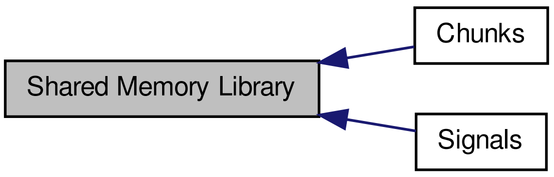
Here is the call graph for this function:



9.82 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.

9.82.1 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, reponsible 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).

9.82.2 Function Documentation

9.82.2.1 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.

Returns

0 on success, <0 on error

Examples:

examples/libs/shmc/prodcons.c.

9.82.2.2 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.

Return values

shmarea	Pointer to shared memory area descriptor to be filled with information for the shared memory area.

Returns

0 on success, <0 on error

Examples:

examples/libs/shmc/prodcons.c.

9.82.2.3 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.

Return values

shmarea	Pointer to shared memory area descriptor to be filled with information for the shared memory area.

Returns

0 on success, <0 on error

9.82.2.4 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.

9.82.2.5 long l4shmc_area_size (l4shmc_area_t ∗ *shmarea*) [inline]

Get size of shared memory area.

Parameters

shmarea	Shared memory area.

Returns

<0 on error, otherwise: size of the shared memory area

9.82.2.6 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.

Returns

<0 on error, otherwise: free capacity in the area.

9.82.2.7 long l4shmc_area_overhead (void)

Get memory overhead per area that is not available for chunks.

Returns

size of the overhead in bytes

9.82.2.8 long l4shmc_chunk_overhead (void)

Get memory overhead required in addition to the chunk capacity for adding one chunk.

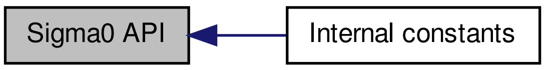
Returns

size of the overhead in bytes

9.83 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.

9.83.1 Detailed Description

Sigma0 API bindings.

Convenience bindings for the Sigma0 protocol.

9.83.2 Enumeration Type Documentation

9.83.2.1 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.

9.83.3 Function Documentation

9.83.3.1 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.

9.83.3.2 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()).

9.83.3.3 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()).

9.83.3.4 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()).

9.83.3.5 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()).

9.83.3.6 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.

9.83.3.7 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.

9.83.3.8 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.

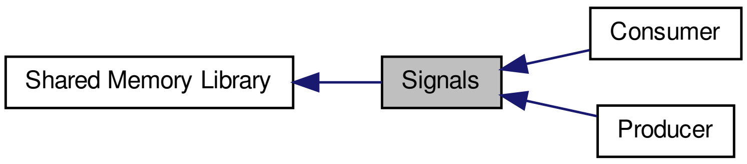
Returns

a string containing the error message.

Definition at line 208 of file sigma0.h.

9.84 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.

9.84.1 Detailed Description

9.84.2 Function Documentation

9.84.2.1 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.

Return values

signal	Signal structure to fill in.

Returns

0 on success, <0 on error

Examples:

examples/libs/shmc/prodcons.c.

9.84.2.2 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.

Return values

signal	Signal data structure to fill.

Returns

0 on success, <0 on error

9.84.2.3 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.

Return values

signal	Signal data structure to fill.

Returns

0 on success, <0 on error

Examples:

examples/libs/shmc/prodcons.c.

9.84.2.4 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

9.84.2.5 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.

9.84.2.6 long l4shmc_check_magic (l4shmc_chunk_t ∗ *chunk*) [inline]

Check magic value of a chunk.

Parameters

chunk	Chunk.

Returns

True if chunk is ok (magic value valid), false if not.

9.85 Small C++ Template Library

Data Structures

- class L4::Alloc_list

  A simple list-based allocator.
- class L4::String

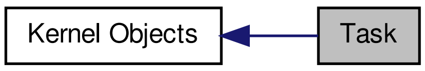
  A null-terminated string container class.

9.85.1 Detailed Description

9.86 Task

Class definition 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_NOTHR↩ OW

  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.

9.86.1 Detailed Description

Class definition of the Task kernel object.

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

The L4 task class represents a combination of the address spaces provided by the L4 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.

A task object can be created using a Factory, see Factory (l4_factory_create_task()).

9.86.2 Enumeration Type Documentation

9.86.2.1 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 163 of file consts.h.

9.86.3 Function Documentation

9.86.3.1 **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

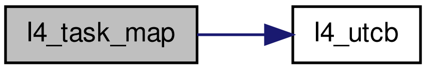
Syscall return tag

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 359 of file task.h.

References l4_utcb().

Here is the call graph for this function:



9.86.3.2 **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).

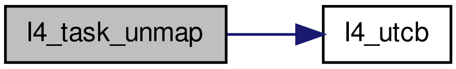
Note

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 366 of file task.h.

References l4_utcb().

Here is the call graph for this function:



9.86.3.3 **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.

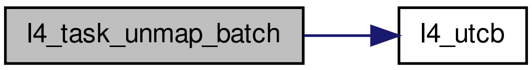
Note

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 373 of file task.h.

References l4_utcb().

Here is the call graph for this function:



9.86.3.4 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

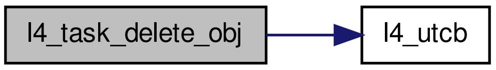
The object will be deleted if the obj has sufficient rights. No error will be reported if the rights are insufficient, however, the capability is removed in all cases.

This operation calls l4_task_unmap() with L4_FP_DELETE_OBJ.

Definition at line 389 of file task.h.

References l4_utcb().

Here is the call graph for this function:



9.86.3.5 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

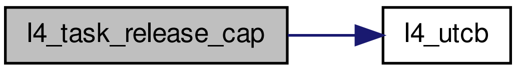
Syscall return tag

This operation unmaps the capability from the specified task.

Definition at line 404 of file task.h.

References l4_utcb().

Here is the call graph for this function:



9.86.3.6 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

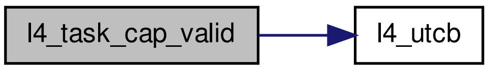
Returns

label contains >0 if valid, 0 if invalid

Definition at line 410 of file task.h.

References l4_utcb().

Here is the call graph for this function:



9.86.3.7 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

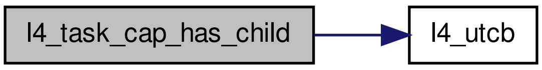
Returns

label contains 1 if it has at least one child, 0 if not or invalid

Definition at line 416 of file task.h.

References l4_utcb().

Here is the call graph for this function:



9.86.3.8 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

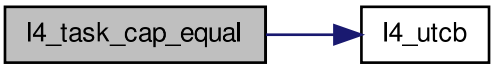
Returns

label contains 1 if equal, 0 if not equal

Definition at line 422 of file task.h.

References l4_utcb().

Here is the call graph for this function:



9.86.3.9 **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.

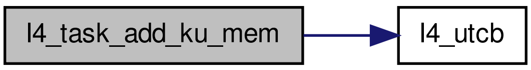
Returns

Syscall return tag

Definition at line 429 of file task.h.

References l4_utcb().

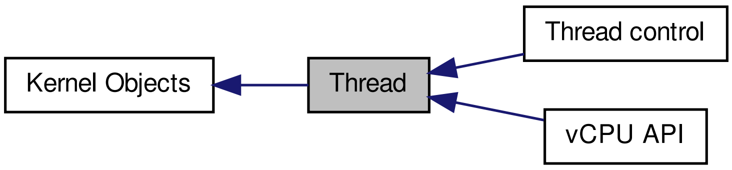
Here is the call graph for this function:



9.87 Thread

Thread object.

Collaboration diagram for Thread:



Modules

- Thread control

    API for Thread Control method.
- vCPU API

    vCPU API

Enumerations

- 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_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_TH↩
READ_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_RE↩
GS_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_N↩
OTHROW

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_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_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_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_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.

9.87.1 Detailed Description

Thread object.

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

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 Factory (l4_factory_create_thread()).

An L4 thread encapsulates:

- CPU state
  - General-purpose registers
  - Program counter
  - Stack pointer
- FPU state
- Scheduling parameters
  - CPU-set
  - Priority (0-255)
  - Time slice length

- 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.

9.87.2 Enumeration Type Documentation

9.87.2.1 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 591 of file thread.h.

9.87.2.2 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 618 of file thread.h.

9.87.2.3 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 641 of file thread.h.

9.87.2.4 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 656 of file thread.h.

9.87.3 Function Documentation

9.87.3.1 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	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 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*).

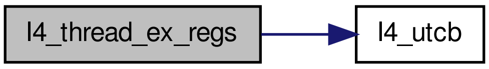
Examples:

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 796 of file thread.h.

References l4_utcb().

Here is the call graph for this function:



9.87.3.2 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

in	*thread*	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 803 of file thread.h.

References l4_utcb().

Here is the call graph for this function:

l4_thread_ex_regs_ret
l4_utcb

9.87.3.3 l4_msgtag_t l4_thread_yield (void) [inline]

Yield current time slice.

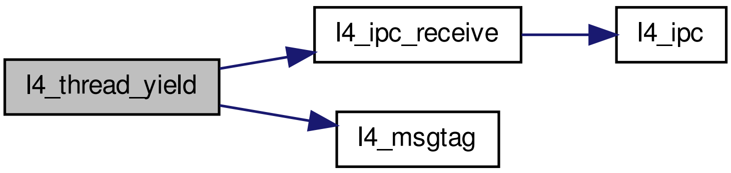
Returns

system call return tag

Definition at line 756 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:



9.87.3.4 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.

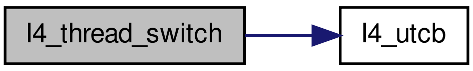
Returns

system call return tag

Definition at line 856 of file thread.h.

References l4_utcb().

Here is the call graph for this function:



9.87.3.5 l4_msgtag_t l4_thread_stats_time (l4_cap_idx_t *thread*) [inline]

Get consumed time of thread in µs.

Parameters

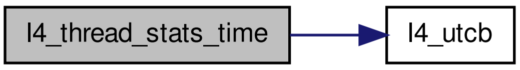
thread	Thread to get the consumed time from.

The consumed time is returned as l4_kernel_clock_t at UTCB message register 0.

Definition at line 865 of file thread.h.

References l4_utcb().

Here is the call graph for this function:



9.87.3.6 l4_msgtag_t l4_thread_vcpu_resume_start (void) [inline]

vCPU return from event handler.

Returns

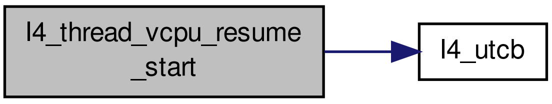
Message tag to be used for l4_sndfpage_add() and l4_thread_vcpu_commit()

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 871 of file thread.h.

References l4_utcb().

Here is the call graph for this function:



9.87.3.7 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 are 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.

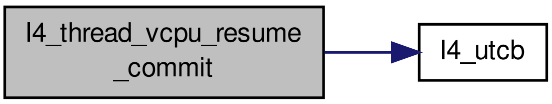
See also

l4_vcpu_state_t

Definition at line 877 of file thread.h.

References l4_utcb().

Here is the call graph for this function:



9.87.3.8 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	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.

Returns

Systemcall result message 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 914 of file thread.h.

References l4_utcb().

Here is the call graph for this function:

9.87.3.9 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	The thread for which the extended 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.

Returns

Systemcall result message tag.

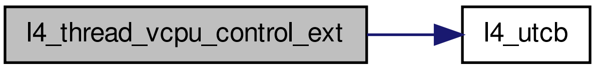
The extended vCPU feature allows the use of hardware-virtualization features such as Intel's VT or AMD's SVM.

This function enables the extended vCPU feature of the *thread* if *vcpu_state* is set to a valid kernel-user-memory address, or disables the vCPU feature if *vcpu_state* is 0.

Definition at line 929 of file thread.h.

References l4_utcb().

Here is the call graph for this function:



9.87.3.10 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.

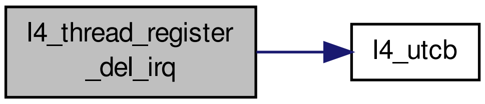
Returns

System call result message tag.

Definition at line 897 of file thread.h.

References l4_utcb().

Here is the call graph for this function:



9.87.3.11 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().

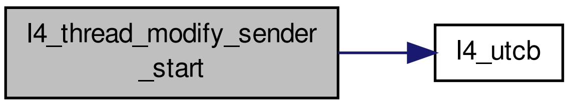
See also

l4_thread_modify_sender_add
l4_thread_modify_sender_commit

Definition at line 970 of file thread.h.

References l4_utcb().

Here is the call graph for this function:



9.87.3.12 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.

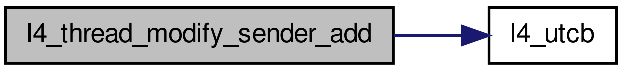
See also

l4_thread_modify_sender_start
l4_thread_modify_sender_commit

Definition at line 976 of file thread.h.

References l4_utcb().

Here is the call graph for this function:



9.87.3.13 l4_msgtag_t l4_thread_modify_sender_commit (l4_cap_idx_t *thread,* l4_msgtag_t *tag*) [inline]

Apply (commit) a sender modification sequence.

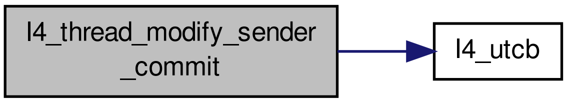
See also

l4_thread_modify_sender_start
l4_thread_modify_sender_add

Definition at line 987 of file thread.h.

References l4_utcb().

Here is the call graph for this function:



9.87.3.14 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

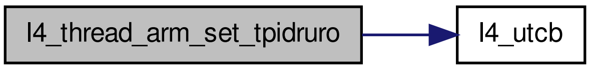
Returns

System call return tag

Definition at line 59 of file thread.h.

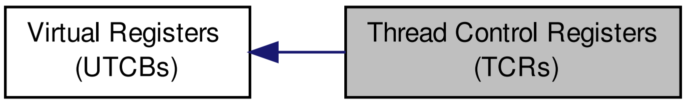
References l4_utcb().

Here is the call graph for this function:



9.88 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

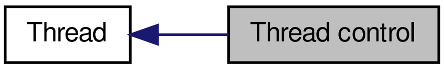
  Encapsulation of the thread-control-register block of the UTCB.

9.88.1 Detailed Description

9.89 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.

