

Jinue

Generated by Doxygen 1.8.5

Wed Mar 20 2019 21:26:14



# Contents

<b>1</b>	<b>Data Structure Index</b>	<b>1</b>
1.1	Data Structures . . . . .	1
<b>2</b>	<b>File Index</b>	<b>3</b>
2.1	File List . . . . .	3
<b>3</b>	<b>Data Structure Documentation</b>	<b>7</b>
3.1	addr_space_t Struct Reference . . . . .	7
3.1.1	Detailed Description . . . . .	8
3.1.2	Field Documentation . . . . .	8
3.1.2.1	cr3 . . . . .	8
3.1.2.2	pd . . . . .	8
3.1.2.3	pdpt . . . . .	8
3.1.2.4	top_level . . . . .	8
3.2	alloc_page Struct Reference . . . . .	8
3.2.1	Detailed Description . . . . .	9
3.2.2	Field Documentation . . . . .	9
3.2.2.1	next . . . . .	9
3.3	boot_alloc_t Struct Reference . . . . .	9
3.3.1	Detailed Description . . . . .	9
3.3.2	Field Documentation . . . . .	10
3.3.2.1	heap_ptr . . . . .	10
3.3.2.2	heap_pushed_state . . . . .	10
3.3.2.3	its_early . . . . .	10
3.3.2.4	kernel_paddr_limit . . . . .	10
3.3.2.5	kernel_paddr_top . . . . .	10
3.3.2.6	kernel_vm_limit . . . . .	10
3.3.2.7	kernel_vm_top . . . . .	10
3.4	boot_heap_pushed_state Struct Reference . . . . .	10

3.4.1	Detailed Description	11
3.4.2	Field Documentation	11
3.4.2.1	next	11
3.5	boot_info_t Struct Reference	11
3.5.1	Detailed Description	12
3.5.2	Field Documentation	12
3.5.2.1	boot_end	12
3.5.2.2	boot_heap	12
3.5.2.3	cmdline	12
3.5.2.4	e820_entries	12
3.5.2.5	e820_map	12
3.5.2.6	image_start	13
3.5.2.7	image_top	13
3.5.2.8	kernel_size	13
3.5.2.9	kernel_start	13
3.5.2.10	page_directory	13
3.5.2.11	page_table	13
3.5.2.12	proc_size	13
3.5.2.13	proc_start	13
3.5.2.14	ramdisk_size	13
3.5.2.15	ramdisk_start	14
3.5.2.16	setup_signature	14
3.6	cpu_data_t Struct Reference	14
3.6.1	Detailed Description	15
3.6.2	Field Documentation	15
3.6.2.1	current_addr_space	15
3.6.2.2	gdt	15
3.6.2.3	self	15
3.6.2.4	tss	15
3.7	cpu_info_t Struct Reference	15
3.7.1	Detailed Description	15
3.7.2	Field Documentation	16
3.7.2.1	dcache_alignment	16
3.7.2.2	family	16
3.7.2.3	features	16
3.7.2.4	model	16
3.7.2.5	stepping	16

3.7.2.6	vendor	16
3.8	e820_t Struct Reference	16
3.8.1	Detailed Description	17
3.8.2	Field Documentation	17
3.8.2.1	addr	17
3.8.2.2	size	17
3.8.2.3	type	17
3.9	Elf32_auxv_t Struct Reference	17
3.9.1	Detailed Description	17
3.9.2	Field Documentation	17
3.9.2.1	a_type	17
3.9.2.2	a_un	18
3.9.2.3	a_val	18
3.10	Elf32_Ehdr Struct Reference	18
3.10.1	Detailed Description	18
3.10.2	Field Documentation	18
3.10.2.1	e_ehsize	18
3.10.2.2	e_entry	19
3.10.2.3	e_flags	19
3.10.2.4	e_ident	19
3.10.2.5	e_machine	19
3.10.2.6	e_phentsize	19
3.10.2.7	e_phnum	19
3.10.2.8	e_phoff	19
3.10.2.9	e_shentsize	19
3.10.2.10	e_shnum	19
3.10.2.11	e_shoff	20
3.10.2.12	e_shstrndx	20
3.10.2.13	e_type	20
3.10.2.14	e_version	20
3.11	Elf32_Phdr Struct Reference	20
3.11.1	Detailed Description	20
3.11.2	Field Documentation	20
3.11.2.1	p_align	20
3.11.2.2	p_filesz	21
3.11.2.3	p_flags	21
3.11.2.4	p_memsz	21

3.11.2.5	p_offset	21
3.11.2.6	p_paddr	21
3.11.2.7	p_type	21
3.11.2.8	p_vaddr	21
3.12	Elf32_Shdr Struct Reference	21
3.12.1	Detailed Description	22
3.12.2	Field Documentation	22
3.12.2.1	sh_addr	22
3.12.2.2	sh_addralign	22
3.12.2.3	sh_entsize	22
3.12.2.4	sh_flags	22
3.12.2.5	sh_info	22
3.12.2.6	sh_link	22
3.12.2.7	sh_name	22
3.12.2.8	sh_offset	23
3.12.2.9	sh_size	23
3.12.2.10	sh_type	23
3.13	Elf32_Sym Struct Reference	23
3.13.1	Detailed Description	23
3.13.2	Field Documentation	23
3.13.2.1	st_info	23
3.13.2.2	st_name	24
3.13.2.3	st_other	24
3.13.2.4	st_shndx	24
3.13.2.5	st_size	24
3.13.2.6	st_value	24
3.14	elf_info_t Struct Reference	24
3.14.1	Detailed Description	25
3.14.2	Field Documentation	25
3.14.2.1	addr_space	25
3.14.2.2	at_phdr	26
3.14.2.3	at_phent	26
3.14.2.4	at_phnum	26
3.14.2.5	entry	26
3.14.2.6	stack_addr	26
3.15	elf_symbol_t Struct Reference	26
3.15.1	Detailed Description	26

3.15.2	Field Documentation	27
3.15.2.1	addr	27
3.15.2.2	name	27
3.16	ipc_t Struct Reference	27
3.16.1	Detailed Description	27
3.16.2	Field Documentation	28
3.16.2.1	header	28
3.16.2.2	recv_list	28
3.16.2.3	send_list	28
3.17	jinue_list_t Struct Reference	28
3.17.1	Detailed Description	28
3.17.2	Field Documentation	29
3.17.2.1	head	29
3.17.2.2	tail	29
3.18	jinue_mem_entry_t Struct Reference	29
3.18.1	Detailed Description	29
3.18.2	Field Documentation	29
3.18.2.1	addr	29
3.18.2.2	size	29
3.18.2.3	type	29
3.19	jinue_mem_map_t Struct Reference	30
3.19.1	Detailed Description	30
3.19.2	Field Documentation	30
3.19.2.1	entry	30
3.19.2.2	num_entries	30
3.20	jinue_message_t Struct Reference	31
3.20.1	Detailed Description	31
3.20.2	Field Documentation	31
3.20.2.1	buffer_size	31
3.20.2.2	cookie	31
3.20.2.3	data_size	31
3.20.2.4	desc_n	31
3.20.2.5	function	31
3.21	jinue_node_t Struct Reference	31
3.21.1	Detailed Description	32
3.21.2	Field Documentation	32
3.21.2.1	next	32

3.22	jinue_reply_t Struct Reference	32
3.22.1	Detailed Description	32
3.22.2	Field Documentation	32
3.22.2.1	data_size	32
3.22.2.2	desc_n	33
3.23	jinue_syscall_args_t Struct Reference	33
3.23.1	Detailed Description	33
3.23.2	Field Documentation	33
3.23.2.1	arg0	33
3.23.2.2	arg1	33
3.23.2.3	arg2	33
3.23.2.4	arg3	33
3.24	kernel_context_t Struct Reference	34
3.24.1	Detailed Description	34
3.24.2	Field Documentation	34
3.24.2.1	ebp	34
3.24.2.2	ebx	34
3.24.2.3	edi	34
3.24.2.4	eip	34
3.24.2.5	esi	34
3.25	message_info_t Struct Reference	35
3.25.1	Detailed Description	35
3.25.2	Field Documentation	35
3.25.2.1	buffer_size	35
3.25.2.2	cookie	35
3.25.2.3	data_size	35
3.25.2.4	desc_n	35
3.25.2.5	function	35
3.25.2.6	total_size	36
3.26	object_header_t Struct Reference	36
3.26.1	Detailed Description	36
3.26.2	Field Documentation	36
3.26.2.1	flags	36
3.26.2.2	ref_count	36
3.26.2.3	type	36
3.27	object_ref_t Struct Reference	37
3.27.1	Detailed Description	37



3.27.2	Field Documentation . . . . .	37
3.27.2.1	cookie . . . . .	37
3.27.2.2	flags . . . . .	37
3.27.2.3	object . . . . .	37
3.28	pdpt_t Struct Reference . . . . .	38
3.28.1	Detailed Description . . . . .	38
3.28.2	Field Documentation . . . . .	38
3.28.2.1	pd . . . . .	38
3.29	pfalloc_cache_t Struct Reference . . . . .	38
3.29.1	Detailed Description . . . . .	39
3.29.2	Field Documentation . . . . .	39
3.29.2.1	count . . . . .	39
3.29.2.2	ptr . . . . .	39
3.30	process_t Struct Reference . . . . .	39
3.30.1	Detailed Description . . . . .	40
3.30.2	Field Documentation . . . . .	40
3.30.2.1	addr_space . . . . .	40
3.30.2.2	descriptors . . . . .	40
3.30.2.3	header . . . . .	40
3.31	pseudo_descriptor_t Struct Reference . . . . .	40
3.31.1	Detailed Description . . . . .	40
3.31.2	Field Documentation . . . . .	41
3.31.2.1	addr . . . . .	41
3.31.2.2	limit . . . . .	41
3.31.2.3	padding . . . . .	41
3.32	pte_t Struct Reference . . . . .	41
3.32.1	Detailed Description . . . . .	41
3.32.2	Field Documentation . . . . .	41
3.32.2.1	entry . . . . .	41
3.32.2.2	entry . . . . .	41
3.33	slab_bufctl_t Struct Reference . . . . .	42
3.33.1	Detailed Description . . . . .	42
3.33.2	Field Documentation . . . . .	42
3.33.2.1	next . . . . .	42
3.34	slab_cache_t Struct Reference . . . . .	42
3.34.1	Detailed Description . . . . .	43
3.34.2	Field Documentation . . . . .	43

3.34.2.1	alignment . . . . .	43
3.34.2.2	alloc_size . . . . .	43
3.34.2.3	bufctl_offset . . . . .	43
3.34.2.4	ctor . . . . .	43
3.34.2.5	dtor . . . . .	44
3.34.2.6	empty_count . . . . .	44
3.34.2.7	flags . . . . .	44
3.34.2.8	max_colour . . . . .	44
3.34.2.9	name . . . . .	44
3.34.2.10	next_colour . . . . .	44
3.34.2.11	obj_size . . . . .	44
3.34.2.12	slabs_empty . . . . .	44
3.34.2.13	slabs_full . . . . .	44
3.34.2.14	slabs_partial . . . . .	45
3.34.2.15	working_set . . . . .	45
3.35	slab_t Struct Reference . . . . .	45
3.35.1	Detailed Description . . . . .	46
3.35.2	Field Documentation . . . . .	46
3.35.2.1	cache . . . . .	46
3.35.2.2	colour . . . . .	46
3.35.2.3	free_list . . . . .	46
3.35.2.4	next . . . . .	46
3.35.2.5	obj_count . . . . .	46
3.35.2.6	prev . . . . .	46
3.36	thread_context_t Struct Reference . . . . .	46
3.36.1	Detailed Description . . . . .	47
3.36.2	Field Documentation . . . . .	47
3.36.2.1	local_storage_addr . . . . .	47
3.36.2.2	local_storage_size . . . . .	47
3.36.2.3	saved_stack_pointer . . . . .	47
3.37	thread_t Struct Reference . . . . .	47
3.37.1	Detailed Description . . . . .	48
3.37.2	Field Documentation . . . . .	48
3.37.2.1	header . . . . .	48
3.37.2.2	message_args . . . . .	48
3.37.2.3	message_buffer . . . . .	48
3.37.2.4	message_info . . . . .	48

3.37.2.5	process . . . . .	48
3.37.2.6	sender . . . . .	48
3.37.2.7	thread_ctx . . . . .	49
3.37.2.8	thread_list . . . . .	49
3.38	trapframe_t Struct Reference . . . . .	49
3.38.1	Detailed Description . . . . .	49
3.38.2	Field Documentation . . . . .	50
3.38.2.1	cs . . . . .	50
3.38.2.2	ds . . . . .	50
3.38.2.3	eax . . . . .	50
3.38.2.4	ebp . . . . .	50
3.38.2.5	ebx . . . . .	50
3.38.2.6	ecx . . . . .	50
3.38.2.7	edi . . . . .	50
3.38.2.8	edx . . . . .	50
3.38.2.9	eflags . . . . .	50
3.38.2.10	eip . . . . .	50
3.38.2.11	errcode . . . . .	51
3.38.2.12	es . . . . .	51
3.38.2.13	esi . . . . .	51
3.38.2.14	esp . . . . .	51
3.38.2.15	fs . . . . .	51
3.38.2.16	gs . . . . .	51
3.38.2.17	ivt . . . . .	51
3.38.2.18	ss . . . . .	51
3.39	tss_t Struct Reference . . . . .	52
3.39.1	Detailed Description . . . . .	52
3.39.2	Field Documentation . . . . .	52
3.39.2.1	cr3 . . . . .	52
3.39.2.2	cs . . . . .	52
3.39.2.3	debug . . . . .	53
3.39.2.4	ds . . . . .	53
3.39.2.5	eax . . . . .	53
3.39.2.6	ebp . . . . .	53
3.39.2.7	ebx . . . . .	53
3.39.2.8	ecx . . . . .	53
3.39.2.9	edi . . . . .	53

3.39.2.10	edx . . . . .	53
3.39.2.11	eflags . . . . .	53
3.39.2.12	eip . . . . .	53
3.39.2.13	es . . . . .	53
3.39.2.14	esi . . . . .	54
3.39.2.15	esp . . . . .	54
3.39.2.16	esp0 . . . . .	54
3.39.2.17	esp1 . . . . .	54
3.39.2.18	esp2 . . . . .	54
3.39.2.19	fs . . . . .	54
3.39.2.20	gs . . . . .	54
3.39.2.21	iomap . . . . .	54
3.39.2.22	ldt . . . . .	54
3.39.2.23	prev . . . . .	54
3.39.2.24	ss . . . . .	55
3.39.2.25	ss0 . . . . .	55
3.39.2.26	ss1 . . . . .	55
3.39.2.27	ss2 . . . . .	55
3.40	vmalloc_block_t Struct Reference . . . . .	55
3.40.1	Detailed Description . . . . .	55
3.40.2	Field Documentation . . . . .	56
3.40.2.1	next . . . . .	56
3.40.2.2	prev . . . . .	56
3.40.2.3	stack_base . . . . .	56
3.40.2.4	stack_ptr . . . . .	56
3.41	vmalloc_t Struct Reference . . . . .	56
3.41.1	Detailed Description . . . . .	57
3.41.2	Field Documentation . . . . .	57
3.41.2.1	base_addr . . . . .	57
3.41.2.2	block_array . . . . .	57
3.41.2.3	block_count . . . . .	57
3.41.2.4	end_addr . . . . .	57
3.41.2.5	free_list . . . . .	57
3.41.2.6	init_limit . . . . .	58
3.41.2.7	start_addr . . . . .	58
3.42	x86_cpuid_regs_t Struct Reference . . . . .	58
3.42.1	Detailed Description . . . . .	58

3.42.2	Field Documentation . . . . .	58
3.42.2.1	eax . . . . .	58
3.42.2.2	ebx . . . . .	58
3.42.2.3	ecx . . . . .	58
3.42.2.4	edx . . . . .	59
<b>4</b>	<b>File Documentation</b>	<b>61</b>
4.1	include/ascii.h File Reference . . . . .	61
4.1.1	Macro Definition Documentation . . . . .	61
4.1.1.1	CHAR_BS . . . . .	61
4.1.1.2	CHAR_CR . . . . .	61
4.1.1.3	CHAR_HT . . . . .	62
4.1.1.4	CHAR_LF . . . . .	62
4.2	include/boot.h File Reference . . . . .	62
4.3	include/hal/asm/boot.h File Reference . . . . .	62
4.3.1	Macro Definition Documentation . . . . .	63
4.3.1.1	BOOT_CMD_LINE_PTR . . . . .	63
4.3.1.2	BOOT_DATA_STRUCT . . . . .	63
4.3.1.3	BOOT_E820_ENTRIES . . . . .	63
4.3.1.4	BOOT_E820_MAP . . . . .	63
4.3.1.5	BOOT_E820_MAP_END . . . . .	63
4.3.1.6	BOOT_E820_MAP_SIZE . . . . .	63
4.3.1.7	BOOT_MAGIC . . . . .	63
4.3.1.8	BOOT_RAMDISK_IMAGE . . . . .	63
4.3.1.9	BOOT_RAMDISK_SIZE . . . . .	64
4.3.1.10	BOOT_SETUP . . . . .	64
4.3.1.11	BOOT_SETUP32_ADDR . . . . .	64
4.3.1.12	BOOT_SETUP32_SIZE . . . . .	64
4.3.1.13	BOOT_SETUP_HEADER . . . . .	64
4.3.1.14	BOOT_SETUP_MAGIC . . . . .	64
4.3.1.15	BOOT_SETUP_SECTS . . . . .	64
4.3.1.16	BOOT_SIGNATURE . . . . .	64
4.3.1.17	BOOT_STACK_HEAP_SIZE . . . . .	64
4.3.1.18	BOOT_SYSIZE . . . . .	64
4.4	include/hal/boot.h File Reference . . . . .	65
4.4.1	Macro Definition Documentation . . . . .	66
4.4.1.1	boot_heap_alloc . . . . .	66

4.4.2	Function Documentation . . . . .	66
4.4.2.1	boot_alloc_init . . . . .	66
4.4.2.2	boot_heap_alloc_size . . . . .	67
4.4.2.3	boot_heap_pop . . . . .	67
4.4.2.4	boot_heap_push . . . . .	68
4.4.2.5	boot_info_check . . . . .	68
4.4.2.6	boot_info_dump . . . . .	69
4.4.2.7	boot_page_alloc . . . . .	70
4.4.2.8	boot_page_alloc_early . . . . .	71
4.4.2.9	boot_page_alloc_image . . . . .	72
4.4.2.10	boot_page_frame_alloc . . . . .	73
4.4.2.11	boot_vmalloc . . . . .	74
4.4.2.12	get_boot_info . . . . .	75
4.5	include/console.h File Reference . . . . .	76
4.6	include/jinue/console.h File Reference . . . . .	76
4.7	include/jinue-common/console.h File Reference . . . . .	76
4.7.1	Macro Definition Documentation . . . . .	77
4.7.1.1	CONSOLE_DEFAULT_COLOR . . . . .	77
4.7.1.2	CONSOLE_SERIAL_BAUD_RATE . . . . .	77
4.7.1.3	CONSOLE_SERIAL_IOPORT . . . . .	77
4.7.2	Function Documentation . . . . .	77
4.7.2.1	console_init . . . . .	77
4.7.2.2	console_print . . . . .	78
4.7.2.3	console_printn . . . . .	78
4.7.2.4	console_putc . . . . .	79
4.8	include/debug.h File Reference . . . . .	79
4.8.1	Function Documentation . . . . .	80
4.8.1.1	dump_call_stack . . . . .	80
4.9	include/elf.h File Reference . . . . .	81
4.10	include/jinue/elf.h File Reference . . . . .	81
4.11	include/jinue-common/elf.h File Reference . . . . .	82
4.11.1	Macro Definition Documentation . . . . .	86
4.11.1.1	AT_BASE . . . . .	86
4.11.1.2	AT_DCACHEBSIZE . . . . .	86
4.11.1.3	AT_ENTRY . . . . .	86
4.11.1.4	AT_EXECFD . . . . .	87
4.11.1.5	AT_FLAGS . . . . .	87

4.11.1.6 AT_HWCAP . . . . .	87
4.11.1.7 AT_HWCAP2 . . . . .	87
4.11.1.8 AT_ICACHEBSIZE . . . . .	87
4.11.1.9 AT_IGNORE . . . . .	87
4.11.1.10 AT_NULL . . . . .	87
4.11.1.11 AT_PAGESZ . . . . .	87
4.11.1.12 AT_PHDR . . . . .	88
4.11.1.13 AT_PHEMT . . . . .	88
4.11.1.14 AT_PHNUM . . . . .	88
4.11.1.15 AT_STACKBASE . . . . .	88
4.11.1.16 AT_SYSINFO_EHDR . . . . .	88
4.11.1.17 AT_UCACHEBSIZE . . . . .	88
4.11.1.18 EI_CLASS . . . . .	88
4.11.1.19 EI_DATA . . . . .	88
4.11.1.20 EI_MAG0 . . . . .	89
4.11.1.21 EI_MAG1 . . . . .	89
4.11.1.22 EI_MAG2 . . . . .	89
4.11.1.23 EI_MAG3 . . . . .	89
4.11.1.24 EI_NIDENT . . . . .	89
4.11.1.25 EI_PAD . . . . .	89
4.11.1.26 EI_VERSION . . . . .	89
4.11.1.27 ELF32_ST_BIND . . . . .	90
4.11.1.28 ELF32_ST_TYPE . . . . .	90
4.11.1.29 ELF_MAGIC0 . . . . .	90
4.11.1.30 ELF_MAGIC1 . . . . .	90
4.11.1.31 ELF_MAGIC2 . . . . .	90
4.11.1.32 ELF_MAGIC3 . . . . .	90
4.11.1.33 ELFCLASS32 . . . . .	90
4.11.1.34 ELFCLASS64 . . . . .	90
4.11.1.35 ELFCLASSNONE . . . . .	91
4.11.1.36 ELFDATA2LSB . . . . .	91
4.11.1.37 ELFDATA2MSB . . . . .	91
4.11.1.38 ELFDATANONE . . . . .	91
4.11.1.39 EM_386 . . . . .	91
4.11.1.40 EM_AARCH64 . . . . .	91
4.11.1.41 EM_ALTERA_NIOS2 . . . . .	91
4.11.1.42 EM_ARM . . . . .	91

4.11.1.43 EM_MICROBLAZE . . . . .	92
4.11.1.44 EM_MIPS . . . . .	92
4.11.1.45 EM_NONE . . . . .	92
4.11.1.46 EM_OPENRISC . . . . .	92
4.11.1.47 EM_SPARC . . . . .	92
4.11.1.48 EM_SPARC32PLUS . . . . .	92
4.11.1.49 EM_X86_64 . . . . .	92
4.11.1.50 ET_CORE . . . . .	92
4.11.1.51 ET_DYN . . . . .	92
4.11.1.52 ET_EXEC . . . . .	93
4.11.1.53 ET_NONE . . . . .	93
4.11.1.54 ET_REL . . . . .	93
4.11.1.55 PF_R . . . . .	93
4.11.1.56 PF_W . . . . .	93
4.11.1.57 PF_X . . . . .	93
4.11.1.58 PT_DYNAMIC . . . . .	93
4.11.1.59 PT_INTERP . . . . .	93
4.11.1.60 PT_LOAD . . . . .	93
4.11.1.61 PT_NOTE . . . . .	94
4.11.1.62 PT_NULL . . . . .	94
4.11.1.63 PT_PHDR . . . . .	94
4.11.1.64 PT_SHLIB . . . . .	94
4.11.1.65 SHT_DYNAMIC . . . . .	94
4.11.1.66 SHT_DYNSYM . . . . .	94
4.11.1.67 SHT_HASH . . . . .	94
4.11.1.68 SHT_NOBITS . . . . .	94
4.11.1.69 SHT_NOTE . . . . .	94
4.11.1.70 SHT_NULL . . . . .	95
4.11.1.71 SHT_PROGBITS . . . . .	95
4.11.1.72 SHT_REL . . . . .	95
4.11.1.73 SHT_RELA . . . . .	95
4.11.1.74 SHT_SHLIB . . . . .	95
4.11.1.75 SHT_STRTAB . . . . .	95
4.11.1.76 SHT_SYMTAB . . . . .	95
4.11.1.77 STB_GLOBAL . . . . .	95
4.11.1.78 STB_LOCAL . . . . .	96
4.11.1.79 STB_WEAK . . . . .	96



4.11.1.80 STN_UNDEF . . . . .	96
4.11.1.81 STT_FILE . . . . .	96
4.11.1.82 STT_FUNCTION . . . . .	96
4.11.1.83 STT_NOTYPE . . . . .	96
4.11.1.84 STT_OBJECT . . . . .	96
4.11.1.85 STT_SECTION . . . . .	96
4.11.2 Typedef Documentation . . . . .	97
4.11.2.1 auxv_t . . . . .	97
4.11.2.2 Elf32_Addr . . . . .	97
4.11.2.3 Elf32_Half . . . . .	97
4.11.2.4 Elf32_Off . . . . .	97
4.11.2.5 Elf32_Sword . . . . .	97
4.11.2.6 Elf32_Word . . . . .	97
4.11.3 Function Documentation . . . . .	97
4.11.3.1 elf_check . . . . .	97
4.11.3.2 elf_load . . . . .	98
4.11.3.3 elf_lookup_symbol . . . . .	100
4.11.3.4 elf_setup_stack . . . . .	101
4.12 include/hal/abi.h File Reference . . . . .	103
4.12.1 Function Documentation . . . . .	104
4.12.1.1 get_caller_fpointer . . . . .	104
4.12.1.2 get_fpointer . . . . .	104
4.12.1.3 get_program_counter . . . . .	104
4.12.1.4 get_ret_addr . . . . .	104
4.13 include/hal/asm/descriptors.h File Reference . . . . .	104
4.13.1 Macro Definition Documentation . . . . .	106
4.13.1.1 GDT_KERNEL_CODE . . . . .	106
4.13.1.2 GDT_KERNEL_DATA . . . . .	106
4.13.1.3 GDT_LENGTH . . . . .	106
4.13.1.4 GDT_NULL . . . . .	106
4.13.1.5 GDT_PER_CPU_DATA . . . . .	106
4.13.1.6 GDT_TSS . . . . .	107
4.13.1.7 GDT_USER_CODE . . . . .	107
4.13.1.8 GDT_USER_DATA . . . . .	107
4.13.1.9 GDT_USER_TLS_DATA . . . . .	107
4.13.1.10 RPL_KERNEL . . . . .	107
4.13.1.11 RPL_USER . . . . .	107

4.13.1.12	SEG_FLAG_16BIT . . . . .	107
4.13.1.13	SEG_FLAG_16BIT_GATE . . . . .	107
4.13.1.14	SEG_FLAG_32BIT . . . . .	108
4.13.1.15	SEG_FLAG_32BIT_GATE . . . . .	108
4.13.1.16	SEG_FLAG_BUSY . . . . .	108
4.13.1.17	SEG_FLAG_IN_BYTES . . . . .	108
4.13.1.18	SEG_FLAG_IN_PAGES . . . . .	108
4.13.1.19	SEG_FLAG_KERNEL . . . . .	108
4.13.1.20	SEG_FLAG_NORMAL . . . . .	108
4.13.1.21	SEG_FLAG_NORMAL_GATE . . . . .	108
4.13.1.22	SEG_FLAG_NOSYSTEM . . . . .	109
4.13.1.23	SEG_FLAG_PRESENT . . . . .	109
4.13.1.24	SEG_FLAG_SYSTEM . . . . .	109
4.13.1.25	SEG_FLAG_TSS . . . . .	109
4.13.1.26	SEG_FLAG_USER . . . . .	109
4.13.1.27	SEG_FLAGS_OFFSET . . . . .	109
4.13.1.28	SEG_SELECTOR . . . . .	109
4.13.1.29	SEG_TYPE_CALL_GATE . . . . .	110
4.13.1.30	SEG_TYPE_CODE . . . . .	110
4.13.1.31	SEG_TYPE_DATA . . . . .	110
4.13.1.32	SEG_TYPE_INTERRUPT_GATE . . . . .	110
4.13.1.33	SEG_TYPE_READ_ONLY . . . . .	110
4.13.1.34	SEG_TYPE_TASK_GATE . . . . .	110
4.13.1.35	SEG_TYPE_TRAP_GATE . . . . .	110
4.13.1.36	SEG_TYPE_TSS . . . . .	110
4.13.1.37	TSS_LIMIT . . . . .	111
4.14	include/hal/descriptors.h File Reference . . . . .	111
4.14.1	Macro Definition Documentation . . . . .	111
4.14.1.1	GATE_DESCRIPTOR . . . . .	112
4.14.1.2	PACK_DESCRIPTOR . . . . .	112
4.14.1.3	SEG_DESCRIPTOR . . . . .	112
4.15	include/hal/asm/irq.h File Reference . . . . .	112
4.15.1	Macro Definition Documentation . . . . .	113
4.15.1.1	EXCEPTION_ALIGNMENT . . . . .	113
4.15.1.2	EXCEPTION_BOUND . . . . .	114
4.15.1.3	EXCEPTION_BREAK . . . . .	114
4.15.1.4	EXCEPTION_DIV_ZERO . . . . .	114

4.15.1.5	EXCEPTION_DOUBLE_FAULT . . . . .	114
4.15.1.6	EXCEPTION_GENERAL_PROTECTION . . . . .	114
4.15.1.7	EXCEPTION_INVALID_OP . . . . .	114
4.15.1.8	EXCEPTION_INVALID_TSS . . . . .	114
4.15.1.9	EXCEPTION_MACHINE_CHECK . . . . .	114
4.15.1.10	EXCEPTION_MATH . . . . .	114
4.15.1.11	EXCEPTION_NMI . . . . .	115
4.15.1.12	EXCEPTION_NO_COPROC . . . . .	115
4.15.1.13	EXCEPTION_OVERFLOW . . . . .	115
4.15.1.14	EXCEPTION_PAGE_FAULT . . . . .	115
4.15.1.15	EXCEPTION_SEGMENT_NOT_PRESENT . . . . .	115
4.15.1.16	EXCEPTION_SIMD . . . . .	115
4.15.1.17	EXCEPTION_STACK_SEGMENT . . . . .	115
4.15.1.18	HAS_ERRCODE . . . . .	115
4.15.1.19	IDT_LAST_EXCEPTION . . . . .	115
4.15.1.20	IDT_PIC8259_BASE . . . . .	116
4.15.1.21	IDT_VECTOR_COUNT . . . . .	116
4.16	include/hal/asm/mem.h File Reference . . . . .	116
4.16.1	Macro Definition Documentation . . . . .	116
4.16.1.1	MEM_ZONE_DMA16_END . . . . .	116
4.16.1.2	MEM_ZONE_DMA16_START . . . . .	117
4.16.1.3	MEM_ZONE_MEM32_END . . . . .	117
4.16.1.4	MEM_ZONE_MEM32_START . . . . .	117
4.17	include/hal/mem.h File Reference . . . . .	117
4.17.1	Function Documentation . . . . .	118
4.17.1.1	mem_check_memory . . . . .	118
4.18	include/hal/asm/pic8259.h File Reference . . . . .	120
4.18.1	Macro Definition Documentation . . . . .	121
4.18.1.1	PIC8259_CASCADE_INPUT . . . . .	121
4.18.1.2	PIC8259_EOI . . . . .	121
4.18.1.3	PIC8259_ICW1_1 . . . . .	121
4.18.1.4	PIC8259_ICW1_IC4 . . . . .	121
4.18.1.5	PIC8259_ICW1_LTIM . . . . .	122
4.18.1.6	PIC8259_ICW1_SNGL . . . . .	122
4.18.1.7	PIC8259_ICW4_AEOI . . . . .	122
4.18.1.8	PIC8259_ICW4_UPM . . . . .	122
4.18.1.9	PIC8259_IRQ_COUNT . . . . .	122

4.18.1.10 PIC8259_MASTER_BASE . . . . .	122
4.18.1.11 PIC8259_SLAVE_BASE . . . . .	122
4.19 include/hal/pic8259.h File Reference . . . . .	122
4.19.1 Function Documentation . . . . .	123
4.19.1.1 pic8259_eoi . . . . .	123
4.19.1.2 pic8259_init . . . . .	124
4.19.1.3 pic8259_mask_irq . . . . .	125
4.19.1.4 pic8259_unmask_irq . . . . .	125
4.20 include/hal/asm/serial.h File Reference . . . . .	127
4.20.1 Macro Definition Documentation . . . . .	127
4.20.1.1 SERIAL_COM1_IOPORT . . . . .	127
4.20.1.2 SERIAL_COM2_IOPORT . . . . .	127
4.20.1.3 SERIAL_COM3_IOPORT . . . . .	127
4.20.1.4 SERIAL_COM4_IOPORT . . . . .	128
4.20.1.5 SERIAL_REG_DATA_BUFFER . . . . .	128
4.20.1.6 SERIAL_REG_DIVISOR_HIGH . . . . .	128
4.20.1.7 SERIAL_REG_DIVISOR_LOW . . . . .	128
4.20.1.8 SERIAL_REG_FIFO_CTRL . . . . .	128
4.20.1.9 SERIAL_REG_INTR_ENABLE . . . . .	128
4.20.1.10 SERIAL_REG_INTR_ID . . . . .	128
4.20.1.11 SERIAL_REG_LINE_CTRL . . . . .	128
4.20.1.12 SERIAL_REG_LINE_STATUS . . . . .	128
4.20.1.13 SERIAL_REG_MODEM_CTRL . . . . .	129
4.20.1.14 SERIAL_REG_MODEM_STATUS . . . . .	129
4.20.1.15 SERIAL_REG_SCRATCH . . . . .	129
4.21 include/hal/serial.h File Reference . . . . .	129
4.21.1 Function Documentation . . . . .	130
4.21.1.1 serial_init . . . . .	130
4.21.1.2 serial_printn . . . . .	130
4.21.1.3 serial_putc . . . . .	131
4.22 include/hal/asm/thread.h File Reference . . . . .	131
4.22.1 Macro Definition Documentation . . . . .	132
4.22.1.1 THREAD_CONTEXT_MASK . . . . .	132
4.22.1.2 THREAD_CONTEXT_SIZE . . . . .	132
4.23 include/hal/thread.h File Reference . . . . .	133
4.23.1 Function Documentation . . . . .	133
4.23.1.1 thread_context_switch . . . . .	133

4.23.1.2	thread_page_init . . . . .	134
4.24	include/thread.h File Reference . . . . .	135
4.24.1	Function Documentation . . . . .	136
4.24.1.1	thread_create . . . . .	136
4.24.1.2	thread_create_boot . . . . .	136
4.24.1.3	thread_destroy . . . . .	137
4.24.1.4	thread_ready . . . . .	137
4.24.1.5	thread_switch . . . . .	137
4.24.1.6	thread_yield_from . . . . .	138
4.25	include/hal/asm/vm.h File Reference . . . . .	139
4.25.1	Macro Definition Documentation . . . . .	140
4.25.1.1	VM_FLAG_ACCESSED . . . . .	140
4.25.1.2	VM_FLAG_DIRTY . . . . .	140
4.25.1.3	VM_FLAG_KERNEL . . . . .	140
4.25.1.4	VM_FLAG_PRESENT . . . . .	140
4.25.1.5	VM_FLAG_READ_ONLY . . . . .	140
4.25.1.6	VM_FLAG_READ_WRITE . . . . .	140
4.25.1.7	VM_FLAG_USER . . . . .	141
4.26	include/hal/vm.h File Reference . . . . .	141
4.26.1	Macro Definition Documentation . . . . .	142
4.26.1.1	ADDR_4GB . . . . .	142
4.26.1.2	EARLY_PHYS_TO_VIRT . . . . .	142
4.26.1.3	EARLY_PTR_TO_PHYS_ADDR . . . . .	142
4.26.1.4	EARLY_VIRT_TO_PHYS . . . . .	142
4.26.2	Function Documentation . . . . .	143
4.26.2.1	vm_boot_init . . . . .	143
4.26.2.2	vm_boot_postinit . . . . .	144
4.26.2.3	vm_change_flags . . . . .	145
4.26.2.4	vm_create_addr_space . . . . .	145
4.26.2.5	vm_create_initial_addr_space . . . . .	146
4.26.2.6	vm_destroy_addr_space . . . . .	147
4.26.2.7	vm_lookup_kernel_paddr . . . . .	147
4.26.2.8	vm_map_early . . . . .	148
4.26.2.9	vm_map_kernel . . . . .	148
4.26.2.10	vm_map_user . . . . .	148
4.26.2.11	vm_switch_addr_space . . . . .	148
4.26.2.12	vm_unmap_kernel . . . . .	149

4.26.2.13	vm_unmap_user . . . . .	149
4.26.3	Variable Documentation . . . . .	149
4.26.3.1	initial_addr_space . . . . .	149
4.27	include/jinue/vm.h File Reference . . . . .	149
4.28	include/jinue-common/asm/vm.h File Reference . . . . .	150
4.28.1	Macro Definition Documentation . . . . .	151
4.28.1.1	KERNEL_EARLY_LIMIT . . . . .	151
4.28.1.2	KERNEL_IMAGE_END . . . . .	151
4.28.1.3	KERNEL_PREALLOC_LIMIT . . . . .	151
4.28.1.4	KLIMIT . . . . .	151
4.28.1.5	PAGE_BITS . . . . .	152
4.28.1.6	PAGE_MASK . . . . .	152
4.28.1.7	PAGE_SIZE . . . . .	152
4.28.1.8	STACK_BASE . . . . .	152
4.28.1.9	STACK_SIZE . . . . .	152
4.28.1.10	STACK_START . . . . .	152
4.29	include/jinue-common/vm.h File Reference . . . . .	152
4.29.1	Macro Definition Documentation . . . . .	153
4.29.1.1	page_address_of . . . . .	153
4.29.1.2	page_number_of . . . . .	153
4.29.1.3	page_offset_of . . . . .	154
4.30	include/hal/asm/x86.h File Reference . . . . .	154
4.30.1	Macro Definition Documentation . . . . .	155
4.30.1.1	X86_CR0_PG . . . . .	155
4.30.1.2	X86_CR0_WP . . . . .	155
4.30.1.3	X86_CR4_PAE . . . . .	155
4.30.1.4	X86_CR4_PGE . . . . .	155
4.30.1.5	X86_CR4_PSE . . . . .	155
4.30.1.6	X86_PDE_PAGE_SIZE . . . . .	156
4.30.1.7	X86_PTE_ACCESSED . . . . .	156
4.30.1.8	X86_PTE_CACHE_DISABLE . . . . .	156
4.30.1.9	X86_PTE_DIRTY . . . . .	156
4.30.1.10	X86_PTE_GLOBAL . . . . .	156
4.30.1.11	X86_PTE_NX . . . . .	156
4.30.1.12	X86_PTE_PRESENT . . . . .	156
4.30.1.13	X86_PTE_READ_WRITE . . . . .	156
4.30.1.14	X86_PTE_USER . . . . .	156

4.30.1.15 X86_PTE_WRITE_THROUGH . . . . .	157
4.31 include/hal/x86.h File Reference . . . . .	157
4.31.1 Typedef Documentation . . . . .	158
4.31.1.1 msr_addr_t . . . . .	158
4.31.2 Function Documentation . . . . .	158
4.31.2.1 cli . . . . .	158
4.31.2.2 cpuid . . . . .	158
4.31.2.3 enable_pae . . . . .	158
4.31.2.4 get_cr0 . . . . .	158
4.31.2.5 get_cr2 . . . . .	158
4.31.2.6 get_cr3 . . . . .	159
4.31.2.7 get_cr4 . . . . .	159
4.31.2.8 get_eflags . . . . .	159
4.31.2.9 get_esp . . . . .	159
4.31.2.10 get_gs_ptr . . . . .	159
4.31.2.11 invalidate_tlb . . . . .	159
4.31.2.12 lgdt . . . . .	159
4.31.2.13 lidt . . . . .	159
4.31.2.14 ltr . . . . .	159
4.31.2.15 rdmsr . . . . .	159
4.31.2.16 rdtsc . . . . .	159
4.31.2.17 set_cr0 . . . . .	159
4.31.2.18 set_cr3 . . . . .	159
4.31.2.19 set_cr4 . . . . .	159
4.31.2.20 set_cs . . . . .	159
4.31.2.21 set_ds . . . . .	159
4.31.2.22 set_eflags . . . . .	159
4.31.2.23 set_es . . . . .	159
4.31.2.24 set_fs . . . . .	159
4.31.2.25 set_gs . . . . .	159
4.31.2.26 set_ss . . . . .	159
4.31.2.27 sti . . . . .	160
4.31.2.28 wrmsr . . . . .	160
4.32 include/hal/cpu.h File Reference . . . . .	160
4.32.1 Macro Definition Documentation . . . . .	161
4.32.1.1 CPU_EFLAGS_ID . . . . .	161
4.32.1.2 CPU_FEATURE_CPUID . . . . .	161

4.32.1.3	CPU_FEATURE_LOCAL_APIC . . . . .	161
4.32.1.4	CPU_FEATURE_PAE . . . . .	162
4.32.1.5	CPU_FEATURE_SYSCALL . . . . .	162
4.32.1.6	CPU_FEATURE_SYSENTER . . . . .	162
4.32.1.7	CPU_VENDOR_AMD . . . . .	162
4.32.1.8	CPU_VENDOR_AMD_DW0 . . . . .	162
4.32.1.9	CPU_VENDOR_AMD_DW1 . . . . .	162
4.32.1.10	CPU_VENDOR_AMD_DW2 . . . . .	162
4.32.1.11	CPU_VENDOR_GENERIC . . . . .	162
4.32.1.12	CPU_VENDOR_INTEL . . . . .	162
4.32.1.13	CPU_VENDOR_INTEL_DW0 . . . . .	163
4.32.1.14	CPU_VENDOR_INTEL_DW1 . . . . .	163
4.32.1.15	CPU_VENDOR_INTEL_DW2 . . . . .	163
4.32.1.16	CPUID_EXT_FEATURE_SYSCALL . . . . .	163
4.32.1.17	CPUID_FEATURE_APIC . . . . .	163
4.32.1.18	CPUID_FEATURE_CLFLUSH . . . . .	163
4.32.1.19	CPUID_FEATURE_FPU . . . . .	163
4.32.1.20	CPUID_FEATURE_HTT . . . . .	163
4.32.1.21	CPUID_FEATURE_PAE . . . . .	163
4.32.1.22	CPUID_FEATURE_SEP . . . . .	164
4.32.1.23	MSR_EFER . . . . .	164
4.32.1.24	MSR_FLAG_STAR_SCE . . . . .	164
4.32.1.25	MSR_IA32_SYSENTER_CS . . . . .	164
4.32.1.26	MSR_IA32_SYSENTER_EIP . . . . .	164
4.32.1.27	MSR_IA32_SYSENTER_ESP . . . . .	164
4.32.1.28	MSR_STAR . . . . .	164
4.32.2	Function Documentation . . . . .	164
4.32.2.1	cpu_detect_features . . . . .	164
4.32.2.2	cpu_init_data . . . . .	166
4.32.3	Variable Documentation . . . . .	167
4.32.3.1	cpu_info . . . . .	168
4.33	include/hal/cpu_data.h File Reference . . . . .	168
4.33.1	Macro Definition Documentation . . . . .	168
4.33.1.1	CPU_DATA_ALIGNMENT . . . . .	168
4.34	include/hal/hal.h File Reference . . . . .	169
4.34.1	Function Documentation . . . . .	169
4.34.1.1	hal_init . . . . .	169



4.35	include/hal/interrupt.h File Reference . . . . .	171
4.35.1	Function Documentation . . . . .	172
4.35.1.1	dispatch_interrupt . . . . .	172
4.35.2	Variable Documentation . . . . .	173
4.35.2.1	idt . . . . .	173
4.36	include/hal/io.h File Reference . . . . .	173
4.36.1	Function Documentation . . . . .	174
4.36.1.1	inb . . . . .	174
4.36.1.2	inl . . . . .	174
4.36.1.3	inw . . . . .	174
4.36.1.4	iodelay . . . . .	174
4.36.1.5	outb . . . . .	174
4.36.1.6	outl . . . . .	174
4.36.1.7	outw . . . . .	174
4.37	include/hal/startup.h File Reference . . . . .	175
4.37.1	Function Documentation . . . . .	175
4.37.1.1	halt . . . . .	175
4.38	include/hal/trap.h File Reference . . . . .	175
4.38.1	Function Documentation . . . . .	176
4.38.1.1	fast_amd_entry . . . . .	176
4.38.1.2	fast_intel_entry . . . . .	176
4.38.1.3	return_from_interrupt . . . . .	176
4.38.2	Variable Documentation . . . . .	176
4.38.2.1	syscall_method . . . . .	176
4.39	include/hal/types.h File Reference . . . . .	176
4.39.1	Macro Definition Documentation . . . . .	177
4.39.1.1	msg_arg0 . . . . .	177
4.39.1.2	msg_arg1 . . . . .	178
4.39.1.3	msg_arg2 . . . . .	178
4.39.1.4	msg_arg3 . . . . .	178
4.39.1.5	PFNULL . . . . .	178
4.39.2	Typedef Documentation . . . . .	178
4.39.2.1	addr_t . . . . .	178
4.39.2.2	cpu_data_t . . . . .	178
4.39.2.3	kern_paddr_t . . . . .	178
4.39.2.4	pdpt_t . . . . .	178
4.39.2.5	pte_t . . . . .	178

4.39.2.6	seg_descriptor_t . . . . .	179
4.39.2.7	seg_selector_t . . . . .	179
4.39.2.8	user_paddr_t . . . . .	179
4.40	include/jinue/types.h File Reference . . . . .	179
4.41	include/jinue-common/asm/types.h File Reference . . . . .	179
4.41.1	Macro Definition Documentation . . . . .	180
4.41.1.1	GB . . . . .	180
4.41.1.2	KB . . . . .	180
4.41.1.3	MB . . . . .	180
4.42	include/jinue-common/types.h File Reference . . . . .	180
4.43	include/types.h File Reference . . . . .	181
4.43.1	Macro Definition Documentation . . . . .	181
4.43.1.1	PROCESS_MAX_DESCRIPTORs . . . . .	182
4.43.2	Typedef Documentation . . . . .	182
4.43.2.1	thread_t . . . . .	182
4.44	include/hal/vga.h File Reference . . . . .	182
4.44.1	Macro Definition Documentation . . . . .	183
4.44.1.1	VGA_COL . . . . .	183
4.44.1.2	VGA_COLOR_BLACK . . . . .	183
4.44.1.3	VGA_COLOR_BLUE . . . . .	183
4.44.1.4	VGA_COLOR_BRIGHTBLUE . . . . .	183
4.44.1.5	VGA_COLOR_BRIGHTCYAN . . . . .	183
4.44.1.6	VGA_COLOR_BRIGHTGREEN . . . . .	184
4.44.1.7	VGA_COLOR_BRIGHTMAGENTA . . . . .	184
4.44.1.8	VGA_COLOR_BRIGHTRED . . . . .	184
4.44.1.9	VGA_COLOR_BRIGHTWHITE . . . . .	184
4.44.1.10	VGA_COLOR_BROWN . . . . .	184
4.44.1.11	VGA_COLOR_CYAN . . . . .	184
4.44.1.12	VGA_COLOR_ERASE . . . . .	184
4.44.1.13	VGA_COLOR_GRAY . . . . .	184
4.44.1.14	VGA_COLOR_GREEN . . . . .	184
4.44.1.15	VGA_COLOR_MAGENTA . . . . .	184
4.44.1.16	VGA_COLOR_RED . . . . .	184
4.44.1.17	VGA_COLOR_WHITE . . . . .	185
4.44.1.18	VGA_COLOR_YELLOW . . . . .	185
4.44.1.19	VGA_CRTC_ADDR . . . . .	185
4.44.1.20	VGA_CRTC_DATA . . . . .	185

4.44.1.21	VGA_FB_FLAG_ACTIVE . . . . .	185
4.44.1.22	VGA_LINE . . . . .	185
4.44.1.23	VGA_LINES . . . . .	185
4.44.1.24	VGA_MISC_OUT_RD . . . . .	185
4.44.1.25	VGA_MISC_OUT_WR . . . . .	185
4.44.1.26	VGA_TAB_WIDTH . . . . .	185
4.44.1.27	VGA_TEXT_VID_BASE . . . . .	186
4.44.1.28	VGA_TEXT_VID_SIZE . . . . .	186
4.44.1.29	VGA_TEXT_VID_TOP . . . . .	186
4.44.1.30	VGA_WIDTH . . . . .	186
4.44.2	Typedef Documentation . . . . .	186
4.44.2.1	vga_pos_t . . . . .	186
4.44.3	Function Documentation . . . . .	186
4.44.3.1	vga_clear . . . . .	186
4.44.3.2	vga_get_cursor_pos . . . . .	186
4.44.3.3	vga_init . . . . .	187
4.44.3.4	vga_print . . . . .	188
4.44.3.5	vga_printrn . . . . .	188
4.44.3.6	vga_putc . . . . .	189
4.44.3.7	vga_scroll . . . . .	189
4.44.3.8	vga_set_base_addr . . . . .	189
4.44.3.9	vga_set_cursor_pos . . . . .	190
4.45	include/hal/vm_pae.h File Reference . . . . .	190
4.45.1	Function Documentation . . . . .	191
4.45.1.1	vm_pae_boot_init . . . . .	191
4.45.1.2	vm_pae_clear_pte . . . . .	192
4.45.1.3	vm_pae_copy_pte . . . . .	192
4.45.1.4	vm_pae_create_addr_space . . . . .	192
4.45.1.5	vm_pae_create_initial_addr_space . . . . .	193
4.45.1.6	vm_pae_create_pdpt_cache . . . . .	194
4.45.1.7	vm_pae_destroy_addr_space . . . . .	195
4.45.1.8	vm_pae_get_pte_flags . . . . .	196
4.45.1.9	vm_pae_get_pte_paddr . . . . .	196
4.45.1.10	vm_pae_get_pte_with_offset . . . . .	196
4.45.1.11	vm_pae_lookup_page_directory . . . . .	196
4.45.1.12	vm_pae_page_directory_offset_of . . . . .	198
4.45.1.13	vm_pae_page_table_offset_of . . . . .	198

4.45.1.14	vm_pae_set_pte . . . . .	198
4.45.1.15	vm_pae_set_pte_flags . . . . .	199
4.45.1.16	vm_pae_unmap_low_alias . . . . .	199
4.46	include/hal/vm_private.h File Reference . . . . .	199
4.46.1	Macro Definition Documentation . . . . .	201
4.46.1.1	PAGE_DIRECTORY_OFFSET_OF . . . . .	201
4.46.1.2	PAGE_TABLE_ENTRIES . . . . .	201
4.46.1.3	PAGE_TABLE_MASK . . . . .	201
4.46.1.4	PAGE_TABLE_OFFSET_OF . . . . .	201
4.46.2	Function Documentation . . . . .	201
4.46.2.1	vm_clone_page_directory . . . . .	201
4.46.2.2	vm_destroy_page_directory . . . . .	202
4.46.2.3	vm_init_initial_page_directory . . . . .	203
4.46.3	Variable Documentation . . . . .	204
4.46.3.1	global_page_tables . . . . .	204
4.46.3.2	page_table_entries . . . . .	204
4.47	include/hal/vm_x86.h File Reference . . . . .	204
4.47.1	Function Documentation . . . . .	205
4.47.1.1	vm_x86_boot_init . . . . .	205
4.47.1.2	vm_x86_clear_pte . . . . .	206
4.47.1.3	vm_x86_copy_pte . . . . .	206
4.47.1.4	vm_x86_create_addr_space . . . . .	206
4.47.1.5	vm_x86_create_initial_addr_space . . . . .	207
4.47.1.6	vm_x86_destroy_addr_space . . . . .	207
4.47.1.7	vm_x86_get_pte_flags . . . . .	208
4.47.1.8	vm_x86_get_pte_paddr . . . . .	208
4.47.1.9	vm_x86_get_pte_with_offset . . . . .	208
4.47.1.10	vm_x86_lookup_page_directory . . . . .	208
4.47.1.11	vm_x86_page_directory_offset_of . . . . .	209
4.47.1.12	vm_x86_page_table_offset_of . . . . .	209
4.47.1.13	vm_x86_set_pte . . . . .	209
4.47.1.14	vm_x86_set_pte_flags . . . . .	210
4.48	include/ipc.h File Reference . . . . .	210
4.49	include/jinue/ipc.h File Reference . . . . .	210
4.49.1	Function Documentation . . . . .	211
4.49.1.1	jinue_create_ipc . . . . .	211
4.49.1.2	jinue_receive . . . . .	211

4.49.1.3	jinue_reply . . . . .	211
4.49.1.4	jinue_send . . . . .	211
4.50	include/jinue-common/asm/ipc.h File Reference . . . . .	211
4.50.1	Macro Definition Documentation . . . . .	211
4.50.1.1	JINUE_ARGS_PACK_BUFFER_SIZE . . . . .	211
4.50.1.2	JINUE_ARGS_PACK_DATA_SIZE . . . . .	212
4.50.1.3	JINUE_ARGS_PACK_N_DESC . . . . .	212
4.50.1.4	JINUE_SEND_BUFFER_SIZE_OFFSET . . . . .	212
4.50.1.5	JINUE_SEND_DATA_SIZE_OFFSET . . . . .	212
4.50.1.6	JINUE_SEND_MAX_N_DESC . . . . .	212
4.50.1.7	JINUE_SEND_MAX_SIZE . . . . .	212
4.50.1.8	JINUE_SEND_N_DESC_BITS . . . . .	212
4.50.1.9	JINUE_SEND_N_DESC_MASK . . . . .	212
4.50.1.10	JINUE_SEND_N_DESC_OFFSET . . . . .	213
4.50.1.11	JINUE_SEND_SIZE_BITS . . . . .	213
4.50.1.12	JINUE_SEND_SIZE_MASK . . . . .	213
4.51	include/jinue-common/ipc.h File Reference . . . . .	213
4.51.1	Macro Definition Documentation . . . . .	214
4.51.1.1	IPC_FLAG_NONE . . . . .	214
4.51.1.2	IPC_FLAG_SYSTEM . . . . .	214
4.51.1.3	JINUE_IPC_NONE . . . . .	214
4.51.1.4	JINUE_IPC_PROC . . . . .	214
4.51.1.5	JINUE_IPC_SYSTEM . . . . .	214
4.51.2	Typedef Documentation . . . . .	215
4.51.2.1	jinue_ipc_descriptor_t . . . . .	215
4.51.3	Function Documentation . . . . .	215
4.51.3.1	ipc_boot_init . . . . .	215
4.51.3.2	ipc_get_proc_object . . . . .	215
4.51.3.3	ipc_object_create . . . . .	216
4.51.3.4	ipc_receive . . . . .	216
4.51.3.5	ipc_reply . . . . .	218
4.51.3.6	ipc_send . . . . .	219
4.52	include/jinue-common/asm/e820.h File Reference . . . . .	221
4.52.1	Macro Definition Documentation . . . . .	222
4.52.1.1	E820_ACPI . . . . .	222
4.52.1.2	E820_RAM . . . . .	222
4.52.1.3	E820_RESERVED . . . . .	222

4.52.1.4	E820_SMAP . . . . .	222
4.53	include/jinue/errno.h File Reference . . . . .	222
4.54	include/jinue-common/errno.h File Reference . . . . .	223
4.54.1	Macro Definition Documentation . . . . .	223
4.54.1.1	JINUE_E2BIG . . . . .	223
4.54.1.2	JINUE_EAGAIN . . . . .	223
4.54.1.3	JINUE_EBADF . . . . .	223
4.54.1.4	JINUE_EINVAL . . . . .	223
4.54.1.5	JINUE_EIO . . . . .	224
4.54.1.6	JINUE_EMORE . . . . .	224
4.54.1.7	JINUE_ENOMEM . . . . .	224
4.54.1.8	JINUE_ENOSYS . . . . .	224
4.54.1.9	JINUE_EPERM . . . . .	224
4.55	include/jinue/list.h File Reference . . . . .	224
4.56	include/jinue-common/list.h File Reference . . . . .	225
4.56.1	Macro Definition Documentation . . . . .	226
4.56.1.1	jinue_cursor_entry . . . . .	226
4.56.1.2	jinue_list_pop . . . . .	226
4.56.1.3	JINUE_LIST_STATIC . . . . .	226
4.56.1.4	jinue_node_entry . . . . .	226
4.56.1.5	JINUE_OFFSETOF . . . . .	226
4.56.2	Typedef Documentation . . . . .	226
4.56.2.1	jinue_cursor_t . . . . .	226
4.56.2.2	jinue_node_t . . . . .	226
4.57	include/jinue/memory.h File Reference . . . . .	226
4.57.1	Function Documentation . . . . .	227
4.57.1.1	jinue_get_phys_memory . . . . .	227
4.57.1.2	jinue_pys_mem_type_description . . . . .	227
4.58	include/jinue/syscall.h File Reference . . . . .	227
4.58.1	Function Documentation . . . . .	228
4.58.1.1	jinue_call . . . . .	228
4.58.1.2	jinue_call_raw . . . . .	228
4.58.1.3	jinue_get_syscall_implementation . . . . .	228
4.58.1.4	jinue_get_syscall_implementation_name . . . . .	228
4.58.1.5	jinue_get_thread_local_storage . . . . .	228
4.58.1.6	jinue_set_thread_local_storage . . . . .	228
4.58.1.7	jinue_thread_create . . . . .	228

4.58.1.8	jinue_thread_exit . . . . .	228
4.58.1.9	jinue_yield . . . . .	228
4.59	include/jinue-common/asm/syscall.h File Reference . . . . .	228
4.59.1	Macro Definition Documentation . . . . .	229
4.59.1.1	SYSCALL_FUNCT_CONSOLE_PUTC . . . . .	229
4.59.1.2	SYSCALL_FUNCT_CONSOLE_PUTS . . . . .	229
4.59.1.3	SYSCALL_FUNCT_CREATE_IPC . . . . .	230
4.59.1.4	SYSCALL_FUNCT_GET_PHYS_MEMORY . . . . .	230
4.59.1.5	SYSCALL_FUNCT_GET_THREAD_LOCAL_ADDR . . . . .	230
4.59.1.6	SYSCALL_FUNCT_PROC_BASE . . . . .	230
4.59.1.7	SYSCALL_FUNCT_RECEIVE . . . . .	230
4.59.1.8	SYSCALL_FUNCT_REPLY . . . . .	230
4.59.1.9	SYSCALL_FUNCT_SET_THREAD_LOCAL_ADDR . . . . .	230
4.59.1.10	SYSCALL_FUNCT_SYSCALL_METHOD . . . . .	231
4.59.1.11	SYSCALL_FUNCT_SYSTEM_BASE . . . . .	231
4.59.1.12	SYSCALL_FUNCT_THREAD_CREATE . . . . .	231
4.59.1.13	SYSCALL_FUNCT_THREAD_YIELD . . . . .	231
4.59.1.14	SYSCALL_FUNCT_USER_BASE . . . . .	231
4.59.1.15	SYSCALL_IRQ . . . . .	231
4.59.1.16	SYSCALL_METHOD_FAST_AMD . . . . .	231
4.59.1.17	SYSCALL_METHOD_FAST_INTEL . . . . .	231
4.59.1.18	SYSCALL_METHOD_INTR . . . . .	232
4.60	include/jinue-common/syscall.h File Reference . . . . .	232
4.61	include/syscall.h File Reference . . . . .	232
4.61.1	Function Documentation . . . . .	233
4.61.1.1	dispatch_syscall . . . . .	233
4.62	include/kbd.h File Reference . . . . .	236
4.62.1	Function Documentation . . . . .	236
4.62.1.1	any_key . . . . .	236
4.63	include/kmain.h File Reference . . . . .	237
4.63.1	Function Documentation . . . . .	237
4.63.1.1	kmain . . . . .	237
4.64	include/kstdc/assert.h File Reference . . . . .	239
4.64.1	Macro Definition Documentation . . . . .	240
4.64.1.1	assert . . . . .	240
4.64.2	Function Documentation . . . . .	240
4.64.2.1	__assert_failed . . . . .	240

4.65	include/kstdc/stdarg.h File Reference . . . . .	241
4.65.1	Macro Definition Documentation . . . . .	241
4.65.1.1	va_arg . . . . .	241
4.65.1.2	va_copy . . . . .	241
4.65.1.3	va_end . . . . .	241
4.65.1.4	va_start . . . . .	241
4.65.2	Typedef Documentation . . . . .	241
4.65.2.1	va_list . . . . .	241
4.66	include/kstdc/stdbool.h File Reference . . . . .	241
4.66.1	Macro Definition Documentation . . . . .	242
4.66.1.1	__bool_true_false_are_defined . . . . .	242
4.66.1.2	bool . . . . .	242
4.66.1.3	false . . . . .	242
4.66.1.4	true . . . . .	242
4.67	include/kstdc/stddef.h File Reference . . . . .	242
4.67.1	Macro Definition Documentation . . . . .	243
4.67.1.1	NULL . . . . .	243
4.67.1.2	offsetof . . . . .	243
4.67.2	Typedef Documentation . . . . .	243
4.67.2.1	ptrdiff_t . . . . .	243
4.67.2.2	size_t . . . . .	243
4.67.2.3	wchar_t . . . . .	243
4.68	include/kstdc/stdint.h File Reference . . . . .	243
4.68.1	Macro Definition Documentation . . . . .	244
4.68.1.1	INT64_C . . . . .	244
4.68.1.2	UINT64_C . . . . .	244
4.68.2	Typedef Documentation . . . . .	244
4.68.2.1	int16_t . . . . .	244
4.68.2.2	int32_t . . . . .	244
4.68.2.3	int64_t . . . . .	244
4.68.2.4	int8_t . . . . .	244
4.68.2.5	intptr_t . . . . .	244
4.68.2.6	uint16_t . . . . .	245
4.68.2.7	uint32_t . . . . .	245
4.68.2.8	uint64_t . . . . .	245
4.68.2.9	uint8_t . . . . .	245
4.68.2.10	uintptr_t . . . . .	245



4.69	include/kstdc/stdlib.h File Reference . . . . .	245
4.69.1	Macro Definition Documentation . . . . .	245
4.69.1.1	EXIT_FAILURE . . . . .	245
4.69.1.2	EXIT_SUCCESS . . . . .	246
4.70	include/kstdc/string.h File Reference . . . . .	246
4.70.1	Function Documentation . . . . .	246
4.70.1.1	memcpy . . . . .	246
4.70.1.2	memset . . . . .	247
4.70.1.3	strlen . . . . .	247
4.71	include/object.h File Reference . . . . .	247
4.71.1	Macro Definition Documentation . . . . .	248
4.71.1.1	OBJECT_FLAG_DESTROYED . . . . .	248
4.71.1.2	OBJECT_FLAG_NONE . . . . .	248
4.71.1.3	OBJECT_REF_FLAG_CLOSED . . . . .	248
4.71.1.4	OBJECT_REF_FLAG_NONE . . . . .	248
4.71.1.5	OBJECT_REF_FLAG_OWNER . . . . .	248
4.71.1.6	OBJECT_REF_FLAG_VALID . . . . .	249
4.71.1.7	OBJECT_TYPE_IPC . . . . .	249
4.71.1.8	OBJECT_TYPE_PROCESS . . . . .	249
4.71.1.9	OBJECT_TYPE_THREAD . . . . .	249
4.72	include/page_alloc.h File Reference . . . . .	249
4.72.1	Function Documentation . . . . .	250
4.72.1.1	add_page_frame . . . . .	250
4.72.1.2	clear_page . . . . .	251
4.72.1.3	page_alloc . . . . .	251
4.72.1.4	page_alloc_is_empty . . . . .	252
4.72.1.5	page_free . . . . .	252
4.72.1.6	remove_page_frame . . . . .	253
4.73	include/panic.h File Reference . . . . .	254
4.73.1	Function Documentation . . . . .	254
4.73.1.1	panic . . . . .	254
4.74	include/pfalloc.h File Reference . . . . .	255
4.74.1	Macro Definition Documentation . . . . .	256
4.74.1.1	KERNEL_PAGE_STACK_INIT . . . . .	256
4.74.1.2	KERNEL_PAGE_STACK_SIZE . . . . .	256
4.74.1.3	pfalloc . . . . .	256
4.74.1.4	pffree . . . . .	256

4.74.2	Function Documentation . . . . .	257
4.74.2.1	init_pmalloc_cache . . . . .	257
4.74.2.2	pmalloc_from . . . . .	257
4.74.2.3	pffree_to . . . . .	258
4.74.3	Variable Documentation . . . . .	258
4.74.3.1	global_pmalloc_cache . . . . .	258
4.75	include/printk.h File Reference . . . . .	258
4.75.1	Function Documentation . . . . .	258
4.75.1.1	print_hex_b . . . . .	258
4.75.1.2	print_hex_l . . . . .	258
4.75.1.3	print_hex_nibble . . . . .	258
4.75.1.4	print_hex_q . . . . .	258
4.75.1.5	print_hex_w . . . . .	259
4.75.1.6	print_unsigned_int . . . . .	259
4.75.1.7	printk . . . . .	259
4.76	include/process.h File Reference . . . . .	259
4.76.1	Function Documentation . . . . .	259
4.76.1.1	process_boot_init . . . . .	259
4.76.1.2	process_create . . . . .	260
4.76.1.3	process_create_initial . . . . .	261
4.76.1.4	process_get_descriptor . . . . .	261
4.76.1.5	process_unused_descriptor . . . . .	262
4.77	include/slab.h File Reference . . . . .	262
4.77.1	Macro Definition Documentation . . . . .	264
4.77.1.1	SLAB_COMPACT . . . . .	264
4.77.1.2	SLAB_DEFAULT_WORKING_SET . . . . .	264
4.77.1.3	SLAB_DEFAULTS . . . . .	264
4.77.1.4	SLAB_HWCACHE_ALIGN . . . . .	264
4.77.1.5	SLAB_POISON . . . . .	265
4.77.1.6	SLAB_POISON_ALIVE_VALUE . . . . .	265
4.77.1.7	SLAB_POISON_DEAD_VALUE . . . . .	265
4.77.1.8	SLAB_RED_ZONE . . . . .	265
4.77.1.9	SLAB_RED_ZONE_VALUE . . . . .	265
4.77.1.10	SLAB_SIZE . . . . .	265
4.77.2	Typedef Documentation . . . . .	265
4.77.2.1	slab_bufctl_t . . . . .	265
4.77.2.2	slab_cache_t . . . . .	265

4.77.2.3	slab_ctor_t . . . . .	265
4.77.2.4	slab_t . . . . .	266
4.77.3	Function Documentation . . . . .	266
4.77.3.1	slab_cache_alloc . . . . .	266
4.77.3.2	slab_cache_free . . . . .	268
4.77.3.3	slab_cache_init . . . . .	270
4.77.3.4	slab_cache_reap . . . . .	272
4.77.3.5	slab_cache_set_working_set . . . . .	272
4.77.4	Variable Documentation . . . . .	273
4.77.4.1	slab_cache_list . . . . .	273
4.78	include/util.h File Reference . . . . .	273
4.78.1	Macro Definition Documentation . . . . .	273
4.78.1.1	ALIGN_END . . . . .	273
4.78.1.2	ALIGN_END_PTR . . . . .	274
4.78.1.3	ALIGN_START . . . . .	274
4.78.1.4	ALIGN_START_PTR . . . . .	274
4.78.1.5	alloc_backward . . . . .	274
4.78.1.6	alloc_forward . . . . .	274
4.78.1.7	OFFSET_OF_PTR . . . . .	274
4.79	include/vmalloc.h File Reference . . . . .	274
4.79.1	Function Documentation . . . . .	275
4.79.1.1	vmalloc . . . . .	275
4.79.1.2	vmalloc_init . . . . .	275
4.79.1.3	vmalloc_is_in_range . . . . .	276
4.79.1.4	vmfree . . . . .	276
4.80	kernel/boot.c File Reference . . . . .	276
4.80.1	Function Documentation . . . . .	277
4.80.1.1	boot_alloc_init . . . . .	277
4.80.1.2	boot_heap_alloc_size . . . . .	278
4.80.1.3	boot_heap_pop . . . . .	278
4.80.1.4	boot_heap_push . . . . .	279
4.80.1.5	boot_page_alloc . . . . .	280
4.80.1.6	boot_page_alloc_early . . . . .	280
4.80.1.7	boot_page_alloc_image . . . . .	282
4.80.1.8	boot_page_frame_alloc . . . . .	283
4.80.1.9	boot_vmalloc . . . . .	284
4.81	kernel/hal/boot.c File Reference . . . . .	285

4.81.1	Function Documentation . . . . .	285
4.81.1.1	boot_info_check . . . . .	286
4.81.1.2	boot_info_dump . . . . .	286
4.81.1.3	get_boot_info . . . . .	287
4.81.2	Variable Documentation . . . . .	287
4.81.2.1	boot_info . . . . .	287
4.82	kernel/build-info.gen.h File Reference . . . . .	288
4.82.1	Macro Definition Documentation . . . . .	288
4.82.1.1	BUILD_HOST . . . . .	288
4.82.1.2	BUILD_TIME . . . . .	288
4.82.1.3	GIT_REVISION . . . . .	288
4.83	kernel/c-assert.c File Reference . . . . .	288
4.83.1	Function Documentation . . . . .	289
4.83.1.1	__assert_failed . . . . .	289
4.84	kernel/c-string.c File Reference . . . . .	290
4.84.1	Function Documentation . . . . .	290
4.84.1.1	memcpy . . . . .	290
4.84.1.2	memset . . . . .	290
4.84.1.3	strlen . . . . .	291
4.85	kernel/console.c File Reference . . . . .	291
4.85.1	Function Documentation . . . . .	291
4.85.1.1	console_init . . . . .	292
4.85.1.2	console_print . . . . .	292
4.85.1.3	console_printn . . . . .	292
4.85.1.4	console_putc . . . . .	293
4.86	kernel/debug.c File Reference . . . . .	293
4.86.1	Function Documentation . . . . .	294
4.86.1.1	dump_call_stack . . . . .	294
4.87	kernel/elf.c File Reference . . . . .	295
4.87.1	Function Documentation . . . . .	296
4.87.1.1	elf_check . . . . .	296
4.87.1.2	elf_load . . . . .	297
4.87.1.3	elf_lookup_symbol . . . . .	299
4.87.1.4	elf_setup_stack . . . . .	300
4.88	kernel/hal/cpu.c File Reference . . . . .	302
4.88.1	Function Documentation . . . . .	303
4.88.1.1	cpu_detect_features . . . . .	303

4.88.1.2	cpu_init_data . . . . .	305
4.88.2	Variable Documentation . . . . .	306
4.88.2.1	cpu_info . . . . .	306
4.89	kernel/hal/hal.c File Reference . . . . .	306
4.89.1	Function Documentation . . . . .	307
4.89.1.1	hal_init . . . . .	307
4.89.2	Variable Documentation . . . . .	309
4.89.2.1	syscall_method . . . . .	309
4.90	kernel/hal/interrupt.c File Reference . . . . .	309
4.90.1	Function Documentation . . . . .	310
4.90.1.1	dispatch_interrupt . . . . .	310
4.91	kernel/hal/mem.c File Reference . . . . .	311
4.91.1	Function Documentation . . . . .	312
4.91.1.1	mem_check_memory . . . . .	312
4.92	kernel/hal/pic8259.c File Reference . . . . .	314
4.92.1	Function Documentation . . . . .	315
4.92.1.1	pic8259_eoi . . . . .	315
4.92.1.2	pic8259_init . . . . .	315
4.92.1.3	pic8259_mask_irq . . . . .	316
4.92.1.4	pic8259_unmask_irq . . . . .	317
4.93	kernel/hal/serial.c File Reference . . . . .	318
4.93.1	Function Documentation . . . . .	318
4.93.1.1	serial_init . . . . .	318
4.93.1.2	serial_printn . . . . .	319
4.93.1.3	serial_putc . . . . .	320
4.94	kernel/hal/thread.c File Reference . . . . .	320
4.94.1	Function Documentation . . . . .	321
4.94.1.1	thread_context_switch . . . . .	321
4.94.1.2	thread_context_switch_stack . . . . .	321
4.94.1.3	thread_page_init . . . . .	322
4.95	kernel/thread.c File Reference . . . . .	322
4.95.1	Function Documentation . . . . .	323
4.95.1.1	thread_create . . . . .	323
4.95.1.2	thread_create_boot . . . . .	324
4.95.1.3	thread_destroy . . . . .	324
4.95.1.4	thread_ready . . . . .	325
4.95.1.5	thread_switch . . . . .	325

4.95.1.6	thread_yield_from . . . . .	326
4.96	kernel/hal/vga.c File Reference . . . . .	326
4.96.1	Function Documentation . . . . .	327
4.96.1.1	vga_clear . . . . .	327
4.96.1.2	vga_get_cursor_pos . . . . .	327
4.96.1.3	vga_init . . . . .	328
4.96.1.4	vga_print . . . . .	329
4.96.1.5	vga_printn . . . . .	329
4.96.1.6	vga_putc . . . . .	330
4.96.1.7	vga_scroll . . . . .	330
4.96.1.8	vga_set_base_addr . . . . .	330
4.96.1.9	vga_set_cursor_pos . . . . .	331
4.97	kernel/hal/vm.c File Reference . . . . .	331
4.97.1	Function Documentation . . . . .	332
4.97.1.1	vm_boot_init . . . . .	332
4.97.1.2	vm_boot_postinit . . . . .	334
4.97.1.3	vm_change_flags . . . . .	335
4.97.1.4	vm_clone_page_directory . . . . .	335
4.97.1.5	vm_create_addr_space . . . . .	336
4.97.1.6	vm_create_initial_addr_space . . . . .	337
4.97.1.7	vm_destroy_addr_space . . . . .	337
4.97.1.8	vm_destroy_page_directory . . . . .	338
4.97.1.9	vm_init_initial_page_directory . . . . .	338
4.97.1.10	vm_lookup_kernel_paddr . . . . .	339
4.97.1.11	vm_map_early . . . . .	340
4.97.1.12	vm_map_kernel . . . . .	340
4.97.1.13	vm_map_user . . . . .	340
4.97.1.14	vm_switch_addr_space . . . . .	341
4.97.1.15	vm_unmap_kernel . . . . .	341
4.97.1.16	vm_unmap_user . . . . .	341
4.97.2	Variable Documentation . . . . .	341
4.97.2.1	global_page_tables . . . . .	341
4.97.2.2	initial_addr_space . . . . .	341
4.97.2.3	page_table_entries . . . . .	342
4.98	kernel/hal/vm_pae.c File Reference . . . . .	342
4.98.1	Macro Definition Documentation . . . . .	343
4.98.1.1	PDPT_BITS . . . . .	343

4.98.1.2	PDPT_ENTRIES . . . . .	343
4.98.2	Function Documentation . . . . .	343
4.98.2.1	vm_pae_boot_init . . . . .	343
4.98.2.2	vm_pae_clear_pte . . . . .	344
4.98.2.3	vm_pae_copy_pte . . . . .	344
4.98.2.4	vm_pae_create_addr_space . . . . .	344
4.98.2.5	vm_pae_create_initial_addr_space . . . . .	345
4.98.2.6	vm_pae_create_pdpt_cache . . . . .	346
4.98.2.7	vm_pae_destroy_addr_space . . . . .	347
4.98.2.8	vm_pae_get_pte_flags . . . . .	348
4.98.2.9	vm_pae_get_pte_paddr . . . . .	348
4.98.2.10	vm_pae_get_pte_with_offset . . . . .	348
4.98.2.11	vm_pae_lookup_page_directory . . . . .	348
4.98.2.12	vm_pae_page_directory_offset_of . . . . .	350
4.98.2.13	vm_pae_page_table_offset_of . . . . .	350
4.98.2.14	vm_pae_set_pte . . . . .	350
4.98.2.15	vm_pae_set_pte_flags . . . . .	351
4.98.2.16	vm_pae_unmap_low_alias . . . . .	351
4.98.3	Variable Documentation . . . . .	351
4.98.3.1	initial_pdpt . . . . .	351
4.99	kernel/hal/vm_x86.c File Reference . . . . .	351
4.99.1	Function Documentation . . . . .	352
4.99.1.1	vm_x86_boot_init . . . . .	352
4.99.1.2	vm_x86_clear_pte . . . . .	353
4.99.1.3	vm_x86_copy_pte . . . . .	353
4.99.1.4	vm_x86_create_addr_space . . . . .	353
4.99.1.5	vm_x86_create_initial_addr_space . . . . .	354
4.99.1.6	vm_x86_destroy_addr_space . . . . .	354
4.99.1.7	vm_x86_get_pte_flags . . . . .	355
4.99.1.8	vm_x86_get_pte_paddr . . . . .	355
4.99.1.9	vm_x86_get_pte_with_offset . . . . .	355
4.99.1.10	vm_x86_lookup_page_directory . . . . .	356
4.99.1.11	vm_x86_page_directory_offset_of . . . . .	356
4.99.1.12	vm_x86_page_table_offset_of . . . . .	356
4.99.1.13	vm_x86_set_pte . . . . .	357
4.99.1.14	vm_x86_set_pte_flags . . . . .	357
4.100	kernel/ipc.c File Reference . . . . .	357

4.100.1 Function Documentation . . . . .	358
4.100.1.1 ipc_boot_init . . . . .	358
4.100.1.2 ipc_get_proc_object . . . . .	358
4.100.1.3 ipc_object_create . . . . .	358
4.100.1.4 ipc_receive . . . . .	359
4.100.1.5 ipc_reply . . . . .	361
4.100.1.6 ipc_send . . . . .	362
4.101kernel/kbd.c File Reference . . . . .	364
4.101.1 Function Documentation . . . . .	364
4.101.1.1 any_key . . . . .	364
4.102kernel/kmain.c File Reference . . . . .	365
4.102.1 Function Documentation . . . . .	366
4.102.1.1 kmain . . . . .	366
4.103kernel/page_alloc.c File Reference . . . . .	368
4.103.1 Function Documentation . . . . .	369
4.103.1.1 add_page_frame . . . . .	369
4.103.1.2 clear_page . . . . .	370
4.103.1.3 page_alloc . . . . .	371
4.103.1.4 page_alloc_is_empty . . . . .	371
4.103.1.5 page_free . . . . .	372
4.103.1.6 remove_page_frame . . . . .	372
4.104kernel/panic.c File Reference . . . . .	373
4.104.1 Function Documentation . . . . .	373
4.104.1.1 panic . . . . .	373
4.105kernel/pfalloc.c File Reference . . . . .	374
4.105.1 Function Documentation . . . . .	375
4.105.1.1 init_pfalloc_cache . . . . .	375
4.105.1.2 pfalloc_from . . . . .	375
4.105.1.3 pffree_to . . . . .	376
4.105.2 Variable Documentation . . . . .	376
4.105.2.1 global_pfalloc_cache . . . . .	376
4.106kernel/process.c File Reference . . . . .	377
4.106.1 Function Documentation . . . . .	377
4.106.1.1 process_boot_init . . . . .	377
4.106.1.2 process_create . . . . .	378
4.106.1.3 process_create_initial . . . . .	379
4.106.1.4 process_get_descriptor . . . . .	379



4.106.1.5 process_unused_descriptor . . . . .	379
4.107kernel/slab.c File Reference . . . . .	380
4.107.1 Detailed Description . . . . .	381
4.107.2 Function Documentation . . . . .	381
4.107.2.1 slab_cache_alloc . . . . .	381
4.107.2.2 slab_cache_free . . . . .	383
4.107.2.3 slab_cache_init . . . . .	385
4.107.2.4 slab_cache_reap . . . . .	387
4.107.2.5 slab_cache_set_working_set . . . . .	387
4.108kernel/syscall.c File Reference . . . . .	388
4.108.1 Function Documentation . . . . .	388
4.108.1.1 dispatch_syscall . . . . .	388
4.109kernel/vmalloc.c File Reference . . . . .	391
4.109.1 Detailed Description . . . . .	393
4.109.2 Macro Definition Documentation . . . . .	393
4.109.2.1 VMALLOC_BLOCK_SIZE . . . . .	393
4.109.2.2 VMALLOC_STACK_ENTRIES . . . . .	393
4.109.3 Typedef Documentation . . . . .	393
4.109.3.1 vmalloc_block_t . . . . .	393
4.109.4 Function Documentation . . . . .	393
4.109.4.1 vmalloc . . . . .	393
4.109.4.2 vmalloc_init . . . . .	394
4.109.4.3 vmalloc_is_in_range . . . . .	394
4.109.4.4 vmfree . . . . .	395



# Chapter 1

## Data Structure Index

### 1.1 Data Structures

Here are the data structures with brief descriptions:

<b>addr_space_t</b>	7
<b>alloc_page</b>	8
<b>boot_alloc_t</b>	9
<b>boot_heap_pushed_state</b>	10
<b>boot_info_t</b>	11
<b>cpu_data_t</b>	14
<b>cpu_info_t</b>	15
<b>e820_t</b>	16
<b>Elf32_auxv_t</b>	17
<b>Elf32_Ehdr</b>	18
<b>Elf32_Phdr</b>	20
<b>Elf32_Shdr</b>	21
<b>Elf32_Sym</b>	23
<b>elf_info_t</b>	24
<b>elf_symbol_t</b>	26
<b>ipc_t</b>	27
<b>jinue_list_t</b>	28
<b>jinue_mem_entry_t</b>	29
<b>jinue_mem_map_t</b>	30
<b>jinue_message_t</b>	31
<b>jinue_node_t</b>	31
<b>jinue_reply_t</b>	32
<b>jinue_syscall_args_t</b>	33
<b>kernel_context_t</b>	34
<b>message_info_t</b>	35
<b>object_header_t</b>	36
<b>object_ref_t</b>	37
<b>pdpt_t</b>	38
<b>pfalloc_cache_t</b>	38
<b>process_t</b>	39
<b>pseudo_descriptor_t</b>	40
<b>pte_t</b>	41
<b>slab_bufctl_t</b>	42
<b>slab_cache_t</b>	42

slab_t . . . . .	45
thread_context_t . . . . .	46
thread_t . . . . .	47
trapframe_t . . . . .	49
tss_t . . . . .	52
vmalloc_block_t . . . . .	55
vmalloc_t . . . . .	56
x86_cpuid_regs_t . . . . .	58

## Chapter 2

# File Index

### 2.1 File List

Here is a list of all files with brief descriptions:

include/ <b>ascii.h</b>	61
include/ <b>boot.h</b>	62
include/ <b>console.h</b>	76
include/ <b>debug.h</b>	79
include/ <b>elf.h</b>	81
include/ <b>ipc.h</b>	210
include/ <b>kbd.h</b>	236
include/ <b>kmain.h</b>	237
include/ <b>object.h</b>	247
include/ <b>page_alloc.h</b>	249
include/ <b>panic.h</b>	254
include/ <b>pfalloc.h</b>	255
include/ <b>printk.h</b>	258
include/ <b>process.h</b>	259
include/ <b>slab.h</b>	262
include/ <b>syscall.h</b>	232
include/ <b>thread.h</b>	135
include/ <b>types.h</b>	181
include/ <b>util.h</b>	273
include/ <b>vmalloc.h</b>	274
include/hal/ <b>abi.h</b>	103
include/hal/ <b>boot.h</b>	65
include/hal/ <b>cpu.h</b>	160
include/hal/ <b>cpu_data.h</b>	168
include/hal/ <b>descriptors.h</b>	111
include/hal/ <b>hal.h</b>	169
include/hal/ <b>interrupt.h</b>	171
include/hal/ <b>io.h</b>	173
include/hal/ <b>mem.h</b>	117
include/hal/ <b>pic8259.h</b>	122
include/hal/ <b>serial.h</b>	129
include/hal/ <b>startup.h</b>	175
include/hal/ <b>thread.h</b>	133
include/hal/ <b>trap.h</b>	175

include/hal/ <b>types.h</b>	176
include/hal/ <b>vga.h</b>	182
include/hal/ <b>vm.h</b>	141
include/hal/ <b>vm_pae.h</b>	190
include/hal/ <b>vm_private.h</b>	199
include/hal/ <b>vm_x86.h</b>	204
include/hal/ <b>x86.h</b>	157
include/hal/asm/ <b>boot.h</b>	62
include/hal/asm/ <b>descriptors.h</b>	104
include/hal/asm/ <b>irq.h</b>	112
include/hal/asm/ <b>mem.h</b>	116
include/hal/asm/ <b>pic8259.h</b>	120
include/hal/asm/ <b>serial.h</b>	127
include/hal/asm/ <b>thread.h</b>	131
include/hal/asm/ <b>vm.h</b>	139
include/hal/asm/ <b>x86.h</b>	154
include/jinue-common/ <b>console.h</b>	76
include/jinue-common/ <b>elf.h</b>	82
include/jinue-common/ <b>errno.h</b>	223
include/jinue-common/ <b>ipc.h</b>	213
include/jinue-common/ <b>list.h</b>	225
include/jinue-common/ <b>syscall.h</b>	232
include/jinue-common/ <b>types.h</b>	180
include/jinue-common/ <b>vm.h</b>	152
include/jinue-common/asm/ <b>e820.h</b>	221
include/jinue-common/asm/ <b>ipc.h</b>	211
include/jinue-common/asm/ <b>syscall.h</b>	228
include/jinue-common/asm/ <b>types.h</b>	179
include/jinue-common/asm/ <b>vm.h</b>	150
include/jinue/ <b>console.h</b>	76
include/jinue/ <b>elf.h</b>	81
include/jinue/ <b>errno.h</b>	222
include/jinue/ <b>ipc.h</b>	210
include/jinue/ <b>list.h</b>	224
include/jinue/ <b>memory.h</b>	226
include/jinue/ <b>syscall.h</b>	227
include/jinue/ <b>types.h</b>	179
include/jinue/ <b>vm.h</b>	149
include/kstdc/ <b>assert.h</b>	239
include/kstdc/ <b>stdarg.h</b>	241
include/kstdc/ <b>stdbool.h</b>	241
include/kstdc/ <b>stddef.h</b>	242
include/kstdc/ <b>stdint.h</b>	243
include/kstdc/ <b>stdlib.h</b>	245
include/kstdc/ <b>string.h</b>	246
kernel/ <b>boot.c</b>	276
kernel/ <b>build-info.gen.h</b>	288
kernel/ <b>c-assert.c</b>	288
kernel/ <b>c-string.c</b>	290
kernel/ <b>console.c</b>	291
kernel/ <b>debug.c</b>	293
kernel/ <b>elf.c</b>	295
kernel/ <b>ipc.c</b>	357
kernel/ <b>kbd.c</b>	364

kernel/ <b>kmain.c</b>	365
kernel/ <b>page_alloc.c</b>	368
kernel/ <b>panic.c</b>	373
kernel/ <b>pfalloc.c</b>	374
kernel/ <b>process.c</b>	377
kernel/ <b>slab.c</b>	
Kernel object allocator	380
kernel/ <b>syscall.c</b>	388
kernel/ <b>thread.c</b>	322
kernel/ <b>vmalloc.c</b>	
Virtual address space allocator	391
kernel/hal/ <b>boot.c</b>	285
kernel/hal/ <b>cpu.c</b>	302
kernel/hal/ <b>hal.c</b>	306
kernel/hal/ <b>interrupt.c</b>	309
kernel/hal/ <b>mem.c</b>	311
kernel/hal/ <b>pic8259.c</b>	314
kernel/hal/ <b>serial.c</b>	318
kernel/hal/ <b>thread.c</b>	320
kernel/hal/ <b>vga.c</b>	326
kernel/hal/ <b>vm.c</b>	331
kernel/hal/ <b>vm_pae.c</b>	342
kernel/hal/ <b>vm_x86.c</b>	351





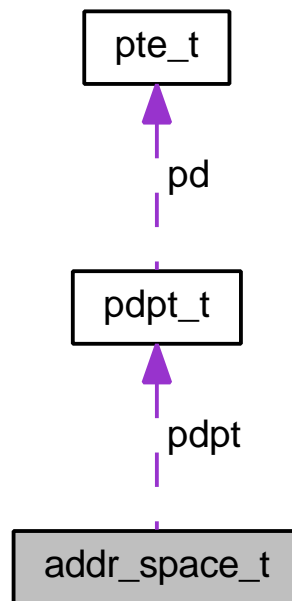
## Chapter 3

# Data Structure Documentation

### 3.1 `addr_space_t` Struct Reference

```
#include <hal/types.h>
```

Collaboration diagram for `addr_space_t`:



#### Data Fields

- `uint32_t cr3`
- union {
  - `kern_paddr_t pd`
  - `pdpt_t * pdpt`
- `top_level`

### 3.1.1 Detailed Description

Definition at line 80 of file types.h.