9.89.1 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);
```

9.89.2 Function Documentation

9.89.2.1 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 810 of file thread.h.

References l4_utcb().

Here is the call graph for this function:

l4_thread_control_start
l4_utcb

9.89.2.2 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).

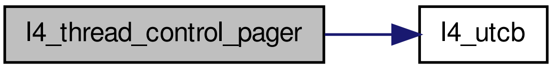
Examples:

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 816 of file thread.h.

References l4_utcb().

Here is the call graph for this function:



9.89.2.3 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).

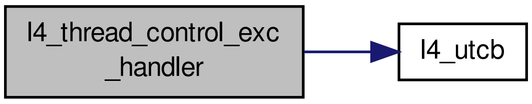
Examples:

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 822 of file thread.h.

References l4_utcb().

Here is the call graph for this function:



9.89.2.4 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.

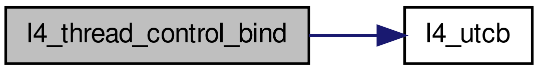
Examples:

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 829 of file thread.h.

References l4_utcb().

Here is the call graph for this function:



9.89.2.5 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 835 of file thread.h.

References l4_utcb().

Here is the call graph for this function:

9.89.2.6 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

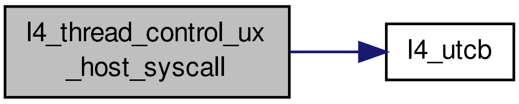
Running on Fiasco-UX

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 841 of file thread.h.

References l4_utcb().

Here is the call graph for this function:



9.89.2.7 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 847 of file thread.h.

References l4_utcb().

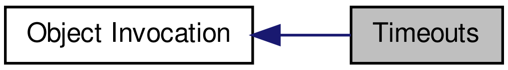
Here is the call graph for this function:

l4_thread_control_commit
l4_utcb

9.90 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.

9.90.1 Detailed Description

All kinds of timeouts and time related functions.

9.90.2 Macro Definition Documentation

9.90.2.1 #define L4_IPC_TIMEOUT_0 ((l4_timeout_s){0x0400})

Timeout constants.

0 timeout

Definition at line 77 of file __timeout.h.

9.90.3 Typedef Documentation

9.90.3.1 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).

9.90.3.2 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.

9.90.4 Enumeration Type Documentation

9.90.4.1 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.

9.90.5 Function Documentation

9.90.5.1 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.

9.90.5.2 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.

9.90.5.3 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.

9.90.5.4 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.

9.90.5.5 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.

9.90.5.6 **l4_kernel_clock_t** l4_timeout_rel_get (**l4_timeout_s** *to*) [inline]

Get clock value of out timeout.

Parameters

to	L4 timeout

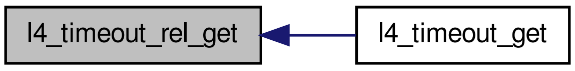
Returns

Clock value

Definition at line 252 of file __timeout.h.

Referenced by l4_timeout_get().

Here is the caller graph for this function:



9.90.5.7 unsigned l4_timeout_is_absolute (**l4_timeout_s** *to*) [inline]

Return whether the given timeout is absolute or not.

Parameters

to	L4 timeout

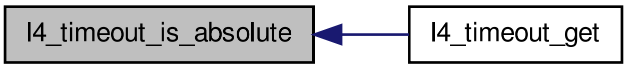
Returns

!= 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:



9.90.5.8 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

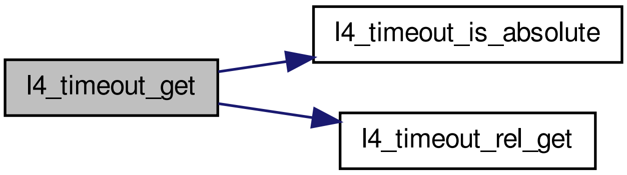
Returns

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:



9.90.5.9 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.

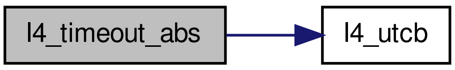
Returns

timeout value

Definition at line 373 of file utcb.h.

References l4_utcb().

Here is the call graph for this function:



9.90.5.10 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

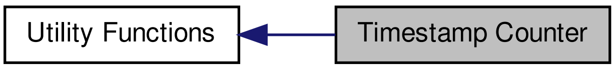
Returns

Index to 64bit message register alias

Definition at line 376 of file utcb.h.

9.91 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.

9.91.1 Detailed Description

9.91.2 Function Documentation

9.91.2.1 l4_cpu_time_t l4_rdtsc (void) [inline]

Read current value of CPU-internal time stamp counter.

Returns

64-bit time stamp

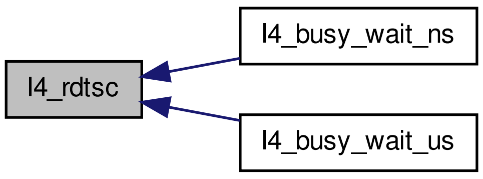
Examples:

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:



9.91.2.2 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.

9.91.2.3 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.

9.91.2.4 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.

9.91.2.5 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.

9.91.2.6 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.

9.91.2.7 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.

9.91.2.8 l4_cpu_time_t l4_ns_to_tsc (l4_uint64_t *ns*) [inline]

Convert nano seconds into CPU ticks.

Parameters

ns	nano seconds

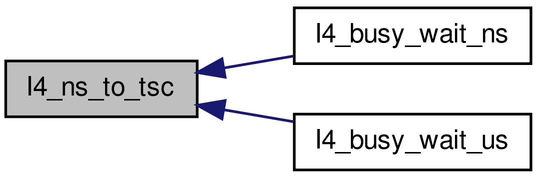
Returns

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:



9.91.2.9 void l4_busy_wait_ns (l4_uint64_t *ns*) [inline]

Wait busy for a small amount of time.

Parameters

ns	nano seconds to wait

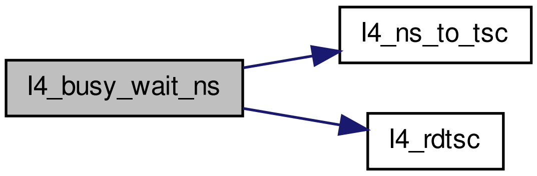
Attention

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:



9.91.2.10 void l4_busy_wait_us (l4_uint64_t *us*) [inline]

Wait busy for a small amount of time.

Parameters

us	micro seconds to wait

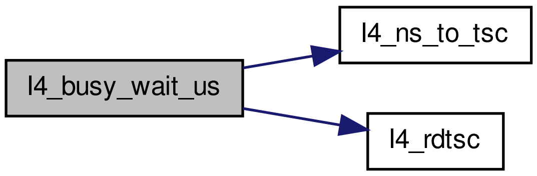
Attention

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:



9.91.2.11 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).

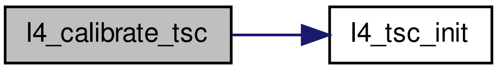
Examples:

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:



9.91.2.12 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

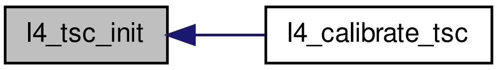
constraint	programmers constraint: • L4_TSC_INIT_AUTO if the kernel exports the scalers then use them. If not, perform calibration using channel 0 of the PIT (i8254). The latter case may lead into short (unpredictable) periods where interrupts are disabled. • L4_TSC_INIT_KERNEL depend on retrieving the 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:



9.91.2.13 l4_uint32_t l4_get_hz (void)

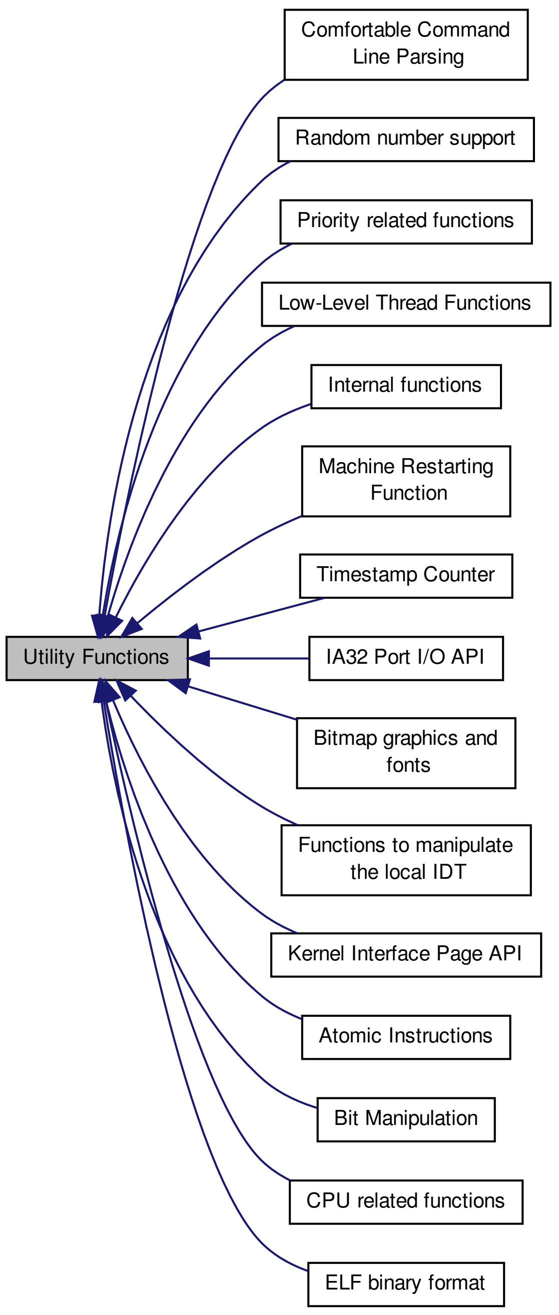
Get CPU frequency in Hz.

Returns

frequency in Hz

9.92 Utility Functions

Collaboration diagram for Utility Functions:


Comfortable Command Line Parsing
Random number support
Priority related functions
Low-Level Thread Functions
Internal functions
Machine Restarting Function
Timestamp Counter
Utility Functions
IA32 Port I/O API
Bitmap graphics and fonts
Functions to manipulate the local IDT
Kernel Interface Page API
Atomic Instructions
Bit Manipulation
CPU related functions
ELF binary format

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
- Priority related functions
- 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.

9.92.1 Detailed Description

9.92.2 Function Documentation

9.92.2.1 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.

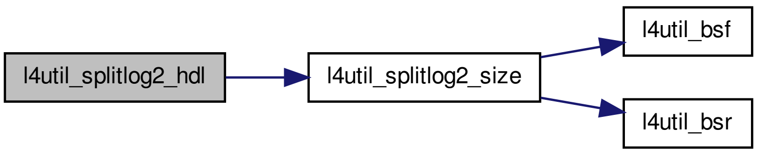
Returns

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:



9.92.2.2 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

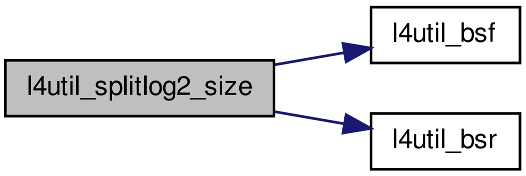
length of elements in log2size (length is 1 $<<$ log2size)

Definition at line 72 of file splitlog2.h.

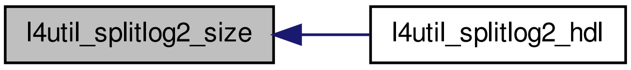
References l4util_bsf(), and l4util_bsr().

Referenced by l4util_splitlog2_hdl().

Here is the call graph for this function:



Here is the caller graph for this function:



9.92.2.3 **l4_timeout_s** l4util_micros2l4to (unsigned int *mus*)

Calculate l4 timeouts.

Parameters

mus	time in microseconds. Special cases: • 0 - > timeout 0 • ∼0U -> timeout NEVER

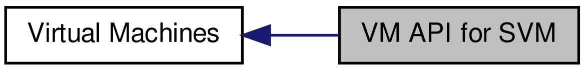
Returns

the corresponding l4_timeout value

9.93 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.

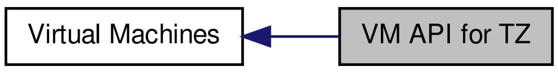
9.93.1 Detailed Description

Virtual machine API for SVM.

9.94 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

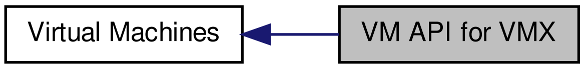
9.94.1 Detailed Description

Virtual Machine API for ARM TrustZone.

9.95 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 }

  Additional 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

  Loades 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.

9.95.1 Detailed Description

Virtual machine API for VMX.

9.95.2 Enumeration Type Documentation

9.95.2.1 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.

9.95.2.2 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.

9.95.2.3 anonymous enum

Additional VMCS fields.

Enumerator

L4_VM_VMX_VMCS_CR2 (virtual) VMCS offset for CR2. 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 version of the VMCS.

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.

Definition at line 100 of file __vm-vmx.h.

9.95.3 Function Documentation

9.95.3.1 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 also

L4_vm_vmx_caps_regs).

Returns

The value of the capability register.

Definition at line 506 of file __vm-vmx.h.

References L4_VCPU_OFFSET_EXT_INFOS.

9.95.3.2 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 also

L4_vm_vmx_dfl1_regs).

Returns

The value of the capability register.

Definition at line 514 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.

9.95.3.3 unsigned l4_vm_vmx_field_len (unsigned *field*) [inline]

Return length in bytes of a VMCS field.

Parameters

field	Field number.

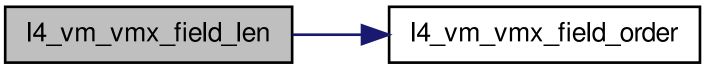
Returns

Width of field in bytes.

Definition at line 335 of file __vm-vmx.h.

References l4_vm_vmx_field_order().

Here is the call graph for this function:



9.95.3.4 unsigned l4_vm_vmx_field_order (unsigned *field*) [inline]

Return length in power of two (bytes) of a VMCS field.

Parameters

field	Field number.

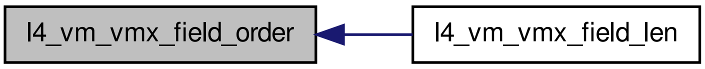
Returns

Width of field in power of two (bytes).

Definition at line 320 of file __vm-vmx.h.

Referenced by l4_vm_vmx_field_len().

Here is the caller graph for this function:



9.95.3.5 void l4_vm_vmx_clear (void ∗ *vmcs*, void ∗ *user_vmcs*) [inline]

Saves cached state from the kernel VMCS to the user VMCS.

Parameters

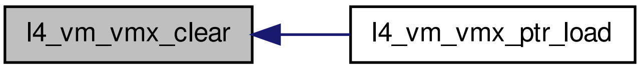
vmcs	Pointer to the kernel VMCS.
user_vmcs	Pointer to the user VMCS.

This function is comparable to VMX vmclear.

Definition at line 411 of file __vm-vmx.h.

Referenced by l4_vm_vmx_ptr_load().

Here is the caller graph for this function:



9.95.3.6 void l4_vm_vmx_ptr_load (void ∗ *vmcs*, void ∗ *user_vmcs*) [inline]

Loades the user_vmcs as the current VMCS.

Parameters

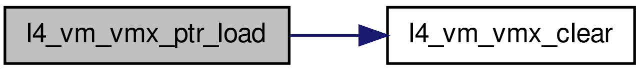
vmcs	Pointer to the kernel VMCS.
user_vmcs	Pointer to the user VMCS.

This function is comparable to VMX vmptrld.

Definition at line 423 of file __vm-vmx.h.

References l4_vm_vmx_clear().

Here is the call graph for this function:



9.95.3.7 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 522 of file __vm-vmx.h.

9.95.3.8 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.

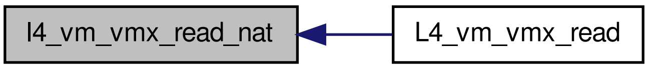
Returns

The value of the VMCS field with the given index.

Definition at line 439 of file __vm-vmx.h.

Referenced by L4_vm_vmx_read().

Here is the caller graph for this function:



9.95.3.9 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.

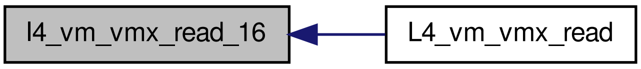
Returns

The value of the VMCS field with the given index.

Definition at line 444 of file __vm-vmx.h.

Referenced by L4_vm_vmx_read().

Here is the caller graph for this function:



9.95.3.10 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.

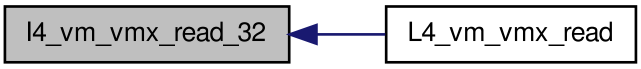
Returns

The value of the VMCS field with the given index.

Definition at line 449 of file __vm-vmx.h.

Referenced by L4_vm_vmx_read().

Here is the caller graph for this function:



9.95.3.11 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.

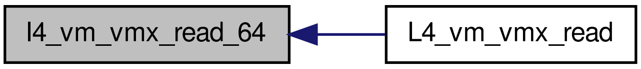
Returns

The value of the VMCS field with the given index.

Definition at line 454 of file __vm-vmx.h.

Referenced by L4_vm_vmx_read().

Here is the caller graph for this function:



9.95.3.12 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.

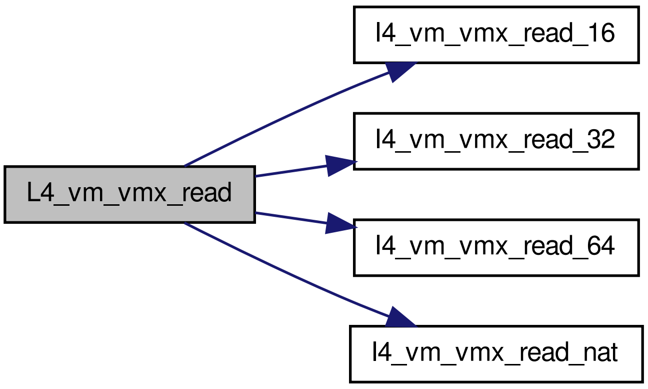
Returns

The value of the VMCS field with the given index.

Definition at line 459 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:



9.95.3.13 void l4_vm_vmx_write_nat (void ∗ *vmcs,* unsigned *field,* l4_umword_t *val*) [inline]

Write to a natural width VMCS field.

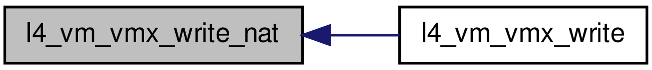
Parameters

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 472 of file __vm-vmx.h.

Referenced by l4_vm_vmx_write().

Here is the caller graph for this function:



9.95.3.14 void l4_vm_vmx_write_16 (void ∗ *vmcs,* unsigned *field,* l4_uint16_t *val*) [inline]

Write to a 16bit VMCS field.

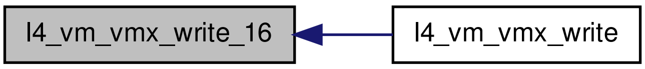
Parameters

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 477 of file __vm-vmx.h.

Referenced by l4_vm_vmx_write().

Here is the caller graph for this function:



9.95.3.15 void l4_vm_vmx_write_32 (void ∗ *vmcs,* unsigned *field,* l4_uint32_t *val*) [inline]

Write to a 32bit VMCS field.

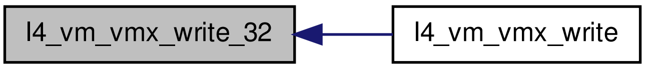
Parameters

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 482 of file __vm-vmx.h.

Referenced by l4_vm_vmx_write().

Here is the caller graph for this function:



9.95.3.16 void l4_vm_vmx_write_64 (void ∗ *vmcs,* unsigned *field,* l4_uint64_t *val*) [inline]

Write to a 64bit VMCS field.

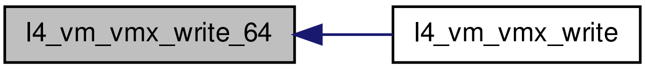
Parameters

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 487 of file __vm-vmx.h.

Referenced by l4_vm_vmx_write().

Here is the caller graph for this function:



9.95.3.17 void l4_vm_vmx_write (void * *vmcs*, unsigned *field*, l4_uint64_t *val*) [inline]

Write to an arbitrary VMCS field.

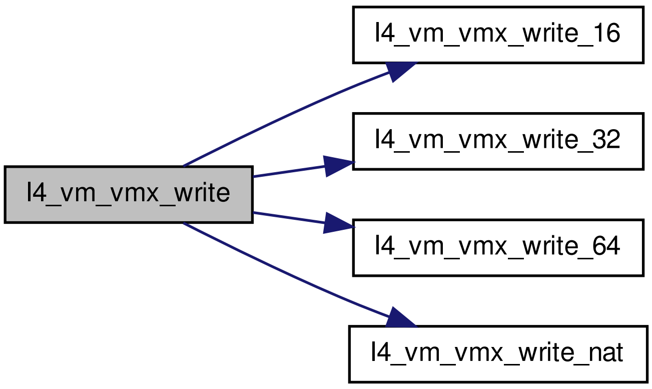
Parameters

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 493 of file __vm-vmx.h.

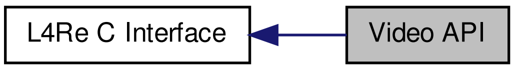
References l4_vm_vmx_write_16(), l4_vm_vmx_write_32(), l4_vm_vmx_write_64(), and l4_vm_vmx_write_nat().

Here is the call graph for this function:



9.96 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_N↩ OTHROW

  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_NO↩ THROW

  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_NOTH↩ ROW

  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.

9.96.1 Detailed Description

9.96.2 Typedef Documentation

9.96.2.1 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.

9.96.3 Enumeration Type Documentation

9.96.3.1 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.

9.96.3.2 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.

9.96.4 Function Documentation

9.96.4.1 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

9.96.4.2 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

9.96.4.3 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

9.96.4.4 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())

9.96.4.5 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.
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.

9.96.4.6 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.

9.96.4.7 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.

9.96.4.8 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().

9.96.4.9 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

9.96.4.10 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.

9.96.4.11 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.

9.96.4.12 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.

9.96.4.13 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.

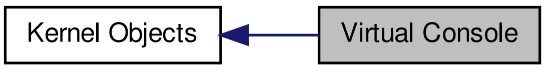
Parameters

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.

9.97 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.
- 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 the vcon objects.

Functions

- l4_msgtag_t l4_vcon_send (l4_cap_idx_t vcon, char const ∗buf, int size) L4_NOTHROW

  Send data to virtual console.
- long l4_vcon_write (l4_cap_idx_t vcon, char const ∗buf, int size) L4_NOTHROW

  Write data to 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_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_get_attr (l4_cap_idx_t vcon, l4_vcon_attr_t ∗attr) L4_NOTHROW

    Get attributes of a Vcon.

9.97.1 Detailed Description

Virtual console for simple character based input and output.

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

Interrupt for read events are provided by the virtual key interrupt.

9.97.2 Enumeration Type Documentation

9.97.2.1 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 83 of file vcon.h.

9.97.2.2 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 169 of file vcon.h.

9.97.2.3 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 ouput CR.

Definition at line 180 of file vcon.h.

9.97.2.4 enum L4_vcon_l_flags

Local flags.

Enumerator

L4_VCON_ICANON	Cannonical mode.
L4_VCON_ECHO	Echo input.

Definition at line 191 of file vcon.h.

9.97.2.5 enum L4_vcon_ops

Operations on the 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 240 of file vcon.h.

9.97.3 Function Documentation

9.97.3.1 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

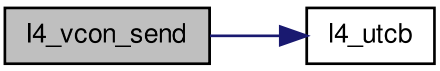
Note

Size must not exceed L4_VCON_WRITE_SIZE, a proper value of the *size* parameter is NOT checked.

Definition at line 265 of file vcon.h.

References l4_utcb().

Here is the call graph for this function:



9.97.3.2 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.

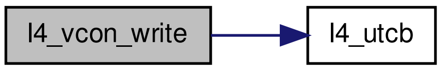
Returns

Number of bytes written to the virtual console.

Definition at line 286 of file vcon.h.

References l4_utcb().

Here is the call graph for this function:



9.97.3.3 int l4_vcon_read (l4_cap_idx_t *vcon*, char ∗ *buf*, int *size*) [inline]

Read data from virtual console.

Parameters

vcon	Vcon object.
buf	Pointer to data buffer.
size	Size of buffer in bytes.

Note

Size must not exceed *L4_VCON_READ_SIZE*.

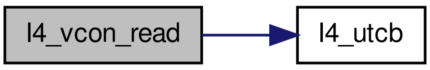
Returns

Negative error code on error, > size if more to read, size bytes are in the buffer, <= size bytes read

Definition at line 341 of file vcon.h.

References l4_utcb().

Here is the call graph for this function:



9.97.3.4 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.
buf	Pointer to data buffer.
size	Size of buffer in bytes.

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*.

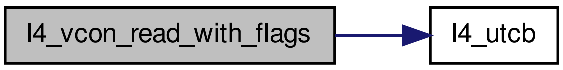
Returns

Negative error code on error, if positive, an enabled *L4_VCON_READ_STAT_BREAK* flag bit indicates a break condition, *L4_VCON_READ_SIZE_MASK* contains the size field, > size if more to read, size bytes are in the buffer, <= size bytes read

Definition at line 325 of file vcon.h.

References l4_utcb().

Here is the call graph for this function:



9.97.3.5 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.

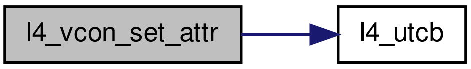
Returns

Syscall return tag

Definition at line 361 of file vcon.h.

References l4_utcb().

Here is the call graph for this function:



9.97.3.6 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.

Return values

attr	Attribute structure.

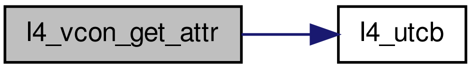
Returns

Syscall return tag

Definition at line 385 of file vcon.h.

References l4_utcb().

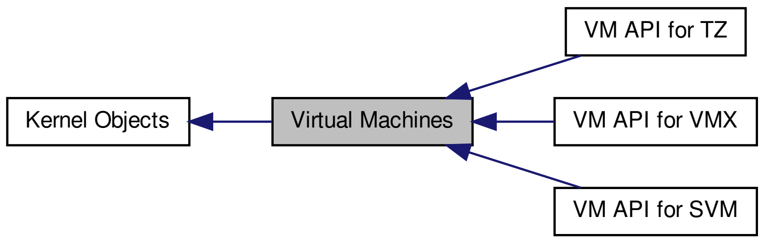
Here is the call graph for this function:



9.98 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.

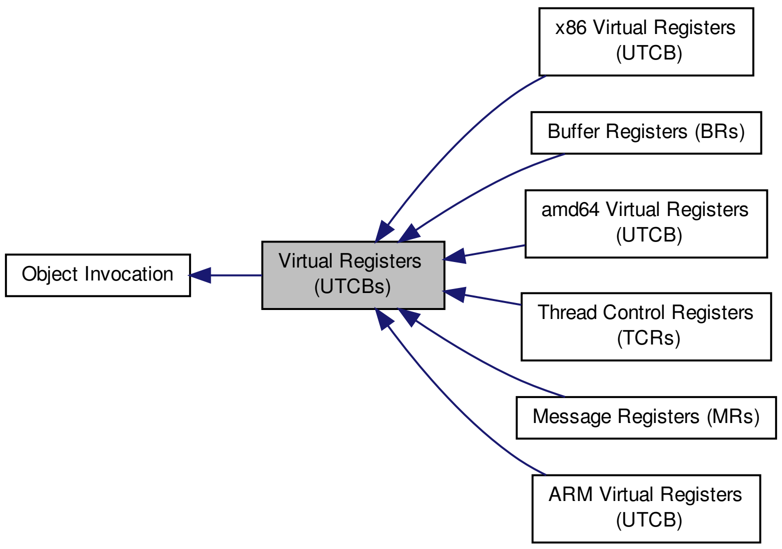
9.98.1 Detailed Description

Virtual Machine API.

9.99 Virtual Registers (UTCBs)

L4 Virtual Registers (UTCB).

Collaboration diagram for Virtual Registers (UTCBs):


x86 Virtual Registers (UTCB)
Buffer Registers (BRs)
amd64 Virtual Registers (UTCB)
Object Invocation
Virtual Registers (UTCBs)
Thread Control Registers (TCRs)
Message Registers (MRs)
ARM Virtual Registers (UTCB)

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.

9.99.1 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.

9.99.2 Typedef Documentation

9.99.2.1 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.

9.99.3 Function Documentation

9.99.3.1 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`.

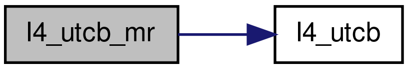
Examples:

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:



9.99.3.2 l4_buf_regs_t ∗ l4_utcb_br (void) [inline]

Get the buffer-register block of a UTCB.

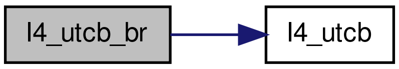
Returns

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:



9.99.3.3 l4_thread_regs_t ∗ l4_utcb_tcr (void) [inline]

Get the thread-control-register block of a UTCB.

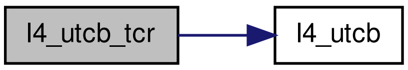
Returns

A pointer to the thread-control-register block of `u`.

Definition at line 348 of file utcb.h.

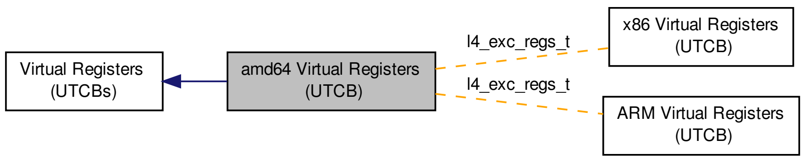
References l4_utcb().

Here is the call graph for this function:



9.100 amd64 Virtual Registers (UTCB)

Collaboration diagram for amd64 Virtual Registers (UTCB):


Virtual Registers (UTCBs)
amd64 Virtual Registers (UTCB)
l4_exc_regs_t
l4_exc_regs_t
x86 Virtual Registers (UTCB)
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_amd64

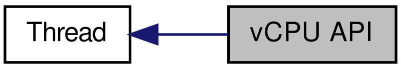
  UTCB constants for AMD64.

9.100.1 Detailed Description

9.101 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.

9.101.1 Detailed Description

vCPU API

9.101.2 Enumeration Type Documentation

9.101.2.1 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.

9.101.2.2 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.

9.101.2.3 enum L4_vcpu_state_offset

Offsets for vCPU state layouts.

Enumerator

L4_VCPU_OFFSET_EXT_STATE Offset where extended state begins.

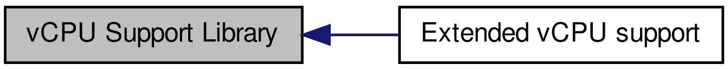
L4_VCPU_OFFSET_EXT_INFOS Offset where extended infos begin.

Definition at line 79 of file vcpu.h.

9.102 vCPU Support Library

vCPU handling functionality.

Collaboration diagram for vCPU Support Library:



Modules

- Extended vCPU support

    extended vCPU handling functionality.

Typedefs

- typedef enum l4vcpu_irq_state_t l4vcpu_irq_state_t

    IRQ/Event enable and disable flags.

Enumerations

- enum l4vcpu_irq_state_t { L4VCPU_IRQ_STATE_DISABLED = 0, L4VCPU_IRQ_STATE_ENABLED = L4↩ _VCPU_F_IRQ }

    IRQ/Event enable and disable flags.

Functions

- l4vcpu_state_t l4vcpu_state (l4_vcpu_state_t const ∗vcpu) L4_NOTHROW

    Return the state flags of a vCPU.
- void l4vcpu_irq_disable (l4_vcpu_state_t ∗vcpu) L4_NOTHROW

    Disable a vCPU for event delivery.
- l4vcpu_irq_state_t 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, l4vcpu_irq_state_t 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 (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 ∗vcpu) L4_NOTHROW

Return whether the entry reason was an IRQ/IPC message.

- int l4vcpu_is_page_fault_entry (l4_vcpu_state_t ∗vcpu) L4_NOTHROW

  Return whether the entry reason was a page fault.

9.102.1 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.

9.102.2 Enumeration Type Documentation

9.102.2.1 enum **l4vcpu_irq_state_t**

IRQ/Event enable and disable flags.

Enumerator

L4VCPU_IRQ_STATE_DISABLED	IRQ/Event delivery disabled.
L4VCPU_IRQ_STATE_ENABLED	IRQ/Event delivery enabled.

Definition at line 44 of file vcpu.h.

9.102.3 Function Documentation

9.102.3.1 l4vcpu_state_t l4vcpu_state (**l4_vcpu_state_t** const ∗ *vcpu*) [inline]

Return the state flags of a vCPU.

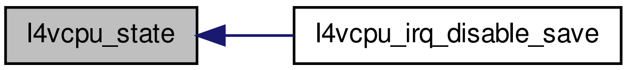
Parameters

vcpu	Pointer to vCPU area.

Definition at line 229 of file vcpu.h.

Referenced by l4vcpu_irq_disable_save().

Here is the caller graph for this function:



9.102.3.2 void l4vcpu_irq_disable (**l4_vcpu_state_t** ∗ *vcpu*) [inline]

Disable a vCPU for event delivery.

Parameters

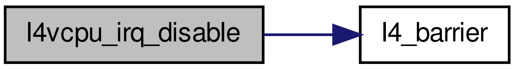
vcpu	Pointer to vCPU area.

Definition at line 236 of file vcpu.h.

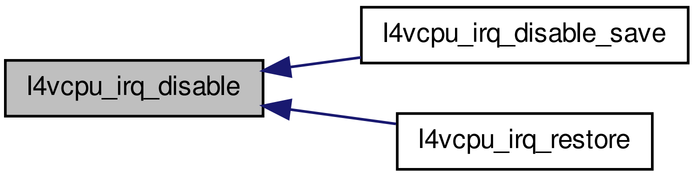
References l4_barrier(), and L4_VCPU_F_IRQ.

Referenced by l4vcpu_irq_disable_save(), and l4vcpu_irq_restore().

Here is the call graph for this function:



Here is the caller graph for this function:



9.102.3.3 l4vcpu_irq_state_t 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.

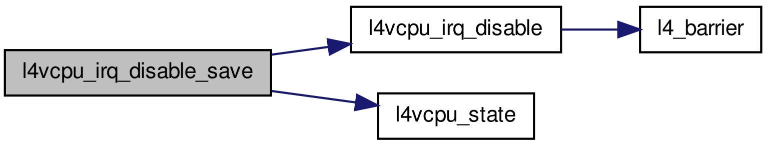
Returns

IRQ state before disabling IRQs.

Definition at line 244 of file vcpu.h.

References l4vcpu_irq_disable(), and l4vcpu_state().

Here is the call graph for this function:



9.102.3.4 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

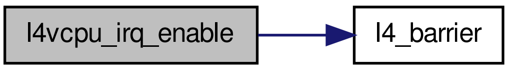
vcpu	Pointer to vCPU area.
utcb	Utcb pointer of the calling vCPU.
do_event_↩ work_cb	Call-back function that is called in case an event (such as an interrupt) is pending.
setup_ipc	Function call-back that is called right before any IPC operation, and before event delivery is enabled.

Definition at line 267 of file vcpu.h.

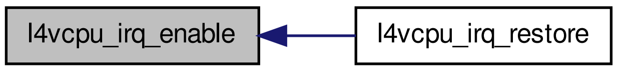
References l4_barrier(), L4_IPC_BOTH_TIMEOUT_0, L4_LIKELY, L4_VCPU_F_IRQ, and L4_VCPU_SF_IRQ_↩ PENDING.

Referenced by l4vcpu_irq_restore().

Here is the call graph for this function:



Here is the caller graph for this function:



9.102.3.5 void l4vcpu_irq_restore (**l4_vcpu_state_t** ∗ *vcpu*, **l4vcpu_irq_state_t** *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.

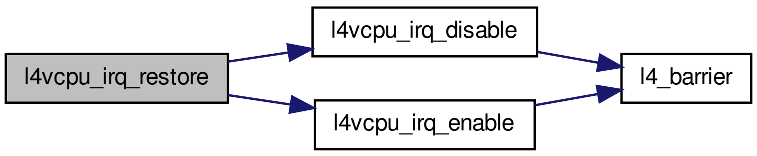
Parameters

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 en-abling.
setup_ipc	Function call-back that is called right before any IPC operation, and before event delivery is enabled.

Definition at line 292 of file vcpu.h.

References L4_VCPU_F_IRQ, l4vcpu_irq_disable(), and l4vcpu_irq_enable().

Here is the call graph for this function:



9.102.3.6 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 305 of file vcpu.h.

References L4_IPC_NEVER.

9.102.3.7 void l4vcpu_print_state (**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.

9.102.3.8 int l4vcpu_is_irq_entry (**l4_vcpu_state_t** ∗ *vcpu*) `[inline]`

Return whether the entry reason was an IRQ/IPC message.

Parameters

vcpu	Pointer to vCPU area.

return 0 if not, !=0 otherwise.

9.102.3.9 int l4vcpu_is_page_fault_entry (**l4_vcpu_state_t** ∗ *vcpu*) `[inline]`

Return whether the entry reason was a page fault.

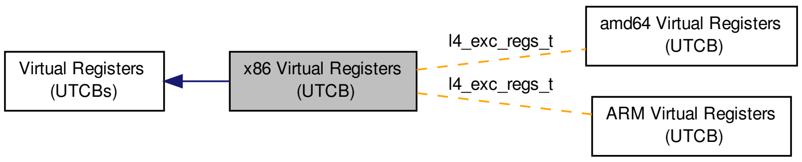
Parameters

vcpu	Pointer to vCPU area.

return 0 if not, !=0 otherwise.

9.103 x86 Virtual Registers (UTCB)

Collaboration diagram for x86 Virtual Registers (UTCB):


Virtual Registers (UTCBs)
x86 Virtual Registers (UTCB)
l4_exc_regs_t
amd64 Virtual Registers (UTCB)
l4_exc_regs_t
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_x86 {
  L4_UTCB_EXCEPTION_REGS_SIZE = 16, L4_UTCB_GENERIC_DATA_SIZE = 63, L4_UTCB_GENERI←C_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.

9.103.1 Detailed Description

9.103.2 Enumeration Type Documentation

9.103.2.1 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 10

Namespace Documentation

10.1 cxx::Bits Namespace Reference

Internal helpers for the cxx package.

Data Structures

- 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.

10.1.1 Detailed Description

Internal helpers for the cxx package.

10.2 L4 Namespace Reference

L4 low-level kernel interface.

Data Structures

- class Alloc_list

    A simple list-based allocator.
- class Kobject_2t

    Helper class to create an L4Re interface class that is derived from two base classes.
- class Kobject_t

    Helper class to create an L4Re interface class that is derived from a single base class.
- class String

    A null-terminated string container class.
- struct Type_info

    Dynamic Type Information for L4Re Interfaces.

Functions

- template<typename T >
  Type_info const ∗ kobject_typeid ()

  Get the L4::Type_info for the L4Re interface given in T.

10.2.1 Detailed Description

L4 low-level kernel interface.

10.2.2 Function Documentation

10.2.2.1 template<typename T > Type_info const∗ L4::kobject_typeid ( ) [inline]

Get the L4::Type_info for the L4Re interface given in *T*.

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 87 of file __typeinfo.h.

10.3 L4Re Namespace Reference

(c) 2014 Steffen Liebergeld `steffen.liebergeld@kernkonzept.com`

Namespaces

- Vfs

  Virtual file system for interfaces POSIX libc.

10.3.1 Detailed Description

(c) 2014 Steffen Liebergeld `steffen.liebergeld@kernkonzept.com`

This file is licensed under the terms of the GNU Lesser General Public License 2.1. See the file COPYING-LGPL-2.1 for details.

10.4 L4Re::Vfs Namespace Reference

Virtual file system for interfaces POSIX libc.

Data Structures

- 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.

10.4.1 Detailed Description

Virtual file system for interfaces POSIX libc.

Chapter 11

Data Structure Documentation

11.1 L4::Alloc_list Class Reference

A simple list-based allocator.

`#include <alloc.h>`

Collaboration diagram for L4::Alloc_list:

L4::Alloc_list
+ Alloc_list()
+ Alloc_list()
+ free()
+ alloc()

11.1.1 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:

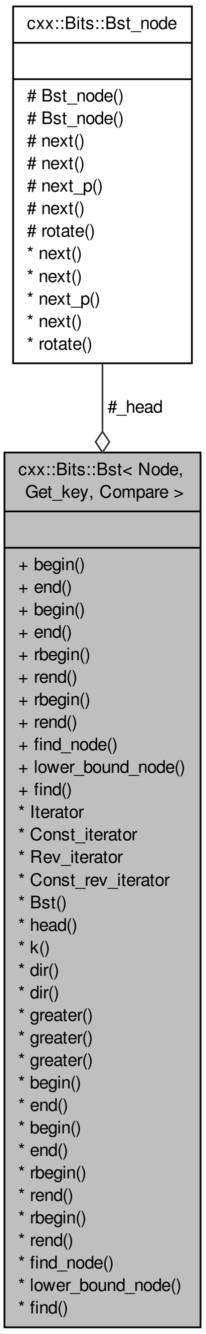
- l4/cxx/alloc.h

11.2 cxx::Bits::Bst< Node, Get_key, Compare > Class Template Reference

Basic binary search tree (BST).

`#include <bst.h>`

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.

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.

11.2.1 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.

11.2.2 Member Function Documentation

11.2.2.1 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.

Returns

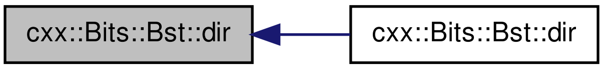
#Direction::L for left, #Direction::R for right, and #Direction::N if *l* is equal to *r*.

Definition at line 117 of file bst.h.

References cxx::Bits::Direction::L, and cxx::Bits::Direction::N.

Referenced by cxx::Bits::Bst< Node, Get_key, Compare >::dir().

Here is the caller graph for this function:



11.2.2.2 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.

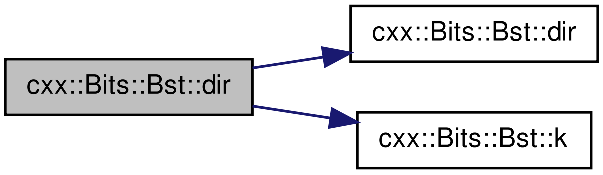
Returns

#Direction::L for left, #Direction::R for right, and #Direction::N if *l* is equal to *r*.

Definition at line 133 of file bst.h.

References cxx::Bits::Bst< Node, Get_key, Compare >::dir(), and cxx::Bits::Bst< Node, Get_key, Compare >::k().

Here is the call graph for this function:



11.2.2.3 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.

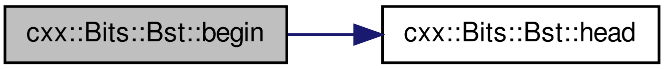
Returns

Constant forward iterator for the first element in the set.

Definition at line 159 of file bst.h.

References cxx::Bits::Bst< Node, Get_key, Compare >::head().

Here is the call graph for this function:



11.2.2.4 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 164 of file bst.h.

11.2.2.5 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.

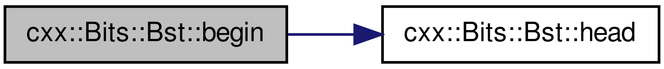
Returns

The mutable forward iterator for the first element of the set.

Definition at line 170 of file bst.h.

References cxx::Bits::Bst< Node, Get_key, Compare >::head().

Here is the call graph for this function:



11.2.2.6 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 175 of file bst.h.

11.2.2.7 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.

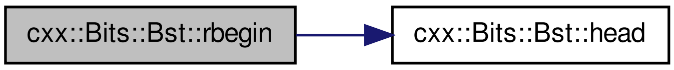
Returns

The constant backward iterator for the last element in the set.

Definition at line 181 of file bst.h.

References cxx::Bits::Bst< Node, Get_key, Compare >::head().

Here is the call graph for this function:



11.2.2.8 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 186 of file bst.h.

11.2.2.9 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.

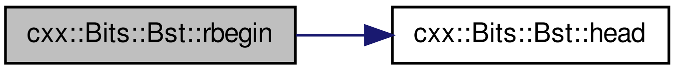
Returns

The mutable backward iterator for the last element of the set.

Definition at line 192 of file bst.h.

References cxx::Bits::Bst< Node, Get_key, Compare >::head().

Here is the call graph for this function:



11.2.2.10 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 197 of file bst.h.

11.2.2.11 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.

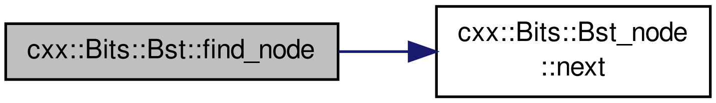
Returns

A pointer to the node with the given *key*, or NULL if *key* was not found.

Definition at line 236 of file bst.h.

References cxx::Bits::Bst_node::next().

Here is the call graph for this function:



11.2.2.12 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.

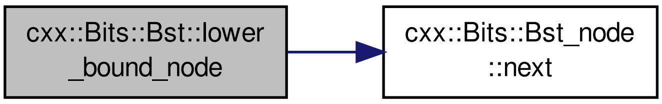
Returns

A pointer to the node with the given *key*, or NULL if *key* was not found.

Definition at line 252 of file bst.h.

References cxx::Bits::Bst_node::next().

Here is the call graph for this function:



11.2.2.13 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.

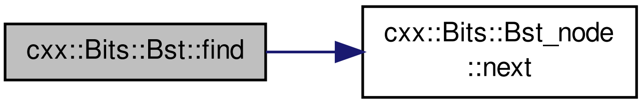
Returns

A valid iterator for the node with the given *key*, or an invalid iterator if *key* was not found.

Definition at line 272 of file bst.h.

References cxx::Bits::Bst_node::next().

Here is the call graph for this function:



The documentation for this class was generated from the following file:

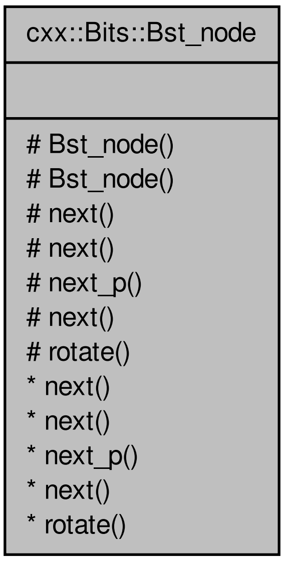
- l4/cxx/bits/bst.h

11.3 cxx::Bits::Bst_node Class Reference

Basic type of a node in a binary search tree (BST).

```
#include <bst_base.h>
```

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.

11.3.1 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:

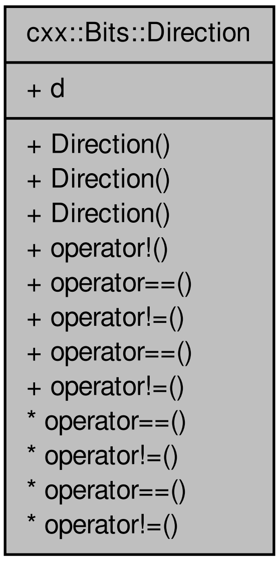
- l4/cxx/bits/bst_base.h

11.4 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

11.4.1 Detailed Description

The direction to go in a binary search tree.

Definition at line 39 of file bst_base.h.

11.4.2 Member Enumeration Documentation

11.4.2.1 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.

11.4.3 Member Function Documentation

11.4.3.1 **Direction** cxx::Bits::Direction::operator! ( ) const `[inline]`

Negate the direction.

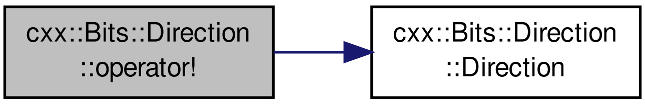
Note

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:


cxx::Bits::Direction ::operator!
cxx::Bits::Direction ::Direction

The documentation for this struct was generated from the following file:

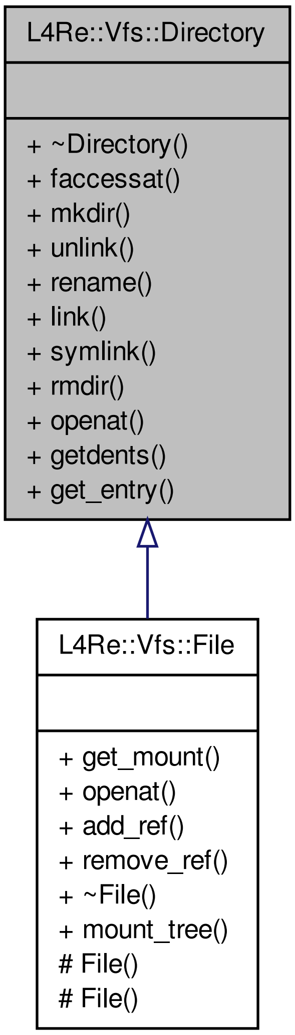
- l4/cxx/bits/bst_base.h

11.5 L4Re::Vfs::Directory Class Reference

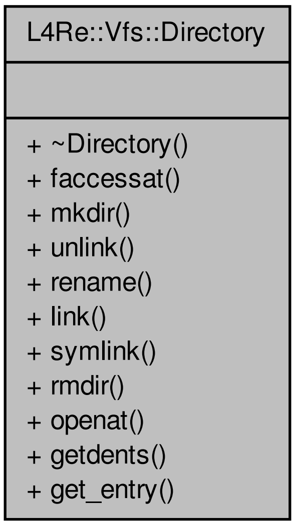
Interface for a POSIX file that is a directory.

`#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 ∗)=0 throw ()

  Delete an empty directory.

11.5.1 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.

11.5.2 Member Function Documentation

11.5.2.1 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.

11.5.2.2 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.

11.5.2.3 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.

11.5.2.4 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.

11.5.2.5 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.

11.5.2.6 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.

11.5.2.7 virtual int L4Re::Vfs::Directory::rmdir (const char ∗) 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.

The documentation for this class was generated from the following file:

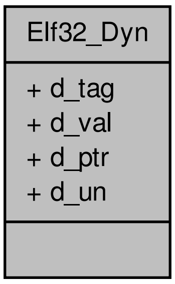
- l4/l4re_vfs/vfs.h

11.6 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

11.6.1 Detailed Description

ELF32 dynamic entry.

Definition at line 460 of file elf.h.

11.6.2 Field Documentation

11.6.2.1 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:

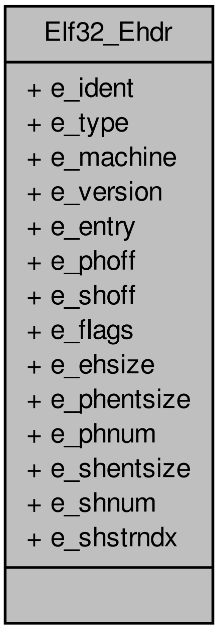
- l4/util/elf.h

11.7 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

11.7.1 Detailed Description

ELF32 header.

Definition at line 118 of file elf.h.

11.7.2 Field Documentation

11.7.2.1 Elf32_Half Elf32_Ehdr::e_phnum

of entries in prog.

head. tab.

Definition at line 129 of file elf.h.

11.7.2.2 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

11.8 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

11.8.1 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

11.9 Elf32_Shdr Struct Reference

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

`#include <elf.h>`

Collaboration diagram for Elf32_Shdr:

Elf32_Shdr
+ sh_name
+ sh_type
+ sh_flags
+ sh_addr
+ sh_offset
+ sh_size
+ sh_link
+ sh_info
+ sh_addralign
+ sh_entsize

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

11.9.1 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:

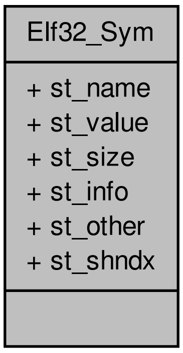
- l4/util/elf.h

11.10 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

11.10.1 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:

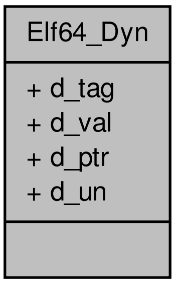
- l4/util/elf.h

11.11 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

11.11.1 Detailed Description

ELF64 dynamic entry.

Definition at line 469 of file elf.h.

11.11.2 Field Documentation

11.11.2.1 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:

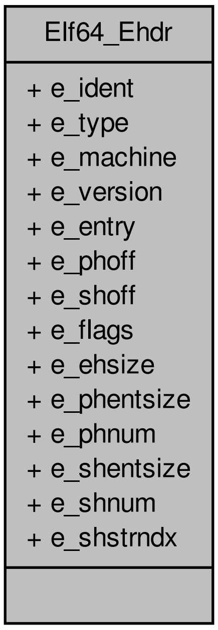
- l4/util/elf.h

11.12 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

11.12.1 Detailed Description

ELF64 header.

Definition at line 138 of file elf.h.

11.12.2 Field Documentation

11.12.2.1 Elf64_Half Elf64_Ehdr::e_phnum

of entries in prog.

head. tab.

Definition at line 149 of file elf.h.

11.12.2.2 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

11.13 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

11.13.1 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

11.14 Elf64_Shdr Struct Reference

ELF64 section header.