### 3.1.2 Field Documentation

#### 3.1.2.1 `uint32_t addr_space_t::cr3`

Definition at line 81 of file types.h.

Referenced by `vm_boot_init()`, `vm_pae_create_addr_space()`, `vm_pae_create_initial_addr_space()`, `vm_switch_addr_space()`, `vm_x86_create_addr_space()`, and `vm_x86_create_initial_addr_space()`.

#### 3.1.2.2 `kern_paddr_t addr_space_t::pd`

Definition at line 83 of file types.h.

Referenced by `vm_x86_create_addr_space()`, `vm_x86_create_initial_addr_space()`, `vm_x86_destroy_addr_space()`, and `vm_x86_lookup_page_directory()`.

#### 3.1.2.3 `pdpt_t* addr_space_t::pdpt`

Definition at line 84 of file types.h.

Referenced by `vm_pae_create_addr_space()`, `vm_pae_create_initial_addr_space()`, `vm_pae_destroy_addr_space()`, `vm_pae_lookup_page_directory()`, and `vm_pae_unmap_low_alias()`.

#### 3.1.2.4 `union { ... } addr_space_t::top_level`

Referenced by `vm_pae_create_addr_space()`, `vm_pae_create_initial_addr_space()`, `vm_pae_destroy_addr_space()`, `vm_pae_lookup_page_directory()`, `vm_pae_unmap_low_alias()`, `vm_x86_create_addr_space()`, `vm_x86_create_initial_addr_space()`, `vm_x86_destroy_addr_space()`, and `vm_x86_lookup_page_directory()`.

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

- `include/hal/types.h`

## 3.2 `alloc_page` Struct Reference

Collaboration diagram for `alloc_page`:



### Data Fields

- struct `alloc_page` \* `next`

### 3.2.1 Detailed Description

Definition at line 38 of file page\_alloc.c.

### 3.2.2 Field Documentation

#### 3.2.2.1 struct alloc\_page\* alloc\_page::next

Definition at line 39 of file page\_alloc.c.

Referenced by page\_alloc(), and page\_free().

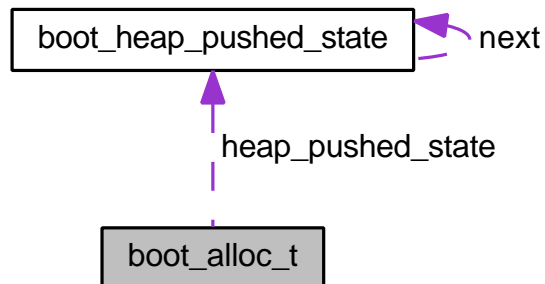
The documentation for this struct was generated from the following file:

- kernel/page\_alloc.c

## 3.3 boot\_alloc\_t Struct Reference

```
#include <types.h>
```

Collaboration diagram for boot\_alloc\_t:



### Data Fields

- void \* **heap\_ptr**
- struct **boot\_heap\_pushed\_state** \* **heap\_pushed\_state**
- **addr\_t** **kernel\_vm\_top**
- **addr\_t** **kernel\_vm\_limit**
- **kern\_paddr\_t** **kernel\_paddr\_top**
- **kern\_paddr\_t** **kernel\_paddr\_limit**
- **bool** **its\_early**

### 3.3.1 Detailed Description

Definition at line 48 of file types.h.

### 3.3.2 Field Documentation

#### 3.3.2.1 `void* boot_alloc_t::heap_ptr`

Definition at line 49 of file `types.h`.

Referenced by `boot_alloc_init()`, `boot_heap_alloc_size()`, and `boot_heap_pop()`.

#### 3.3.2.2 `struct boot_heap_pushed_state* boot_alloc_t::heap_pushed_state`

Definition at line 50 of file `types.h`.

Referenced by `boot_heap_pop()`, and `boot_heap_push()`.

#### 3.3.2.3 `bool boot_alloc_t::its_early`

Definition at line 55 of file `types.h`.

Referenced by `boot_alloc_init()`, `boot_page_alloc_early()`, `boot_page_frame_alloc()`, `boot_vmalloc()`, and `vm_boot_init()`.

#### 3.3.2.4 `kern_paddr_t boot_alloc_t::kernel_paddr_limit`

Definition at line 54 of file `types.h`.

Referenced by `boot_page_alloc_early()`, `boot_page_frame_alloc()`, and `mem_check_memory()`.

#### 3.3.2.5 `kern_paddr_t boot_alloc_t::kernel_paddr_top`

Definition at line 53 of file `types.h`.

Referenced by `boot_page_alloc_early()`, `boot_page_frame_alloc()`, and `mem_check_memory()`.

#### 3.3.2.6 `addr_t boot_alloc_t::kernel_vm_limit`

Definition at line 52 of file `types.h`.

Referenced by `boot_page_alloc_early()`, `boot_vmalloc()`, and `mem_check_memory()`.

#### 3.3.2.7 `addr_t boot_alloc_t::kernel_vm_top`

Definition at line 51 of file `types.h`.

Referenced by `boot_page_alloc_early()`, `boot_vmalloc()`, and `mem_check_memory()`.

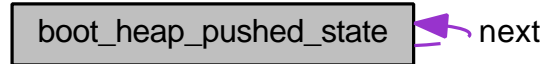
The documentation for this struct was generated from the following file:

- `include/types.h`

## 3.4 `boot_heap_pushed_state` Struct Reference

```
#include <types.h>
```

Collaboration diagram for boot\_heap\_pushed\_state:



### Data Fields

- struct **boot\_heap\_pushed\_state** \* **next**

#### 3.4.1 Detailed Description

Definition at line 44 of file types.h.

#### 3.4.2 Field Documentation

##### 3.4.2.1 struct boot\_heap\_pushed\_state\* boot\_heap\_pushed\_state::next

Definition at line 45 of file types.h.

Referenced by boot\_heap\_pop(), and boot\_heap\_push().

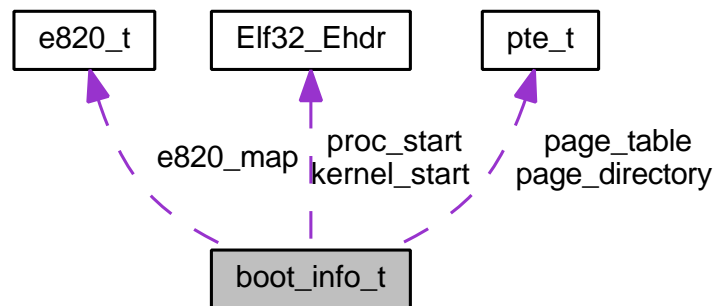
The documentation for this struct was generated from the following file:

- include/**types.h**

### 3.5 boot\_info\_t Struct Reference

```
#include <hal/types.h>
```

Collaboration diagram for boot\_info\_t:



### Data Fields

- **Elf32\_Ehdr** \* **kernel\_start**
- **uint32\_t** **kernel\_size**
- **Elf32\_Ehdr** \* **proc\_start**

- **uint32\_t proc\_size**
- **void \* image\_start**
- **void \* image\_top**
- **uint32\_t ramdisk\_start**
- **uint32\_t ramdisk\_size**
- **uint32\_t e820\_entries**
- **const e820\_t \* e820\_map**
- **void \* cmdline**
- **void \* boot\_heap**
- **void \* boot\_end**
- **pte\_t \* page\_table**
- **pte\_t \* page\_directory**
- **uint32\_t setup\_signature**

### 3.5.1 Detailed Description

Definition at line 96 of file types.h.

### 3.5.2 Field Documentation

#### 3.5.2.1 void\* boot\_info\_t::boot\_end

Definition at line 109 of file types.h.

Referenced by boot\_info\_dump(), and mem\_check\_memory().

#### 3.5.2.2 void\* boot\_info\_t::boot\_heap

Definition at line 108 of file types.h.

Referenced by boot\_info\_dump(), and kmain().

#### 3.5.2.3 void\* boot\_info\_t::cmdline

Definition at line 107 of file types.h.

Referenced by kmain().

#### 3.5.2.4 uint32\_t boot\_info\_t::e820\_entries

Definition at line 105 of file types.h.

Referenced by boot\_info\_dump(), dispatch\_syscall(), and mem\_check\_memory().

#### 3.5.2.5 const e820\_t\* boot\_info\_t::e820\_map

Definition at line 106 of file types.h.

Referenced by boot\_info\_dump(), dispatch\_syscall(), and mem\_check\_memory().

**3.5.2.6 void\* boot\_info\_t::image\_start**

Definition at line 101 of file types.h.

Referenced by boot\_info\_check(), boot\_info\_dump(), and vm\_boot\_init().

**3.5.2.7 void\* boot\_info\_t::image\_top**

Definition at line 102 of file types.h.

Referenced by boot\_info\_dump().

**3.5.2.8 uint32\_t boot\_info\_t::kernel\_size**

Definition at line 98 of file types.h.

Referenced by boot\_info\_dump(), and kmain().

**3.5.2.9 Elf32\_Ehdr\* boot\_info\_t::kernel\_start**

Definition at line 97 of file types.h.

Referenced by boot\_info\_check(), boot\_info\_dump(), and dump\_call\_stack().

**3.5.2.10 pte\_t\* boot\_info\_t::page\_directory**

Definition at line 111 of file types.h.

Referenced by boot\_info\_dump().

**3.5.2.11 pte\_t\* boot\_info\_t::page\_table**

Definition at line 110 of file types.h.

Referenced by boot\_info\_dump().

**3.5.2.12 uint32\_t boot\_info\_t::proc\_size**

Definition at line 100 of file types.h.

Referenced by boot\_info\_dump().

**3.5.2.13 Elf32\_Ehdr\* boot\_info\_t::proc\_start**

Definition at line 99 of file types.h.

Referenced by boot\_info\_dump().

**3.5.2.14 uint32\_t boot\_info\_t::ramdisk\_size**

Definition at line 104 of file types.h.

Referenced by kmain(), and mem\_check\_memory().

### 3.5.2.15 uint32\_t boot\_info\_t::ramdisk\_start

Definition at line 103 of file types.h.

Referenced by kmain(), and mem\_check\_memory().

### 3.5.2.16 uint32\_t boot\_info\_t::setup\_signature

Definition at line 112 of file types.h.

Referenced by boot\_info\_check(), and boot\_info\_dump().

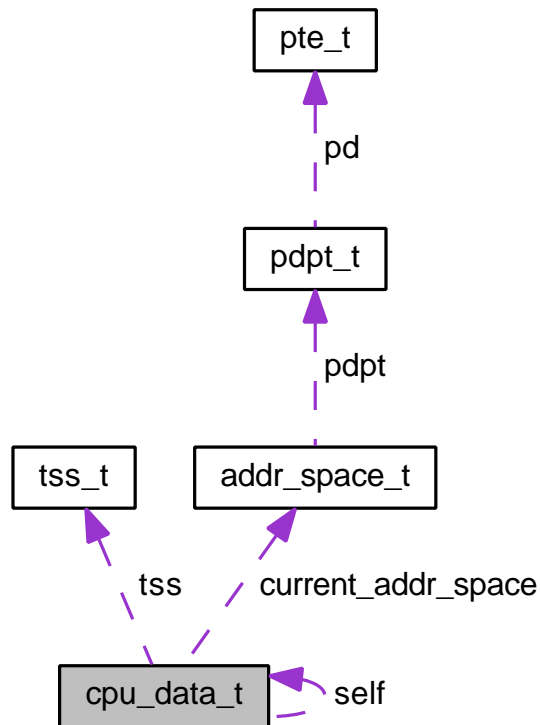
The documentation for this struct was generated from the following file:

- include/hal/**types.h**

## 3.6 cpu\_data\_t Struct Reference

```
#include <hal/types.h>
```

Collaboration diagram for cpu\_data\_t:



### Data Fields

- `seg_descriptor_t gdt [GDT_LENGTH]`
- `tss_t tss`
- `struct cpu_data_t * self`
- `addr_space_t * current_addr_space`



### 3.6.1 Detailed Description

Definition at line 181 of file `types.h`.

### 3.6.2 Field Documentation

#### 3.6.2.1 `addr_space_t* cpu_data_t::current_addr_space`

Definition at line 188 of file `types.h`.

Referenced by `cpu_init_data()`, and `vm_switch_addr_space()`.

#### 3.6.2.2 `seg_descriptor_t cpu_data_t::gdt[GDT_LENGTH]`

Definition at line 182 of file `types.h`.

Referenced by `cpu_init_data()`.

#### 3.6.2.3 `struct cpu_data_t* cpu_data_t::self`

Definition at line 187 of file `types.h`.

Referenced by `cpu_init_data()`.

#### 3.6.2.4 `tss_t cpu_data_t::tss`

Definition at line 186 of file `types.h`.

Referenced by `cpu_init_data()`.

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

- `include/hal/types.h`

## 3.7 `cpu_info_t` Struct Reference

```
#include <hal/cpu.h>
```

### Data Fields

- unsigned int `dcache_alignment`
- `uint32_t` `features`
- int `vendor`
- int `family`
- int `model`
- int `stepping`

### 3.7.1 Detailed Description

Definition at line 97 of file `cpu.h`.

### 3.7.2 Field Documentation

#### 3.7.2.1 unsigned int cpu\_info\_t::dcache\_alignment

Definition at line 98 of file cpu.h.

Referenced by cpu\_detect\_features(), and slab\_cache\_init().

#### 3.7.2.2 int cpu\_info\_t::family

Definition at line 101 of file cpu.h.

Referenced by cpu\_detect\_features().

#### 3.7.2.3 uint32\_t cpu\_info\_t::features

Definition at line 99 of file cpu.h.

Referenced by cpu\_detect\_features().

#### 3.7.2.4 int cpu\_info\_t::model

Definition at line 102 of file cpu.h.

Referenced by cpu\_detect\_features().

#### 3.7.2.5 int cpu\_info\_t::stepping

Definition at line 103 of file cpu.h.

Referenced by cpu\_detect\_features().

#### 3.7.2.6 int cpu\_info\_t::vendor

Definition at line 100 of file cpu.h.

Referenced by cpu\_detect\_features().

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

- include/hal/cpu.h

## 3.8 e820\_t Struct Reference

```
#include <hal/types.h>
```

### Data Fields

- uint64\_t addr
- uint64\_t size
- uint32\_t type

### 3.8.1 Detailed Description

Definition at line 88 of file types.h.

### 3.8.2 Field Documentation

#### 3.8.2.1 uint64\_t e820\_t::addr

Definition at line 89 of file types.h.

Referenced by dispatch\_syscall(), and mem\_check\_memory().

#### 3.8.2.2 uint64\_t e820\_t::size

Definition at line 90 of file types.h.

Referenced by dispatch\_syscall().

#### 3.8.2.3 uint32\_t e820\_t::type

Definition at line 91 of file types.h.

Referenced by dispatch\_syscall(), and mem\_check\_memory().

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

- include/hal/**types.h**

## 3.9 Elf32\_auxv\_t Struct Reference

```
#include <jinue-common/elf.h>
```

### Data Fields

- int **a\_type**
- union {  
    **int32\_t a\_val**  
} **a\_un**

### 3.9.1 Detailed Description

Definition at line 308 of file elf.h.

### 3.9.2 Field Documentation

#### 3.9.2.1 int Elf32\_auxv\_t::a\_type

Definition at line 309 of file elf.h.

Referenced by `elf_setup_stack()`.

#### 3.9.2.2 `union { ... } Elf32_auxv_t::a_un`

Referenced by `elf_setup_stack()`.

#### 3.9.2.3 `int32_t Elf32_auxv_t::a_val`

Definition at line 311 of file `elf.h`.

Referenced by `elf_setup_stack()`.

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

- `include/jinue-common/elf.h`

## 3.10 Elf32\_Ehdr Struct Reference

```
#include <jinue-common/elf.h>
```

### Data Fields

- unsigned char **e\_ident** [**EI\_NIDENT**]
- **Elf32\_Half e\_type**
- **Elf32\_Half e\_machine**
- **Elf32\_Word e\_version**
- **Elf32\_Addr e\_entry**
- **Elf32\_Off e\_phoff**
- **Elf32\_Off e\_shoff**
- **Elf32\_Word e\_flags**
- **Elf32\_Half e\_ehsize**
- **Elf32\_Half e\_phentsize**
- **Elf32\_Half e\_phnum**
- **Elf32\_Half e\_shentsize**
- **Elf32\_Half e\_shnum**
- **Elf32\_Half e\_shstrndx**

### 3.10.1 Detailed Description

Definition at line 258 of file `elf.h`.

### 3.10.2 Field Documentation

#### 3.10.2.1 `Elf32_Half Elf32_Ehdr::e_ehsize`

Definition at line 267 of file `elf.h`.

### 3.10.2.2 Elf32\_Addr Elf32\_Ehdr::e\_entry

Definition at line 263 of file elf.h.

Referenced by elf\_check(), and elf\_load().

### 3.10.2.3 Elf32\_Word Elf32\_Ehdr::e\_flags

Definition at line 266 of file elf.h.

Referenced by elf\_check().

### 3.10.2.4 unsigned char Elf32\_Ehdr::e\_ident[EI\_NIDENT]

Definition at line 259 of file elf.h.

Referenced by elf\_check().

### 3.10.2.5 Elf32\_Half Elf32\_Ehdr::e\_machine

Definition at line 261 of file elf.h.

Referenced by elf\_check().

### 3.10.2.6 Elf32\_Half Elf32\_Ehdr::e\_phentsize

Definition at line 268 of file elf.h.

Referenced by elf\_check(), and elf\_load().

### 3.10.2.7 Elf32\_Half Elf32\_Ehdr::e\_phnum

Definition at line 269 of file elf.h.

Referenced by elf\_check(), and elf\_load().

### 3.10.2.8 Elf32\_Off Elf32\_Ehdr::e\_phoff

Definition at line 264 of file elf.h.

Referenced by elf\_check(), and elf\_load().

### 3.10.2.9 Elf32\_Half Elf32\_Ehdr::e\_shentsize

Definition at line 270 of file elf.h.

### 3.10.2.10 Elf32\_Half Elf32\_Ehdr::e\_shnum

Definition at line 271 of file elf.h.

Referenced by elf\_lookup\_symbol().

#### 3.10.2.11 Elf32\_Off Elf32\_Ehdr::e\_shoff

Definition at line 265 of file elf.h.

#### 3.10.2.12 Elf32\_Half Elf32\_Ehdr::e\_shstrndx

Definition at line 272 of file elf.h.

#### 3.10.2.13 Elf32\_Half Elf32\_Ehdr::e\_type

Definition at line 260 of file elf.h.

Referenced by elf\_check().

#### 3.10.2.14 Elf32\_Word Elf32\_Ehdr::e\_version

Definition at line 262 of file elf.h.

Referenced by elf\_check().

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

- include/jinue-common/elf.h

### 3.11 Elf32\_Phdr Struct Reference

```
#include <jinue-common/elf.h>
```

#### Data Fields

- Elf32\_Word p\_type
- Elf32\_Off p\_offset
- Elf32\_Addr p\_vaddr
- Elf32\_Addr p\_paddr
- Elf32\_Word p\_filesz
- Elf32\_Word p\_memsz
- Elf32\_Word p\_flags
- Elf32\_Word p\_align

#### 3.11.1 Detailed Description

Definition at line 275 of file elf.h.

#### 3.11.2 Field Documentation

##### 3.11.2.1 Elf32\_Word Elf32\_Phdr::p\_align

Definition at line 283 of file elf.h.

### 3.11.2.2 Elf32\_Word Elf32\_Phdr::p\_filesz

Definition at line 280 of file elf.h.

Referenced by elf\_load().

### 3.11.2.3 Elf32\_Word Elf32\_Phdr::p\_flags

Definition at line 282 of file elf.h.

### 3.11.2.4 Elf32\_Word Elf32\_Phdr::p\_memsz

Definition at line 281 of file elf.h.

Referenced by elf\_load().

### 3.11.2.5 Elf32\_Off Elf32\_Phdr::p\_offset

Definition at line 277 of file elf.h.

### 3.11.2.6 Elf32\_Addr Elf32\_Phdr::p\_paddr

Definition at line 279 of file elf.h.

### 3.11.2.7 Elf32\_Word Elf32\_Phdr::p\_type

Definition at line 276 of file elf.h.

### 3.11.2.8 Elf32\_Addr Elf32\_Phdr::p\_vaddr

Definition at line 278 of file elf.h.

Referenced by elf\_load().

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

- include/jinue-common/elf.h

## 3.12 Elf32\_Shdr Struct Reference

```
#include <jinue-common/elf.h>
```

### Data Fields

- Elf32\_Word sh\_name
- Elf32\_Word sh\_type
- Elf32\_Word sh\_flags
- Elf32\_Addr sh\_addr

- **Elf32\_Off sh\_offset**
- **Elf32\_Word sh\_size**
- **Elf32\_Word sh\_link**
- **Elf32\_Word sh\_info**
- **Elf32\_Word sh\_addralign**
- **Elf32\_Word sh\_entsize**

### 3.12.1 Detailed Description

Definition at line 286 of file elf.h.

### 3.12.2 Field Documentation

#### 3.12.2.1 Elf32\_Addr Elf32\_Shdr::sh\_addr

Definition at line 290 of file elf.h.

#### 3.12.2.2 Elf32\_Word Elf32\_Shdr::sh\_addralign

Definition at line 295 of file elf.h.

#### 3.12.2.3 Elf32\_Word Elf32\_Shdr::sh\_entsize

Definition at line 296 of file elf.h.

Referenced by elf\_lookup\_symbol().

#### 3.12.2.4 Elf32\_Word Elf32\_Shdr::sh\_flags

Definition at line 289 of file elf.h.

#### 3.12.2.5 Elf32\_Word Elf32\_Shdr::sh\_info

Definition at line 294 of file elf.h.

#### 3.12.2.6 Elf32\_Word Elf32\_Shdr::sh\_link

Definition at line 293 of file elf.h.

Referenced by elf\_lookup\_symbol().

#### 3.12.2.7 Elf32\_Word Elf32\_Shdr::sh\_name

Definition at line 287 of file elf.h.



#### 3.12.2.8 Elf32\_Off Elf32\_Shdr::sh\_offset

Definition at line 291 of file elf.h.

Referenced by elf\_lookup\_symbol().

#### 3.12.2.9 Elf32\_Word Elf32\_Shdr::sh\_size

Definition at line 292 of file elf.h.

Referenced by elf\_lookup\_symbol().

#### 3.12.2.10 Elf32\_Word Elf32\_Shdr::sh\_type

Definition at line 288 of file elf.h.

Referenced by elf\_lookup\_symbol().

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

- include/jinue-common/elf.h

### 3.13 Elf32\_Sym Struct Reference

```
#include <jinue-common/elf.h>
```

#### Data Fields

- Elf32\_Word **st\_name**
- Elf32\_Addr **st\_value**
- Elf32\_Word **st\_size**
- unsigned char **st\_info**
- unsigned char **st\_other**
- Elf32\_Half **st\_shndx**

#### 3.13.1 Detailed Description

Definition at line 299 of file elf.h.

#### 3.13.2 Field Documentation

##### 3.13.2.1 unsigned char Elf32\_Sym::st\_info

Definition at line 303 of file elf.h.

Referenced by elf\_lookup\_symbol().

### 3.13.2.2 Elf32\_Word Elf32\_Sym::st\_name

Definition at line 300 of file elf.h.

Referenced by elf\_lookup\_symbol().

### 3.13.2.3 unsigned char Elf32\_Sym::st\_other

Definition at line 304 of file elf.h.

### 3.13.2.4 Elf32\_Half Elf32\_Sym::st\_shndx

Definition at line 305 of file elf.h.

### 3.13.2.5 Elf32\_Word Elf32\_Sym::st\_size

Definition at line 302 of file elf.h.

Referenced by elf\_lookup\_symbol().

### 3.13.2.6 Elf32\_Addr Elf32\_Sym::st\_value

Definition at line 301 of file elf.h.

Referenced by elf\_lookup\_symbol().

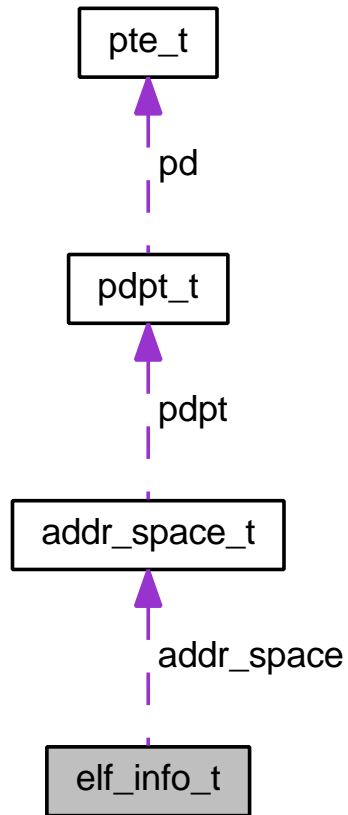
The documentation for this struct was generated from the following file:

- include/jinue-common/elf.h

## 3.14 elf\_info\_t Struct Reference

```
#include <jinue-common/elf.h>
```

Collaboration diagram for elf\_info\_t:



### Data Fields

- `addr_t entry`
- `addr_t stack_addr`
- `addr_t at_phdr`
- `int at_phent`
- `int at_phnum`
- `addr_space_t * addr_space`

### 3.14.1 Detailed Description

Definition at line 39 of file `elf.h`.

### 3.14.2 Field Documentation

#### 3.14.2.1 `addr_space_t * elf_info_t::addr_space`

Definition at line 45 of file `elf.h`.

Referenced by `elf_load()`, and `elf_setup_stack()`.

#### 3.14.2.2 `addr_t elf_info_t::at_phdr`

Definition at line 42 of file elf.h.

Referenced by `elf_load()`, and `elf_setup_stack()`.

#### 3.14.2.3 `int elf_info_t::at_phent`

Definition at line 43 of file elf.h.

Referenced by `elf_load()`, and `elf_setup_stack()`.

#### 3.14.2.4 `int elf_info_t::at_phnum`

Definition at line 44 of file elf.h.

Referenced by `elf_load()`, and `elf_setup_stack()`.

#### 3.14.2.5 `addr_t elf_info_t::entry`

Definition at line 40 of file elf.h.

Referenced by `elf_load()`, `elf_setup_stack()`, and `kmain()`.

#### 3.14.2.6 `addr_t elf_info_t::stack_addr`

Definition at line 41 of file elf.h.

Referenced by `elf_setup_stack()`, and `kmain()`.

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

- `include/jinue-common/elf.h`

### 3.15 `elf_symbol_t` Struct Reference

```
#include <jinue-common/elf.h>
```

#### Data Fields

- **Elf32\_Addr** `addr`
- `const char *` **name**

#### 3.15.1 Detailed Description

Definition at line 48 of file elf.h.

### 3.15.2 Field Documentation

#### 3.15.2.1 Elf32\_Addr elf\_symbol\_t::addr

Definition at line 49 of file elf.h.

Referenced by dump\_call\_stack(), and elf\_lookup\_symbol().

#### 3.15.2.2 const char\* elf\_symbol\_t::name

Definition at line 50 of file elf.h.

Referenced by dump\_call\_stack(), and elf\_lookup\_symbol().

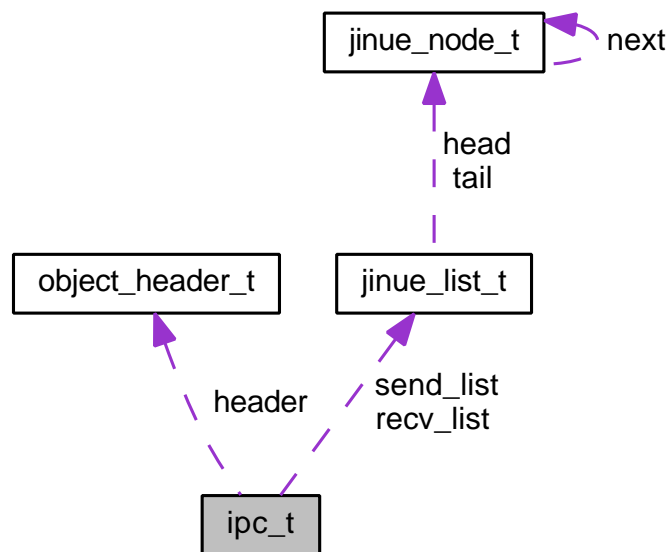
The documentation for this struct was generated from the following file:

- include/jinue-common/elf.h

## 3.16 ipc\_t Struct Reference

```
#include <types.h>
```

Collaboration diagram for ipc\_t:



### Data Fields

- `object_header_t header`
- `jinue_list_t send_list`
- `jinue_list_t recv_list`

### 3.16.1 Detailed Description

Definition at line 100 of file types.h.

### 3.16.2 Field Documentation

#### 3.16.2.1 `object_header_t ipc_t::header`

Definition at line 101 of file `types.h`.

Referenced by `dispatch_syscall()`, and `ipc_object_create()`.

#### 3.16.2.2 `jinue_list_t ipc_t::recv_list`

Definition at line 103 of file `types.h`.

Referenced by `ipc_receive()`, and `ipc_send()`.

#### 3.16.2.3 `jinue_list_t ipc_t::send_list`

Definition at line 102 of file `types.h`.

Referenced by `ipc_receive()`, and `ipc_send()`.

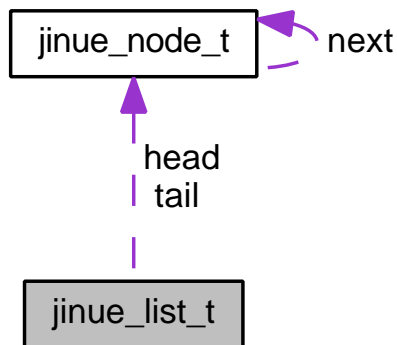
The documentation for this struct was generated from the following file:

- `include/types.h`

## 3.17 `jinue_list_t` Struct Reference

```
#include <jinue-common/list.h>
```

Collaboration diagram for `jinue_list_t`:



### Data Fields

- `jinue_node_t * head`
- `jinue_node_t * tail`

#### 3.17.1 Detailed Description

Definition at line 55 of file `list.h`.

### 3.17.2 Field Documentation

#### 3.17.2.1 jinue\_node\_t\* jinue\_list\_t::head

Definition at line 56 of file list.h.

#### 3.17.2.2 jinue\_node\_t\* jinue\_list\_t::tail

Definition at line 57 of file list.h.

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

- include/jinue-common/list.h

## 3.18 jinue\_mem\_entry\_t Struct Reference

```
#include <jinue-common/types.h>
```

### Data Fields

- **uint64\_t** addr
- **uint64\_t** size
- **uint32\_t** type

### 3.18.1 Detailed Description

Definition at line 38 of file types.h.

### 3.18.2 Field Documentation

#### 3.18.2.1 uint64\_t jinue\_mem\_entry\_t::addr

Definition at line 39 of file types.h.

Referenced by dispatch\_syscall().

#### 3.18.2.2 uint64\_t jinue\_mem\_entry\_t::size

Definition at line 40 of file types.h.

Referenced by dispatch\_syscall().

#### 3.18.2.3 uint32\_t jinue\_mem\_entry\_t::type

Definition at line 41 of file types.h.

Referenced by dispatch\_syscall().

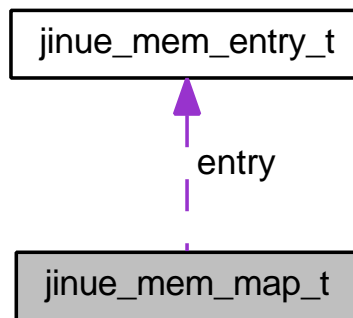
The documentation for this struct was generated from the following file:

- `include/jinue-common/types.h`

### 3.19 jinue\_mem\_map\_t Struct Reference

```
#include <jinue-common/types.h>
```

Collaboration diagram for `jinue_mem_map_t`:



#### Data Fields

- `uint32_t num_entries`
- `jinue_mem_entry_t entry []`

#### 3.19.1 Detailed Description

Definition at line 44 of file `types.h`.

#### 3.19.2 Field Documentation

##### 3.19.2.1 `jinue_mem_entry_t jinue_mem_map_t::entry[]`

Definition at line 46 of file `types.h`.

Referenced by `dispatch_syscall()`.

##### 3.19.2.2 `uint32_t jinue_mem_map_t::num_entries`

Definition at line 45 of file `types.h`.

Referenced by `dispatch_syscall()`.

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

- `include/jinue-common/types.h`



## 3.20 jinue\_message\_t Struct Reference

```
#include <jinue/ipc.h>
```

### Data Fields

- **uintptr\_t** function
- **uintptr\_t** cookie
- **size\_t** buffer\_size
- **size\_t** data\_size
- **size\_t** desc\_n

### 3.20.1 Detailed Description

Definition at line 38 of file ipc.h.

### 3.20.2 Field Documentation

#### 3.20.2.1 size\_t jinue\_message\_t::buffer\_size

Definition at line 41 of file ipc.h.

#### 3.20.2.2 uintptr\_t jinue\_message\_t::cookie

Definition at line 40 of file ipc.h.

#### 3.20.2.3 size\_t jinue\_message\_t::data\_size

Definition at line 42 of file ipc.h.

#### 3.20.2.4 size\_t jinue\_message\_t::desc\_n

Definition at line 43 of file ipc.h.

#### 3.20.2.5 uintptr\_t jinue\_message\_t::function

Definition at line 39 of file ipc.h.

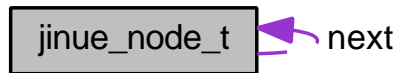
The documentation for this struct was generated from the following file:

- include/jinue/**ipc.h**

## 3.21 jinue\_node\_t Struct Reference

```
#include <jinue-common/list.h>
```

Collaboration diagram for `jinue_node_t`:



## Data Fields

- struct `jinue_node_t` \* `next`

### 3.21.1 Detailed Description

Definition at line 38 of file `list.h`.

### 3.21.2 Field Documentation

#### 3.21.2.1 struct `jinue_node_t`\* `jinue_node_t::next`

Definition at line 39 of file `list.h`.

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

- `include/jinue-common/list.h`

## 3.22 `jinue_reply_t` Struct Reference

```
#include <jinue/ipc.h>
```

## Data Fields

- `size_t` `data_size`
- `size_t` `desc_n`

### 3.22.1 Detailed Description

Definition at line 46 of file `ipc.h`.

### 3.22.2 Field Documentation

#### 3.22.2.1 `size_t` `jinue_reply_t::data_size`

Definition at line 47 of file `ipc.h`.

### 3.22.2.2 size\_t jinue\_reply\_t::desc\_n

Definition at line 48 of file ipc.h.

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

- include/jinue/ipc.h

## 3.23 jinue\_syscall\_args\_t Struct Reference

```
#include <jinue-common/syscall.h>
```

### Data Fields

- uintptr\_t arg0
- uintptr\_t arg1
- uintptr\_t arg2
- uintptr\_t arg3

### 3.23.1 Detailed Description

Definition at line 39 of file syscall.h.

### 3.23.2 Field Documentation

#### 3.23.2.1 uintptr\_t jinue\_syscall\_args\_t::arg0

Definition at line 40 of file syscall.h.

Referenced by dispatch\_syscall(), ipc\_receive(), and ipc\_send().

#### 3.23.2.2 uintptr\_t jinue\_syscall\_args\_t::arg1

Definition at line 41 of file syscall.h.

Referenced by dispatch\_syscall(), ipc\_receive(), and ipc\_send().

#### 3.23.2.3 uintptr\_t jinue\_syscall\_args\_t::arg2

Definition at line 42 of file syscall.h.

Referenced by dispatch\_syscall(), ipc\_receive(), ipc\_reply(), and ipc\_send().

#### 3.23.2.4 uintptr\_t jinue\_syscall\_args\_t::arg3

Definition at line 43 of file syscall.h.

Referenced by dispatch\_syscall(), ipc\_receive(), and ipc\_reply().

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

- `include/jinue-common/syscall.h`

## 3.24 `kernel_context_t` Struct Reference

```
#include <hal/types.h>
```

### Data Fields

- `uint32_t edi`
- `uint32_t esi`
- `uint32_t ebx`
- `uint32_t ebp`
- `uint32_t eip`

### 3.24.1 Detailed Description

Definition at line 219 of file `types.h`.

### 3.24.2 Field Documentation

#### 3.24.2.1 `uint32_t kernel_context_t::ebp`

Definition at line 223 of file `types.h`.

#### 3.24.2.2 `uint32_t kernel_context_t::ebx`

Definition at line 222 of file `types.h`.

#### 3.24.2.3 `uint32_t kernel_context_t::edi`

Definition at line 220 of file `types.h`.

#### 3.24.2.4 `uint32_t kernel_context_t::eip`

Definition at line 224 of file `types.h`.

Referenced by `thread_page_init()`.

#### 3.24.2.5 `uint32_t kernel_context_t::esi`

Definition at line 221 of file `types.h`.

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

- `include/hal/types.h`

## 3.25 message\_info\_t Struct Reference

```
#include <types.h>
```

### Data Fields

- **uintptr\_t** function
- **uintptr\_t** cookie
- **size\_t** buffer\_size
- **size\_t** data\_size
- **size\_t** desc\_n
- **size\_t** total\_size

### 3.25.1 Detailed Description

Definition at line 78 of file types.h.

### 3.25.2 Field Documentation

#### 3.25.2.1 **size\_t** message\_info\_t::buffer\_size

Definition at line 81 of file types.h.

Referenced by ipc\_reply(), and ipc\_send().

#### 3.25.2.2 **uintptr\_t** message\_info\_t::cookie

Definition at line 80 of file types.h.

Referenced by ipc\_send().

#### 3.25.2.3 **size\_t** message\_info\_t::data\_size

Definition at line 82 of file types.h.

Referenced by ipc\_receive(), ipc\_reply(), and ipc\_send().

#### 3.25.2.4 **size\_t** message\_info\_t::desc\_n

Definition at line 83 of file types.h.

Referenced by ipc\_reply(), and ipc\_send().

#### 3.25.2.5 **uintptr\_t** message\_info\_t::function

Definition at line 79 of file types.h.

Referenced by ipc\_send().

### 3.25.2.6 `size_t message_info_t::total_size`

Definition at line 84 of file `types.h`.

Referenced by `ipc_receive()`, and `ipc_send()`.

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

- `include/types.h`

## 3.26 `object_header_t` Struct Reference

```
#include <types.h>
```

### Data Fields

- `int type`
- `int ref_count`
- `int flags`

### 3.26.1 Detailed Description

Definition at line 58 of file `types.h`.

### 3.26.2 Field Documentation

#### 3.26.2.1 `int object_header_t::flags`

Definition at line 61 of file `types.h`.

Referenced by `ipc_object_create()`, `ipc_receive()`, and `ipc_send()`.

#### 3.26.2.2 `int object_header_t::ref_count`

Definition at line 60 of file `types.h`.

#### 3.26.2.3 `int object_header_t::type`

Definition at line 59 of file `types.h`.

Referenced by `ipc_receive()`, and `ipc_send()`.

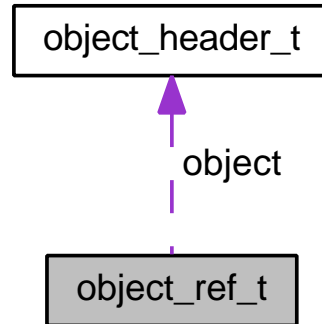
The documentation for this struct was generated from the following file:

- `include/types.h`

## 3.27 object\_ref\_t Struct Reference

```
#include <types.h>
```

Collaboration diagram for object\_ref\_t:



### Data Fields

- `object_header_t * object`
- `uintptr_t flags`
- `uintptr_t cookie`

#### 3.27.1 Detailed Description

Definition at line 64 of file `types.h`.

#### 3.27.2 Field Documentation

##### 3.27.2.1 `uintptr_t object_ref_t::cookie`

Definition at line 67 of file `types.h`.

Referenced by `dispatch_syscall()`.

##### 3.27.2.2 `uintptr_t object_ref_t::flags`

Definition at line 66 of file `types.h`.

Referenced by `dispatch_syscall()`.

##### 3.27.2.3 `object_header_t * object_ref_t::object`

Definition at line 65 of file `types.h`.

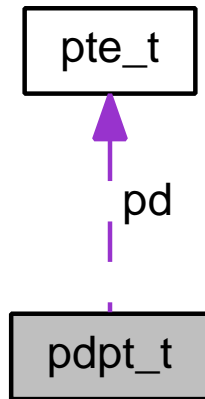
Referenced by `dispatch_syscall()`.

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

- `include/types.h`

### 3.28 pdpt\_t Struct Reference

Collaboration diagram for pdpt\_t:



#### Data Fields

- **pte\_t** **pd** [PDPT\_ENTRIES]

#### 3.28.1 Detailed Description

Definition at line 56 of file vm\_pae.c.

#### 3.28.2 Field Documentation

##### 3.28.2.1 pte\_t pdpt\_t::pd[PDPT\_ENTRIES]

Definition at line 57 of file vm\_pae.c.

Referenced by vm\_pae\_create\_addr\_space(), vm\_pae\_create\_initial\_addr\_space(), vm\_pae\_destroy\_addr\_space(), vm\_pae\_lookup\_page\_directory(), and vm\_pae\_unmap\_low\_alias().

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

- kernel/hal/**vm\_pae.c**

### 3.29 pfalloc\_cache\_t Struct Reference

```
#include <pfalloc.h>
```

#### Data Fields

- **kern\_paddr\_t** \* **ptr**
- **uint32\_t** **count**



### 3.29.1 Detailed Description

Definition at line 42 of file pfalloc.h.

### 3.29.2 Field Documentation

#### 3.29.2.1 uint32\_t pfalloc\_cache\_t::count

Definition at line 44 of file pfalloc.h.

Referenced by `init_pfalloc_cache()`, `pfalloc_from()`, and `pf_free_to()`.

#### 3.29.2.2 kern\_paddr\_t\* pfalloc\_cache\_t::ptr

Definition at line 43 of file pfalloc.h.

Referenced by `init_pfalloc_cache()`, `pfalloc_from()`, and `pf_free_to()`.

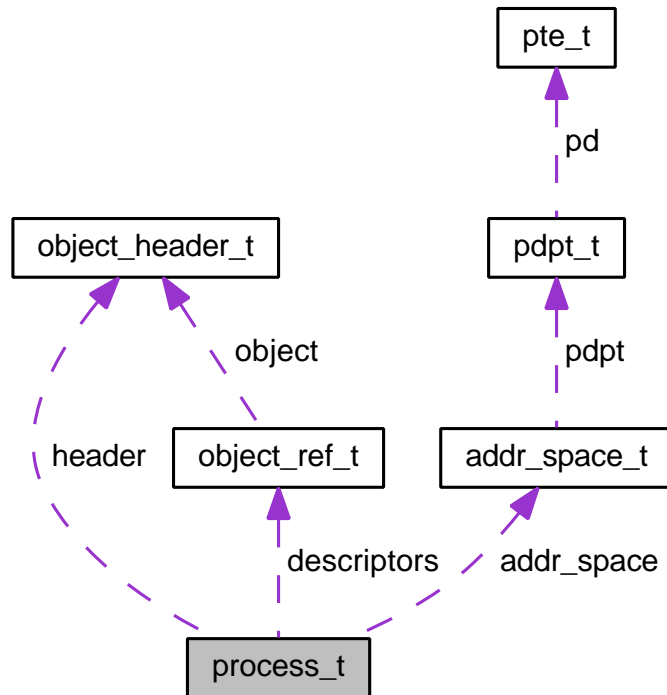
The documentation for this struct was generated from the following file:

- `include/pfalloc.h`

## 3.30 process\_t Struct Reference

```
#include <types.h>
```

Collaboration diagram for `process_t`:



## Data Fields

- **object\_header\_t** header
- **addr\_space\_t** addr\_space
- **object\_ref\_t** descriptors [PROCESS\_MAX\_DESCRIPTOR]

### 3.30.1 Detailed Description

Definition at line 72 of file types.h.

### 3.30.2 Field Documentation

#### 3.30.2.1 **addr\_space\_t** process\_t::addr\_space

Definition at line 74 of file types.h.

Referenced by kmain(), process\_create(), process\_create\_initial(), and thread\_switch().

#### 3.30.2.2 **object\_ref\_t** process\_t::descriptors[PROCESS\_MAX\_DESCRIPTOR]

Definition at line 75 of file types.h.

Referenced by process\_get\_descriptor().

#### 3.30.2.3 **object\_header\_t** process\_t::header

Definition at line 73 of file types.h.

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

- include/types.h

## 3.31 pseudo\_descriptor\_t Struct Reference

```
#include <hal/types.h>
```

## Data Fields

- **uint16\_t** padding
- **uint16\_t** limit
- **addr\_t** addr

### 3.31.1 Detailed Description

Definition at line 119 of file types.h.

### 3.31.2 Field Documentation

#### 3.31.2.1 addr\_t pseudo\_descriptor\_t::addr

Definition at line 122 of file types.h.

#### 3.31.2.2 uint16\_t pseudo\_descriptor\_t::limit

Definition at line 121 of file types.h.

#### 3.31.2.3 uint16\_t pseudo\_descriptor\_t::padding

Definition at line 120 of file types.h.

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

- include/hal/types.h

## 3.32 pte\_t Struct Reference

### Data Fields

- uint64\_t entry
- uint32\_t entry

### 3.32.1 Detailed Description

Definition at line 52 of file vm\_pae.c.

### 3.32.2 Field Documentation

#### 3.32.2.1 uint32\_t pte\_t::entry

Definition at line 38 of file vm\_x86.c.

#### 3.32.2.2 uint64\_t pte\_t::entry

Definition at line 53 of file vm\_pae.c.

Referenced by vm\_pae\_clear\_pte(), vm\_pae\_copy\_pte(), vm\_pae\_destroy\_addr\_space(), vm\_pae\_get\_pte\_flags(), vm\_pae\_get\_pte\_paddr(), vm\_pae\_set\_pte(), vm\_pae\_set\_pte\_flags(), vm\_x86\_clear\_pte(), vm\_x86\_copy\_pte(), vm\_x86\_get\_pte\_flags(), vm\_x86\_get\_pte\_paddr(), vm\_x86\_set\_pte(), and vm\_x86\_set\_pte\_flags().

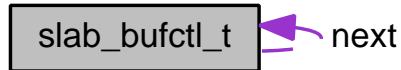
The documentation for this struct was generated from the following files:

- kernel/hal/vm\_pae.c
- kernel/hal/vm\_x86.c

### 3.33 slab\_bufctl\_t Struct Reference

```
#include <slab.h>
```

Collaboration diagram for slab\_bufctl\_t:



#### Data Fields

- struct **slab\_bufctl\_t** \* **next**

#### 3.33.1 Detailed Description

Definition at line 84 of file slab.h.

#### 3.33.2 Field Documentation

3.33.2.1 struct **slab\_bufctl\_t**\* slab\_bufctl\_t::next

Definition at line 85 of file slab.h.

Referenced by slab\_cache\_alloc(), and slab\_cache\_free().

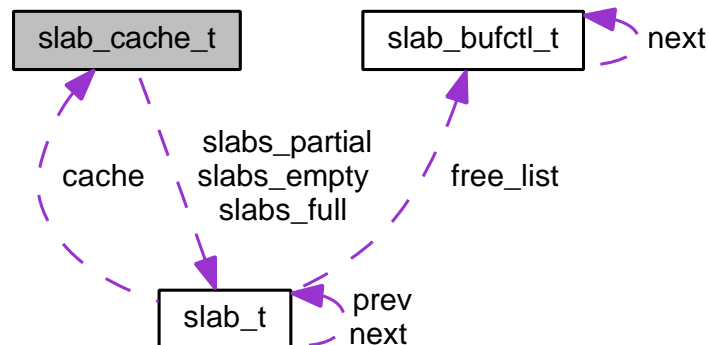
The documentation for this struct was generated from the following file:

- include/**slab.h**

### 3.34 slab\_cache\_t Struct Reference

```
#include <slab.h>
```

Collaboration diagram for slab\_cache\_t:



## Data Fields

- struct **slab\_t** \* **slabs\_empty**
- struct **slab\_t** \* **slabs\_partial**
- struct **slab\_t** \* **slabs\_full**
- unsigned int **empty\_count**
- **size\_t** **obj\_size**
- **size\_t** **alloc\_size**
- **size\_t** **alignment**
- **size\_t** **bufctl\_offset**
- **size\_t** **next\_colour**
- **size\_t** **max\_colour**
- unsigned int **working\_set**
- **slab\_ctor\_t** **ctor**
- **slab\_ctor\_t** **dtor**
- char \* **name**
- int **flags**

### 3.34.1 Detailed Description

Definition at line 64 of file slab.h.

### 3.34.2 Field Documentation

#### 3.34.2.1 **size\_t** slab\_cache\_t::alignment

Definition at line 71 of file slab.h.

Referenced by slab\_cache\_init().

#### 3.34.2.2 **size\_t** slab\_cache\_t::alloc\_size

Definition at line 70 of file slab.h.

Referenced by slab\_cache\_init().

#### 3.34.2.3 **size\_t** slab\_cache\_t::bufctl\_offset

Definition at line 72 of file slab.h.

Referenced by slab\_cache\_alloc(), slab\_cache\_free(), and slab\_cache\_init().

#### 3.34.2.4 **slab\_ctor\_t** slab\_cache\_t::ctor

Definition at line 76 of file slab.h.

Referenced by slab\_cache\_alloc(), and slab\_cache\_init().

**3.34.2.5 slab\_ctor\_t slab\_cache\_t::ctor**

Definition at line 77 of file slab.h.

Referenced by slab\_cache\_free(), and slab\_cache\_init().

**3.34.2.6 unsigned int slab\_cache\_t::empty\_count**

Definition at line 68 of file slab.h.

Referenced by slab\_cache\_alloc(), slab\_cache\_free(), slab\_cache\_init(), and slab\_cache\_reap().

**3.34.2.7 int slab\_cache\_t::flags**

Definition at line 79 of file slab.h.

Referenced by slab\_cache\_alloc(), slab\_cache\_free(), and slab\_cache\_init().

**3.34.2.8 size\_t slab\_cache\_t::max\_colour**

Definition at line 74 of file slab.h.

Referenced by slab\_cache\_init().

**3.34.2.9 char\* slab\_cache\_t::name**

Definition at line 78 of file slab.h.

Referenced by slab\_cache\_alloc(), slab\_cache\_free(), and slab\_cache\_init().

**3.34.2.10 size\_t slab\_cache\_t::next\_colour**

Definition at line 73 of file slab.h.

Referenced by slab\_cache\_init().

**3.34.2.11 size\_t slab\_cache\_t::obj\_size**

Definition at line 69 of file slab.h.

Referenced by slab\_cache\_alloc(), slab\_cache\_free(), and slab\_cache\_init().

**3.34.2.12 struct slab\_t\* slab\_cache\_t::slabs\_empty**

Definition at line 65 of file slab.h.

Referenced by slab\_cache\_alloc(), slab\_cache\_free(), slab\_cache\_init(), and slab\_cache\_reap().

**3.34.2.13 struct slab\_t\* slab\_cache\_t::slabs\_full**

Definition at line 67 of file slab.h.

Referenced by slab\_cache\_alloc(), slab\_cache\_free(), and slab\_cache\_init().

## 3.34.2.14 struct slab\_t\* slab\_cache\_t::slabs\_partial

Definition at line 66 of file slab.h.

Referenced by slab\_cache\_alloc(), slab\_cache\_free(), and slab\_cache\_init().

## 3.34.2.15 unsigned int slab\_cache\_t::working\_set

Definition at line 75 of file slab.h.

Referenced by slab\_cache\_init(), slab\_cache\_reap(), and slab\_cache\_set\_working\_set().

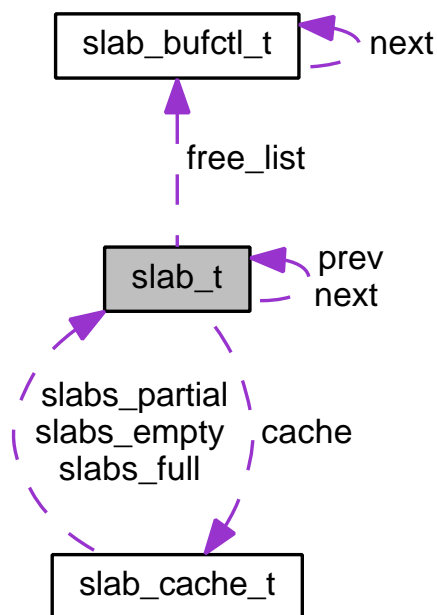
The documentation for this struct was generated from the following file:

- include/slab.h

## 3.35 slab\_t Struct Reference

```
#include <slab.h>
```

Collaboration diagram for slab\_t:



## Data Fields

- struct **slab\_t** \* **prev**
- struct **slab\_t** \* **next**
- **slab\_cache\_t** \* **cache**
- unsigned int **obj\_count**
- **size\_t** **colour**
- **slab\_bufctl\_t** \* **free\_list**

### 3.35.1 Detailed Description

Definition at line 90 of file slab.h.

### 3.35.2 Field Documentation

#### 3.35.2.1 `slab_cache_t* slab_t::cache`

Definition at line 94 of file slab.h.

Referenced by `slab_cache_free()`.

#### 3.35.2.2 `size_t slab_t::colour`

Definition at line 97 of file slab.h.

#### 3.35.2.3 `slab_bufctl_t* slab_t::free_list`

Definition at line 98 of file slab.h.

Referenced by `slab_cache_alloc()`, and `slab_cache_free()`.

#### 3.35.2.4 `struct slab_t* slab_t::next`

Definition at line 92 of file slab.h.

Referenced by `slab_cache_alloc()`, `slab_cache_free()`, and `slab_cache_reap()`.

#### 3.35.2.5 `unsigned int slab_t::obj_count`

Definition at line 96 of file slab.h.

Referenced by `slab_cache_alloc()`, and `slab_cache_free()`.

#### 3.35.2.6 `struct slab_t* slab_t::prev`

Definition at line 91 of file slab.h.

Referenced by `slab_cache_alloc()`, and `slab_cache_free()`.

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

- `include/slab.h`

## 3.36 `thread_context_t` Struct Reference

```
#include <hal/types.h>
```



## Data Fields

- **addr\_t** saved\_stack\_pointer
- **addr\_t** local\_storage\_addr
- **size\_t** local\_storage\_size

### 3.36.1 Detailed Description

Definition at line 72 of file types.h.

### 3.36.2 Field Documentation

#### 3.36.2.1 **addr\_t** thread\_context\_t::local\_storage\_addr

Definition at line 76 of file types.h.

Referenced by thread\_page\_init().

#### 3.36.2.2 **size\_t** thread\_context\_t::local\_storage\_size

Definition at line 77 of file types.h.

#### 3.36.2.3 **addr\_t** thread\_context\_t::saved\_stack\_pointer

Definition at line 75 of file types.h.

Referenced by thread\_page\_init().

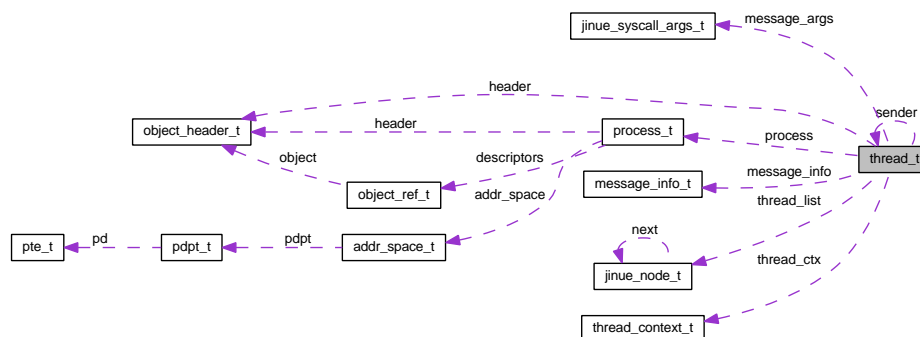
The documentation for this struct was generated from the following file:

- include/hal/**types.h**

## 3.37 thread\_t Struct Reference

```
#include <types.h>
```

Collaboration diagram for thread\_t:



## Data Fields

- **object\_header\_t** header
- **thread\_context\_t** thread\_ctx
- **jinue\_node\_t** thread\_list
- **process\_t** \* process
- struct **thread\_t** \* sender
- **jinue\_syscall\_args\_t** \* message\_args
- **message\_info\_t** message\_info
- char **message\_buffer** [JINUE\_SEND\_MAX\_SIZE]

### 3.37.1 Detailed Description

Definition at line 87 of file types.h.

### 3.37.2 Field Documentation

#### 3.37.2.1 **object\_header\_t** thread\_t::header

Definition at line 88 of file types.h.

Referenced by ipc\_receive(), ipc\_reply(), and ipc\_send().

#### 3.37.2.2 **jinue\_syscall\_args\_t**\* thread\_t::message\_args

Definition at line 93 of file types.h.

Referenced by ipc\_receive(), ipc\_reply(), and ipc\_send().

#### 3.37.2.3 **char** thread\_t::message\_buffer[JINUE\_SEND\_MAX\_SIZE]

Definition at line 95 of file types.h.

Referenced by ipc\_receive(), ipc\_reply(), and ipc\_send().

#### 3.37.2.4 **message\_info\_t** thread\_t::message\_info

Definition at line 94 of file types.h.

Referenced by ipc\_receive(), ipc\_reply(), and ipc\_send().

#### 3.37.2.5 **process\_t**\* thread\_t::process

Definition at line 91 of file types.h.

Referenced by dispatch\_syscall(), ipc\_receive(), ipc\_send(), and thread\_switch().

#### 3.37.2.6 **struct thread\_t**\* thread\_t::sender

Definition at line 92 of file types.h.

Referenced by ipc\_receive(), ipc\_reply(), and ipc\_send().

### 3.37.2.7 thread\_context\_t thread\_t::thread\_ctx

Definition at line 89 of file types.h.

Referenced by thread\_page\_init(), and thread\_switch().

### 3.37.2.8 jinue\_node\_t thread\_t::thread\_list

Definition at line 90 of file types.h.

Referenced by ipc\_receive(), ipc\_send(), and thread\_ready().

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

- include/types.h

## 3.38 trapframe\_t Struct Reference

```
#include <hal/types.h>
```

### Data Fields

- uint32\_t eax
- uint32\_t ebx
- uint32\_t esi
- uint32\_t edi
- uint32\_t edx
- uint32\_t ecx
- uint32\_t ds
- uint32\_t es
- uint32\_t fs
- uint32\_t gs
- uint32\_t errcode
- uint32\_t ivt
- uint32\_t ebp
- uint32\_t eip
- uint32\_t cs
- uint32\_t eflags
- uint32\_t esp
- uint32\_t ss

### 3.38.1 Detailed Description

Definition at line 193 of file types.h.

### 3.38.2 Field Documentation

#### 3.38.2.1 `uint32_t trapframe_t::cs`

Definition at line 213 of file types.h.

Referenced by `thread_page_init()`.

#### 3.38.2.2 `uint32_t trapframe_t::ds`

Definition at line 205 of file types.h.

Referenced by `thread_page_init()`.

#### 3.38.2.3 `uint32_t trapframe_t::eax`

Definition at line 196 of file types.h.

#### 3.38.2.4 `uint32_t trapframe_t::ebp`

Definition at line 211 of file types.h.

#### 3.38.2.5 `uint32_t trapframe_t::ebx`

Definition at line 198 of file types.h.

#### 3.38.2.6 `uint32_t trapframe_t::ecx`

Definition at line 204 of file types.h.

#### 3.38.2.7 `uint32_t trapframe_t::edi`

Definition at line 202 of file types.h.

#### 3.38.2.8 `uint32_t trapframe_t::edx`

Definition at line 203 of file types.h.

#### 3.38.2.9 `uint32_t trapframe_t::eflags`

Definition at line 214 of file types.h.

Referenced by `thread_page_init()`.

#### 3.38.2.10 `uint32_t trapframe_t::eip`

Definition at line 212 of file types.h.

Referenced by `dispatch_interrupt()`, and `thread_page_init()`.

#### 3.38.2.11 uint32\_t trapframe\_t::errcode

Definition at line 209 of file types.h.

Referenced by dispatch\_interrupt().

#### 3.38.2.12 uint32\_t trapframe\_t::es

Definition at line 206 of file types.h.

Referenced by thread\_page\_init().

#### 3.38.2.13 uint32\_t trapframe\_t::esi

Definition at line 200 of file types.h.

#### 3.38.2.14 uint32\_t trapframe\_t::esp

Definition at line 215 of file types.h.

Referenced by thread\_page\_init().

#### 3.38.2.15 uint32\_t trapframe\_t::fs

Definition at line 207 of file types.h.

Referenced by thread\_page\_init().

#### 3.38.2.16 uint32\_t trapframe\_t::gs

Definition at line 208 of file types.h.

Referenced by thread\_page\_init().

#### 3.38.2.17 uint32\_t trapframe\_t::ivt

Definition at line 210 of file types.h.

Referenced by dispatch\_interrupt().

#### 3.38.2.18 uint32\_t trapframe\_t::ss

Definition at line 216 of file types.h.

Referenced by thread\_page\_init().

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

- include/hal/types.h

### 3.39 tss\_t Struct Reference

```
#include <hal/types.h>
```

#### Data Fields

- `uint16_t prev`
- `addr_t esp0`
- `uint16_t ss0`
- `addr_t esp1`
- `uint16_t ss1`
- `addr_t esp2`
- `uint16_t ss2`
- `uint32_t cr3`
- `uint32_t eip`
- `uint32_t eflags`
- `uint32_t eax`
- `uint32_t ecx`
- `uint32_t edx`
- `uint32_t ebx`
- `uint32_t esp`
- `uint32_t ebp`
- `uint32_t esi`
- `uint32_t edi`
- `uint16_t es`
- `uint16_t cs`
- `uint16_t ss`
- `uint16_t ds`
- `uint16_t fs`
- `uint16_t gs`
- `uint16_t ldt`
- `uint16_t debug`
- `uint16_t iomap`

#### 3.39.1 Detailed Description

Definition at line 125 of file `types.h`.

#### 3.39.2 Field Documentation

##### 3.39.2.1 `uint32_t tss_t::cr3`

Definition at line 141 of file `types.h`.

##### 3.39.2.2 `uint16_t tss_t::cs`

Definition at line 165 of file `types.h`.

#### 3.39.2.3 uint16\_t tss\_t::debug

Definition at line 177 of file types.h.

#### 3.39.2.4 uint16\_t tss\_t::ds

Definition at line 169 of file types.h.

#### 3.39.2.5 uint32\_t tss\_t::eax

Definition at line 147 of file types.h.

#### 3.39.2.6 uint32\_t tss\_t::ebp

Definition at line 157 of file types.h.

#### 3.39.2.7 uint32\_t tss\_t::ebx

Definition at line 153 of file types.h.

#### 3.39.2.8 uint32\_t tss\_t::ecx

Definition at line 149 of file types.h.

#### 3.39.2.9 uint32\_t tss\_t::edi

Definition at line 161 of file types.h.

#### 3.39.2.10 uint32\_t tss\_t::edx

Definition at line 151 of file types.h.

#### 3.39.2.11 uint32\_t tss\_t::eflags

Definition at line 145 of file types.h.

#### 3.39.2.12 uint32\_t tss\_t::eip

Definition at line 143 of file types.h.

#### 3.39.2.13 uint16\_t tss\_t::es

Definition at line 163 of file types.h.

**3.39.2.14 uint32\_t tss\_t::esi**

Definition at line 159 of file types.h.

**3.39.2.15 uint32\_t tss\_t::esp**

Definition at line 155 of file types.h.

**3.39.2.16 addr\_t tss\_t::esp0**

Definition at line 129 of file types.h.

Referenced by `cpu_init_data()`, and `thread_context_switch()`.

**3.39.2.17 addr\_t tss\_t::esp1**

Definition at line 133 of file types.h.

Referenced by `cpu_init_data()`, and `thread_context_switch()`.

**3.39.2.18 addr\_t tss\_t::esp2**

Definition at line 137 of file types.h.

Referenced by `cpu_init_data()`, and `thread_context_switch()`.

**3.39.2.19 uint16\_t tss\_t::fs**

Definition at line 171 of file types.h.

**3.39.2.20 uint16\_t tss\_t::gs**

Definition at line 173 of file types.h.

**3.39.2.21 uint16\_t tss\_t::iomap**

Definition at line 178 of file types.h.

**3.39.2.22 uint16\_t tss\_t::ldt**

Definition at line 175 of file types.h.

**3.39.2.23 uint16\_t tss\_t::prev**

Definition at line 127 of file types.h.



## 3.39.2.24 uint16\_t tss\_t::ss

Definition at line 167 of file types.h.

## 3.39.2.25 uint16\_t tss\_t::ss0

Definition at line 131 of file types.h.

Referenced by cpu\_init\_data().

## 3.39.2.26 uint16\_t tss\_t::ss1

Definition at line 135 of file types.h.

Referenced by cpu\_init\_data().

## 3.39.2.27 uint16\_t tss\_t::ss2

Definition at line 139 of file types.h.

Referenced by cpu\_init\_data().

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

- include/hal/**types.h**

## 3.40 vmalloc\_block\_t Struct Reference

Collaboration diagram for vmalloc\_block\_t:



### Data Fields

- **addr\_t \* stack\_ptr**  
*stack pointer for stack of free pages in partially allocated blocks*
- **addr\_t \* stack\_base**  
*base address of free page stack*
- struct **vmalloc\_block\_t \* prev**  
*link previous block in free list*
- struct **vmalloc\_block\_t \* next**  
*link next block in free list*

### 3.40.1 Detailed Description

Definition at line 110 of file vmalloc.c.

### 3.40.2 Field Documentation

#### 3.40.2.1 `struct vmalloc_block_t* vmalloc_block_t::next`

link next block in free list

Definition at line 121 of file vmalloc.c.

#### 3.40.2.2 `struct vmalloc_block_t* vmalloc_block_t::prev`

link previous block in free list

Definition at line 118 of file vmalloc.c.

#### 3.40.2.3 `addr_t* vmalloc_block_t::stack_base`

base address of free page stack

Definition at line 115 of file vmalloc.c.

#### 3.40.2.4 `addr_t* vmalloc_block_t::stack_ptr`

stack pointer for stack of free pages in partially allocated blocks

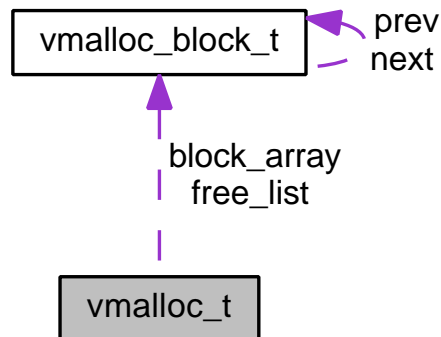
Definition at line 112 of file vmalloc.c.

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

- kernel/vmalloc.c

## 3.41 vmalloc\_t Struct Reference

Collaboration diagram for vmalloc\_t:



### Data Fields

- `addr_t base_addr`

*start address of the first block managed by the allocator*

- **addr\_t start\_addr**  
*start address of the address space region managed by the allocator*
- **addr\_t end\_addr**  
*end address of the address space region managed by the allocator*
- **addr\_t init\_limit**  
*address up to which blocks have been initialized*
- unsigned int **block\_count**  
*number of memory blocks managed by this allocator*
- struct **vmalloc\_block\_t** \* **block\_array**  
*array of memory block descriptors*
- struct **vmalloc\_block\_t** \* **free\_list**  
*list of completely or partially free blocks*

### 3.41.1 Detailed Description

Definition at line 87 of file vmalloc.c.

### 3.41.2 Field Documentation

#### 3.41.2.1 **addr\_t vmalloc\_t::base\_addr**

start address of the first block managed by the allocator

Definition at line 89 of file vmalloc.c.

#### 3.41.2.2 **struct vmalloc\_block\_t\* vmalloc\_t::block\_array**

array of memory block descriptors

Definition at line 104 of file vmalloc.c.

#### 3.41.2.3 **unsigned int vmalloc\_t::block\_count**

number of memory blocks managed by this allocator

Definition at line 101 of file vmalloc.c.

#### 3.41.2.4 **addr\_t vmalloc\_t::end\_addr**

end address of the address space region managed by the allocator

Definition at line 95 of file vmalloc.c.

#### 3.41.2.5 **struct vmalloc\_block\_t\* vmalloc\_t::free\_list**

list of completely or partially free blocks

Definition at line 107 of file vmalloc.c.

#### 3.41.2.6 `addr_t vmalloc_t::init_limit`

address up to which blocks have been initialized

Definition at line 98 of file `vmalloc.c`.

#### 3.41.2.7 `addr_t vmalloc_t::start_addr`

start address of the address space region managed by the allocator

Definition at line 92 of file `vmalloc.c`.

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

- `kernel/vmalloc.c`

## 3.42 `x86_cpuid_regs_t` Struct Reference

```
#include <hal/x86.h>
```

### Data Fields

- `uint32_t eax`
- `uint32_t ebx`
- `uint32_t ecx`
- `uint32_t edx`

#### 3.42.1 Detailed Description

Definition at line 38 of file `x86.h`.

#### 3.42.2 Field Documentation

##### 3.42.2.1 `uint32_t x86_cpuid_regs_t::eax`

Definition at line 39 of file `x86.h`.

Referenced by `cpu_detect_features()`.

##### 3.42.2.2 `uint32_t x86_cpuid_regs_t::ebx`

Definition at line 40 of file `x86.h`.

Referenced by `cpu_detect_features()`.

##### 3.42.2.3 `uint32_t x86_cpuid_regs_t::ecx`

Definition at line 41 of file `x86.h`.

Referenced by `cpu_detect_features()`.

#### 3.42.2.4 uint32\_t x86\_cpuid\_regs\_t::edx

Definition at line 42 of file x86.h.

Referenced by `cpu_detect_features()`.

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

- `include/hal/x86.h`

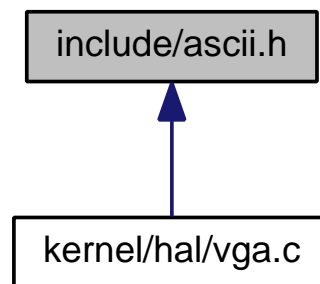


## Chapter 4

# File Documentation

### 4.1 include/ascii.h File Reference

This graph shows which files directly or indirectly include this file:



#### Macros

- `#define CHAR_BS 0x08`
- `#define CHAR_HT 0x09`
- `#define CHAR_LF 0x0a`
- `#define CHAR_CR 0x0d`

#### 4.1.1 Macro Definition Documentation

##### 4.1.1.1 `#define CHAR_BS 0x08`

Definition at line 35 of file `ascii.h`.

##### 4.1.1.2 `#define CHAR_CR 0x0d`

Definition at line 41 of file `ascii.h`.

#### 4.1.1.3 #define CHAR\_HT 0x09

Definition at line 37 of file ascii.h.

#### 4.1.1.4 #define CHAR\_LF 0x0a

Definition at line 39 of file ascii.h.