```
#include <elf.h>
```

Collaboration diagram for Elf64_Shdr:

Elf64_Shdr
+ sh_name
+ sh_type
+ sh_flags
+ sh_addr
+ sh_offset
+ sh_size
+ sh_link
+ sh_info
+ sh_addralign
+ sh_entsize

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

11.14.1 Detailed Description

ELF64 section header.

Definition at line 316 of file elf.h.

The documentation for this struct was generated from the following file:

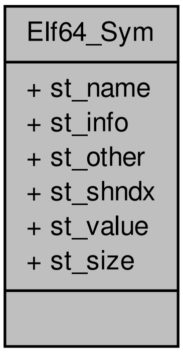
- l4/util/elf.h

11.15 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

11.15.1 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:

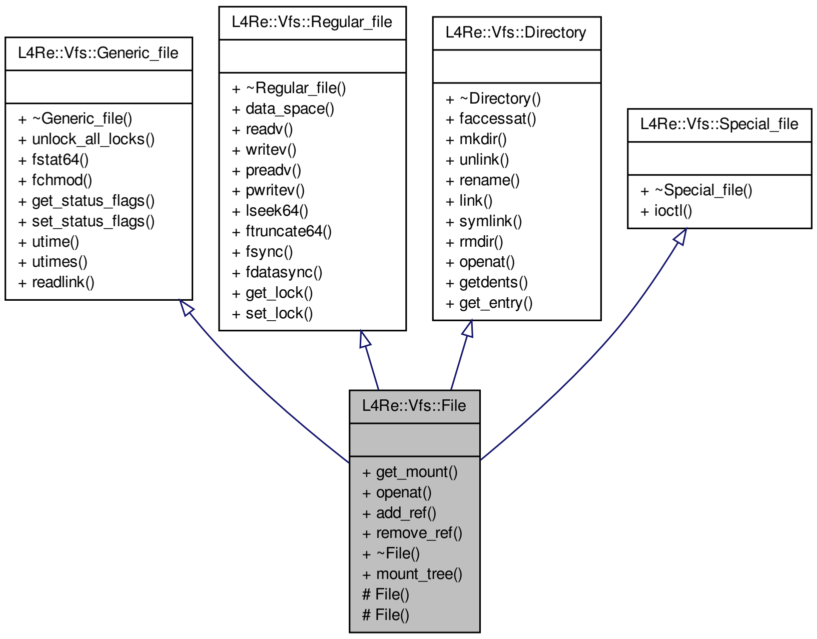
- l4/util/elf.h

11.16 L4Re::Vfs::File Class Reference

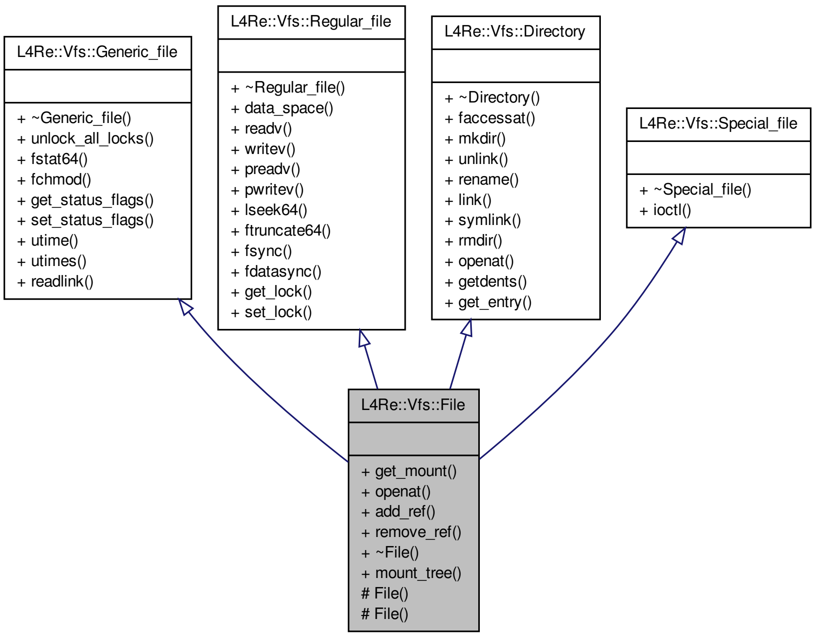
The basic interface for an open POSIX file.

`#include <vfs.h>`

Inheritance diagram for L4Re::Vfs::File:



Collaboration diagram for L4Re::Vfs::File:



Additional Inherited Members

11.16.1 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:

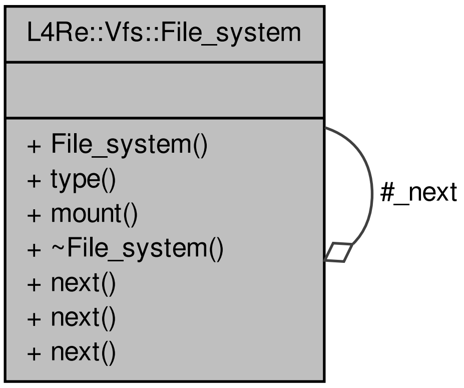
- l4/l4re_vfs/vfs.h

11.17 L4Re::Vfs::File_system Class Reference

Basic interface for an L4Re::Vfs file system.