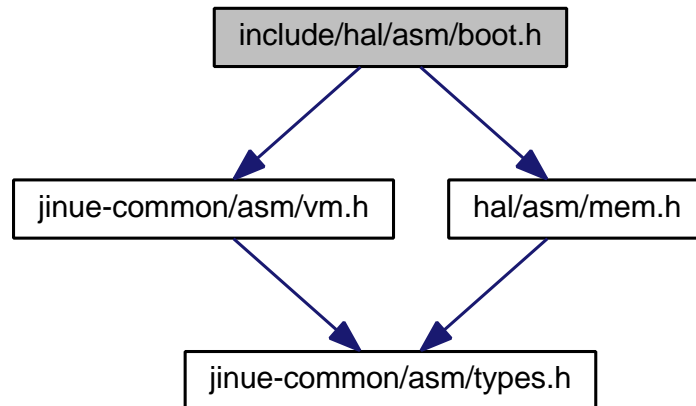
## 4.2 include/boot.h File Reference

## 4.3 include/hal/asm/boot.h File Reference

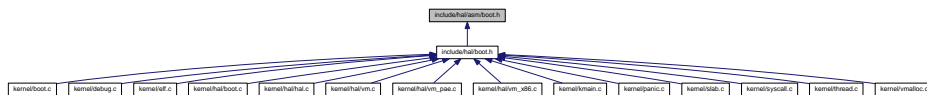
```
#include <jinue-common/asm/vm.h>
```

```
#include <hal/asm/mem.h>
```

Include dependency graph for boot.h:



This graph shows which files directly or indirectly include this file:



## Macros

- #define **BOOT\_E820\_ENTRIES** 0x1e8
- #define **BOOT\_SETUP\_SECTS** 0x1f1
- #define **BOOT\_SYSIZE** 0x1f4
- #define **BOOT\_SIGNATURE** 0x1fe
- #define **BOOT\_MAGIC** 0xaa55
- #define **BOOT\_SETUP** 0x200
- #define **BOOT\_SETUP\_HEADER** 0x202
- #define **BOOT\_SETUP\_MAGIC** 0x53726448 /\* "HdrS", reversed \*/



- `#define BOOT_RAMDISK_IMAGE 0x218`
- `#define BOOT_RAMDISK_SIZE 0x21C`
- `#define BOOT_CMD_LINE_PTR 0x228`
- `#define BOOT_E820_MAP 0x2d0`
- `#define BOOT_E820_MAP_END 0xd00`
- `#define BOOT_E820_MAP_SIZE (BOOT_E820_MAP_END - BOOT_E820_MAP)`
- `#define BOOT_SETUP32_ADDR MEM_ZONE_DMA16_START`
- `#define BOOT_SETUP32_SIZE PAGE_SIZE`
- `#define BOOT_DATA_STRUCT BOOT_E820_ENTRIES`
- `#define BOOT_STACK_HEAP_SIZE (4 * PAGE_SIZE)`

### 4.3.1 Macro Definition Documentation

#### 4.3.1.1 `#define BOOT_CMD_LINE_PTR 0x228`

Definition at line 59 of file boot.h.

#### 4.3.1.2 `#define BOOT_DATA_STRUCT BOOT_E820_ENTRIES`

Definition at line 71 of file boot.h.

#### 4.3.1.3 `#define BOOT_E820_ENTRIES 0x1e8`

Definition at line 39 of file boot.h.

#### 4.3.1.4 `#define BOOT_E820_MAP 0x2d0`

Definition at line 61 of file boot.h.

#### 4.3.1.5 `#define BOOT_E820_MAP_END 0xd00`

Definition at line 63 of file boot.h.

#### 4.3.1.6 `#define BOOT_E820_MAP_SIZE (BOOT_E820_MAP_END - BOOT_E820_MAP)`

Definition at line 65 of file boot.h.

#### 4.3.1.7 `#define BOOT_MAGIC 0xaa55`

Definition at line 47 of file boot.h.

#### 4.3.1.8 `#define BOOT_RAMDISK_IMAGE 0x218`

Definition at line 55 of file boot.h.

4.3.1.9 **#define BOOT\_RAMDISK\_SIZE 0x21C**

Definition at line 57 of file boot.h.

4.3.1.10 **#define BOOT\_SETUP 0x200**

Definition at line 49 of file boot.h.

4.3.1.11 **#define BOOT\_SETUP32\_ADDR MEM\_ZONE\_DMA16\_START**

Definition at line 67 of file boot.h.

4.3.1.12 **#define BOOT\_SETUP32\_SIZE PAGE\_SIZE**

Definition at line 69 of file boot.h.

4.3.1.13 **#define BOOT\_SETUP\_HEADER 0x202**

Definition at line 51 of file boot.h.

4.3.1.14 **#define BOOT\_SETUP\_MAGIC 0x53726448 /\* "HdrS", reversed \*/**

Definition at line 53 of file boot.h.

Referenced by boot\_info\_check().

4.3.1.15 **#define BOOT\_SETUP\_SECTS 0x1f1**

Definition at line 41 of file boot.h.

4.3.1.16 **#define BOOT\_SIGNATURE 0x1fe**

Definition at line 45 of file boot.h.

4.3.1.17 **#define BOOT\_STACK\_HEAP\_SIZE (4 \* PAGE\_SIZE)**

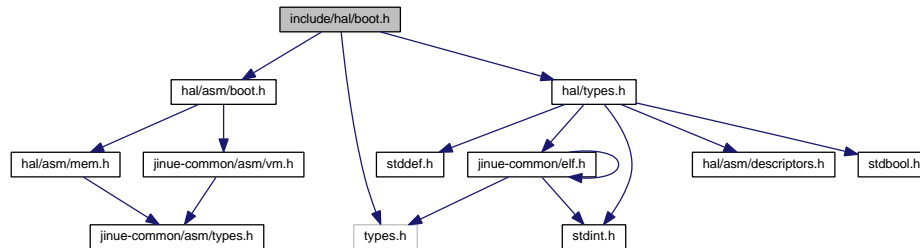
Definition at line 73 of file boot.h.

4.3.1.18 **#define BOOT\_SYSIZE 0x1f4**

Definition at line 43 of file boot.h.

## 4.4 include/hal/boot.h File Reference

```
#include <types.h>
#include <hal/asm/boot.h>
#include <hal/types.h>
Include dependency graph for boot.h:
```



This graph shows which files directly or indirectly include this file:



## Macros

- **#define boot\_heap\_alloc**(boot\_alloc, t, align) ((t \*)**boot\_heap\_alloc\_size**(boot\_alloc, sizeof(t), align))  
Allocate an object on the boot heap.

## Functions

- void **boot\_alloc\_init**(boot\_alloc\_t \*boot\_alloc, void \*heap\_ptr)  
Initialize the boot allocator.
- void \* **boot\_heap\_alloc\_size**(boot\_alloc\_t \*boot\_alloc, size\_t size, size\_t align)  
Allocate an object on the boot heap.
- void **boot\_heap\_push**(boot\_alloc\_t \*boot\_alloc)  
Push the current state of the boot allocator heap.
- void **boot\_heap\_pop**(boot\_alloc\_t \*boot\_alloc)  
Pop the last pushed boot allocator heap.
- **addr\_t boot\_page\_alloc\_early**(boot\_alloc\_t \*boot\_alloc)  
Early page allocation.
- **kern\_paddr\_t boot\_page\_frame\_alloc**(boot\_alloc\_t \*boot\_alloc)  
Allocate a page frame, that is, a page of physical memory.
- **addr\_t boot\_vmalloc**(boot\_alloc\_t \*boot\_alloc)  
Allocate a page of address space.
- **addr\_t boot\_page\_alloc**(boot\_alloc\_t \*boot\_alloc)  
Allocate a page in the allocations region of the kernel address space.
- **addr\_t boot\_page\_alloc\_image**(boot\_alloc\_t \*boot\_alloc)  
Allocate a page in the image region of the kernel address space.
- **bool boot\_info\_check**(bool panic\_on\_failure)
- const **boot\_info\_t** \* **get\_boot\_info**(void)
- void **boot\_info\_dump**(void)

#### 4.4.1 Macro Definition Documentation

4.4.1.1 `#define boot_heap_alloc( boot_alloc, t, align ) ((t *)boot_heap_alloc_size(boot_alloc, sizeof(t), align))`

Allocate an object on the boot heap.

This macro is a wrapper for `boot_heap_alloc_size` that takes a type as the second argument instead of an object size.

Parameters

<i>boot_alloc</i>	the boot allocator state
<i>t</i>	the type of object to allocate
<i>align</i>	the required start address alignment of the object, zero for no constraint

Returns

the allocated object

Definition at line 51 of file `boot.h`.

Referenced by `boot_heap_push()`, `hal_init()`, and `vm_pae_create_initial_addr_space()`.

#### 4.4.2 Function Documentation

4.4.2.1 `void boot_alloc_init ( boot_alloc_t * boot_alloc, void * heap_ptr )`

Initialize the boot allocator.

The boot allocator is used for heap and page allocation during kernel initialization. After this function is called, the boot heap is ready to use (see **`boot_heap_alloc()`** (p. 66)). However, the page and page frame allocators require additional initialization by the machine-dependent code before they can be used.

Parameters

<i>boot_alloc</i>	the allocator state initialized by this function
<i>heap_ptr</i>	the current top of the boot heap

Definition at line 54 of file `boot.c`.

References `boot_alloc_t::heap_ptr`, `boot_alloc_t::its_early`, and `memset()`.

Referenced by `kmain()`.

```

54                                     {
55     memset(boot_alloc, 0, sizeof(boot_alloc_t));
56     boot_alloc->heap_ptr = heap_ptr;
57     boot_alloc->its_early = true;
58     /* TODO handle heap limit. */
59 }
```

Here is the call graph for this function:



#### 4.4.2.2 void\* boot\_heap\_alloc\_size ( boot\_alloc\_t\* boot\_alloc, size\_t size, size\_t align )

Allocate an object on the boot heap.

Callers do not call this function directly but instead use the **boot\_heap\_alloc()** (p. 66) macro that takes a type as the second argument instead of an object size.

##### Parameters

<i>boot_alloc</i>	the boot allocator state
<i>size</i>	the size of the object to allocate, in bytes
<i>align</i>	the required start address alignment of the object, zero for no constraint

##### Returns

the allocated object

Definition at line 73 of file boot.c.

References `ALIGN_END_PTR`, and `boot_alloc_t::heap_ptr`.

```

73                                     {
74     if(align != 0) {
75         boot_alloc->heap_ptr = ALIGN_END_PTR(boot_alloc->heap_ptr, align);
76     }
77
78     void *object          = boot_alloc->heap_ptr;
79     boot_alloc->heap_ptr  = (char *)boot_alloc->heap_ptr + size;
80
81     return object;
82 }
```

#### 4.4.2.3 void boot\_heap\_pop ( boot\_alloc\_t\* boot\_alloc )

Pop the last pushed boot allocator heap.

This function frees all heap allocations performed since the matching call to **boot\_heap\_push()** (p. 68).

##### Parameters

<i>boot_alloc</i>	the boot allocator state
-------------------	--------------------------

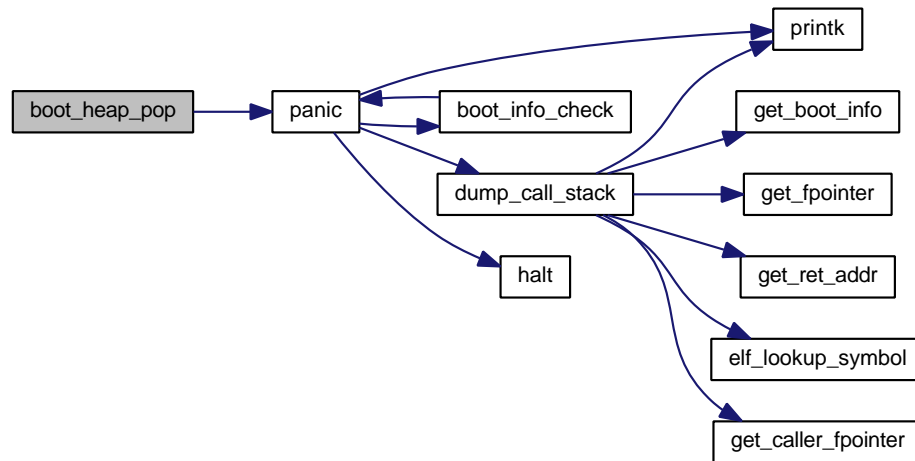
Definition at line 112 of file boot.c.

References `boot_alloc_t::heap_ptr`, `boot_alloc_t::heap_pushed_state`, `boot_heap_pushed_state::next`, `NULL`, and `panic()`.

```

112                                     {
113     if(boot_alloc->heap_pushed_state == NULL) {
114         panic("No more boot heap pushed state to pop.");
115     }
116
117     boot_alloc->heap_ptr          = boot_alloc->heap_pushed_state;
118     boot_alloc->heap_pushed_state = boot_alloc->heap_pushed_state->next;
119 }
```

Here is the call graph for this function:



#### 4.4.2.4 void boot\_heap\_push ( boot\_alloc\_t\* boot\_alloc )

Push the current state of the boot allocator heap.

All heap allocations performed after calling this function are freed by the matching call to **boot\_heap\_pop()** (p. 67). This function can be called multiple times before calling **boot\_heap\_pop()** (p. 67). Heap states pushed by this function are popped by **boot\_heap\_pop()** (p. 67)() in the reverse order they were pushed.

##### Parameters

<i>boot_alloc</i>	the boot allocator state
-------------------	--------------------------

Definition at line 95 of file boot.c.

References boot\_heap\_alloc, boot\_alloc\_t::heap\_pushed\_state, and boot\_heap\_pushed\_state::next.

```

95     {
96     struct boot_heap_pushed_state *pushed_state =
97         boot_heap_alloc(boot_alloc, struct boot_heap_pushed_state, 0);
98
99     pushed_state->next
100     boot_alloc->heap_pushed_state = pushed_state;
101 }

```

#### 4.4.2.5 bool boot\_info\_check ( bool panic\_on\_failure )

Definition at line 42 of file boot.c.

References BOOT\_SETUP\_MAGIC, boot\_info\_t::image\_start, boot\_info\_t::kernel\_start, NULL, page\_offset\_of, panic(), and boot\_info\_t::setup\_signature.

Referenced by kmain(), and panic().

```

42     {
43     const char *error_description = NULL;
44
45     /* This data structure is accessed early during the boot process, before
46      * paging is enabled. What this means is that, if boot_info is NULL and we
47      * dereference it, it does *not* cause a page fault or any other CPU

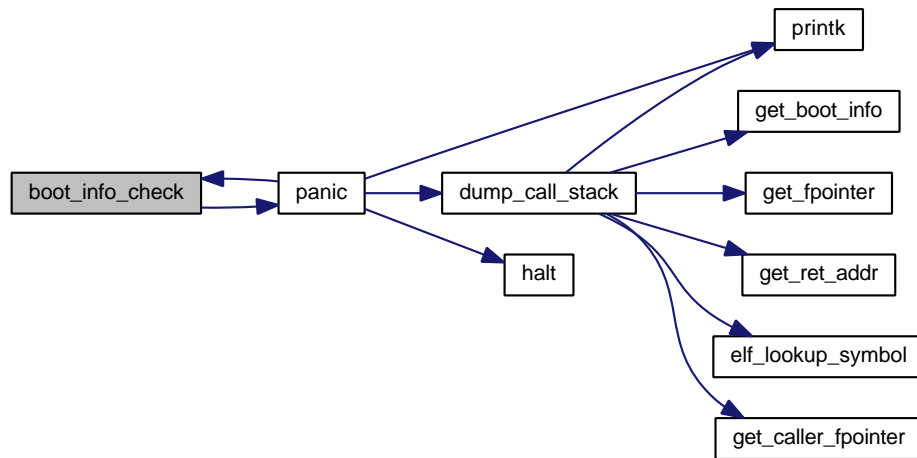
```

```

48     * exception. */
49     if (boot_info == NULL) {
50         error_description = "Boot information structure pointer is NULL.";
51     }
52     else if (boot_info->setup_signature != BOOT_SETUP_MAGIC) {
53         error_description = "Bad setup header signature.";
54     }
55     else if (page_offset_of(boot_info->image_start) != 0) {
56         error_description = "Bad image alignment.";
57     }
58     else if (page_offset_of(boot_info->kernel_start) != 0) {
59         error_description = "Bad kernel alignment.";
60     }
61     else {
62         return true;
63     }
64
65     if (panic_on_failure) {
66         panic(error_description);
67     }
68
69     return false;
70 }

```

Here is the call graph for this function:



#### 4.4.2.6 void boot\_info\_dump ( void )

Definition at line 76 of file boot.c.

References boot\_info\_t::boot\_end, boot\_info\_t::boot\_heap, boot\_info\_t::e820\_entries, boot\_info\_t::e820\_map, boot\_info\_t::image\_start, boot\_info\_t::image\_top, boot\_info\_t::kernel\_size, boot\_info\_t::kernel\_start, boot\_info\_t::page\_directory, boot\_info\_t::page\_table, printk(), boot\_info\_t::proc\_size, boot\_info\_t::proc\_start, and boot\_info\_t::setup\_signature.

```

76     {
77         printk("Boot information structure:\n");
78         printk("    kernel_start    %x %u\n", boot_info->kernel_start , boot_info->
kernel_start );
79         printk("    kernel_size     %x %u\n", boot_info->kernel_size , boot_info->
kernel_size );
80         printk("    proc_start      %x %u\n", boot_info->proc_start , boot_info->
proc_start );
81         printk("    proc_size       %x %u\n", boot_info->proc_size , boot_info->
proc_size );
82         printk("    image_start    %x %u\n", boot_info->image_start , boot_info->
image_start );

```

```

83     printk("    image_top      %x %u\n", boot_info->image_top      , boot_info->
image_top
);
84     printk("    e820_entries   %x %u\n", boot_info->e820_entries   , boot_info->
e820_entries
);
85     printk("    e820_map      %x %u\n", boot_info->e820_map      , boot_info->
e820_map
);
86     printk("    boot_heap     %x %u\n", boot_info->boot_heap     , boot_info->
boot_heap
);
87     printk("    boot_end      %x %u\n", boot_info->boot_end      , boot_info->
boot_end
);
88     printk("    page_table    %x %u\n", boot_info->page_table    , boot_info->
page_table
);
89     printk("    page_directory %x %u\n", boot_info->page_directory , boot_info->
page_directory
);
90     printk("    setup_signature %x %u\n", boot_info->setup_signature, boot_info->
setup_signature
);
91 }

```

Here is the call graph for this function:



#### 4.4.2.7 `addr_t boot_page_alloc ( boot_alloc_t * boot_alloc )`

Allocate a page in the allocations region of the kernel address space.

The physical memory is allocated just after the kernel image and other initialization-time allocations by calling **boot\_page\_frame\_alloc()** (p. 73) whereas the address space page is allocated in the allocations region by calling **vmalloc()** (p. 275).

If either of these two conditions is met, you must use `boot_pgalloc_image()` instead of this function: 1) The address space page allocator has not yet been initialized by calling **vmalloc\_init()** (p. 275); or 2) It is necessary to allocate multiple contiguous pages.

##### Parameters

<i>boot_alloc</i>	the boot allocator state
-------------------	--------------------------

##### Returns

address of allocated page

Definition at line 282 of file `boot.c`.

References `boot_page_frame_alloc()`, `clear_page()`, `VM_FLAG_READ_WRITE`, `vm_map_kernel()`, and `vmalloc()`.

Referenced by `hal_init()`, `slab_cache_init()`, and `thread_create_boot()`.

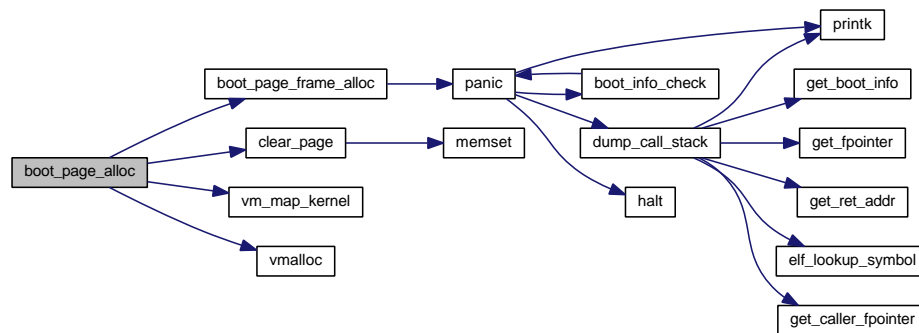
```

282                                     {
283     kern_paddr_t paddr = boot_page_frame_alloc(boot_alloc);
284     addr_t vaddr      = vmalloc();
285
286     vm_map_kernel(vaddr, paddr, VM_FLAG_READ_WRITE);
287
288     /* This newly allocated page may have data left from a previous boot which
289      * may contain sensitive information. Let's clear it. */
290     clear_page(vaddr);
291
292     return vaddr;
293 }

```



Here is the call graph for this function:



#### 4.4.2.8 `addr_t boot_page_alloc_early ( boot_alloc_t * boot_alloc )`

Early page allocation.

When the kernel is first entered, the setup code has set up temporary page tables that map a contiguous region of physical memory (RAM) that contains the kernel image at KLIMIT. The setup code itself allocates a few pages, notably for the temporary page tables and for the boot stack and heap. These pages are allocated sequentially just after the kernel image.

This function allocates pages sequentially following the kernel image and the setup code allocations. It is meant to be called early in the initialization process, while the temporary page tables set up by the setup code are still being used, which means before the kernel switches to the initial address space it sets up.

Because the page tables set up by the setup code are being used, there is a fixed relation between the virtual address of the pages allocated by this function and the physical address of the underlying page frames. This relation is expressed by the **EARLY\_PTR\_TO\_PHYS\_ADDR()** (p. 142) macro.

This function must not be called once the kernel has switched away from the page tables set up by the setup code to the initial address space it has set up itself. This function checks for this and triggers a kernel panic if it happens.

##### Parameters

<i>boot_alloc</i>	the boot allocator state
-------------------	--------------------------

##### Returns

address of allocated page

Definition at line 150 of file boot.c.

References `clear_page()`, `EARLY_PTR_TO_PHYS_ADDR`, `boot_alloc_t::its_early`, `boot_alloc_t::kernel_paddr_limit`, `boot_alloc_t::kernel_paddr_top`, `boot_alloc_t::kernel_vm_limit`, `boot_alloc_t::kernel_vm_top`, `NULL`, `PAGE_MASK`, `PAGE_SIZE`, and `panic()`.

Referenced by `vm_init_initial_page_directory()`, `vm_pae_create_initial_addr_space()`, and `vm_x86_create_initial_addr_space()`.

```

150                                     {
151     /* Preconditions */
152     if(! boot_alloc->its_early) {
153         panic("boot_pgalloc_early() called too late");
154     }
155
156     if(boot_alloc->kernel_vm_top == NULL) {

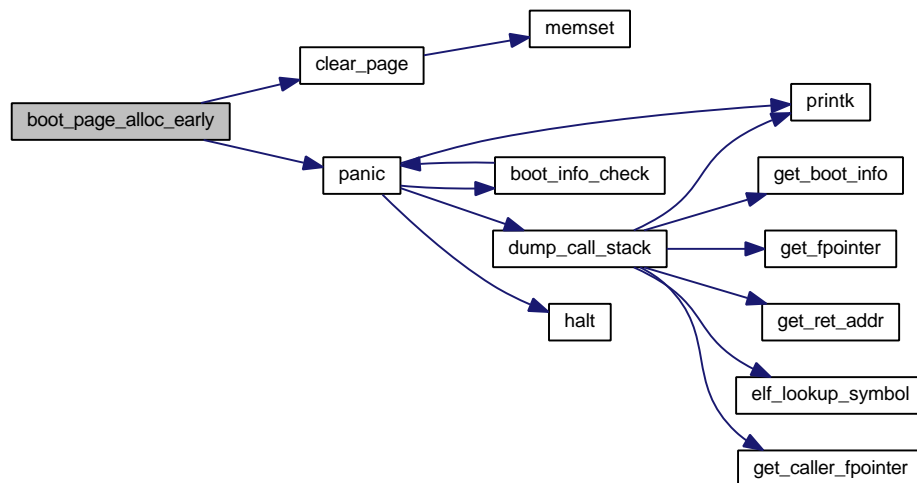
```

```

157     panic("boot_pgalloc_early(): allocator is uninitialized");
158 }
159
160 if(((uintptr_t)boot_alloc->kernel_vm_top & PAGE_MASK) != 0) {
161     panic("boot_pgalloc_early(): bad kernel region top VM address alignment");
162 }
163
164 if(boot_alloc->kernel_paddr_top != EARLY_PTR_TO_PHYS_ADDR(boot_alloc->
kernel_vm_top)) {
165     panic("boot_pgalloc_early(): inconsistent allocator state");
166 }
167
168 /* address of allocated page */
169 addr_t allocated_page = boot_alloc->kernel_vm_top;
170
171 /* Update allocator state.
172  *
173  * In this early allocator function that is called while the temporary page
174  * tables set up by the setup code are still being used, there is a fixed
175  * relationship between virtual and physical addresses. */
176 boot_alloc->kernel_vm_top = allocated_page + PAGE_SIZE;
177 boot_alloc->kernel_paddr_top = boot_alloc->kernel_paddr_top + PAGE_SIZE;
178
179 /* Check updated state against allocation limits. */
180 if(boot_alloc->kernel_vm_top > boot_alloc->kernel_vm_limit) {
181     panic("vmalloc_early(): kernel address space exhausted");
182 }
183
184 if(boot_alloc->kernel_paddr_top > boot_alloc->kernel_paddr_limit) {
185     panic("vmalloc_early(): available memory exhausted");
186 }
187
188 /* This newly allocated page may have data left from a previous boot which
189  * may contain sensitive information. Let's clear it. */
190 clear_page(allocated_page);
191
192 /* Post-condition */
193 if(boot_alloc->kernel_paddr_top != EARLY_PTR_TO_PHYS_ADDR(boot_alloc->
kernel_vm_top)) {
194     panic("boot_pgalloc_early(): inconsistent allocator state on return");
195 }
196
197 return allocated_page;
198 }

```

Here is the call graph for this function:



#### 4.4.2.9 `addr_t boot_page_alloc_image ( boot_alloc_t * boot_alloc )`

Allocate a page in the image region of the kernel address space.

Since the size of the image region is limited, use `boot_pgalloc()` instead of this function whenever possible.

The difference between this function and `boot_pgalloc()` is that the address space page is allocated by this function using **`boot_vmalloc()`** (p.74) instead of **`vmalloc()`** (p.275). Pages allocated by subsequent calls to this function are allocated sequentially.

#### Parameters

<i>boot_alloc</i>	the boot allocator state
-------------------	--------------------------

#### Returns

address of allocated page

Definition at line 310 of file `boot.c`.

References `boot_page_frame_alloc()`, `boot_vmalloc()`, `clear_page()`, `VM_FLAG_READ_WRITE`, and `vm_map_kernel()`.

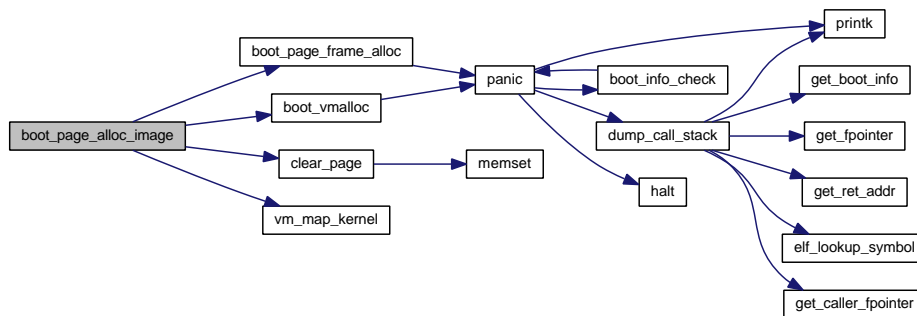
Referenced by `hal_init()`.

```

310
311     kern_paddr_t paddr = boot_page_frame_alloc(boot_alloc);
312     addr_t vaddr      = boot_vmalloc(boot_alloc);
313
314     vm_map_kernel(vaddr, paddr, VM_FLAG_READ_WRITE);
315
316     /* This newly allocated page may have data left from a previous boot which
317      * may contain sensitive information. Let's clear it. */
318     clear_page(vaddr);
319
320     return vaddr;
321 }

```

Here is the call graph for this function:



#### 4.4.2.10 kern\_paddr\_t boot\_page\_frame\_alloc ( boot\_alloc\_t \* boot\_alloc )

Allocate a page frame, that is, a page of physical memory.

The allocated page frame is not mapped anywhere. For a mapped page, call `boot_pgalloc()` or `boot_pgalloc_image()` instead;

## Parameters

<i>boot_alloc</i>	the boot allocator state
-------------------	--------------------------

## Returns

physical address of allocated page frame

Definition at line 210 of file boot.c.

References `boot_alloc_t::its_early`, `boot_alloc_t::kernel_paddr_limit`, `boot_alloc_t::kernel_paddr_top`, `PAGE_SIZE`, and `panic()`.

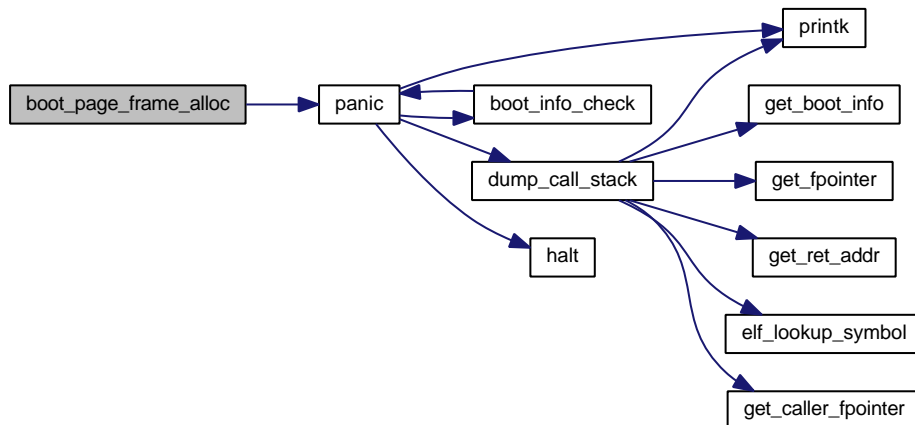
Referenced by `boot_page_alloc()`, `boot_page_alloc_image()`, `elf_load()`, and `elf_setup_stack()`.

```

210                                     {
211     if(boot_alloc->its_early) {
212         panic("boot_pmalloc() called too soon");
213     }
214
215     /* address of allocated page */
216     kern_paddr_t paddr = boot_alloc->kernel_paddr_top;
217
218     /* Update allocator state. */
219     boot_alloc->kernel_paddr_top = paddr + PAGE_SIZE;
220
221     /* Check bounds. */
222     if(boot_alloc->kernel_paddr_top > boot_alloc->kernel_paddr_limit) {
223         panic("pfalloc_boot(): available memory exhausted");
224     }
225
226     return paddr;
227 }

```

Here is the call graph for this function:



#### 4.4.2.11 `addr_t boot_vmalloc ( boot_alloc_t* boot_alloc )`

Allocate a page of address space.

No memory is mapped to the allocated page. The page is allocated from the image region of the kernel address space, just after the kernel image and other initialization-time page allocations. This function allocates pages sequentially.

## Parameters

<i>boot_alloc</i>	the boot allocator state
-------------------	--------------------------

## Returns

address of allocated page

Definition at line 241 of file boot.c.

References `boot_alloc_t::its_early`, `boot_alloc_t::kernel_vm_limit`, `boot_alloc_t::kernel_vm_top`, `NULL`, `PAGE_SIZE`, and `panic()`.

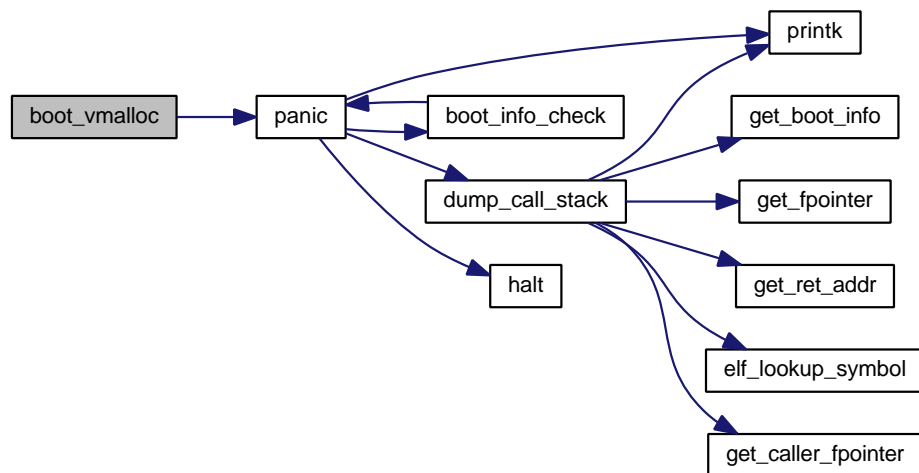
Referenced by `boot_page_alloc_image()`, and `vm_boot_init()`.

```

241                                     {
242     if(boot_alloc->its_early) {
243         panic("boot_vmalloc() called too soon");
244     }
245
246     if(boot_alloc->kernel_vm_top == NULL) {
247         panic("boot_pgalloc_early(): allocator is uninitialized");
248     }
249
250     /* address of allocated page */
251     addr_t page = boot_alloc->kernel_vm_top;
252
253     /* Update allocator state. */
254     boot_alloc->kernel_vm_top = page + PAGE_SIZE;
255
256     /* Check bounds. */
257     if(boot_alloc->kernel_vm_top > boot_alloc->kernel_vm_limit) {
258         panic("vmalloc_boot(): kernel address space exhausted");
259     }
260
261     return page;
262 }

```

Here is the call graph for this function:

4.4.2.12 `const boot_info_t* get_boot_info ( void )`

Definition at line 72 of file boot.c.

References boot\_info.

Referenced by dispatch\_syscall(), dump\_call\_stack(), and kmain().

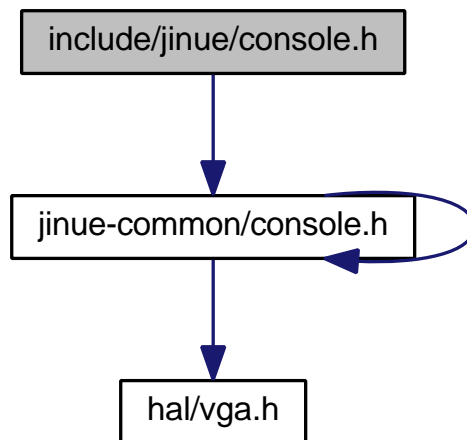
```
72                                     {  
73     return boot_info;  
74 }
```

## 4.5 include/console.h File Reference

## 4.6 include/jinue/console.h File Reference

```
#include <jinue-common/console.h>
```

Include dependency graph for console.h:

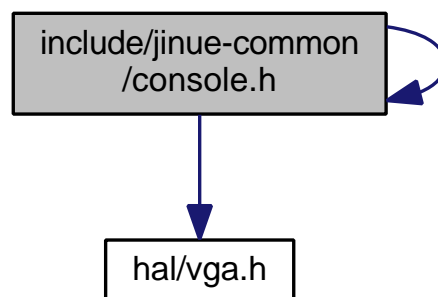


## 4.7 include/jinue-common/console.h File Reference

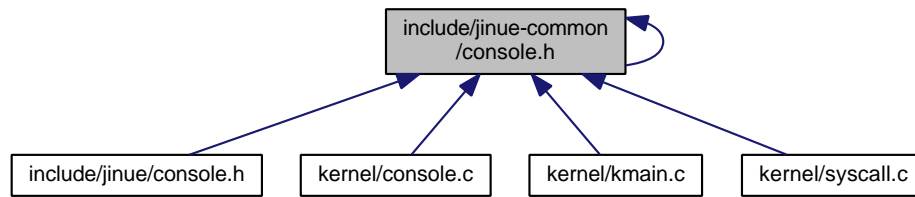
```
#include <jinue-common/console.h>
```

```
#include <hal/vga.h>
```

Include dependency graph for console.h:



This graph shows which files directly or indirectly include this file:



## Macros

- `#define CONSOLE_SERIAL_IOPORT SERIAL_COM1_IOPORT`
- `#define CONSOLE_SERIAL_BAUD_RATE 115200`
- `#define CONSOLE_DEFAULT_COLOR 0x0a /* VGA_COLOR_BRIGHTGREEN */`

## Functions

- void **console\_init** (void)
- void **console\_printn** (const char \*message, unsigned int n, int colour)
- void **console\_putc** (char c, int colour)
- void **console\_print** (const char \*message, int colour)

### 4.7.1 Macro Definition Documentation

#### 4.7.1.1 `#define CONSOLE_DEFAULT_COLOR 0x0a /* VGA_COLOR_BRIGHTGREEN */`

Definition at line 35 of file console.h.

Referenced by `dispatch_syscall()`.

#### 4.7.1.2 `#define CONSOLE_SERIAL_BAUD_RATE 115200`

Definition at line 40 of file console.h.

Referenced by `console_init()`.

#### 4.7.1.3 `#define CONSOLE_SERIAL_IOPORT SERIAL_COM1_IOPORT`

Definition at line 38 of file console.h.

Referenced by `console_init()`, `console_printn()`, and `console_putc()`.

### 4.7.2 Function Documentation

#### 4.7.2.1 void **console\_init** ( void )

Definition at line 38 of file console.c.

References `CONSOLE_SERIAL_BAUD_RATE`, `CONSOLE_SERIAL_IOPORT`, `serial_init()`, and `vga_init()`.

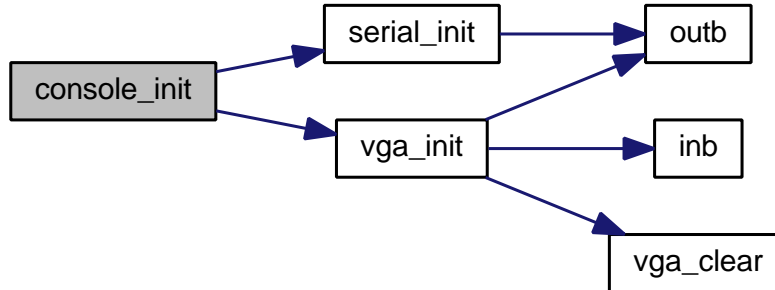
Referenced by kmain().

```

38         {
39     vga_init();
40     serial_init(CONSOLE_SERIAL_IOPORT, CONSOLE_SERIAL_BAUD_RATE);
41 }

```

Here is the call graph for this function:



#### 4.7.2.2 void console\_print ( const char \* message, int colour )

Definition at line 53 of file console.c.

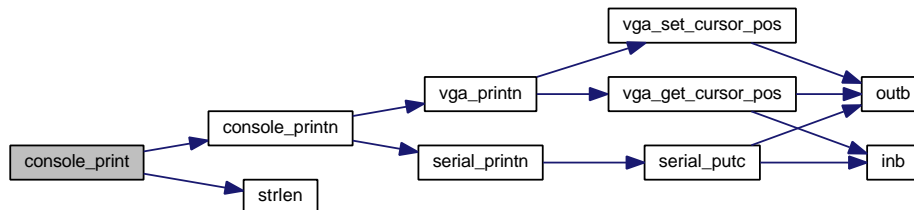
References `console_printn()`, and `strlen()`.

```

53     {
54     console_printn(message, strlen(message), colour);
55 }

```

Here is the call graph for this function:



#### 4.7.2.3 void console\_printn ( const char \* message, unsigned int n, int colour )

Definition at line 43 of file console.c.

References `CONSOLE_SERIAL_IOPORT`, `serial_printn()`, and `vga_printn()`.

Referenced by `console_print()`, and `dispatch_syscall()`.

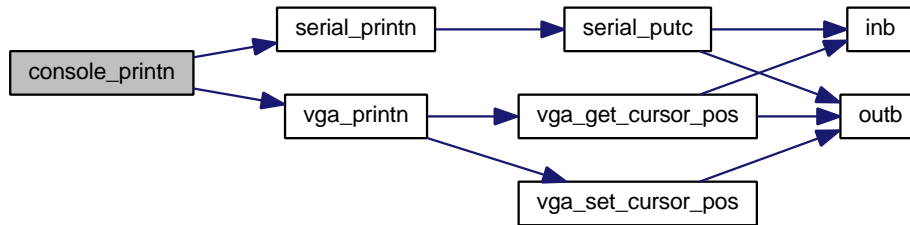
```

43     {
44     vga_printn(message, n, colour);
45     serial_printn(CONSOLE_SERIAL_IOPORT, message, n);
46 }

```



Here is the call graph for this function:



#### 4.7.2.4 void console\_putc ( char c, int colour )

Definition at line 48 of file console.c.

References `CONSOLE_SERIAL_IOPORT`, `serial_putc()`, and `vga_putc()`.

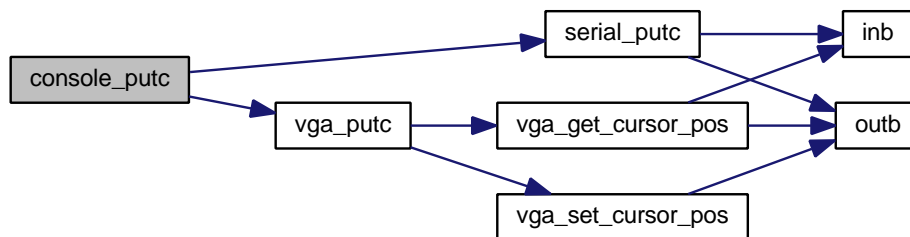
Referenced by `dispatch_syscall()`.

```

48     {
49         vga_putc(c, colour);
50         serial_putc(CONSOLE_SERIAL_IOPORT, c);
51     }

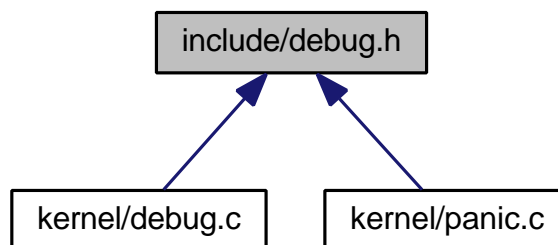
```

Here is the call graph for this function:



## 4.8 include/debug.h File Reference

This graph shows which files directly or indirectly include this file:



## Functions

- void **dump\_call\_stack** (void)

### 4.8.1 Function Documentation

#### 4.8.1.1 void dump\_call\_stack ( void )

Definition at line 41 of file debug.c.

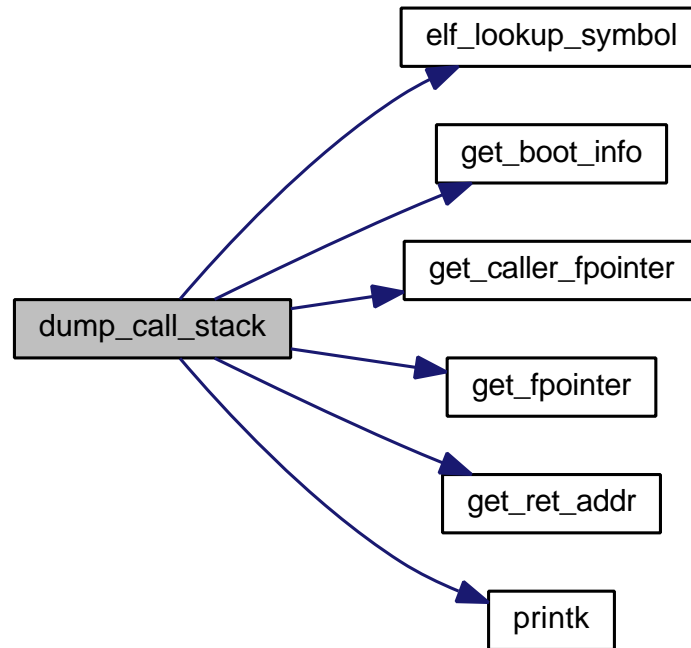
References `elf_symbol_t::addr`, `boot_info`, `elf_lookup_symbol()`, `get_boot_info()`, `get_caller_fpointer()`, `get_fpointer()`, `get_ret_addr()`, `boot_info_t::kernel_start`, `elf_symbol_t::name`, `NULL`, `printk()`, and `STT_FUNCTION`.

Referenced by `panic()`.

```

41      {
42      addr_t      fptr;
43
44      const boot_info_t *boot_info = get_boot_info();
45
46      printk("Call stack dump:\n");
47
48      fptr = get_fpointer();
49
50      while(fptr != NULL) {
51          addr_t return_addr = get_ret_addr(fptr);
52          if(return_addr == NULL) {
53              break;
54          }
55
56          /* assume e8 xx xx xx xx for call instruction encoding */
57          return_addr -= 5;
58
59          elf_symbol_t symbol;
60          int retval = elf_lookup_symbol(
61              boot_info->kernel_start,
62              (Elf32_Addr)return_addr,
63              STT_FUNCTION,
64              &symbol);
65
66          if(retval < 0) {
67              printk("\t0x%x (unknown)\n", return_addr);
68          }
69          else {
70              const char *name = symbol.name;
71
72              if(name == NULL) {
73                  name = "[unknown]";
74              }
75
76              printk(
77                  "\t0x%x (%s+u)\n",
78                  return_addr,
79                  name,
80                  return_addr - symbol.addr);
81          }
82
83          fptr = get_caller_fpointer(fptr);
84      }
85  }
```

Here is the call graph for this function:

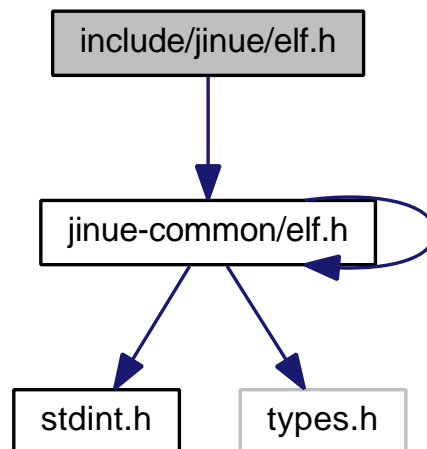


## 4.9 include/elf.h File Reference

### 4.10 include/jinue/elf.h File Reference

```
#include <jinue-common/elf.h>
```

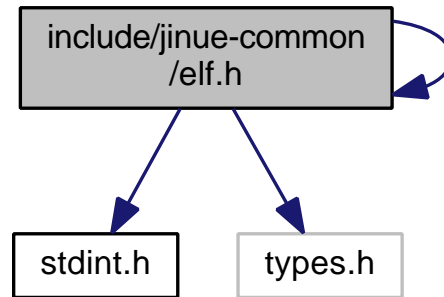
Include dependency graph for `elf.h`:



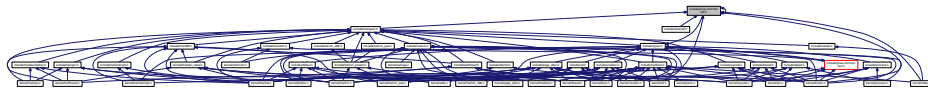
## 4.11 include/jinue-common/elf.h File Reference

```
#include <stdint.h>
#include <jinue-common/elf.h>
#include <types.h>
```

Include dependency graph for elf.h:



This graph shows which files directly or indirectly include this file:



## Data Structures

- struct **elf\_info\_t**
- struct **elf\_symbol\_t**
- struct **Elf32\_Ehdr**
- struct **Elf32\_Phdr**
- struct **Elf32\_Shdr**
- struct **Elf32\_Sym**
- struct **Elf32\_auxv\_t**

## Macros

- **#define EI\_MAG0** 0  
*Index of file identification - byte 0.*
- **#define EI\_MAG1** 1  
*Index of file identification - byte 1.*
- **#define EI\_MAG2** 2  
*Index of file identification - byte 2.*
- **#define EI\_MAG3** 3  
*Index of file identification - byte 3.*
- **#define EI\_CLASS** 4  
*File class.*
- **#define EI\_DATA** 5  
*Data encoding.*

- **#define EI\_VERSION 6**  
*File version.*
- **#define EI\_PAD 7**  
*Start of padding bytes.*
- **#define EI\_NIDENT 16**  
*size of e\_ident[]*
- **#define ELF\_MAGIC0 0x7f**  
*File identification - byte 0 (0x7f)*
- **#define ELF\_MAGIC1 'E'**  
*File identification - byte 1 ('E')*
- **#define ELF\_MAGIC2 'L'**  
*File identification - byte 2 ('L')*
- **#define ELF\_MAGIC3 'F'**  
*File identification - byte 3 ('F')*
- **#define EM\_NONE 0**  
*No machine.*
- **#define EM\_SPARC 2**  
*SPARC.*
- **#define EM\_386 3**  
*Intel 80386.*
- **#define EM\_MIPS 8**  
*MIPS RS3000.*
- **#define EM\_SPARC32PLUS 18**  
*Enhanced instruction set SPARC.*
- **#define EM\_ARM 40**  
*32-bit ARM*
- **#define EM\_X86\_64 62**  
*AMD64/X86-64.*
- **#define EM\_OPENRISC 92**  
*OpenRISC 32-bit embedded processor.*
- **#define EM\_ALTERA\_NIOS2 113**  
*Altera Nios 2 32-bit soft processor.*
- **#define EM\_AARCH64 183**  
*64-bit AARCH64 ARM*
- **#define EM\_MICROBLAZE 189**  
*Xilinx MicroBlaze 32-bit soft processor.*
- **#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 ELFCLASSNONE 0**

- Invalid class.*
- #define **ELFCLASS32** 1
  - 32-bit objects*
- #define **ELFCLASS64** 2
  - 64-bit objects*
- #define **ELFDATANONE** 0
  - Invalid data encoding.*
- #define **ELFDATA2LSB** 1
  - Little-endian.*
- #define **ELFDATA2MSB** 2
  - Big-endian.*
- #define **PT\_NULL** 0
  - Unused entry.*
- #define **PT\_LOAD** 1
  - Loadable segment.*
- #define **PT\_DYNAMIC** 2
  - Dynamic linking information.*
- #define **PT\_INTERP** 3
  - Path to program interpreter.*
- #define **PT\_NOTE** 4
  - Location and size of notes.*
- #define **PT\_SHLIB** 5
  - Unspecified semantics.*
- #define **PT\_PHDR** 6
  - Program header table.*
- #define **SHT\_NULL** 0
  - Inactive section.*
- #define **SHT\_PROGBITS** 1
  - Program data.*
- #define **SHT\_SYMTAB** 2
  - Symbol table.*
- #define **SHT\_STRTAB** 3
  - String table.*
- #define **SHT\_RELA** 4
  - Relocations with addends.*
- #define **SHT\_HASH** 5
  - Symbol hash table.*
- #define **SHT\_DYNAMIC** 6
  - Information for dynamic linking.*
- #define **SHT\_NOTE** 7
  - Notes section.*
- #define **SHT\_NOBITS** 8
  - Section without data (.bss)*
- #define **SHT\_REL** 9
  - Relocations without addends.*
- #define **SHT\_SHLIB** 10
  - Reserved, unspecified semantic, not ABI compliant.*

- **#define SHT\_DYNSYM 11**  
*Dynamic symbols table.*
- **#define STB\_LOCAL 0**  
*Local binding.*
- **#define STB\_GLOBAL 1**  
*Global binding.*
- **#define STB\_WEAK 2**  
*Weak binding.*
- **#define STT\_NOTYPE 0**  
*Unspecified type.*
- **#define STT\_OBJECT 1**  
*Data object.*
- **#define STT\_FUNCTION 2**  
*Function or other executable code.*
- **#define STT\_SECTION 3**  
*Section symbol.*
- **#define STT\_FILE 4**  
*Source file.*
- **#define ELF32\_ST\_BIND(i) ((i) >> 4)**
- **#define ELF32\_ST\_TYPE(i) ((i) & 0xf)**
- **#define STN\_UNDEF 0**  
*Undefined symbol index.*
- **#define PF\_R (1 << 2)**
- **#define PF\_W (1 << 1)**
- **#define PF\_X (1 << 0)**
- **#define AT\_NULL 0**  
*Last entry.*
- **#define AT\_IGNORE 1**  
*Ignore entry.*
- **#define AT\_EXECD 2**  
*Program file descriptor.*
- **#define AT\_PHDR 3**  
*Program headers address.*
- **#define AT\_PHEM 4**  
*Size of program header entry.*
- **#define AT\_PNUM 5**  
*Number of program header entries.*
- **#define AT\_PAGESZ 6**  
*Page size.*
- **#define AT\_BASE 7**  
*Base address.*
- **#define AT\_FLAGS 8**  
*Flags.*
- **#define AT\_ENTRY 9**  
*Program entry point.*
- **#define AT\_DCACHEBSIZE 10**  
*Data cache block size.*

- **#define AT\_ICACHEBSIZE 11**  
*Instruction cache block size.*
- **#define AT\_UCACHEBSIZE 12**  
*Unified cache block size.*
- **#define AT\_STACKBASE 13**  
*Stack base address for main thread.*
- **#define AT\_HWCAP 16**  
*Machine-dependent processor feature flags.*
- **#define AT\_HWCAP2 26**  
*More machine-dependent processor feature flags.*
- **#define AT\_SYSINFO\_EHDR 33**  
*Address of vDSO.*

## Typedefs

- typedef **uint32\_t Elf32\_Addr**
- typedef **uint16\_t Elf32\_Half**
- typedef **uint32\_t Elf32\_Off**
- typedef **int32\_t Elf32\_Sword**
- typedef **uint32\_t Elf32\_Word**
- typedef **Elf32\_auxv\_t auxv\_t**

## Functions

- void **elf\_check** (**Elf32\_Ehdr** \*elf)
- void **elf\_load** (**elf\_info\_t** \*info, **Elf32\_Ehdr** \*elf, **addr\_space\_t** \*addr\_space, **boot\_alloc\_t** \*boot\_alloc)
- void **elf\_setup\_stack** (**elf\_info\_t** \*info, **boot\_alloc\_t** \*boot\_alloc)
- int **elf\_lookup\_symbol** (const **Elf32\_Ehdr** \*elf\_header, **Elf32\_Addr** addr, int type, **elf\_symbol\_t** \*result)

### 4.11.1 Macro Definition Documentation

#### 4.11.1.1 #define AT\_BASE 7

Base address.

Definition at line 339 of file elf.h.

#### 4.11.1.2 #define AT\_DCACHEBSIZE 10

Data cache block size.

Definition at line 348 of file elf.h.

#### 4.11.1.3 #define AT\_ENTRY 9

Program entry point.

Definition at line 345 of file elf.h.

Referenced by `elf_setup_stack()`.



#### 4.11.1.4 #define AT\_EXECFD 2

Program file descriptor.

Definition at line 324 of file elf.h.

#### 4.11.1.5 #define AT\_FLAGS 8

Flags.

Definition at line 342 of file elf.h.

#### 4.11.1.6 #define AT\_HWCAP 16

Machine-dependent processor feature flags.

Definition at line 360 of file elf.h.

#### 4.11.1.7 #define AT\_HWCAP2 26

More machine-dependent processor feature flags.

Definition at line 363 of file elf.h.

#### 4.11.1.8 #define AT\_ICACHEBSIZE 11

Instruction cache block size.

Definition at line 351 of file elf.h.

#### 4.11.1.9 #define AT\_IGNORE 1

Ignore entry.

Definition at line 321 of file elf.h.

#### 4.11.1.10 #define AT\_NULL 0

Last entry.

Definition at line 318 of file elf.h.

Referenced by elf\_setup\_stack().

#### 4.11.1.11 #define AT\_PAGESZ 6

Page size.

Definition at line 336 of file elf.h.

Referenced by elf\_setup\_stack().

**4.11.1.12 #define AT\_PHDR 3**

Program headers address.

Definition at line 327 of file elf.h.

Referenced by elf\_setup\_stack().

**4.11.1.13 #define AT\_PHENT 4**

Size of program header entry.

Definition at line 330 of file elf.h.

Referenced by elf\_setup\_stack().

**4.11.1.14 #define AT\_PHNUM 5**

Number of program header entries.

Definition at line 333 of file elf.h.

Referenced by elf\_setup\_stack().

**4.11.1.15 #define AT\_STACKBASE 13**

Stack base address for main thread.

Definition at line 357 of file elf.h.

Referenced by elf\_setup\_stack().

**4.11.1.16 #define AT\_SYSINFO\_EHDR 33**

Address of vDSO.

Definition at line 366 of file elf.h.

**4.11.1.17 #define AT\_UCACHEBSIZE 12**

Unified cache block size.

Definition at line 354 of file elf.h.

**4.11.1.18 #define EI\_CLASS 4**

File class.

Definition at line 50 of file elf.h.

Referenced by elf\_check().

**4.11.1.19 #define EI\_DATA 5**

Data encoding.

Definition at line 53 of file elf.h.

Referenced by elf\_check().

#### 4.11.1.20 #define EI\_MAG0 0

Index of file identification - byte 0.

Definition at line 38 of file elf.h.

Referenced by elf\_check().

#### 4.11.1.21 #define EI\_MAG1 1

Index of file identification - byte 1.

Definition at line 41 of file elf.h.

Referenced by elf\_check().

#### 4.11.1.22 #define EI\_MAG2 2

Index of file identification - byte 2.

Definition at line 44 of file elf.h.

Referenced by elf\_check().

#### 4.11.1.23 #define EI\_MAG3 3

Index of file identification - byte 3.

Definition at line 47 of file elf.h.

Referenced by elf\_check().

#### 4.11.1.24 #define EI\_NIDENT 16

size of e\_ident[]

Definition at line 62 of file elf.h.

#### 4.11.1.25 #define EI\_PAD 7

Start of padding bytes.

Definition at line 59 of file elf.h.

#### 4.11.1.26 #define EI\_VERSION 6

File version.

Definition at line 56 of file elf.h.

Referenced by elf\_check().

4.11.1.27 `#define ELF32_ST_BIND( i ) ((i) >> 4)`

Definition at line 233 of file elf.h.

4.11.1.28 `#define ELF32_ST_TYPE( i ) ((i) & 0xf)`

Definition at line 235 of file elf.h.

Referenced by elf\_lookup\_symbol().

4.11.1.29 `#define ELF_MAGIC0 0x7f`

File identification - byte 0 (0x7f)

Definition at line 66 of file elf.h.

Referenced by elf\_check().

4.11.1.30 `#define ELF_MAGIC1 'E'`

File identification - byte 1 ('E')

Definition at line 69 of file elf.h.

Referenced by elf\_check().

4.11.1.31 `#define ELF_MAGIC2 'L'`

File identification - byte 2 ('L')

Definition at line 72 of file elf.h.

Referenced by elf\_check().

4.11.1.32 `#define ELF_MAGIC3 'F'`

File identification - byte 3 ('F')

Definition at line 75 of file elf.h.

Referenced by elf\_check().

4.11.1.33 `#define ELFCLASS32 1`

32-bit objects

Definition at line 132 of file elf.h.

Referenced by elf\_check().

4.11.1.34 `#define ELFCLASS64 2`

64-bit objects

Definition at line 135 of file elf.h.

**4.11.1.35 #define ELFCLASSNONE 0**

Invalid class.

Definition at line 129 of file elf.h.

**4.11.1.36 #define ELFDATA2LSB 1**

Little-endian.

Definition at line 142 of file elf.h.

Referenced by elf\_check().

**4.11.1.37 #define ELFDATA2MSB 2**

Big-endian.

Definition at line 145 of file elf.h.

**4.11.1.38 #define ELFDATANONE 0**

Invalid data encoding.

Definition at line 139 of file elf.h.

**4.11.1.39 #define EM\_386 3**

Intel 80386.

Definition at line 85 of file elf.h.

Referenced by elf\_check().

**4.11.1.40 #define EM\_AARCH64 183**

64-bit AARCH64 ARM

Definition at line 106 of file elf.h.

**4.11.1.41 #define EM\_ALTERA\_NIOS2 113**

Altera Nios 2 32-bit soft processor.

Definition at line 103 of file elf.h.

**4.11.1.42 #define EM\_ARM 40**

32-bit ARM

Definition at line 94 of file elf.h.

**4.11.1.43 #define EM\_MICROBLAZE 189**

Xilinx MicroBlaze 32-bit soft processor.

Definition at line 109 of file elf.h.

**4.11.1.44 #define EM\_MIPS 8**

MIPS RS3000.

Definition at line 88 of file elf.h.

**4.11.1.45 #define EM\_NONE 0**

No machine.

Definition at line 79 of file elf.h.

**4.11.1.46 #define EM\_OPENRISC 92**

OpenRISC 32-bit embedded processor.

Definition at line 100 of file elf.h.

**4.11.1.47 #define EM\_SPARC 2**

SPARC.

Definition at line 82 of file elf.h.

**4.11.1.48 #define EM\_SPARC32PLUS 18**

Enhanced instruction set SPARC.

Definition at line 91 of file elf.h.

**4.11.1.49 #define EM\_X86\_64 62**

AMD64/X86-64.

Definition at line 97 of file elf.h.

**4.11.1.50 #define ET\_CORE 4**

Core file.

Definition at line 125 of file elf.h.

**4.11.1.51 #define ET\_DYN 3**

Shared object file.

Definition at line 122 of file elf.h.

**4.11.1.52 #define ET\_EXEC 2**

Executable file.

Definition at line 119 of file elf.h.

Referenced by elf\_check().

**4.11.1.53 #define ET\_NONE 0**

No file type.

Definition at line 113 of file elf.h.

**4.11.1.54 #define ET\_REL 1**

Relocatable file.

Definition at line 116 of file elf.h.

**4.11.1.55 #define PF\_R (1 << 2)**

Definition at line 242 of file elf.h.

**4.11.1.56 #define PF\_W (1 << 1)**

Definition at line 244 of file elf.h.

Referenced by elf\_load().

**4.11.1.57 #define PF\_X (1 << 0)**

Definition at line 246 of file elf.h.

**4.11.1.58 #define PT\_DYNAMIC 2**

Dynamic linking information.

Definition at line 155 of file elf.h.

**4.11.1.59 #define PT\_INTERP 3**

Path to program interpreter.

Definition at line 158 of file elf.h.

**4.11.1.60 #define PT\_LOAD 1**

Loadable segment.

Definition at line 152 of file elf.h.

Referenced by elf\_load().

**4.11.1.61 #define PT\_NOTE 4**

Location and size of notes.

Definition at line 161 of file elf.h.

**4.11.1.62 #define PT\_NULL 0**

Unused entry.

Definition at line 149 of file elf.h.

**4.11.1.63 #define PT\_PHDR 6**

Program header table.

Definition at line 167 of file elf.h.

**4.11.1.64 #define PT\_SHLIB 5**

Unspecified semantics.

Definition at line 164 of file elf.h.

**4.11.1.65 #define SHT\_DYNAMIC 6**

Information for dynamic linking.

Definition at line 189 of file elf.h.

**4.11.1.66 #define SHT\_DYNSYM 11**

Dynamic symbols table.

Definition at line 204 of file elf.h.

**4.11.1.67 #define SHT\_HASH 5**

Symbol hash table.

Definition at line 186 of file elf.h.

**4.11.1.68 #define SHT\_NOBITS 8**

Section without data (.bss)

Definition at line 195 of file elf.h.

**4.11.1.69 #define SHT\_NOTE 7**

Notes section.

Definition at line 192 of file elf.h.



**4.11.1.70 #define SHT\_NULL 0**

Inactive section.

Definition at line 171 of file elf.h.

**4.11.1.71 #define SHT\_PROGBITS 1**

Program data.

Definition at line 174 of file elf.h.

**4.11.1.72 #define SHT\_REL 9**

Relocations without addends.

Definition at line 198 of file elf.h.

**4.11.1.73 #define SHT\_RELA 4**

Relocations with addends.

Definition at line 183 of file elf.h.

**4.11.1.74 #define SHT\_SHLIB 10**

Reserved, unspecified semantic, not ABI compliant.

Definition at line 201 of file elf.h.

**4.11.1.75 #define SHT\_STRTAB 3**

String table.

Definition at line 180 of file elf.h.

**4.11.1.76 #define SHT\_SYMTAB 2**

Symbol table.

Definition at line 177 of file elf.h.

Referenced by elf\_lookup\_symbol().

**4.11.1.77 #define STB\_GLOBAL 1**

Global binding.

Definition at line 211 of file elf.h.

**4.11.1.78 #define STB\_LOCAL 0**

Local binding.

Definition at line 208 of file elf.h.

**4.11.1.79 #define STB\_WEAK 2**

Weak binding.

Definition at line 214 of file elf.h.

**4.11.1.80 #define STN\_UNDEF 0**

Undefined symbol index.

Definition at line 239 of file elf.h.

**4.11.1.81 #define STT\_FILE 4**

Source file.

Definition at line 230 of file elf.h.

**4.11.1.82 #define STT\_FUNCTION 2**

Function or other executable code.

Definition at line 224 of file elf.h.

Referenced by `dump_call_stack()`.

**4.11.1.83 #define STT\_NOTYPE 0**

Unspecified type.

Definition at line 218 of file elf.h.

**4.11.1.84 #define STT\_OBJECT 1**

Data object.

Definition at line 221 of file elf.h.

**4.11.1.85 #define STT\_SECTION 3**

Section symbol.

Definition at line 227 of file elf.h.

## 4.11.2 Typedef Documentation

### 4.11.2.1 typedef Elf32\_auxv\_t auxv\_t

Definition at line 315 of file elf.h.

### 4.11.2.2 typedef uint32\_t Elf32\_Addr

Definition at line 248 of file elf.h.

### 4.11.2.3 typedef uint16\_t Elf32\_Half

Definition at line 250 of file elf.h.

### 4.11.2.4 typedef uint32\_t Elf32\_Off

Definition at line 252 of file elf.h.

### 4.11.2.5 typedef int32\_t Elf32\_Sword

Definition at line 254 of file elf.h.

### 4.11.2.6 typedef uint32\_t Elf32\_Word

Definition at line 256 of file elf.h.

## 4.11.3 Function Documentation

### 4.11.3.1 void elf\_check ( Elf32\_Ehdr \* elf )

Definition at line 43 of file elf.c.

References Elf32\_Ehdr::e\_entry, Elf32\_Ehdr::e\_flags, Elf32\_Ehdr::e\_ident, Elf32\_Ehdr::e\_machine, Elf32\_Ehdr::e\_phentsize, Elf32\_Ehdr::e\_phnum, Elf32\_Ehdr::e\_phoff, Elf32\_Ehdr::e\_type, Elf32\_Ehdr::e\_version, EI\_CLASS, EI\_DATA, EI\_MAG0, EI\_MAG1, EI\_MAG2, EI\_MAG3, EI\_VERSION, ELF\_MAGIC0, ELF\_MAGIC1, ELF\_MAGIC2, ELF\_MAGIC3, ELFCLASS32, ELFDATA2LSB, EM\_386, ET\_EXEC, and panic().

Referenced by elf\_load().

```

43                                     {
44     /* check: valid ELF binary magic number */
45     if ( elf->e_ident[EI_MAG0] != ELF_MAGIC0 ||
46         elf->e_ident[EI_MAG1] != ELF_MAGIC1 ||
47         elf->e_ident[EI_MAG2] != ELF_MAGIC2 ||
48         elf->e_ident[EI_MAG3] != ELF_MAGIC3 ) {
49         panic("Not an ELF binary");
50     }
51
52     /* check: 32-bit objects */
53     if (elf->e_ident[EI_CLASS] != ELFCLASS32) {
54         panic("Bad file class");
55     }
56
57     /* check: endianness */
58     if (elf->e_ident[EI_DATA] != ELFDATA2LSB) {

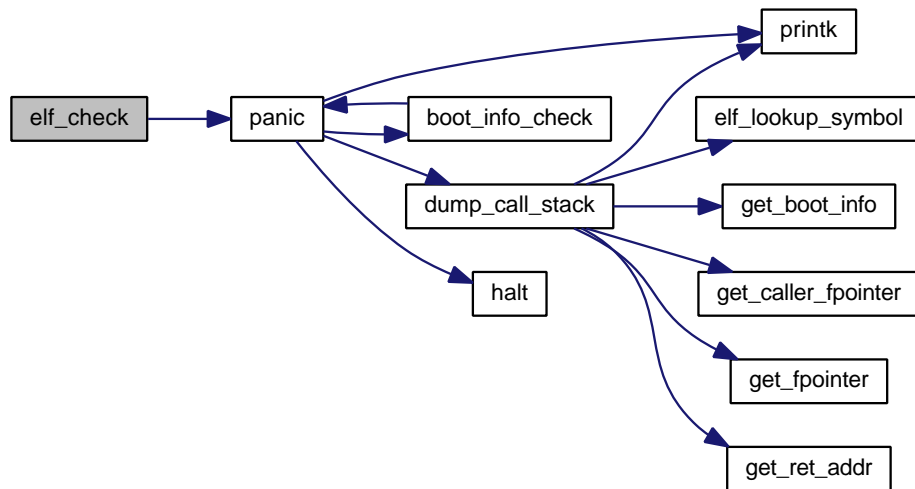
```

```

59     panic("Bad endianness");
60 }
61
62 /* check: version */
63 if(elf->e_version != 1 || elf->e_ident[EI_VERSION] != 1) {
64     panic("Not ELF version 1");
65 }
66
67 /* check: machine */
68 if(elf->e_machine != EM_386) {
69     panic("This process manager binary does not target the x86 architecture");
70 }
71
72 /* check: the 32-bit Intel architecture defines no flags */
73 if(elf->e_flags != 0) {
74     panic("Invalid flags specified");
75 }
76
77 /* check: file type is executable */
78 if(elf->e_type != ET_EXEC) {
79     panic("process manager binary is not an an executable");
80 }
81
82 /* check: must have a program header */
83 if(elf->e_phoff == 0 || elf->e_phnum == 0) {
84     panic("No program headers");
85 }
86
87 /* check: must have an entry point */
88 if(elf->e_entry == 0) {
89     panic("No entry point for process manager");
90 }
91
92 /* check: program header entry size */
93 if(elf->e_phentsize != sizeof(Elf32_Phdr)) {
94     panic("Unsupported program header size");
95 }
96 }

```

Here is the call graph for this function:



4.11.3.2 void elf\_load ( elf\_info\_t\* info, Elf32\_Ehdr\* elf, addr\_space\_t\* addr\_space, boot\_alloc\_t\* boot\_alloc )

TODO: add exec flag once PAE is enabled

TODO add exec flag once PAE is enabled TODO lookup actual address of page frame

Definition at line 98 of file elf.c.

References `elf_info_t::addr_space`, `ALIGN_END_PTR`, `ALIGN_START_PTR`, `elf_info_t::at_phdr`, `elf_info_t::at_phent`, `elf_info_t::at_phnum`, `boot_page_frame_alloc()`, `Elf32_Ehdr::e_entry`, `Elf32_Ehdr::e_phentsize`, `Elf32_Ehdr::e_phnum`, `Elf32_Ehdr::e_phoff`, `EARLY_PTR_TO_PHYS_ADDR`, `elf_check()`, `elf_setup_stack()`, `elf_info_t::entry`, `Elf32_Phdr::p_filesz`, `Elf32_Phdr::p_memsz`, `Elf32_Phdr::p_vaddr`, `PAGE_SIZE`, `panic()`, `PF_W`, `printk()`, `PT_LOAD`, `VM_FLAG_READ_ONLY`, `VM_FLAG_READ_WRITE`, and `vm_map_user()`.

Referenced by `kmain()`.

```

102                                     {
103
104     unsigned int idx;
105
106     /* check that ELF binary is valid */
107     elf_check(elf);
108
109     /* get the program header table */
110     Elf32_Phdr * phdr = (Elf32_Phdr *)((char *)elf + elf->e_phoff);
111
112     info->at_phdr = (addr_t)phdr;
113     info->at_phnum = elf->e_phnum;
114     info->at_phent = elf->e_phentsize;
115     info->addr_space = addr_space;
116     info->entry = (addr_t)elf->e_entry;
117
118     for(idx = 0; idx < elf->e_phnum; ++idx) {
119         if(phdr[idx].p_type != PT_LOAD) {
120             continue;
121         }
122
123         /* check that the segment is not in the region reserved for kernel use */
124         if(! user_buffer_check((void *)phdr[idx].p_vaddr, phdr[idx].p_memsz)) {
125             panic("process manager memory layout -- address of segment too low");
126         }
127
128         /* set start and end addresses for mapping and copying */
129         char *file_ptr = (char *)elf + phdr[idx].p_offset;
130         char *vptr = (char *)phdr[idx].p_vaddr;
131         char *vend = vptr + phdr[idx].p_memsz; /* limit for padding */
132         char *vfend = vptr + phdr[idx].p_filesz; /* limit for copy */
133
134         /* align on page boundaries, be inclusive,
135          note that vfend is not aligned */
136         file_ptr = ALIGN_START_PTR(file_ptr, PAGE_SIZE);
137         vptr = ALIGN_START_PTR(vptr, PAGE_SIZE);
138         vend = ALIGN_END_PTR(vend, PAGE_SIZE);
139
140         /* copy if we have to */
141         if( (phdr[idx].p_flags & PF_W) || (phdr[idx].p_filesz != phdr[idx].p_memsz) ) {
142             while(vptr < vend) {
143                 unsigned long flags;
144                 char *stop;
145
146                 /* start of this page and next page */
147                 char *vnext = vptr + PAGE_SIZE;
148
149                 /* set flags */
150                 if(phdr[idx].p_flags & PF_W) {
151                     flags = VM_FLAG_READ_WRITE;
152                 }
153                 else {
154                     flags = VM_FLAG_READ_ONLY;
155                 }
156
157                 /* allocate and map the new page */
158                 kern_paddr_t page = boot_page_frame_alloc(boot_alloc);
159                 vm_map_user(addr_space, (addr_t)vptr, page, flags);
160
161                 /* copy */
162                 if(vnext > vfend) {
163                     stop = vfend;
164                 }
165                 else {
166                     stop = vnext;
167                 }
168
169                 while(vptr < stop) {
170                     *(vptr++) = *(file_ptr++);
171                 }
172             }

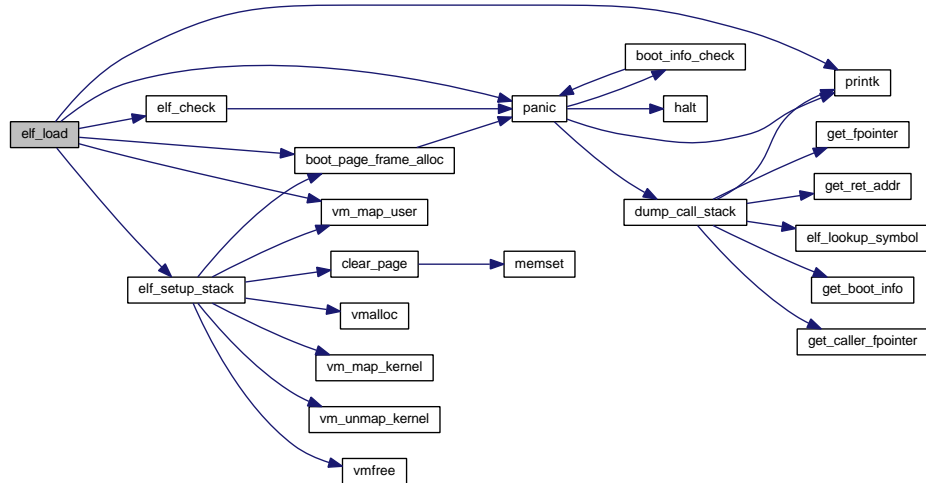
```

```

173
174         /* pad */
175         while(vptr < vnext) {
176             *(vptr++) = 0;
177         }
178     }
179 }
180 else {
181     while(vptr < vend) {
182         /* perform mapping */
183         vm_map_user(addr_space, (addr_t)vptr, EARLY_PTR_TO_PHYS_ADDR(file_ptr),
184 VM_FLAG_READ_ONLY);
185
186         vptr += PAGE_SIZE;
187         file_ptr += PAGE_SIZE;
188     }
189 }
190 }
191 }
192
193 elf_setup_stack(info, boot_alloc);
194
195 printk("ELF binary loaded.\n");
196 }

```

Here is the call graph for this function:



#### 4.11.3.3 int elf\_lookup\_symbol ( const Elf32\_Ehdr \* elf\_header, Elf32\_Addr addr, int type, elf\_symbol\_t \* result )

Definition at line 272 of file elf.c.

References elf\_symbol\_t::addr, Elf32\_Ehdr::e\_shnum, ELF32\_ST\_TYPE, elf\_symbol\_t::name, NULL, Elf32\_Shdr::sh\_entsize, Elf32\_Shdr::sh\_link, Elf32\_Shdr::sh\_offset, Elf32\_Shdr::sh\_size, Elf32\_Shdr::sh\_type, SHT\_SYMTAB, Elf32\_Sym::st\_info, Elf32\_Sym::st\_name, Elf32\_Sym::st\_size, and Elf32\_Sym::st\_value.

Referenced by dump\_call\_stack().

```

276
277     {
278         int idx;
279         size_t symbol_entry_size;
280         size_t symbol_table_size;
281
282         const char *elf_file = elf_file_bytes(elf_header);
283         const char *symbols_table = NULL;
284         const char *string_table = NULL;
285
286         for(idx = 0; idx < elf_header->e_shnum; ++idx) {

```

```

287     const Elf32_Shdr *section_header = elf_get_section_header(elf_header, idx);
288
289     if(section_header->sh_type == SHT_SYMTAB) {
290         symbols_table = &elf_file[section_header->sh_offset];
291         symbol_entry_size = section_header->sh_entsize;
292         symbol_table_size = section_header->sh_size;
293
294         const Elf32_Shdr *string_section_header = elf_get_section_header(
295             elf_header,
296             section_header->sh_link);
297
298         string_table = &elf_file[string_section_header->sh_offset];
299
300         break;
301     }
302 }
303
304 if(symbols_table == NULL) {
305     /* no symbol table */
306     return -1;
307 }
308
309 const char *symbol = symbols_table;
310
311 while(symbol < symbols_table + symbol_table_size) {
312     const Elf32_Sym *symbol_header = (const Elf32_Sym *)symbol;
313
314     if(ELF32_ST_TYPE(symbol_header->st_info) == type) {
315         Elf32_Addr lookup_addr = (Elf32_Addr)addr;
316         Elf32_Addr start = symbol_header->st_value;
317         Elf32_Addr end = start + symbol_header->st_size;
318
319         if(lookup_addr >= start && lookup_addr < end) {
320             result->addr = symbol_header->st_value;
321             result->name = &string_table[symbol_header->st_name];
322
323             return 0;
324         }
325     }
326
327     symbol += symbol_entry_size;
328 }
329
330 /* not found */
331 return -1;
332 }

```

#### 4.11.3.4 void elf\_setup\_stack ( elf\_info\_t \* info, boot\_alloc\_t \* boot\_alloc )

TODO: check for overlap of stack with loaded segments

Definition at line 198 of file elf.c.

References Elf32\_auxv\_t::a\_type, Elf32\_auxv\_t::a\_un, Elf32\_auxv\_t::a\_val, elf\_info\_t::addr\_space, AT\_ENTRY, AT\_NULL, AT\_PAGESZ, elf\_info\_t::at\_phdr, AT\_PHDR, elf\_info\_t::at\_phent, AT\_PHENT, elf\_info\_t::at\_phnum, AT\_PHNUM, AT\_STACKBASE, boot\_page\_frame\_alloc(), clear\_page(), elf\_info\_t::entry, PAGE\_SIZE, elf\_info\_t::stack\_addr, STACK\_BASE, STACK\_START, VM\_FLAG\_READ\_WRITE, vm\_map\_kernel(), vm\_map\_user(), vm\_unmap\_kernel(), vmalloc(), and vmfree().