```
#include <vfs.h>
```

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.

11.17.1 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 827 of file vfs.h.

11.17.2 Member Function Documentation

11.17.2.1 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.

11.17.2.2 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

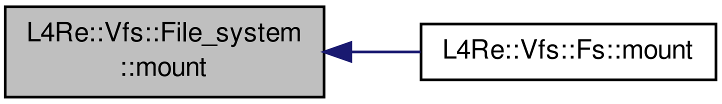
dir	A new directory object representing the file-system root directory.

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:



The documentation for this class was generated from the following file:

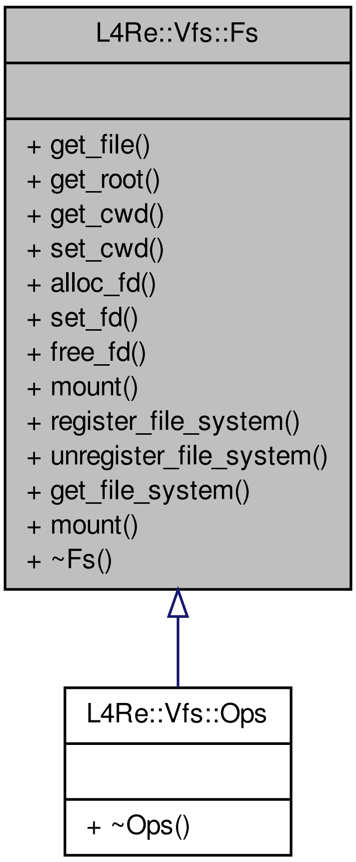
- l4/l4re_vfs/vfs.h

11.18 L4Re::Vfs::Fs Class Reference

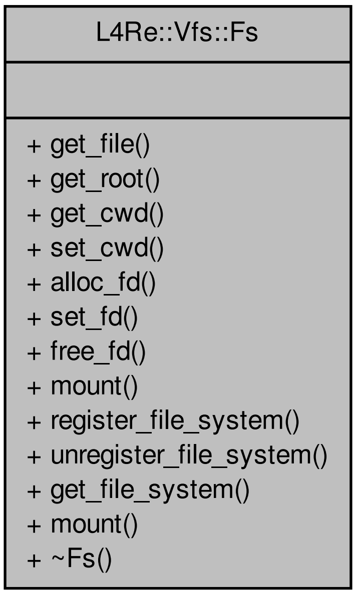
POSIX File-system related functionality.

```
#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.

11.18.1 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 879 of file vfs.h.

11.18.2 Member Function Documentation

11.18.2.1 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.

11.18.2.2 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.

11.18.2.3 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.

11.18.2.4 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.

11.18.2.5 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 968 of file vfs.h.

The documentation for this class was generated from the following file:

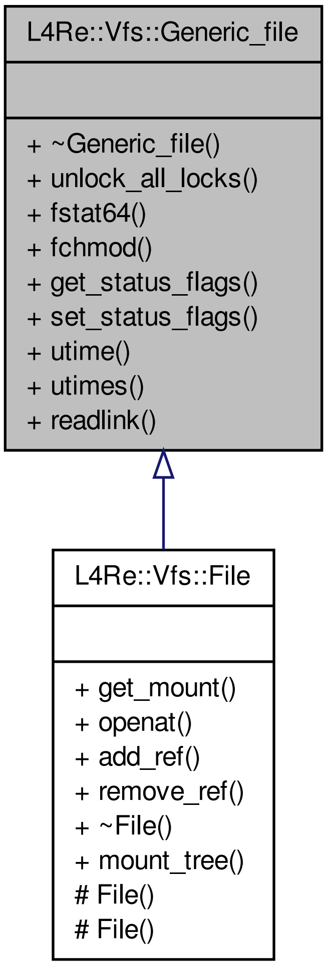
- l4/l4re_vfs/vfs.h

11.19 L4Re::Vfs::Generic_file Class Reference

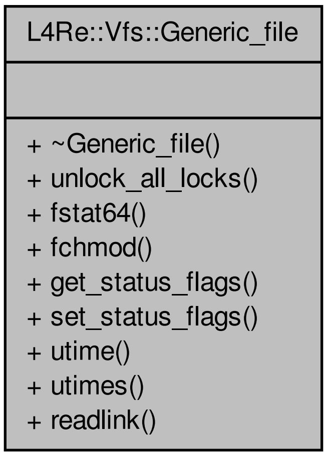
The common interface for an open POSIX file.

`#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).

11.19.1 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.

11.19.2 Member Function Documentation

11.19.2.1 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.

11.19.2.2 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.

Return values

buf	This buffer is filled with the status information.

Returns

0 on success, or <0 on error.

11.19.2.3 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.

11.19.2.4 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_NO←NBLOCK, or <0 on error.

11.19.2.5 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_RDWR, #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.

The documentation for this class was generated from the following file:

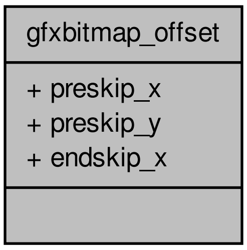
- l4/l4re_vfs/vfs.h

11.20 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

11.20.1 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

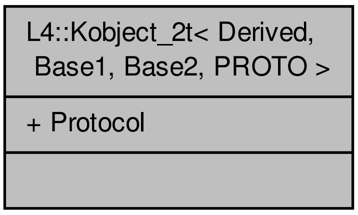
11.21 L4::Kobject_2t< Derived, Base1, Base2, PROTO > Class Template Reference

Helper class to create an L4Re interface class that is derived from two base classes.

`#include <__typeinfo.h>`

Inherits Base1, and Base2.

Collaboration diagram for L4::Kobject_2t< Derived, Base1, Base2, PROTO >:



Friends

- template<typename T >
  Type_info const ∗ kobject_typeid ()

  Get the L4::Type_info for the L4Re interface given in T.

11.21.1 Detailed Description

template<typename Derived, typename Base1, typename Base2, long PROTO = 0>class L4::Kobject_2t< Derived, Base1, Base2, PROTO >

Helper class to create an L4Re interface class that is derived from two base classes.

Parameters

Derived	is the name of the new interface.
Base1	is the name of the interfaces first base class.
Base2	is the name of the interfaces second base class.
PROTO	may be set to the statically assigned protocol number used to communicate with this interface.

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 175 of file __typeinfo.h.

11.21.2 Friends And Related Function Documentation

11.21.2.1 template<typename Derived , typename Base1 , typename Base2 , long PROTO = 0> template<typename T > Type_info const∗ kobject_typeid ( ) [friend]

Get the L4::Type_info for the L4Re interface given in *T*.

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 87 of file __typeinfo.h.

The documentation for this class was generated from the following file:

- l4/sys/__typeinfo.h

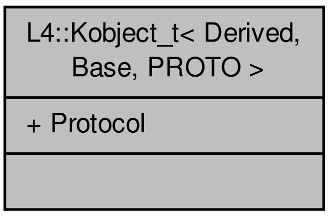
11.22 L4::Kobject_t< Derived, Base, PROTO > Class Template Reference

Helper class to create an L4Re interface class that is derived from a single base class.

```
#include <__typeinfo.h>
```

Inherits Base.

Collaboration diagram for L4::Kobject_t< Derived, Base, PROTO >:



Friends

- template<typename T >
  Type_info const ∗ kobject_typeid ()

  Get the L4::Type_info for the L4Re interface given in T.

11.22.1 Detailed Description

template<typename Derived, typename Base, long PROTO = 0>class L4::Kobject_t< Derived, Base, PROTO >

Helper class to create an L4Re interface class that is derived from a single base class.

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.

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 137 of file __typeinfo.h.

11.22.2 Friends And Related Function Documentation

11.22.2.1 template<typename Derived , typename Base , long PROTO = 0> template<typename T > Type_info const∗ kobject_typeid ( ) [friend]

Get the L4::Type_info for the L4Re interface given in *T*.

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 87 of file __typeinfo.h.

The documentation for this class was generated from the following file:

- l4/sys/__typeinfo.h

11.23 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:

l4_buf_regs_t
+ bdr
+ br

Data Fields

- l4_umword_t bdr

  Buffer descriptor.
- l4_umword_t br [L4_UTCB_GENERIC_BUFFERS_SIZE]

  Buffer registers.

11.23.1 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:

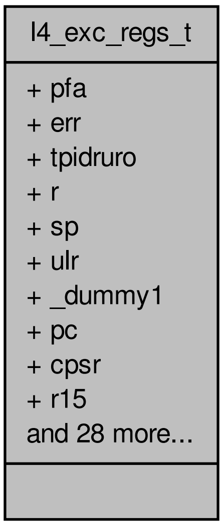
- l4/sys/utcb.h

11.24 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

11.24.1 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.

11.24.2 Field Documentation

11.24.2.1 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:

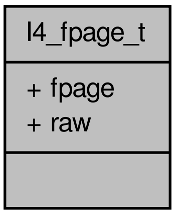
- arm/l4/sys/utcb.h

11.25 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.

11.25.1 Detailed Description

L4 flexpage type.

Definition at line 78 of file __l4_fpage.h.

The documentation for this union was generated from the following file:

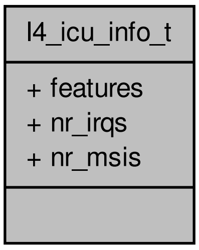
- l4/sys/__l4_fpage.h

11.26 l4_icu_info_t Struct Reference

Info structure for an ICU.

```
#include <icu.h>
```

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,.

11.26.1 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 160 of file icu.h.

11.26.2 Field Documentation

11.26.2.1 unsigned l4_icu_info_t::features

Feature flags.

If L4_ICU_FLAG_MSI is set the ICU supports MSIs.

Definition at line 167 of file icu.h.

The documentation for this struct was generated from the following file:

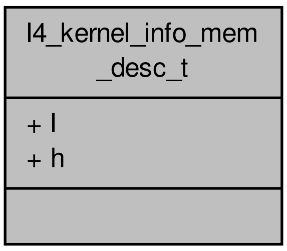
- l4/sys/icu.h

11.27 l4_kernel_info_mem_desc_t Struct Reference

Memory descriptor data structure.

```
#include <memdesc.h>
```

Collaboration diagram for l4_kernel_info_mem_desc_t:



11.27.1 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:

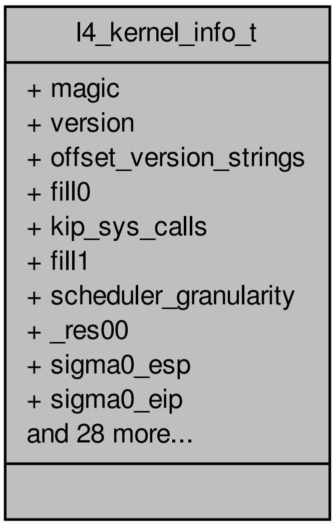
- l4/sys/memdesc.h

11.28 l4_kernel_info_t Struct Reference

L4 Kernel Interface Page.

`#include <__kip-32bit.h>`

Collaboration diagram for l4_kernel_info_t:



Data Fields

- l4_uint32_t magic

  Kernel Info Page identifier ("L4µK").
- l4_uint32_t version

  Kernel version.
- l4_uint8_t offset_version_strings

  offset to version string
- l4_uint8_t fill0 [3]

  reserved

- l4_uint8_t kip_sys_calls

    pointer to system calls
- l4_uint8_t fill1 [3]

    reserved
- l4_umword_t scheduler_granularity

    for rounding time slices
- l4_umword_t _res00 [3]

    default_kdebug_end
- l4_umword_t sigma0_esp

    Sigma0 start stack pointer.
- l4_umword_t sigma0_eip

    Sigma0 instruction pointer.
- l4_umword_t _res01 [2]

    reserved
- l4_umword_t sigma1_esp

    Sigma1 start stack pointer.
- l4_umword_t sigma1_eip

    Sigma1 instruction pointer.
- l4_umword_t _res02 [2]

    reserved
- l4_umword_t root_esp

    Root task stack pointer.
- l4_umword_t root_eip

    Root task instruction pointer.
- l4_umword_t _res03 [2]

    reserved
- l4_umword_t _res50 [1]

    reserved
- l4_umword_t mem_info

    memory information
- l4_umword_t _res58 [2]

    reserved
- l4_umword_t _res04 [16]

    reserved
- l4_umword_t _res05 [2]

    reserved
- l4_umword_t frequency_cpu

    CPU frequency in kHz.
- l4_umword_t frequency_bus

    Bus frequency.
- l4_umword_t _res06 [10]

    reserved
- l4_umword_t user_ptr

    user_ptr
- l4_umword_t vhw_offset

    offset to vhw structure
- l4_uint64_t magic

    Kernel Info Page identifier ("L4µK").
- l4_uint64_t version

    Kernel version.
- l4_uint8_t fill2 [7]

reserved
- l4_uint8_t fill3 [7]

  reserved
- l4_umword_t _res_a0 [1]

  reserved
- l4_umword_t _res_b0 [2]

  reserver

11.28.1 Detailed Description

L4 Kernel Interface Page.

Examples:

examples/sys/ux-vhw/main.c.

Definition at line 38 of file __kip-32bit.h.

The documentation for this struct was generated from the following files:

- l4/sys/__kip-32bit.h
- l4/sys/__kip-64bit.h

11.29 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.

11.29.1 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:

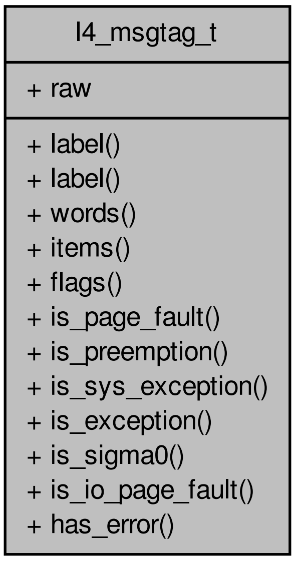
- l4/sys/utcb.h

11.30 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

11.30.1 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/clntsrv/server.cc, 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 158 of file types.h.

11.30.2 Member Function Documentation

11.30.2.1 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 176 of file types.h.

The documentation for this struct was generated from the following file:

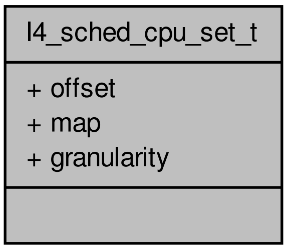
- l4/sys/types.h

11.31 l4_sched_cpu_set_t Struct Reference

CPU sets.

`#include <scheduler.h>`

Collaboration diagram for l4_sched_cpu_set_t:



Data Fields

- l4_umword_t offset

  First CPU of interest (must be aligned to 2^granularity).
- l4_umword_t map

  Bitmap of CPUs.
- unsigned char granularity

  One bit in map represents 2^granularity CPUs.

11.31.1 Detailed Description

CPU sets.

Examples:

examples/sys/migrate/thread_migrate.cc.

Definition at line 40 of file scheduler.h.

The documentation for this struct was generated from the following file:

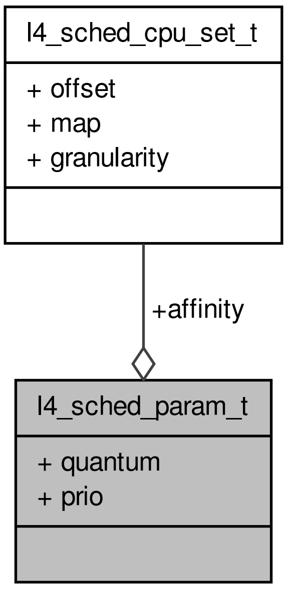
- l4/sys/scheduler.h

11.32 l4_sched_param_t Struct Reference

Scheduler parameter set.

`#include <scheduler.h>`

Collaboration diagram for l4_sched_param_t:



Data Fields

- l4_cpu_time_t quantum

  Timeslice in micro seconds.
- unsigned prio

  Priority for scheduling.
- l4_sched_cpu_set_t affinity

  CPU affinity.

11.32.1 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 101 of file scheduler.h.

The documentation for this struct was generated from the following file:

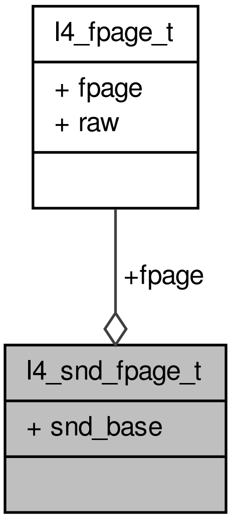
- l4/sys/scheduler.h

11.33 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.