Referenced by elf\_load().

```

198
199     kern_paddr_t page;
200     addr_t vpage;
201
202     /* initial stack allocation */
203     for(vpage = (addr_t)STACK_START; vpage < (addr_t)STACK_BASE; vpage +=
204         PAGE_SIZE) {
205         page = boot_page_frame_alloc(boot_alloc);
206         vm_map_user(info->addr_space, vpage, page, VM_FLAG_READ_WRITE);
207
208         /* This newly allocated page may have data left from a previous boot which
209          * may contain sensitive information. Let's clear it. */
210

```

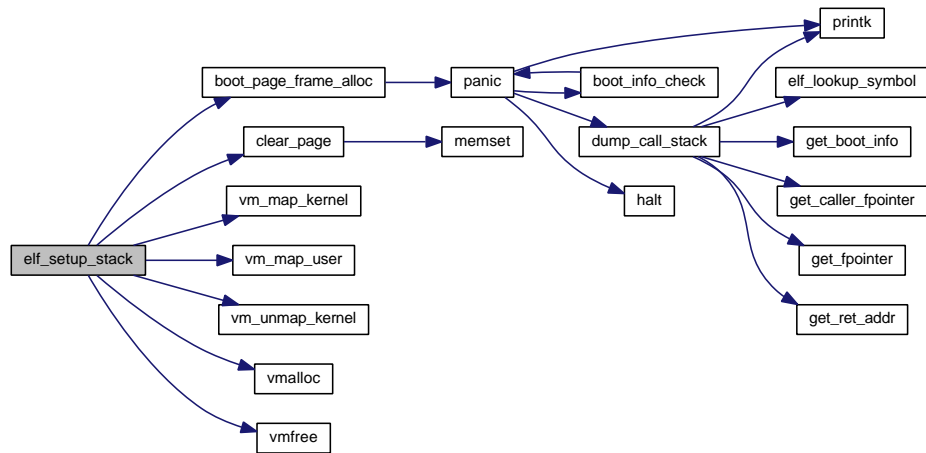
```

211     clear_page(vpage);
212 }
213
214 /* At this point, page has the address of the stack's top-most page frame,
215  * which is the one in which we are about to copy the auxiliary vectors. Map
216  * it temporarily in this address space so we can write to it. */
217 addr_t top_page = vmalloc();
218 vm_map_kernel(top_page, page, VM_FLAG_READ_WRITE);
219
220 /* start at the top */
221 uint32_t *sp = (uint32_t *) (top_page + PAGE_SIZE);
222
223 /* Program name string: "proc", null-terminated */
224 *(--sp) = 0;
225 *(--sp) = 0x636f7270;
226
227 char *argv0 = (char *)STACK_BASE - 2 * sizeof(uint32_t);
228
229 /* auxiliary vectors */
230 Elf32_auxv_t *auxvp = (Elf32_auxv_t *)sp - 7;
231
232 auxvp[0].a_type = AT_PHDR;
233 auxvp[0].a_un.a_val = (int32_t)info->at_phdr;
234
235 auxvp[1].a_type = AT_PHENT;
236 auxvp[1].a_un.a_val = (int32_t)info->at_phent;
237
238 auxvp[2].a_type = AT_PHNUM;
239 auxvp[2].a_un.a_val = (int32_t)info->at_phnum;
240
241 auxvp[3].a_type = AT_PAGESZ;
242 auxvp[3].a_un.a_val = PAGE_SIZE;
243
244 auxvp[4].a_type = AT_ENTRY;
245 auxvp[4].a_un.a_val = (int32_t)info->entry;
246
247 auxvp[5].a_type = AT_STACKBASE;
248 auxvp[5].a_un.a_val = STACK_BASE;
249
250 auxvp[6].a_type = AT_NULL;
251 auxvp[6].a_un.a_val = 0;
252
253 sp = (uint32_t *)auxvp;
254
255 /* empty environment variables */
256 *(--sp) = 0;
257
258 /* argv with only program name */
259 *(--sp) = 0;
260 *(--sp) = (uint32_t)argv0;
261
262 /* argc */
263 *(--sp) = 1;
264
265 info->stack_addr = (addr_t)STACK_BASE - PAGE_SIZE + ((addr_t)sp - top_page);
266
267 /* unmap and free temporary page */
268 vm_unmap_kernel(top_page);
269 vmfree(top_page);
270 }

```



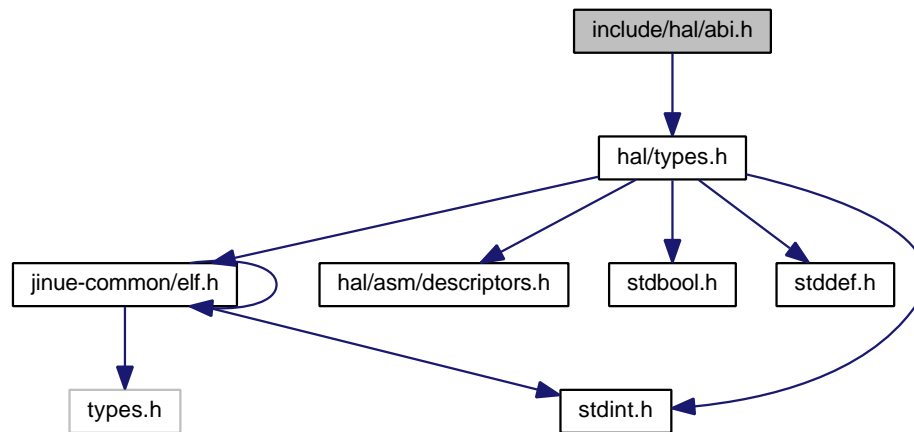
Here is the call graph for this function:



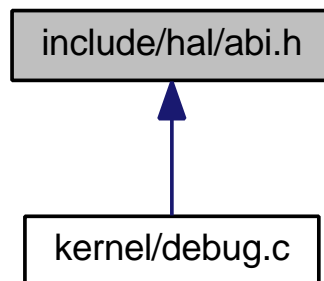
## 4.12 include/hal/abi.h File Reference

```
#include <hal/types.h>
```

Include dependency graph for abi.h:



This graph shows which files directly or indirectly include this file:



## Functions

- **addr\_t get\_fpointer** (void)
- **addr\_t get\_caller\_fpointer** (addr\_t fptr)
- **addr\_t get\_ret\_addr** (addr\_t fptr)
- **addr\_t get\_program\_counter** (void)

### 4.12.1 Function Documentation

#### 4.12.1.1 **addr\_t get\_caller\_fpointer** ( addr\_t fptr )

Referenced by dump\_call\_stack().

#### 4.12.1.2 **addr\_t get\_fpointer** ( void )

Referenced by dump\_call\_stack().

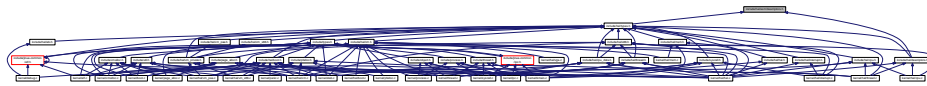
#### 4.12.1.3 **addr\_t get\_program\_counter** ( void )

#### 4.12.1.4 **addr\_t get\_ret\_addr** ( addr\_t fptr )

Referenced by dump\_call\_stack().

## 4.13 include/hal/asm/descriptors.h File Reference

This graph shows which files directly or indirectly include this file:



## Macros

- **#define SEG\_SELECTOR**(index, rpl) ( ((index) << 3) | ((rpl) & 0x3) )
- **#define RPL\_KERNEL** 0
- **#define RPL\_USER** 3
- **#define GDT\_NULL** 0  
*GDT entry for the null descriptor.*
- **#define GDT\_KERNEL\_CODE** 1  
*GDT entry for kernel code segment.*
- **#define GDT\_KERNEL\_DATA** 2  
*GDT entry for kernel data segment.*
- **#define GDT\_USER\_CODE** 3  
*GDT entry for user code segment.*
- **#define GDT\_USER\_DATA** 4  
*GDT entry for user data segment.*
- **#define GDT\_TSS** 5

- GDT entry for task-state segment (TSS)*
- #define **GDT\_PER\_CPU\_DATA** 6
  - GDT entry for per-cpu data (includes the TSS)*
- #define **GDT\_USER\_TLS\_DATA** 7
  - GDT entry for thread-local storage.*
- #define **GDT\_LENGTH** 8
  - number of descriptors in GDT*
- #define **SEG\_FLAGS\_OFFSET** 40
  - offset of descriptor type in descriptor*
- #define **TSS\_LIMIT** 104
  - size of the task-state segment (TSS)*
- #define **SEG\_FLAG\_PRESENT** (1<<7)
  - segment is present*
- #define **SEG\_FLAG\_SYSTEM** 0
  - system segment (i.e.*
- #define **SEG\_FLAG\_NOSYSTEM** (1<<4)
  - code/data/stack segment*
- #define **SEG\_FLAG\_32BIT** (1<<14)
  - 32-bit segment*
- #define **SEG\_FLAG\_16BIT** 0
  - 16-bit segment*
- #define **SEG\_FLAG\_32BIT\_GATE** (1<<3)
  - 32-bit gate*
- #define **SEG\_FLAG\_16BIT\_GATE** 0
  - 16-bit gate*
- #define **SEG\_FLAG\_BUSY** (1<<1)
  - task is busy (for TSS descriptor)*
- #define **SEG\_FLAG\_IN\_PAGES** (1<<15)
  - limit has page granularity*
- #define **SEG\_FLAG\_IN\_BYTES** 0
  - limit has byte granularity*
- #define **SEG\_FLAG\_KERNEL** 0
  - kernel/supervisor segment (privilege level 0)*
- #define **SEG\_FLAG\_USER** (3<<5)
  - user segment (privilege level 3)*
- #define **SEG\_FLAG\_NORMAL** (SEG\_FLAG\_32BIT | SEG\_FLAG\_IN\_PAGES | SEG\_FLAG\_NOSYSTEM | SEG\_FLAG\_PRESENT)
  - commonly used segment flags*
- #define **SEG\_FLAG\_NORMAL\_GATE** (SEG\_FLAG\_32BIT\_GATE | SEG\_FLAG\_SYSTEM | SEG\_FLAG\_PRESENT)
  - commonly used gate flags*
- #define **SEG\_FLAG\_TSS** (SEG\_FLAG\_IN\_BYTES | SEG\_FLAG\_SYSTEM | SEG\_FLAG\_PRESENT)
  - commonly used flags for task-state segment*
- #define **SEG\_TYPE\_READ\_ONLY** 0
  - read-only data segment*
- #define **SEG\_TYPE\_DATA** 2
  - read/write data segment*

- **#define SEG\_TYPE\_TASK\_GATE 5**  
*task gate*
- **#define SEG\_TYPE\_INTERRUPT\_GATE 6**  
*interrupt gate*
- **#define SEG\_TYPE\_TRAP\_GATE 7**  
*trap gate*
- **#define SEG\_TYPE\_TSS 9**  
*task-state segment (TSS)*
- **#define SEG\_TYPE\_CODE 10**  
*code segment*
- **#define SEG\_TYPE\_CALL\_GATE 12**  
*call gate*

### 4.13.1 Macro Definition Documentation

#### 4.13.1.1 **#define GDT\_KERNEL\_CODE 1**

GDT entry for kernel code segment.

Definition at line 46 of file descriptors.h.

Referenced by `cpu_init_data()`.

#### 4.13.1.2 **#define GDT\_KERNEL\_DATA 2**

GDT entry for kernel data segment.

Definition at line 49 of file descriptors.h.

Referenced by `cpu_init_data()`.

#### 4.13.1.3 **#define GDT\_LENGTH 8**

number of descriptors in GDT

Definition at line 67 of file descriptors.h.

#### 4.13.1.4 **#define GDT\_NULL 0**

GDT entry for the null descriptor.

Definition at line 43 of file descriptors.h.

Referenced by `cpu_init_data()`.

#### 4.13.1.5 **#define GDT\_PER\_CPU\_DATA 6**

GDT entry for per-cpu data (includes the TSS)

Definition at line 61 of file descriptors.h.

Referenced by `cpu_init_data()`.

#### 4.13.1.6 `#define GDT_TSS 5`

GDT entry for task-state segment (TSS)

Definition at line 58 of file descriptors.h.

Referenced by `cpu_init_data()`.

#### 4.13.1.7 `#define GDT_USER_CODE 3`

GDT entry for user code segment.

Definition at line 52 of file descriptors.h.

Referenced by `cpu_init_data()`, and `thread_page_init()`.

#### 4.13.1.8 `#define GDT_USER_DATA 4`

GDT entry for user data segment.

Definition at line 55 of file descriptors.h.

Referenced by `cpu_init_data()`, and `thread_page_init()`.

#### 4.13.1.9 `#define GDT_USER_TLS_DATA 7`

GDT entry for thread-local storage.

Definition at line 64 of file descriptors.h.

Referenced by `cpu_init_data()`.

#### 4.13.1.10 `#define RPL_KERNEL 0`

Definition at line 38 of file descriptors.h.

Referenced by `cpu_init_data()`.

#### 4.13.1.11 `#define RPL_USER 3`

Definition at line 40 of file descriptors.h.

Referenced by `thread_page_init()`.

#### 4.13.1.12 `#define SEG_FLAG_16BIT 0`

16-bit segment

Definition at line 88 of file descriptors.h.

#### 4.13.1.13 `#define SEG_FLAG_16BIT_GATE 0`

16-bit gate

Definition at line 94 of file descriptors.h.

**4.13.1.14 #define SEG\_FLAG\_32BIT (1<<14)**

32-bit segment

Definition at line 85 of file descriptors.h.

Referenced by `cpu_init_data()`.

**4.13.1.15 #define SEG\_FLAG\_32BIT\_GATE (1<<3)**

32-bit gate

Definition at line 91 of file descriptors.h.

**4.13.1.16 #define SEG\_FLAG\_BUSY (1<<1)**

task is busy (for TSS descriptor)

Definition at line 97 of file descriptors.h.

**4.13.1.17 #define SEG\_FLAG\_IN\_BYTES 0**

limit has byte granularity

Definition at line 103 of file descriptors.h.

Referenced by `cpu_init_data()`.

**4.13.1.18 #define SEG\_FLAG\_IN\_PAGES (1<<15)**

limit has page granularity

Definition at line 100 of file descriptors.h.

**4.13.1.19 #define SEG\_FLAG\_KERNEL 0**

kernel/supervisor segment (privilege level 0)

Definition at line 106 of file descriptors.h.

Referenced by `cpu_init_data()`.

**4.13.1.20 #define SEG\_FLAG\_NORMAL (SEG\_FLAG\_32BIT | SEG\_FLAG\_IN\_PAGES | SEG\_FLAG\_NOSYSTEM | SEG\_FLAG\_PRESENT)**

commonly used segment flags

Definition at line 112 of file descriptors.h.

Referenced by `cpu_init_data()`.

**4.13.1.21 #define SEG\_FLAG\_NORMAL\_GATE (SEG\_FLAG\_32BIT\_GATE | SEG\_FLAG\_SYSTEM | SEG\_FLAG\_PRESENT)**

commonly used gate flags

Definition at line 116 of file descriptors.h.

#### 4.13.1.22 `#define SEG_FLAG_NOSYSTEM (1<<4)`

code/data/stack segment

Definition at line 82 of file descriptors.h.

Referenced by `cpu_init_data()`.

#### 4.13.1.23 `#define SEG_FLAG_PRESENT (1<<7)`

segment is present

Definition at line 76 of file descriptors.h.

Referenced by `cpu_init_data()`.

#### 4.13.1.24 `#define SEG_FLAG_SYSTEM 0`

system segment (i.e.

call-gate, etc.)

Definition at line 79 of file descriptors.h.

#### 4.13.1.25 `#define SEG_FLAG_TSS (SEG_FLAG_IN_BYTES | SEG_FLAG_SYSTEM | SEG_FLAG_PRESENT)`

commonly used flags for task-state segment

Definition at line 120 of file descriptors.h.

Referenced by `cpu_init_data()`.

#### 4.13.1.26 `#define SEG_FLAG_USER (3<<5)`

user segment (privilege level 3)

Definition at line 109 of file descriptors.h.

Referenced by `cpu_init_data()`.

#### 4.13.1.27 `#define SEG_FLAGS_OFFSET 40`

offset of descriptor type in descriptor

Definition at line 70 of file descriptors.h.

#### 4.13.1.28 `#define SEG_SELECTOR( index, rpl ) (((index) << 3) | ((rpl) & 0x3))`

Definition at line 35 of file descriptors.h.

Referenced by `cpu_init_data()`, and `thread_page_init()`.

**4.13.1.29 #define SEG\_TYPE\_CALL\_GATE 12**

call gate

Definition at line 146 of file descriptors.h.

**4.13.1.30 #define SEG\_TYPE\_CODE 10**

code segment

Definition at line 143 of file descriptors.h.

Referenced by cpu\_init\_data().

**4.13.1.31 #define SEG\_TYPE\_DATA 2**

read/write data segment

Definition at line 128 of file descriptors.h.

Referenced by cpu\_init\_data().

**4.13.1.32 #define SEG\_TYPE\_INTERRUPT\_GATE 6**

interrupt gate

Definition at line 134 of file descriptors.h.

**4.13.1.33 #define SEG\_TYPE\_READ\_ONLY 0**

read-only data segment

Definition at line 125 of file descriptors.h.

**4.13.1.34 #define SEG\_TYPE\_TASK\_GATE 5**

task gate

Definition at line 131 of file descriptors.h.

**4.13.1.35 #define SEG\_TYPE\_TRAP\_GATE 7**

trap gate

Definition at line 137 of file descriptors.h.

**4.13.1.36 #define SEG\_TYPE\_TSS 9**

task-state segment (TSS)

Definition at line 140 of file descriptors.h.

Referenced by cpu\_init\_data().



## 4.13.1.37 #define TSS\_LIMIT 104

size of the task-state segment (TSS)

Definition at line 73 of file descriptors.h.

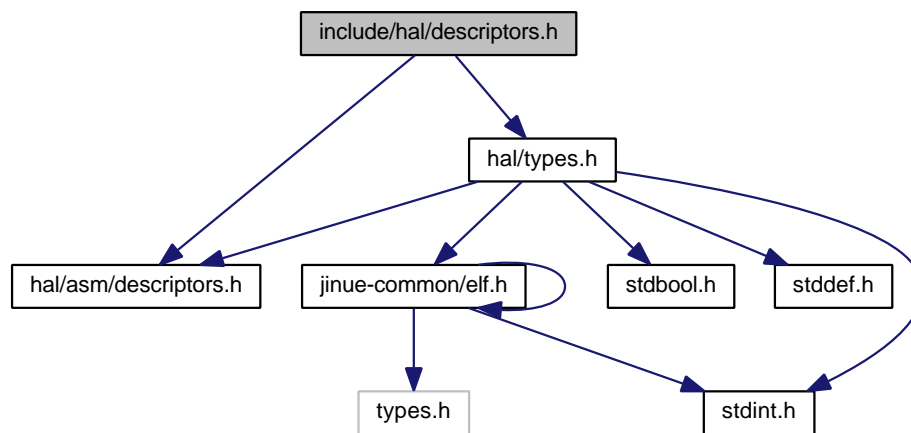
Referenced by cpu\_init\_data().

## 4.14 include/hal/descriptors.h File Reference

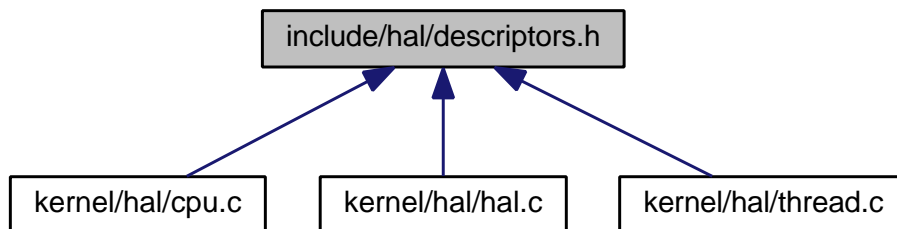
```
#include <hal/asm/descriptors.h>
```

```
#include <hal/types.h>
```

Include dependency graph for descriptors.h:



This graph shows which files directly or indirectly include this file:



## Macros

- #define **PACK\_DESCRIPTOR**(val, mask, shamt1, shamt2) ( (((uint64\_t)(uintptr\_t)(val)) >> shamt1) & mask << shamt2 )
- #define **SEG\_DESCRIPTOR**(base, limit, type)
- #define **GATE\_DESCRIPTOR**(segment, offset, type, param\_count)

### 4.14.1 Macro Definition Documentation

#### 4.14.1.1 `#define GATE_DESCRIPTOR( segment, offset, type, param_count )`

**Value:**

```
(
    PACK_DESCRIPTOR((type),      0xff,    0,  SEG_FLAGS_OFFSET) \
| PACK_DESCRIPTOR((param_count), 0xf,    0,  32) \
| PACK_DESCRIPTOR((segment),    0xffff,  0,  16) \
| PACK_DESCRIPTOR((offset),     0xffff, 16,  48) \
| PACK_DESCRIPTOR((offset),     0xffff,  0,   0) \
)
```

Definition at line 52 of file descriptors.h.

#### 4.14.1.2 `#define PACK_DESCRIPTOR( val, mask, shamt1, shamt2 ) (((uint64_t)(uintptr_t)(val) >> shamt1) & mask) << shamt2 )`

Definition at line 40 of file descriptors.h.

#### 4.14.1.3 `#define SEG_DESCRIPTOR( base, limit, type )`

**Value:**

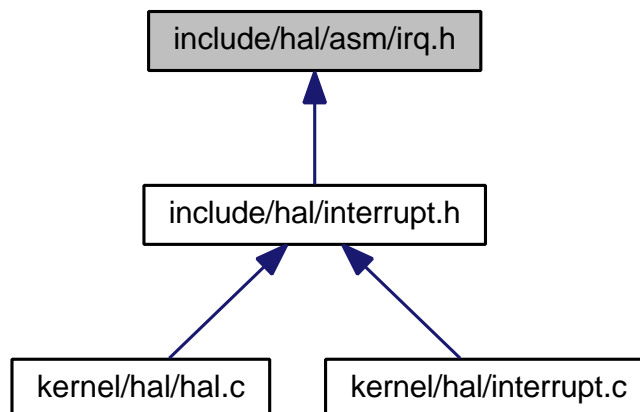
```
(
    PACK_DESCRIPTOR((type), 0xf0ff, 0, SEG_FLAGS_OFFSET) \
| PACK_DESCRIPTOR((base),  0xff,   24, 56) \
| PACK_DESCRIPTOR((base),  0xff,   16, 32) \
| PACK_DESCRIPTOR((base),  0xffff,  0,  16) \
| PACK_DESCRIPTOR((limit), 0xf,    16, 48) \
| PACK_DESCRIPTOR((limit), 0xffff,  0,   0) \
)
```

Definition at line 43 of file descriptors.h.

Referenced by `cpu_init_data()`.

## 4.15 `include/hal/asm/irq.h` File Reference

This graph shows which files directly or indirectly include this file:



## Macros

- **#define IDT\_VECTOR\_COUNT** 256
- **#define IDT\_LAST\_EXCEPTION** 31
- **#define IDT\_PIC8259\_BASE** (IDT\_LAST\_EXCEPTION + 1)
- **#define EXCEPTION\_DIV\_ZERO** 0  
*Divide Error.*
- **#define EXCEPTION\_NMI** 2  
*NMI Interrupt.*
- **#define EXCEPTION\_BREAK** 3  
*Breakpoint.*
- **#define EXCEPTION\_OVERFLOW** 4  
*Overflow.*
- **#define EXCEPTION\_BOUND** 5  
*BOUND Range Exceeded.*
- **#define EXCEPTION\_INVALID\_OP** 6  
*Invalid Opcode (Undefined Opcode)*
- **#define EXCEPTION\_NO\_COPROC** 7  
*Device Not Available (No Math Coprocessor)*
- **#define EXCEPTION\_DOUBLE\_FAULT** 8  
*Double Fault.*
- **#define EXCEPTION\_INVALID\_TSS** 10  
*Invalid TSS.*
- **#define EXCEPTION\_SEGMENT\_NOT\_PRESENT** 11  
*Segment Not Present.*
- **#define EXCEPTION\_STACK\_SEGMENT** 12  
*Stack-Segment Fault.*
- **#define EXCEPTION\_GENERAL\_PROTECTION** 13  
*General Protection.*
- **#define EXCEPTION\_PAGE\_FAULT** 14  
*Page Fault.*
- **#define EXCEPTION\_MATH** 16  
*x87 FPU Floating-Point Error (Math Fault)*
- **#define EXCEPTION\_ALIGNMENT** 17  
*Alignment Check.*
- **#define EXCEPTION\_MACHINE\_CHECK** 18  
*Machine Check.*
- **#define EXCEPTION\_SIMD** 19  
*SIMD Floating-Point Exception.*
- **#define HAS\_ERRCODE(x)** ((x) == EXCEPTION\_DOUBLE\_FAULT || (x) == EXCEPTION\_ALIGNMENT || ((x) >= EXCEPTION\_INVALID\_TSS && (x) <= EXCEPTION\_PAGE\_FAULT))

### 4.15.1 Macro Definition Documentation

#### 4.15.1.1 #define EXCEPTION\_ALIGNMENT 17

Alignment Check.

Definition at line 84 of file irq.h.

**4.15.1.2 #define EXCEPTION\_BOUND 5**

BOUND Range Exceeded.

Definition at line 54 of file irq.h.

**4.15.1.3 #define EXCEPTION\_BREAK 3**

Breakpoint.

Definition at line 48 of file irq.h.

**4.15.1.4 #define EXCEPTION\_DIV\_ZERO 0**

Divide Error.

Definition at line 42 of file irq.h.

**4.15.1.5 #define EXCEPTION\_DOUBLE\_FAULT 8**

Double Fault.

Definition at line 63 of file irq.h.

**4.15.1.6 #define EXCEPTION\_GENERAL\_PROTECTION 13**

General Protection.

Definition at line 75 of file irq.h.

**4.15.1.7 #define EXCEPTION\_INVALID\_OP 6**

Invalid Opcode (Undefined Opcode)

Definition at line 57 of file irq.h.

**4.15.1.8 #define EXCEPTION\_INVALID\_TSS 10**

Invalid TSS.

Definition at line 66 of file irq.h.

**4.15.1.9 #define EXCEPTION\_MACHINE\_CHECK 18**

Machine Check.

Definition at line 87 of file irq.h.

**4.15.1.10 #define EXCEPTION\_MATH 16**

x87 FPU Floating-Point Error (Math Fault)

Definition at line 81 of file irq.h.

**4.15.1.11 #define EXCEPTION\_NMI 2**

NMI Interrupt.

Definition at line 45 of file irq.h.

**4.15.1.12 #define EXCEPTION\_NO\_COPROC 7**

Device Not Available (No Math Coprocessor)

Definition at line 60 of file irq.h.

**4.15.1.13 #define EXCEPTION\_OVERFLOW 4**

Overflow.

Definition at line 51 of file irq.h.

**4.15.1.14 #define EXCEPTION\_PAGE\_FAULT 14**

Page Fault.

Definition at line 78 of file irq.h.

**4.15.1.15 #define EXCEPTION\_SEGMENT\_NOT\_PRESENT 11**

Segment Not Present.

Definition at line 69 of file irq.h.

**4.15.1.16 #define EXCEPTION\_SIMD 19**

SIMD Floating-Point Exception.

Definition at line 90 of file irq.h.

**4.15.1.17 #define EXCEPTION\_STACK\_SEGMENT 12**

Stack-Segment Fault.

Definition at line 72 of file irq.h.

**4.15.1.18 #define HAS\_ERRCODE( x ) ((x) == EXCEPTION\_DOUBLE\_FAULT || (x) == EXCEPTION\_ALIGNMENT || ((x) >= EXCEPTION\_INVALID\_TSS && (x) <= EXCEPTION\_PAGE\_FAULT))**

Definition at line 92 of file irq.h.

**4.15.1.19 #define IDT\_LAST\_EXCEPTION 31**

Definition at line 37 of file irq.h.

Referenced by dispatch\_interrupt().



4.16.1.2 `#define MEM_ZONE_DMA16_START (1 * MB)`

Definition at line 37 of file mem.h.

Referenced by `mem_check_memory()`.

4.16.1.3 `#define MEM_ZONE_MEM32_END 0xc0000000`

Definition at line 43 of file mem.h.

Referenced by `mem_check_memory()`.

4.16.1.4 `#define MEM_ZONE_MEM32_START (16 * MB)`

Definition at line 41 of file mem.h.

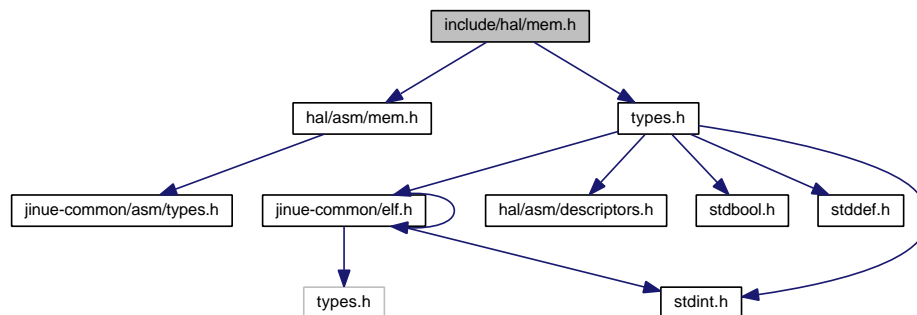
Referenced by `mem_check_memory()`.

## 4.17 include/hal/mem.h File Reference

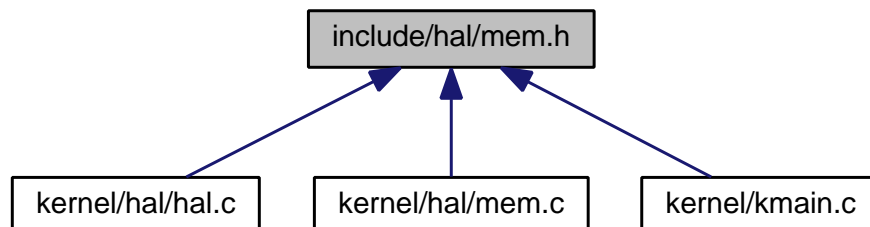
```
#include <hal/asm/mem.h>
```

```
#include <types.h>
```

Include dependency graph for mem.h:



This graph shows which files directly or indirectly include this file:



## Functions

- void `mem_check_memory` (`boot_alloc_t *boot_alloc`, const `boot_info_t *boot_info`)

## 4.17.1 Function Documentation

### 4.17.1.1 void mem\_check\_memory ( boot\_alloc\_t \* boot\_alloc, const boot\_info\_t \* boot\_info )

ASSERTION: Any unsigned value less than MEM\_ZONE\_MEM32\_END can be stored in 32 bits.

Definition at line 57 of file mem.c.

References e820\_t::addr, assert, boot\_info\_t::boot\_end, boot\_info\_t::e820\_entries, boot\_info\_t::e820\_map, E820\_RAM, EARLY\_VIRT\_TO\_PHYS, GB, KERNEL\_EARLY\_LIMIT, boot\_alloc\_t::kernel\_paddr\_limit, boot\_alloc\_t::kernel\_paddr\_top, boot\_alloc\_t::kernel\_vm\_limit, boot\_alloc\_t::kernel\_vm\_top, MEM\_ZONE\_DMA16\_END, MEM\_ZONE\_DMA16\_START, MEM\_ZONE\_MEM32\_END, MEM\_ZONE\_MEM32\_START, panic(), boot\_info\_t::ramdisk\_size, boot\_info\_t::ramdisk\_start, and e820\_t::type.

Referenced by kmain().

```

57                                     {
58     int idx;
59
60     #if 0
61         if((uint64_t)boot_info->ramdisk_start + boot_info->ramdisk_size >
        MEM_ZONE_MEM32_END) {
62             panic("Initial RAM disk loaded too high in memory.");
63         }
64
65         uint32_t ramdisk_end = boot_info->ramdisk_start + boot_info->ramdisk_size;
66     #endif
67
68     /* -----
69     * We consult the memory map provided by the BIOS to figure out how much
70     * memory is available in both zones usable by the kernel. We also want to
71     * make sure the initial RAM disk image is in available RAM.
72     *
73     * The first step in accomplishing this is to iterate over all entries that
74     * are reported as available RAM and confirm at least one of them covers
75     * each of both zones (at least the start) and the initial RAM disk.
76     * ----- */
77     uint32_t zone_dmal6_top = 0;
78     uint32_t zone_mem32_top = 0;
79     #if 0
80         bool ramdisk_ok = false;
81     #endif
82
83     assert(MEM_ZONE_MEM32_END < (uint64_t)4 * GB);
84
85     for(idx = 0; idx < boot_info->e820_entries; ++idx) {
86         const e820_t *entry = &boot_info->e820_map[idx];
87
88         /* Consider only usable RAM entries. */
89         if(entry->type != E820_RAM) {
90             continue;
91         }
92
93         /* Ignore entries that start past MEM_ZONE_MEM32_END since the kernel
94         * cannot use them. Past this check, entry->addr is assumed to be
95         * representable in 32 bits. */
96         if(entry->addr >= MEM_ZONE_MEM32_END) {
97             continue;
98         }
99
100         uint32_t entry_end = clip_e820_entry_end(entry);
101
102         /* If this entry covers the start the DMA16 zone, adjust the top pointer
103         * accordingly. Overlapping entries are resolved in favor of the largest
104         * entry. */
105         if(entry->addr <= MEM_ZONE_DMA16_START && entry_end >
        MEM_ZONE_DMA16_START) {
106             /* This condition covers the initial case where zone_dmal6_top is zero. */
107             if(entry_end > zone_dmal6_top) {
108                 zone_dmal6_top = entry_end;
109             }
110         }
111
112         /* Do the same for the MEM32 zone. */
113         if(entry->addr <= MEM_ZONE_MEM32_START && entry_end >
        MEM_ZONE_MEM32_START) {

```



```

115         if(entry_end > zone_mem32_top) {
116             zone_mem32_top = entry_end;
117         }
118     }
119
120     /* If this entry covers the initial RAM disk, this is good argument in
121     * favor of it being in available RAM (one more check below). Unlike the
122     * above, the entry must cover the initial RAM disk image completely,
123     * not just the start. */
124 #if 0
125     if(entry->addr <= boot_info->ramdisk_start && entry_end >= ramdisk_end) {
126         ramdisk_ok = true;
127     }
128 #endif
129 }
130
131 /* -----
132  * Next, iterate over non-available RAM entries of the map to ensure nothing
133  * is relevant there.
134  * ----- */
135 for(idx = 0; idx < boot_info->e820_entries; ++idx) {
136     const e820_t *entry = &boot_info->e820_map[idx];
137
138     /* Consider only non-usable RAM entries. */
139     if(entry->type == E820_RAM) {
140         continue;
141     }
142
143     if(entry->addr >= MEM_ZONE_MEM32_END) {
144         continue;
145     }
146
147     uint32_t entry_end = clip_e820_entry_end(entry);
148
149     if(ranges_overlap(MEM_ZONE_DMA16_START, MEM_ZONE_DMA16_END, entry->addr, entry_end)) {
150         if(entry->addr > MEM_ZONE_DMA16_START) {
151             if(entry->addr < zone_dmal6_top) {
152                 zone_dmal6_top = entry->addr;
153             }
154         }
155         else {
156             /* This reserved entry covers the start of the zone. */
157             zone_dmal6_top = 0;
158         }
159     }
160
161     if(ranges_overlap(MEM_ZONE_MEM32_START, MEM_ZONE_MEM32_END, entry->addr, entry_end)) {
162         if(entry->addr > MEM_ZONE_MEM32_START) {
163             if(entry->addr < zone_mem32_top) {
164                 zone_mem32_top = entry->addr;
165             }
166         }
167         else {
168             zone_mem32_top = 0;
169         }
170     }
171
172     /* Check for overlap with the initial RAM disk. */
173 #if 0
174     if(ranges_overlap(boot_info->ramdisk_start, ramdisk_end, entry->addr, entry_end)) {
175         ramdisk_ok = false;
176     }
177 #endif
178 }
179
180 /* -----
181  * Now that we are done, let's look at the results.
182  * ----- */
183 #if 0
184 if(!ramdisk_ok) {
185     panic("Initial RAM disk was loaded in reserved memory.");
186 }
187 #endif
188
189 /* It is early during the boot process and the page table set up by the
190  * setup code is still being used. This (single) page table maps the first
191  * two megabytes of RAM linearly starting at KLIMIT in the virtual address
192  * space. */
193 boot_alloc->kernel_vm_top = boot_info->boot_end;
194 boot_alloc->kernel_vm_limit = (addr_t)KERNEL_EARLY_LIMIT;
195 boot_alloc->kernel_paddr_top = EARLY_VIRT_TO_PHYS(boot_alloc->

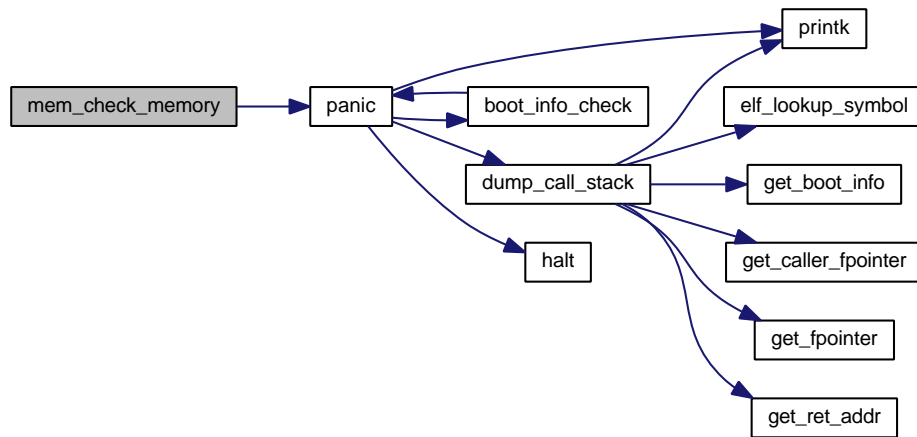
```

```

kernel_vm_top);
196     boot_alloc->kernel_paddr_limit = zone_dma16_top;
197
198     if(boot_alloc->kernel_paddr_top > boot_alloc->kernel_paddr_limit) {
199         panic("Kernel image was loaded in reserved memory.");
200     }
201
202     /* TODO Compute sequential allocation limit taking initrd into account */
203     /* TODO Report zone limits */
204 }

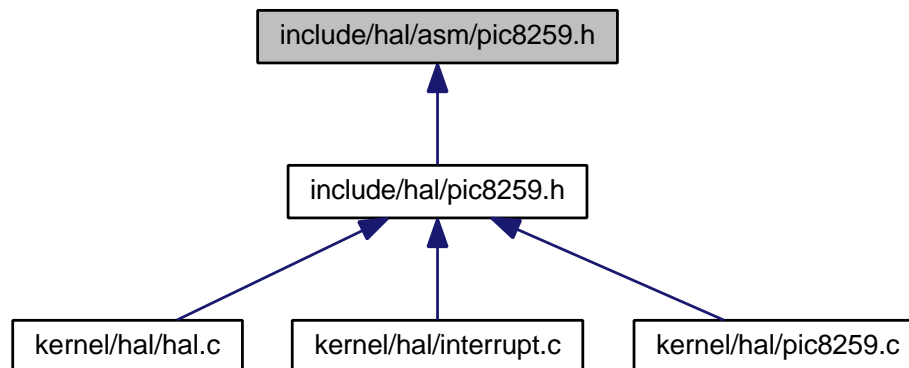
```

Here is the call graph for this function:



#### 4.18 include/hal/asm/pic8259.h File Reference

This graph shows which files directly or indirectly include this file:



#### Macros

- **#define PIC8259\_MASTER\_BASE** 0x20  
*Base I/O port for the master interrupt controller.*
- **#define PIC8259\_SLAVE\_BASE** 0xa0  
*Base I/O port for the slave interrupt controller.*
- **#define PIC8259\_ICW1\_IC4** (1<<0)

- ICW1 bit 0: ICW4 needed.*
- **#define PIC8259\_ICW1\_SNGL** (1<<1)  
*ICW1 bit 1: single (1) or cascade (0) mode.*
- **#define PIC8259\_ICW1\_LTIM** (1<<3)  
*ICW1 bit 3: level-triggered (1) or edge-triggered (0) interrupts.*
- **#define PIC8259\_ICW1\_1** (1<<4)  
*ICW1 bit 4: a control word with this bit set indicates this is ICW1.*
- **#define PIC8259\_ICW4\_UPM** (1<<0)  
*ICW4 bit 0: 8086/8088 mode (1) or MCS-80/85 mode (0)*
- **#define PIC8259\_ICW4\_AEOI** (1<<1)  
*ICW4 bit 1: Auto EOI.*
- **#define PIC8259\_EOI** 0x20  
*OCW2: non-specific EOI command.*
- **#define PIC8259\_CASCADE\_INPUT** 2  
*Slave PIC is connected to input 2 of the master.*
- **#define PIC8259\_IRQ\_COUNT** 16  
*Number of IRQs handled by both cascaded PIC8259s together.*

#### 4.18.1 Macro Definition Documentation

##### 4.18.1.1 #define PIC8259\_CASCADE\_INPUT 2

Slave PIC is connected to input 2 of the master.

Definition at line 63 of file pic8259.h.

Referenced by pic8259\_init(), pic8259\_mask\_irq(), and pic8259\_unmask\_irq().

##### 4.18.1.2 #define PIC8259\_EOI 0x20

OCW2: non-specific EOI command.

Definition at line 60 of file pic8259.h.

Referenced by pic8259\_eoi().

##### 4.18.1.3 #define PIC8259\_ICW1\_1 (1<<4)

ICW1 bit 4: a control word with this bit set indicates this is ICW1.

Definition at line 51 of file pic8259.h.

Referenced by pic8259\_init().

##### 4.18.1.4 #define PIC8259\_ICW1\_IC4 (1<<0)

ICW1 bit 0: ICW4 needed.

Definition at line 42 of file pic8259.h.

Referenced by pic8259\_init().

#### 4.18.1.5 `#define PIC8259_ICW1_LTIM (1<<3)`

ICW1 bit 3: level-triggered (1) or edge-triggered (0) interrupts.

Definition at line 48 of file pic8259.h.

#### 4.18.1.6 `#define PIC8259_ICW1_SNGL (1<<1)`

ICW1 bit 1: single (1) or cascade (0) mode.

Definition at line 45 of file pic8259.h.

#### 4.18.1.7 `#define PIC8259_ICW4_AEOI (1<<1)`

ICW4 bit 1: Auto EOI.

Definition at line 57 of file pic8259.h.

#### 4.18.1.8 `#define PIC8259_ICW4_UPM (1<<0)`

ICW4 bit 0: 8086/8088 mode (1) or MCS-80/85 mode (0)

Definition at line 54 of file pic8259.h.

Referenced by pic8259\_init().

#### 4.18.1.9 `#define PIC8259_IRQ_COUNT 16`

Number of IRQs handled by both cascaded PIC8259s together.

Definition at line 66 of file pic8259.h.

Referenced by dispatch\_interrupt(), pic8259\_eoi(), pic8259\_mask\_irq(), and pic8259\_unmask\_irq().

#### 4.18.1.10 `#define PIC8259_MASTER_BASE 0x20`

Base I/O port for the master interrupt controller.

Definition at line 36 of file pic8259.h.

Referenced by pic8259\_eoi(), pic8259\_init(), pic8259\_mask\_irq(), and pic8259\_unmask\_irq().

#### 4.18.1.11 `#define PIC8259_SLAVE_BASE 0xa0`

Base I/O port for the slave interrupt controller.

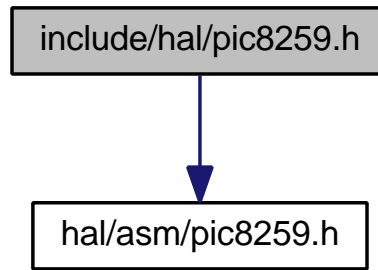
Definition at line 39 of file pic8259.h.

Referenced by pic8259\_eoi(), pic8259\_init(), pic8259\_mask\_irq(), and pic8259\_unmask\_irq().

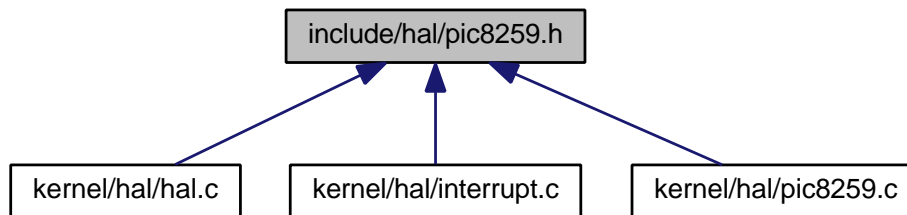
## 4.19 `include/hal/pic8259.h` File Reference

```
#include <hal/asm/pic8259.h>
```

Include dependency graph for pic8259.h:



This graph shows which files directly or indirectly include this file:



## Functions

- void **pic8259\_init** (int intrvect\_base)
- void **pic8259\_mask\_irq** (int irq)
- void **pic8259\_unmask\_irq** (int irq)
- void **pic8259\_eoi** (int irq)

### 4.19.1 Function Documentation

#### 4.19.1.1 void pic8259\_eoi ( int irq )

Definition at line 124 of file pic8259.c.

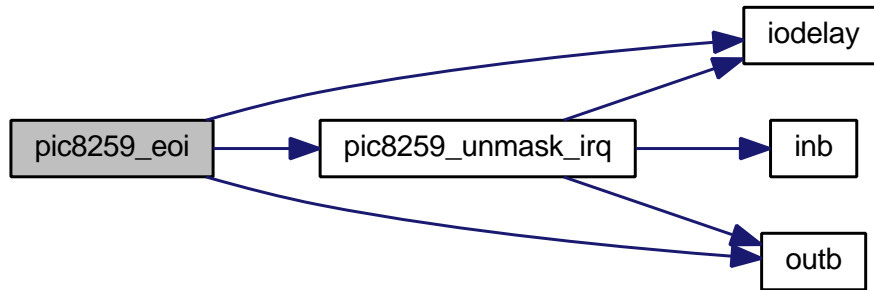
References iodelay(), outb(), PIC8259\_EOI, PIC8259\_IRQ\_COUNT, PIC8259\_MASTER\_BASE, PIC8259\_SLAVE\_BASE, and pic8259\_unmask\_irq().

Referenced by dispatch\_interrupt().

```

124         {
125     if(irq < PIC8259_IRQ_COUNT) {
126         if(irq >= 8) {
127             outb(PIC8259_SLAVE_BASE+0, PIC8259_EOI);
128             iodelay();
129         }
130
131         outb(PIC8259_MASTER_BASE+0, PIC8259_EOI);
132         iodelay();
133
134         pic8259_unmask_irq(irq);
135     }
136 }
  
```

Here is the call graph for this function:



#### 4.19.1.2 void pic8259\_init ( int intrvect\_base )

Definition at line 36 of file pic8259.c.

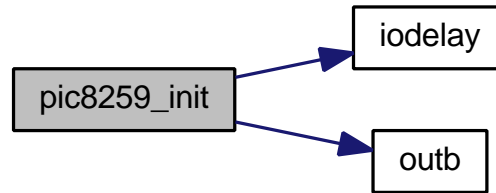
References iodelay(), outb(), PIC8259\_CASCADE\_INPUT, PIC8259\_ICW1\_1, PIC8259\_ICW1\_IC4, PIC8259\_ICW4\_UPM, PIC8259\_MASTER\_BASE, and PIC8259\_SLAVE\_BASE.

Referenced by hal\_init().

```

36      {
37          /* Issue ICW1 to start initialization sequence of both interrupt controllers.
38           * Specify there will be an ICW4 in the sequence. Specify the interrupts are
39           * edge-triggered and that the PICs are in a cascaded configuration by
40           * leaving the relevant flags cleared. */
41          outb(PIC8259_MASTER_BASE+0, PIC8259_ICW1_1 | PIC8259_ICW1_IC4);
42          iodelay();
43          outb(PIC8259_SLAVE_BASE+0, PIC8259_ICW1_1 | PIC8259_ICW1_IC4);
44          iodelay();
45
46          /* ICW2: base interrupt vector */
47          outb(PIC8259_MASTER_BASE+1, intrvect_base);
48          iodelay();
49          outb(PIC8259_SLAVE_BASE+1, intrvect_base + 8);
50          iodelay();
51
52          /* ICW3: master-slave connections */
53          outb(PIC8259_MASTER_BASE+1, (1<<PIC8259_CASCADE_INPUT));
54          iodelay();
55          outb(PIC8259_SLAVE_BASE+1, PIC8259_CASCADE_INPUT);
56          iodelay();
57
58          /* ICW4: Use 8088/8086 mode */
59          outb(PIC8259_MASTER_BASE+1, PIC8259_ICW4_UPM);
60          iodelay();
61          outb(PIC8259_SLAVE_BASE+1, PIC8259_ICW4_UPM);
62          iodelay();
63
64          /* Set interrupt mask: all masked */
65          outb(PIC8259_MASTER_BASE+1, 0xff & ~(1<<PIC8259_CASCADE_INPUT));
66          iodelay();
67          outb(PIC8259_SLAVE_BASE+1, 0xff);
68          iodelay();
69      }
  
```

Here is the call graph for this function:



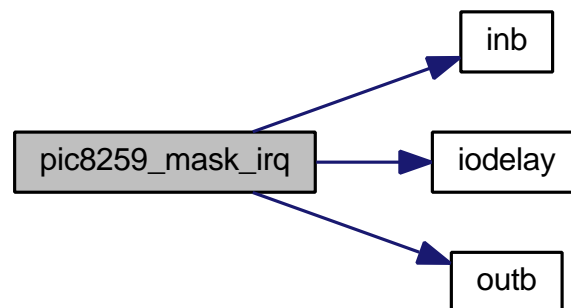
#### 4.19.1.3 void pic8259\_mask\_irq ( int irq )

Definition at line 71 of file `pic8259.c`.

References `inb()`, `iodelay()`, `outb()`, `PIC8259_CASCADE_INPUT`, `PIC8259_IRQ_COUNT`, `PIC8259_MASTER_BASE`, and `PIC8259_SLAVE_BASE`.

```
71     {
72     if(irq < PIC8259_IRQ_COUNT) {
73         if(irq < 8 && irq != PIC8259_CASCADE_INPUT) {
74             int mask = inb(PIC8259_MASTER_BASE+1);
75             iodelay();
76
77             mask |= (1<<irq);
78             outb(PIC8259_MASTER_BASE+1, mask);
79             iodelay();
80         }
81         else {
82             int mask = inb(PIC8259_SLAVE_BASE+1);
83             iodelay();
84
85             mask |= (1<<(irq - 8));
86             outb(PIC8259_SLAVE_BASE+1, mask);
87             iodelay();
88         }
89     }
90 }
```

Here is the call graph for this function:



#### 4.19.1.4 void pic8259\_unmask\_irq ( int irq )

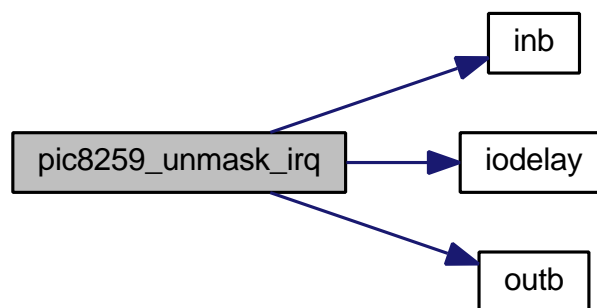
Definition at line 92 of file `pic8259.c`.

References `inb()`, `iodelay()`, `outb()`, `PIC8259_CASCADE_INPUT`, `PIC8259_IRQ_COUNT`, `PIC8259_MASTER_BASE`, and `PIC8259_SLAVE_BASE`.

Referenced by `pic8259_eoi()`.

```
92
93     if(irq < PIC8259_IRQ_COUNT) {
94         int master_irq;
95
96         if(irq < 8) {
97             master_irq = irq;
98         }
99         else {
100             /* Unmask interrupt in slave PIC. */
101             int mask = inb(PIC8259_SLAVE_BASE+1);
102             iodelay();
103
104             mask &= ~(1<<(irq - 8));
105             outb(PIC8259_SLAVE_BASE+1, mask);
106             iodelay();
107
108             /* We will also want to unmask the cascaded interrupt line in the
109              * master PIC. */
110             master_irq = PIC8259_CASCADE_INPUT;
111         }
112
113         if(irq != PIC8259_CASCADE_INPUT) {
114             int mask = inb(PIC8259_MASTER_BASE+1);
115             iodelay();
116
117             mask &= ~(1<<master_irq);
118             outb(PIC8259_MASTER_BASE+1, mask);
119             iodelay();
120         }
121     }
122 }
```

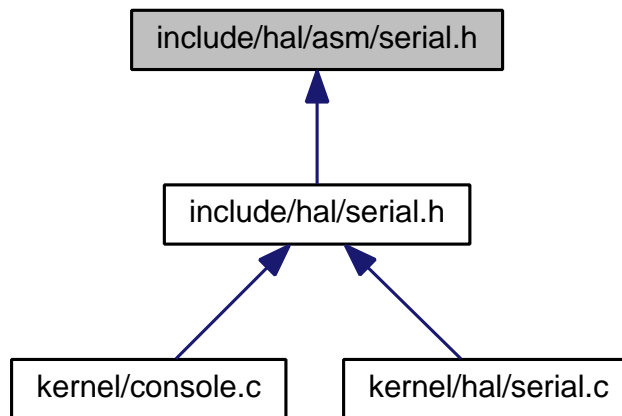
Here is the call graph for this function:





## 4.20 include/hal/asm/serial.h File Reference

This graph shows which files directly or indirectly include this file:



### Macros

- `#define SERIAL_COM1_IOPORT 0x3f8`
- `#define SERIAL_COM2_IOPORT 0x2f8`
- `#define SERIAL_COM3_IOPORT 0x3e8`
- `#define SERIAL_COM4_IOPORT 0x2e8`
- `#define SERIAL_REG_DATA_BUFFER 0`
- `#define SERIAL_REG_DIVISOR_LOW 0`
- `#define SERIAL_REG_INTR_ENABLE 1`
- `#define SERIAL_REG_DIVISOR_HIGH 1`
- `#define SERIAL_REG_INTR_ID 2`
- `#define SERIAL_REG_FIFO_CTRL 2`
- `#define SERIAL_REG_LINE_CTRL 3`
- `#define SERIAL_REG_MODEM_CTRL 4`
- `#define SERIAL_REG_LINE_STATUS 5`
- `#define SERIAL_REG_MODEM_STATUS 6`
- `#define SERIAL_REG_SCRATCH 7`

### 4.20.1 Macro Definition Documentation

#### 4.20.1.1 `#define SERIAL_COM1_IOPORT 0x3f8`

Definition at line 4 of file serial.h.

#### 4.20.1.2 `#define SERIAL_COM2_IOPORT 0x2f8`

Definition at line 6 of file serial.h.

#### 4.20.1.3 `#define SERIAL_COM3_IOPORT 0x3e8`

Definition at line 8 of file serial.h.

4.20.1.4 `#define SERIAL_COM4_IOPORT 0x2e8`

Definition at line 10 of file serial.h.

4.20.1.5 `#define SERIAL_REG_DATA_BUFFER 0`

Definition at line 13 of file serial.h.

Referenced by serial\_putc().

4.20.1.6 `#define SERIAL_REG_DIVISOR_HIGH 1`

Definition at line 19 of file serial.h.

Referenced by serial\_init().

4.20.1.7 `#define SERIAL_REG_DIVISOR_LOW 0`

Definition at line 15 of file serial.h.

Referenced by serial\_init().

4.20.1.8 `#define SERIAL_REG_FIFO_CTRL 2`

Definition at line 23 of file serial.h.

Referenced by serial\_init().

4.20.1.9 `#define SERIAL_REG_INTR_ENABLE 1`

Definition at line 17 of file serial.h.

Referenced by serial\_init().

4.20.1.10 `#define SERIAL_REG_INTR_ID 2`

Definition at line 21 of file serial.h.

4.20.1.11 `#define SERIAL_REG_LINE_CTRL 3`

Definition at line 25 of file serial.h.

Referenced by serial\_init().

4.20.1.12 `#define SERIAL_REG_LINE_STATUS 5`

Definition at line 29 of file serial.h.

Referenced by serial\_putc().

4.20.1.13 `#define SERIAL_REG_MODEM_CTRL 4`

Definition at line 27 of file serial.h.

Referenced by serial\_init().

4.20.1.14 `#define SERIAL_REG_MODEM_STATUS 6`

Definition at line 31 of file serial.h.

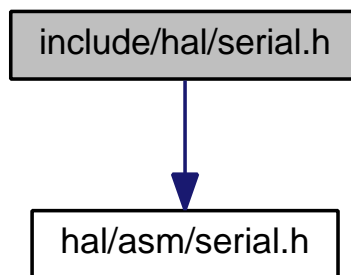
4.20.1.15 `#define SERIAL_REG_SCRATCH 7`

Definition at line 33 of file serial.h.

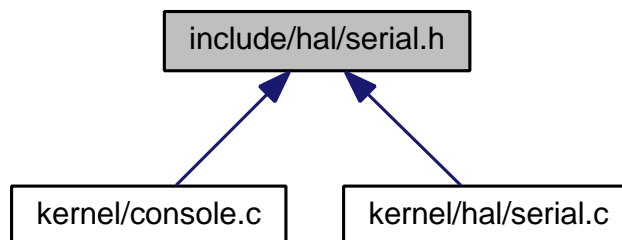
## 4.21 include/hal/serial.h File Reference

```
#include <hal/asm/serial.h>
```

Include dependency graph for serial.h:



This graph shows which files directly or indirectly include this file:



## Functions

- void **serial\_init** (int base\_ioport, unsigned int baud\_rate)
- void **serial\_printn** (int base\_ioport, const char \*message, unsigned int n)
- void **serial\_putc** (int base\_ioport, char c)

## 4.21.1 Function Documentation

### 4.21.1.1 void serial\_init ( int *base\_ioport*, unsigned int *baud\_rate* )

Definition at line 4 of file serial.c.

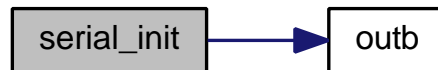
References `outb()`, `SERIAL_REG_DIVISOR_HIGH`, `SERIAL_REG_DIVISOR_LOW`, `SERIAL_REG_FIFO_CTRL`, `SERIAL_REG_INTR_ENABLE`, `SERIAL_REG_LINE_CTRL`, and `SERIAL_REG_MODEM_CTRL`.

Referenced by `console_init()`.

```

4      {
5      unsigned int divisor = 115200 / baud_rate;
6
7      /* disable interrupts */
8      outb(base_ioport + SERIAL_REG_INTR_ENABLE, 0);
9
10     /* 8N1, enable DLAB to allow setting baud rate */
11     outb(base_ioport + SERIAL_REG_LINE_CTRL, 0x83);
12
13     /* set baud rate */
14     outb(base_ioport + SERIAL_REG_DIVISOR_LOW, (divisor & 0xff));
15     outb(base_ioport + SERIAL_REG_DIVISOR_HIGH, ((divisor >> 8) & 0xff));
16
17     /* 8N1, disable DLAB */
18     outb(base_ioport + SERIAL_REG_LINE_CTRL, 0x03);
19
20     /* enable and clear FIFO
21     *
22     * Receive FIFO trigger level is not relevant for us as we are only
23     * transmitting. */
24     outb(base_ioport + SERIAL_REG_FIFO_CTRL, 0x07);
25
26     /* assert DTR and RTS */
27     outb(base_ioport + SERIAL_REG_MODEM_CTRL, 0x03);
28 }
```

Here is the call graph for this function:



### 4.21.1.2 void serial\_printn ( int *base\_ioport*, const char \* *message*, unsigned int *n* )

Definition at line 30 of file serial.c.

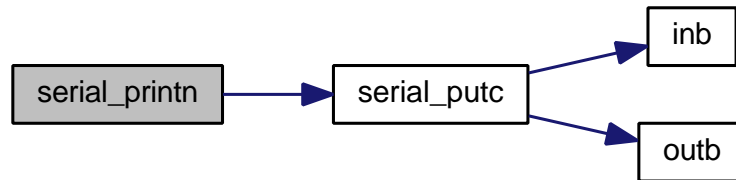
References `serial_putc()`.

Referenced by `console_printn()`.

```

30      {
31      int idx;
32
33      for(idx = 0; idx < n; ++idx) {
34          serial_putc(base_ioport, message[idx]);
35      }
36 }
```

Here is the call graph for this function:



#### 4.21.1.3 void serial\_putc ( int *base\_ioport*, char *c* )

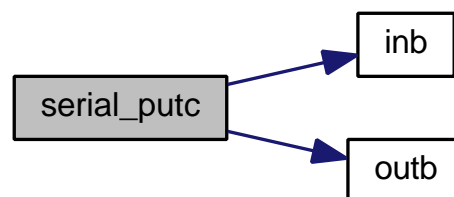
Definition at line 38 of file `serial.c`.

References `inb()`, `outb()`, `SERIAL_REG_DATA_BUFFER`, and `SERIAL_REG_LINE_STATUS`.

Referenced by `console_putc()`, and `serial_printn()`.

```
38      {
39      /* wait for the UART to be ready to accept a new character */
40      while( (inb(base_ioport + SERIAL_REG_LINE_STATUS) & 0x20) == 0) {}
41
42      outb(base_ioport + SERIAL_REG_DATA_BUFFER, c);
43 }
```

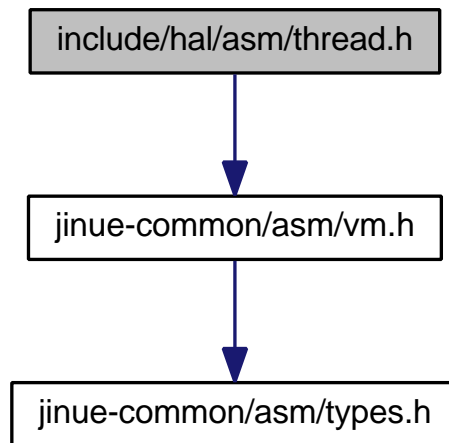
Here is the call graph for this function:



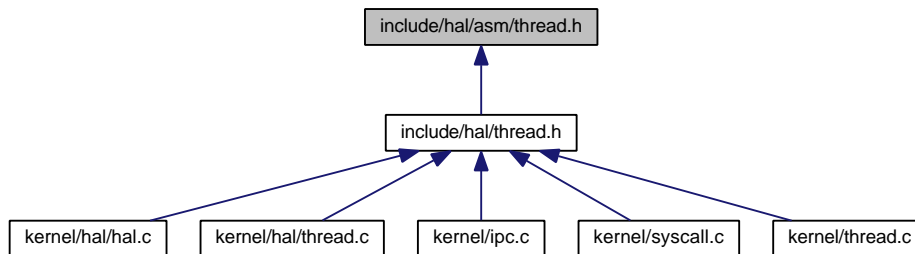
## 4.22 include/hal/asm/thread.h File Reference

```
#include <jinue-common/asm/vm.h>
```

Include dependency graph for thread.h:



This graph shows which files directly or indirectly include this file:



## Macros

- `#define THREAD_CONTEXT_SIZE PAGE_SIZE`
- `#define THREAD_CONTEXT_MASK (~(THREAD_CONTEXT_SIZE - 1))`

### 4.22.1 Macro Definition Documentation

#### 4.22.1.1 `#define THREAD_CONTEXT_MASK (~(THREAD_CONTEXT_SIZE - 1))`

Definition at line 40 of file thread.h.

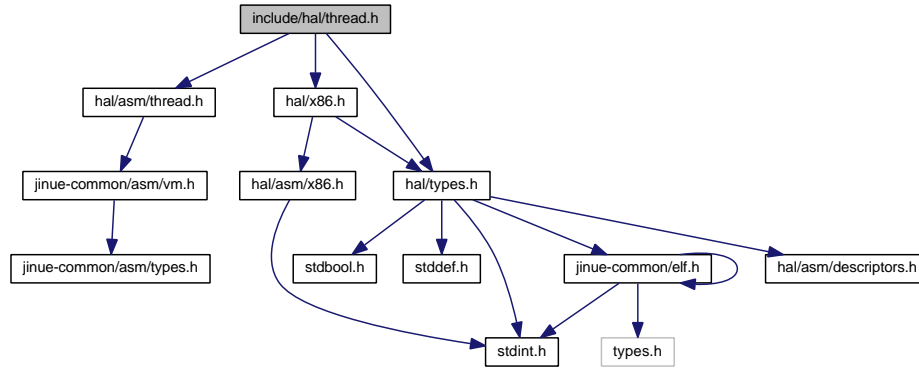
#### 4.22.1.2 `#define THREAD_CONTEXT_SIZE PAGE_SIZE`

Definition at line 38 of file thread.h.

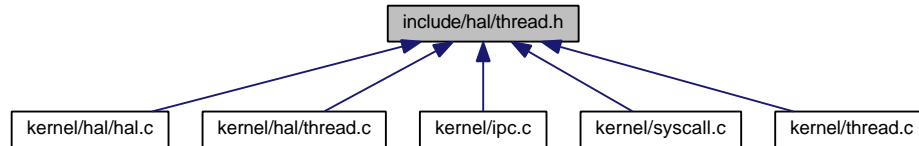
## 4.23 include/hal/thread.h File Reference

```
#include <hal/asm/thread.h>
#include <hal/x86.h>
#include <types.h>
```

Include dependency graph for thread.h:



This graph shows which files directly or indirectly include this file:



## Functions

- **thread\_t \* thread\_page\_init** (addr\_t thread\_page, addr\_t entry, addr\_t user\_stack)
- **void thread\_context\_switch** (thread\_context\_t \*from\_ctx, thread\_context\_t \*to\_ctx, bool destroy\_from)

### 4.23.1 Function Documentation

4.23.1.1 void thread\_context\_switch ( thread\_context\_t \* from\_ctx, thread\_context\_t \* to\_ctx, bool destroy\_from )

ASSERTION: to\_ctx argument must not be NULL

ASSERTION: from\_ctx argument must not be NULL if destroy\_from is true

Definition at line 122 of file thread.c.

References assert, CPU\_FEATURE\_SYSENTER, tss\_t::esp0, tss\_t::esp1, tss\_t::esp2, MSR\_IA32\_SYSENTER\_ESP, NULL, thread\_context\_switch\_stack(), and wrmsr().

Referenced by thread\_switch().

```

125                                     {
126
128     assert(to_ctx != NULL);
129
131     assert(from_ctx != NULL || ! destroy_from);

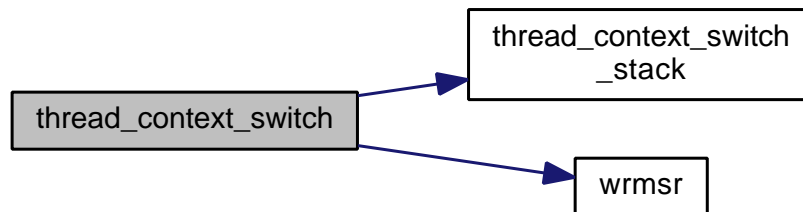
```

```

132
133     /* nothing to do if this is already the current thread */
134     if(from_ctx != to_ctx) {
135         /* setup TSS with kernel stack base for this thread context */
136         addr_t kernel_stack_base = get_kernel_stack_base(to_ctx);
137         tss_t *tss = get_tss();
138
139         tss->esp0 = kernel_stack_base;
140         tss->esp1 = kernel_stack_base;
141         tss->esp2 = kernel_stack_base;
142
143         /* update kernel stack address for SYSENTER instruction */
144         if(cpu_has_feature(CPU_FEATURE_SYSENTER)) {
145             wrmsr(MSR_IA32_SYSENTER_ESP, (uint64_t)(uintptr_t)kernel_stack_base);
146         }
147
148         /* switch thread context stack */
149         thread_context_switch_stack(from_ctx, to_ctx, destroy_from);
150     }
151 }

```

Here is the call graph for this function:



#### 4.23.1.2 thread\_t\* thread\_page\_init ( addr\_t thread\_page, addr\_t entry, addr\_t user\_stack )

Definition at line 81 of file thread.c.

References trapframe\_t::cs, trapframe\_t::ds, trapframe\_t::eflags, trapframe\_t::eip, kernel\_context\_t::eip, trapframe\_t::es, trapframe\_t::esp, trapframe\_t::fs, GDT\_USER\_CODE, GDT\_USER\_DATA, trapframe\_t::gs, thread\_context\_t::local\_storage\_addr, memset(), NULL, return\_from\_interrupt(), RPL\_USER, thread\_context\_t::saved\_stack\_pointer, SEG\_SELECTOR, trapframe\_t::ss, and thread\_t::thread\_ctx.

Referenced by thread\_create(), and thread\_create\_boot().

```

84     {
85
86         /* initialize fields */
87         thread_t *thread = (thread_t *)thread_page;
88         thread_context_t *thread_ctx = &thread->thread_ctx;
89
90         thread_ctx->local_storage_addr = NULL;
91
92         /* setup stack for initial return to user space */
93         void *kernel_stack_base = get_kernel_stack_base(thread_ctx);
94
95         trapframe_t *trapframe = (trapframe_t *)kernel_stack_base - 1;
96
97         memset(trapframe, 0, sizeof(trapframe_t));
98
99         trapframe->eip = (uint32_t)entry;
100         trapframe->esp = (uint32_t)user_stack;
101         trapframe->eflags = 2;
102         trapframe->cs = SEG_SELECTOR(GDT_USER_CODE, RPL_USER);
103         trapframe->ss = SEG_SELECTOR(GDT_USER_DATA, RPL_USER);
104         trapframe->ds = SEG_SELECTOR(GDT_USER_DATA, RPL_USER);
105         trapframe->es = SEG_SELECTOR(GDT_USER_DATA, RPL_USER);
106         trapframe->fs = SEG_SELECTOR(GDT_USER_DATA, RPL_USER);
107         trapframe->gs = SEG_SELECTOR(GDT_USER_DATA, RPL_USER);

```

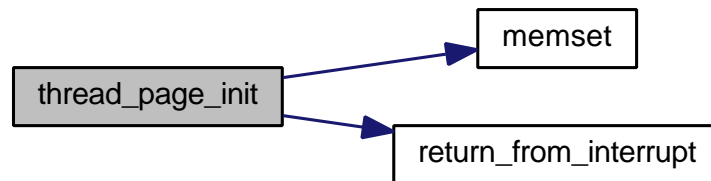


```

108
109     kernel_context_t *kernel_context = (kernel_context_t *)trapframe - 1;
110
111     memset(kernel_context, 0, sizeof(kernel_context_t));
112
113     /* This is the address to which thread_context_switch_stack() will return. */
114     kernel_context->eip = (uint32_t)return_from_interrupt;
115
116     /* set thread stack pointer */
117     thread_ctx->saved_stack_pointer = (addr_t)kernel_context;
118
119     return thread;
120 }

```

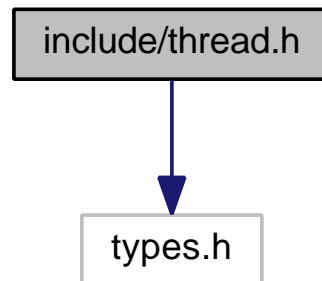
Here is the call graph for this function:



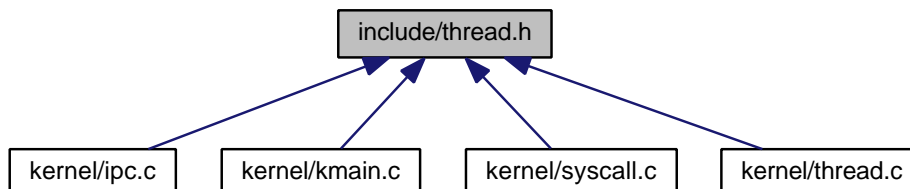
## 4.24 include/thread.h File Reference

```
#include <types.h>
```

Include dependency graph for thread.h:



This graph shows which files directly or indirectly include this file:



## Functions

- **thread\_t \* thread\_create** (**process\_t** \*process, **addr\_t** entry, **addr\_t** user\_stack)

- **thread\_t** \* **thread\_create\_boot** (**process\_t** \*process, **addr\_t** entry, **addr\_t** user\_stack, **boot\_alloc\_t** \*boot\_alloc)
- void **thread\_destroy** (**thread\_t** \*thread)
- void **thread\_ready** (**thread\_t** \*thread)
- void **thread\_switch** (**thread\_t** \*from\_thread, **thread\_t** \*to\_thread, **bool** blocked, **bool** do\_destroy)
- void **thread\_yield\_from** (**thread\_t** \*from\_thread, **bool** blocked, **bool** do\_destroy)

#### 4.24.1 Function Documentation

##### 4.24.1.1 **thread\_t**\* **thread\_create** ( **process\_t** \* process, **addr\_t** entry, **addr\_t** user\_stack )

Definition at line 56 of file thread.c.

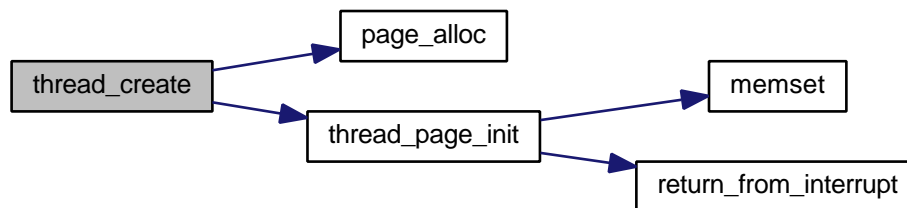
References NULL, page\_alloc(), and thread\_page\_init().

Referenced by dispatch\_syscall().

```

59                                     {
60
61     void *thread_page = page_alloc();
62
63     if(thread_page == NULL) {
64         return NULL;
65     }
66
67     thread_t *thread = thread_page_init(thread_page, entry, user_stack);
68     thread_init(thread, process);
69
70     return thread;
71 }
```

Here is the call graph for this function:



##### 4.24.1.2 **thread\_t**\* **thread\_create\_boot** ( **process\_t** \* process, **addr\_t** entry, **addr\_t** user\_stack, **boot\_alloc\_t** \* boot\_alloc )

Definition at line 73 of file thread.c.

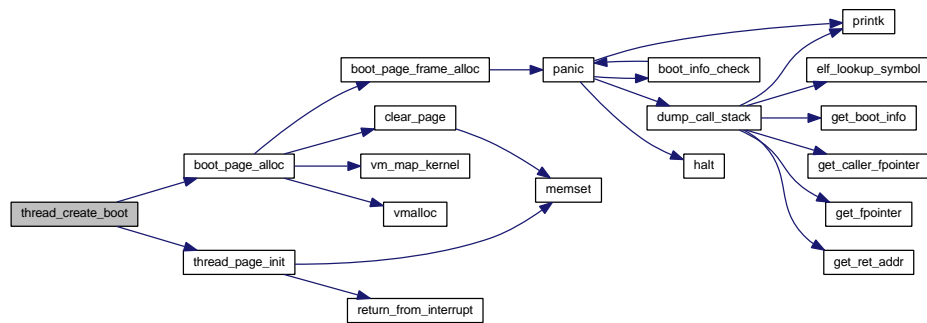
References boot\_page\_alloc(), and thread\_page\_init().

Referenced by kmain().

```

77                                     {
78
79     /* The kernel panics if this allocation fails. */
80     void *thread_page = boot_page_alloc(boot_alloc);
81     thread_t *thread = thread_page_init(thread_page, entry, user_stack);
82     thread_init(thread, process);
83
84     return thread;
85 }
```

Here is the call graph for this function:



#### 4.24.1.3 void thread\_destroy ( thread\_t \* thread )

Definition at line 88 of file thread.c.

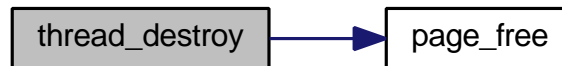
References page\_free(), and PAGE\_MASK.

```

88     {
89         void * thread_page = (void *) ((uintptr_t) thread & ~PAGE_MASK);
90         page_free(thread_page);
91     }

```

Here is the call graph for this function:



#### 4.24.1.4 void thread\_ready ( thread\_t \* thread )

Definition at line 93 of file thread.c.

References thread\_t::thread\_list.

Referenced by thread\_switch().

```

93     {
94         /* add thread to the tail of the ready list to give other threads a chance
95          * to run */
96         jinue_list_enqueue(&ready_list, &thread->thread_list);
97     }

```

#### 4.24.1.5 void thread\_switch ( thread\_t \* from\_thread, thread\_t \* to\_thread, bool blocked, bool do\_destroy )

Definition at line 99 of file thread.c.

References process\_t::addr\_space, NULL, thread\_t::process, thread\_context\_switch(), thread\_t::thread\_ctx, thread\_ready(), and vm\_switch\_addr\_space().

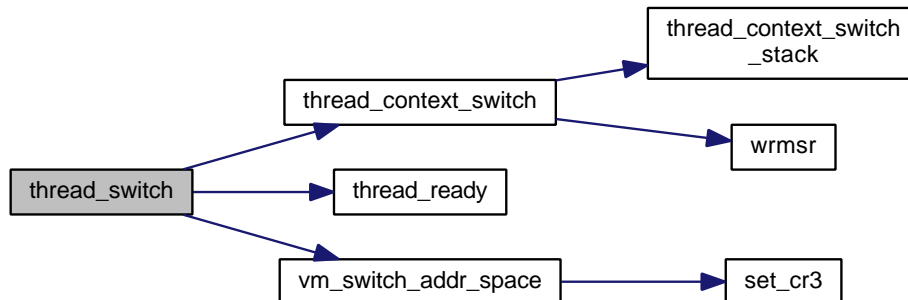
Referenced by ipc\_receive(), ipc\_reply(), ipc\_send(), and thread\_yield\_from().

```

103         {
104
105     if(to_thread != from_thread) {
106         thread_context_t    *from_context;
107         process_t           *from_process;
108
109         if(from_thread == NULL) {
110             from_context = NULL;
111             from_process = NULL;
112         }
113         else {
114             from_context = &from_thread->thread_ctx;
115             from_process = from_thread->process;
116
117             /* Put the the thread we are switching away from (the current thread)
118              * back into the ready list, unless it just blocked or it is being
119              * destroyed. */
120             if(! (do_destroy || blocked)) {
121                 thread_ready(from_thread);
122             }
123         }
124
125         if(from_process != to_thread->process) {
126             vm_switch_addr_space(
127                 &to_thread->process->addr_space,
128                 get_cpu_local_data()
129             );
130         }
131
132         thread_context_switch(
133             from_context,
134             &to_thread->thread_ctx,
135             do_destroy);
136     }
137 }

```

Here is the call graph for this function:



#### 4.24.1.6 void thread\_yield\_from ( thread\_t \* from\_thread, bool blocked, bool do\_destroy )

Definition at line 163 of file thread.c.

References `thread_switch()`.

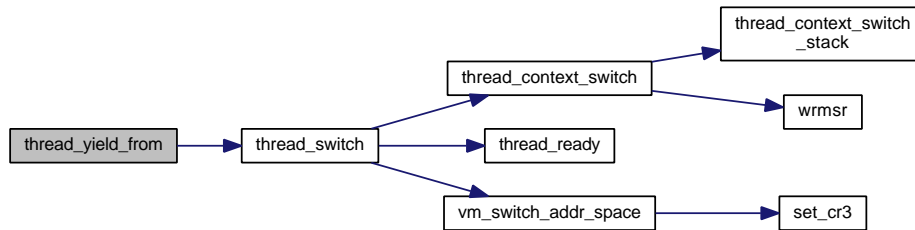
Referenced by `dispatch_syscall()`, `ipc_receive()`, `ipc_send()`, and `kmain()`.

```

163                                     {
164     bool from_can_run = ! (blocked || do_destroy);
165
166     thread_switch(
167         from_thread,
168         reschedule(from_thread, from_can_run),
169         blocked,
170         do_destroy);
171 }

```

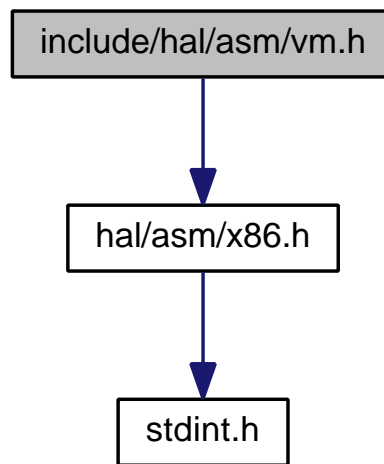
Here is the call graph for this function:



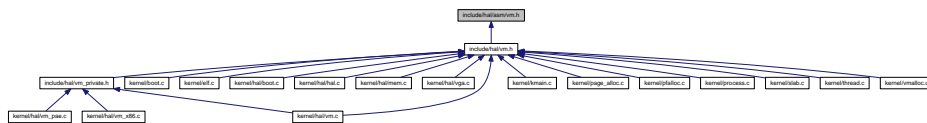
## 4.25 include/hal/asm/vm.h File Reference

```
#include <hal/asm/x86.h>
```

Include dependency graph for vm.h:



This graph shows which files directly or indirectly include this file:



## Macros

- **#define VM\_FLAG\_PRESENT X86\_PTE\_PRESENT**  
*page is present in memory*
- **#define VM\_FLAG\_READ\_ONLY 0**  
*page is read only*
- **#define VM\_FLAG\_READ\_WRITE X86\_PTE\_READ\_WRITE**  
*page is read/write accessible*
- **#define VM\_FLAG\_KERNEL X86\_PTE\_GLOBAL**

*kernel mode page*

- **#define VM\_FLAG\_USER X86\_PTE\_USER**

*user mode page*

- **#define VM\_FLAG\_ACCESSED X86\_PTE\_ACCESSED**

*page was accessed (read)*

- **#define VM\_FLAG\_DIRTY X86\_PTE\_DIRTY**

*page was written to*

## 4.25.1 Macro Definition Documentation

### 4.25.1.1 #define VM\_FLAG\_ACCESSED X86\_PTE\_ACCESSED

page was accessed (read)

Definition at line 53 of file vm.h.