11.33.1 Detailed Description

Send-flex-page types.

Definition at line 95 of file __l4_fpage.h.

The documentation for this struct was generated from the following file:

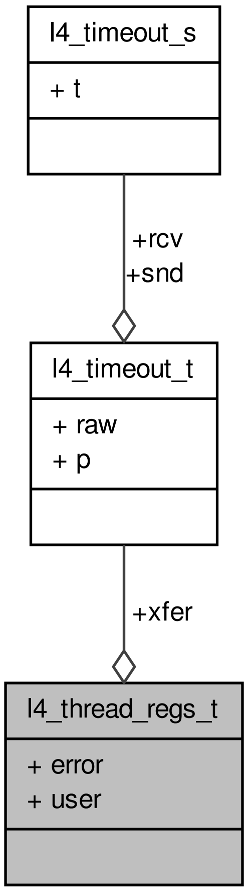
- l4/sys/__l4_fpage.h

11.34 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)

11.34.1 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:

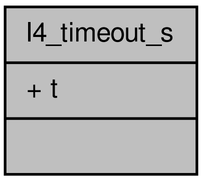
- l4/sys/utcb.h

11.35 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

11.35.1 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:

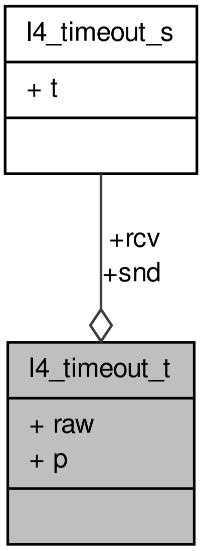
- l4/sys/__timeout.h

11.36 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

11.36.1 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:

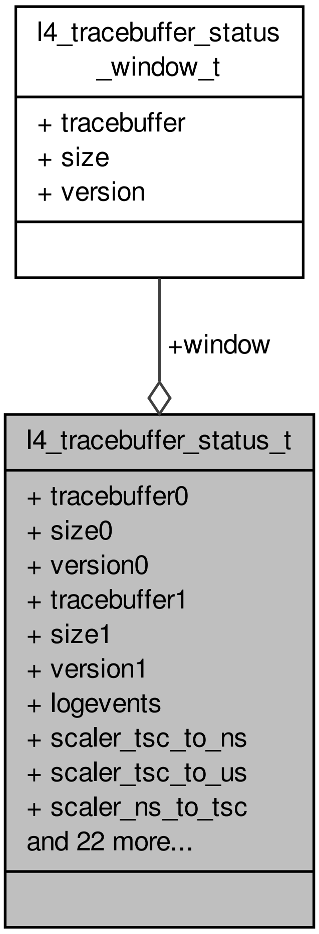
- l4/sys/__timeout.h

11.37 l4_tracebuffer_status_t Struct Reference

Trace buffer status.

```
#include <ktrace.h>
```

Collaboration diagram for l4_tracebuffer_status_t:



Data Fields

- l4_umword_t tracebuffer0

    Address of trace buffer 0.
- l4_umword_t size0

    Size of trace buffer 0.
- l4_umword_t version0

    Version number of trace buffer 0 (incremented if tb0 overruns)
- l4_umword_t tracebuffer1

    Address of trace buffer 1 (there is no gap between tb0 and tb1)
- l4_umword_t size1

    Size of trace buffer 1 (same as tb0)
- l4_umword_t version1

    Version number of trace buffer 1 (incremented if tb1 overruns)
- l4_umword_t logevents [16]

Available LOG events.
- l4_umword_t scaler_tsc_to_ns

  Scaler used for translation of CPU cycles to nano seconds.
- l4_umword_t scaler_tsc_to_us

  Scaler used for translation of CPU cycles to micro seconds.
- l4_umword_t scaler_ns_to_tsc

  Scaler used for translation of nano seconds to CPU cycles.
- l4_umword_t cnt_context_switch

  Number of context switches (intra AS or inter AS)
- l4_umword_t cnt_addr_space_switch

  Number of inter AS context switches.
- l4_umword_t cnt_shortcut_failed

  How often was the IPC shortcut not taken.
- l4_umword_t cnt_shortcut_success

  How often was the IPC shortcut taken.
- l4_umword_t cnt_irq

  Number of hardware interrupts (without kernel scheduling interrupt)
- l4_umword_t cnt_ipc_long

  Number of long IPCs.
- l4_umword_t cnt_page_fault

  Number of page faults.
- l4_umword_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)
- l4_umword_t cnt_task_create

  Number of tasks created.
- l4_umword_t schedule

  Number of reschedules.
- volatile l4_tracebuffer_entry_t ∗ current_entry

  Address of the most current event in trace-buffer.
- volatile l4_umword_t cnt_context_switch

  Number of context switches (intra AS or inter AS)
- volatile l4_umword_t cnt_addr_space_switch

  Number of inter AS context switches.
- volatile l4_umword_t cnt_shortcut_failed

  How often was the IPC shortcut taken.
- volatile l4_umword_t cnt_shortcut_success

  How often was the IPC shortcut not taken.
- volatile l4_umword_t cnt_irq

  Number of hardware interrupts (without kernel scheduling interrupt)
- volatile l4_umword_t cnt_ipc_long

  Number of long IPCs.
- volatile l4_umword_t cnt_page_fault

  Number of page faults.
- volatile l4_umword_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 in the I/O bitmap)
- volatile l4_umword_t cnt_task_create

  Number of tasks created.
- volatile l4_umword_t cnt_schedule

  Number of reschedules.
- volatile l4_umword_t cnt_iobmap_tlb_flush

  Number of flushes of the I/O bitmap.

11.37.1 Detailed Description

Trace buffer status.

Trace-buffer status.

Tracebuffer status.

Definition at line 67 of file ktrace.h.

11.37.2 Field Documentation

11.37.2.1 l4_umword_t l4_tracebuffer_status_t::tracebuffer0

Address of trace buffer 0.

Address of tracebuffer 0.

Definition at line 70 of file ktrace.h.

11.37.2.2 l4_umword_t l4_tracebuffer_status_t::size0

Size of trace buffer 0.

Size of tracebuffer 0.

Definition at line 72 of file ktrace.h.

11.37.2.3 l4_umword_t l4_tracebuffer_status_t::version0

Version number of trace buffer 0 (incremented if tb0 overruns)

Version number of tracebuffer 0 (incremented if tb0 overruns)

Definition at line 74 of file ktrace.h.

11.37.2.4 l4_umword_t l4_tracebuffer_status_t::tracebuffer1

Address of trace buffer 1 (there is no gap between tb0 and tb1)

Address of tracebuffer 1 (there is no gap between tb0 and tb1)

Definition at line 76 of file ktrace.h.

11.37.2.5 l4_umword_t l4_tracebuffer_status_t::size1

Size of trace buffer 1 (same as tb0)

Size of tracebuffer 1 (same as tb0)

Definition at line 78 of file ktrace.h.

11.37.2.6 l4_umword_t l4_tracebuffer_status_t::version1

Version number of trace buffer 1 (incremented if tb1 overruns)

Version number of tracebuffer 1 (incremented if tb1 overruns)

Definition at line 80 of file ktrace.h.

11.37.2.7 volatile l4_umword_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 99 of file ktrace.h.

The documentation for this struct was generated from the following file:

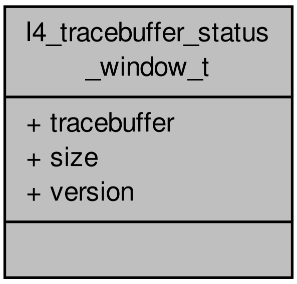
- arm/l4/sys/ktrace.h

11.38 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)

11.38.1 Detailed Description

Trace-buffer status window descriptor.

Definition at line 45 of file ktrace.h.

The documentation for this struct was generated from the following file:

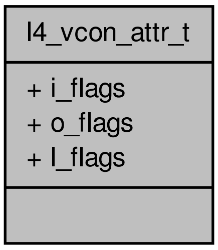
- x86/l4/sys/ktrace.h

11.39 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

11.39.1 Detailed Description

Vcon attribute structure.

Definition at line 158 of file vcon.h.

The documentation for this struct was generated from the following file:

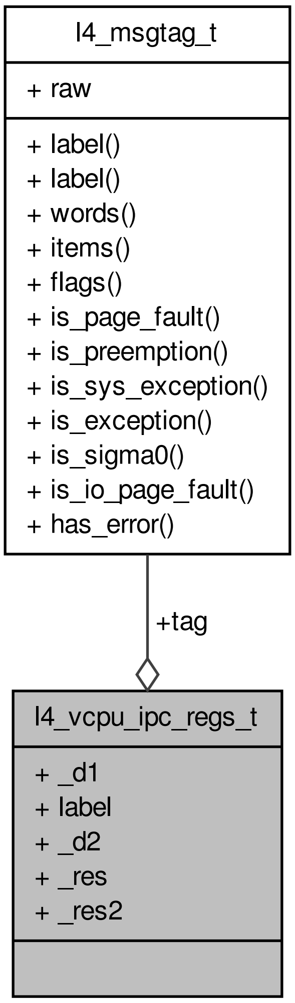
- l4/sys/vcon.h

11.40 l4_vcpu_ipc_regs_t Struct Reference

vCPU message registers.

`#include <__vcpu-arch.h>`

Collaboration diagram for l4_vcpu_ipc_regs_t:



11.40.1 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:

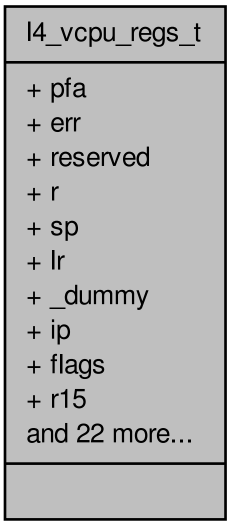
- arm/l4/sys/__vcpu-arch.h

11.41 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

11.41.1 Detailed Description

vCPU registers.

Definition at line 27 of file __vcpu-arch.h.

11.41.2 Field Documentation

11.41.2.1 l4_umword_t l4_vcpu_regs_t::di

rdi register

edi register

Definition at line 54 of file __vcpu-arch.h.

11.41.2.2 l4_umword_t l4_vcpu_regs_t::si

rsi register

esi register

Definition at line 55 of file __vcpu-arch.h.

11.41.2.3 l4_umword_t l4_vcpu_regs_t::bp

rbp register

ebp register

Definition at line 56 of file __vcpu-arch.h.

11.41.2.4 l4_umword_t l4_vcpu_regs_t::bx

rbx register

ebx register

Definition at line 58 of file __vcpu-arch.h.

11.41.2.5 l4_umword_t l4_vcpu_regs_t::dx

rdx register

edx register

Definition at line 59 of file __vcpu-arch.h.

11.41.2.6 l4_umword_t l4_vcpu_regs_t::cx

rcx register

ecx register

Definition at line 60 of file __vcpu-arch.h.

11.41.2.7 l4_umword_t l4_vcpu_regs_t::ax

rax register

eax register

Definition at line 61 of file __vcpu-arch.h.

The documentation for this struct was generated from the following file:

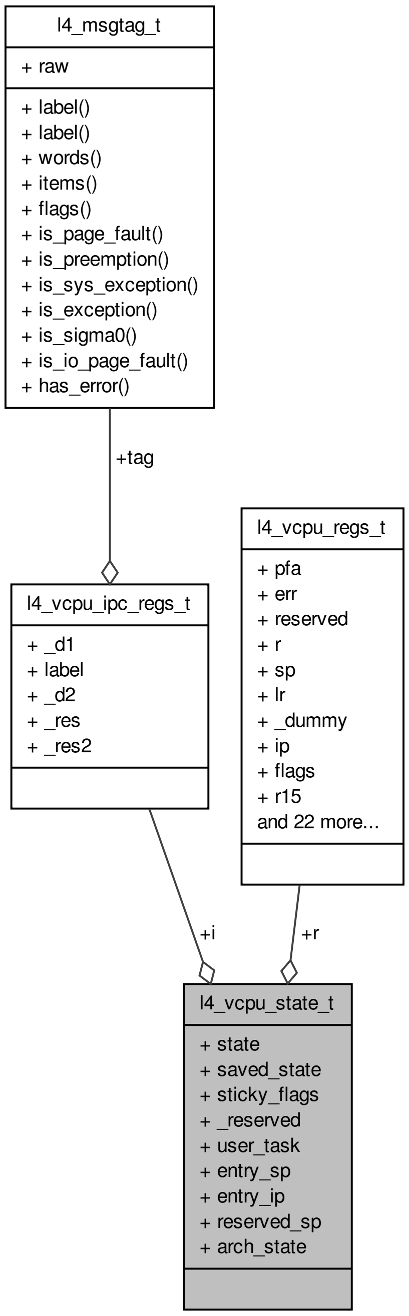
- arm/l4/sys/__vcpu-arch.h

11.42 l4_vcpu_state_t Struct Reference

State of a vCPU.

```
#include <vcpu.h>
```

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.

11.42.1 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:

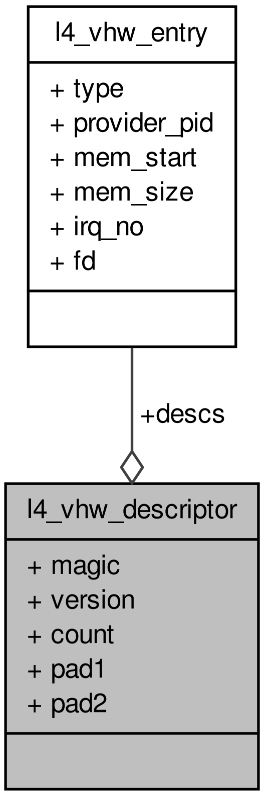
- l4/sys/vcpu.h

11.43 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.

11.43.1 Detailed Description

Virtual hardware devices description.

Examples:

examples/sys/ux-vhw/main.c.

Definition at line 70 of file vhw.h.

11.43.2 Field Documentation

11.43.2.1 l4_uint32_t l4_vhw_descriptor::magic

Magic.

Examples:

examples/sys/ux-vhw/main.c.

Definition at line 71 of file vhw.h.

11.43.2.2 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.

11.43.2.3 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.

11.43.2.4 struct l4_vhw_entry l4_vhw_descriptor::descs[ ]

Array of device descriptions.

Definition at line 77 of file vhw.h.

The documentation for this struct was generated from the following file:

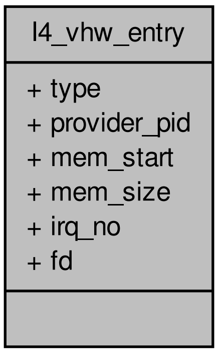
- l4/sys/vhw.h

11.44 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.

11.44.1 Detailed Description

Description of a device.

Examples:

examples/sys/ux-vhw/main.c.

Definition at line 55 of file vhw.h.

11.44.2 Field Documentation

11.44.2.1 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.

11.44.2.2 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.

11.44.2.3 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.

11.44.2.4 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.

11.44.2.5 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.

11.44.2.6 l4_uint32_t l4_vhw_entry::fd

File descriptor.

Definition at line 63 of file vhw.h.

The documentation for this struct was generated from the following file:

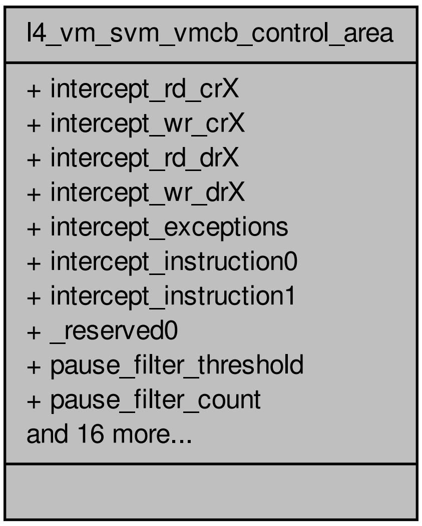
- l4/sys/vhw.h

11.45 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:



11.45.1 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:

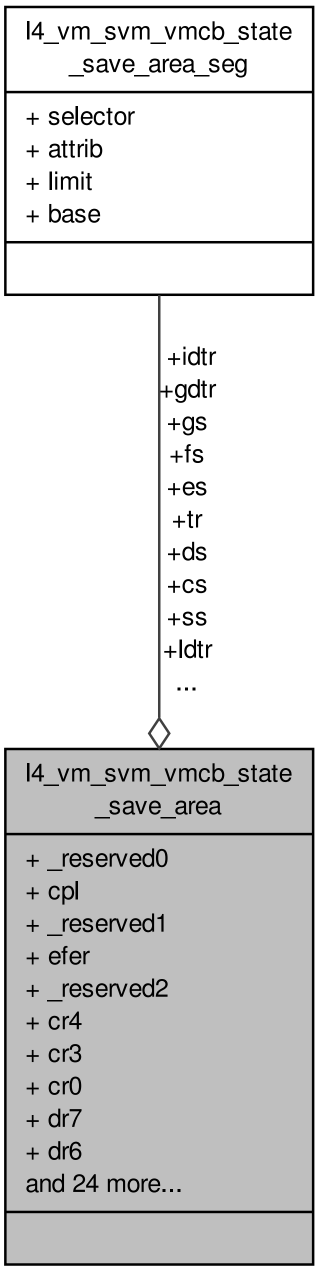
- l4/sys/__vm-svm.h

11.46 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:



11.46.1 Detailed Description

State save area structure for SVM VMs.

Definition at line 94 of file __vm-svm.h.