### 4.25.1.2 #define VM\_FLAG\_DIRTY X86\_PTE\_DIRTY

page was written to

Definition at line 56 of file vm.h.

### 4.25.1.3 #define VM\_FLAG\_KERNEL X86\_PTE\_GLOBAL

kernel mode page

Definition at line 47 of file vm.h.

Referenced by vm\_boot\_init(), and vm\_map\_kernel().

### 4.25.1.4 #define VM\_FLAG\_PRESENT X86\_PTE\_PRESENT

page is present in memory

Definition at line 38 of file vm.h.

Referenced by vm\_pae\_create\_initial\_addr\_space(), vm\_pae\_destroy\_addr\_space(), and vm\_pae\_lookup\_page\_directory().

### 4.25.1.5 #define VM\_FLAG\_READ\_ONLY 0

page is read only

Definition at line 41 of file vm.h.

Referenced by elf\_load().

### 4.25.1.6 #define VM\_FLAG\_READ\_WRITE X86\_PTE\_READ\_WRITE

page is read/write accessible

Definition at line 44 of file vm.h.

Generated on Wed Mar 20 2019 21:26:14 for Jinue by Doxygen

- void **vm\_boot\_postinit** (const **boot\_info\_t** \***boot\_info**, **boot\_alloc\_t** \***boot\_alloc**, **bool** use\_pae)
- void **vm\_map\_kernel** (**addr\_t** vaddr, **kern\_paddr\_t** paddr, int flags)
- void **vm\_map\_user** (**addr\_space\_t** \***addr\_space**, **addr\_t** vaddr, **user\_paddr\_t** paddr, int flags)
- void **vm\_unmap\_kernel** (**addr\_t** addr)
- void **vm\_unmap\_user** (**addr\_space\_t** \***addr\_space**, **addr\_t** addr)
- **kern\_paddr\_t** **vm\_lookup\_kernel\_paddr** (**addr\_t** addr)
- void **vm\_change\_flags** (**addr\_space\_t** \***addr\_space**, **addr\_t** addr, int flags)
- void **vm\_map\_early** (**addr\_t** vaddr, **kern\_paddr\_t** paddr, int flags)
- **addr\_space\_t** \* **vm\_create\_addr\_space** (**addr\_space\_t** \***addr\_space**)
- **addr\_space\_t** \* **vm\_create\_initial\_addr\_space** (**bool** use\_pae, **boot\_alloc\_t** \***boot\_alloc**)
- void **vm\_destroy\_addr\_space** (**addr\_space\_t** \***addr\_space**)
- void **vm\_switch\_addr\_space** (**addr\_space\_t** \***addr\_space**, **cpu\_data\_t** \***cpu\_data**)

## Variables

- **addr\_space\_t** **initial\_addr\_space**

## 4.26.1 Macro Definition Documentation

### 4.26.1.1 #define ADDR\_4GB UINT64\_C(0x100000000)

Definition at line 51 of file vm.h.

### 4.26.1.2 #define EARLY\_PHYS\_TO\_VIRT( x ) (((uintptr\_t)(x)) + KLIMIT)

This header file contains the public interface of the low-level page table management code located in **hal/vm.c** (p. 331) and **hal/vm\_pae.c** (p. 342).

convert a physical address to a virtual address before the switch to the first address space

Definition at line 43 of file vm.h.

Referenced by **vm\_pae\_create\_initial\_addr\_space()**.

### 4.26.1.3 #define EARLY\_PTR\_TO\_PHYS\_ADDR( x ) ((kern\_paddr\_t)EARLY\_VIRT\_TO\_PHYS(x))

convert a pointer to a page frame address (early mappings)

Definition at line 49 of file vm.h.

Referenced by **boot\_page\_alloc\_early()**, **elf\_load()**, **vm\_boot\_init()**, **vm\_init\_initial\_page\_directory()**, **vm\_pae\_create\_initial\_addr\_space()**, and **vm\_x86\_create\_initial\_addr\_space()**.

### 4.26.1.4 #define EARLY\_VIRT\_TO\_PHYS( x ) (((uintptr\_t)(x)) - KLIMIT)

convert a virtual address to a physical address before the switch to the first address space

Definition at line 46 of file vm.h.

Referenced by **mem\_check\_memory()**, **vm\_map\_early()**, **vm\_pae\_create\_initial\_addr\_space()**, and **vm\_x86\_create\_initial\_addr\_space()**.



## 4.26.2 Function Documentation

**4.26.2.1** `void vm_boot_init ( const boot_info_t * boot_info, bool use_pae, cpu_data_t * cpu_data, boot_alloc_t * boot_alloc )`

below this point, it is no longer safe to call `pfalloc_early()`

Definition at line 57 of file `vm.c`.

References `boot_vmalloc()`, `addr_space_t::cr3`, `EARLY_PTR_TO_PHYS_ADDR`, `enable_pae()`, `boot_info_t::image_start`, `boot_alloc_t::its_early`, `PAGE_SIZE`, `printk()`, `vga_set_base_addr()`, `VGA_TEXT_VID_BASE`, `VGA_TEXT_VID_TOP`, `vm_create_initial_addr_space()`, `VM_FLAG_KERNEL`, `VM_FLAG_READ_WRITE`, `vm_map_early()`, `vm_pae_boot_init()`, `vm_pae_unmap_low_alias()`, `vm_switch_addr_space()`, and `vm_x86_boot_init()`.

Referenced by `hal_init()`.

```

61                                     {
62
63     addr_t addr;
64
65     if(use_pae) {
66         printk("Enabling Physical Address Extension (PAE).\n");
67         vm_pae_boot_init();
68     }
69     else {
70         vm_x86_boot_init();
71     }
72
73     pgtable_format_pae = use_pae;
74
75     /* create initial address space */
76     addr_space_t *addr_space = vm_create_initial_addr_space(use_pae, boot_alloc);
77
78     boot_alloc->its_early = false;
79
80
81     /* perform 1:1 mapping of kernel image and data */
82     for(addr = (addr_t)boot_info->image_start; addr < boot_alloc->kernel_vm_top; addr +=
PAGE_SIZE) {
83         vm_map_early(addr, EARLY_PTR_TO_PHYS_ADDR(addr), VM_FLAG_KERNEL |
VM_FLAG_READ_WRITE);
84     }
85
86     /* map VGA text buffer in the new address space
87     *
88     * This is a good place to do this because:
89     *
90     * 1) It is our last chance to allocate a continuous region of virtual memory.
91     *    Once the page allocator is initialized (see call to vm_alloc_init_allocator()
92     *    below) and we start using vm_alloc() to allocate memory, pages can only
93     *    be allocated one at a time.
94     *
95     * 2) Doing this last makes things simpler because this is the only place where
96     *    we have to allocate a continuous region of virtual memory but no physical
97     *    memory to back it. To allocate it, we just have to increase kernel_vm_top,
98     *    which represents the end of the virtual memory region that is used by the
99     *    kernel. */
100    addr_t vga_text_base;
101    kern_paddr_t paddr = VGA_TEXT_VID_BASE;
102
103    while(paddr < VGA_TEXT_VID_TOP) {
104        /* Pages allocated by successive calls to vmalloc_boot() are guaranteed
105        * to be contiguous. */
106        addr = boot_vmalloc(boot_alloc);
107
108        if(paddr == VGA_TEXT_VID_BASE) {
109            /* First iteration */
110            vga_text_base = addr;
111        }
112
113        vm_map_early(addr, paddr, VM_FLAG_KERNEL | VM_FLAG_READ_WRITE);
114        paddr += PAGE_SIZE;
115    }
116
117    /* remap VGA text buffer
118    *
119    * Note: after the call to vga_set_base_addr() below until we switch to the

```

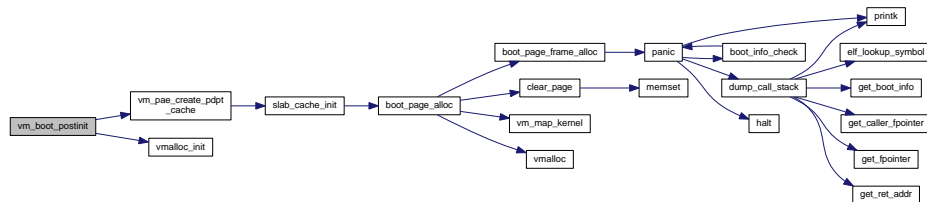


```

160
161  /* create slab cache to allocate PDPTs
162  *
163  * This must be done after the global page allocator has been initialized
164  * because the slab allocator needs to allocate a slab to allocate the new
165  * slab cache on the slab cache cache.
166  *
167  * This must be done before the first time vm_create_addr_space() is called. */
168  if(use_pae) {
169      vm_pae_create_pdpt_cache(boot_alloc);
170  }
171 }

```

Here is the call graph for this function:



#### 4.26.2.3 void vm\_change\_flags ( addr\_space\_t\* addr\_space, addr\_t addr, int flags )

ASSERTION: there is a page table entry marked present for this address

Definition at line 463 of file vm.c.

References assert, invalidate\_tlb(), and NULL.

```

463
464  pte_t *pte = vm_lookup_page_table_entry(addr_space, addr, false);
465
466  assert(pte != NULL && (get_pte_flags(pte) & VM_FLAG_PRESENT));
467
468  /* perform the flags change */
469  set_pte_flags(pte, flags | VM_FLAG_PRESENT);
470
471  vm_free_page_table_entry(addr, pte);
472
473  /* invalidate TLB entry for the affected page */
474  invalidate_tlb(addr);
475 }
476 }

```

Here is the call graph for this function:



#### 4.26.2.4 addr\_space\_t\* vm\_create\_addr\_space ( addr\_space\_t\* addr\_space )

Definition at line 520 of file vm.c.

References vm\_pae\_create\_addr\_space(), and vm\_x86\_create\_addr\_space().

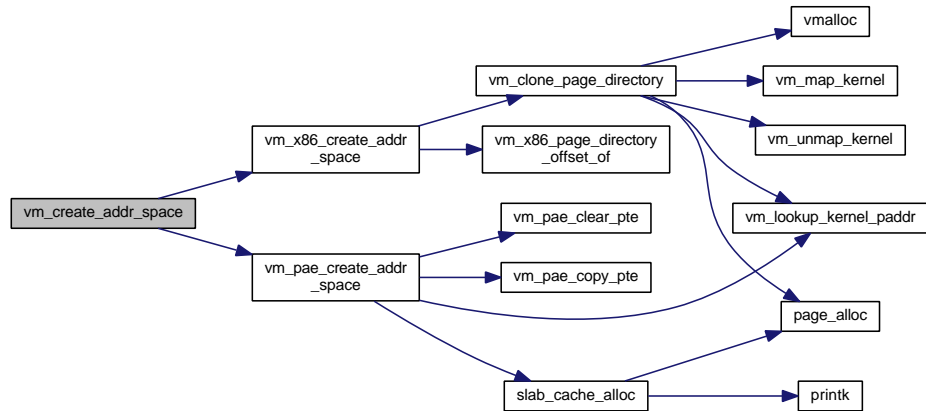
Referenced by process\_create().

```

520                                     {
521     if(pgtable_format_pae) {
522         return vm_pae_create_addr_space(addr_space);
523     }
524     else {
525         return vm_x86_create_addr_space(addr_space);
526     }
527 }

```

Here is the call graph for this function:



#### 4.26.2.5 addr\_space\_t\* vm\_create\_initial\_addr\_space ( bool use\_pae, boot\_alloc\_t \* boot\_alloc )

Definition at line 572 of file vm.c.

References `vm_pae_create_initial_addr_space()`, and `vm_x86_create_initial_addr_space()`.

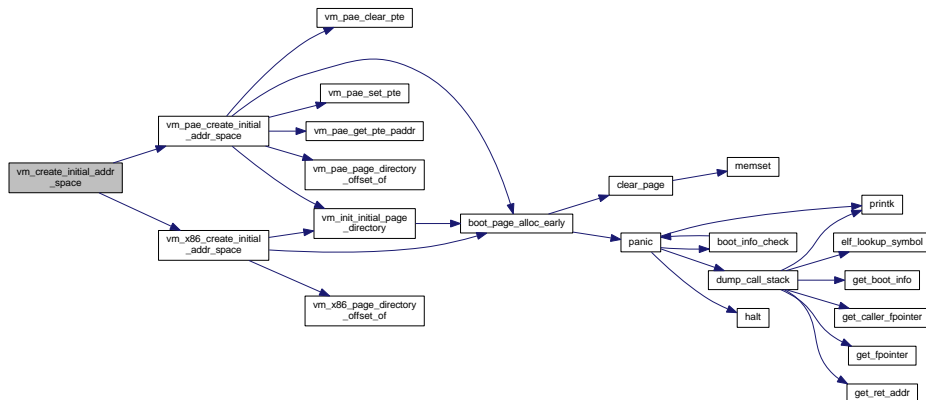
Referenced by `vm_boot_init()`.

```

574                                     {
575
576     if(use_pae) {
577         return vm_pae_create_initial_addr_space(boot_alloc);
578     }
579     else {
580         return vm_x86_create_initial_addr_space(boot_alloc);
581     }
582 }

```

Here is the call graph for this function:



## 4.26.2.6 void vm\_destroy\_addr\_space ( addr\_space\_t \* addr\_space )

ASSERTION: address space must not be NULL

ASSERTION: the initial address space should not be destroyed

ASSERTION: the current address space should not be destroyed

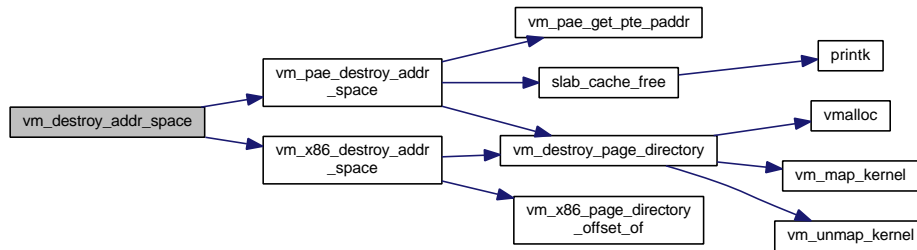
Definition at line 603 of file vm.c.

References `assert`, `NULL`, `vm_pae_destroy_addr_space()`, and `vm_x86_destroy_addr_space()`.

```

603                                     {
605     assert(addr_space != NULL);
606
608     assert(addr_space != &initial_addr_space);
609
611     assert( addr_space != get_current_addr_space() );
612
613     if(pgtable_format_pae) {
614         vm_pae_destroy_addr_space(addr_space);
615     }
616     else {
617         vm_x86_destroy_addr_space(addr_space);
618     }
619 }
```

Here is the call graph for this function:



## 4.26.2.7 kern\_paddr\_t vm\_lookup\_kernel\_paddr ( addr\_t addr )

ASSERTION: there is a page table entry marked present for this address

Definition at line 450 of file vm.c.

References `assert`, and `NULL`.

Referenced by `hal_init()`, `remove_page_frame()`, `vm_clone_page_directory()`, and `vm_pae_create_addr_space()`.

```

450                                     {
451     pte_t *pte = vm_lookup_page_table_entry(NULL, addr, false);
452
454     assert(pte != NULL && (get_pte_flags(pte) & VM_FLAG_PRESENT));
455
456     kern_paddr_t paddr = (kern_paddr_t) get_pte_paddr(pte);
457
458     vm_free_page_table_entry(addr, pte);
459
460     return paddr;
461 }
```

#### 4.26.2.8 void vm\_map\_early ( addr\_t vaddr, kern\_paddr\_t paddr, int flags )

ASSERTION: we are within the mapping set up by the setup code

ASSERTION: we assume vaddr is aligned on a page boundary

Definition at line 478 of file vm.c.

References assert, EARLY\_VIRT\_TO\_PHYS, page\_number\_of, and page\_offset\_of.

Referenced by vm\_boot\_init().

```

478                                     {
480     assert( is_early_pointer(vaddr) );
481
483     assert( page_offset_of(vaddr) == 0 );
484
485     pte_t *pte = get_pte_with_offset(global_page_tables, page_number_of(
EARLY_VIRT_TO_PHYS((uintptr_t)vaddr) ));
486     set_pte(pte, paddr, flags | VM_FLAG_PRESENT);
487 }
```

#### 4.26.2.9 void vm\_map\_kernel ( addr\_t vaddr, kern\_paddr\_t paddr, int flags )

Definition at line 434 of file vm.c.

References NULL, and VM\_FLAG\_KERNEL.

Referenced by add\_page\_frame(), boot\_page\_alloc(), boot\_page\_alloc\_image(), elf\_setup\_stack(), vm\_clone\_page\_directory(), vm\_destroy\_page\_directory(), vm\_pae\_lookup\_page\_directory(), and vm\_x86\_lookup\_page\_directory().

```

434                                     {
435     vm_map(NULL, vaddr, paddr, flags | VM_FLAG_KERNEL);
436 }
```

#### 4.26.2.10 void vm\_map\_user ( addr\_space\_t\* addr\_space, addr\_t vaddr, user\_paddr\_t paddr, int flags )

Definition at line 438 of file vm.c.

References VM\_FLAG\_USER.

Referenced by elf\_load(), and elf\_setup\_stack().

```

438                                     {
439     vm_map(addr_space, vaddr, paddr, flags | VM_FLAG_USER);
440 }
```

#### 4.26.2.11 void vm\_switch\_addr\_space ( addr\_space\_t\* addr\_space, cpu\_data\_t\* cpu\_data )

Definition at line 621 of file vm.c.

References addr\_space\_t::cr3, cpu\_data\_t::current\_addr\_space, and set\_cr3().

Referenced by thread\_switch(), and vm\_boot\_init().

```

621                                     {
622     set_cr3(addr_space->cr3);
623
624     cpu_data->current_addr_space = addr_space;
625 }
```

Here is the call graph for this function:



#### 4.26.2.12 void vm\_unmap\_kernel ( addr\_t addr )

Definition at line 442 of file vm.c.

References NULL.

Referenced by elf\_setup\_stack(), remove\_page\_frame(), vm\_clone\_page\_directory(), and vm\_destroy\_page\_directory().

```

442                                     {
443     vm_unmap(NULL, addr);
444 }
```

#### 4.26.2.13 void vm\_unmap\_user ( addr\_space\_t\* addr\_space, addr\_t addr )

Definition at line 446 of file vm.c.

```

446                                     {
447     vm_unmap(addr_space, addr);
448 }
```

### 4.26.3 Variable Documentation

#### 4.26.3.1 addr\_space\_t initial\_addr\_space

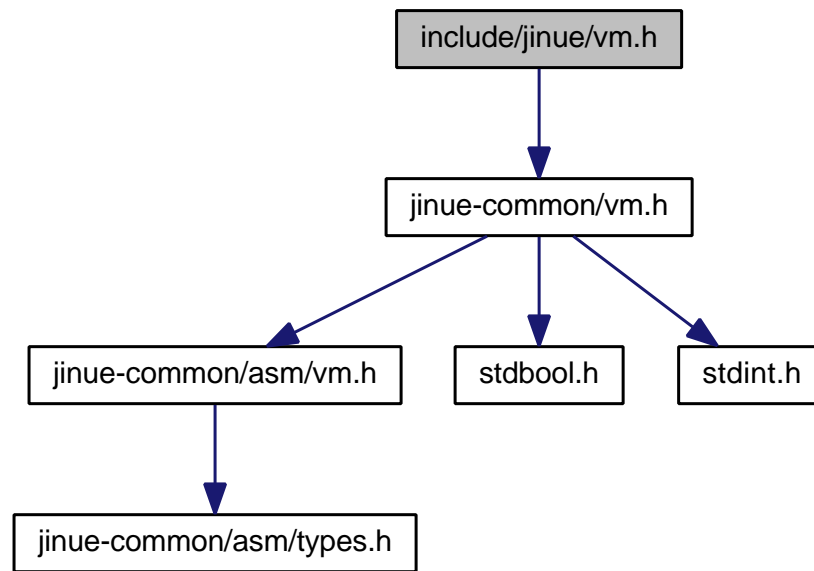
Definition at line 51 of file vm.c.

Referenced by process\_create\_initial(), vm\_pae\_create\_addr\_space(), vm\_pae\_create\_initial\_addr\_space(), vm\_x86\_create\_addr\_space(), and vm\_x86\_create\_initial\_addr\_space().

## 4.27 include/jinue/vm.h File Reference

```
#include <jinue-common/vm.h>
```

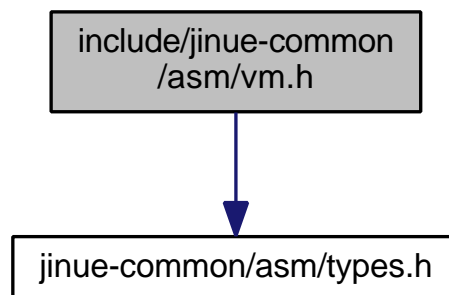
Include dependency graph for vm.h:



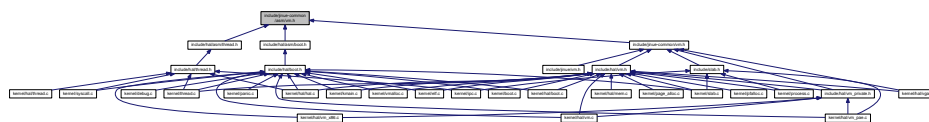
#### 4.28 include/jinue-common/asm/vm.h File Reference

```
#include <jinue-common/asm/types.h>
```

Include dependency graph for vm.h:



This graph shows which files directly or indirectly include this file:



#### Macros

- **#define PAGE\_BITS 12**  
*number of bits in virtual address for offset inside page*



- **#define PAGE\_SIZE** (1 << **PAGE\_BITS**) /\* 4096 \*/  
*size of page*
- **#define PAGE\_MASK** (**PAGE\_SIZE** - 1)  
*bit mask for offset in page*
- **#define KLIMIT** 0xc0000000  
*The virtual address range starting at KLIMIT is reserved by the kernel.*
- **#define KERNEL\_EARLY\_LIMIT** (**KLIMIT** + 2 \* **MB**)  
*limit of initial mapping performed by the 32-bit setup code*
- **#define KERNEL\_IMAGE\_END** (**KLIMIT** + 16 \* **MB**)  
*limit of the kernel image region*
- **#define KERNEL\_PREALLOC\_LIMIT** (**KLIMIT** + 32 \* **MB**)  
*limit up to which page tables are preallocated during kernel initialization*
- **#define STACK\_BASE** **KLIMIT**  
*stack base address (stack top)*
- **#define STACK\_SIZE** (8 \* **PAGE\_SIZE**)  
*initial stack size*
- **#define STACK\_START** (**STACK\_BASE** - **STACK\_SIZE**)  
*initial stack lower address*

## 4.28.1 Macro Definition Documentation

### 4.28.1.1 **#define KERNEL\_EARLY\_LIMIT** (**KLIMIT** + 2 \* **MB**)

limit of initial mapping performed by the 32-bit setup code

Definition at line 53 of file vm.h.

Referenced by mem\_check\_memory().

### 4.28.1.2 **#define KERNEL\_IMAGE\_END** (**KLIMIT** + 16 \* **MB**)

limit of the kernel image region

Definition at line 56 of file vm.h.

Referenced by vm\_boot\_postinit().

### 4.28.1.3 **#define KERNEL\_PREALLOC\_LIMIT** (**KLIMIT** + 32 \* **MB**)

limit up to which page tables are preallocated during kernel initialization

Definition at line 59 of file vm.h.

Referenced by vm\_boot\_postinit(), vm\_pae\_create\_initial\_addr\_space(), and vm\_x86\_create\_initial\_addr\_space().

### 4.28.1.4 **#define KLIMIT** 0xc0000000

The virtual address range starting at KLIMIT is reserved by the kernel.

The region above KLIMIT has the same mapping in all address spaces. KLIMIT must be aligned on a page directory boundary in PAE mode.

Definition at line 50 of file vm.h.

Referenced by vm\_pae\_create\_addr\_space(), vm\_pae\_create\_initial\_addr\_space(), vm\_pae\_destroy\_addr\_space(), vm\_x86\_create\_addr\_space(), vm\_x86\_create\_initial\_addr\_space(), and vm\_x86\_destroy\_addr\_space().

#### 4.28.1.5 #define PAGE\_BITS 12

number of bits in virtual address for offset inside page

Definition at line 39 of file vm.h.

#### 4.28.1.6 #define PAGE\_MASK (PAGE\_SIZE - 1)

bit mask for offset in page

Definition at line 45 of file vm.h.

Referenced by boot\_page\_alloc\_early(), thread\_destroy(), vm\_pae\_get\_pte\_flags(), vm\_pae\_get\_pte\_paddr(), vm\_pae\_set\_pte\_flags(), vm\_x86\_get\_pte\_flags(), vm\_x86\_get\_pte\_paddr(), and vm\_x86\_set\_pte\_flags().

#### 4.28.1.7 #define PAGE\_SIZE (1<<PAGE\_BITS)/\* 4096 \*/

size of page

Definition at line 42 of file vm.h.

Referenced by boot\_page\_alloc\_early(), boot\_page\_frame\_alloc(), boot\_vmalloc(), clear\_page(), elf\_load(), elf\_setup\_stack(), vm\_boot\_init(), and vm\_pae\_lookup\_page\_directory().

#### 4.28.1.8 #define STACK\_BASE KLIMIT

stack base address (stack top)

Definition at line 62 of file vm.h.

Referenced by elf\_setup\_stack().

#### 4.28.1.9 #define STACK\_SIZE (8 \* PAGE\_SIZE)

initial stack size

Definition at line 65 of file vm.h.

#### 4.28.1.10 #define STACK\_START (STACK\_BASE - STACK\_SIZE)

initial stack lower address

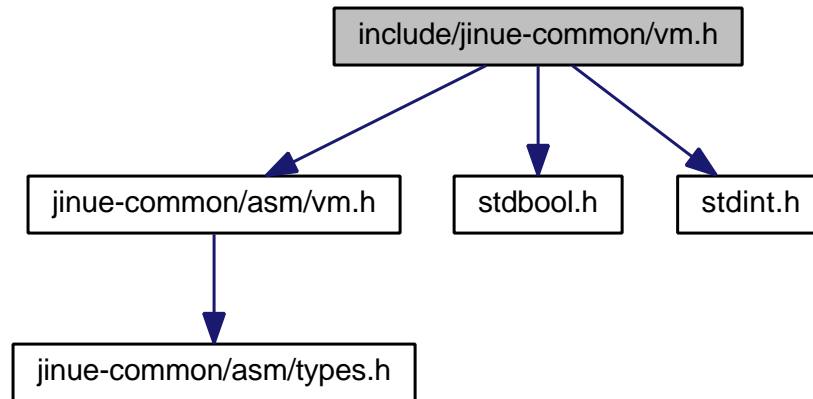
Definition at line 68 of file vm.h.

Referenced by elf\_setup\_stack().

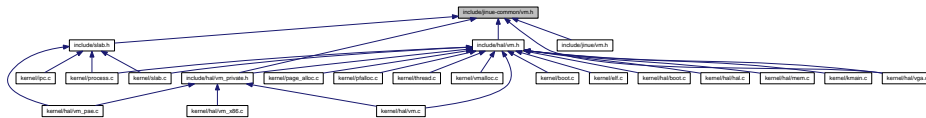
## 4.29 include/jinue-common/vm.h File Reference

```
#include <jinue-common/asm/vm.h>
```

```
#include <stdbool.h>
#include <stdint.h>
Include dependency graph for vm.h:
```



This graph shows which files directly or indirectly include this file:



## Macros

- #define **page\_offset\_of**(x) ((uintptr\_t)(x) & PAGE\_MASK)  
*byte offset in page of virtual (linear) address*
- #define **page\_address\_of**(x) ((uintptr\_t)(x) & ~PAGE\_MASK)  
*address of the page that contains a virtual (linear) address*
- #define **page\_number\_of**(x) ((uintptr\_t)(x) >> PAGE\_BITS)  
*sequential page number of virtual (linear) address*

### 4.29.1 Macro Definition Documentation

```
4.29.1.1 #define page_address_of( x ) ((uintptr_t)(x) & ~PAGE_MASK)
```

address of the page that contains a virtual (linear) address

Definition at line 45 of file vm.h.

Referenced by `vm_pae_create_addr_space()`.

#### 4.29.1.2 #define page\_number\_of( x ) ((uintptr\_t)(x) >> PAGE\_BITS)

sequential page number of virtual (linear) address

Definition at line 48 of file vm.h.

Referenced by `vm map early()`.

#### 4.29.1.3 #define page\_offset\_of( x )((uintptr\_t)(x) & PAGE\_MASK)

byte offset in page of virtual (linear) address

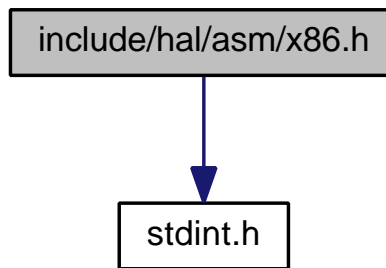
Definition at line 42 of file vm.h.

Referenced by boot\_info\_check(), vm\_map\_early(), and vm\_pae\_create\_addr\_space().

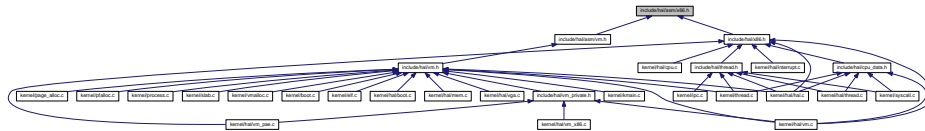
## 4.30 include/hal/asm/x86.h File Reference

```
#include <stdint.h>
```

Include dependency graph for x86.h:



This graph shows which files directly or indirectly include this file:



## Macros

- #define **X86\_CR0\_WP** (1<<16)  
*CR0 register: Write Protect.*
- #define **X86\_CR0\_PG** (1<<31)  
*CR0 register: Paging.*
- #define **X86\_CR4\_PSE** (1<<4)  
*CR4 register: Page Size Extension (PSE)*
- #define **X86\_CR4\_PAE** (1<<5)  
*CR4 register: Physical Address Extension (PAE)*
- #define **X86\_CR4\_PGE** (1<<7)  
*CR4 register: global pages.*
- #define **X86\_PTE\_PRESENT** (1<<0)  
*page is present in memory*
- #define **X86\_PTE\_READ\_WRITE** (1<<1)  
*page is read/write accessible*
- #define **X86\_PTE\_USER** (1<<2)  
*user mode page*

- **#define X86\_PTE\_WRITE\_THROUGH** (1<< 3)  
*write-through cache policy for page*
- **#define X86\_PTE\_CACHE\_DISABLE** (1<< 4)  
*uncached page*
- **#define X86\_PTE\_ACCESSED** (1<< 5)  
*page was accessed (read)*
- **#define X86\_PTE\_DIRTY** (1<< 6)  
*page was written to*
- **#define X86\_PDE\_PAGE\_SIZE** (1<< 7)  
*page directory entry describes a 4M page*
- **#define X86\_PTE\_GLOBAL** (1<< 8)  
*page is global (mapped in every address space)*
- **#define X86\_PTE\_NX** (UINT64\_C(1)<< 63)  
*do not execute bit*

### 4.30.1 Macro Definition Documentation

#### 4.30.1.1 **#define X86\_CR0\_PG** (1<<31)

CR0 register: Paging.

Definition at line 43 of file x86.h.

#### 4.30.1.2 **#define X86\_CR0\_WP** (1<<16)

CR0 register: Write Protect.

Definition at line 40 of file x86.h.

#### 4.30.1.3 **#define X86\_CR4\_PAE** (1<<5)

CR4 register: Physical Address Extension (PAE)

Definition at line 50 of file x86.h.

#### 4.30.1.4 **#define X86\_CR4\_PGE** (1<<7)

CR4 register: global pages.

Definition at line 53 of file x86.h.

#### 4.30.1.5 **#define X86\_CR4\_PSE** (1<<4)

CR4 register: Page Size Extension (PSE)

Definition at line 47 of file x86.h.

**4.30.1.6 #define X86\_PDE\_PAGE\_SIZE (1<< 7)**

page directory entry describes a 4M page

Definition at line 78 of file x86.h.

**4.30.1.7 #define X86\_PTE\_ACCESSED (1<< 5)**

page was accessed (read)

Definition at line 72 of file x86.h.

**4.30.1.8 #define X86\_PTE\_CACHE\_DISABLE (1<< 4)**

uncached page

Definition at line 69 of file x86.h.

**4.30.1.9 #define X86\_PTE\_DIRTY (1<< 6)**

page was written to

Definition at line 75 of file x86.h.

**4.30.1.10 #define X86\_PTE\_GLOBAL (1<< 8)**

page is global (mapped in every address space)

Definition at line 81 of file x86.h.

**4.30.1.11 #define X86\_PTE\_NX (UINT64\_C(1)<< 63)**

do not execute bit

Definition at line 84 of file x86.h.

**4.30.1.12 #define X86\_PTE\_PRESENT (1<< 0)**

page is present in memory

Definition at line 57 of file x86.h.

**4.30.1.13 #define X86\_PTE\_READ\_WRITE (1<< 1)**

page is read/write accessible

Definition at line 60 of file x86.h.

**4.30.1.14 #define X86\_PTE\_USER (1<< 2)**

user mode page

Definition at line 63 of file x86.h.

4.30.1.15 `#define X86_PTE_WRITE_THROUGH (1 << 3)`

write-through cache policy for page

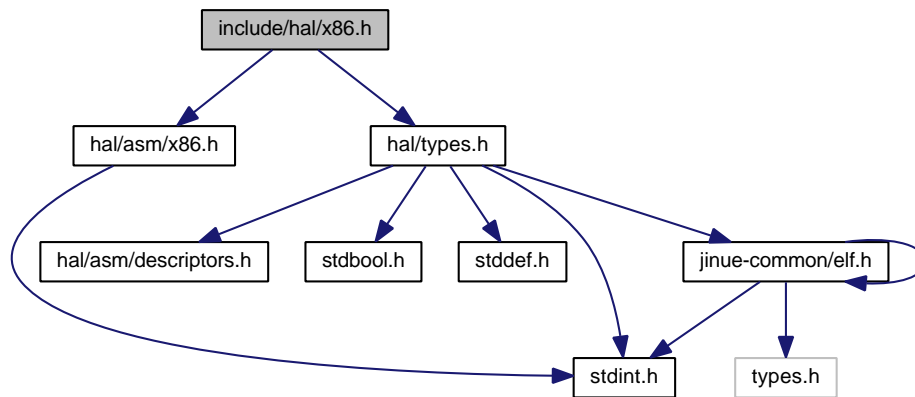
Definition at line 66 of file x86.h.

## 4.31 include/hal/x86.h File Reference

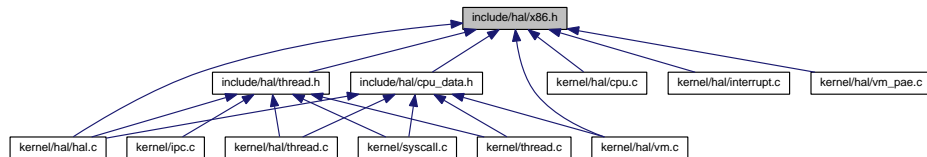
```
#include <hal/asm/x86.h>
```

```
#include <hal/types.h>
```

Include dependency graph for x86.h:



This graph shows which files directly or indirectly include this file:



## Data Structures

- struct `x86_cpuid_regs_t`

## Typedefs

- typedef `uint32_t msr_addr_t`

## Functions

- void `cli` (void)
- void `sti` (void)
- void `invalidate_tlb` (`addr_t vaddr`)
- void `lgdt` (`pseudo_descriptor_t *gdt_info`)

- void **lidt** (**pseudo\_descriptor\_t** \*idt\_info)
- void **ltr** (**seg\_selector\_t** sel)
- **uint32\_t** **cpuid** (**x86\_cpuid\_regs\_t** \*regs)
- **uint32\_t** **get\_esp** (void)
- **uint32\_t** **get\_cr0** (void)
- **uint32\_t** **get\_cr2** (void)
- **uint32\_t** **get\_cr3** (void)
- **uint32\_t** **get\_cr4** (void)
- void **set\_cr0** (**uint32\_t** val)
- void **set\_cr3** (**uint32\_t** val)
- void **set\_cr4** (**uint32\_t** val)
- **uint32\_t** **get\_eflags** (void)
- void **set\_eflags** (**uint32\_t** val)
- void **set\_cs** (**uint32\_t** val)
- void **set\_ds** (**uint32\_t** val)
- void **set\_es** (**uint32\_t** val)
- void **set\_fs** (**uint32\_t** val)
- void **set\_gs** (**uint32\_t** val)
- void **set\_ss** (**uint32\_t** val)
- **uint64\_t** **rdmsr** (**msr\_addr\_t** addr)
- void **wrmsr** (**msr\_addr\_t** addr, **uint64\_t** val)
- **uint32\_t** **get\_gs\_ptr** (**uint32\_t** \*ptr)
- **uint64\_t** **rdtsc** (void)
- void **enable\_pae** (**uint32\_t** cr3\_value)

#### 4.31.1 Typedef Documentation

##### 4.31.1.1 typedef **uint32\_t** **msr\_addr\_t**

Definition at line 45 of file x86.h.

#### 4.31.2 Function Documentation

##### 4.31.2.1 void **cli** ( void )

##### 4.31.2.2 **uint32\_t** **cpuid** ( **x86\_cpuid\_regs\_t** \* *regs* )

Referenced by `cpu_detect_features()`.

##### 4.31.2.3 void **enable\_pae** ( **uint32\_t** *cr3\_value* )

Referenced by `vm_boot_init()`.

##### 4.31.2.4 **uint32\_t** **get\_cr0** ( void )

##### 4.31.2.5 **uint32\_t** **get\_cr2** ( void )

Referenced by `dispatch_interrupt()`.



4.31.2.6 `uint32_t get_cr3 ( void )`

4.31.2.7 `uint32_t get_cr4 ( void )`

4.31.2.8 `uint32_t get_eflags ( void )`

Referenced by `cpu_detect_features()`.

4.31.2.9 `uint32_t get_esp ( void )`

4.31.2.10 `uint32_t get_gs_ptr ( uint32_t * ptr )`

4.31.2.11 `void invalidate_tlb ( addr_t vaddr )`

Referenced by `vm_change_flags()`.

4.31.2.12 `void lgdt ( pseudo_descriptor_t * gdt_info )`

4.31.2.13 `void lidt ( pseudo_descriptor_t * idt_info )`

4.31.2.14 `void ltr ( seg_selector_t sel )`

4.31.2.15 `uint64_t rdmsr ( msr_addr_t addr )`

4.31.2.16 `uint64_t rdtsc ( void )`

4.31.2.17 `void set_cr0 ( uint32_t val )`

4.31.2.18 `void set_cr3 ( uint32_t val )`

Referenced by `vm_switch_addr_space()`.

4.31.2.19 `void set_cr4 ( uint32_t val )`

4.31.2.20 `void set_cs ( uint32_t val )`

4.31.2.21 `void set_ds ( uint32_t val )`

4.31.2.22 `void set_eflags ( uint32_t val )`

Referenced by `cpu_detect_features()`.

4.31.2.23 `void set_es ( uint32_t val )`

4.31.2.24 `void set_fs ( uint32_t val )`

4.31.2.25 `void set_gs ( uint32_t val )`

4.31.2.26 `void set_ss ( uint32_t val )`

4.31.2.27 void sti ( void )

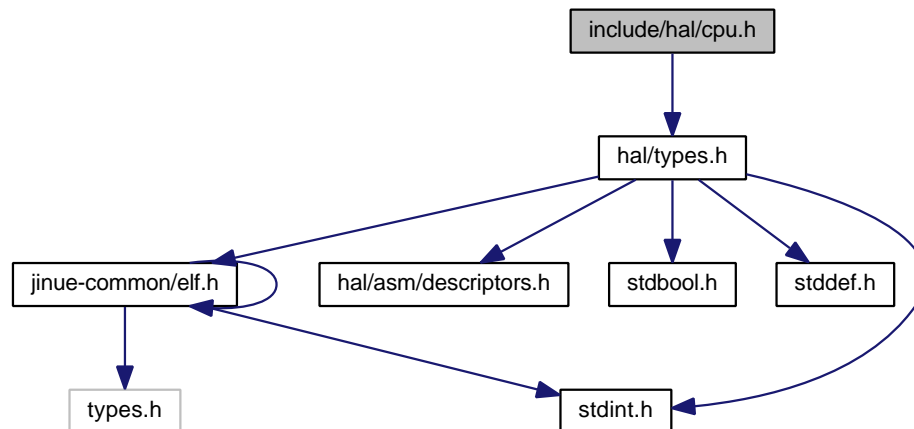
4.31.2.28 void wrmsr ( msr\_addr\_t addr, uint64\_t val )

Referenced by thread\_context\_switch().

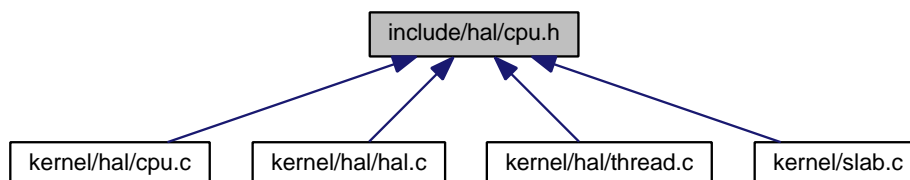
## 4.32 include/hal/cpu.h File Reference

```
#include <hal/types.h>
```

Include dependency graph for cpu.h:



This graph shows which files directly or indirectly include this file:



## Data Structures

- struct `cpu_info_t`

## Macros

- #define `MSR_IA32_SYSENTER_CS` 0x174
- #define `MSR_IA32_SYSENTER_ESP` 0x175
- #define `MSR_IA32_SYSENTER_EIP` 0x176
- #define `MSR_EFER` 0xC0000080
- #define `MSR_STAR` 0xC0000081
- #define `MSR_FLAG_STAR_SCE` (1<<0)
- #define `CPU_FEATURE_CPUID` (1<<0)

- `#define CPU_FEATURE_SYSENTER (1<<1)`
- `#define CPU_FEATURE_SYSCALL (1<<2)`
- `#define CPU_FEATURE_LOCAL_APIC (1<<3)`
- `#define CPU_FEATURE_PAE (1<<4)`
- `#define CPU_EFLAGS_ID (1<<21)`
- `#define CPUID_FEATURE_FPU (1<<0)`
- `#define CPUID_FEATURE_PAE (1<<6)`
- `#define CPUID_FEATURE_APIC (1<<9)`
- `#define CPUID_FEATURE_SEP (1<<11)`
- `#define CPUID_FEATURE_CLFLUSH (1<<19)`
- `#define CPUID_FEATURE_HTTP (1<<28)`
- `#define CPUID_EXT_FEATURE_SYSCALL (1<<11)`
- `#define CPU_VENDOR_GENERIC 0`
- `#define CPU_VENDOR_AMD 1`
- `#define CPU_VENDOR_INTEL 2`
- `#define CPU_VENDOR_AMD_DW0 0x68747541 /* Auth */`
- `#define CPU_VENDOR_AMD_DW1 0x69746e65 /* enti */`
- `#define CPU_VENDOR_AMD_DW2 0x444d4163 /* cAMD */`
- `#define CPU_VENDOR_INTEL_DW0 0x756e6547 /* Genu */`
- `#define CPU_VENDOR_INTEL_DW1 0x49656e69 /* inel */`
- `#define CPU_VENDOR_INTEL_DW2 0x6c65746e /* ntel */`

## Functions

- void `cpu_init_data(cpu_data_t *data)`
- void `cpu_detect_features(void)`

## Variables

- `cpu_info_t cpu_info`

### 4.32.1 Macro Definition Documentation

#### 4.32.1.1 `#define CPU_EFLAGS_ID (1<<21)`

Definition at line 63 of file `cpu.h`.

Referenced by `cpu_detect_features()`.

#### 4.32.1.2 `#define CPU_FEATURE_CPUID (1<<0)`

Definition at line 52 of file `cpu.h`.

Referenced by `cpu_detect_features()`.

#### 4.32.1.3 `#define CPU_FEATURE_LOCAL_APIC (1<<3)`

Definition at line 58 of file `cpu.h`.

Referenced by `cpu_detect_features()`.

**4.32.1.4 #define CPU\_FEATURE\_PAE (1<<4)**

Definition at line 60 of file cpu.h.

Referenced by cpu\_detect\_features(), and hal\_init().

**4.32.1.5 #define CPU\_FEATURE\_SYSCALL (1<<2)**

Definition at line 56 of file cpu.h.

Referenced by cpu\_detect\_features().

**4.32.1.6 #define CPU\_FEATURE\_SYSENTER (1<<1)**

Definition at line 54 of file cpu.h.

Referenced by cpu\_detect\_features(), and thread\_context\_switch().

**4.32.1.7 #define CPU\_VENDOR\_AMD 1**

Definition at line 84 of file cpu.h.

Referenced by cpu\_detect\_features().

**4.32.1.8 #define CPU\_VENDOR\_AMD\_DW0 0x68747541 /\* Auth \*/**

Definition at line 89 of file cpu.h.

Referenced by cpu\_detect\_features().

**4.32.1.9 #define CPU\_VENDOR\_AMD\_DW1 0x69746e65 /\* enti \*/**

Definition at line 90 of file cpu.h.

Referenced by cpu\_detect\_features().

**4.32.1.10 #define CPU\_VENDOR\_AMD\_DW2 0x444d4163 /\* cAMD \*/**

Definition at line 91 of file cpu.h.

Referenced by cpu\_detect\_features().

**4.32.1.11 #define CPU\_VENDOR\_GENERIC 0**

Definition at line 82 of file cpu.h.

Referenced by cpu\_detect\_features().

**4.32.1.12 #define CPU\_VENDOR\_INTEL 2**

Definition at line 86 of file cpu.h.

Referenced by cpu\_detect\_features().

4.32.1.13 `#define CPU_VENDOR_INTEL_DW0 0x756e6547 /* Genu */`

Definition at line 93 of file `cpu.h`.

Referenced by `cpu_detect_features()`.

4.32.1.14 `#define CPU_VENDOR_INTEL_DW1 0x49656e69 /* intel */`

Definition at line 94 of file `cpu.h`.

Referenced by `cpu_detect_features()`.

4.32.1.15 `#define CPU_VENDOR_INTEL_DW2 0x6c65746e /* intel */`

Definition at line 95 of file `cpu.h`.

Referenced by `cpu_detect_features()`.

4.32.1.16 `#define CPUID_EXT_FEATURE_SYSCALL (1<<11)`

Definition at line 79 of file `cpu.h`.

Referenced by `cpu_detect_features()`.

4.32.1.17 `#define CPUID_FEATURE_APIC (1<<9)`

Definition at line 70 of file `cpu.h`.

Referenced by `cpu_detect_features()`.

4.32.1.18 `#define CPUID_FEATURE_CLFLUSH (1<<19)`

Definition at line 74 of file `cpu.h`.

Referenced by `cpu_detect_features()`.

4.32.1.19 `#define CPUID_FEATURE_FPU (1<<0)`

Definition at line 66 of file `cpu.h`.

4.32.1.20 `#define CPUID_FEATURE_HTT (1<<28)`

Definition at line 76 of file `cpu.h`.

4.32.1.21 `#define CPUID_FEATURE_PAE (1<<6)`

Definition at line 68 of file `cpu.h`.

Referenced by `cpu_detect_features()`.

#### 4.32.1.22 `#define CPUID_FEATURE_SEP (1<<11)`

Definition at line 72 of file `cpu.h`.

Referenced by `cpu_detect_features()`.

#### 4.32.1.23 `#define MSR_EFER 0xC0000080`

Definition at line 44 of file `cpu.h`.

#### 4.32.1.24 `#define MSR_FLAG_STAR_SCE (1<<0)`

Definition at line 49 of file `cpu.h`.

#### 4.32.1.25 `#define MSR_IA32_SYSENTER_CS 0x174`

Definition at line 38 of file `cpu.h`.

#### 4.32.1.26 `#define MSR_IA32_SYSENTER_EIP 0x176`

Definition at line 42 of file `cpu.h`.

#### 4.32.1.27 `#define MSR_IA32_SYSENTER_ESP 0x175`

Definition at line 40 of file `cpu.h`.

Referenced by `thread_context_switch()`.

#### 4.32.1.28 `#define MSR_STAR 0xC0000081`

Definition at line 46 of file `cpu.h`.

### 4.32.2 Function Documentation

#### 4.32.2.1 `void cpu_detect_features ( void )`

Definition at line 87 of file `cpu.c`.

References `CPU_EFLAGS_ID`, `CPU_FEATURE_CPUID`, `CPU_FEATURE_LOCAL_APIC`, `CPU_FEATURE_PAE`, `CPU_FEATURE_SYSCALL`, `CPU_FEATURE_SYSENTER`, `CPU_VENDOR_AMD`, `CPU_VENDOR_AMD_DW0`, `CPU_VENDOR_AMD_DW1`, `CPU_VENDOR_AMD_DW2`, `CPU_VENDOR_GENERIC`, `CPU_VENDOR_INTEL`, `CPU_VENDOR_INTEL_DW0`, `CPU_VENDOR_INTEL_DW1`, `CPU_VENDOR_INTEL_DW2`, `cpuid()`, `CPUID_EXT_FEATURE_SYSCALL`, `CPUID_FEATURE_APIC`, `CPUID_FEATURE_CLFLUSH`, `CPUID_FEATURE_PAE`, `CPUID_FEATURE_SEP`, `cpu_info_t::dcache_alignment`, `x86_cpuid_regs_t::eax`, `x86_cpuid_regs_t::ebx`, `x86_cpuid_regs_t::ecx`, `x86_cpuid_regs_t::edx`, `cpu_info_t::family`, `cpu_info_t::features`, `get_eflags()`, `cpu_info_t::model`, `set_eflags()`, `cpu_info_t::stepping`, and `cpu_info_t::vendor`.

Referenced by `hal_init()`.

```

87     {
88         uint32_t temp_eflags;
89
90         /* default values */
91         cpu_info.dcache_alignment = 32;
92         cpu_info.features         = 0;
93         cpu_info.vendor           = CPU_VENDOR_GENERIC;
94         cpu_info.family           = 0;
95         cpu_info.model            = 0;
96         cpu_info.stepping         = 0;
97
98         /* The CPUID instruction is available if we can change the value of eflags
99          * bit 21 (ID) */
100        temp_eflags = get_eflags();
101        temp_eflags ^= CPU_EFLAGS_ID;
102        set_eflags(temp_eflags);
103
104        if(temp_eflags == get_eflags()) {
105            cpu_info.features |= CPU_FEATURE_CPUID;
106        }
107
108        if(cpu_has_feature(CPU_FEATURE_CPUID)) {
109            uint32_t signature;
110            uint32_t flags, ext_flags;
111            uint32_t vendor_dw0, vendor_dw1, vendor_dw2;
112            uint32_t cpuid_max;
113            uint32_t cpuid_ext_max;
114            x86_cpuid_regs_t regs;
115
116            /* default values */
117            flags = 0;
118            ext_flags = 0;
119
120            /* function 0: vendor ID string, max value of eax when calling CPUID */
121            regs.eax = 0;
122
123            /* call CPUID instruction */
124            cpuid_max = cpuid(&regs);
125            vendor_dw0 = regs.ebx;
126            vendor_dw1 = regs.edx;
127            vendor_dw2 = regs.ecx;
128
129            /* identify vendor */
130            if( vendor_dw0 == CPU_VENDOR_AMD_DW0
131               && vendor_dw1 == CPU_VENDOR_AMD_DW1
132               && vendor_dw2 == CPU_VENDOR_AMD_DW2) {
133
134                cpu_info.vendor = CPU_VENDOR_AMD;
135            }
136            else if (vendor_dw0 == CPU_VENDOR_INTEL_DW0
137                    && vendor_dw1 == CPU_VENDOR_INTEL_DW1
138                    && vendor_dw2 == CPU_VENDOR_INTEL_DW2) {
139
140                cpu_info.vendor = CPU_VENDOR_INTEL;
141            }
142
143            /* get processor signature (family/model/stepping) and feature flags */
144            if(cpuid_max >= 1) {
145                /* function 1: processor signature and feature flags */
146                regs.eax = 1;
147
148                /* call CPUID instruction */
149                signature = cpuid(&regs);
150
151                /* set processor signature */
152                cpu_info.stepping = signature & 0xf;
153                cpu_info.model    = (signature >> 4) & 0xf;
154                cpu_info.family    = (signature >> 8) & 0xf;
155
156                /* feature flags */
157                flags = regs.edx;
158
159                /* cache alignment */
160                if(flags & CPUID_FEATURE_CLFLUSH) {
161                    cpu_info.dcache_alignment = ((regs.ebx >> 8) & 0xff) * 8;
162                }
163            }
164
165            /* extended function 0: max value of eax when calling CPUID (extended function) */
166            regs.eax = 0x80000000;
167            cpuid_ext_max = cpuid(&regs);

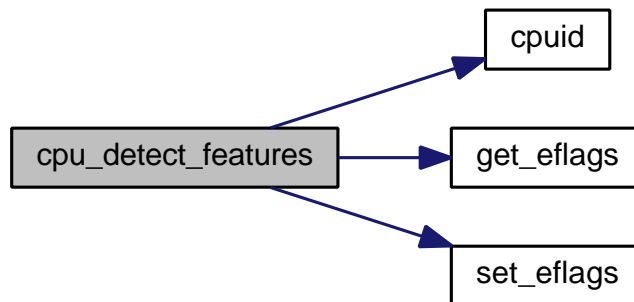
```

```

168
169     /* get extended feature flags */
170     if(cpuid_ext_max >= 0x80000001) {
171         /* extended function 1: extended feature flags */
172         regs.eax = 0x80000001;
173         (void)cpuid(&regs);
174
175         /* extended feature flags */
176         ext_flags = regs.edx;
177     }
178
179     /* support for SYSENTER/SYSEXIT instructions */
180     if(flags & CPUID_FEATURE_SEP) {
181         if(cpu_info.vendor == CPU_VENDOR_AMD) {
182             cpu_info.features |= CPU_FEATURE_SYSENTER;
183         }
184         else if(cpu_info.vendor == CPU_VENDOR_INTEL) {
185             if(cpu_info.family == 6 && cpu_info.model < 3 && cpu_info.
stepping < 3) {
186                 /* not supported */
187             }
188             else {
189                 cpu_info.features |= CPU_FEATURE_SYSENTER;
190             }
191         }
192     }
193
194     /* support for SYSCALL/SYSRET instructions */
195     if(cpu_info.vendor == CPU_VENDOR_AMD) {
196         if(ext_flags & CPUID_EXT_FEATURE_SYSCALL) {
197             cpu_info.features |= CPU_FEATURE_SYSCALL;
198         }
199     }
200
201     /* support for local APIC */
202     if(cpu_info.vendor == CPU_VENDOR_AMD || cpu_info.vendor ==
CPU_VENDOR_INTEL) {
203         if(flags & CPUID_FEATURE_APIC) {
204             cpu_info.features |= CPU_FEATURE_LOCAL_APIC;
205         }
206     }
207
208     /* support for physical address extension (PAE) */
209     if(cpu_info.vendor == CPU_VENDOR_AMD || cpu_info.vendor ==
CPU_VENDOR_INTEL) {
210         if(flags & CPUID_FEATURE_PAE) {
211             cpu_info.features |= CPU_FEATURE_PAE;
212         }
213     }
214 }
215 }

```

Here is the call graph for this function:



#### 4.32.2.2 void cpu\_init\_data ( cpu\_data\_t \* data )

Definition at line 42 of file `cpu.c`.



References `cpu_data_t::current_addr_space`, `tss_t::esp0`, `tss_t::esp1`, `tss_t::esp2`, `cpu_data_t::gdt`, `GDT_KERNEL_CODE`, `GDT_KERNEL_DATA`, `GDT_NULL`, `GDT_PER_CPU_DATA`, `GDT_TSS`, `GDT_USER_CODE`, `GDT_USER_DATA`, `GDT_USER_TLS_DATA`, `memset()`, `NULL`, `RPL_KERNEL`, `SEG_DESCRIPTOR`, `SEG_FLAG_32BIT`, `SEG_FLAG_IN_BYTES`, `SEG_FLAG_KERNEL`, `SEG_FLAG_NORMAL`, `SEG_FLAG_NOSYSTEM`, `SEG_FLAG_PRESENT`, `SEG_FLAG_TSS`, `SEG_FLAG_USER`, `SEG_SELECTOR`, `SEG_TYPE_CODE`, `SEG_TYPE_DATA`, `SEG_TYPE_TSS`, `cpu_data_t::self`, `tss_t::ss0`, `tss_t::ss1`, `tss_t::ss2`, `cpu_data_t::tss`, and `TSS_LIMIT`.

Referenced by `hal_init()`.

```

42         {
43     tss_t *tss;
44
45     tss = &data->tss;
46
47     /* initialize with zeroes */
48     memset(data, '\0', sizeof(cpu_data_t));
49
50     data->self = data;
51     data->current_addr_space = NULL;
52
53     /* initialize GDT */
54     data->gdt[GDT_NULL] = SEG_DESCRIPTOR(0, 0, 0);
55
56     data->gdt[GDT_KERNEL_CODE] =
57         SEG_DESCRIPTOR( 0, 0xffff, SEG_TYPE_CODE |
58         SEG_FLAG_KERNEL | SEG_FLAG_NORMAL);
59
60     data->gdt[GDT_KERNEL_DATA] =
61         SEG_DESCRIPTOR( 0, 0xffff, SEG_TYPE_DATA |
62         SEG_FLAG_KERNEL | SEG_FLAG_NORMAL);
63
64     data->gdt[GDT_USER_CODE] =
65         SEG_DESCRIPTOR( 0, 0xffff, SEG_TYPE_CODE |
66         SEG_FLAG_USER | SEG_FLAG_NORMAL);
67
68     data->gdt[GDT_USER_DATA] =
69         SEG_DESCRIPTOR( 0, 0xffff, SEG_TYPE_DATA |
70         SEG_FLAG_USER | SEG_FLAG_NORMAL);
71
72     data->gdt[GDT_TSS] =
73         SEG_DESCRIPTOR( tss, TSS_LIMIT-1, SEG_TYPE_TSS |
74         SEG_FLAG_KERNEL | SEG_FLAG_TSS);
75
76     data->gdt[GDT_PER_CPU_DATA] =
77         SEG_DESCRIPTOR( data, sizeof(cpu_data_t)-1, SEG_TYPE_DATA |
78         SEG_FLAG_KERNEL | SEG_FLAG_32BIT | SEG_FLAG_IN_BYTES | SEG_FLAG_NOSYSTEM |
79         SEG_FLAG_PRESENT);
80
81     data->gdt[GDT_USER_TLS_DATA] = SEG_DESCRIPTOR(0, 0, 0);
82
83     /* setup kernel stack in TSS */
84     tss->ss0 = SEG_SELECTOR(GDT_KERNEL_DATA, RPL_KERNEL);
85     tss->ss1 = SEG_SELECTOR(GDT_KERNEL_DATA, RPL_KERNEL);
86     tss->ss2 = SEG_SELECTOR(GDT_KERNEL_DATA, RPL_KERNEL);
87
88     /* kernel stack address is updated by thread_context_switch() */
89     tss->esp0 = NULL;
90     tss->esp1 = NULL;
91     tss->esp2 = NULL;
92 }

```

Here is the call graph for this function:



### 4.32.3 Variable Documentation

#### 4.32.3.1 `cpu_info_t` `cpu_info`

Definition at line 39 of file `cpu.c`.

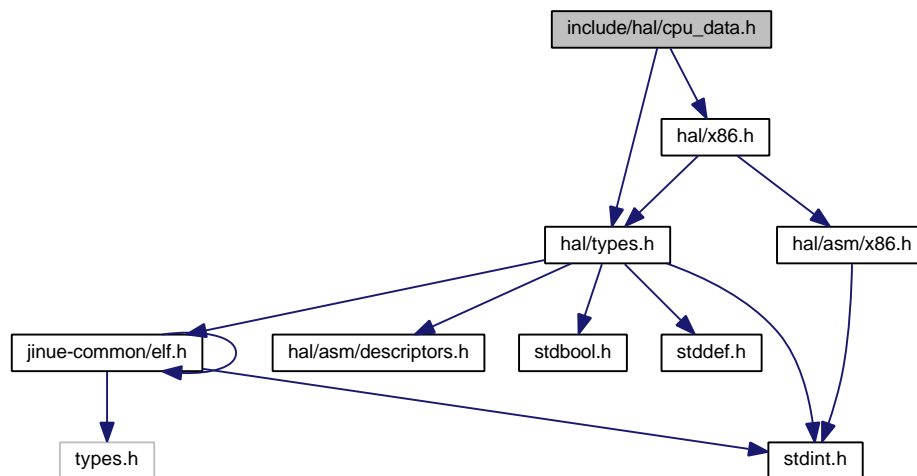
Referenced by `slab_cache_init()`.

### 4.33 `include/hal/cpu_data.h` File Reference

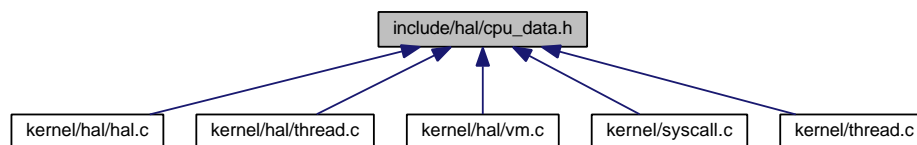
```
#include <hal/types.h>
```

```
#include <hal/x86.h>
```

Include dependency graph for `cpu_data.h`:



This graph shows which files directly or indirectly include this file:



### Macros

- `#define CPU_DATA_ALIGNMENT 256`

#### 4.33.1 Macro Definition Documentation

##### 4.33.1.1 `#define CPU_DATA_ALIGNMENT 256`

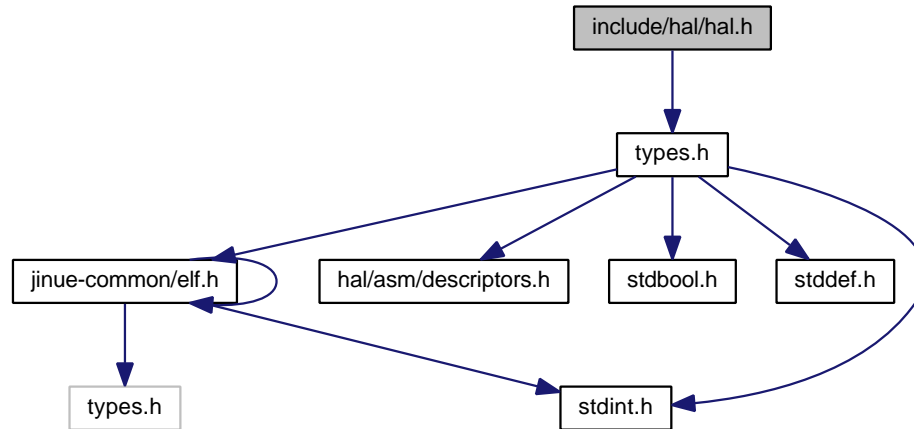
Definition at line 39 of file `cpu_data.h`.

Referenced by `hal_init()`.

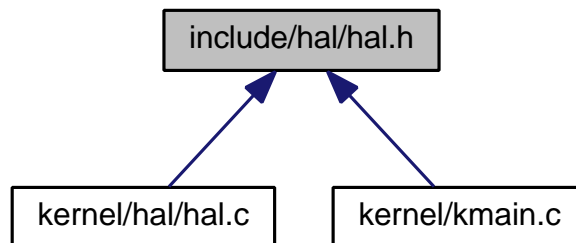
## 4.34 include/hal/hal.h File Reference

```
#include <types.h>
```

Include dependency graph for hal.h:



This graph shows which files directly or indirectly include this file:



### Functions

- void **hal\_init** (**boot\_alloc\_t** \*boot\_alloc, const **boot\_info\_t** \*boot\_info)

#### 4.34.1 Function Documentation

##### 4.34.1.1 void hal\_init ( boot\_alloc\_t \* boot\_alloc, const boot\_info\_t \* boot\_info )

Definition at line 156 of file hal.c.

References `assert`, `boot_heap_alloc`, `boot_page_alloc()`, `boot_page_alloc_image()`, `CPU_DATA_ALIGNMENT`, `cpu_detect_features()`, `CPU_FEATURE_PAE`, `cpu_init_data()`, `global_pmalloc_cache`, `IDT_PIC8259_BASE`, `init_pmalloc_cache()`, `KERNEL_PAGE_STACK_INIT`, `page_free()`, `pffree`, `pic8259_init()`, `vm_boot_init()`, `vm_boot_postinit()`, and `vm_lookup_kernel_paddr()`.

Referenced by `kmain()`.

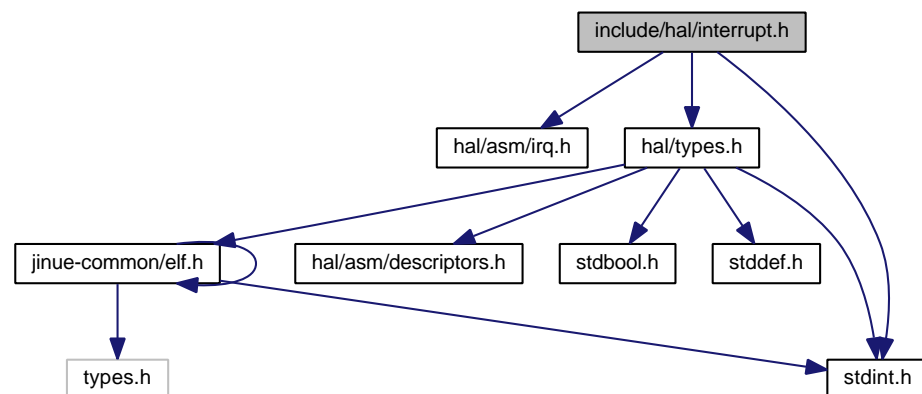
```

156                                     {
157     int idx;
158 
```

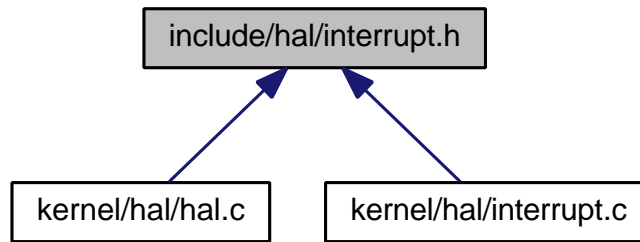
```
159  /* get cpu info */
160  cpu_detect_features();
161
162  /* allocate per-CPU data
163  *
164  * We need to ensure that the Task State Segment (TSS) contained in this
165  * memory block does not cross a page boundary. */
166  assert(sizeof(cpu_data_t) < CPU_DATA_ALIGNMENT);
167
168  cpu_data_t *cpu_data = boot_heap_alloc(boot_alloc, cpu_data_t,
CPU_DATA_ALIGNMENT);
169
170  /* initialize per-CPU data */
171  cpu_init_data(cpu_data);
172
173  /* Initialize interrupt descriptor table (IDT)
174  *
175  * This function modifies the IDT in-place (see trap.asm). This must be
176  * done before vm_boot_init() because the page protection bits set up by
177  * vm_boot_init() prevent this. */
178  hal_init_idt();
179
180  /* Initialize programmable interrupt controller. */
181  pic8259_init(IDT_PIC8259_BASE);
182
183  /* initialize virtual memory management, enable paging
184  *
185  * below this point, it is no longer safe to call pfalloc_early() */
186  bool use_pae = cpu_has_feature(CPU_FEATURE_PAE);
187  vm_boot_init(boot_info, use_pae, cpu_data, boot_alloc);
188
189  /* Initialize GDT and TSS */
190  hal_init_descriptors(cpu_data, boot_alloc);
191
192  /* initialize the page frame allocator */
193  kern_paddr_t *page_stack_buffer = (kern_paddr_t *)boot_page_alloc_image(boot_alloc);
194  init_pfalloc_cache(&global_pfalloc_cache, page_stack_buffer);
195
196  /* TODO Remove this once vm.c rework is done. */
197  for(idx = 0; idx < KERNEL_PAGE_STACK_INIT; ++idx) {
198      pffree(
199          vm_lookup_kernel_paddr(
200              boot_page_alloc_image(boot_alloc)));
201  }
202
203  /* Initialize virtual memory allocator and VM management caches. */
204  vm_boot_postinit(boot_info, boot_alloc, use_pae);
205
206  /* TODO Remove this once add page frame system call is implemented.
207  *
208  * The test user space program needs one page to create a new thread. */
209  page_free(boot_page_alloc(boot_alloc));
210
211  /* choose system call method */
212  hal_select_syscall_method();
213 }
```

[illegible]

```
#include <hal/asm/irq.h>
#include <hal/types.h>
#include <stdint.h>
Include dependency graph for interrupt.h:
```



This graph shows which files directly or indirectly include this file:



## Functions

- void **dispatch\_interrupt** (trapframe\_t \*trapframe)

## Variables

- **seg\_descriptor\_t** idt []

### 4.35.1 Function Documentation

#### 4.35.1.1 void dispatch\_interrupt ( trapframe\_t \* trapframe )

Definition at line 42 of file interrupt.c.

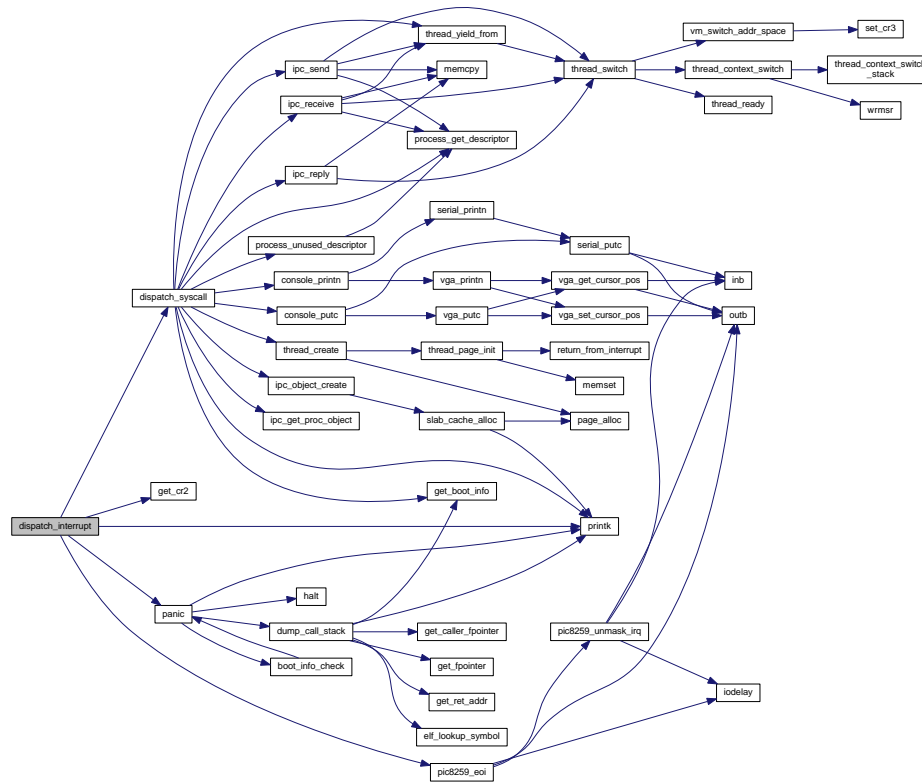
References `dispatch_syscall()`, `trapframe_t::eip`, `trapframe_t::errcode`, `get_cr2()`, `IDT_LAST_EXCEPTION`, `IDT_PIC8259_BASE`, `trapframe_t::ivt`, `panic()`, `pic8259_eoi()`, `PIC8259_IRQ_COUNT`, `printk()`, and `SYSCALL_IRQ`.

```

42     {
43     unsigned int    ivt      = trapframe->ivt;
44     uintptr_t      eip      = trapframe->eip;
45     uint32_t       errcode   = trapframe->errcode;
46
47     /* exceptions */
48     if(ivt <= IDT_LAST_EXCEPTION) {
49         printk("EXCEPT: %u cr2=0x%x errcode=0x%x eip=0x%x\n", ivt, get_cr2(), errcode, eip);
50
51         /* never returns */
52         panic("caught exception");
53     }
54
55     if(ivt == SYSCALL_IRQ) {
56         /* slow system call method */
57         dispatch_syscall(trapframe);
58     }
59     else if(ivt >= IDT_PIC8259_BASE && ivt < IDT_PIC8259_BASE +
60 PIC8259_IRQ_COUNT) {
61         int irq = ivt - IDT_PIC8259_BASE;
62         printk("IRQ: %u (vector %u)\n", irq, ivt);
63         pic8259_eoi(irq);
64     }
65     else {
66         printk("INTR: vector %u\n", ivt);
67     }
68 }

```

Here is the call graph for this function:



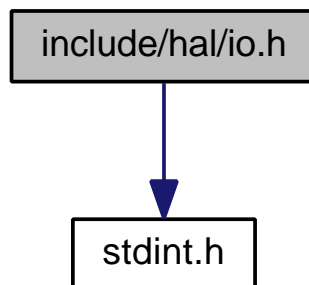
## 4.35.2 Variable Documentation

### 4.35.2.1 seg\_descriptor\_t idt[]

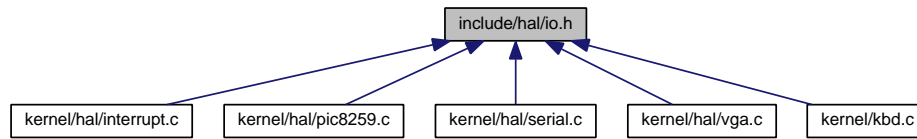
## 4.36 include/hal/io.h File Reference

```
#include <stdint.h>
```

Include dependency graph for io.h:



This graph shows which files directly or indirectly include this file:



## Functions

- **uint8\_t inb** (uint16\_t port)
- **uint16\_t inw** (uint16\_t port)
- **uint32\_t inl** (uint16\_t port)
- void **outb** (uint16\_t port, uint8\_t value)
- void **outw** (uint16\_t port, uint16\_t value)
- void **outl** (uint16\_t port, uint32\_t value)
- void **iodelay** (void)

### 4.36.1 Function Documentation

#### 4.36.1.1 uint8\_t inb ( uint16\_t port )

Referenced by any\_key(), pic8259\_mask\_irq(), pic8259\_unmask\_irq(), serial\_putc(), vga\_get\_cursor\_pos(), and vga\_init().

#### 4.36.1.2 uint32\_t inl ( uint16\_t port )

#### 4.36.1.3 uint16\_t inw ( uint16\_t port )

#### 4.36.1.4 void iodelay ( void )

Referenced by pic8259\_eoi(), pic8259\_init(), pic8259\_mask\_irq(), and pic8259\_unmask\_irq().

#### 4.36.1.5 void outb ( uint16\_t port, uint8\_t value )

Referenced by pic8259\_eoi(), pic8259\_init(), pic8259\_mask\_irq(), pic8259\_unmask\_irq(), serial\_init(), serial\_putc(), vga\_get\_cursor\_pos(), vga\_init(), and vga\_set\_cursor\_pos().

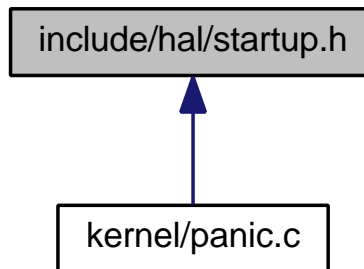
#### 4.36.1.6 void outl ( uint16\_t port, uint32\_t value )

#### 4.36.1.7 void outw ( uint16\_t port, uint16\_t value )



## 4.37 include/hal/startup.h File Reference

This graph shows which files directly or indirectly include this file:



### Functions

- void **halt** (void)

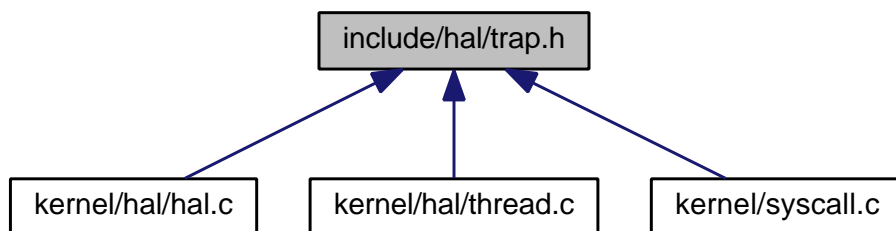
#### 4.37.1 Function Documentation

##### 4.37.1.1 void halt ( void )

Referenced by panic().

## 4.38 include/hal/trap.h File Reference

This graph shows which files directly or indirectly include this file:



### Functions

- void **fast\_intel\_entry** (void)  
*entry point for Intel fast system call mechanism (SYSENTER/SYSEXIT)*
- void **fast\_amd\_entry** (void)  
*entry point for AMD fast system call mechanism (SYSCALL/SYSRET)*
- void **return\_from\_interrupt** (void)

## Variables

- **int `syscall_method`**

*Specifies the entry point to use for system calls.*

### 4.38.1 Function Documentation

#### 4.38.1.1 `void fast_amd_entry ( void )`

entry point for AMD fast system call mechanism (SYSCALL/SYSRET)

#### 4.38.1.2 `void fast_intel_entry ( void )`

entry point for Intel fast system call mechanism (SYSENTER/SYSEXIT)

#### 4.38.1.3 `void return_from_interrupt ( void )`

Referenced by `thread_page_init()`.

### 4.38.2 Variable Documentation

#### 4.38.2.1 `int syscall_method`

Specifies the entry point to use for system calls.

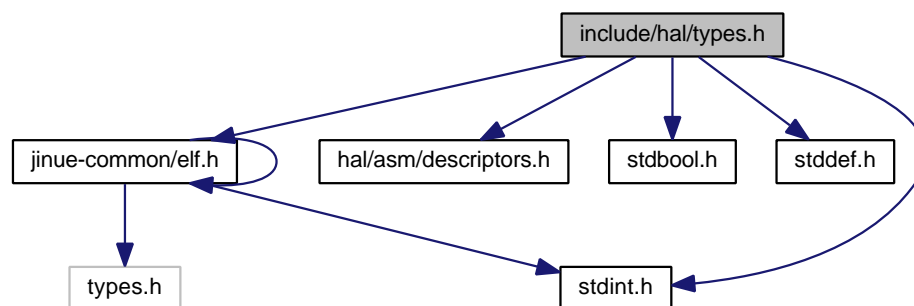
Definition at line 58 of file `hal.c`.

Referenced by `dispatch_syscall()`.

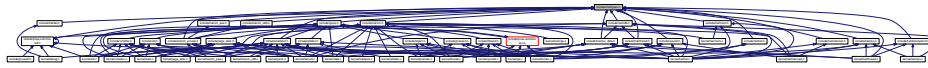
## 4.39 `include/hal/types.h` File Reference

```
#include <jinue-common/elf.h>
#include <hal/asm/descriptors.h>
#include <stdbool.h>
#include <stddef.h>
#include <stdint.h>
```

Include dependency graph for `types.h`:



This graph shows which files directly or indirectly include this file:



## Data Structures

- struct **thread\_context\_t**
- struct **addr\_space\_t**
- struct **e820\_t**
- struct **boot\_info\_t**
- struct **pseudo\_descriptor\_t**
- struct **tss\_t**
- struct **cpu\_data\_t**
- struct **trapframe\_t**
- struct **kernel\_context\_t**

## Macros

- #define **PFNULL** ((**kern\_paddr\_t**)-1)  
*an invalid page frame address used as null value*
- #define **msg\_arg0** `eax`
- #define **msg\_arg1** `ebx`
- #define **msg\_arg2** `esi`
- #define **msg\_arg3** `edi`

## Typedefs

- typedef unsigned char \* **addr\_t**  
*Virtual memory address (pointer) with pointer arithmetic allowed.*
- typedef uint32\_t **kern\_paddr\_t**  
*Physical memory address for use by the kernel.*
- typedef uint64\_t **user\_paddr\_t**  
*Physical memory address for use by user space.*
- typedef struct **pte\_t pte\_t**  
*type of a page table entry*
- typedef struct **pdpt\_t pdpt\_t**
- typedef uint64\_t **seg\_descriptor\_t**
- typedef uint32\_t **seg\_selector\_t**
- typedef struct **cpu\_data\_t cpu\_data\_t**

### 4.39.1 Macro Definition Documentation

#### 4.39.1.1 #define msg\_arg0 `eax`

Definition at line 195 of file types.h.

#### 4.39.1.2 `#define msg_arg1 ebx`

Definition at line 197 of file types.h.

#### 4.39.1.3 `#define msg_arg2 esi`

Definition at line 199 of file types.h.

#### 4.39.1.4 `#define msg_arg3 edi`

Definition at line 201 of file types.h.

#### 4.39.1.5 `#define PFNULL ((kern_paddr_t)-1)`

an invalid page frame address used as null value

Definition at line 51 of file types.h.

Referenced by `init_pmalloc_cache()`, and `remove_page_frame()`.

### 4.39.2 Typedef Documentation

#### 4.39.2.1 `typedef unsigned char* addr_t`

Virtual memory address (pointer) with pointer arithmetic allowed.

Definition at line 42 of file types.h.

#### 4.39.2.2 `typedef struct cpu_data_t cpu_data_t`

Definition at line 191 of file types.h.

#### 4.39.2.3 `typedef uint32_t kern_paddr_t`

Physical memory address for use by the kernel.

Definition at line 45 of file types.h.

#### 4.39.2.4 `typedef struct pdpt_t pdpt_t`

Definition at line 70 of file types.h.

#### 4.39.2.5 `typedef struct pte_t pte_t`

type of a page table entry

Definition at line 66 of file types.h.

## 4.39.2.6 typedef uint64\_t seg\_descriptor\_t

Definition at line 115 of file types.h.

## 4.39.2.7 typedef uint32\_t seg\_selector\_t

Definition at line 117 of file types.h.

## 4.39.2.8 typedef uint64\_t user\_paddr\_t

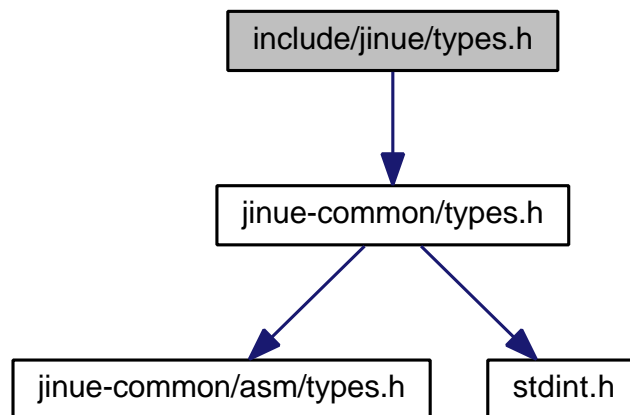
Physical memory address for use by user space.

Definition at line 48 of file types.h.

## 4.40 include/jinue/types.h File Reference

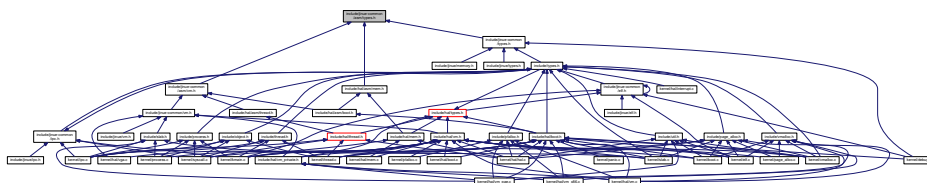
```
#include <jinue-common/types.h>
```

Include dependency graph for types.h:



## 4.41 include/jinue-common/asm/types.h File Reference

This graph shows which files directly or indirectly include this file:



## Macros

- #define **KB** (1024)

- `#define MB (1024 * KB)`
- `#define GB (1024 * MB)`

#### 4.41.1 Macro Definition Documentation

##### 4.41.1.1 `#define GB (1024 * MB)`

Definition at line 39 of file types.h.

Referenced by `mem_check_memory()`.

##### 4.41.1.2 `#define KB (1024)`

Definition at line 35 of file types.h.

##### 4.41.1.3 `#define MB (1024 * KB)`

Definition at line 37 of file types.h.

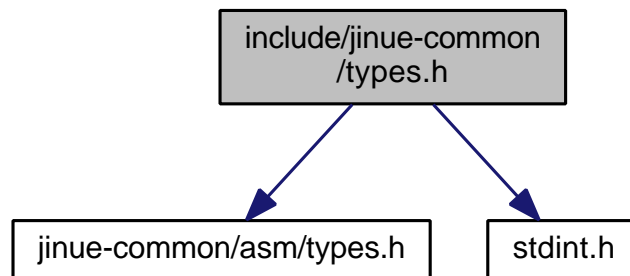
Referenced by `vm_boot_postinit()`.

#### 4.42 `include/jinue-common/types.h` File Reference

```
#include <jinue-common/asm/types.h>
```

```
#include <stdint.h>
```

Include dependency graph for types.h:



This graph shows which files directly or indirectly include this file:



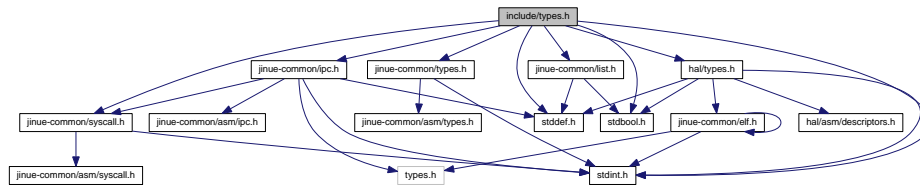
#### Data Structures

- struct `jinue_mem_entry_t`
- struct `jinue_mem_map_t`

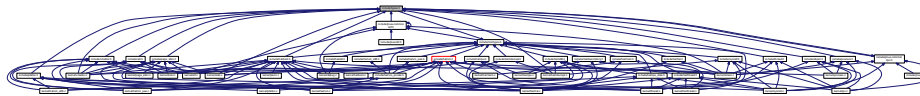
#### 4.43 include/types.h File Reference

```
#include <jinue-common/ipc.h>
#include <jinue-common/list.h>
#include <jinue-common/syscall.h>
#include <jinue-common/types.h>
#include <hal/types.h>
#include <stddef.h>
#include <stdint.h>
#include <stdbool.h>
```

Include dependency graph for types.h:



This graph shows which files directly or indirectly include this file:



## Data Structures

- struct **boot\_heap\_pushed\_state**
- struct **boot\_alloc\_t**
- struct **object\_header\_t**
- struct **object\_ref\_t**
- struct **process\_t**
- struct **message\_info\_t**
- struct **thread\_t**
- struct **ipc\_t**

## Macros

- `#define PROCESS_MAX_DESCRIPTOR 12`

## Typedefs

- typedef struct **thread\_t** thread\_t

### 4.43.1 Macro Definition Documentation

#### 4.43.1.1 #define PROCESS\_MAX\_DESCRIPTOR 12

Definition at line 70 of file types.h.

Referenced by process\_get\_descriptor(), and process\_unused\_descriptor().

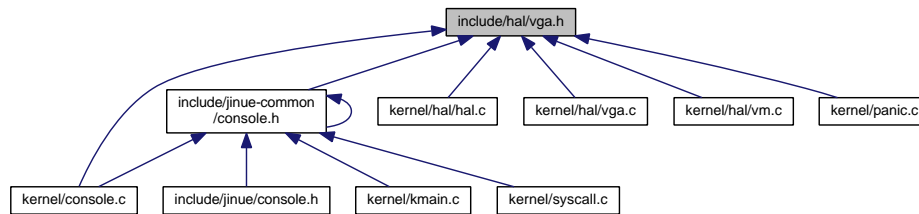
### 4.43.2 Typedef Documentation

#### 4.43.2.1 typedef struct thread\_t thread\_t

Definition at line 98 of file types.h.

## 4.44 include/hal/vga.h File Reference

This graph shows which files directly or indirectly include this file:



## Macros

- #define **VGA\_TEXT\_VID\_BASE** 0xb8000
- #define **VGA\_TEXT\_VID\_TOP** 0xc0000
- #define **VGA\_TEXT\_VID\_SIZE** (VGA\_TEXT\_VID\_TOP - VGA\_TEXT\_VID\_BASE)
- #define **VGA\_MISC\_OUT\_WR** 0x3c2
- #define **VGA\_MISC\_OUT\_RD** 0x3cc
- #define **VGA\_CRTC\_ADDR** 0x3d4
- #define **VGA\_CRTC\_DATA** 0x3d5
- #define **VGA\_FB\_FLAG\_ACTIVE** 1
- #define **VGA\_COLOR\_BLACK** 0x00
- #define **VGA\_COLOR\_BLUE** 0x01
- #define **VGA\_COLOR\_GREEN** 0x02
- #define **VGA\_COLOR\_CYAN** 0x03
- #define **VGA\_COLOR\_RED** 0x04
- #define **VGA\_COLOR\_MAGENTA** 0x05
- #define **VGA\_COLOR\_BROWN** 0x06
- #define **VGA\_COLOR\_WHITE** 0x07
- #define **VGA\_COLOR\_GRAY** 0x08
- #define **VGA\_COLOR\_BRIGHTBLUE** 0x09
- #define **VGA\_COLOR\_BRIGHTGREEN** 0x0a
- #define **VGA\_COLOR\_BRIGHTCYAN** 0x0b
- #define **VGA\_COLOR\_BRIGHTRED** 0x0c
- #define **VGA\_COLOR\_BRIGHTMAGENTA** 0x0d
- #define **VGA\_COLOR\_YELLOW** 0x0e



- `#define VGA_COLOR_BRIGHTWHITE 0x0f`
- `#define VGA_COLOR_ERASE VGA_COLOR_RED`
- `#define VGA_LINES 25`
- `#define VGA_WIDTH 80`
- `#define VGA_TAB_WIDTH 8`
- `#define VGA_LINE(x) ((x) / (VGA_WIDTH))`
- `#define VGA_COL(x) ((x) % (VGA_WIDTH))`

## Typedefs

- typedef unsigned int **vga\_pos\_t**

## Functions

- void **vga\_init** (void)
- void **vga\_set\_base\_addr** (void \*base\_addr)
- void **vga\_clear** (void)
- void **vga\_print** (const char \*message, int colour)
- void **vga\_printn** (const char \*message, unsigned int n, int colour)
- void **vga\_putc** (char c, int colour)
- void **vga\_scroll** (void)
- **vga\_pos\_t vga\_get\_cursor\_pos** (void)
- void **vga\_set\_cursor\_pos** (**vga\_pos\_t** pos)

### 4.44.1 Macro Definition Documentation

#### 4.44.1.1 `#define VGA_COL( x ) ((x) % (VGA_WIDTH))`

Definition at line 68 of file vga.h.

#### 4.44.1.2 `#define VGA_COLOR_BLACK 0x00`

Definition at line 45 of file vga.h.

#### 4.44.1.3 `#define VGA_COLOR_BLUE 0x01`

Definition at line 46 of file vga.h.

#### 4.44.1.4 `#define VGA_COLOR_BRIGHTBLUE 0x09`

Definition at line 54 of file vga.h.

#### 4.44.1.5 `#define VGA_COLOR_BRIGHTCYAN 0x0b`

Definition at line 56 of file vga.h.

4.44.1.6 `#define VGA_COLOR_BRIGHTGREEN 0x0a`

Definition at line 55 of file vga.h.

4.44.1.7 `#define VGA_COLOR_BRIGHTMAGENTA 0x0d`

Definition at line 58 of file vga.h.

4.44.1.8 `#define VGA_COLOR_BRIGHTRED 0x0c`

Definition at line 57 of file vga.h.

4.44.1.9 `#define VGA_COLOR_BRIGHTWHITE 0x0f`

Definition at line 60 of file vga.h.

4.44.1.10 `#define VGA_COLOR_BROWN 0x06`

Definition at line 51 of file vga.h.

4.44.1.11 `#define VGA_COLOR_CYAN 0x03`

Definition at line 48 of file vga.h.

4.44.1.12 `#define VGA_COLOR_ERASE VGA_COLOR_RED`

Definition at line 61 of file vga.h.

Referenced by `vga_clear()`, and `vga_scroll()`.

4.44.1.13 `#define VGA_COLOR_GRAY 0x08`

Definition at line 53 of file vga.h.

4.44.1.14 `#define VGA_COLOR_GREEN 0x02`

Definition at line 47 of file vga.h.

4.44.1.15 `#define VGA_COLOR_MAGENTA 0x05`

Definition at line 50 of file vga.h.

4.44.1.16 `#define VGA_COLOR_RED 0x04`

Definition at line 49 of file vga.h.

Referenced by `panic()`.

**4.44.1.17 #define VGA\_COLOR\_WHITE 0x07**

Definition at line 52 of file vga.h.

**4.44.1.18 #define VGA\_COLOR\_YELLOW 0x0e**

Definition at line 59 of file vga.h.

Referenced by kmain().

**4.44.1.19 #define VGA\_CRTC\_ADDR 0x3d4**

Definition at line 40 of file vga.h.

Referenced by vga\_get\_cursor\_pos(), vga\_init(), and vga\_set\_cursor\_pos().

**4.44.1.20 #define VGA\_CRTC\_DATA 0x3d5**

Definition at line 41 of file vga.h.

Referenced by vga\_get\_cursor\_pos(), vga\_init(), and vga\_set\_cursor\_pos().

**4.44.1.21 #define VGA\_FB\_FLAG\_ACTIVE 1**

Definition at line 43 of file vga.h.

**4.44.1.22 #define VGA\_LINE( x )((x) / (VGA\_WIDTH))**

Definition at line 67 of file vga.h.

**4.44.1.23 #define VGA\_LINES 25**

Definition at line 63 of file vga.h.

Referenced by vga\_clear(), and vga\_scroll().

**4.44.1.24 #define VGA\_MISC\_OUT\_RD 0x3cc**

Definition at line 39 of file vga.h.

Referenced by vga\_init().

**4.44.1.25 #define VGA\_MISC\_OUT\_WR 0x3c2**

Definition at line 38 of file vga.h.

Referenced by vga\_init().

**4.44.1.26 #define VGA\_TAB\_WIDTH 8**

Definition at line 65 of file vga.h.

#### 4.44.1.27 `#define VGA_TEXT_VID_BASE 0xb8000`

Definition at line 35 of file vga.h.

Referenced by `vm_boot_init()`.

#### 4.44.1.28 `#define VGA_TEXT_VID_SIZE (VGA_TEXT_VID_TOP - VGA_TEXT_VID_BASE)`

Definition at line 37 of file vga.h.

#### 4.44.1.29 `#define VGA_TEXT_VID_TOP 0xc0000`

Definition at line 36 of file vga.h.

Referenced by `vm_boot_init()`.

#### 4.44.1.30 `#define VGA_WIDTH 80`

Definition at line 64 of file vga.h.

Referenced by `vga_clear()`, and `vga_scroll()`.

### 4.44.2 Typedef Documentation

#### 4.44.2.1 `typedef unsigned int vga_pos_t`

Definition at line 71 of file vga.h.

### 4.44.3 Function Documentation

#### 4.44.3.1 `void vga_clear ( void )`

Definition at line 65 of file vga.c.

References `VGA_COLOR_ERASE`, `VGA_LINES`, and `VGA_WIDTH`.

Referenced by `vga_init()`.

```
65     {
66         unsigned int idx = 0;
67
68         while( idx < (VGA_LINES * VGA_WIDTH * 2) ) {
69             video_base_addr[idx++] = 0x20;
70             video_base_addr[idx++] = VGA_COLOR_ERASE;
71         }
72     }
```

#### 4.44.3.2 `vga_pos_t vga_get_cursor_pos ( void )`

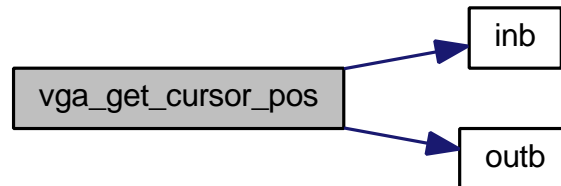
Definition at line 89 of file vga.c.

References `inb()`, `outb()`, `VGA_CRTC_ADDR`, and `VGA_CRTC_DATA`.

Referenced by `vga_print()`, `vga_printn()`, and `vga_putc()`.

```
89     {
90     unsigned char h, l;
91
92     outb(VGA_CRTC_ADDR, 0x0e);
93     h = inb(VGA_CRTC_DATA);
94     outb(VGA_CRTC_ADDR, 0x0f);
95     l = inb(VGA_CRTC_DATA);
96
97     return (h << 8) | l;
98 }
```

Here is the call graph for this function:



#### 4.44.3.3 void vga\_init ( void )

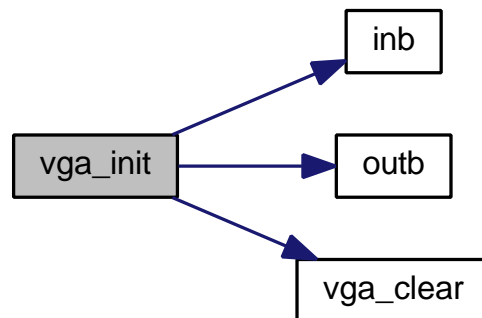
Definition at line 43 of file vga.c.

References `inb()`, `outb()`, `vga_clear()`, `VGA_CRTC_ADDR`, `VGA_CRTC_DATA`, `VGA_MISC_OUT_RD`, and `VGA_MISC_OUT_WR`.

Referenced by `console_init()`.

```
43     {
44     unsigned char data;
45
46     /* Set address select bit in a known state: CRTC regs at 0x3dx */
47     data = inb(VGA_MISC_OUT_RD);
48     data |= 1;
49     outb(VGA_MISC_OUT_WR, data);
50
51     /* Move cursor to line 0 col 0 */
52     outb(VGA_CRTC_ADDR, 0x0e);
53     outb(VGA_CRTC_DATA, 0x0);
54     outb(VGA_CRTC_ADDR, 0x0f);
55     outb(VGA_CRTC_DATA, 0x0);
56
57     /* Clear the screen */
58     vga_clear();
59 }
```

Here is the call graph for this function:



#### 4.44.3.4 void vga\_print ( const char \* message, int colour )

Definition at line 111 of file vga.c.

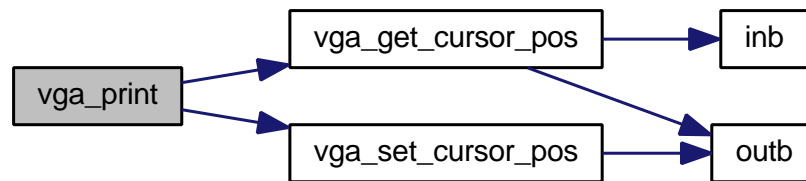
References vga\_get\_cursor\_pos(), and vga\_set\_cursor\_pos().

```

111     {
112         unsigned short int pos = vga_get_cursor_pos();
113         char c;
114
115         while( (c = *(message++)) ) {
116             pos = vga_raw_putc(c, pos, colour);
117         }
118
119         vga_set_cursor_pos(pos);
120     }

```

Here is the call graph for this function:



#### 4.44.3.5 void vga\_printn ( const char \* message, unsigned int n, int colour )

Definition at line 122 of file vga.c.

References vga\_get\_cursor\_pos(), and vga\_set\_cursor\_pos().

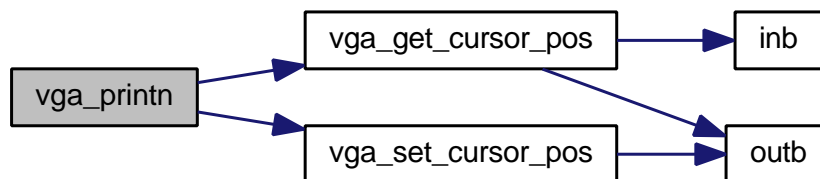
Referenced by console\_printn().

```

122     {
123         vga_pos_t pos = vga_get_cursor_pos();
124         char c;
125
126         while(n) {
127             c = *(message++);
128             pos = vga_raw_putc(c, pos, colour);
129             --n;
130         }
131
132         vga_set_cursor_pos(pos);
133     }

```

Here is the call graph for this function:



4.44.3.6 void vga\_putc ( char *c*, int *colour* )

Definition at line 135 of file vga.c.

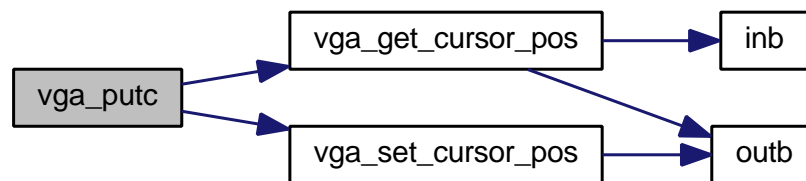
References `vga_get_cursor_pos()`, and `vga_set_cursor_pos()`.

Referenced by `console_putc()`.

```

135     {
136     vga_pos_t pos = vga_get_cursor_pos();
137
138     pos = vga_raw_putc(c, pos, colour);
139
140     vga_set_cursor_pos(pos);
141 }
```

Here is the call graph for this function:



## 4.44.3.7 void vga\_scroll ( void )

Definition at line 74 of file vga.c.

References `VGA_COLOR_ERASE`, `VGA_LINES`, and `VGA_WIDTH`.

```

74     {
75     unsigned char *di = video_base_addr;
76     unsigned char *si = video_base_addr + 2 * VGA_WIDTH;
77     unsigned int idx;
78
79     for(idx = 0; idx < 2 * VGA_WIDTH * (VGA_LINES - 1); ++idx) {
80         *(di++) = *(si++);
81     }
82
83     for(idx = 0; idx < VGA_WIDTH; ++idx) {
84         *(di++) = 0x20;
85         *(di++) = VGA_COLOR_ERASE;
86     }
87 }
```

4.44.3.8 void vga\_set\_base\_addr ( void \* *base\_addr* )

Definition at line 61 of file vga.c.

Referenced by `vm_boot_init()`.

```

61     {
62     video_base_addr = base_addr;
63 }
```

#### 4.44.3.9 void vga\_set\_cursor\_pos ( vga\_pos\_t pos )

Definition at line 100 of file vga.c.

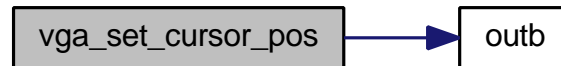
References outb(), VGA\_CRTC\_ADDR, and VGA\_CRTC\_DATA.

Referenced by vga\_print(), vga\_printn(), and vga\_putc().

```

100
101     unsigned char h = pos >> 8;      {
102     unsigned char l = pos;
103
104     outb(VGA_CRTC_ADDR, 0x0e);
105     outb(VGA_CRTC_DATA, h);
106     outb(VGA_CRTC_ADDR, 0x0f);
107     outb(VGA_CRTC_DATA, l);
108 }
```

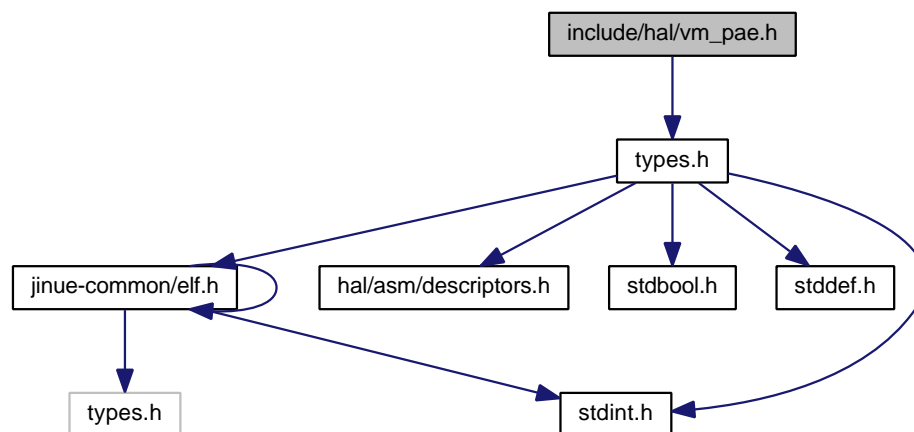
Here is the call graph for this function:



#### 4.45 include/hal/vm\_pae.h File Reference

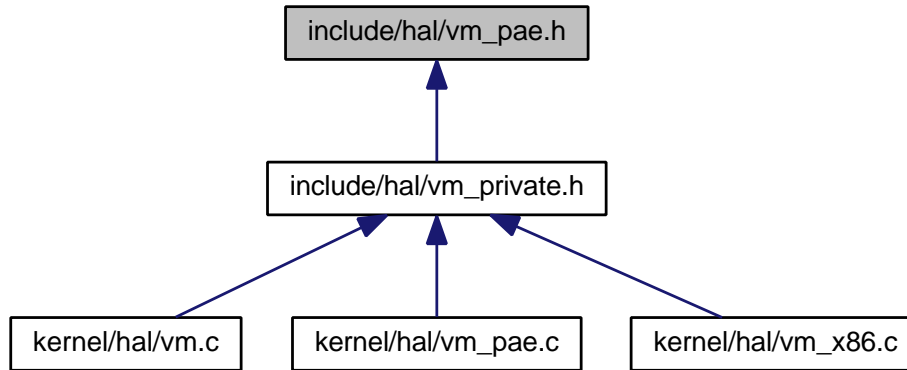
```
#include <types.h>
```

Include dependency graph for vm\_pae.h:





This graph shows which files directly or indirectly include this file:



## Functions

- void **vm\_pae\_boot\_init** (void)
  - This header file contains declarations for the PAE functions defined in **hal/vm\_pae.c** (p. 342).*
- **pte\_t \* vm\_pae\_lookup\_page\_directory** (**addr\_space\_t** \*addr\_space, void \*addr, **bool** create\_as\_needed)
  - Lookup and map the page directory for a specified address and address space.*
- unsigned int **vm\_pae\_page\_table\_offset\_of** (**addr\_t** addr)
- unsigned int **vm\_pae\_page\_directory\_offset\_of** (**addr\_t** addr)
- **pte\_t \* vm\_pae\_get\_pte\_with\_offset** (**pte\_t** \*pte, unsigned int offset)
- void **vm\_pae\_set\_pte** (**pte\_t** \*pte, **uint64\_t** paddr, int flags)
  - TODO handle flag bit position > 31 for NX bit support.*
- void **vm\_pae\_set\_pte\_flags** (**pte\_t** \*pte, int flags)
  - TODO handle flag bit position > 31 for NX bit support.*
- int **vm\_pae\_get\_pte\_flags** (const **pte\_t** \*pte)
- **uint64\_t** **vm\_pae\_get\_pte\_paddr** (const **pte\_t** \*pte)
  - TODO mask NX bit as well, maximum 52 bits supported.*
- void **vm\_pae\_clear\_pte** (**pte\_t** \*pte)
- void **vm\_pae\_copy\_pte** (**pte\_t** \*dest, const **pte\_t** \*src)
- **addr\_space\_t \* vm\_pae\_create\_addr\_space** (**addr\_space\_t** \*addr\_space)
- **addr\_space\_t \* vm\_pae\_create\_initial\_addr\_space** (**boot\_alloc\_t** \*boot\_alloc)
- void **vm\_pae\_destroy\_addr\_space** (**addr\_space\_t** \*addr\_space)
- void **vm\_pae\_create\_pdpt\_cache** (**boot\_alloc\_t** \*boot\_alloc)
- void **vm\_pae\_unmap\_low\_alias** (**addr\_space\_t** \*addr\_space)

### 4.45.1 Function Documentation

#### 4.45.1.1 void vm\_pae\_boot\_init ( void )

This header file contains declarations for the PAE functions defined in **hal/vm\_pae.c** (p. 342).

It is intended to be included by **hal/vm.c** (p. 331) and **hal/vm\_pae.c** (p. 342). There should be no reason to include it anywhere else.

Definition at line 72 of file **vm\_pae.c**.

References **PAGE\_TABLE\_ENTRIES**, and **page\_table\_entries**.

Referenced by `vm_boot_init()`.

```

72         {
73     page_table_entries = (size_t)PAGE_TABLE_ENTRIES;
74 }

```

#### 4.45.1.2 void vm\_pae\_clear\_pte ( pte\_t \* pte )

Definition at line 150 of file `vm_pae.c`.

References `pte_t::entry`.

Referenced by `vm_pae_create_addr_space()`, `vm_pae_create_initial_addr_space()`, and `vm_pae_unmap_low_alias()`.

```

150                                     {
151     pte->entry = 0;
152 }

```

#### 4.45.1.3 void vm\_pae\_copy\_pte ( pte\_t \* dest, const pte\_t \* src )

Definition at line 154 of file `vm_pae.c`.

References `pte_t::entry`.

Referenced by `vm_pae_create_addr_space()`.

```

154                                     {
155     dest->entry = src->entry;
156 }

```

#### 4.45.1.4 addr\_space\_t \* vm\_pae\_create\_addr\_space ( addr\_space\_t \* addr\_space )

Definition at line 170 of file `vm_pae.c`.

References `addr_space_t::cr3`, `initial_addr_space`, `KLIMIT`, `NULL`, `page_address_of`, `page_offset_of`, `pdpt_t::pd`, `addr_space_t::pdpt`, `PDPT_ENTRIES`, `slab_cache_alloc()`, `addr_space_t::top_level`, `vm_lookup_kernel_paddr()`, `vm_pae_clear_pte()`, and `vm_pae_copy_pte()`.

Referenced by `vm_create_addr_space()`.

```

170                                     {
171     unsigned int idx;
172     pte_t *pdpte;
173
174     /* Create a PDPT for the new address space */
175     pdpt_t *pdpt = slab_cache_alloc(&pdpt_cache);
176
177     if(pdpt == NULL) {
178         return NULL;
179     }
180
181     /* Use the initial address space as a template for the kernel address range
182      * (address KLIMIT and above). The page tables for that range are shared by
183      * all address spaces. */
184     pdpt_t *template_pdpt = initial_addr_space.top_level.pdpt;
185
186     for(idx = 0; idx < PDPT_ENTRIES; ++idx) {
187         pdpte = &pdpt->pd[idx];
188
189         if(idx < pdpt_offset_of((addr_t)KLIMIT)) {
190             /* This PDPT entry describes an address range entirely under KLIMIT
191              * so it is all user space: do not create a page directory at this

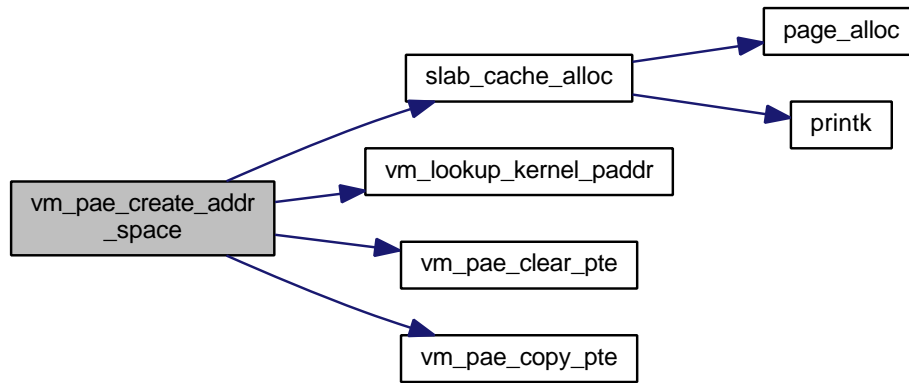
```

```

192         * time. */
193         vm_pae_clear_pte(pdpte);
194     }
195     else {
196         /* This page directory describes an address range entirely above
197          * KLIMIT: share the template's page directory. */
198         vm_pae_copy_pte(pdpte, &template_pdpt->pd[idx]);
199     }
200 }
201
202 /* Lookup the physical address of the page where the PDPT resides. */
203 kern_paddr_t pdpt_page_paddr = vm_lookup_kernel_paddr((addr_t)
page_address_of(pdpt));
204
205 /* physical address of PDPT */
206 kern_paddr_t pdpt_paddr = pdpt_page_paddr | page_offset_of(pdpt);
207
208 addr_space->top_level.pdpt = pdpt;
209 addr_space->cr3 = pdpt_paddr;
210
211 return addr_space;
212 }

```

Here is the call graph for this function:



#### 4.45.1.5 `addr_space_t* vm_pae_create_initial_addr_space ( boot_alloc_t* boot_alloc )`

Definition at line 258 of file `vm_pae.c`.

References `boot_heap_alloc`, `boot_page_alloc_early()`, `addr_space_t::cr3`, `EARLY_PHYS_TO_VIRT`, `EARLY_PTR_TO_PHYS_ADDR`, `EARLY_VIRT_TO_PHYS`, `initial_addr_space`, `initial_pdpt`, `KERNEL_PREALLOC_LIMIT`, `KLIMIT`, `page_table_entries`, `pdpt_t::pd`, `addr_space_t::pdpt`, `PDPT_ENTRIES`, `addr_space_t::top_level`, `VM_FLAG_PRESENT`, `vm_init_initial_page_directory()`, `vm_pae_clear_pte()`, `vm_pae_get_pte_paddr()`, `vm_pae_page_directory_offset_of()`, and `vm_pae_set_pte()`.

Referenced by `vm_create_initial_addr_space()`.

```

258     {
259
260         unsigned int idx;
261
262         /* Allocate initial PDPT. PDPT must be 32-byte aligned. */
263         initial_pdpt = boot_heap_alloc(boot_alloc, pdpt_t, 32);
264
265         /* We want the pre-allocated kernel page tables to be contiguous. For this
266          * reason, we allocate the page directories first, and then the page tables.
267          *
268          * This function allocates pages in this order:
269          * +-----+-----+-----+-----+-----+
270          * | Low alias | pre-allocated | pre-allocated |

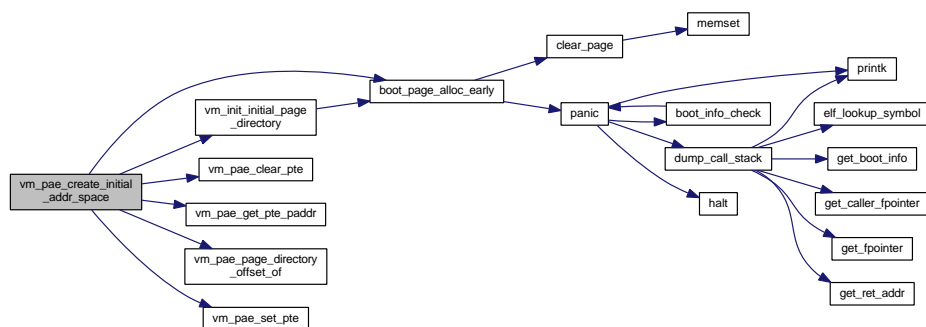
```

```

271      *      | page directory |      kernel      |      kernel      |
272      *      | and page table | page directories | page tables   |
273      *      +-----+-----+-----+-----+
274      */
275
276  for(idx = 0; idx < PDPT_ENTRIES; ++idx) {
277      vm_pae_clear_pte(&initial_pdpt->pd[idx]);
278  }
279
280  vm_pae_init_low_alias(initial_pdpt, boot_alloc);
281
282  const unsigned int last_idx = pdpt_offset_of((addr_t)KERNEL_PREALLOC_LIMIT - 1);
283
284  for(idx = pdpt_offset_of((addr_t)KLIMIT); idx <= last_idx; ++idx) {
285      pte_t *const pdpte = &initial_pdpt->pd[idx];
286      pte_t *page_directory = (pte_t *)boot_page_alloc_early(boot_alloc);
287
288      vm_pae_set_pte(
289          pdpte,
290          EARLY_PTR_TO_PHYS_ADDR(page_directory),
291          VM_FLAG_PRESENT);
292  }
293
294  for(idx = pdpt_offset_of((addr_t)KLIMIT); idx <= last_idx; ++idx) {
295      unsigned int end_index;
296
297      pte_t *const pdpte = &initial_pdpt->pd[idx];
298      pte_t *const page_directory = (pte_t *)EARLY_PHYS_TO_VIRT(
vm_pae_get_pte_paddr(pdpte));
299
300      if(idx < pdpt_offset_of((addr_t)KERNEL_PREALLOC_LIMIT)) {
301          end_index = page_table_entries;
302      }
303      else {
304          end_index = vm_pae_page_directory_offset_of((addr_t)KERNEL_PREALLOC_LIMIT);
305      }
306
307      vm_init_initial_page_directory(
308          page_directory,
309          boot_alloc,
310          0,
311          end_index,
312          idx == pdpt_offset_of((addr_t)KLIMIT));
313  }
314
315  initial_addr_space.top_level.pdpt = initial_pdpt;
316  initial_addr_space.cr3 = EARLY_VIRT_TO_PHYS(initial_pdpt);
317
318  return &initial_addr_space;
319 }

```

Here is the call graph for this function:



#### 4.45.1.6 void vm\_pae\_create\_pdpt\_cache ( boot\_alloc\_t \* boot\_alloc )

Definition at line 158 of file vm\_pae.c.

References NULL, slab\_cache\_init(), and SLAB\_DEFAULTS.

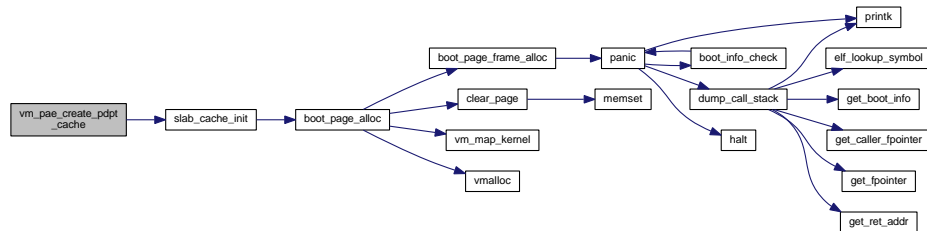
Referenced by vm\_boot\_postinit().

```

158                                     {
159     slab_cache_init(
160         &pdpt_cache,
161         "vm_pae_pdpt_cache",
162         sizeof(pdpt_t),
163         sizeof(pdpt_t),
164         NULL,
165         NULL,
166         SLAB_DEFAULTS,
167         boot_alloc);
168 }

```

Here is the call graph for this function:



#### 4.45.1.7 void vm\_pae\_destroy\_addr\_space ( addr\_space\_t \* addr\_space )

Definition at line 321 of file vm\_pae.c.

References `pte_t::entry`, `KLIMIT`, `pdpt_t::pd`, `addr_space_t::pdpt`, `PDPT_ENTRIES`, `slab_cache_free()`, `addr_space_t::top_level`, `vm_destroy_page_directory()`, `VM_FLAG_PRESENT`, and `vm_pae_get_pte_paddr()`.

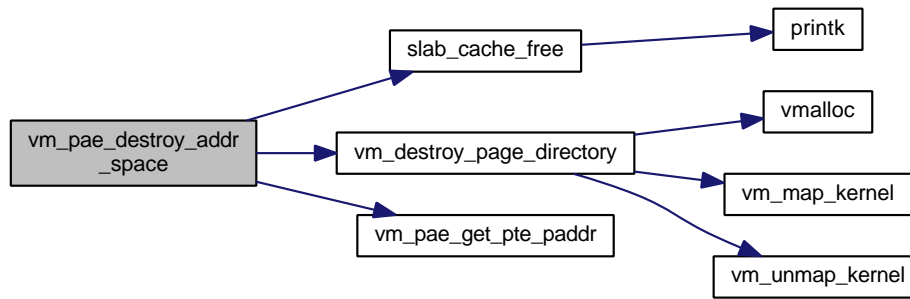
Referenced by `vm_destroy_addr_space()`.

```

321                                     {
322     unsigned int idx;
323     pte_t pdpte;
324
325     pdpt_t *pdpt = addr_space->top_level.pdpt;
326
327     for(idx = 0; idx < PDPT_ENTRIES; ++idx) {
328         pdpte.entry = pdpt->pd[idx].entry;
329
330         if(idx < pdpt_offset_of((addr_t)KLIMIT)) {
331             /* This page directory describes an address range entirely under
332              * KLIMIT so it is all user space: free all page tables in this
333              * page directory as well as the page directory itself. */
334             if(pdpte.entry & VM_FLAG_PRESENT) {
335                 vm_destroy_page_directory(
336                     vm_pae_get_pte_paddr(&pdpte),
337                     0,
338                     page_table_entries);
339             }
340         }
341         else {
342             /* This page directory describes an address range entirely above
343              * KLIMIT: do nothing.
344              *
345              * The page directory must not be freed because it is shared by all
346              * address spaces. */
347         }
348     }
349     slab_cache_free(pdpt);
350 }
351 }

```

Here is the call graph for this function:



#### 4.45.1.8 int vm\_pae\_get\_pte\_flags ( const pte\_t \* pte )

Definition at line 141 of file vm\_pae.c.

References pte\_t::entry, and PAGE\_MASK.

Referenced by vm\_pae\_lookup\_page\_directory().

```

141                                     {
142     return pte->entry & PAGE_MASK;
143 }
```

#### 4.45.1.9 uint64\_t vm\_pae\_get\_pte\_paddr ( const pte\_t \* pte )

TODO mask NX bit as well, maximum 52 bits supported.

Definition at line 146 of file vm\_pae.c.

References pte\_t::entry, and PAGE\_MASK.

Referenced by vm\_pae\_create\_initial\_addr\_space(), vm\_pae\_destroy\_addr\_space(), and vm\_pae\_lookup\_page\_directory().

```

146                                     {
147     return (pte->entry & ~(uint64_t)PAGE_MASK);
148 }
```

#### 4.45.1.10 pte\_t\* vm\_pae\_get\_pte\_with\_offset ( pte\_t \* pte, unsigned int offset )

Definition at line 127 of file vm\_pae.c.

```

127                                     {
128     return &pte[offset];
129 }
```

#### 4.45.1.11 pte\_t\* vm\_pae\_lookup\_page\_directory ( addr\_space\_t \* addr\_space, void \* addr, bool create\_as\_needed )

Lookup and map the page directory for a specified address and address space.

Important note: it is the caller's responsibility to unmap and free the returned page directory when it is done with it.

## Parameters

<i>addr_space</i>	address space in which the address is looked up.
<i>addr</i>	address to look up
<i>create_as_need</i>	Whether a page table is allocated if it does not exist

Definition at line 86 of file vm\_pae.c.

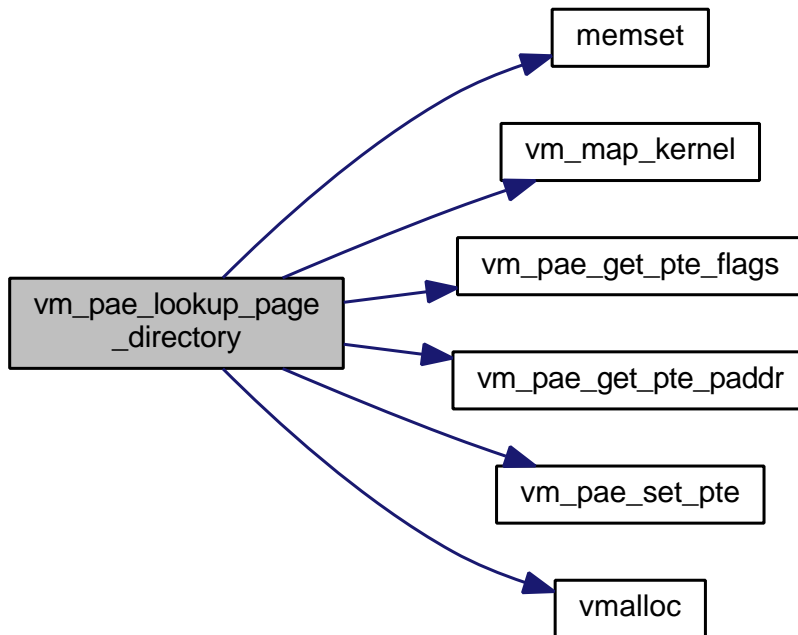
References `memset()`, `NULL`, `PAGE_SIZE`, `pdpt_t::pd`, `addr_space_t::pdpt`, `pfalloc`, `addr_space_t::top_level`, `VM_FLAG_PRESENT`, `VM_FLAG_READ_WRITE`, `vm_map_kernel()`, `vm_pae_get_pte_flags()`, `vm_pae_get_pte_paddr()`, `vm_pae_set_pte()`, and `vmalloc()`.

```

86                                     {
87     pdpt_t *pdpt    = addr_space->top_level.pdpt;
88     pte_t *pdpte    = &pdpt->pd[pdpt_offset_of(addr)];
89
90     if (vm_pae_get_pte_flags(pdpte) & VM_FLAG_PRESENT) {
91         /* map page directory */
92         pte_t *page_directory = (pte_t *)vmalloc();
93         vm_map_kernel((addr_t)page_directory, vm_pae_get_pte_paddr(pdpte),
VM_FLAG_READ_WRITE);
94
95         return page_directory;
96     }
97     else {
98         if (create_as_needed) {
99             /* allocate a new page directory and map it */
100             pte_t *page_directory = (pte_t *)vmalloc();
101             kern_paddr_t pgdir_paddr = pfalloc();
102
103             vm_map_kernel((addr_t)page_directory, pgdir_paddr,
VM_FLAG_READ_WRITE);
104
105             /* zero content of page directory */
106             memset(page_directory, 0, PAGE_SIZE);
107
108             /* link page directory in PDPT */
109             vm_pae_set_pte(pdpte, pgdir_paddr, VM_FLAG_PRESENT);
110
111             return page_directory;
112         }
113         else {
114             return NULL;
115         }
116     }
117 }

```

Here is the call graph for this function:



#### 4.45.1.12 unsigned int vm\_pae\_page\_directory\_offset\_of ( addr\_t addr )

Definition at line 123 of file `vm_pae.c`.

References `PAGE_DIRECTORY_OFFSET_OF`.

Referenced by `vm_pae_create_initial_addr_space()`.

```

123                                     {
124     return PAGE_DIRECTORY_OFFSET_OF(addr);
125 }
```

#### 4.45.1.13 unsigned int vm\_pae\_page\_table\_offset\_of ( addr\_t addr )

Definition at line 119 of file `vm_pae.c`.

References `PAGE_TABLE_OFFSET_OF`.

```

119                                     {
120     return PAGE_TABLE_OFFSET_OF(addr);
121 }
```

#### 4.45.1.14 void vm\_pae\_set\_pte ( pte\_t\* pte, uint64\_t paddr, int flags )

TODO handle flag bit position > 31 for NX bit support.

Definition at line 132 of file `vm_pae.c`.

References `pte_t::entry`.

Referenced by `vm_pae_create_initial_addr_space()`, and `vm_pae_lookup_page_directory()`.



```

132                                     {
133     pte->entry = paddr | flags;
134 }

```

#### 4.45.1.15 void vm\_pae\_set\_pte\_flags ( pte\_t \* pte, int flags )

TODO handle flag bit position > 31 for NX bit support.

Definition at line 137 of file vm\_pae.c.

References pte\_t::entry, and PAGE\_MASK.

```

137                                     {
138     pte->entry = (pte->entry & ~(uint64_t)PAGE_MASK) | flags;
139 }

```

#### 4.45.1.16 void vm\_pae\_unmap\_low\_alias ( addr\_space\_t \* addr\_space )

Definition at line 353 of file vm\_pae.c.

References pdpt\_t::pd, addr\_space\_t::pdpt, addr\_space\_t::top\_level, and vm\_pae\_clear\_pte().

Referenced by vm\_boot\_init().

```

353                                     {
354     /* Enabling PAE requires disabling paging temporarily, which in turn requires
355      * an alias of the kernel image region at address 0 to match its physical
356      * address. This function gets rid of this alias once PAE is enabled.
357      *
358      * There is no need for TLB invalidation because the caller reloads CR3 just
359      * after calling this function. */
360     vm_pae_clear_pte(&addr_space->top_level.pdpt->pd[0]);
361 }

```

Here is the call graph for this function:



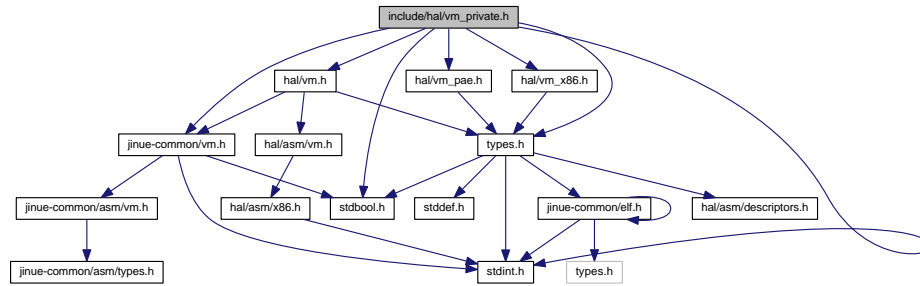
## 4.46 include/hal/vm\_private.h File Reference

```

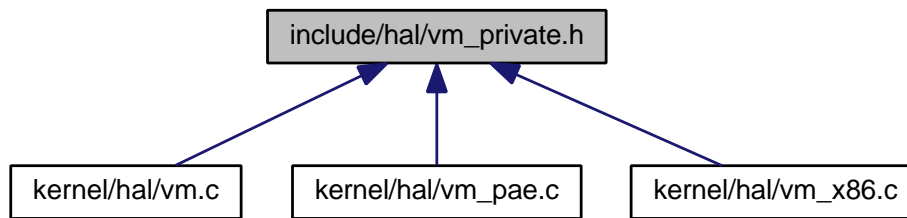
#include <jinue-common/vm.h>
#include <hal/vm.h>
#include <hal/vm_pae.h>
#include <hal/vm_x86.h>
#include <stdbool.h>
#include <stdint.h>
#include <types.h>

```

Include dependency graph for `vm_private.h`:



This graph shows which files directly or indirectly include this file:



## Macros

- **#define PAGE\_TABLE\_ENTRIES (PAGE\_SIZE / sizeof(pte\_t))**  
*This header file contains private definitions shared by **hal/vm.c** (p. 331), **hal/vm\_pae.c** (p. 342) and **hal/vm\_x86.c** (p. 351).*
- **#define PAGE\_TABLE\_MASK (PAGE\_TABLE\_ENTRIES - 1)**  
*bit mask for page table or page directory offset*
- **#define PAGE\_TABLE\_OFFSET\_OF(x) ( ((uint32\_t)(x) / PAGE\_SIZE) & PAGE\_TABLE\_MASK )**  
*page table entry offset of virtual (linear) address*
- **#define PAGE\_DIRECTORY\_OFFSET\_OF(x) ( ((uint32\_t)(x) / (PAGE\_SIZE \* PAGE\_TABLE\_ENTRIES)) & PAGE\_TABLE\_MASK )**  
*page directory entry offset of virtual (linear address)*

## Functions

- **kern\_paddr\_t vm\_clone\_page\_directory(kern\_paddr\_t template\_paddr, unsigned int start\_index)**
- **void vm\_init\_initial\_page\_directory(pte\_t \*page\_directory, boot\_alloc\_t \*boot\_alloc, unsigned int start\_index, unsigned int end\_index, bool first\_directory)**
- **void vm\_destroy\_page\_directory(kern\_paddr\_t pgdir\_paddr, unsigned int from\_index, unsigned int to\_index)**

## Variables

- **pte\_t \* global\_page\_tables**
- **size\_t page\_table\_entries**

## 4.46.1 Macro Definition Documentation

### 4.46.1.1 `#define PAGE_DIRECTORY_OFFSET_OF( x ) ( ((uint32_t)(x) / (PAGE_SIZE * PAGE_TABLE_ENTRIES)) & PAGE_TABLE_MASK )`

page directory entry offset of virtual (linear address)

Definition at line 56 of file `vm_private.h`.

Referenced by `vm_pae_page_directory_offset_of()`, and `vm_x86_page_directory_offset_of()`.

### 4.46.1.2 `#define PAGE_TABLE_ENTRIES (PAGE_SIZE / sizeof(pte_t))`

This header file contains private definitions shared by `hal/vm.c` (p. 331), `hal/vm_pae.c` (p. 342) and `hal/vm_x86.c` (p. 351).

There should be no reason to include it anywhere else. number of entries in page table or page directory

Definition at line 47 of file `vm_private.h`.

Referenced by `vm_pae_boot_init()`, and `vm_x86_boot_init()`.

### 4.46.1.3 `#define PAGE_TABLE_MASK (PAGE_TABLE_ENTRIES - 1)`

bit mask for page table or page directory offset

Definition at line 50 of file `vm_private.h`.

### 4.46.1.4 `#define PAGE_TABLE_OFFSET_OF( x ) ( ((uint32_t)(x) / PAGE_SIZE) & PAGE_TABLE_MASK )`

page table entry offset of virtual (linear) address

Definition at line 53 of file `vm_private.h`.

Referenced by `vm_pae_page_table_offset_of()`, and `vm_x86_page_table_offset_of()`.

## 4.46.2 Function Documentation

### 4.46.2.1 `kern_paddr_t vm_clone_page_directory ( kern_paddr_t template_paddr, unsigned int start_index )`

Definition at line 489 of file `vm.c`.

References `page_alloc()`, `page_table_entries`, `VM_FLAG_READ_WRITE`, `vm_lookup_kernel_paddr()`, `vm_map_kernel()`, `vm_unmap_kernel()`, and `vmalloc()`.

Referenced by `vm_x86_create_addr_space()`.

```

489                                     {
490     unsigned int    idx;
491
492     /* Allocate new page directory.
493      *
494      * TODO handle allocation failure */
495     pte_t *page_directory = (pte_t *)page_alloc();
496
497     /* map page directory template */
498     pte_t *template = (pte_t *)vmalloc();
499     vm_map_kernel((addr_t)template, template_paddr, VM_FLAG_READ_WRITE);
500
501     /* clear all entries below index start_index */

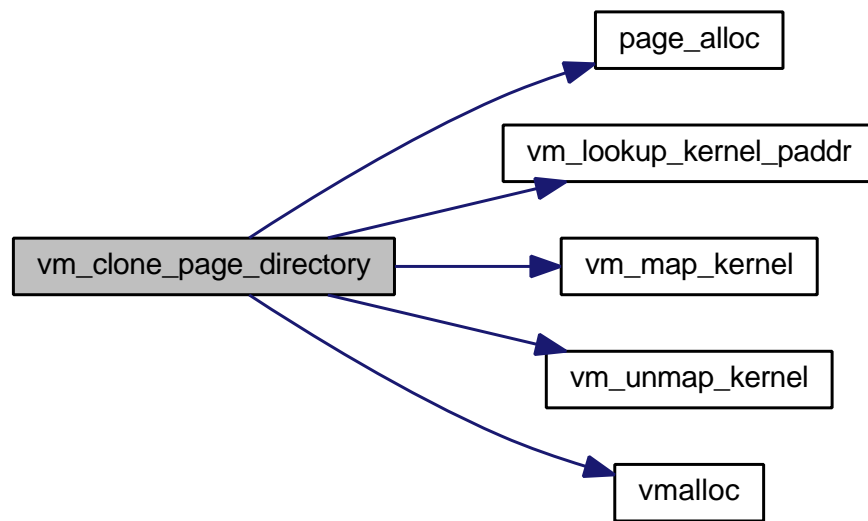
```

```

502     for(idx = 0; idx < start_index; ++idx) {
503         clear_pte( get_pte_with_offset(page_directory, idx) );
504     }
505
506     /* copy entries from template for indexes start_index and above */
507     for(idx = start_index; idx < page_table_entries; ++idx) {
508         copy_pte(
509             get_pte_with_offset(page_directory, idx),
510             get_pte_with_offset(template, idx)
511         );
512     }
513
514     vm_unmap_kernel((addr_t)page_directory);
515     vm_unmap_kernel((addr_t)template);
516
517     return vm_lookup_kernel_paddr((addr_t)page_directory);
518 }

```

Here is the call graph for this function:



#### 4.46.2.2 void vm\_destroy\_page\_directory ( kern\_paddr\_t pgdir\_paddr, unsigned int from\_index, unsigned int to\_index )

Definition at line 584 of file vm.c.

References `pf_free`, `VM_FLAG_READ_WRITE`, `vm_map_kernel()`, `vm_unmap_kernel()`, and `vmalloc()`.

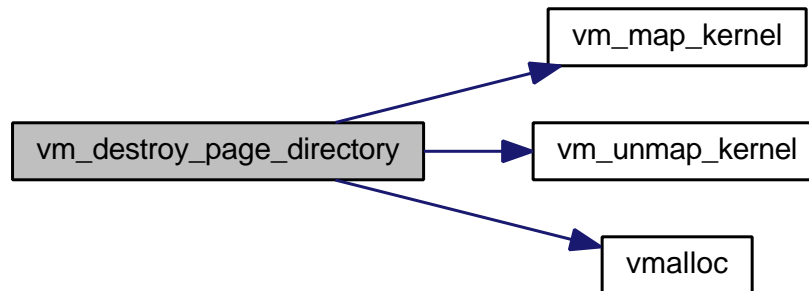
Referenced by `vm_pae_destroy_addr_space()`, and `vm_x86_destroy_addr_space()`.

```

584     {
585         unsigned int idx;
586
587         pte_t *page_directory = (pte_t *)vmalloc();
588         vm_map_kernel((addr_t)page_directory, pgdir_paddr, VM_FLAG_READ_WRITE);
589
590         /* be careful not to free the kernel page tables */
591         for(idx = from_index; idx < to_index; ++idx) {
592             pte_t *pte = get_pte_with_offset(page_directory, idx);
593
594             if(get_pte_flags(pte) & VM_FLAG_PRESENT) {
595                 pf_free( get_pte_paddr(pte) );
596             }
597         }
598
599         vm_unmap_kernel((addr_t)page_directory);
600         pf_free(pgdir_paddr);
601     }

```

Here is the call graph for this function:



**4.46.2.3** void vm\_init\_initial\_page\_directory ( pte\_t \* page\_directory, boot\_alloc\_t \* boot\_alloc, unsigned int start\_index, unsigned int end\_index, bool first\_directory )

Definition at line 529 of file vm.c.

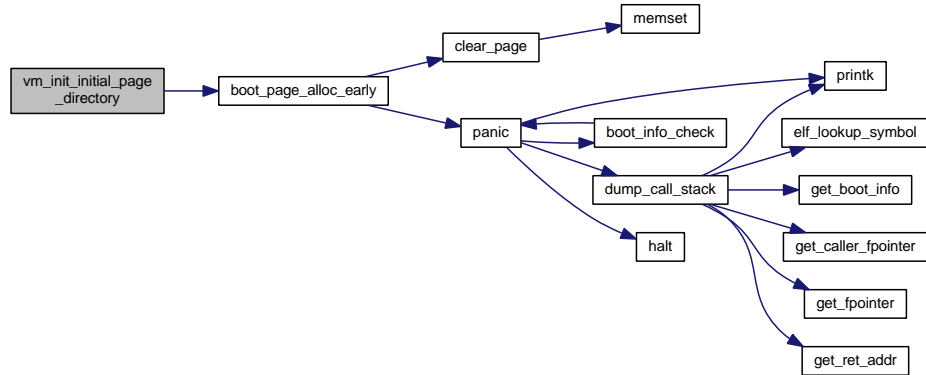
References boot\_page\_alloc\_early(), EARLY\_PTR\_TO\_PHYS\_ADDR, page\_table\_entries, and VM\_FLAG\_READ\_WRITE.

Referenced by vm\_pae\_create\_initial\_addr\_space(), and vm\_x86\_create\_initial\_addr\_space().

```

534                                     {
535
536     unsigned int idx, idy;
537
538     /* Allocate page tables and initialize page directory entries. */
539     for(idx = 0; idx < page_table_entries; ++idx) {
540         if(idx < start_index || idx >= end_index) {
541             /* Clear page directory entries for user space and non-preallocated
542              * kernel page tables. */
543             clear_pte( get_pte_with_offset(page_directory, idx) );
544         }
545         else {
546             /* Allocate page tables for kernel data/code region.
547              *
548              * Note that the use of pfallot_early() here guarantees that the
549              * page tables are allocated contiguously, and that they keep the
550              * same address once paging is enabled. */
551             pte_t *page_table = (pte_t *)boot_page_alloc_early(boot_alloc);
552
553             if(first_directory && idx == start_index) {
554                 /* remember the address of the first page table for use by
555                  * vm_map() later */
556                 global_page_tables = page_table;
557             }
558
559             set_pte(
560                 get_pte_with_offset(page_directory, idx),
561                 EARLY_PTR_TO_PHYS_ADDR(page_table),
562                 VM_FLAG_PRESENT | VM_FLAG_READ_WRITE);
563
564             /* clear page table */
565             for(idy = 0; idy < page_table_entries; ++idy) {
566                 clear_pte( get_pte_with_offset(page_table, idy) );
567             }
568         }
569     }
570 }
  
```

Here is the call graph for this function:



### 4.46.3 Variable Documentation

#### 4.46.3.1 `pte_t* global_page_tables`

Definition at line 49 of file `vm.c`.

#### 4.46.3.2 `size_t page_table_entries`

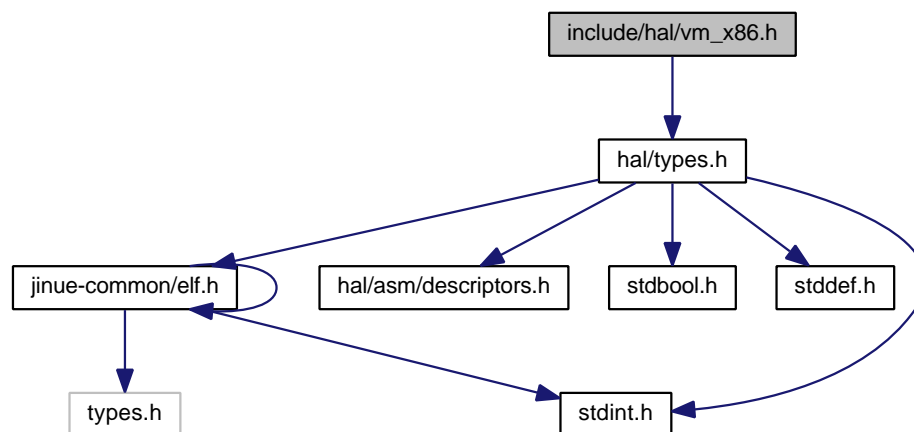
Definition at line 53 of file `vm.c`.

Referenced by `vm_clone_page_directory()`, `vm_init_initial_page_directory()`, `vm_pae_boot_init()`, `vm_pae_create_initial_addr_space()`, and `vm_x86_boot_init()`.

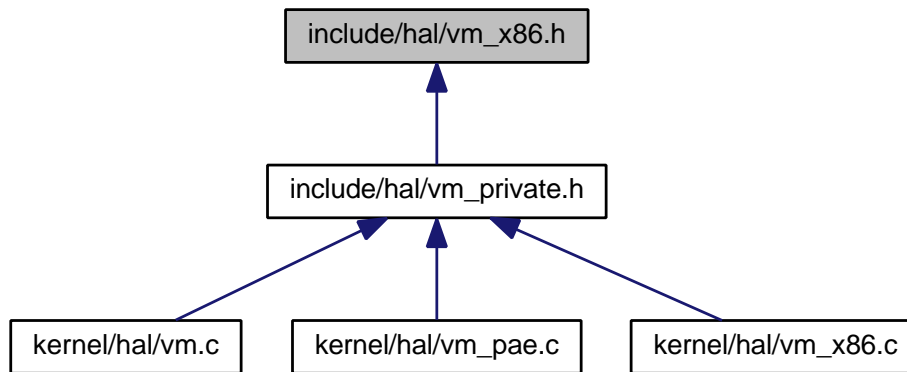
## 4.47 `include/hal/vm_x86.h` File Reference

```
#include <hal/types.h>
```

Include dependency graph for `vm_x86.h`:



This graph shows which files directly or indirectly include this file:



## Functions

- void **vm\_x86\_boot\_init** (void)

*This header file contains declarations for the non-PAE functions defined in **hal/vm\_x86.c** (p. 351).*

- **addr\_space\_t** \* **vm\_x86\_create\_addr\_space** (**addr\_space\_t** \*addr\_space)
- **addr\_space\_t** \* **vm\_x86\_create\_initial\_addr\_space** (**boot\_alloc\_t** \*boot\_alloc)
- void **vm\_x86\_destroy\_addr\_space** (**addr\_space\_t** \*addr\_space)
- unsigned int **vm\_x86\_page\_table\_offset\_of** (**addr\_t** addr)
- unsigned int **vm\_x86\_page\_directory\_offset\_of** (**addr\_t** addr)
- **pte\_t** \* **vm\_x86\_lookup\_page\_directory** (**addr\_space\_t** \*addr\_space)

*Lookup and map the page directory for a specified address and address space.*

- **pte\_t** \* **vm\_x86\_get\_pte\_with\_offset** (**pte\_t** \*pte, unsigned int offset)
- void **vm\_x86\_set\_pte** (**pte\_t** \*pte, **uint32\_t** paddr, int flags)
- void **vm\_x86\_set\_pte\_flags** (**pte\_t** \*pte, int flags)
- int **vm\_x86\_get\_pte\_flags** (const **pte\_t** \*pte)
- **uint32\_t** **vm\_x86\_get\_pte\_paddr** (const **pte\_t** \*pte)
- void **vm\_x86\_clear\_pte** (**pte\_t** \*pte)
- void **vm\_x86\_copy\_pte** (**pte\_t** \*dest, const **pte\_t** \*src)

### 4.47.1 Function Documentation

#### 4.47.1.1 void vm\_x86\_boot\_init ( void )

This header file contains declarations for the non-PAE functions defined in **hal/vm\_x86.c** (p. 351).

It is intended to be included by **hal/vm.c** (p. 331) and **hal/vm\_x86.c** (p. 351). There should be no reason to include it anywhere else.

Definition at line 41 of file **vm\_x86.c**.

References **PAGE\_TABLE\_ENTRIES**, and **page\_table\_entries**.

Referenced by **vm\_boot\_init()**.

```

41      {
42      page_table_entries = (size_t)PAGE_TABLE_ENTRIES;
43  }
```

#### 4.47.1.2 void vm\_x86\_clear\_pte ( pte\_t \* pte )

Definition at line 133 of file vm\_x86.c.

References pte\_t::entry.

```
133                                     {
134     pte->entry = 0;
135 }
```

#### 4.47.1.3 void vm\_x86\_copy\_pte ( pte\_t \* dest, const pte\_t \* src )

Definition at line 137 of file vm\_x86.c.

References pte\_t::entry.

```
137                                     {
138     dest->entry = src->entry;
139 }
```

#### 4.47.1.4 addr\_space\_t \* vm\_x86\_create\_addr\_space ( addr\_space\_t \* addr\_space )

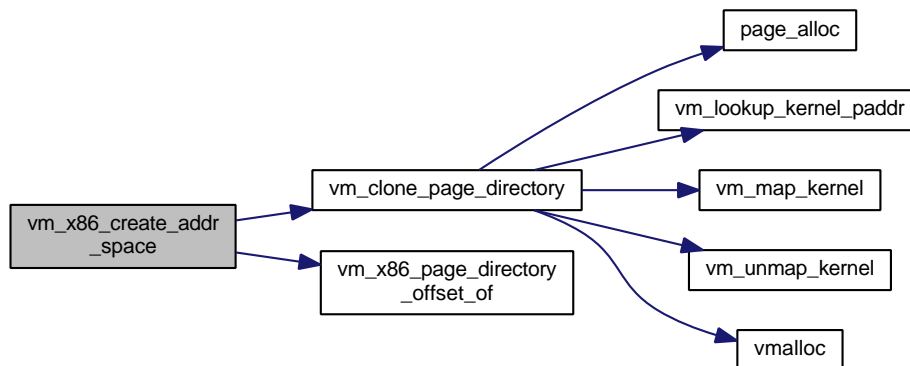
Definition at line 45 of file vm\_x86.c.

References addr\_space\_t::cr3, initial\_addr\_space, KLIMIT, addr\_space\_t::pd, addr\_space\_t::top\_level, vm\_clone\_page\_directory(), and vm\_x86\_page\_directory\_offset\_of().

Referenced by vm\_create\_addr\_space().

```
45                                     {
46     /* Create a new page directory where entries for the address range starting
47      * at KLIMIT are copied from the initial address space. The mappings starting
48      * at KLIMIT belong to the kernel and are identical in all address spaces. */
49     kern_paddr_t paddr = vm_clone_page_directory(
50         initial_addr_space.top_level.pd,
51         vm_x86_page_directory_offset_of((addr_t)KLIMIT));
52
53     addr_space->top_level.pd = paddr;
54     addr_space->cr3          = paddr;
55
56     return addr_space;
57 }
```

Here is the call graph for this function:





4.47.1.5 `addr_space_t* vm_x86_create_initial_addr_space ( boot_alloc_t* boot_alloc )`

Definition at line 59 of file `vm_x86.c`.

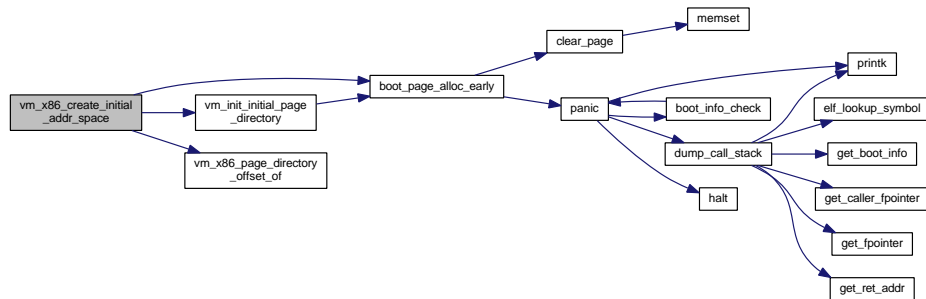
References `boot_page_alloc_early()`, `addr_space_t::cr3`, `EARLY_PTR_TO_PHYS_ADDR`, `EARLY_VIRT_TO_PHYS`, `initial_addr_space`, `KERNEL_PREALLOC_LIMIT`, `KLIMIT`, `addr_space_t::pd`, `addr_space_t::top_level`, `vm_init_initial_page_directory()`, and `vm_x86_page_directory_offset_of()`.

Referenced by `vm_create_initial_addr_space()`.

```

59
60     pte_t *page_directory = (pte_t *)boot_page_alloc_early(boot_alloc);
61
62     vm_init_initial_page_directory(
63         page_directory,
64         boot_alloc,
65         vm_x86_page_directory_offset_of((addr_t) KLIMIT),
66         vm_x86_page_directory_offset_of((addr_t) KERNEL_PREALLOC_LIMIT),
67         true);
68
69     initial_addr_space.top_level.pd = EARLY_PTR_TO_PHYS_ADDR(page_directory);
70     initial_addr_space.cr3          = EARLY_VIRT_TO_PHYS((uintptr_t)page_directory);
71
72     return &initial_addr_space;
73 }
```

Here is the call graph for this function:

4.47.1.6 `void vm_x86_destroy_addr_space ( addr_space_t* addr_space )`

Definition at line 75 of file `vm_x86.c`.

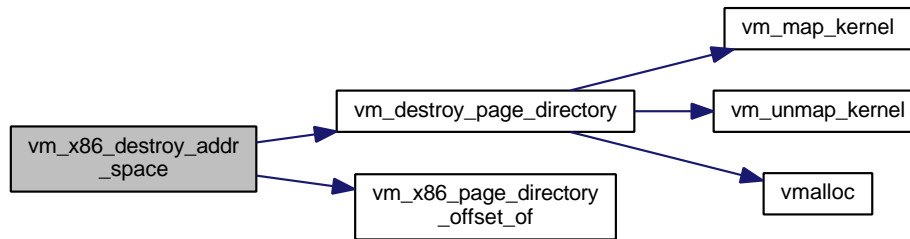
References `KLIMIT`, `addr_space_t::pd`, `addr_space_t::top_level`, `vm_destroy_page_directory()`, and `vm_x86_page_directory_offset_of()`.

Referenced by `vm_destroy_addr_space()`.

```

75
76     vm_destroy_page_directory(
77         addr_space->top_level.pd,
78         /* Free page tables for addresses 0..KLIMIT, be careful not to free
79          * the kernel page tables starting at KLIMIT. */
80         0,
81         vm_x86_page_directory_offset_of((addr_t) KLIMIT));
82 }
```

Here is the call graph for this function:



#### 4.47.1.7 int vm\_x86\_get\_pte\_flags ( const pte\_t \* pte )

Definition at line 125 of file vm\_x86.c.

References pte\_t::entry, and PAGE\_MASK.

```

125                                     {
126     return pte->entry & PAGE_MASK;
127 }
```

#### 4.47.1.8 uint32\_t vm\_x86\_get\_pte\_paddr ( const pte\_t \* pte )

Definition at line 129 of file vm\_x86.c.

References pte\_t::entry, and PAGE\_MASK.

```

129                                     {
130     return pte->entry & ~PAGE_MASK;
131 }
```

#### 4.47.1.9 pte\_t\* vm\_x86\_get\_pte\_with\_offset ( pte\_t \* pte, unsigned int offset )

Definition at line 113 of file vm\_x86.c.

```

113                                     {
114     return &pte[offset];
115 }
```

#### 4.47.1.10 pte\_t\* vm\_x86\_lookup\_page\_directory ( addr\_space\_t \* addr\_space )

Lookup and map the page directory for a specified address and address space.

This is the implementation for standard 32-bit (i.e. non-PAE) paging. This means that there is only one preallocated page directory, so the addr and create\_as\_needed arguments are both irrelevant.

Important note: it is the caller's responsibility to unmap and free the returned page directory when it is done with it.

## Parameters

<i>addr_space</i>	address space in which the address is looked up.
<i>addr</i>	address to look up
<i>create_as_need</i>	Whether a page table is allocated if it does not exist

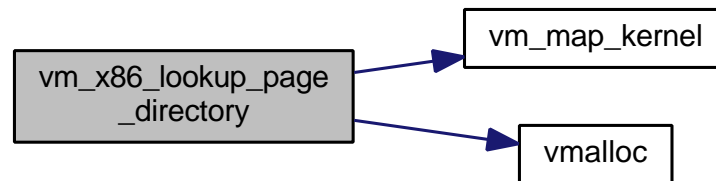
Definition at line 106 of file vm\_x86.c.

References `addr_space_t::pd`, `addr_space_t::top_level`, `VM_FLAG_READ_WRITE`, `vm_map_kernel()`, and `vmalloc()`.

```

106                                     {
107     pte_t *page_directory = (pte_t *)vmalloc();
108     vm_map_kernel((addr_t)page_directory, addr_space->top_level.pd,
109                 VM_FLAG_READ_WRITE);
110     return page_directory;
111 }
```

Here is the call graph for this function:



#### 4.47.1.11 unsigned int vm\_x86\_page\_directory\_offset\_of ( addr\_t addr )

Definition at line 88 of file vm\_x86.c.

References `PAGE_DIRECTORY_OFFSET_OF`.

Referenced by `vm_x86_create_addr_space()`, `vm_x86_create_initial_addr_space()`, and `vm_x86_destroy_addr_space()`.

```

88                                     {
89     return PAGE_DIRECTORY_OFFSET_OF(addr);
90 }
```

#### 4.47.1.12 unsigned int vm\_x86\_page\_table\_offset\_of ( addr\_t addr )

Definition at line 84 of file vm\_x86.c.

References `PAGE_TABLE_OFFSET_OF`.

```

84                                     {
85     return PAGE_TABLE_OFFSET_OF(addr);
86 }
```

#### 4.47.1.13 void vm\_x86\_set\_pte ( pte\_t \*pte, uint32\_t paddr, int flags )

Definition at line 117 of file vm\_x86.c.

References `pte_t::entry`.

```

117                                     {
118     pte->entry = paddr | flags;
119 }

```

#### 4.47.1.14 void vm\_x86\_set\_pte\_flags ( pte\_t \* pte, int flags )

Definition at line 121 of file vm\_x86.c.

References pte\_t::entry, and PAGE\_MASK.

```

121                                     {
122     pte->entry = (pte->entry & ~PAGE_MASK) | flags;
123 }

```

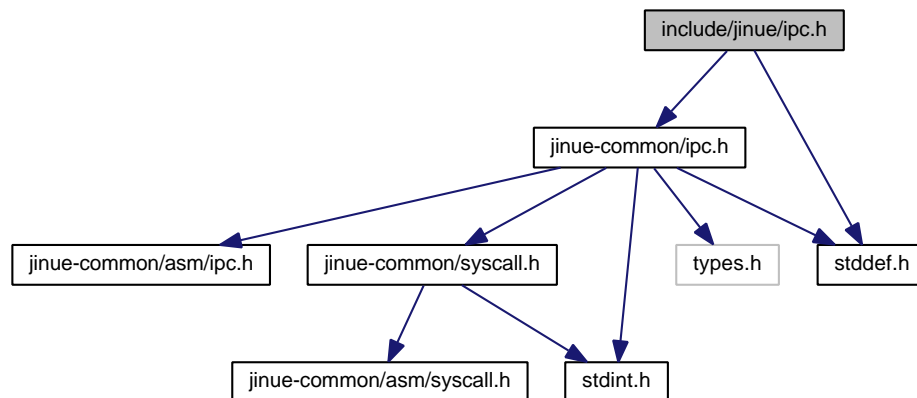
## 4.48 include/ipc.h File Reference

## 4.49 include/jinue/ipc.h File Reference

```
#include <jinue-common/ipc.h>
```

```
#include <stddef.h>
```

Include dependency graph for ipc.h:



## Data Structures

- struct `jinue_message_t`
- struct `jinue_reply_t`

## Functions

- int `jinue_send` (int function, int fd, char \*buffer, **size\_t** buffer\_size, **size\_t** data\_size, unsigned int n\_desc, int \*perrno)
- int `jinue_receive` (int fd, char \*buffer, **size\_t** buffer\_size, **jinue\_message\_t** \*message, int \*perrno)
- int `jinue_reply` (char \*buffer, **size\_t** buffer\_size, **size\_t** data\_size, unsigned int n\_desc, int \*perrno)
- int `jinue_create_ipc` (int flags, int \*perrno)

### 4.49.1 Function Documentation

4.49.1.1 `int jinue_create_ipc ( int flags, int * perrno )`

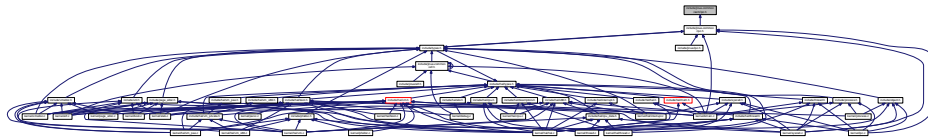
4.49.1.2 `int jinue_receive ( int fd, char * buffer, size_t buffer_size, jinue_message_t * message, int * perrno )`

4.49.1.3 `int jinue_reply ( char * buffer, size_t buffer_size, size_t data_size, unsigned int n_desc, int * perrno )`

4.49.1.4 `int jinue_send ( int function, int fd, char * buffer, size_t buffer_size, size_t data_size, unsigned int n_desc, int * perrno )`

## 4.50 include/jinue-common/asm/ipc.h File Reference

This graph shows which files directly or indirectly include this file:



### Macros

- `#define JINUE_SEND_SIZE_BITS 12`  
*number of bits reserved for the message buffer size and data size fields*
- `#define JINUE_SEND_N_DESC_BITS 8`  
*number of bits reserved for the number of message descriptors*
- `#define JINUE_SEND_MAX_SIZE (1 << (JINUE_SEND_SIZE_BITS - 1))`  
*maximum size of a message buffer and of the data inside that buffer*
- `#define JINUE_SEND_MAX_N_DESC ((1 << JINUE_SEND_N_DESC_BITS) - 1)`  
*maximum number of descriptors inside a message*
- `#define JINUE_SEND_SIZE_MASK ((1 << JINUE_SEND_SIZE_BITS) - 1)`  
*mask to extract the message buffer or data size fields*
- `#define JINUE_SEND_N_DESC_MASK JINUE_SEND_MAX_N_DESC`  
*mask to extract the number of descriptors inside a message*
- `#define JINUE_SEND_BUFFER_SIZE_OFFSET (JINUE_SEND_N_DESC_BITS + JINUE_SEND_SIZE_BITS)`  
*offset of buffer size within arg3*
- `#define JINUE_SEND_DATA_SIZE_OFFSET JINUE_SEND_N_DESC_BITS`  
*offset of data size within arg3*
- `#define JINUE_SEND_N_DESC_OFFSET 0`  
*offset of number of descriptors within arg3*
- `#define JINUE_ARGS_PACK_BUFFER_SIZE(s) ((s) << JINUE_SEND_BUFFER_SIZE_OFFSET)`
- `#define JINUE_ARGS_PACK_DATA_SIZE(s) ((s) << JINUE_SEND_DATA_SIZE_OFFSET)`
- `#define JINUE_ARGS_PACK_N_DESC(n) ((n) << JINUE_SEND_N_DESC_OFFSET)`

### 4.50.1 Macro Definition Documentation

4.50.1.1 `#define JINUE_ARGS_PACK_BUFFER_SIZE( s ) ((s) << JINUE_SEND_BUFFER_SIZE_OFFSET)`

Definition at line 68 of file ipc.h.