The documentation for this struct was generated from the following file:

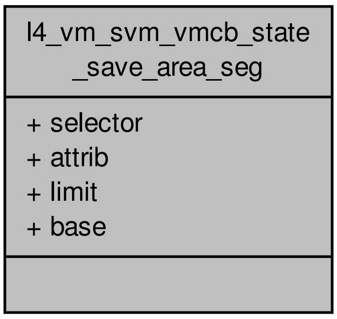
- l4/sys/__vm-svm.h

11.47 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:



11.47.1 Detailed Description

State save area segment selector struct.

Definition at line 82 of file __vm-svm.h.

The documentation for this struct was generated from the following file:

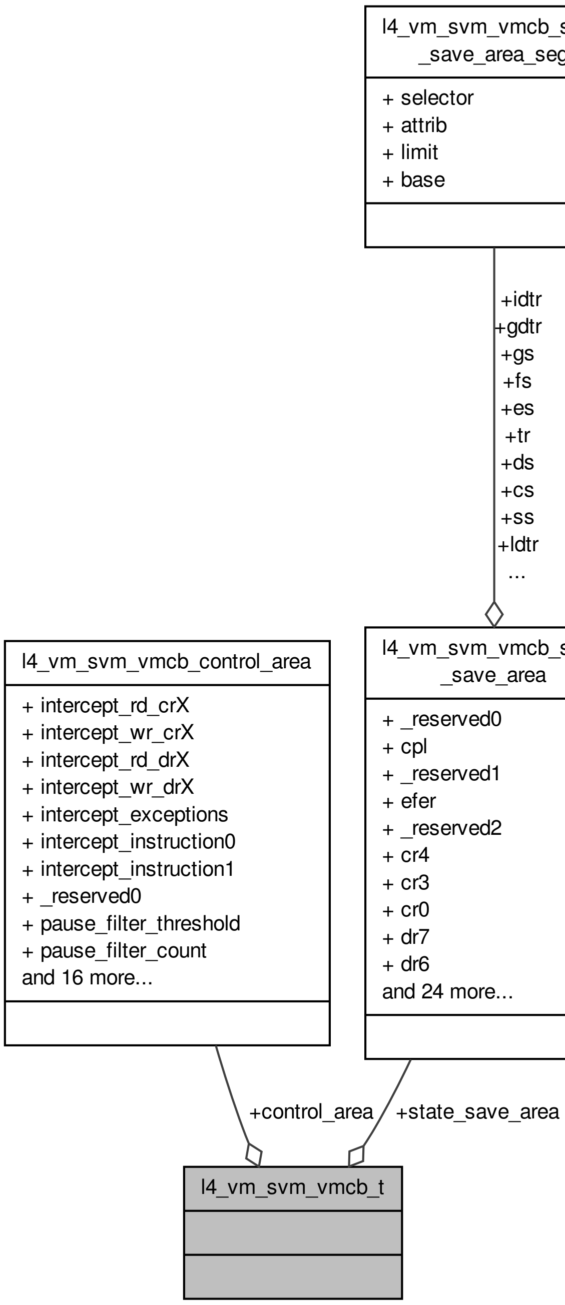
- l4/sys/__vm-svm.h

11.48 l4_vm_svm_vmcb_t Struct Reference

Control structure for SVM VMs.

`#include <__vm-svm.h>`

Collaboration diagram for l4_vm_svm_vmcb_t:



11.48.1 Detailed Description

Control structure for SVM VMs.

Definition at line 163 of file __vm-svm.h.

The documentation for this struct was generated from the following file:

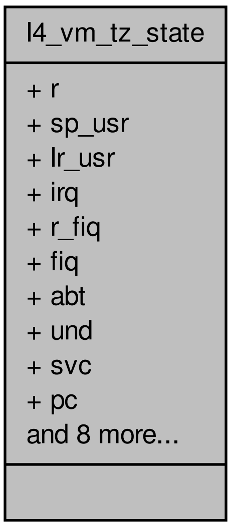
- l4/sys/__vm-svm.h

11.49 l4_vm_tz_state Struct Reference

state structure for TrustZone VMs

`#include <vm.h>`

Collaboration diagram for l4_vm_tz_state:



11.49.1 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

11.50 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.

11.50.1 Detailed Description

Auxiliary descriptor.

Definition at line 51 of file l4aux.h.

The documentation for this struct was generated from the following file:

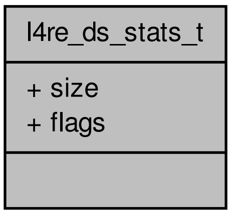
- l4/re/l4aux.h

11.51 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

11.51.1 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:

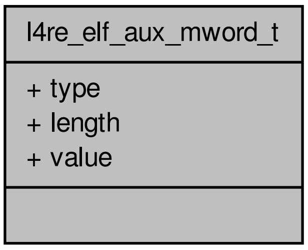
- l4/re/c/dataspace.h

11.52 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:



11.52.1 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:

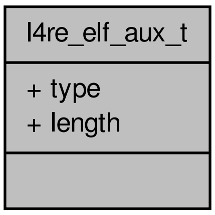
- l4/re/elf_aux.h

11.53 l4re_elf_aux_t Struct Reference

Generic header for each auxiliary vector element.

`#include <elf_aux.h>`

Collaboration diagram for l4re_elf_aux_t:



11.53.1 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:

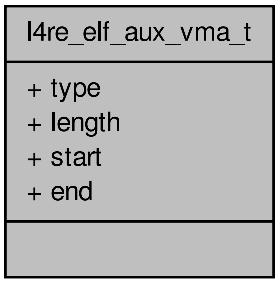
- l4/re/elf_aux.h

11.54 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:



11.54.1 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:

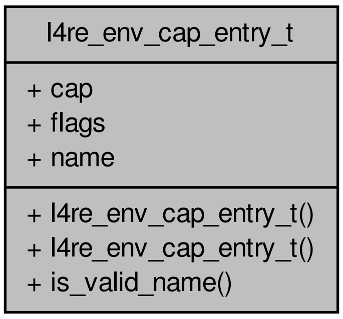
- l4/re/elf_aux.h

11.55 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.

11.55.1 Detailed Description

Entry in the L4Re environment array for the named inital objects.

Definition at line 35 of file env.h.

11.55.2 Constructor & Destructor Documentation

11.55.2.1 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.

11.55.3 Field Documentation

11.55.3.1 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.

The documentation for this struct was generated from the following file:

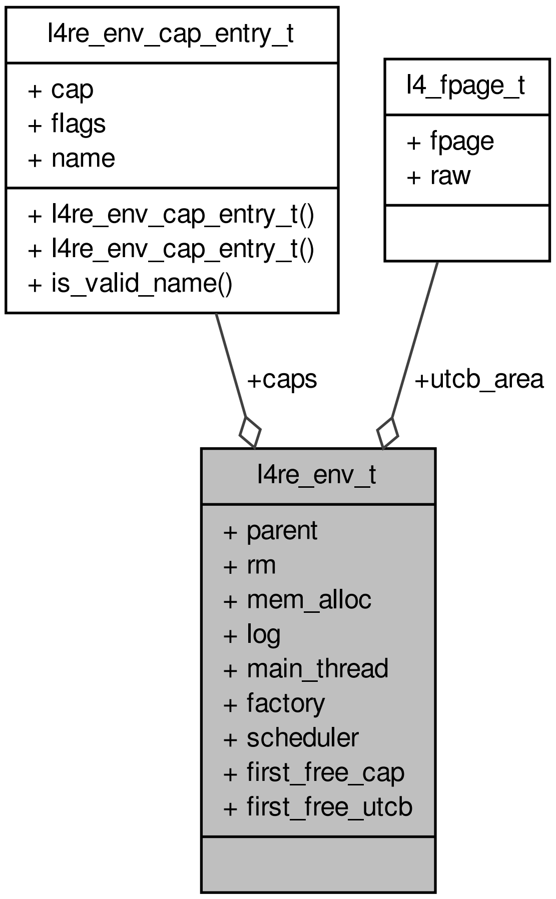
- l4/re/env.h

11.56 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.

11.56.1 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:

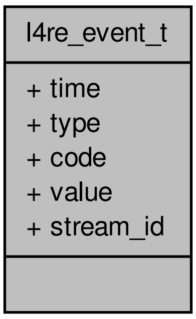
- l4/re/env.h

11.57 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.

11.57.1 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:

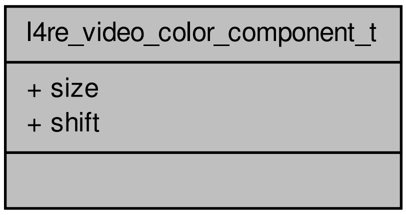
- l4/re/c/event.h

11.58 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

11.58.1 Detailed Description

Color component structure.

Definition at line 29 of file colors.h.

The documentation for this struct was generated from the following file:

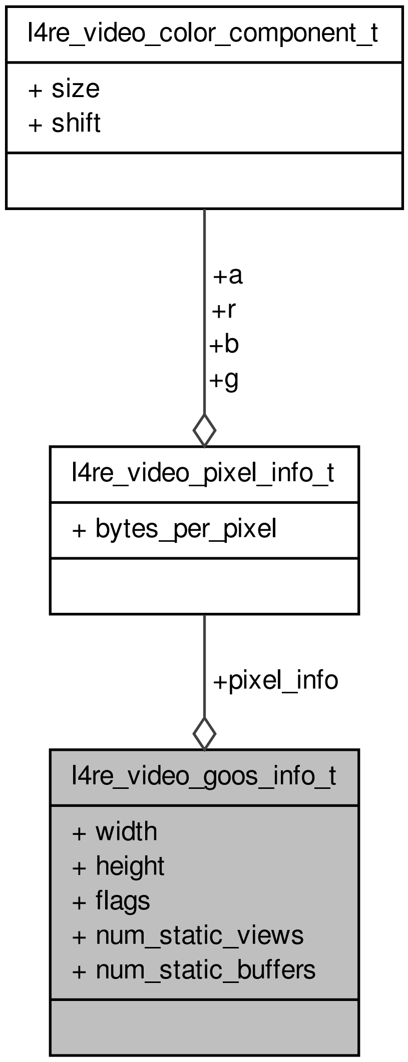
- l4/re/c/video/colors.h

11.59 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.

- 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.

11.59.1 Detailed Description

Goos information structure.

Definition at line 51 of file goos.h.

The documentation for this struct was generated from the following file:

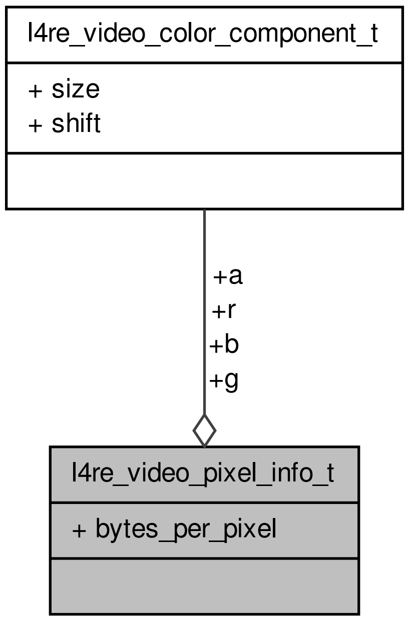
- l4/re/c/video/goos.h

11.60 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.

11.60.1 Detailed Description

Pixel_info structure.

Definition at line 39 of file colors.h.

The documentation for this struct was generated from the following file:

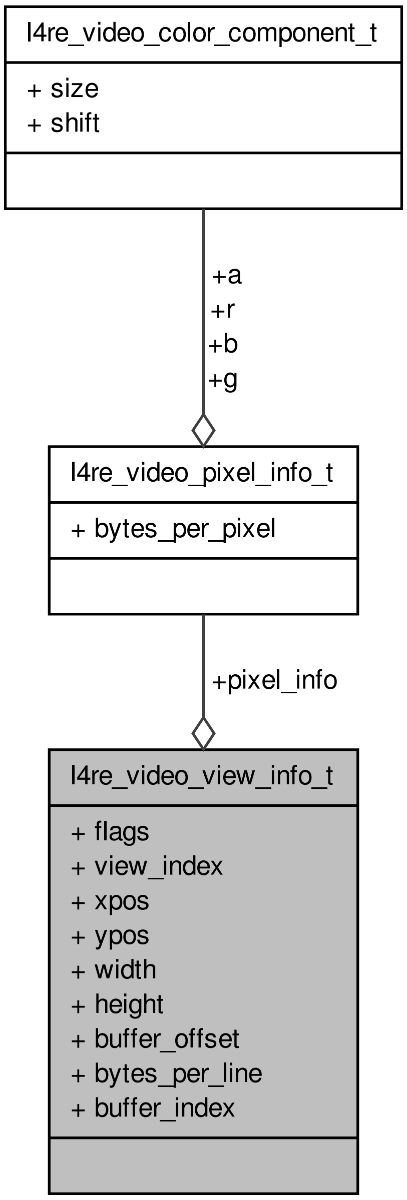
- l4/re/c/video/colors.h

11.61 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.

11.61.1 Detailed Description

View information structure.

Definition at line 59 of file view.h.

The documentation for this struct was generated from the following file:

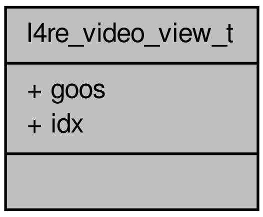
- l4/re/c/video/view.h

11.62 l4re_video_view_t Struct Reference

C representation of a goos view.

```
#include <view.h>
```

Collaboration diagram for l4re_video_view_t:



11.62.1 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:

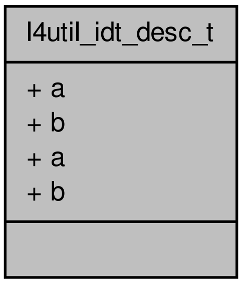
- l4/re/c/video/view.h

11.63 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

11.63.1 Detailed Description

IDT entry.

Definition at line 33 of file idt.h.

The documentation for this struct was generated from the following file:

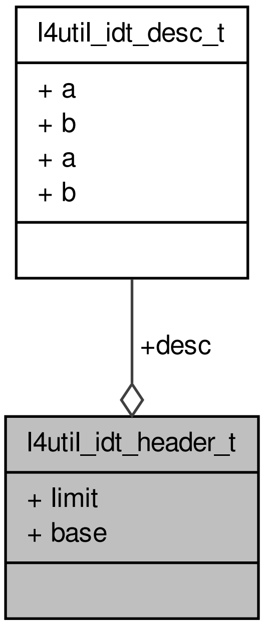
- amd64/l4/util/idt.h

11.64 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)

11.64.1 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:

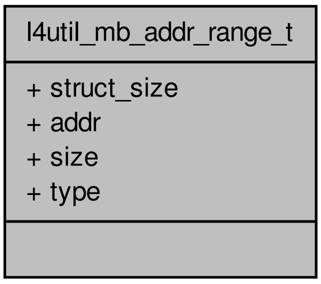
- amd64/l4/util/idt.h

11.65 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

11.65.1 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:

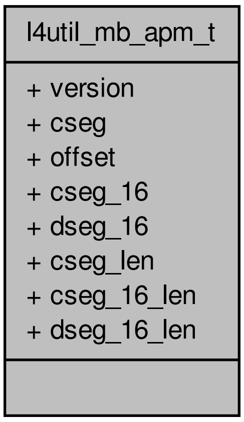
- l4/util/mb_info.h

11.66 l4util_mb_apm_t Struct Reference

APM BIOS info.

```
#include <mb_info.h>
```

Collaboration diagram for l4util_mb_apm_t:



11.66.1 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:

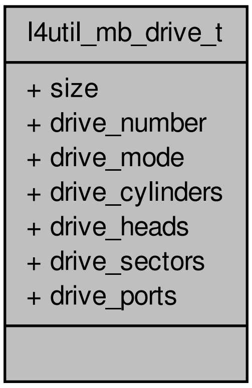
- l4/util/mb_info.h

11.67 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

11.67.1 Detailed Description

Drive Info structure.

Definition at line 74 of file mb_info.h.

11.67.2 Field Documentation

11.67.2.1 l4_uint8_t l4util_mb_drive_t::drive_number

<The size of this structure.

Definition at line 77 of file mb_info.h.

11.67.2.2 l4_uint8_t l4util_mb_drive_t::drive_mode

<The BIOS drive number.

Definition at line 78 of file mb_info.h.

11.67.2.3 l4_uint16_t l4util_mb_drive_t::drive_cylinders

<The access mode (see below).

Definition at line 79 of file mb_info.h.

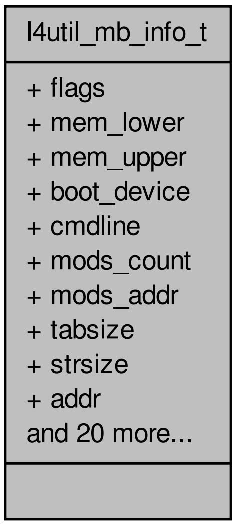
The documentation for this struct was generated from the following file:

- l4/util/mb_info.h

11.68 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

11.68.1 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.

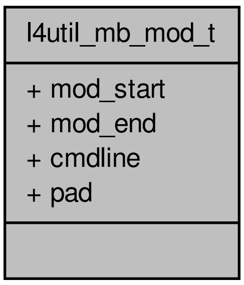
The documentation for this struct was generated from the following file:

- l4/util/mb_info.h

11.69 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

11.69.1 Detailed Description

The structure type "mod_list" is used by the multiboot_info structure.

Definition at line 27 of file mb_info.h.

11.69.2 Field Documentation

11.69.2.1 l4_uint32_t l4util_mb_mod_t::mod_start

Starting address of module in memory.

Definition at line 29 of file mb_info.h.

11.69.2.2 l4_uint32_t l4util_mb_mod_t::mod_end

End address of module in memory.

Definition at line 30 of file mb_info.h.