**4.50.1.2** `#define JINUE_ARGS_PACK_DATA_SIZE( s ) ((s) << JINUE_SEND_DATA_SIZE_OFFSET)`

Definition at line 70 of file ipc.h.

**4.50.1.3** `#define JINUE_ARGS_PACK_N_DESC( n ) ((n) << JINUE_SEND_N_DESC_OFFSET)`

Definition at line 72 of file ipc.h.

**4.50.1.4** `#define JINUE_SEND_BUFFER_SIZE_OFFSET (JINUE_SEND_N_DESC_BITS + JINUE_SEND_SIZE_BITS)`

offset of buffer size within arg3

Definition at line 59 of file ipc.h.

Referenced by ipc\_reply().

**4.50.1.5** `#define JINUE_SEND_DATA_SIZE_OFFSET JINUE_SEND_N_DESC_BITS`

offset of data size within arg3

Definition at line 62 of file ipc.h.

**4.50.1.6** `#define JINUE_SEND_MAX_N_DESC ((1 << JINUE_SEND_N_DESC_BITS) - 1)`

maximum number of descriptors inside a message

Definition at line 50 of file ipc.h.

Referenced by ipc\_reply(), and ipc\_send().

**4.50.1.7** `#define JINUE_SEND_MAX_SIZE (1 << (JINUE_SEND_SIZE_BITS - 1))`

maximum size of a message buffer and of the data inside that buffer

Definition at line 47 of file ipc.h.

Referenced by ipc\_reply(), and ipc\_send().

**4.50.1.8** `#define JINUE_SEND_N_DESC_BITS 8`

number of bits reserved for the number of message descriptors

Definition at line 44 of file ipc.h.

**4.50.1.9** `#define JINUE_SEND_N_DESC_MASK JINUE_SEND_MAX_N_DESC`

mask to extract the number of descriptors inside a message

Definition at line 56 of file ipc.h.

4.50.1.10 `#define JINUE_SEND_N_DESC_OFFSET 0`

offset of number of descriptors within arg3

Definition at line 65 of file ipc.h.

4.50.1.11 `#define JINUE_SEND_SIZE_BITS 12`

number of bits reserved for the message buffer size and data size fields

Definition at line 41 of file ipc.h.

4.50.1.12 `#define JINUE_SEND_SIZE_MASK ((1 << JINUE_SEND_SIZE_BITS) - 1)`

mask to extract the message buffer or data size fields

Definition at line 53 of file ipc.h.

Referenced by ipc\_reply().

## 4.51 include/jinue-common/ipc.h File Reference

```
#include <jinue-common/asm/ipc.h>
```

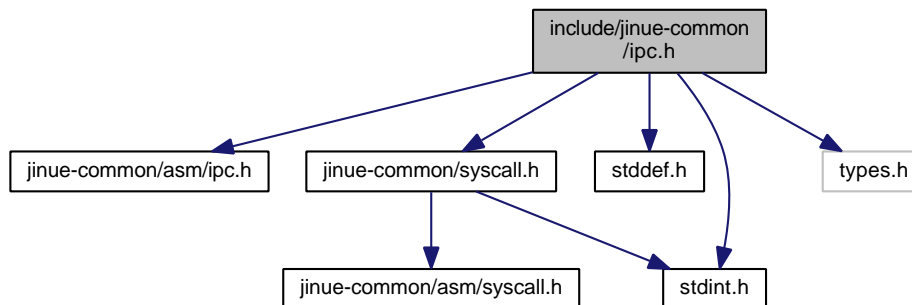
```
#include <jinue-common/syscall.h>
```

```
#include <stddef.h>
```

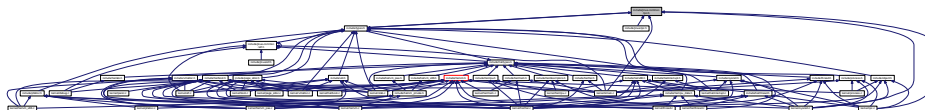
```
#include <stdint.h>
```

```
#include <types.h>
```

Include dependency graph for ipc.h:



This graph shows which files directly or indirectly include this file:



## Macros

- `#define IPC_FLAG_NONE 0`

- `#define IPC_FLAG_SYSTEM (1<<8)`
- `#define JINUE_IPC_NONE 0`
- `#define JINUE_IPC_SYSTEM (1<<0)`
- `#define JINUE_IPC_PROC (1<<1)`

## Typedefs

- `typedef int jinue_ipc_descriptor_t`

## Functions

- `void ipc_boot_init (boot_alloc_t *boot_alloc)`
- `ipc_t * ipc_object_create (int flags)`
- `ipc_t * ipc_get_proc_object (void)`
- `void ipc_send (jinue_syscall_args_t *args)`
- `void ipc_receive (jinue_syscall_args_t *args)`
- `void ipc_reply (jinue_syscall_args_t *args)`

### 4.51.1 Macro Definition Documentation

#### 4.51.1.1 `#define IPC_FLAG_NONE 0`

Definition at line 41 of file ipc.h.

Referenced by `dispatch_syscall()`.

#### 4.51.1.2 `#define IPC_FLAG_SYSTEM (1<<8)`

Definition at line 43 of file ipc.h.

Referenced by `dispatch_syscall()`.

#### 4.51.1.3 `#define JINUE_IPC_NONE 0`

Definition at line 41 of file ipc.h.

#### 4.51.1.4 `#define JINUE_IPC_PROC (1<<1)`

Definition at line 45 of file ipc.h.

Referenced by `dispatch_syscall()`.

#### 4.51.1.5 `#define JINUE_IPC_SYSTEM (1<<0)`

Definition at line 43 of file ipc.h.

Referenced by `dispatch_syscall()`.



## 4.51.2 Typedef Documentation

### 4.51.2.1 typedef int jinue\_ipc\_descriptor\_t

Definition at line 48 of file ipc.h.

## 4.51.3 Function Documentation

### 4.51.3.1 void ipc\_boot\_init ( boot\_alloc\_t \* boot\_alloc )

Definition at line 58 of file ipc.c.

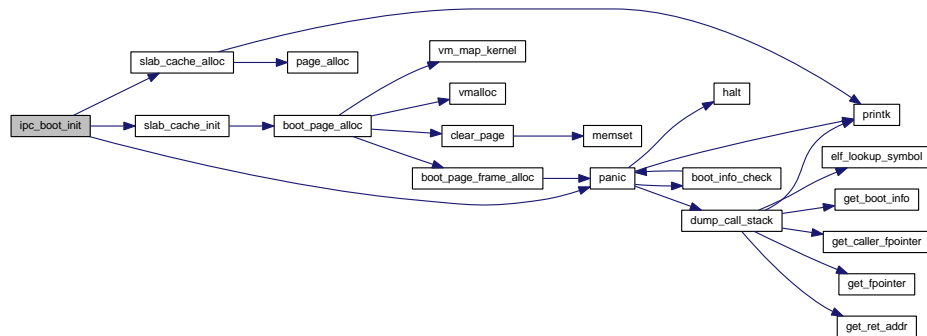
References `NULL`, `panic()`, `slab_cache_alloc()`, `slab_cache_init()`, and `SLAB_DEFAULTS`.

Referenced by `kmmain()`.

```

58                                     {
59     slab_cache_init (
60         &ipc_object_cache,
61         "ipc_object_cache",
62         sizeof(ipc_t),
63         0,
64         ipc_object_ctor,
65         NULL,
66         SLAB_DEFAULTS,
67         boot_alloc);
68
69     proc_ipc = slab_cache_alloc(&ipc_object_cache);
70
71     if(proc_ipc == NULL) {
72         panic("Cannot create process manager IPC object.");
73     }
74 }
```

Here is the call graph for this function:



### 4.51.3.2 ipc\_t\* ipc\_get\_proc\_object ( void )

Definition at line 86 of file ipc.c.

Referenced by `dispatch_syscall()`.

```

86                                     {
87     return proc_ipc;
88 }
```

#### 4.51.3.3 ipc\_t\* ipc\_object\_create ( int flags )

Definition at line 76 of file ipc.c.

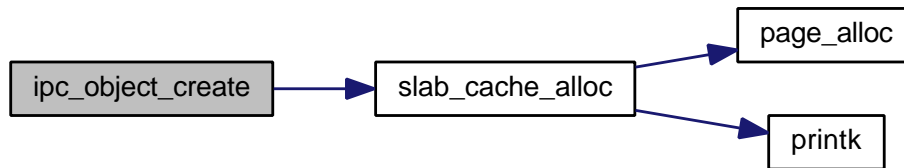
References object\_header\_t::flags, ipc\_t::header, NULL, and slab\_cache\_alloc().

Referenced by dispatch\_syscall().

```

76      {
77      ipc_t *ipc = slab_cache_alloc(&ipc_object_cache);
78
79      if(ipc != NULL) {
80          ipc->header.flags = flags;
81      }
82
83      return ipc;
84  }
```

Here is the call graph for this function:



#### 4.51.3.4 void ipc\_receive ( jinue\_syscall\_args\_t\* args )

Definition at line 205 of file ipc.c.

References jinue\_syscall\_args\_t::arg0, jinue\_syscall\_args\_t::arg1, jinue\_syscall\_args\_t::arg2, jinue\_syscall\_args\_t::arg3, message\_info\_t::data\_size, object\_header\_t::flags, thread\_t::header, JINUE\_E2BIG, JINUE\_EBADF, JINUE\_EINVAL, JINUE\_EIO, JINUE\_EPERM, jinue\_node\_entry, memcpy(), thread\_t::message\_args, thread\_t::message\_buffer, thread\_t::message\_info, NULL, OBJECT\_REF\_FLAG\_CLOSED, OBJECT\_TYPE\_IPC, thread\_t::process, process\_get\_descriptor(), ipc\_t::recv\_list, ipc\_t::send\_list, thread\_t::sender, thread\_t::thread\_list, thread\_switch(), thread\_yield\_from(), message\_info\_t::total\_size, and object\_header\_t::type.

Referenced by dispatch\_syscall().

```

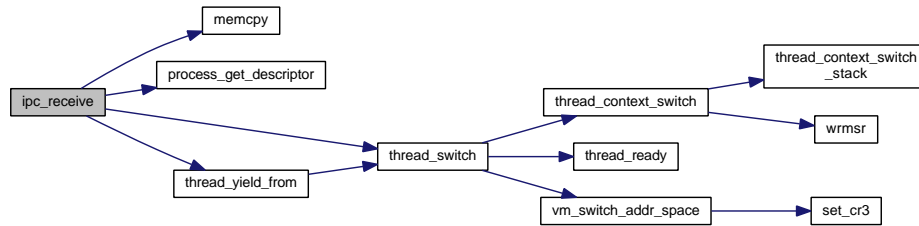
205      {
206      thread_t *thread = get_current_thread();
207
208      int fd = (int)args->arg1;
209
210      object_ref_t *ref = process_get_descriptor(thread->process, fd);
211
212      if(! object_ref_is_valid(ref)) {
213          syscall_args_set_error(args, JINUE_EBADF);
214          return;
215      }
216
217      if(object_ref_is_closed(ref)) {
218          syscall_args_set_error(args, JINUE_EIO);
219          return;
220      }
221
222      if(! object_ref_is_owner(ref)) {
223          syscall_args_set_error(args, JINUE_EPERM);
224          return;
225      }
226
227      object_header_t *header = ref->object;
228  }
```

```

229     if(object_is_destroyed(header)) {
230         ref->flags |= OBJECT_REF_FLAG_CLOSED;
231         object_subref(header);
232     }
233     syscall_args_set_error(args, JINUE_EIO);
234     return;
235 }
236
237 if(header->type != OBJECT_TYPE_IPC) {
238     syscall_args_set_error(args, JINUE_EBADF);
239     return;
240 }
241
242 ipc_t *ipc = (ipc_t *)header;
243
244 char *user_ptr = (char *)args->arg2;
245 size_t buffer_size = jinue_args_get_buffer_size(args);
246
247 if(! user_buffer_check(user_ptr, buffer_size)) {
248     syscall_args_set_error(args, JINUE_EINVAL);
249     return;
250 }
251
252 thread_t *send_thread = jinue_node_entry(
253     jinue_list_dequeue(&ipc->send_list),
254     thread_t,
255     thread_list);
256
257 if(send_thread == NULL) {
258     /* No thread is waiting to send a message, so we must wait on the receive
259      * list. */
260     jinue_list_enqueue(&ipc->recv_list, &thread->thread_list);
261
262     thread_yield_from(
263         thread,
264         true,          /* make thread block */
265         false);       /* don't destroy */
266
267     /* set by sending thread */
268     send_thread = thread->sender;
269 }
270 else {
271     object_addrref(&send_thread->header);
272     thread->sender = send_thread;
273 }
274
275 if(send_thread->message_info.total_size > buffer_size) {
276     /* message is too big for receive buffer */
277     object_subref(&send_thread->header);
278     thread->sender = NULL;
279
280     syscall_args_set_error(send_thread->message_args, JINUE_E2BIG);
281     syscall_args_set_error(args, JINUE_E2BIG);
282
283     /* switch back to sender thread to return from call immediately */
284     thread_switch(
285         thread,
286         send_thread,
287         false,      /* don't block (put this thread back in ready queue) */
288         false);     /* don't destroy */
289
290     return;
291 }
292
293 memcpy(
294     user_ptr,
295     send_thread->message_buffer,
296     send_thread->message_info.data_size);
297
298 args->arg0 = send_thread->message_args->arg0;
299 args->arg1 = ref->cookie;
300 /* argument 2 is left intact (buffer pointer) */
301 args->arg3 = send_thread->message_args->arg3;
302 }

```

Here is the call graph for this function:



#### 4.51.3.5 void ipc\_reply ( jinue\_syscall\_args\_t\* args )

TODO is there a better error number for this situation?

TODO remove this check when descriptor passing is implemented

TODO copy descriptors

TODO set return value and error number

Definition at line 304 of file ipc.c.

References jinue\_syscall\_args\_t::arg2, jinue\_syscall\_args\_t::arg3, message\_info\_t::buffer\_size, message\_info\_t::data\_size, message\_info\_t::desc\_n, thread\_t::header, JINUE\_EINVAL, JINUE\_ENOSYS, JINUE\_SEND\_BUFFER\_SIZE\_OFFSET, JINUE\_SEND\_MAX\_N\_DESC, JINUE\_SEND\_MAX\_SIZE, JINUE\_SEND\_SIZE\_MASK, memcpy(), thread\_t::message\_args, thread\_t::message\_buffer, thread\_t::message\_info, NULL, thread\_t::sender, and thread\_switch().

Referenced by dispatch\_syscall().

```

304                                     {
305     thread_t *thread                = get_current_thread();
306     thread_t *send_thread           = thread->sender;
307
308     if(send_thread == NULL) {
309         syscall_args_set_error(args, JINUE_EINVAL);
310         return;
311     }
312
313
314     size_t buffer_size              = jinue_args_get_buffer_size(args);
315     size_t data_size                = jinue_args_get_data_size(args);
316     size_t desc_n                   = jinue_args_get_n_desc(args);
317     size_t total_size               =
318         data_size +
319         desc_n * sizeof(jinue_ipc_descriptor_t);
320
321     if(buffer_size > JINUE_SEND_MAX_SIZE) {
322         syscall_args_set_error(args, JINUE_EINVAL);
323         return;
324     }
325
326     if(total_size > buffer_size) {
327         syscall_args_set_error(args, JINUE_EINVAL);
328         return;
329     }
330
331     if(desc_n > JINUE_SEND_MAX_N_DESC) {
332         syscall_args_set_error(args, JINUE_EINVAL);
333         return;
334     }
335
336     /* the reply must fit in the sender's buffer */
337     if(total_size > send_thread->message_info.buffer_size) {
338         syscall_args_set_error(args, JINUE_EINVAL);
339         return;
340     }
341
342     if(desc_n > 0) {

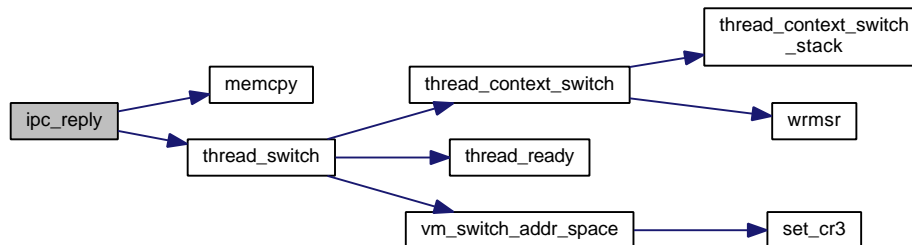
```

```

344     syscall_args_set_error(args, JINUE_ENOSYS);
345     return;
346 }
347
348 const char *user_ptr = (const char *)args->arg2;
349
350 if(! user_buffer_check(user_ptr, buffer_size)) {
351     syscall_args_set_error(args, JINUE_EINVAL);
352     return;
353 }
354
355 memcpy(&send_thread->message_buffer, user_ptr, data_size);
356
357 syscall_args_set_return(send_thread->message_args, 0);
358 send_thread->message_args->arg3 =
359     args->arg3 & ~(JINUE_SEND_SIZE_MASK << JINUE_SEND_BUFFER_SIZE_OFFSET);
360
361 send_thread->message_info.data_size = data_size;
362 send_thread->message_info.desc_n    = desc_n;
363
364 object_subref(&send_thread->header);
365 thread->sender = NULL;
366
367 syscall_args_set_return(args, 0);
368
369 /* switch back to sender thread to return from call immediately */
370 thread_switch(
371     thread,
372     send_thread,
373     false, /* don't block (put this thread back in ready queue) */
374     false); /* don't destroy */
375 }

```

Here is the call graph for this function:



#### 4.51.3.6 void ipc\_send ( jinue\_syscall\_args\_t\* args )

TODO remove this check when descriptor passing is implemented

TODO copy descriptors

TODO copy descriptors

Definition at line 90 of file ipc.c.

References jinue\_syscall\_args\_t::arg0, jinue\_syscall\_args\_t::arg1, jinue\_syscall\_args\_t::arg2, message\_info\_t::buffer\_size, message\_info\_t::cookie, message\_info\_t::data\_size, message\_info\_t::desc\_n, object\_header\_t::flags, message\_info\_t::function, thread\_t::header, JINUE\_EBADF, JINUE\_EINVAL, JINUE\_EIO, JINUE\_ENOSYS, jinue\_node\_entry, JINUE\_SEND\_MAX\_N\_DESC, JINUE\_SEND\_MAX\_SIZE, memcpy(), thread\_t::message\_args, thread\_t::message\_buffer, thread\_t::message\_info, NULL, OBJECT\_REF\_FLAG\_CLOSED, OBJECT\_TYPE\_IPC, thread\_t::process, process\_get\_descriptor(), ipc\_t::recv\_list, ipc\_t::send\_list, thread\_t::sender, thread\_t::thread\_list, thread\_switch(), thread\_yield\_from(), message\_info\_t::total\_size, and object\_header\_t::type.

Referenced by dispatch\_syscall().

```

91     thread_t *thread = get_current_thread();
92
93     message_info_t *message_info = &thread->message_info;
94
95     message_info->function      = args->arg0;
96     message_info->buffer_size   = jinue_args_get_buffer_size(args);
97     message_info->data_size     = jinue_args_get_data_size(args);
98     message_info->desc_n        = jinue_args_get_n_desc(args);
99     message_info->total_size    =
100         message_info->data_size +
101         message_info->desc_n * sizeof(jinue_ipc_descriptor_t);
102
103     if(message_info->buffer_size > JINUE_SEND_MAX_SIZE) {
104         syscall_args_set_error(args, JINUE_EINVAL);
105         return;
106     }
107
108     if(message_info->total_size > message_info->buffer_size) {
109         syscall_args_set_error(args, JINUE_EINVAL);
110         return;
111     }
112
113     if(message_info->desc_n > JINUE_SEND_MAX_N_DESC) {
114         syscall_args_set_error(args, JINUE_EINVAL);
115         return;
116     }
117
118     if(message_info->desc_n > 0) {
119         syscall_args_set_error(args, JINUE_ENOSYS);
120         return;
121     }
122
123     int fd = (int)args->arg1;
124
125     object_ref_t *ref = process_get_descriptor(thread->process, fd);
126
127     if(! object_ref_is_valid(ref)) {
128         syscall_args_set_error(args, JINUE_EBADF);
129         return;
130     }
131
132     if(object_ref_is_closed(ref)) {
133         syscall_args_set_error(args, JINUE_EIO);
134         return;
135     }
136
137     message_info->cookie = ref->cookie;
138
139     object_header_t *header = ref->object;
140
141     if(object_is_destroyed(header)) {
142         ref->flags |= OBJECT_REF_FLAG_CLOSED;
143         object_subref(header);
144
145         syscall_args_set_error(args, JINUE_EIO);
146         return;
147     }
148
149     if(header->type != OBJECT_TYPE_IPC) {
150         syscall_args_set_error(args, JINUE_EBADF);
151         return;
152     }
153
154     ipc_t *ipc = (ipc_t *)header;
155
156     char *user_ptr = (char *)args->arg2;
157
158     if(! user_buffer_check(user_ptr, message_info->buffer_size)) {
159         syscall_args_set_error(args, JINUE_EINVAL);
160         return;
161     }
162
163     memcpy(&thread->message_buffer, user_ptr, message_info->data_size);
164
165     /* return values are set by ipc_reply() (or by ipc_receive() if the call
166      * fails because the message is too big for the receiver's buffer) */
167     thread->message_args = args;
168
169     thread_t *recv_thread = jinue_node_entry(
170         jinue_list_dequeue(&ipc->recv_list),
171         thread_t,

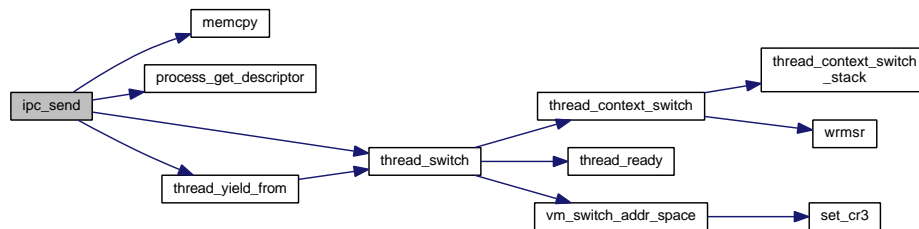
```

```

175         thread_list);
176
177     if(recv_thread == NULL) {
178         /* No thread is waiting to receive this message, so we must wait on the
179          * sender list. */
180         jinue_list_enqueue(&ipc->send_list, &thread->thread_list);
181
182         thread_yield_from(
183             thread,
184             true,          /* make thread block */
185             false);       /* don't destroy */
186     }
187     else {
188         object_addrref(&thread->header);
189         recv_thread->sender = thread;
190
191         /* switch to receiver thread, which will resume inside syscall_receive() */
192         thread_switch(
193             thread,
194             recv_thread,
195             true,          /* block sender thread */
196             false);       /* don't destroy sender */
197     }
198
199     /* copy reply to user space buffer */
200     memcpy(user_ptr, &thread->message_buffer, message_info->data_size);
201
202 }
203

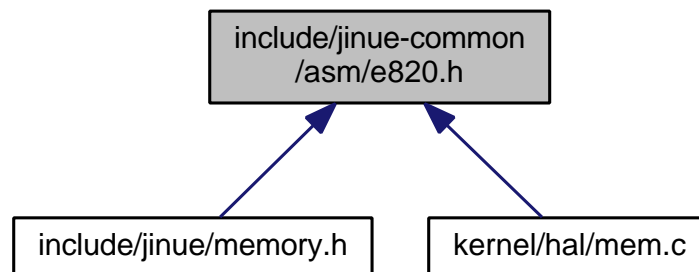
```

Here is the call graph for this function:



## 4.52 include/jinue-common/asm/e820.h File Reference

This graph shows which files directly or indirectly include this file:



### Macros

- `#define E820_RAM 1`
- `#define E820_RESERVED 2`
- `#define E820_ACPI 3`

- `#define E820_SMAP 0x534d4150`

## 4.52.1 Macro Definition Documentation

### 4.52.1.1 `#define E820_ACPI 3`

Definition at line 39 of file e820.h.

### 4.52.1.2 `#define E820_RAM 1`

Definition at line 35 of file e820.h.

Referenced by `mem_check_memory()`.

### 4.52.1.3 `#define E820_RESERVED 2`

Definition at line 37 of file e820.h.

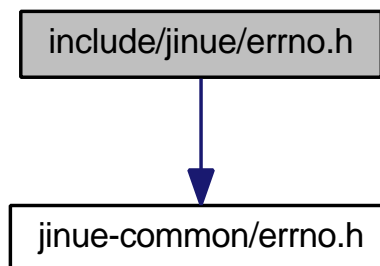
### 4.52.1.4 `#define E820_SMAP 0x534d4150`

Definition at line 41 of file e820.h.

## 4.53 `include/jinue/errno.h` File Reference

```
#include <jinue-common/errno.h>
```

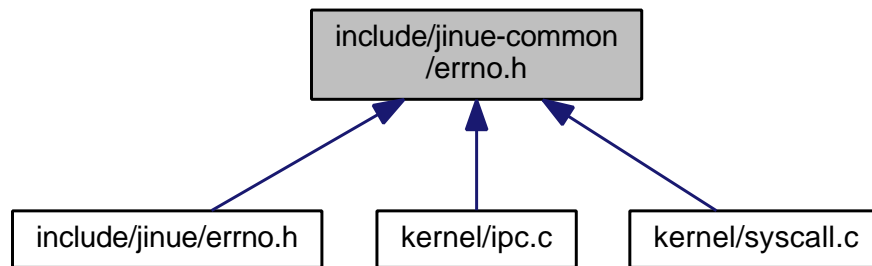
Include dependency graph for `errno.h`:





## 4.54 include/jinue-common/errno.h File Reference

This graph shows which files directly or indirectly include this file:



### Macros

- `#define JINUE_EMORE 1`
- `#define JINUE_ENOMEM 2`
- `#define JINUE_ENOSYS 3`
- `#define JINUE_EINVAL 4`
- `#define JINUE_EAGAIN 5`
- `#define JINUE_EBADF 6`
- `#define JINUE_EIO 7`
- `#define JINUE_EPERM 8`
- `#define JINUE_E2BIG 9`

### 4.54.1 Macro Definition Documentation

#### 4.54.1.1 `#define JINUE_E2BIG 9`

Definition at line 51 of file `errno.h`.

Referenced by `ipc_receive()`.

#### 4.54.1.2 `#define JINUE_EAGAIN 5`

Definition at line 43 of file `errno.h`.

Referenced by `dispatch_syscall()`.

#### 4.54.1.3 `#define JINUE_EBADF 6`

Definition at line 45 of file `errno.h`.

Referenced by `ipc_receive()`, and `ipc_send()`.

#### 4.54.1.4 `#define JINUE_EINVAL 4`

Definition at line 41 of file `errno.h`.

Referenced by `dispatch_syscall()`, `ipc_receive()`, `ipc_reply()`, and `ipc_send()`.

#### 4.54.1.5 #define JINUE\_EIO 7

Definition at line 47 of file errno.h.

Referenced by ipc\_receive(), and ipc\_send().

#### 4.54.1.6 #define JINUE\_EMORE 1

Definition at line 35 of file errno.h.

#### 4.54.1.7 #define JINUE\_ENOMEM 2

Definition at line 37 of file errno.h.

#### 4.54.1.8 #define JINUE\_ENOSYS 3

Definition at line 39 of file errno.h.

Referenced by dispatch\_syscall(), ipc\_reply(), and ipc\_send().

#### 4.54.1.9 #define JINUE\_EPERM 8

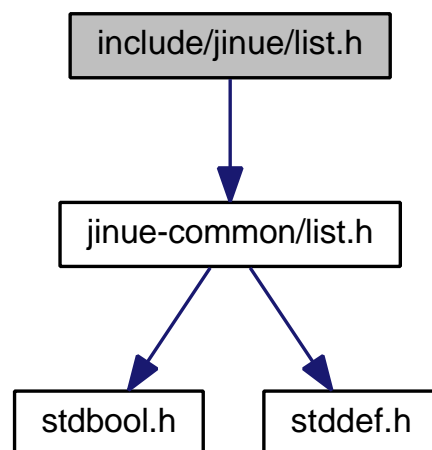
Definition at line 49 of file errno.h.

Referenced by ipc\_receive().

## 4.55 include/jinue/list.h File Reference

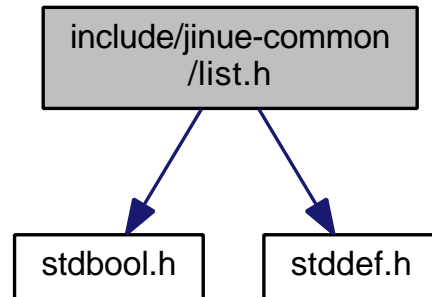
```
#include <jinue-common/list.h>
```

Include dependency graph for list.h:

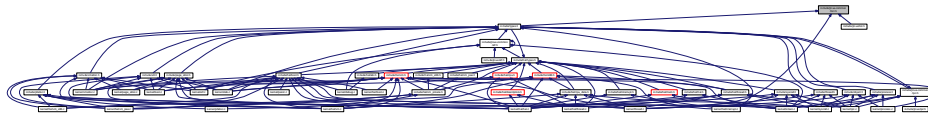


## 4.56 include/jinue-common/list.h File Reference

```
#include <stdbool.h>
#include <stddef.h>
Include dependency graph for list.h:
```



This graph shows which files directly or indirectly include this file:



### Data Structures

- struct **jinue\_node\_t**
- struct **jinue\_list\_t**

### Macros

- #define **JINUE\_LIST\_STATIC** {.head = **NULL**, .tail = **NULL**}
- #define **jinue\_list\_pop**(l) ( jinue\_list\_dequeue((l)) )
- #define **JINUE\_OFFSETOF**(type, member) ((**size\_t**)(&((type \*)0)->member))  
*TODO move this to a more general-purpose header file.*
- #define **jinue\_node\_entry**(node, type, member) (jinue\_node\_entry\_by\_offset(node, **JINUE\_OFFSETOF**(type, member)))
- #define **jinue\_cursor\_entry**(cur, type, member) (jinue\_cursor\_entry\_by\_offset(cur, **JINUE\_OFFSETOF**(type, member)))

### Typedefs

- typedef struct **jinue\_node\_t** **jinue\_node\_t**
- typedef **jinue\_node\_t** \*\* **jinue\_cursor\_t**

### 4.56.1 Macro Definition Documentation

4.56.1.1 `#define jinue_cursor_entry( cur, type, member ) (jinue_cursor_entry_by_offset(cur, JINUE_OFFSETOF(type, member)))`

Definition at line 158 of file list.h.

4.56.1.2 `#define jinue_list_pop( l ) ( jinue_list_dequeue((l)) )`

Definition at line 121 of file list.h.

4.56.1.3 `#define JINUE_LIST_STATIC { .head = NULL, .tail = NULL }`

Definition at line 62 of file list.h.

4.56.1.4 `#define jinue_node_entry( node, type, member ) (jinue_node_entry_by_offset(node, JINUE_OFFSETOF(type, member)))`

Definition at line 144 of file list.h.

Referenced by `ipc_receive()`, and `ipc_send()`.

4.56.1.5 `#define JINUE_OFFSETOF( type, member ) ((size_t)((type *)0)->member)`

TODO move this to a more general-purpose header file.

Definition at line 142 of file list.h.

### 4.56.2 Typedef Documentation

4.56.2.1 `typedef jinue_node_t** jinue_cursor_t`

Definition at line 60 of file list.h.

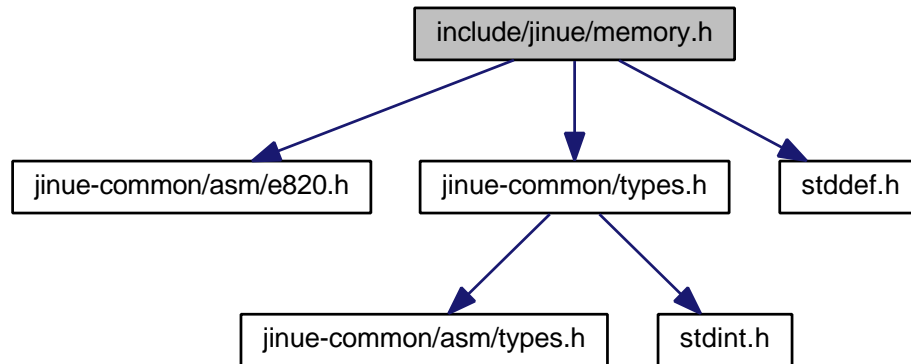
4.56.2.2 `typedef struct jinue_node_t jinue_node_t`

Definition at line 42 of file list.h.

## 4.57 include/jinue/memory.h File Reference

```
#include <jinue-common/asm/e820.h>
#include <jinue-common/types.h>
#include <stddef.h>
```

Include dependency graph for memory.h:



## Functions

- `const char * jinue_pys_mem_type_description (uint32_t type)`
- `int jinue_get_phys_memory (jinue_mem_map_t *buffer, size_t buffer_size, int *perrno)`

### 4.57.1 Function Documentation

4.57.1.1 `int jinue_get_phys_memory ( jinue_mem_map_t * buffer, size_t buffer_size, int * perrno )`

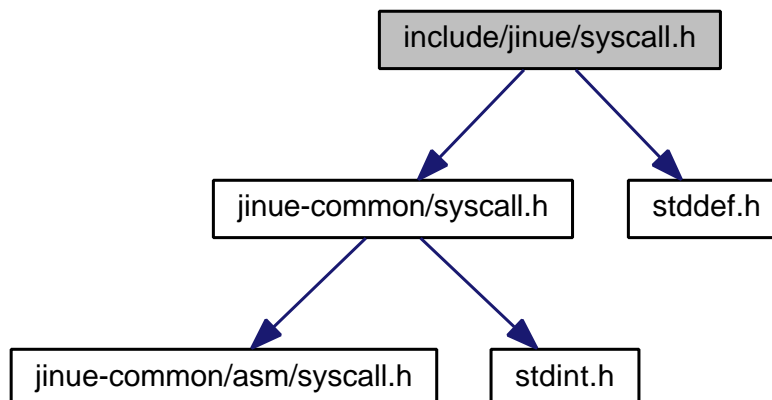
4.57.1.2 `const char* jinue_pys_mem_type_description ( uint32_t type )`

## 4.58 include/jinue/syscall.h File Reference

```
#include <jinue-common/syscall.h>
```

```
#include <stddef.h>
```

Include dependency graph for syscall.h:



## Functions

- `void jinue_call_raw (jinue_syscall_args_t *args)`

- `int jinue_call (jinue_syscall_args_t *args, int *perrno)`
- `void jinue_get_syscall_implementation (void)`
- `const char * jinue_get_syscall_implementation_name (void)`
- `void jinue_set_thread_local_storage (void *addr, size_t size)`
- `void * jinue_get_thread_local_storage (void)`
- `int jinue_thread_create (void(*entry)(), void *stack, int *perrno)`
- `int jinue_yield (void)`
- `void jinue_thread_exit (void)`

### 4.58.1 Function Documentation

4.58.1.1 `int jinue_call ( jinue_syscall_args_t * args, int * perrno )`

4.58.1.2 `void jinue_call_raw ( jinue_syscall_args_t * args )`

4.58.1.3 `void jinue_get_syscall_implementation ( void )`

4.58.1.4 `const char* jinue_get_syscall_implementation_name ( void )`

4.58.1.5 `void* jinue_get_thread_local_storage ( void )`

4.58.1.6 `void jinue_set_thread_local_storage ( void * addr, size_t size )`

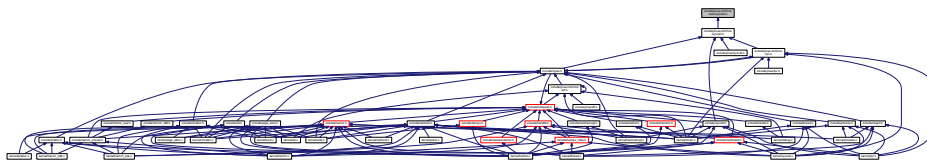
4.58.1.7 `int jinue_thread_create ( void(*)() entry, void * stack, int * perrno )`

4.58.1.8 `void jinue_thread_exit ( void )`

4.58.1.9 `int jinue_yield ( void )`

## 4.59 include/jinue-common/asm/syscall.h File Reference

This graph shows which files directly or indirectly include this file:



### Macros

- `#define SYSCALL_IRQ 0x80`  
*interrupt vector for system call software interrupt*
- `#define SYSCALL_FUNCT_SYSCALL_METHOD 1`  
*get best system call implementation number based on CPU features*
- `#define SYSCALL_FUNCT_CONSOLE_PUTC 2`  
*send a character to in-kernel console driver*
- `#define SYSCALL_FUNCT_CONSOLE_PUTS 3`

- send a fixed-length string to in-kernel console driver*
- **#define SYSCALL\_FUNCT\_THREAD\_CREATE 4**  
*create a new thread*
- **#define SYSCALL\_FUNCT\_THREAD\_YIELD 5**  
*relinquish the CPU and allow the next thread to run*
- **#define SYSCALL\_FUNCT\_SET\_THREAD\_LOCAL\_ADDR 6**  
*set address and size of thread local storage for current thread*
- **#define SYSCALL\_FUNCT\_GET\_THREAD\_LOCAL\_ADDR 7**  
*get address of thread local storage for current thread*
- **#define SYSCALL\_FUNCT\_GET\_PHYS\_MEMORY 8**  
*get free memory block list for management by process manager*
- **#define SYSCALL\_FUNCT\_CREATE\_IPC 9**  
*create an IPC object to receive messages*
- **#define SYSCALL\_FUNCT\_RECEIVE 10**  
*receive a message on an IPC object*
- **#define SYSCALL\_FUNCT\_REPLY 11**  
*reply to current message*
- **#define SYSCALL\_FUNCT\_PROC\_BASE 0x400**  
*start of function numbers for process manager system calls*
- **#define SYSCALL\_FUNCT\_SYSTEM\_BASE 0x1000**  
*start of function numbers for system IPC objects*
- **#define SYSCALL\_FUNCT\_USER\_BASE 0x4000**  
*start of function numbers for user IPC objects*
- **#define SYSCALL\_METHOD\_FAST\_INTEL 0**  
*Intel's fast system call method (SYSENTER/SYSEXIT)*
- **#define SYSCALL\_METHOD\_FAST\_AMD 1**  
*AMD's fast system call method (SYSCALL/SYSLEAVE)*
- **#define SYSCALL\_METHOD\_INTR 2**  
*slow/safe system call method using interrupts*

## 4.59.1 Macro Definition Documentation

### 4.59.1.1 #define SYSCALL\_FUNCT\_CONSOLE\_PUTC 2

send a character to in-kernel console driver

Definition at line 42 of file syscall.h.

Referenced by dispatch\_syscall().

### 4.59.1.2 #define SYSCALL\_FUNCT\_CONSOLE\_PUTS 3

send a fixed-length string to in-kernel console driver

Definition at line 45 of file syscall.h.

Referenced by dispatch\_syscall().

**4.59.1.3 #define SYSCALL\_FUNCT\_CREATE\_IPC 9**

create an IPC object to receive messages

Definition at line 63 of file syscall.h.

Referenced by dispatch\_syscall().

**4.59.1.4 #define SYSCALL\_FUNCT\_GET\_PHYS\_MEMORY 8**

get free memory block list for management by process manager

Definition at line 60 of file syscall.h.

Referenced by dispatch\_syscall().

**4.59.1.5 #define SYSCALL\_FUNCT\_GET\_THREAD\_LOCAL\_ADDR 7**

get address of thread local storage for current thread

Definition at line 57 of file syscall.h.

Referenced by dispatch\_syscall().

**4.59.1.6 #define SYSCALL\_FUNCT\_PROC\_BASE 0x400**

start of function numbers for process manager system calls

Definition at line 72 of file syscall.h.

Referenced by dispatch\_syscall().

**4.59.1.7 #define SYSCALL\_FUNCT\_RECEIVE 10**

receive a message on an IPC object

Definition at line 66 of file syscall.h.

Referenced by dispatch\_syscall().

**4.59.1.8 #define SYSCALL\_FUNCT\_REPLY 11**

reply to current message

Definition at line 69 of file syscall.h.

Referenced by dispatch\_syscall().

**4.59.1.9 #define SYSCALL\_FUNCT\_SET\_THREAD\_LOCAL\_ADDR 6**

set address and size of thread local storage for current thread

Definition at line 54 of file syscall.h.

Referenced by dispatch\_syscall().



**4.59.1.10 #define SYSCALL\_FUNCT\_SYSCALL\_METHOD 1**

get best system call implementation number based on CPU features

Definition at line 39 of file syscall.h.

Referenced by dispatch\_syscall().

**4.59.1.11 #define SYSCALL\_FUNCT\_SYSTEM\_BASE 0x1000**

start of function numbers for system IPC objects

Definition at line 75 of file syscall.h.

Referenced by dispatch\_syscall().

**4.59.1.12 #define SYSCALL\_FUNCT\_THREAD\_CREATE 4**

create a new thread

Definition at line 48 of file syscall.h.

Referenced by dispatch\_syscall().

**4.59.1.13 #define SYSCALL\_FUNCT\_THREAD\_YIELD 5**

relinquish the CPU and allow the next thread to run

Definition at line 51 of file syscall.h.

Referenced by dispatch\_syscall().

**4.59.1.14 #define SYSCALL\_FUNCT\_USER\_BASE 0x4000**

start of function numbers for user IPC objects

Definition at line 78 of file syscall.h.

**4.59.1.15 #define SYSCALL\_IRQ 0x80**

interrupt vector for system call software interrupt

Definition at line 36 of file syscall.h.

Referenced by dispatch\_interrupt().

**4.59.1.16 #define SYSCALL\_METHOD\_FAST\_AMD 1**

AMD's fast system call method (SYSCALL/SYSLEAVE)

Definition at line 85 of file syscall.h.

**4.59.1.17 #define SYSCALL\_METHOD\_FAST\_INTEL 0**

Intel's fast system call method (SYSENTER/SYSEXIT)

Definition at line 82 of file syscall.h.

#### 4.59.1.18 `#define SYSCALL_METHOD_INTR 2`

slow/safe system call method using interrupts

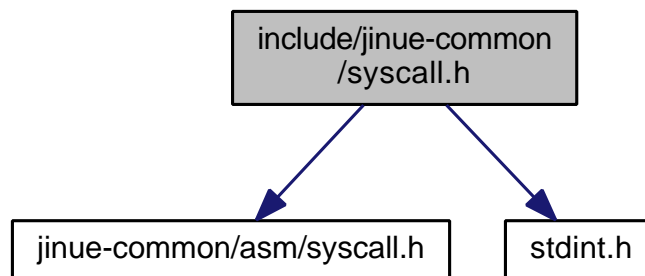
Definition at line 88 of file syscall.h.

## 4.60 `include/jinue-common/syscall.h` File Reference

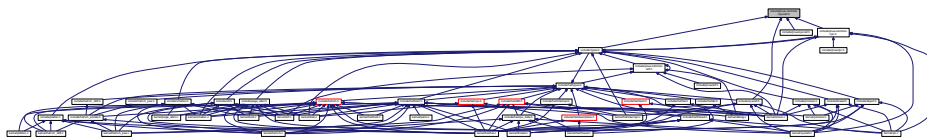
```
#include <jinue-common/asm/syscall.h>
```

```
#include <stdint.h>
```

Include dependency graph for syscall.h:



This graph shows which files directly or indirectly include this file:



## Data Structures

- struct `jinue_syscall_args_t`

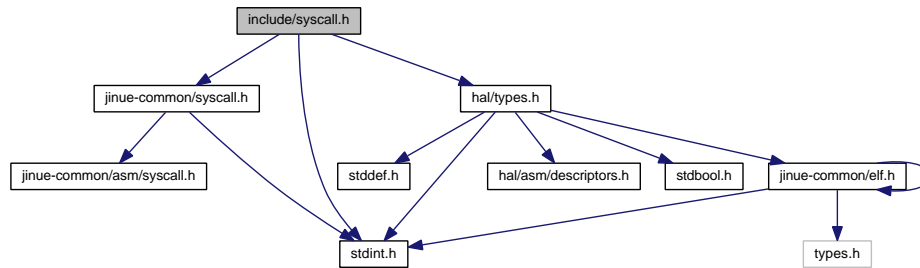
## 4.61 `include/syscall.h` File Reference

```
#include <jinue-common/syscall.h>
```

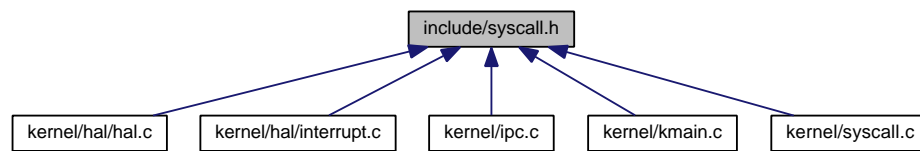
```
#include <hal/types.h>
```

```
#include <stdint.h>
```

Include dependency graph for syscall.h:



This graph shows which files directly or indirectly include this file:



## Functions

- void **dispatch\_syscall** (trapframe\_t \*trapframe)

### 4.61.1 Function Documentation

#### 4.61.1.1 void dispatch\_syscall ( trapframe\_t \* trapframe )

TODO for check negative values (especially -1)

TODO: permission check

TODO: permission check, sanity check (data size vs buffer size)

TODO: check user pointer

Definition at line 48 of file syscall.c.

References jinue\_mem\_entry\_t::addr, e820\_t::addr, jinue\_syscall\_args\_t::arg0, jinue\_syscall\_args\_t::arg1, jinue\_syscall\_args\_t::arg2, jinue\_syscall\_args\_t::arg3, boot\_info, CONSOLE\_DEFAULT\_COLOR, console\_printn(), console\_putc(), object\_ref\_t::cookie, boot\_info\_t::e820\_entries, boot\_info\_t::e820\_map, jinue\_mem\_map\_t::entry, object\_ref\_t::flags, get\_boot\_info(), ipc\_t::header, IPC\_FLAG\_NONE, IPC\_FLAG\_SYSTEM, ipc\_get\_proc\_object(), ipc\_object\_create(), ipc\_receive(), ipc\_reply(), ipc\_send(), JINUE\_EAGAIN, JINUE\_EINVAL, JINUE\_ENOSYS, JINUE\_IPC\_PROC, JINUE\_IPC\_SYSTEM, NULL, jinue\_mem\_map\_t::num\_entries, object\_ref\_t::object, OBJECT\_REF\_FLAG\_OWNED, OBJECT\_REF\_FLAG\_VALID, printk(), thread\_t::process, process\_get\_descriptor(), process\_unused\_descriptor(), jinue\_mem\_entry\_t::size, e820\_t::size, SYSCALL\_FUNCT\_CONSOLE\_PUTC, SYSCALL\_FUNCT\_CONSOLE\_PUTS, SYSCALL\_FUNCT\_CREATE\_IPC, SYSCALL\_FUNCT\_GET\_PHYS\_MEMORY, SYSCALL\_FUNCT\_GET\_THREAD\_LOCAL\_ADDR, SYSCALL\_FUNCT\_PROC\_BASE, SYSCALL\_FUNCT\_RECEIVE, SYSCALL\_FUNCT\_REPLY, SYSCALL\_FUNCT\_SET\_THREAD\_LOCAL\_ADDR, SYSCALL\_FUNCT\_SYSCALL\_METHOD, SYSCALL\_FUNCT\_SYSTEM\_BASE, SYSCALL\_FUNCT\_THREAD\_CREATE, SYSCALL\_FUNCT\_THREAD\_YIELD, syscall\_method, thread\_create(), thread\_yield\_from(), jinue\_mem\_entry\_t::type, and e820\_t::type.

Referenced by dispatch\_interrupt().

```

48         {
49     jinue_syscall_args_t *args = (jinue_syscall_args_t *)&trapframe->msg_arg0;
50
51     uintptr_t function_number = args->arg0;
52
53     if(function_number < SYSCALL_FUNC_PROC_BASE) {
54         /* microkernel system calls */
55         switch(function_number) {
56
57             case SYSCALL_FUNC_SYSCALL_METHOD:
58                 syscall_args_set_return(args, syscall_method);
59                 break;
60
61             case SYSCALL_FUNC_CONSOLE_PUTC:
62                 console_putc(
63                     (char)args->arg1,
64                     CONSOLE_DEFAULT_COLOR);
65                 syscall_args_set_return(args, 0);
66                 break;
67
68             case SYSCALL_FUNC_CONSOLE_PUTS:
69                 console_printn(
70                     (char *)args->arg2,
71                     jinue_args_get_data_size(args),
72                     CONSOLE_DEFAULT_COLOR);
73                 syscall_args_set_return(args, 0);
74                 break;
75
76             case SYSCALL_FUNC_THREAD_CREATE:
77             {
78                 thread_t *thread = thread_create(
79                     /* TODO use arg1 as an address space reference if specified */
80                     get_current_thread()->process,
81                     (addr_t)args->arg2,
82                     (addr_t)args->arg3);
83
84                 if(thread == NULL) {
85                     syscall_args_set_error(args, JINUE_EAGAIN);
86                 }
87                 else {
88                     syscall_args_set_return(args, 0);
89                 }
90             }
91             break;
92
93             case SYSCALL_FUNC_THREAD_YIELD:
94                 thread_yield_from(
95                     get_current_thread(),
96                     false, /* don't block */
97                     args->arg1); /* destroy (aka. exit) thread if true */
98                 syscall_args_set_return(args, 0);
99                 break;
100
101             case SYSCALL_FUNC_SET_THREAD_LOCAL_ADDR:
102                 thread_context_set_local_storage(
103                     &get_current_thread()->thread_ctx,
104                     (addr_t)args->arg1,
105                     (size_t)args->arg2);
106                 syscall_args_set_return(args, 0);
107                 break;
108
109             case SYSCALL_FUNC_GET_THREAD_LOCAL_ADDR:
110                 syscall_args_set_return_ptr(
111                     args,
112                     thread_context_get_local_storage(
113                         &get_current_thread()->thread_ctx));
114                 break;
115
116             case SYSCALL_FUNC_GET_PHYS_MEMORY:
117             {
118                 unsigned int idx;
119
120                 size_t buffer_size = jinue_args_get_buffer_size(args);
121                 jinue_mem_map_t *map = (jinue_mem_map_t *)jinue_args_get_buffer_ptr(args);
122                 const boot_info_t *boot_info = get_boot_info();
123
124                 if(buffer_size < sizeof(jinue_mem_map_t) + boot_info->e820_entries * sizeof(
125                     jinue_mem_entry_t) ) {
126                     syscall_args_set_error(args, JINUE_EINVAL);
127                 }
128                 else {

```

```

132         map->num_entries = boot_info->e820_entries;
133
134         for(idx = 0; idx < map->num_entries; ++idx) {
135             map->entry[idx].addr = boot_info->e820_map[idx].addr;
136             map->entry[idx].size = boot_info->e820_map[idx].size;
137             map->entry[idx].type = boot_info->e820_map[idx].type;
138         }
139
140         syscall_args_set_return(args, 0);
141     }
142 }
143
144     break;
145
146 case SYSCALL_FUNCT_CREATE_IPC:
147 {
148     ipc_t *ipc;
149
150     thread_t *thread = get_current_thread();
151
152     int fd = process_unused_descriptor(thread->process);
153
154     if(fd < 0) {
155         syscall_args_set_error(args, JINUE_EAGAIN);
156         break;
157     }
158
159     if(args->arg1 & JINUE_IPC_PROC) {
160         ipc = ipc_get_proc_object();
161     }
162     else {
163         int flags = IPC_FLAG_NONE;
164
165         if(args->arg1 & JINUE_IPC_SYSTEM) {
166             flags |= IPC_FLAG_SYSTEM;
167         }
168
169         ipc = ipc_object_create(flags);
170
171         if(ipc == NULL) {
172             syscall_args_set_error(args, JINUE_EAGAIN);
173             break;
174         }
175     }
176
177     object_ref_t *ref = process_get_descriptor(thread->process, fd);
178
179     object_addr_t(&ipc->header);
180
181     ref->object = &ipc->header;
182     ref->flags = OBJECT_REF_FLAG_VALID | OBJECT_REF_FLAG_OWNER;
183     ref->cookie = 0;
184
185     syscall_args_set_return(args, fd);
186 }
187
188     break;
189 case SYSCALL_FUNCT_RECEIVE:
190     ipc_receive(args);
191     break;
192
193 case SYSCALL_FUNCT_REPLY:
194     ipc_reply(args);
195     break;
196
197 default:
198     printk("SYSCALL: function %u arg1=%u(0x%x) arg2=%u(0x%x) arg3=%u(0x%x)\n",
199           function_number,
200           args->arg1, args->arg1,
201           args->arg2, args->arg2,
202           args->arg3, args->arg3 );
203
204     syscall_args_set_error(args, JINUE_ENOSYS);
205 }
206
207 else if(function_number < SYSCALL_FUNCT_SYSTEM_BASE) {
208     /* process manager system calls */
209     printk("PROC SYSCALL: function %u arg1=%u(0x%x) arg2=%u(0x%x) arg3=%u(0x%x)\n",
210           function_number,
211           args->arg1, args->arg1,
212           args->arg2, args->arg2,
213           args->arg3, args->arg3 );

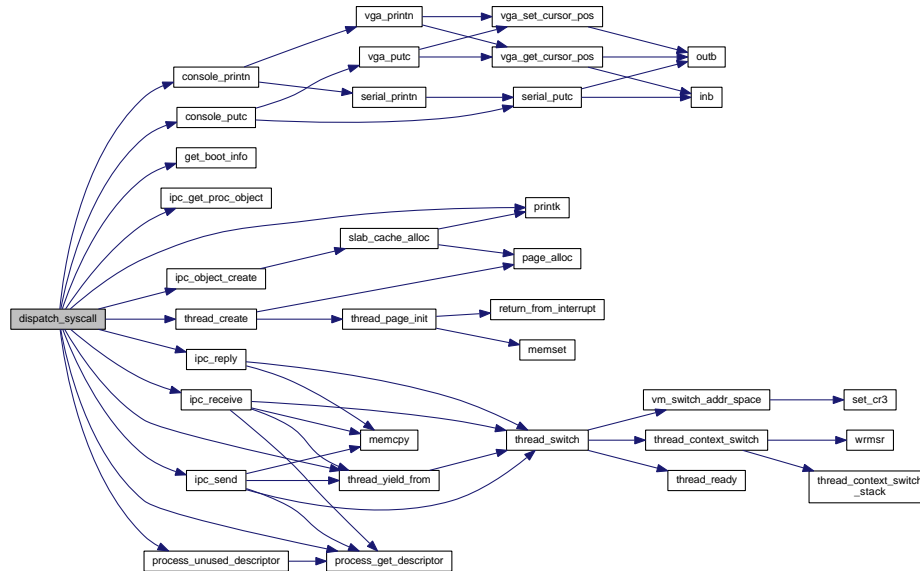
```

```

213
214     syscall_args_set_error(args, JINUE_ENOSYS);
215 }
216 else {
217     /* inter-process message */
218     ipc_send(args);
219 }
220 }

```

Here is the call graph for this function:



## 4.62 include/kbd.h File Reference

### Functions

- void **any\_key** (void)

### 4.62.1 Function Documentation

#### 4.62.1.1 void any\_key ( void )

Definition at line 36 of file kbd.c.

References [inb\(\)](#), and [printk\(\)](#).

```

36     {
37     unsigned char buffer;
38     bool ignore;
39
40     /* prompt */
41     printk("(press enter)");
42
43     /* wait for key, ignore break codes */
44     ignore = false;
45     while(1) {
46         do {
47             buffer = inb(0x64);
48         } while( (buffer & 1) == 0 );

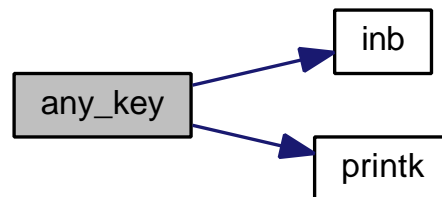
```

```

49
50     buffer = inb(0x60);
51
52     if(buffer == 0x0e || buffer == 0x0f) {
53         ignore = true;
54         continue;
55     }
56
57     if(ignore) {
58         ignore = false;
59         continue;
60     }
61
62     if(buffer == 0x1c || buffer == 0x5a) {
63         break;
64     }
65 }
66
67 /* advance cursor */
68 printk("\n");
69 }

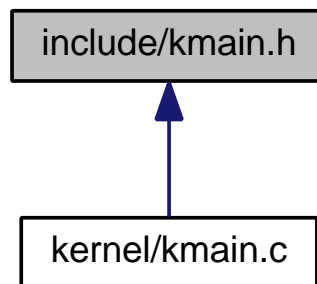
```

Here is the call graph for this function:



## 4.63 include/kmain.h File Reference

This graph shows which files directly or indirectly include this file:



### Functions

- void **kmain** (void)

#### 4.63.1 Function Documentation

##### 4.63.1.1 void kmain ( void )

Definition at line 67 of file kmain.c.

References `process_t::addr_space`, `boot_alloc_init()`, `boot_info_t::boot_heap`, `boot_info_check()`, `BUILD_HOST`, `BUILD_TIME`, `boot_info_t::cmdline`, `console_init()`, `elf_load()`, `elf_info_t::entry`, `get_boot_info()`, `GIT_REVISION`, `hal_init()`, `ipc_boot_init()`, `boot_info_t::kernel_size`, `mem_check_memory()`, `NULL`, `panic()`, `printk()`, `process_boot_init()`, `process_create_initial()`, `boot_info_t::ramdisk_size`, `boot_info_t::ramdisk_start`, `elf_info_t::stack_addr`, `thread_create_boot()`, `thread_yield_from()`, and `VGA_COLOR_YELLOW`.

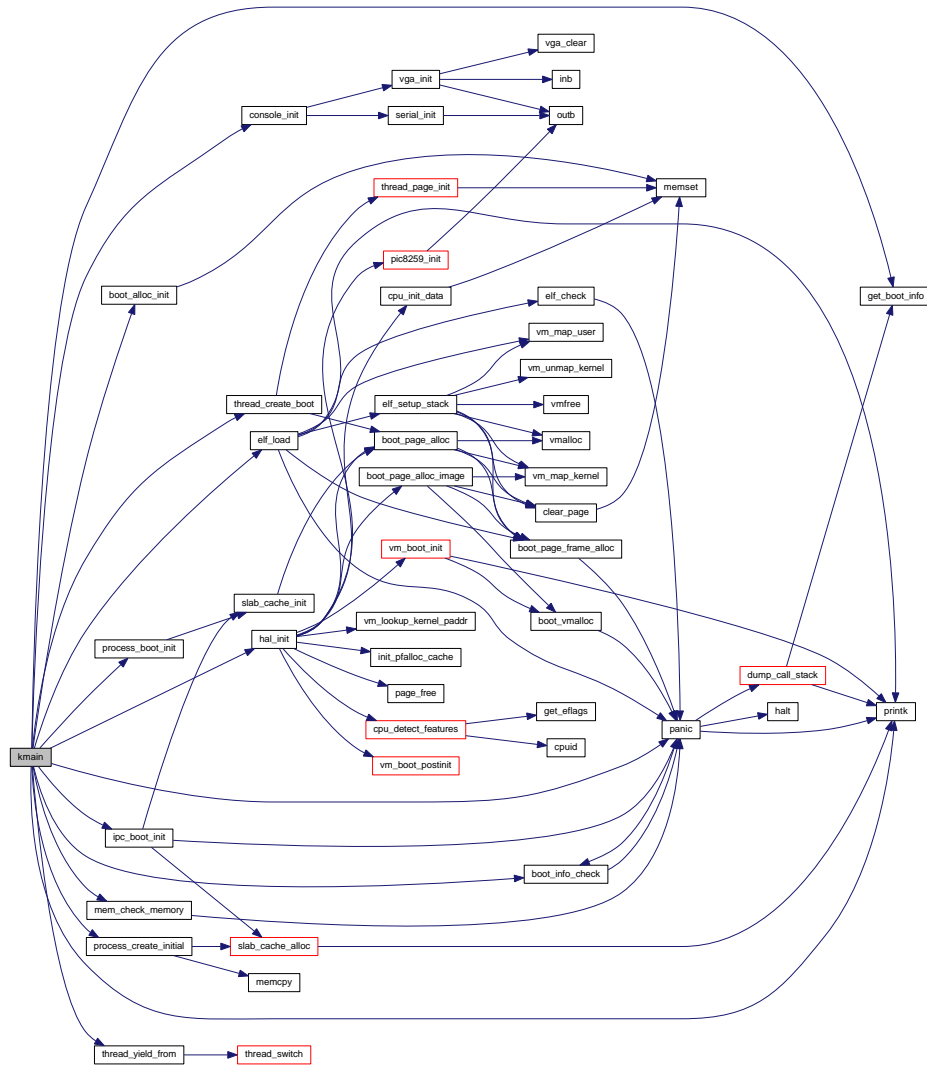
```

67     {
68     elf_info_t elf_info;
69
70     /* initialize console and say hello */
71     console_init();
72
73     /* Say hello. */
74     printk("Kernel revision " GIT_REVISION " built " BUILD_TIME " on "
75           BUILD_HOST "\n");
76
77     const boot_info_t *boot_info = get_boot_info();
78     (void)boot_info_check(true);
79
80     printk("Kernel size is %u bytes.\n", boot_info->kernel_size);
81
82     if(boot_info->ramdisk_start == 0 || boot_info->ramdisk_size == 0) {
83         printk("%kWarning: no initial RAM disk loaded.\n", VGA_COLOR_YELLOW);
84     }
85     else {
86         printk("RAM disk with size %u bytes loaded at address %x.\n", boot_info->
87               ramdisk_size, boot_info->ramdisk_start);
88     }
89
90     printk("Kernel command line:\n", boot_info->kernel_size);
91     printk("    %s\n", boot_info->cmdline);
92
93     /* Initialize the boot allocator. */
94     boot_alloc_t boot_alloc;
95     boot_alloc_init(&boot_alloc, boot_info->boot_heap);
96     mem_check_memory(&boot_alloc, boot_info);
97
98     /* initialize hardware abstraction layer */
99     hal_init(&boot_alloc, boot_info);
100
101     /* initialize caches */
102     ipc_boot_init(&boot_alloc);
103     process_boot_init(&boot_alloc);
104
105     /* create process for process manager */
106     process_t *process = process_create_initial();
107
108     if(process == NULL) {
109         panic("Could not create initial process.");
110     }
111
112     /* load process manager binary */
113     Elf32_Ehdr *elf = find_process_manager();
114     elf_load(&elf_info, elf, &process->addr_space, &boot_alloc);
115
116     /* create initial thread */
117     thread_t *thread = thread_create_boot(
118         process,
119         elf_info.entry,
120         elf_info.stack_addr,
121         &boot_alloc);
122
123     if(thread == NULL) {
124         panic("Could not create initial thread.");
125     }
126
127     /* start process manager
128     *
129     * We switch from NULL since this is the first thread. */
130     thread_yield_from(
131         NULL,
132         false, /* don't block */
133         false); /* don't destroy */
134
135     /* should never happen */
136     panic("thread_yield_from() returned in kmain()");
137 }

```

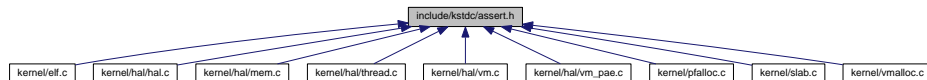


Here is the call graph for this function:



## 4.64 include/kstdd/assert.h File Reference

This graph shows which files directly or indirectly include this file:



## Macros

- `#define assert(expr)`

## Functions

- void **\_\_assert\_failed** (const char \*expr, const char \*file, unsigned int line, const char \*func)

### 4.64.1 Macro Definition Documentation

#### 4.64.1.1 #define assert( *expr* )

##### Value:

```
( \
    (expr)?(void)0:( __assert_failed(#expr, __FILE__, __LINE__, __func__ ) ) \
)
```

Definition at line 46 of file assert.h.

Referenced by hal\_init(), mem\_check\_memory(), slab\_cache\_alloc(), slab\_cache\_init(), thread\_context\_switch(), vm\_change\_flags(), vm\_destroy\_addr\_space(), vm\_lookup\_kernel\_paddr(), and vm\_map\_early().

### 4.64.2 Function Documentation

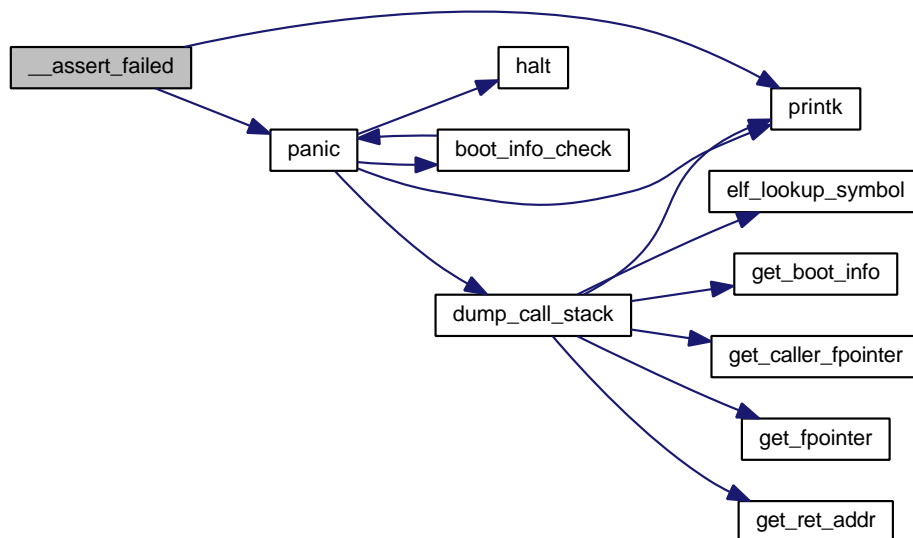
#### 4.64.2.1 void \_\_assert\_failed ( const char \* *expr*, const char \* *file*, unsigned int *line*, const char \* *func* )

Definition at line 36 of file c-assert.c.

References panic(), and printk().

```
40             {
41
42     printk(
43         "ASSERTION FAILED [%s]: %s at line %u in function %s.\n",
44         expr, file, line, func );
45
46     panic("Assertion failed.");
47 }
```

Here is the call graph for this function:



## 4.65 include/kstdc/stdarg.h File Reference

### Macros

- `#define va_start(ap, parmN) __builtin_va_start((ap), (parmN))`
- `#define va_arg __builtin_va_arg`
- `#define va_end __builtin_va_end`
- `#define va_copy(dest, src) __builtin_va_copy((dest), (src))`

### Typedefs

- `typedef __builtin_va_list va_list`

#### 4.65.1 Macro Definition Documentation

##### 4.65.1.1 `#define va_arg __builtin_va_arg`

Definition at line 38 of file stdarg.h.

##### 4.65.1.2 `#define va_copy( dest, src ) __builtin_va_copy((dest), (src))`

Definition at line 40 of file stdarg.h.

##### 4.65.1.3 `#define va_end __builtin_va_end`

Definition at line 39 of file stdarg.h.

##### 4.65.1.4 `#define va_start( ap, parmN ) __builtin_va_start((ap), (parmN))`

Definition at line 37 of file stdarg.h.

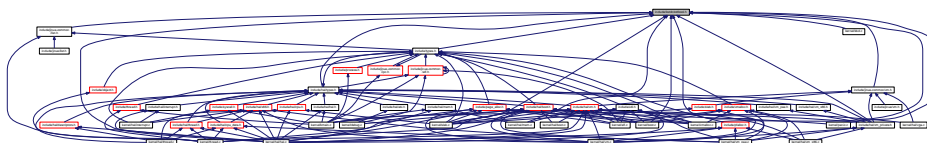
#### 4.65.2 Typedef Documentation

##### 4.65.2.1 `typedef __builtin_va_list va_list`

Definition at line 35 of file stdarg.h.

## 4.66 include/kstdc/stdbool.h File Reference

This graph shows which files directly or indirectly include this file:



## Macros

- `#define bool _Bool`
- `#define true 1`
- `#define false 0`
- `#define __bool_true_false_are_defined 1`

### 4.66.1 Macro Definition Documentation

#### 4.66.1.1 `#define __bool_true_false_are_defined 1`

Definition at line 39 of file `stdbool.h`.

#### 4.66.1.2 `#define bool _Bool`

Definition at line 35 of file `stdbool.h`.

#### 4.66.1.3 `#define false 0`

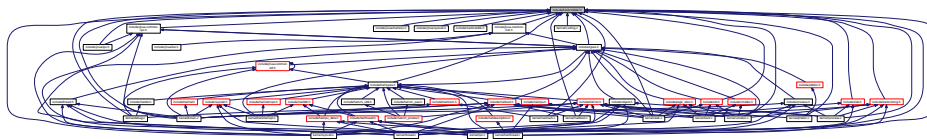
Definition at line 37 of file `stdbool.h`.

#### 4.66.1.4 `#define true 1`

Definition at line 36 of file `stdbool.h`.

## 4.67 `include/kstdc/stddef.h` File Reference

This graph shows which files directly or indirectly include this file:



## Macros

- `#define NULL 0`
- `#define offsetof(type, member) ( (size_t) &((type *)0)->member )`

## Typedefs

- typedef signed long **ptrdiff\_t**
- typedef unsigned long **size\_t**
- typedef int **wchar\_t**

### 4.67.1 Macro Definition Documentation

#### 4.67.1.1 `#define NULL 0`

Definition at line 40 of file `stddef.h`.

Referenced by `add_page_frame()`, `boot_heap_pop()`, `boot_info_check()`, `boot_page_alloc_early()`, `boot_vmalloc()`, `cpu_init_data()`, `dispatch_syscall()`, `dump_call_stack()`, `elf_lookup_symbol()`, `ipc_boot_init()`, `ipc_object_create()`, `ipc_receive()`, `ipc_reply()`, `ipc_send()`, `kmain()`, `page_alloc()`, `page_alloc_is_empty()`, `process_boot_init()`, `process_create()`, `process_create_initial()`, `process_get_descriptor()`, `remove_page_frame()`, `slab_cache_alloc()`, `slab_cache_free()`, `slab_cache_init()`, `thread_context_switch()`, `thread_create()`, `thread_page_init()`, `thread_switch()`, `vm_change_flags()`, `vm_destroy_addr_space()`, `vm_lookup_kernel_paddr()`, `vm_map_kernel()`, `vm_pae_create_addr_space()`, `vm_pae_create_pdpt_cache()`, `vm_pae_lookup_page_directory()`, and `vm_unmap_kernel()`.

#### 4.67.1.2 `#define offsetof( type, member ) ( (size_t) &((type *)0)->member )`

Definition at line 43 of file `stddef.h`.

### 4.67.2 Typedef Documentation

#### 4.67.2.1 `typedef signed long ptrdiff_t`

Definition at line 35 of file `stddef.h`.

#### 4.67.2.2 `typedef unsigned long size_t`

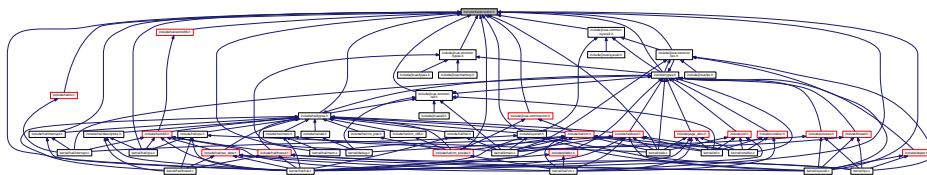
Definition at line 36 of file `stddef.h`.

#### 4.67.2.3 `typedef int wchar_t`

Definition at line 37 of file `stddef.h`.

## 4.68 include/kstdc/stdint.h File Reference

This graph shows which files directly or indirectly include this file:



### Macros

- `#define INT64_C(x) (x##LL)`
- `#define UINT64_C(x) (x##ULL)`

## Typedefs

- typedef signed char **int8\_t**
- typedef short int **int16\_t**
- typedef int **int32\_t**
- typedef long long int **int64\_t**
- typedef unsigned char **uint8\_t**
- typedef unsigned short int **uint16\_t**
- typedef unsigned int **uint32\_t**
- typedef unsigned long long int **uint64\_t**
- typedef int **intptr\_t**
- typedef unsigned int **uintptr\_t**

### 4.68.1 Macro Definition Documentation

#### 4.68.1.1 `#define INT64_C( x )(x##LL)`

Definition at line 35 of file stdint.h.

#### 4.68.1.2 `#define UINT64_C( x )(x##ULL)`

Definition at line 37 of file stdint.h.

### 4.68.2 Typedef Documentation

#### 4.68.2.1 `typedef short int int16_t`

Definition at line 41 of file stdint.h.

#### 4.68.2.2 `typedef int int32_t`

Definition at line 43 of file stdint.h.

#### 4.68.2.3 `typedef long long int int64_t`

Definition at line 45 of file stdint.h.

#### 4.68.2.4 `typedef signed char int8_t`

Definition at line 39 of file stdint.h.

#### 4.68.2.5 `typedef int intptr_t`

Definition at line 57 of file stdint.h.

#### 4.68.2.6 typedef unsigned short int uint16\_t

Definition at line 50 of file stdint.h.

#### 4.68.2.7 typedef unsigned int uint32\_t

Definition at line 52 of file stdint.h.

#### 4.68.2.8 typedef unsigned long long int uint64\_t

Definition at line 54 of file stdint.h.

#### 4.68.2.9 typedef unsigned char uint8\_t

Definition at line 48 of file stdint.h.

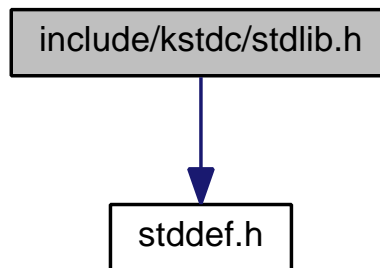
#### 4.68.2.10 typedef unsigned int uintptr\_t

Definition at line 59 of file stdint.h.

## 4.69 include/kstdc/stdlib.h File Reference

```
#include <stddef.h>
```

Include dependency graph for stdlib.h:



### Macros

- `#define EXIT_SUCCESS 0`
- `#define EXIT_FAILURE 1`

#### 4.69.1 Macro Definition Documentation

##### 4.69.1.1 `#define EXIT_FAILURE 1`

Definition at line 39 of file stdlib.h.

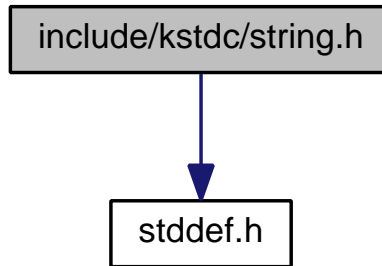
#### 4.69.1.2 #define EXIT\_SUCCESS 0

Definition at line 37 of file stdlib.h.

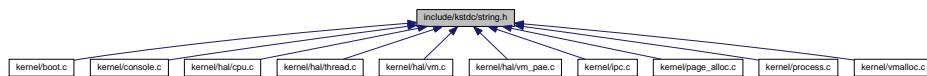
## 4.70 include/kstdc/string.h File Reference

```
#include <stddef.h>
```

Include dependency graph for string.h:



This graph shows which files directly or indirectly include this file:



## Functions

- void \* **memset** (void \*s, int c, **size\_t** n)
- void \* **memcpy** (void \*dest, const void \*src, **size\_t** n)
- **size\_t** **strlen** (const char \*s)

### 4.70.1 Function Documentation

#### 4.70.1.1 void\* memcpy ( void \* dest, const void \* src, size\_t n )

Definition at line 45 of file c-string.c.

Referenced by ipc\_receive(), ipc\_reply(), ipc\_send(), and process\_create\_initial().

```

45         {
46     size_t      idx;
47     char        *cdest  = dest;
48     const char  *csrc   = src;
49
50     for(idx = 0; idx < n; ++idx) {
51         cdest[idx] = csrc[idx];
52     }
53
54     return dest;
55 }
```



#### 4.70.1.2 void\* memset ( void \* s, int c, size\_t n )

Definition at line 34 of file c-string.c.

Referenced by boot\_alloc\_init(), clear\_page(), cpu\_init\_data(), thread\_page\_init(), and vm\_pae\_lookup\_page\_directory().

```
34                                     {
35     size_t   idx;
36     char     *cs = s;
37
38     for(idx = 0; idx < n; ++idx) {
39         cs[idx] = c;
40     }
41
42     return s;
43 }
```

#### 4.70.1.3 size\_t strlen ( const char \* s )

Definition at line 57 of file c-string.c.

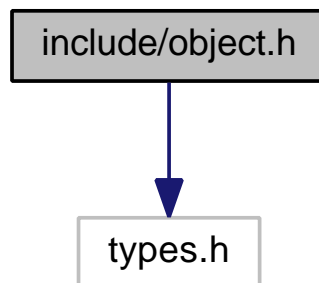
Referenced by console\_print().

```
57                                     {
58     size_t count = 0;
59
60     while(*s != 0) {
61         ++s;
62         ++count;
63     }
64
65     return count;
66 }
```

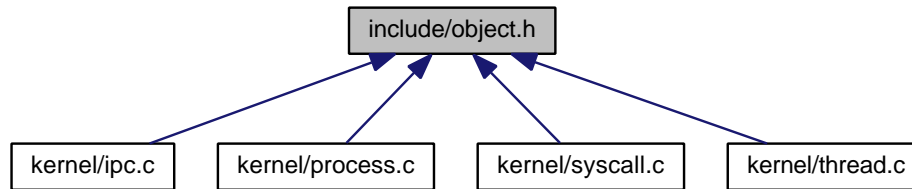
## 4.71 include/object.h File Reference

```
#include <types.h>
```

Include dependency graph for object.h:



This graph shows which files directly or indirectly include this file:



## Macros

- `#define OBJECT_FLAG_NONE 0`
- `#define OBJECT_FLAG_DESTROYED (1<<0)`
- `#define OBJECT_REF_FLAG_NONE 0`
- `#define OBJECT_REF_FLAG_VALID (1<<0)`
- `#define OBJECT_REF_FLAG_CLOSED (1<<1)`
- `#define OBJECT_REF_FLAG_OWNER (1<<2)`
- `#define OBJECT_TYPE_THREAD 1`
- `#define OBJECT_TYPE_IPC 2`
- `#define OBJECT_TYPE_PROCESS 3`

### 4.71.1 Macro Definition Documentation

#### 4.71.1.1 `#define OBJECT_FLAG_DESTROYED (1<<0)`

Definition at line 42 of file object.h.

#### 4.71.1.2 `#define OBJECT_FLAG_NONE 0`

Definition at line 40 of file object.h.

#### 4.71.1.3 `#define OBJECT_REF_FLAG_CLOSED (1<<1)`

Definition at line 49 of file object.h.

Referenced by `ipc_receive()`, and `ipc_send()`.

#### 4.71.1.4 `#define OBJECT_REF_FLAG_NONE 0`

Definition at line 45 of file object.h.

#### 4.71.1.5 `#define OBJECT_REF_FLAG_OWNER (1<<2)`

Definition at line 51 of file object.h.

Referenced by `dispatch_syscall()`.