The documentation for this struct was generated from the following file:

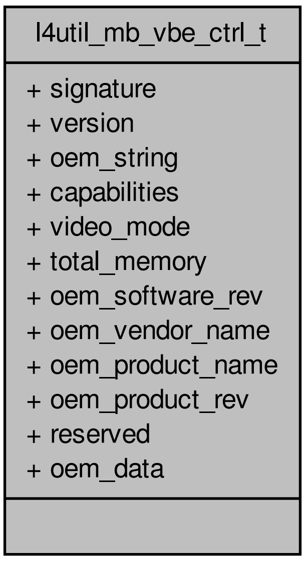
- l4/util/mb_info.h

11.70 l4util_mb_vbe_ctrl_t Struct Reference

VBE controller information.

`#include <mb_info.h>`

Collaboration diagram for l4util_mb_vbe_ctrl_t:



11.70.1 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:

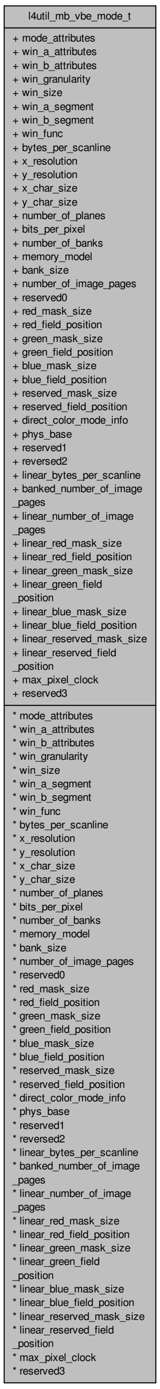
- l4/util/mb_info.h

11.71 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]

11.71.1 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:

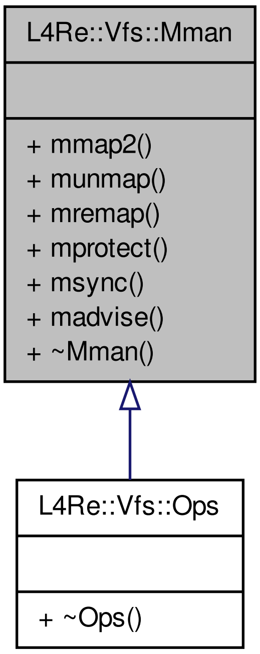
- l4/util/mb_info.h

11.72 L4Re::Vfs::Mman Class Reference

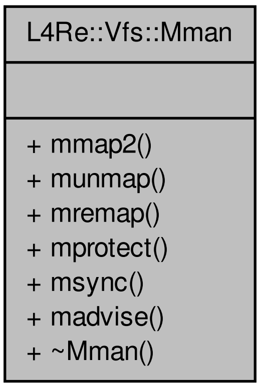
Interface for the POSIX memory management.

`#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_adr)=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.

11.72.1 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 785 of file vfs.h.

The documentation for this class was generated from the following file:

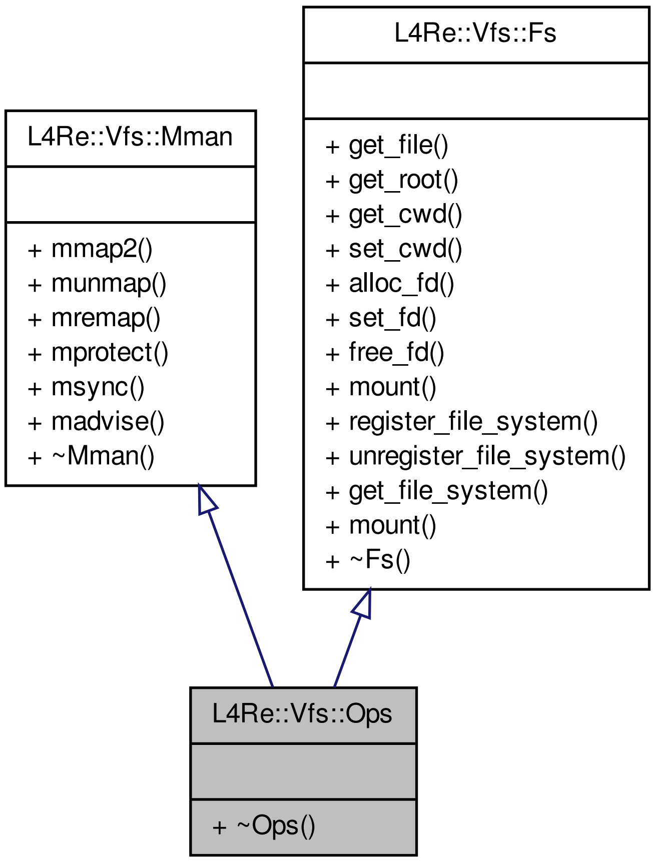
- l4/l4re_vfs/vfs.h

11.73 L4Re::Vfs::Ops Class Reference

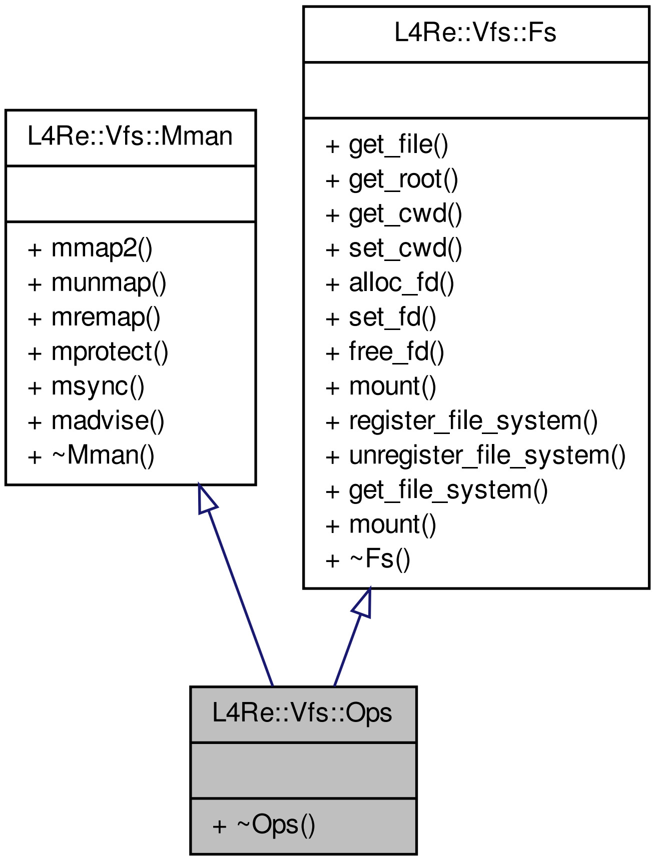
Interface for the POSIX backends for an application.

```
#include <vfs.h>
```

Inheritance diagram for L4Re::Vfs::Ops:



Collaboration diagram for L4Re::Vfs::Ops:



Additional Inherited Members

11.73.1 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 1016 of file vfs.h.

The documentation for this class was generated from the following file:

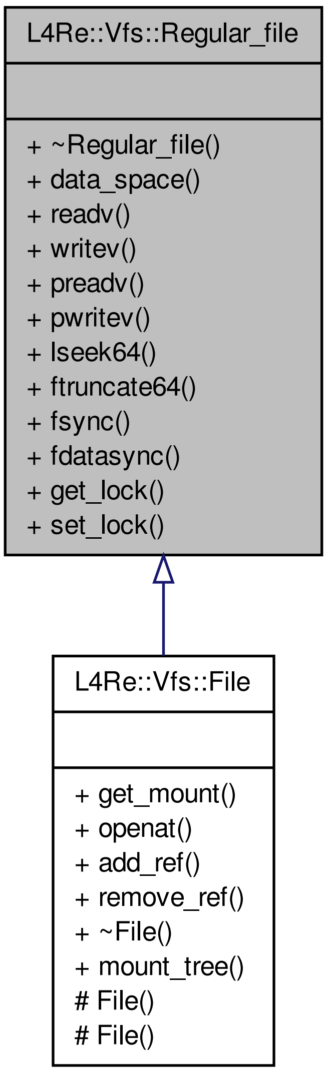
- l4/l4re_vfs/vfs.h

11.74 L4Re::Vfs::Regular_file Class Reference

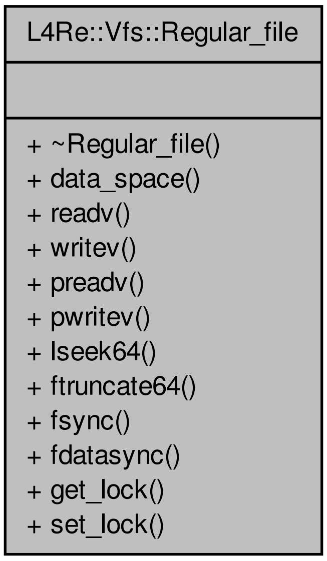
Interface for a POSIX file that provides regular file semantics.

```
#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.

11.74.1 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.

11.74.2 Member Function Documentation

11.74.2.1 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.

11.74.2.2 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-

11.74.2.3 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.

11.74.2.4 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.

11.74.2.5 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.

11.74.2.6 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.

11.74.2.7 virtual int L4Re::Vfs::Regular_file::fdatasync ( ) const throw) [pure virtual]

Sync the data to persistent storage.

This is the backend for POSIX fdatasync.

11.74.2.8 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.

11.74.2.9 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.

The documentation for this class was generated from the following file:

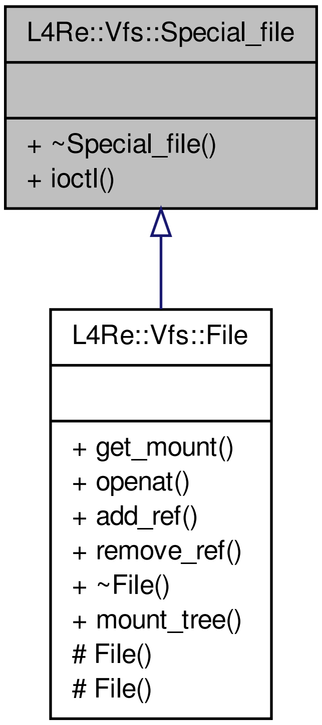
- l4/l4re_vfs/vfs.h

11.75 L4Re::Vfs::Special_file Class Reference

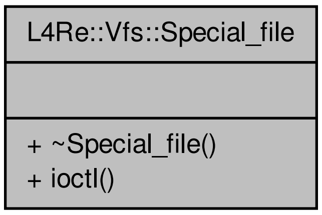
Interface for a POSIX file that provides special file semantics.

`#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.

11.75.1 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.

11.75.2 Member Function Documentation

11.75.2.1 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.

The documentation for this class was generated from the following file:

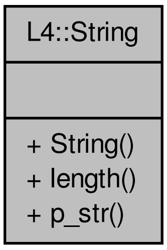
- l4/l4re_vfs/vfs.h

11.76 L4::String Class Reference

A null-terminated string container class.

`#include <string.h>`

Collaboration diagram for L4::String:



11.76.1 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:

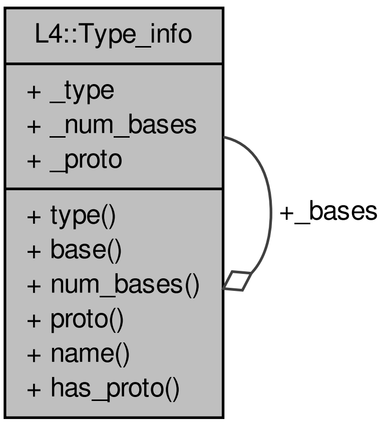
- l4/cxx/string.h

11.77 L4::Type_info Struct Reference

Dynamic Type Information for L4Re Interfaces.

`#include <__typeinfo.h>`

Collaboration diagram for L4::Type_info:



11.77.1 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 50 of file __typeinfo.h.

The documentation for this struct was generated from the following file:

- l4/sys/__typeinfo.h

Chapter 12

Example Documentation

12.1 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 <l4/cxx/ipc_stream>

#include <stdio.h>

#include "shared.h"

static int
func_neg_call(L4::Cap<void> const &server, l4_uint32_t *result,
              l4_uint32_t val)
{
  L4::Ipc::Iostream s(l4_utcb());
  s << l4_umword_t(Opcode::func_neg) << val;
  int r = l4_error(s.call(server.cap(), Protocol::Calc));
  if (r)
    return r; // failure
  s >> *result;
  return 0; // ok
}

static int
func_sub_call(L4::Cap<void> const &server, l4_uint32_t *result,
              l4_uint32_t val1, l4_uint32_t val2)
{
  L4::Ipc::Iostream s(l4_utcb());
  s << l4_umword_t(Opcode::func_sub) << val1 << val2;
  int r = l4_error(s.call(server.cap(), Protocol::Calc));
  if (r)
    return r; // failure
  s >> *result;
  return 0; // ok
}

int
main()
{
  L4::Cap<void> server = L4Re::Env::env()->get_cap<void>("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 (func_sub_call(server, &val1, val1, val2))
    {
      printf("Error talking to server\n");
      return 1;
    }
  printf("Result of substract call: %d\n", val1);
  printf("Asking for -%d\n", val1);
  if (func_neg_call(server, &val1, val1))
    {
      printf("Error talking to server\n");
      return 1;
    }
  printf("Result of negate call: %d\n", val1);

  return 0;
}
```

12.2 examples/clntsrv/clntsrv.cfg

Sample configuration file for the client/server example.

```
-- vim:set ft=lua:

-- Include L4 functionality
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");
```

12.3 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/cxx/ipc_server>

#include "shared.h"

static L4Re::Util::Registry_server<> server;

class Calculation_server : public L4::Server_object
{
public:
  int dispatch(l4_umword_t obj, L4::Ipc::Iostream &ios);
};
```

```
int
Calculation_server::dispatch(l4_umword_t, L4::Ipc::Iostream &ios)
{
  l4_msgtag_t t;
  ios >> t;

  // We're only talking the calculation protocol
  if (t.label() != Protocol::Calc)
    return -L4_EBADPROTO;

  L4::Opcode opcode;
  ios >> opcode;

  switch (opcode)
    {
    case Opcode::func_neg:
      l4_uint32_t val;
      ios >> val;
      val = -val;
      ios << val;
      return L4_EOK;
    case Opcode::func_sub:
      l4_uint32_t val1, val2;
      ios >> val1 >> val2;
      val1 -= val2;
      ios << val1;
      return L4_EOK;
    default:
      return -L4_ENOSYS;
    }
}

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;
}
```

12.4 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, 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;
}
```

12.5 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, 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;
}
```

12.6 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
{
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::Server_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,
                                        *_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;
}
```

12.7 examples/libs/l4re/c++/shared_ds/shared_ds.lua

Sharing memory between applications, configuration file.

```

-- Include L4 functionality
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"
);

-- Start the client, giving it the channel with read only rights. The
-- log output will be green.
L4.default_loader:start(
  {
    caps = { shm = channel },
    log  = { "client", "green"  },
    l4re_dbg = L4.Dbg.Warn
  },
  "rom/ex_l4re_ds_clnt"
);
```

12.8 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;
}
```

12.9 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_umword_t(Opcode::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(), Protocol::Map_example));
  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;
}
```

12.10 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
{
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() != Protocol::Map_example)
    return -L4_EBADPROTO;

  L4::Opcode opcode;
  ios >> opcode;

  switch (opcode)
    {
    case Opcode::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;
}
```

12.11 examples/libs/l4re/streammap/streammap.cfg

Sample configuration file for the client/server map example.

```
-- vim:set ft=lua:

-- Include L4 functionality
require("L4");

-- Channel for the communication between the server and the client.
local smap_channel = L4.default_loader:new_channel();

-- The server program, using the 'smap' channel in server
-- mode. The log prefix will be 'server', colored yellow.
L4.default_loader:start({ caps = { smap = smap_channel:svr() },
                          log = { "server", "yellow" }},
                        "rom/ex_smap-server");


-- The client program.
-- It is given the 'smap' channel to be able to talk to the server.
-- The log prefix will be 'client', colored green.
L4.default_loader:start({ caps = { smap = smap_channel },
                          log = { "client", "green" } },
                        "rom/ex_smap-client");
```

12.12 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;
}
```

12.13 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;
}
```

12.14 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_getl4cap(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_getl4cap(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;
}
```

12.15 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;
}
```

12.16 examples/sys/ipc/ipc.cfg

Sample configuration file for the IPC example.

```
# vim:se ft=lua:

require("L4");

L4.default_loader:start({}, "rom/ex_ipc1");
```

12.17 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_getl4cap(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;
}
```

12.18 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;
}
```

12.19 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_getl4cap(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;
}
```

12.20 examples/sys/migrate/thread_migrate.cfg

Sample configuration file for the thread migration example.

```
-- vim:set ft=lua:

-- The log prefix will be 'migrate', colored green.
L4.default_loader:start({ log = { "migrate", "green" } },
                         "rom/ex_thread_migrate");
```

12.21 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;
}
```

12.22 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 architecture.
 */

#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 %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"
" pusha     \n\t"
" push %esp   \n\t"
" call thread_func  \n\t"
);
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;
  extern char _start[], _end[], _sdata[];

  if (l4_is_invalid_cap(t1))
    return 1;

  /* Prevent pagefaults of our new thread because we do not want to
   * implement a pager as well. */
  l4_touch_ro(_start, _sdata - _start + 1);
  l4_touch_rw(_sdata, _end - _sdata);

  /* 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->ip = (l4_umword_t)thread;
  e->sp = (l4_umword_t)(thread_stack + sizeof(thread_stack));
  e->eax = 1;
  e->ebx = 4;
  e->ecx = 2;
  e->edx = 3;
  e->esi = 6;
  e->edi = 7;
  e->ebp = 5;
  /* 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;
}
```

12.23 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;
}
```

12.24 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;
}
```

12.25 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);
    }
}
```

12.26 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 = 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 = new Tmpfs_dir(cxx::ref_ptr_static_cast<Pers_dir>(path));
  else
    *file = 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