4.71.1.6 `#define OBJECT_REF_FLAG_VALID (1<<0)`

Definition at line 47 of file object.h.

Referenced by `dispatch_syscall()`.

4.71.1.7 `#define OBJECT_TYPE_IPC 2`

Definition at line 56 of file object.h.

Referenced by `ipc_receive()`, and `ipc_send()`.

4.71.1.8 `#define OBJECT_TYPE_PROCESS 3`

Definition at line 58 of file object.h.

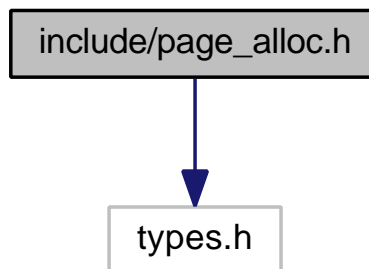
4.71.1.9 `#define OBJECT_TYPE_THREAD 1`

Definition at line 54 of file object.h.

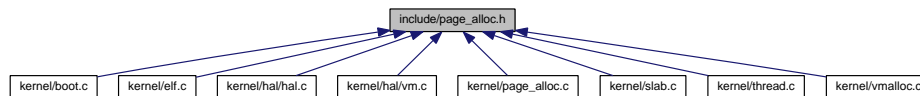
## 4.72 include/page\_alloc.h File Reference

```
#include <types.h>
```

Include dependency graph for `page_alloc.h`:



This graph shows which files directly or indirectly include this file:



## Functions

- `void * page_alloc (void)`  
Allocate a page of kernel memory.
- `void page_free (void *page)`  
Free a page of kernel memory.

- **bool** `page_alloc_is_empty` (void)

*Check that pages are available to be allocated.*

- **bool** `add_page_frame` (kern\_paddr\_t paddr)

*Map a page frame and add it to the page allocator.*

- **kern\_paddr\_t** `remove_page_frame` (void)

*Remove a page frame from the allocator.*

- void **clear\_page** (void \*page)

*Clear a page by writing all bytes to zero.*

## 4.72.1 Function Documentation

### 4.72.1.1 bool add\_page\_frame ( kern\_paddr\_t paddr )

Map a page frame and add it to the page allocator.

This function is used to implement a system call that allows userspace to provide additional page frames to the kernel. This function fails when no more pages of kernel address space can be allocated with **vmalloc()** (p.275) to map the provided page frame.

#### Parameters

<i>paddr</i>	physical address of the provided page frame
--------------	---

#### Returns

true if the function succeeded

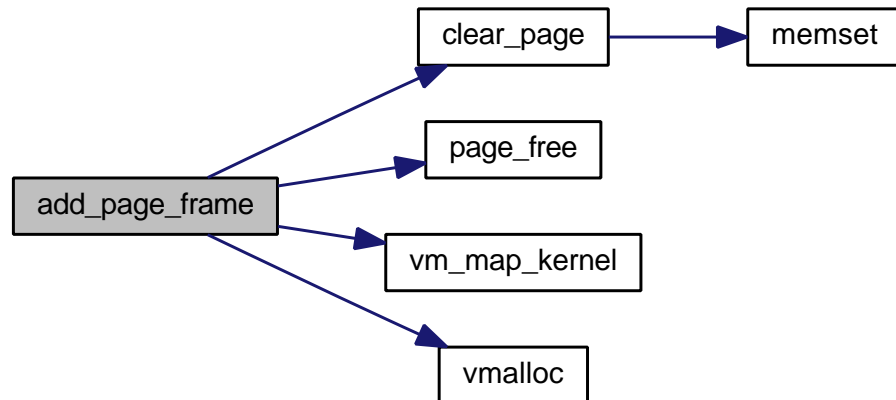
Definition at line 111 of file `page_alloc.c`.

References `clear_page()`, `NULL`, `page_free()`, `VM_FLAG_READ_WRITE`, `vm_map_kernel()`, and `vmalloc()`.

```

111                                     {
112     void *page = vmalloc();
113
114     if (page == NULL) {
115         return false;
116     }
117
118     vm_map_kernel (page, paddr, VM_FLAG_READ_WRITE);
119
120     /* Since this page is coming from userspace, is is important to clear it:
121      * 1) The page may contain sensitive information, which we don't want to
122      *    leak through Meltdown-like vulnerabilities; and
123      * 2) Since the content is userspace-chosen, it could be used for kernel
124      *    vulnerability exploits. */
125     clear_page (page);
126     page_free (page);
127
128     return true;
129 }
```

Here is the call graph for this function:



#### 4.72.1.2 void clear\_page ( void \* *page* )

Clear a page by writing all bytes to zero.

**Parameters**

<i>page</i>	the page to clear
-------------	-------------------

Definition at line 173 of file `page_alloc.c`.

References `memset()`, and `PAGE_SIZE`.

Referenced by `add_page_frame()`, `boot_page_alloc()`, `boot_page_alloc_early()`, `boot_page_alloc_image()`, `elf_setup_stack()`, and `remove_page_frame()`.

```

173     {
174         memset (page, 0, PAGE_SIZE);
175     }
  
```

Here is the call graph for this function:



#### 4.72.1.3 void\* page\_alloc ( void )

Allocate a page of kernel memory.

Pages allocated by this function can be used for any purpose in the kernel, e.g. as slabs for the slab allocator or as page tables.

Pages allocated by this function are not guaranteed to be mapped in the allocations region of the kernel address space (that is, the region managed by `vmalloc()` (p. 275)). While most will be, pages originally allocated in the image region during initialization by calling `boot_page_alloc_image()` (p. 72) can be reclaimed with `page_free()` (p. 252) and then re-allocated by this function.

**Returns**

allocated page

Definition at line 59 of file page\_alloc.c.

References alloc\_page::next, and NULL.

Referenced by remove\_page\_frame(), slab\_cache\_alloc(), thread\_create(), and vm\_clone\_page\_directory().

```

59      {
60  struct alloc_page *alloc_page = head_page;
61
62  if(alloc_page != NULL) {
63      head_page = alloc_page->next;
64  }
65
66  return alloc_page;
67 }
```

**4.72.1.4 bool page\_alloc\_is\_empty ( void )**

Check that pages are available to be allocated.

Page availability can be checked with this function before calling either **page\_alloc()** (p. 251) or **remove\_page\_frame()** (p. 253).

**Returns**

true if pages are available (one or more)

Definition at line 95 of file page\_alloc.c.

References NULL.

```

95      {
96  return head_page == NULL;
97 }
```

**4.72.1.5 void page\_free ( void \* page )**

Free a page of kernel memory.

Pages freed by calling this function are available to be re-allocated by the **page\_alloc()** (p. 251) function. This function can be used to free pages allocated by **page\_alloc()** (p. 251) or to reclaim pages allocated during kernel initialization by **boot\_page\_alloc()** (p. 70) or **boot\_page\_alloc\_image()** (p. 72).

**Parameters**

<i>page</i>	the page to free
-------------	------------------

Definition at line 80 of file page\_alloc.c.

References alloc\_page::next.

Referenced by add\_page\_frame(), hal\_init(), and thread\_destroy().

```

80      {
81  struct alloc_page *alloc_page = page;
82  alloc_page->next = head_page;
83  head_page = alloc_page;
84 }
```

## 4.72.1.6 kern\_paddr\_t remove\_page\_frame ( void )

Remove a page frame from the allocator.

This function is used implement a system call that allows userspace to reclaim free kernel memory for its own use. The address space page is freed with **vmfree()** (p. 276) and the physical address of the underlying page frame is returned.

## Returns

physical address of the freed page frame, or PFNULL if none is available

Definition at line 142 of file page\_alloc.c.

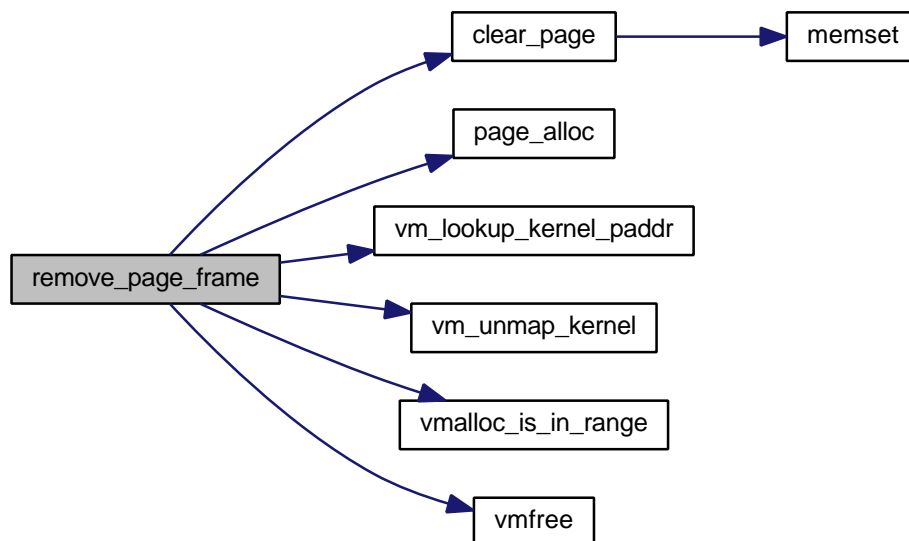
References `clear_page()`, `NULL`, `page_alloc()`, `PFNULL`, `vm_lookup_kernel_paddr()`, `vm_unmap_kernel()`, `vmalloc_is_in_range()`, and `vmfree()`.

```

142
143     void *page = page_alloc();
144
145     if(page == NULL) {
146         return PFNULL;
147     }
148
149     /* This page is going to userspace. Let's clear its content so we don't
150      * leak information about the kernel's internal state that could be useful
151      * for exploiting vulnerabilities. */
152     clear_page(page);
153
154     kern_paddr_t paddr = vm_lookup_kernel_paddr(page);
155
156     vm_unmap_kernel(page);
157
158     /* The page may be in the image region instead of the allocations region if
159      * it was allocated during kernel initialization. */
160     if(vmalloc_is_in_range(page)) {
161         vmfree(page);
162     }
163
164     return paddr;
165 }

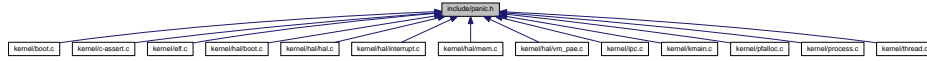
```

Here is the call graph for this function:



## 4.73 include/panic.h File Reference

This graph shows which files directly or indirectly include this file:



### Functions

- **void panic** (const char \*message)

#### 4.73.1 Function Documentation

##### 4.73.1.1 void panic ( const char \* message )

Definition at line 40 of file panic.c.

References `boot_info_check()`, `dump_call_stack()`, `halt()`, `printk()`, and `VGA_COLOR_RED`.

Referenced by `__assert_failed()`, `boot_heap_pop()`, `boot_info_check()`, `boot_page_alloc_early()`, `boot_page_frame_alloc()`, `boot_vmalloc()`, `dispatch_interrupt()`, `elf_check()`, `elf_load()`, `ipc_boot_init()`, `kmain()`, `mem_check_memory()`, and `pfalloc_from()`.

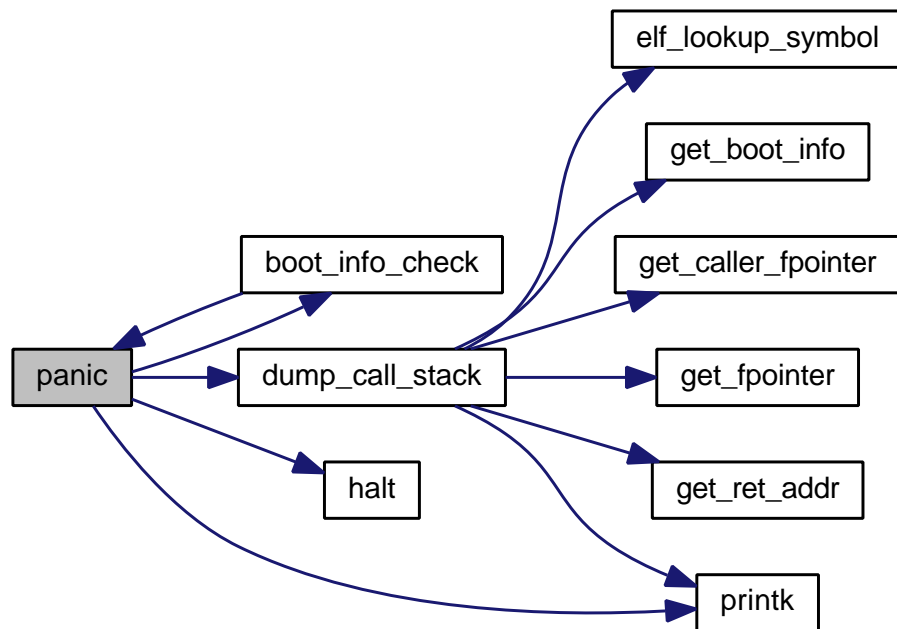
```

40     {
41         static int enter_count = 0;
42
43         ++enter_count;
44
45         /* When things go seriously wrong, things that panic does itself can create
46          * a further panic, for example by triggering a hardware exception. The
47          * enter_count static variable keeps count of the number of times panic()
48          * is entered. */
49         switch(enter_count) {
50             case 1:
51             case 2:
52                 /* The first two times panic() is entered, a panic message is displayed
53                  * along with a full call sack dump. */
54                 printk( "%kKERNEL PANIC%s: %s\n",
55                     VGA_COLOR_RED,
56                     enter_count==1?"": " (recursive)",
57                     message);
58
59                 if( boot_info_check(false) ) {
60                     dump_call_stack();
61                 }
62             else {
63                 printk("Cannot dump call stack because boot information structure is invalid.\n");
64             }
65             break;
66             case 3:
67                 /* The third time, a "recursive count exceeded" message is displayed. We
68                  * try to limit the number of actions we take to limit the chances of a
69                  * further panic. */
70                 printk("%kKERNEL PANIC (recursive count exceeded)\n", VGA_COLOR_RED);
71                 break;
72             default:
73                 /* The fourth time, we do nothing but halt the CPU. */
74                 break;
75         }
76
77         halt();
78     }

```



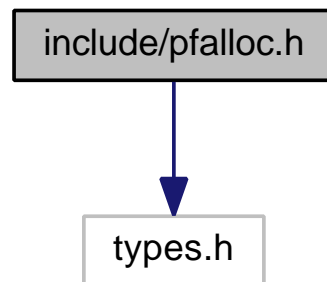
Here is the call graph for this function:



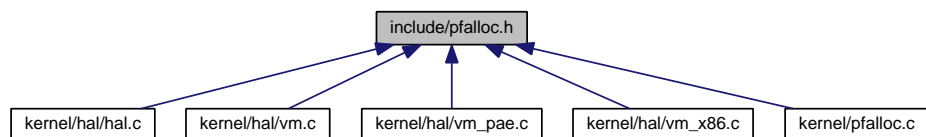
## 4.74 include/pfalloc.h File Reference

```
#include <types.h>
```

Include dependency graph for `pfalloc.h`:



This graph shows which files directly or indirectly include this file:



## Data Structures

- struct **pfalloc\_cache\_t**

## Macros

- **#define KERNEL\_PAGE\_STACK\_SIZE** 1024
- **#define KERNEL\_PAGE\_STACK\_INIT** 32
- **#define pfalloc()** pfalloc\_from(&global\_pfalloc\_cache)
- **#define pffree(p)** pffree\_to(&global\_pfalloc\_cache, (p))

## Functions

- void **init\_pfalloc\_cache** (pfalloc\_cache\_t \*pfcache, kern\_paddr\_t \*stack\_page)
- kern\_paddr\_t **pfalloc\_from** (pfalloc\_cache\_t \*pfcache)
- void **pffree\_to** (pfalloc\_cache\_t \*pfcache, kern\_paddr\_t paddr)

## Variables

- **pfalloc\_cache\_t** global\_pfalloc\_cache

### 4.74.1 Macro Definition Documentation

#### 4.74.1.1 **#define KERNEL\_PAGE\_STACK\_INIT** 32

Definition at line 39 of file pfalloc.h.

Referenced by hal\_init().

#### 4.74.1.2 **#define KERNEL\_PAGE\_STACK\_SIZE** 1024

Definition at line 37 of file pfalloc.h.

Referenced by init\_pfalloc\_cache(), and pffree\_to().

#### 4.74.1.3 **#define pfalloc( )** pfalloc\_from(&global\_pfalloc\_cache)

Definition at line 50 of file pfalloc.h.

Referenced by vm\_pae\_lookup\_page\_directory().

#### 4.74.1.4 **#define pffree( p )** pffree\_to(&global\_pfalloc\_cache, (p))

Definition at line 52 of file pfalloc.h.

Referenced by hal\_init(), and vm\_destroy\_page\_directory().

### 4.74.2 Function Documentation

#### 4.74.2.1 void init\_pfalloc\_cache ( pfalloc\_cache\_t \* pfcache, kern\_paddr\_t \* stack\_page )

Definition at line 40 of file pfalloc.c.

References pfalloc\_cache\_t::count, KERNEL\_PAGE\_STACK\_SIZE, PFNULL, and pfalloc\_cache\_t::ptr.

Referenced by hal\_init().

```

40                                     {
41     kern_paddr_t    *ptr;
42     unsigned int    idx;
43
44     ptr = stack_page;
45
46     for(idx = 0; idx < KERNEL_PAGE_STACK_SIZE; ++idx) {
47         ptr[idx] = PFNULL;
48     }
49
50     pfcache->ptr    = stack_page;
51     pfcache->count  = 0;
52 }
```

#### 4.74.2.2 kern\_paddr\_t pfalloc\_from ( pfalloc\_cache\_t \* pfcache )

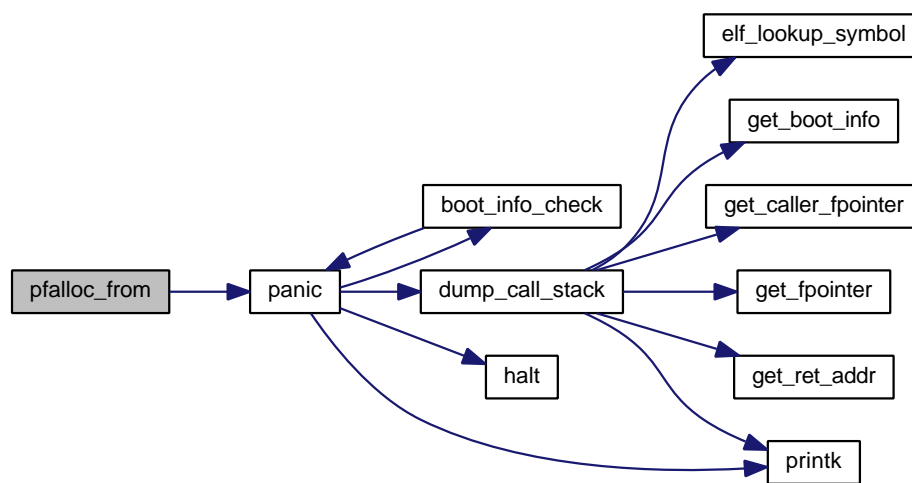
Definition at line 54 of file pfalloc.c.

References pfalloc\_cache\_t::count, panic(), and pfalloc\_cache\_t::ptr.

```

54                                     {
55     if(pfcache->count == 0) {
56         panic("pfalloc_from(): no more pages to allocate");
57     }
58     --pfcache->count;
59
60     return * (--pfcache->ptr);
61 }
62 }
```

Here is the call graph for this function:



#### 4.74.2.3 void pffree\_to ( pfalloc\_cache\_t \* pfcache, kern\_paddr\_t paddr )

We are leaking memory here. Should we panic instead?

Definition at line 64 of file pfalloc.c.

References pfalloc\_cache\_t::count, KERNEL\_PAGE\_STACK\_SIZE, and pfalloc\_cache\_t::ptr.

```

64                                     {
65     if (pfcache->count >= KERNEL_PAGE_STACK_SIZE) {
66         return;
67     }
68 }
69 ++pfcache->count;
70
71 (pfcache->ptr++)[0] = paddr;
72 }
73 }
```

### 4.74.3 Variable Documentation

#### 4.74.3.1 pfalloc\_cache\_t global\_pfalloc\_cache

Definition at line 38 of file pfalloc.c.

Referenced by hal\_init().

## 4.75 include/printk.h File Reference

This graph shows which files directly or indirectly include this file:



## Functions

- void **printk** (const char \*format,...)
- void **print\_unsigned\_int** (unsigned int n, int colour)
- void **print\_hex\_nibble** (unsigned char byte, int colour)
- void **print\_hex\_b** (unsigned char byte, int colour)
- void **print\_hex\_w** (unsigned short word, int colour)
- void **print\_hex\_l** (unsigned long dword, int colour)
- void **print\_hex\_q** (unsigned long long qword, int colour)

### 4.75.1 Function Documentation

#### 4.75.1.1 void print\_hex\_b ( unsigned char byte, int colour )

#### 4.75.1.2 void print\_hex\_l ( unsigned long dword, int colour )

#### 4.75.1.3 void print\_hex\_nibble ( unsigned char byte, int colour )

#### 4.75.1.4 void print\_hex\_q ( unsigned long long qword, int colour )

4.75.1.5 void `print_hex_w` ( unsigned short *word*, int *colour* )

4.75.1.6 void `print_unsigned_int` ( unsigned int *n*, int *colour* )

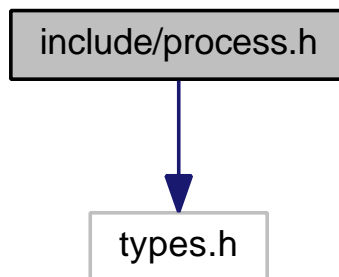
4.75.1.7 void `printk` ( const char \* *format*, ... )

Referenced by `__assert_failed()`, `any_key()`, `boot_info_dump()`, `dispatch_interrupt()`, `dispatch_syscall()`, `dump_call_stack()`, `elf_load()`, `kmain()`, `panic()`, `slab_cache_alloc()`, `slab_cache_free()`, and `vm_boot_init()`.

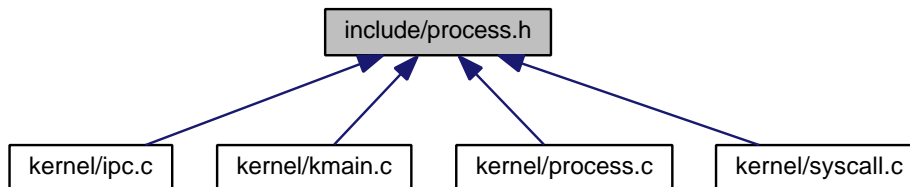
## 4.76 include/process.h File Reference

```
#include <types.h>
```

Include dependency graph for process.h:



This graph shows which files directly or indirectly include this file:



### Functions

- void **process\_boot\_init** ( **boot\_alloc\_t** \**boot\_alloc* )
- **process\_t** \* **process\_create** (void)
- **process\_t** \* **process\_create\_initial** (void)
- **object\_ref\_t** \* **process\_get\_descriptor** ( **process\_t** \**process*, int *fd* )
- int **process\_unused\_descriptor** ( **process\_t** \**process* )

### 4.76.1 Function Documentation

4.76.1.1 void `process_boot_init` ( **boot\_alloc\_t** \* *boot\_alloc* )

Definition at line 49 of file `process.c`.

References NULL, slab\_cache\_init(), and SLAB\_DEFAULTS.

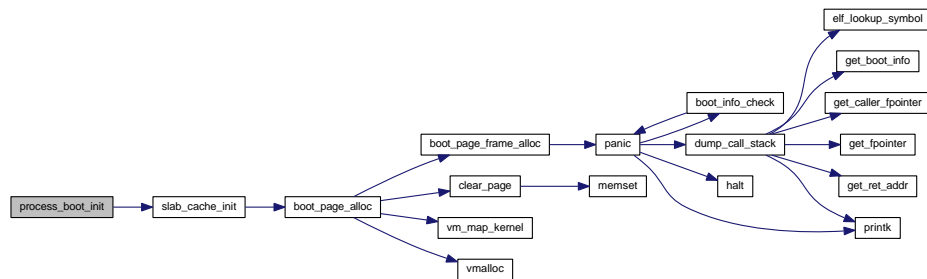
Referenced by kmain().

```

49                                     {
50     slab_cache_init(
51         &process_cache,
52         "process_cache",
53         sizeof(process_t),
54         0,
55         process_ctor,
56         NULL,
57         SLAB_DEFAULTS,
58         boot_alloc);
59 }

```

Here is the call graph for this function:



#### 4.76.1.2 process\_t\* process\_create ( void )

Definition at line 65 of file process.c.

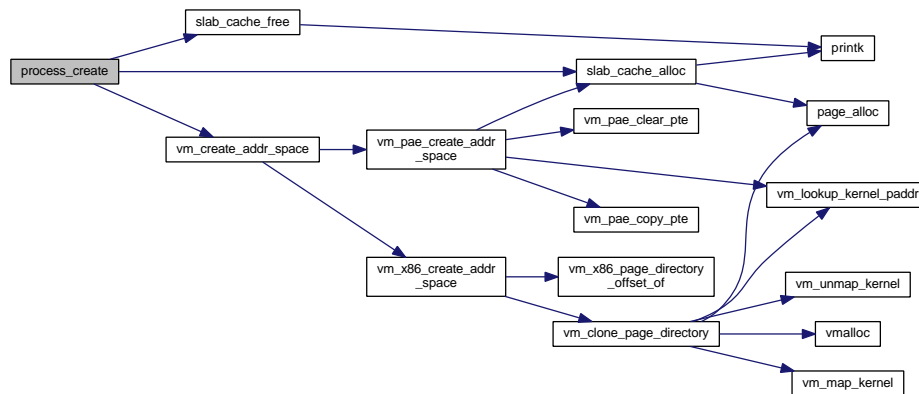
References process\_t::addr\_space, NULL, slab\_cache\_alloc(), slab\_cache\_free(), and vm\_create\_addr\_space().

```

65                                     {
66     process_t *process = slab_cache_alloc(&process_cache);
67
68     if(process != NULL) {
69         addr_space_t *addr_space = vm_create_addr_space(&process->addr_space);
70
71         /* The address space object is located inside the process object but the
72          * call to vm_create_addr_space() above can still fail if we cannot
73          * allocate the initial page directory/tables or, when PAE is enabled,
74          * if we cannot allocate a PDPT. */
75         if(addr_space == NULL) {
76             slab_cache_free(process);
77             return NULL;
78         }
79         process_init(process);
80     }
81     return process;
82 }
83
84 }

```

Here is the call graph for this function:



#### 4.76.1.3 process\_t\* process\_create\_initial ( void )

Definition at line 86 of file process.c.

References process\_t::addr\_space, initial\_addr\_space, memcpy(), NULL, and slab\_cache\_alloc().

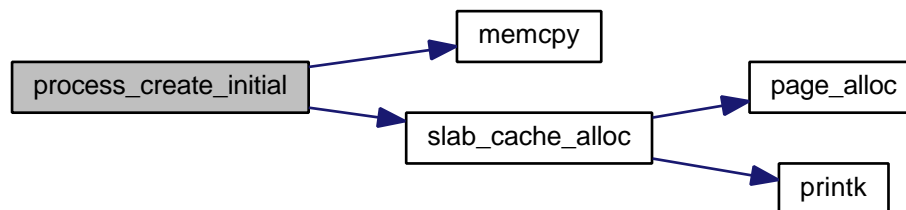
Referenced by kmain().

```

86     {
87     process_t *process = slab_cache_alloc(&process_cache);
88
89     if(process != NULL) {
90         memcpy(&process->addr_space, &initial_addr_space, sizeof(addr_space_t));
91         process_init(process);
92     }
93
94     return process;
95 }

```

Here is the call graph for this function:



#### 4.76.1.4 object\_ref\_t\* process\_get\_descriptor ( process\_t\* process, int fd )

Definition at line 97 of file process.c.

References process\_t::descriptors, NULL, and PROCESS\_MAX\_DESCRIPTOR.

Referenced by dispatch\_syscall(), ipc\_receive(), ipc\_send(), and process\_unused\_descriptor().

97

{

```
98     if(fd < 0 || fd > PROCESS_MAX_DESCRIPTOR) {
99         return NULL;
100     }
101
102     return &process->descriptors[fd];
103 }
```

#### 4.76.1.5 int process\_unused\_descriptor ( process\_t \* process )

Definition at line 105 of file process.c.

References process\_get\_descriptor(), and PROCESS\_MAX\_DESCRIPTOR.

Referenced by dispatch\_syscall().

```
105                                     {
106     int idx;
107
108     for(idx = 0; idx < PROCESS_MAX_DESCRIPTOR; ++idx) {
109         object_ref_t *ref = process_get_descriptor(process, idx);
110
111         if(! object_ref_is_valid(ref)) {
112             return idx;
113         }
114     }
115
116     return -1;
117 }
```

Here is the call graph for this function:

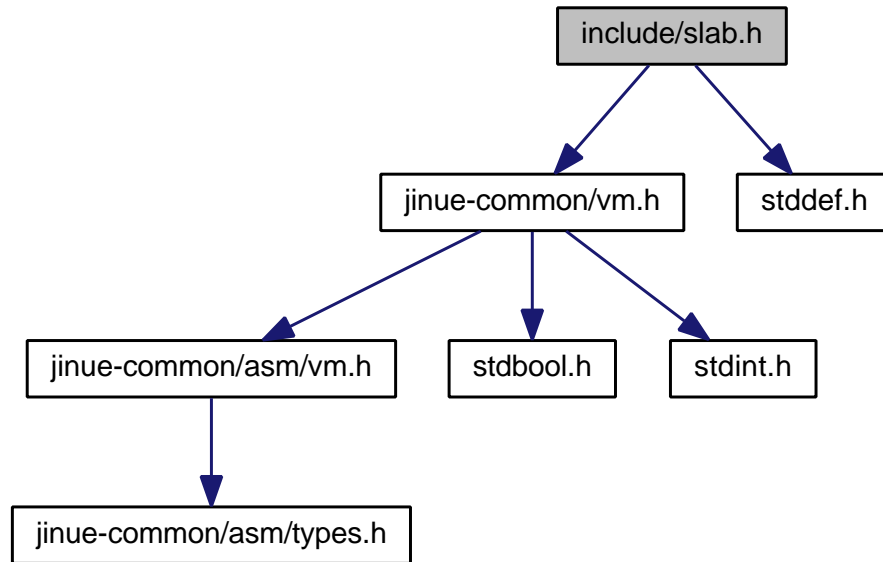


## 4.77 include/slab.h File Reference

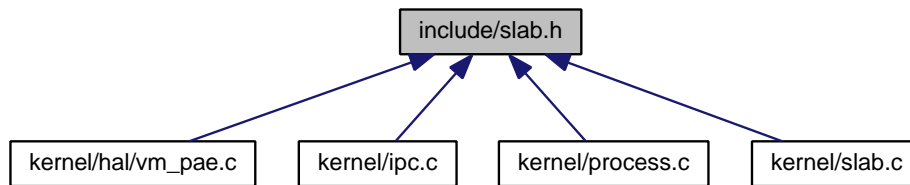
```
#include <jinue-common/vm.h>
#include <stddef.h>
```



Include dependency graph for slab.h:



This graph shows which files directly or indirectly include this file:



## Data Structures

- struct **slab\_cache\_t**
- struct **slab\_bufctl\_t**
- struct **slab\_t**

## Macros

- #define **SLAB\_SIZE** `PAGE_SIZE`
- #define **SLAB\_POISON\_ALIVE\_VALUE** `0x0BADCAFE`
- #define **SLAB\_POISON\_DEAD\_VALUE** `0xDEADBEEF`
- #define **SLAB\_RED\_ZONE\_VALUE** `0x5711600D`
- #define **SLAB\_DEFAULT\_WORKING\_SET** `2`
- #define **SLAB\_DEFAULTS** `(0)`
- #define **SLAB\_RED\_ZONE** `(1<<0)`
- #define **SLAB\_POISON** `(1<<1)`
- #define **SLAB\_HWCACHE\_ALIGN** `(1<<2)`
- #define **SLAB\_COMPACT** `(1<<3)`

## Typedefs

- typedef void(\* **slab\_ctor\_t**)(void \*, **size\_t**)
- typedef struct **slab\_cache\_t** **slab\_cache\_t**
- typedef struct **slab\_bufctl\_t** **slab\_bufctl\_t**
- typedef struct **slab\_t** **slab\_t**

## Functions

- void **slab\_cache\_init**(**slab\_cache\_t** \*cache, char \*name, **size\_t** size, **size\_t** alignment, **slab\_ctor\_t** ctor, **slab\_ctor\_t** dtor, int flags, **boot\_alloc\_t** \*boot\_alloc)  
*Initialize an object cache.*
- void \* **slab\_cache\_alloc**(**slab\_cache\_t** \*cache)  
*Allocate an object from the specified cache.*
- void **slab\_cache\_free**(void \*buffer)  
*Free an object.*
- void **slab\_cache\_reap**(**slab\_cache\_t** \*cache)  
*Return memory to the page allocator.*
- void **slab\_cache\_set\_working\_set**(**slab\_cache\_t** \*cache, unsigned int n)  
*Set a cache's working set.*

## Variables

- **slab\_cache\_t** \* **slab\_cache\_list**

### 4.77.1 Macro Definition Documentation

#### 4.77.1.1 #define SLAB\_COMPACT (1<<3)

Definition at line 57 of file slab.h.

Referenced by slab\_cache\_init().

#### 4.77.1.2 #define SLAB\_DEFAULT\_WORKING\_SET 2

Definition at line 46 of file slab.h.

Referenced by slab\_cache\_init().

#### 4.77.1.3 #define SLAB\_DEFAULTS (0)

Definition at line 49 of file slab.h.

Referenced by ipc\_boot\_init(), process\_boot\_init(), and vm\_pae\_create\_pdpt\_cache().

#### 4.77.1.4 #define SLAB\_HWCACHE\_ALIGN (1<<2)

Definition at line 55 of file slab.h.

Referenced by slab\_cache\_init().

#### 4.77.1.5 `#define SLAB_POISON (1<<1)`

Definition at line 53 of file slab.h.

Referenced by `slab_cache_alloc()`, `slab_cache_free()`, and `slab_cache_init()`.

#### 4.77.1.6 `#define SLAB_POISON_ALIVE_VALUE 0x0BADCAFE`

Definition at line 40 of file slab.h.

Referenced by `slab_cache_alloc()`.

#### 4.77.1.7 `#define SLAB_POISON_DEAD_VALUE 0xDEADBEEF`

Definition at line 42 of file slab.h.

Referenced by `slab_cache_alloc()`, and `slab_cache_free()`.

#### 4.77.1.8 `#define SLAB_RED_ZONE (1<<0)`

Definition at line 51 of file slab.h.

Referenced by `slab_cache_alloc()`, `slab_cache_free()`, and `slab_cache_init()`.

#### 4.77.1.9 `#define SLAB_RED_ZONE_VALUE 0x5711600D`

Definition at line 44 of file slab.h.

Referenced by `slab_cache_alloc()`, and `slab_cache_free()`.

#### 4.77.1.10 `#define SLAB_SIZE PAGE_SIZE`

Definition at line 38 of file slab.h.

Referenced by `slab_cache_free()`, and `slab_cache_init()`.

### 4.77.2 Typedef Documentation

#### 4.77.2.1 `typedef struct slab_bufctl_t slab_bufctl_t`

Definition at line 88 of file slab.h.

#### 4.77.2.2 `typedef struct slab_cache_t slab_cache_t`

Definition at line 82 of file slab.h.

#### 4.77.2.3 `typedef void(* slab_ctor_t)(void *, size_t)`

Definition at line 60 of file slab.h.

#### 4.77.2.4 typedef struct slab\_t slab\_t

Definition at line 101 of file slab.h.

### 4.77.3 Function Documentation

#### 4.77.3.1 void\* slab\_cache\_alloc ( slab\_cache\_t\* cache )

Allocate an object from the specified cache.

The cache must have been initialized with **slab\_cache\_init()** (p. 270). If no more space is available on existing slabs, this function tries to allocate a new slab using the kernel's page allocator (i.e. **page\_alloc()** (p. 251)). If page allocation fails, this function fails by returning NULL.

##### Parameters

<i>cache</i>	the cache from which to allocate an object
--------------	--

##### Returns

the address of the allocated object, or NULL if allocation failed

ASSERTION: now that slab\_cache\_grow() has run, we should have found at least one empty slab

Important note regarding the slab lists: The empty, partial and full slab lists are doubly-linked lists. This is done to allow the deletion of an arbitrary link given a pointer to it. We do not allow reverse traversal: we do not maintain a tail pointer and, more importantly: we do *NOT* maintain the previous pointer of the first link in the list (i.e. it is garbage data, not NULL).

ASSERTION: there is at least one buffer on the free list

ASSERT: the slab is the head of the partial list

Definition at line 232 of file slab.c.

References assert, slab\_cache\_t::bufctl\_offset, slab\_cache\_t::ctor, slab\_cache\_t::empty\_count, slab\_cache\_t::flags, slab\_t::free\_list, slab\_cache\_t::name, slab\_bufctl\_t::next, slab\_t::next, NULL, slab\_t::obj\_count, slab\_cache\_t::obj\_size, page\_alloc(), slab\_t::prev, printk(), SLAB\_POISON, SLAB\_POISON\_ALIVE\_VALUE, SLAB\_POISON\_DEAD\_VALUE, SLAB\_RED\_ZONE, SLAB\_RED\_ZONE\_VALUE, slab\_cache\_t::slabs\_empty, slab\_cache\_t::slabs\_full, and slab\_cache\_t::slabs\_partial.

Referenced by ipc\_boot\_init(), ipc\_object\_create(), process\_create(), process\_create\_initial(), and vm\_pae\_create\_addr\_space().

```

232                                     {
233     slab_t          *slab;
234
235     if(cache->slabs_partial != NULL) {
236         slab = cache->slabs_partial;
237     }
238     else {
239         if(cache->slabs_empty == NULL) {
240             void *slab_addr = page_alloc();
241
242             if(slab_addr == NULL) {
243                 return NULL;
244             }
245
246             init_and_add_slab(cache, slab_addr);
247         }
248
249         slab = cache->slabs_empty;
250
251         assert(slab != NULL);

```

```

253
263     /* We are about to allocate one object from this slab, so it will
264      * not be empty anymore...*/
265     cache->slabs_empty      = slab->next;
266
267     --(cache->empty_count);
268
269     slab->next              = cache->slabs_partial;
270     if(slab->next != NULL) {
271         slab->next->prev = slab;
272     }
273     cache->slabs_partial    = slab;
274 }
275
276 slab_bufctl_t *bufctl = slab->free_list;
277
279 assert(bufctl != NULL);
280
281 slab->free_list = bufctl->next;
282 slab->obj_count += 1;
283
284 /* If we just allocated the last buffer, move the slab to the full
285  * list */
286 if(slab->free_list == NULL) {
287     /* remove from the partial slabs list */
288
289     assert(cache->slabs_partial == slab);
290
291     cache->slabs_partial = slab->next;
292
293     if(slab->next != NULL) {
294         slab->next->prev = slab->prev;
295     }
296
297     /* add to the full slabs list */
298     slab->next      = cache->slabs_full;
299     cache->slabs_full = slab;
300
301     if(slab->next != NULL) {
302         slab->next->prev = slab;
303     }
304 }
305
306 uint32_t *buffer = (uint32_t *) ( (char *)bufctl - cache->bufctl_offset );
307
308 if(cache->flags & SLAB_POISON) {
309     unsigned int idx;
310     unsigned int dump_lines = 0;
311
312     for(idx = 0; idx < cache->obj_size / sizeof(uint32_t); ++idx) {
313         if(buffer[idx] != SLAB_POISON_DEAD_VALUE) {
314             if(dump_lines == 0) {
315                 printk("detected write to freed object, cache: %s buffer: 0x%x:\n",
316                     cache->name,
317                     (unsigned int)buffer
318                 );
319             }
320
321             if(dump_lines < 4) {
322                 printk(" value 0x%x at byte offset %u\n", buffer[idx], idx * sizeof(
323 uint32_t));
324             }
325
326             ++dump_lines;
327         }
328
329         buffer[idx] = SLAB_POISON_ALIVE_VALUE;
330     }
331
332     /* If both SLAB_POISON and SLAB_RED_ZONE are enabled, we perform
333      * redzone checking even on freed objects. */
334     if(cache->flags & SLAB_RED_ZONE) {
335         if(buffer[idx] != SLAB_RED_ZONE_VALUE) {
336             printk("detected write past the end of freed object, cache: %s buffer: 0x%x value: 0x%x\n",
337                 cache->name,
338                 (unsigned int)buffer,
339                 buffer[idx]
340             );
341         }
342
343         buffer[idx] = SLAB_RED_ZONE_VALUE;

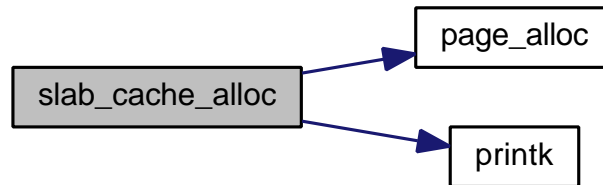
```

```

344     }
345
346     if(cache->ctor != NULL) {
347         cache->ctor((void *)buffer, cache->obj_size);
348     }
349 }
350 else if(cache->flags & SLAB_RED_ZONE) {
351     buffer[cache->obj_size / sizeof(uint32_t)] = SLAB_RED_ZONE_VALUE;
352 }
353
354 return (void *)buffer;
355 }

```

Here is the call graph for this function:



#### 4.77.3.2 void slab\_cache\_free ( void \* buffer )

Free an object.

##### Parameters

<i>buffer</i>	the object to free
---------------	--------------------

Definition at line 363 of file slab.c.

References `ALIGN_START_PTR`, `slab_cache_t::bufctl_offset`, `slab_t::cache`, `slab_cache_t::dtor`, `slab_cache_t::empty_count`, `slab_cache_t::flags`, `slab_t::free_list`, `slab_cache_t::name`, `slab_bufctl_t::next`, `slab_t::next`, `NULL`, `slab_t::obj_count`, `slab_cache_t::obj_size`, `slab_t::prev`, `printk()`, `SLAB_POISON`, `SLAB_POISON_DEAD_VALUE`, `SLAB_RED_ZONE`, `SLAB_RED_ZONE_VALUE`, `SLAB_SIZE`, `slab_cache_t::slabs_empty`, `slab_cache_t::slabs_full`, and `slab_cache_t::slabs_partial`.

Referenced by `process_create()`, and `vm_pae_destroy_addr_space()`.

```

363     {
364         /* compute address of slab data structure */
365         addr_t slab_start = ALIGN_START_PTR(buffer, SLAB_SIZE);
366         slab_t *slab      = (slab_t *) (slab_start + SLAB_SIZE - sizeof(slab_t) );
367
368         /* obtain address of cache and bufctl */
369         slab_cache_t *cache = slab->cache;
370         slab_bufctl_t *bufctl = (slab_bufctl_t *) ((char *)buffer + cache->
371         bufctl_offset);
372
373         /* If slab is on the full slabs list, move it to the partial list
374          * since we are about to return a buffer to it. */
375         if(slab->free_list == NULL) {
376             /* remove from full slabs list */
377             if(cache->slabs_full == slab) {
378                 cache->slabs_full = slab->next;
379             }
380             else {
381                 slab->prev->next = slab->next;
382             }
383             if(slab->next != NULL) {
384                 slab->next->prev = slab->prev;
385             }
386

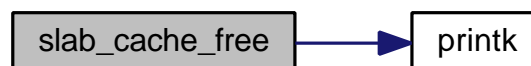
```

```

387     /* add to partial slabs list */
388     slab->next      = cache->slabs_partial;
389     cache->slabs_partial = slab;
390
391     if(slab->next != NULL) {
392         slab->next->prev = slab;
393     }
394 }
395
396 if(cache->flags & SLAB_RED_ZONE) {
397     uint32_t *rz_word = (uint32_t *)((char *)buffer + cache->obj_size);
398
399     if(*rz_word != SLAB_RED_ZONE_VALUE) {
400         printk("detected write past the end of object, cache: %s buffer: 0x%x value: 0x%x\n",
401             cache->name,
402             (unsigned int)buffer,
403             *rz_word
404         );
405     }
406
407     *rz_word = SLAB_RED_ZONE_VALUE;
408 }
409
410 if(cache->flags & SLAB_POISON) {
411     unsigned int idx;
412
413     if(cache->dtor != NULL) {
414         cache->dtor(buffer, cache->obj_size);
415     }
416
417     uint32_t *buffer32 = (uint32_t *)buffer;
418
419     for(idx = 0; idx < cache->obj_size / sizeof(uint32_t); ++idx) {
420         buffer32[idx] = SLAB_POISON_DEAD_VALUE;
421     }
422 }
423
424 /* link buffer into slab free list */
425 bufctl->next      = slab->free_list;
426 slab->free_list    = bufctl;
427 slab->obj_count    -= 1;
428
429 /* If we just returned the last object to the slab, move the slab to
430  * the empty list. */
431 if(slab->obj_count == 0) {
432     /* remove from partial slabs list */
433     if(cache->slabs_partial == slab) {
434         cache->slabs_partial = slab->next;
435     }
436     else {
437         slab->prev->next = slab->next;
438     }
439
440     if(slab->next != NULL) {
441         slab->next->prev = slab->prev;
442     }
443
444     /* add to empty slabs list */
445     slab->next      = cache->slabs_empty;
446     cache->slabs_empty = slab;
447
448     if(slab->next != NULL) {
449         slab->next->prev = slab;
450     }
451
452     ++(cache->empty_count);
453 }
454 }

```

Here is the call graph for this function:



4.77.3.3 `void slab_cache_init ( slab_cache_t * cache, char * name, size_t size, size_t alignment, slab_ctor_t ctor, slab_dtor_t dtor, int flags, boot_alloc_t * boot_alloc )`

Initialize an object cache.

The following flags are supported:

- `SLAB_HWCACHE_ALIGN` Align objects on at least the line size of the CPU's data cache.
- `SLAB_COMPACT` the bufctl can safely be put inside the object without destroying the constructed state. If not set, additional space is reserved specifically for the bufctl to prevent corruption of the constructed state.
- `SLAB_RED_ZONE` (redzone checking - debugging) Add a guard word at the end of each object and use this to detect writes past the end of the object.
- `SLAB_POISON` (debugging) Fill uninitialized objects with a recognizable pattern before calling the constructor function to help identify members that do not get initialized. Do the same when freeing objects and use this to detect writes to freed objects.

This function uses the kernel's boot-time page allocator to allocate an initial slab. This helps with bootstrapping because it allows a few objects (up to a slab's worth) to be allocated before the main page allocator has been initialized and then replenished by user space. It also means this function can only be called during kernel initialization (it would not make sense to call it later).

#### Parameters

<i>cache</i>	the cache to initialize
<i>name</i>	a human-readable name for the cache, used in debugging messages
<i>size</i>	the size of objects allocated on this cache
<i>alignment</i>	the minimum object alignment, or zero for no constraint
<i>ctor</i>	the object constructor function
<i>dtor</i>	the object destructor function
<i>flags</i>	see description
<i>boot_alloc</i>	the kernel boot-time page allocator structure

ASSERTION: buffer size is at least the size of a pointer

ASSERTION: name is not NULL string

Definition at line 137 of file slab.c.

References `ALIGN_END`, `slab_cache_t::alignment`, `slab_cache_t::alloc_size`, `assert`, `boot_page_alloc()`, `slab_cache_t::bufctl_offset`, `cpu_info`, `slab_cache_t::ctor`, `cpu_info::dcache_alignment`, `slab_cache_t::dtor`, `slab_cache_t::empty_count`, `slab_cache_t::flags`, `slab_cache_t::max_colour`, `slab_cache_t::name`, `slab_cache_t::next_colour`, `NULL`, `slab_cache_t::obj_size`, `SLAB_COMPACT`, `SLAB_DEFAULT_WORKING_SET`, `SLAB_HWCACHE_ALIGN`, `SLAB_POISON`, `SLAB_RED_ZONE`, `SLAB_SIZE`, `slab_cache_t::slabs_empty`, `slab_cache_t::slabs_full`, `slab_cache_t::slabs_partial`, and `slab_cache_t::working_set`.

Referenced by `ipc_boot_init()`, `process_boot_init()`, and `vm_pae_create_pdpt_cache()`.

```

145                                     {
146
148     assert(size >= sizeof(void *));
149
151     assert(name != NULL);
152
153     cache->name           = name;
154     cache->ctor            = ctor;
155     cache->dtor            = dtor;
156     cache->slabs_empty     = NULL;
157     cache->slabs_partial   = NULL;
158     cache->slabs_full      = NULL;
159     cache->empty_count     = 0;

```

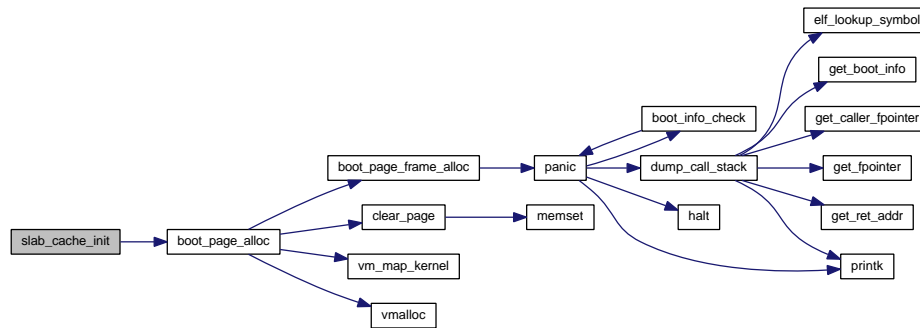


```

160     cache->flags          = flags;
161     cache->next_colour    = 0;
162     cache->working_set    = SLAB_DEFAULT_WORKING_SET;
163
164     /* Compute actual alignment. */
165     if(alignment == 0) {
166         cache->alignment = sizeof(uint32_t);
167     }
168     else {
169         cache->alignment = alignment;
170     }
171
172     if((flags & SLAB_HWCACHE_ALIGN) && cache->alignment < cpu_info.
dcache_alignment) {
173         cache->alignment = cpu_info.dcache_alignment;
174     }
175
176     cache->alignment = ALIGN_END(cache->alignment, sizeof(uint32_t));
177
178     /* Reserve space for bufctl and/or redzone word. */
179     cache->obj_size = ALIGN_END(size, sizeof(uint32_t));
180
181     if((flags & SLAB_POISON) && (flags & SLAB_RED_ZONE)) {
182         /* bufctl and redzone word appended to buffer */
183         cache->alloc_size = cache->obj_size + sizeof(uint32_t) + sizeof(
slab_bufctl_t);
184     }
185     else if((flags & SLAB_POISON) || (flags & SLAB_RED_ZONE)) {
186         /* bufctl or redzone word appended to buffer (can be shared) */
187         cache->alloc_size = cache->obj_size + sizeof(uint32_t);
188     }
189     else if(ctor != NULL && ! (flags & SLAB_COMPACT)) {
190         /* If a constructor is defined, we cannot put the bufctl inside
191          * the object because that could overwrite constructed state,
192          * unless client explicitly says it's ok (SLAB_COMPACT flag). */
193         cache->alloc_size = cache->obj_size + sizeof(slab_bufctl_t);
194     }
195     else {
196         cache->alloc_size = cache->obj_size;
197     }
198
199     if(cache->alloc_size % cache->alignment != 0) {
200         cache->alloc_size += cache->alignment - cache->alloc_size % cache->
alignment;
201     }
202
203     size_t avail_space = SLAB_SIZE - sizeof(slab_t);
204
205     unsigned int buffers_per_slab = avail_space / cache->alloc_size;
206
207     size_t wasted_space = avail_space - buffers_per_slab * cache->alloc_size;
208
209     cache->max_colour = (wasted_space / cache->alignment) * cache->alignment;
210
211     cache->bufctl_offset = cache->alloc_size - sizeof(slab_bufctl_t);
212
213     /* Allocate first slab.
214      *
215      * This is needed to allow a few objects to be allocated during kernel
216      * initialization. */
217     init_and_add_slab(cache, boot_page_alloc(boot_alloc));
218 }

```

Here is the call graph for this function:



#### 4.77.3.4 void slab\_cache\_reap ( slab\_cache\_t\* cache )

Return memory to the page allocator.

Free slabs in excess to the cache's working set are finalized and freed.

##### Parameters

<i>cache</i>	the cache from which to reclaim memory
--------------	--

Definition at line 544 of file slab.c.

References slab\_cache\_t::empty\_count, slab\_t::next, slab\_cache\_t::slabs\_empty, and slab\_cache\_t::working\_set.

```

544
545     while(cache->empty_count > cache->working_set) {
546         /* select the first empty slab */
547         slab_t *slab = cache->slabs_empty;
548
549         /* unlink it and update count */
550         cache->slabs_empty = slab->next;
551         cache->empty_count -= 1;
552
553         /* destroy slab */
554         destroy_slab(cache, slab);
555     }
556 }

```

#### 4.77.3.5 void slab\_cache\_set\_working\_set ( slab\_cache\_t\* cache, unsigned int n )

Set a cache's working set.

The working set is defined as the number of free slabs the cache keeps for itself when pages are reclaimed from it. (This is terminology used in the Bonwick paper.) This provides some hysteresis to prevent slabs from being continuously created and destroyed, which requires calling the constructor and destructor functions on individual objects on the slabs.

##### Parameters

<i>cache</i>	the cache for which to set the working set
<i>n</i>	the size of the working set (number of pages)

Definition at line 571 of file slab.c.

References slab\_cache\_t::working\_set.

```

571                                     {
572     cache->working_set = n;
573 }

```

#### 4.77.4 Variable Documentation

##### 4.77.4.1 slab\_cache\_t\* slab\_cache\_list

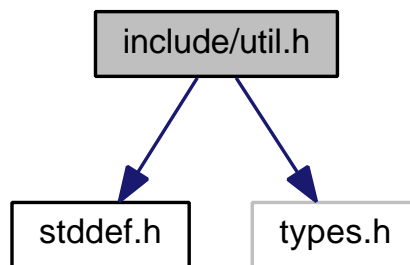
## 4.78 include/util.h File Reference

```

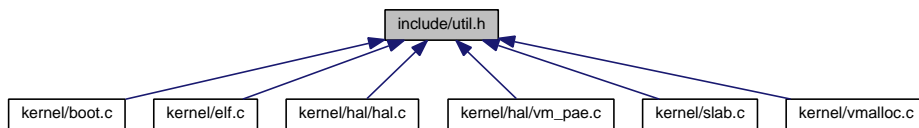
#include <stddef.h>
#include <types.h>

```

Include dependency graph for util.h:



This graph shows which files directly or indirectly include this file:



## Macros

- **#define ALIGN\_START(x, s)** ( (x) & ~((s)-1) )
- **#define ALIGN\_END(x, s)** ( ALIGN\_START((x) + s - 1, (s)) )
- **#define OFFSET\_OF\_PTR(x, s)** ( (uintptr\_t)(x) & ((s)-1) )
- **#define ALIGN\_START\_PTR(x, s)** ( (void \*)ALIGN\_START((uintptr\_t)(x), (s)) )
- **#define ALIGN\_END\_PTR(x, s)** ( (void \*)ALIGN\_END((uintptr\_t)(x), (s)) )
- **#define alloc\_forward(T, p)** ((T \*)alloc\_forward\_func(sizeof(T), &(p)))
- **#define alloc\_backward(T, p)** ((T \*)alloc\_forward\_func(sizeof(T), &(p)))

#### 4.78.1 Macro Definition Documentation

##### 4.78.1.1 #define ALIGN\_END( x, s ) ( ALIGN\_START((x) + s - 1, (s)) )

Definition at line 40 of file util.h.

Referenced by slab\_cache\_init().

4.78.1.2 `#define ALIGN_END_PTR( x, s ) ( (void *)ALIGN_END((uintptr_t)(x), (s)) )`

Definition at line 46 of file util.h.

Referenced by `boot_heap_alloc_size()`, and `elf_load()`.

4.78.1.3 `#define ALIGN_START( x, s ) ( (x) & ~((s)-1) )`

Definition at line 38 of file util.h.

4.78.1.4 `#define ALIGN_START_PTR( x, s ) ( (void *)ALIGN_START((uintptr_t)(x), (s)) )`

Definition at line 44 of file util.h.

Referenced by `elf_load()`, and `slab_cache_free()`.

4.78.1.5 `#define alloc_backward( T, p ) ((T *)alloc_forward_func(sizeof(T), &(p)))`

Definition at line 67 of file util.h.

4.78.1.6 `#define alloc_forward( T, p ) ((T *)alloc_forward_func(sizeof(T), &(p)))`

Definition at line 65 of file util.h.

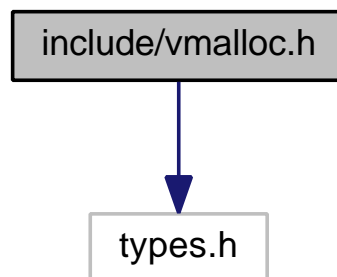
4.78.1.7 `#define OFFSET_OF_PTR( x, s ) ((uintptr_t)(x) & ((s)-1) )`

Definition at line 42 of file util.h.

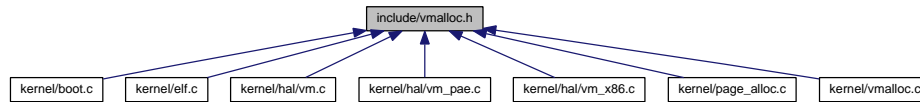
## 4.79 include/vmalloc.h File Reference

`#include <types.h>`

Include dependency graph for vmalloc.h:



This graph shows which files directly or indirectly include this file:



## Functions

- **addr\_t vmalloc** (void)  
*Allocate a page of virtual address space.*
- void **vmfree** (addr\_t page)  
*Free a page of virtual address space.*
- void **vmalloc\_init** (addr\_t start\_addr, addr\_t end\_addr, addr\_t preinit\_limit, boot\_alloc\_t \*boot\_alloc)  
*Basic initialization of the virtual memory allocator.*
- bool **vmalloc\_is\_in\_range** (addr\_t page)  
*Check whether the specified page is in the region managed by the allocator.*

### 4.79.1 Function Documentation

#### 4.79.1.1 addr\_t vmalloc ( void )

Allocate a page of virtual address space.

##### Returns

address of allocated page or NULL if allocation failed

Definition at line 150 of file vmalloc.c.

Referenced by add\_page\_frame(), boot\_page\_alloc(), elf\_setup\_stack(), vm\_clone\_page\_directory(), vm\_destroy\_page\_directory(), vm\_pae\_lookup\_page\_directory(), and vm\_x86\_lookup\_page\_directory().

```

150     {
151         return vmalloc_from(&kernel_vmallocator);
152     }

```

#### 4.79.1.2 void vmalloc\_init ( addr\_t start\_addr, addr\_t end\_addr, addr\_t preinit\_limit, boot\_alloc\_t \* boot\_alloc )

Basic initialization of the virtual memory allocator.

This function initializes the allocator structure, and then initializes the first few blocks up to the limit set by the preinit\_limit argument (more precisely, up to and including the block that contains preinit\_limit - 1). TODO mention how to initialize the rest once this is implemented.

##### Parameters

<i>start_addr</i>	the start address of the region managed by the allocator
-------------------	--

<i>end_addr</i>	the end address of the region managed by the allocator
<i>preinit_limit</i>	the limit address for preinitialized blocks
<i>boot_alloc</i>	the initialization-time page allocator structure

Definition at line 178 of file vmalloc.c.

Referenced by `vm_boot_postinit()`.

```

182                                     {
183
184     init_allocator(
185         &kernel_vmallocator,
186         start_addr,
187         end_addr,
188         preinit_limit,
189         boot_alloc);
190 }
```

#### 4.79.1.3 `bool vmalloc_is_in_range ( addr_t page )`

Check whether the specified page is in the region managed by the allocator.

##### Parameters

<i>page</i>	the address of the page
-------------	-------------------------

##### Returns

true if it is in the region, false otherwise

Definition at line 199 of file vmalloc.c.

Referenced by `remove_page_frame()`.

```

199                                     {
200     return addr_is_in_initialized_range(&kernel_vmallocator, page);
201 }
```

#### 4.79.1.4 `void vmfree ( addr_t page )`

Free a page of virtual address space.

##### Parameters

<i>page</i>	the address of the page to free
-------------	---------------------------------

Definition at line 160 of file vmalloc.c.

Referenced by `elf_setup_stack()`, and `remove_page_frame()`.

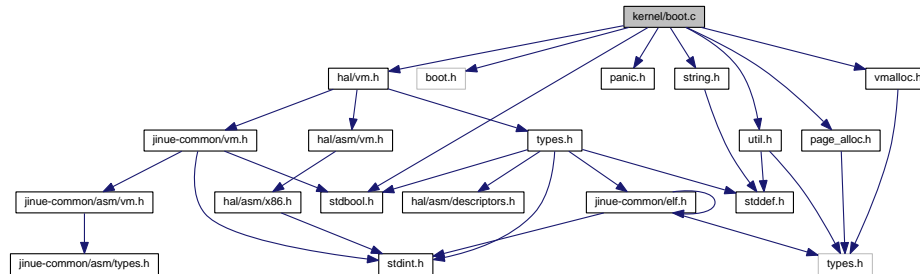
```

160                                     {
161     vmfree_to(&kernel_vmallocator, page);
162 }
```

## 4.80 `kernel/boot.c` File Reference

```
#include <hal/vm.h>
```

Include dependency graph for boot.c:



- void **boot\_alloc\_init**(**boot\_alloc\_t** \*boot\_alloc, void \*heap\_ptr)  
*Initialize the boot allocator.*
- void \* **boot\_heap\_alloc\_size**(**boot\_alloc\_t** \*boot\_alloc, **size\_t** size, **size\_t** align)  
*Allocate an object on the boot heap.*
- void **boot\_heap\_push**(**boot\_alloc\_t** \*boot\_alloc)  
*Push the current state of the boot allocator heap.*
- void **boot\_heap\_pop**(**boot\_alloc\_t** \*boot\_alloc)  
*Pop the last pushed boot allocator heap.*
- **addr\_t boot\_page\_alloc\_early**(**boot\_alloc\_t** \*boot\_alloc)  
*Early page allocation.*
- **kern\_paddr\_t boot\_page\_frame\_alloc**(**boot\_alloc\_t** \*boot\_alloc)  
*Allocate a page frame, that is, a page of physical memory.*
- **addr\_t boot\_vmalloc**(**boot\_alloc\_t** \*boot\_alloc)  
*Allocate a page of address space.*
- **addr\_t boot\_page\_alloc**(**boot\_alloc\_t** \*boot\_alloc)  
*Allocate a page in the allocations region of the kernel address space.*
- **addr\_t boot\_page\_alloc\_image**(**boot\_alloc\_t** \*boot\_alloc)  
*Allocate a page in the image region of the kernel address space.*

**4.80.1.1** void boot\_alloc\_init ( boot\_alloc t\* *boot\_alloc*, void \* *heap\_ptr* )

The boot allocator is used for heap and page allocation during kernel initialization. After this function is called, the boot heap is ready to use (see **boot\_heap\_alloc()** (p. 66)). However, the page and page frame allocators require additional initialization by the machine-dependent code before they can be used.

## Parameters

<i>boot_alloc</i>	the allocator state initialized by this function
<i>heap_ptr</i>	the current top of the boot heap

Definition at line 54 of file boot.c.

References `boot_alloc_t::heap_ptr`, `boot_alloc_t::its_early`, and `memset()`.

Referenced by `kmain()`.

```

54                                     {
55     memset(boot_alloc, 0, sizeof(boot_alloc_t));
56     boot_alloc->heap_ptr = heap_ptr;
57     boot_alloc->its_early = true;
58     /* TODO handle heap limit. */
59 }
```

Here is the call graph for this function:



#### 4.80.1.2 void\* boot\_heap\_alloc\_size ( boot\_alloc\_t\* boot\_alloc, size\_t size, size\_t align )

Allocate an object on the boot heap.

Callers do not call this function directly but instead use the **boot\_heap\_alloc()** (p. 66) macro that takes a type as the second argument instead of an object size.

## Parameters

<i>boot_alloc</i>	the boot allocator state
<i>size</i>	the size of the object to allocate, in bytes
<i>align</i>	the required start address alignment of the object, zero for no constraint

## Returns

the allocated object

Definition at line 73 of file boot.c.

References `ALIGN_END_PTR`, and `boot_alloc_t::heap_ptr`.

```

73                                     {
74     if(align != 0) {
75         boot_alloc->heap_ptr = ALIGN_END_PTR(boot_alloc->heap_ptr, align);
76     }
77
78     void *object = boot_alloc->heap_ptr;
79     boot_alloc->heap_ptr = (char *)boot_alloc->heap_ptr + size;
80
81     return object;
82 }
```

#### 4.80.1.3 void boot\_heap\_pop ( boot\_alloc\_t\* boot\_alloc )

Pop the last pushed boot allocator heap.

This function frees all heap allocations performed since the matching call to **boot\_heap\_push()** (p. 68).



## Parameters

<i>boot_alloc</i>	the boot allocator state
-------------------	--------------------------

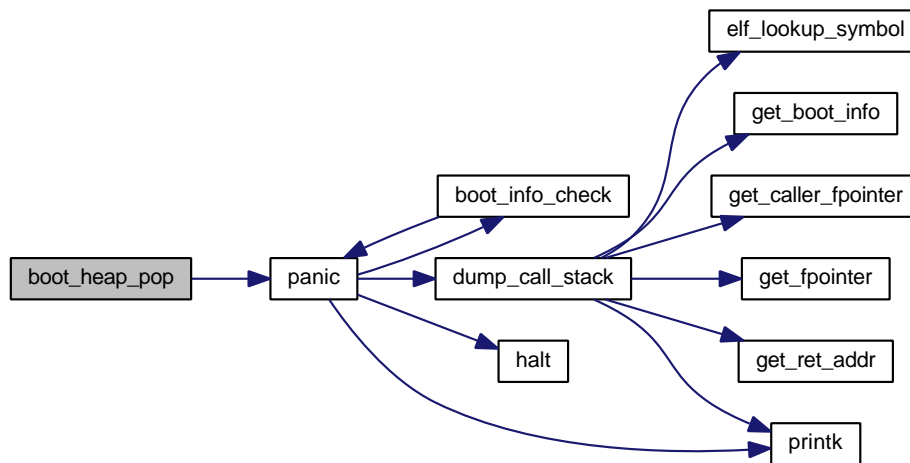
Definition at line 112 of file boot.c.

References `boot_alloc_t::heap_ptr`, `boot_alloc_t::heap_pushed_state`, `boot_heap_pushed_state::next`, `NULL`, and `panic()`.

```

112
113     if(boot_alloc->heap_pushed_state == NULL) {
114         panic("No more boot heap pushed state to pop.");
115     }
116
117     boot_alloc->heap_ptr          = boot_alloc->heap_pushed_state;
118     boot_alloc->heap_pushed_state = boot_alloc->heap_pushed_state->next;
119 }
```

Here is the call graph for this function:



#### 4.80.1.4 void boot\_heap\_push ( boot\_alloc\_t \* boot\_alloc )

Push the current state of the boot allocator heap.

All heap allocations performed after calling this function are freed by the matching call to **`boot_heap_pop()`** (p. 67). This function can be called multiple times before calling **`boot_heap_pop()`** (p. 67). Heap states pushed by this function are popped by **`boot_heap_pop()`** (p. 67)() in the reverse order they were pushed.

## Parameters

<i>boot_alloc</i>	the boot allocator state
-------------------	--------------------------

Definition at line 95 of file boot.c.

References `boot_heap_alloc`, `boot_alloc_t::heap_pushed_state`, and `boot_heap_pushed_state::next`.

```

95
96     struct boot_heap_pushed_state *pushed_state =
97         boot_heap_alloc(boot_alloc, struct boot_heap_pushed_state, 0);
98
99     pushed_state->next          = boot_alloc->heap_pushed_state;
100     boot_alloc->heap_pushed_state = pushed_state;
101 }
```

#### 4.80.1.5 `addr_t boot_page_alloc ( boot_alloc_t * boot_alloc )`

Allocate a page in the allocations region of the kernel address space.

The physical memory is allocated just after the kernel image and other initialization-time allocations by calling **boot\_page\_frame\_alloc()** (p. 73) whereas the address space page is allocated in the allocations region by calling **vmalloc()** (p. 275).

If either of these two conditions is met, you must use `boot_pgalloc_image()` instead of this function: 1) The address space page allocator has not yet been initialized by calling **vmalloc\_init()** (p. 275); or 2) It is necessary to allocate multiple contiguous pages.

##### Parameters

<code>boot_alloc</code>	the boot allocator state
-------------------------	--------------------------

##### Returns

address of allocated page

Definition at line 282 of file `boot.c`.

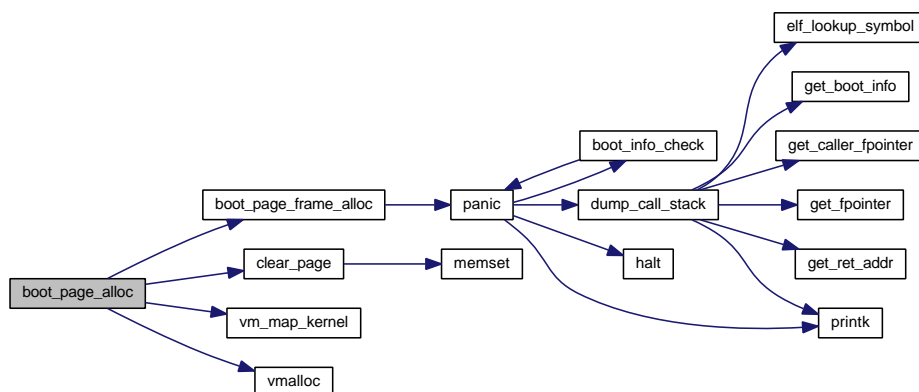
References `boot_page_frame_alloc()`, `clear_page()`, `VM_FLAG_READ_WRITE`, `vm_map_kernel()`, and `vmalloc()`.

Referenced by `hal_init()`, `slab_cache_init()`, and `thread_create_boot()`.

```

282     {
283     kern_paddr_t paddr  = boot_page_frame_alloc(boot_alloc);
284     addr_t vaddr       = vmalloc();
285
286     vm_map_kernel(vaddr, paddr, VM_FLAG_READ_WRITE);
287
288     /* This newly allocated page may have data left from a previous boot which
289      * may contain sensitive information. Let's clear it. */
290     clear_page(vaddr);
291
292     return vaddr;
293 }
```

Here is the call graph for this function:



#### 4.80.1.6 `addr_t boot_page_alloc_early ( boot_alloc_t * boot_alloc )`

Early page allocation.

When the kernel is first entered, the setup code has set up temporary page tables that map a contiguous region of physical memory (RAM) that contains the kernel image at KLIMIT. The setup code itself allocates a few pages, notably for the temporary page tables and for the boot stack and heap. These pages are allocated sequentially just after the kernel image.

This function allocates pages sequentially following the kernel image and the setup code allocations. It is meant to be called early in the initialization process, while the temporary page tables set up by the setup code are still being used, which means before the kernel switches to the initial address space it sets up.

Because the page tables set up by the setup code are being used, there is a fixed relation between the virtual address of the pages allocated by this function and the physical address of the underlying page frames. This relation is expressed by the **EARLY\_PTR\_TO\_PHYS\_ADDR()** (p. 142) macro.

This function must not be called once the kernel has switched away from the page tables set up by the setup code to the initial address space it has set up itself. This function checks for this and triggers a kernel panic if it happens.

#### Parameters

<i>boot_alloc</i>	the boot allocator state
-------------------	--------------------------

#### Returns

address of allocated page

Definition at line 150 of file boot.c.

References `clear_page()`, `EARLY_PTR_TO_PHYS_ADDR`, `boot_alloc_t::its_early`, `boot_alloc_t::kernel_paddr_limit`, `boot_alloc_t::kernel_paddr_top`, `boot_alloc_t::kernel_vm_limit`, `boot_alloc_t::kernel_vm_top`, `NULL`, `PAGE_MASK`, `PAGE_SIZE`, and `panic()`.

Referenced by `vm_init_initial_page_directory()`, `vm_pae_create_initial_addr_space()`, and `vm_x86_create_initial_addr_space()`.

```

150                                     {
151     /* Preconditions */
152     if(! boot_alloc->its_early) {
153         panic("boot_pgalloc_early() called too late");
154     }
155
156     if(boot_alloc->kernel_vm_top == NULL) {
157         panic("boot_pgalloc_early(): allocator is uninitialized");
158     }
159
160     if(((uintptr_t)boot_alloc->kernel_vm_top & PAGE_MASK) != 0) {
161         panic("boot_pgalloc_early(): bad kernel region top VM address alignment");
162     }
163
164     if(boot_alloc->kernel_paddr_top != EARLY_PTR_TO_PHYS_ADDR(boot_alloc->
kernel_vm_top)) {
165         panic("boot_pgalloc_early(): inconsistent allocator state");
166     }
167
168     /* address of allocated page */
169     addr_t allocated_page = boot_alloc->kernel_vm_top;
170
171     /* Update allocator state.
172     *
173     * In this early allocator function that is called while the temporary page
174     * tables set up by the setup code are still being used, there is a fixed
175     * relationship between virtual and physical addresses. */
176     boot_alloc->kernel_vm_top      = allocated_page + PAGE_SIZE;
177     boot_alloc->kernel_paddr_top  = boot_alloc->kernel_paddr_top + PAGE_SIZE;
178
179     /* Check updated state against allocation limits. */
180     if(boot_alloc->kernel_vm_top > boot_alloc->kernel_vm_limit) {
181         panic("vmalloc_early(): kernel address space exhausted");
182     }
183
184     if(boot_alloc->kernel_paddr_top > boot_alloc->kernel_paddr_limit) {
185         panic("vmalloc_early(): available memory exhausted");

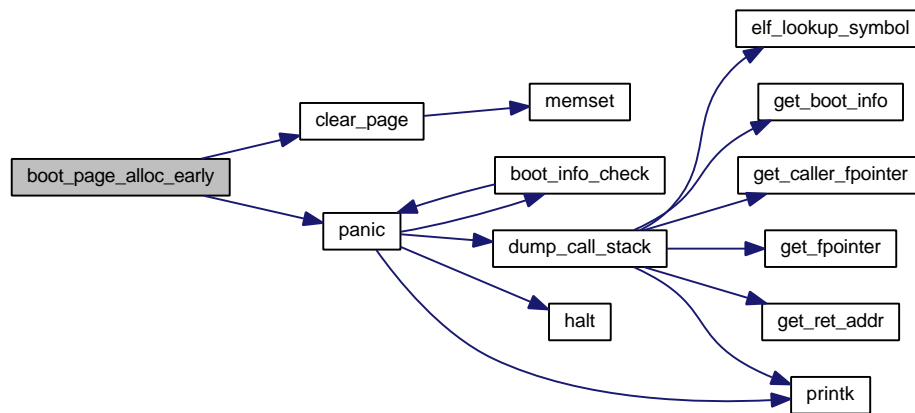
```

```

186     }
187
188     /* This newly allocated page may have data left from a previous boot which
189      * may contain sensitive information. Let's clear it. */
190     clear_page(allocated_page);
191
192     /* Post-condition */
193     if (boot_alloc->kernel_paddr_top != EARLY_PTR_TO_PHYS_ADDR(boot_alloc->
194         kernel_vm_top)) {
195         panic("boot_pgalloc_early(): inconsistent allocator state on return");
196     }
197     return allocated_page;
198 }

```

Here is the call graph for this function:



#### 4.80.1.7 `addr_t boot_page_alloc_image ( boot_alloc_t * boot_alloc )`

Allocate a page in the image region of the kernel address space.

Since the size of the image region is limited, use `boot_pgalloc()` instead of this function whenever possible.

The difference between this function and `boot_pgalloc()` is that the address space page is allocated by this function using `boot_vmalloc()` (p.74) instead of `vmalloc()` (p.275). Pages allocated by subsequent calls to this function are allocated sequentially.

##### Parameters

<code>boot_alloc</code>	the boot allocator state
-------------------------	--------------------------

##### Returns

address of allocated page

Definition at line 310 of file `boot.c`.

References `boot_page_frame_alloc()`, `boot_vmalloc()`, `clear_page()`, `VM_FLAG_READ_WRITE`, and `vm_map_kernel()`.

Referenced by `hal_init()`.

```

310     {
311         kern_paddr_t paddr = boot_page_frame_alloc(boot_alloc);
312         addr_t vaddr      = boot_vmalloc(boot_alloc);
313     }

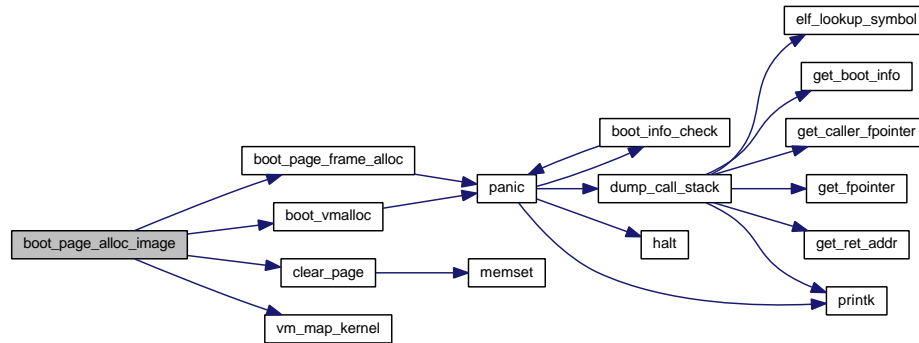
```

```

314     vm_map_kernel(vaddr, paddr, VM_FLAG_READ_WRITE);
315
316     /* This newly allocated page may have data left from a previous boot which
317      * may contain sensitive information. Let's clear it. */
318     clear_page(vaddr);
319
320     return vaddr;
321 }

```

Here is the call graph for this function:



#### 4.80.1.8 kern\_paddr\_t boot\_page\_frame\_alloc ( boot\_alloc\_t \* boot\_alloc )

Allocate a page frame, that is, a page of physical memory.

The allocated page frame is not mapped anywhere. For a mapped page, call `boot_pgalloc()` or `boot_pgalloc_image()` instead;

##### Parameters

<i>boot_alloc</i>	the boot allocator state
-------------------	--------------------------

##### Returns

physical address of allocated page frame

Definition at line 210 of file `boot.c`.

References `boot_alloc_t::its_early`, `boot_alloc_t::kernel_paddr_limit`, `boot_alloc_t::kernel_paddr_top`, `PAGE_SIZE`, and `panic()`.

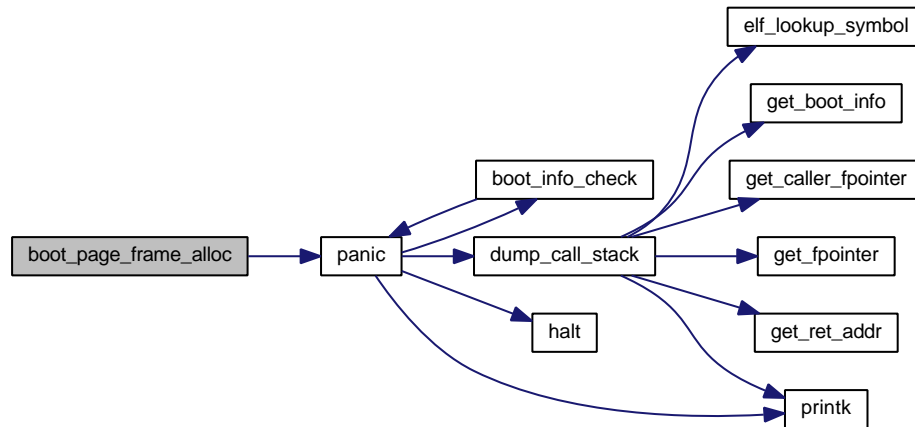
Referenced by `boot_page_alloc()`, `boot_page_alloc_image()`, `elf_load()`, and `elf_setup_stack()`.

```

210                                     {
211     if(boot_alloc->its_early) {
212         panic("boot_pfalloc() called too soon");
213     }
214
215     /* address of allocated page */
216     kern_paddr_t paddr = boot_alloc->kernel_paddr_top;
217
218     /* Update allocator state. */
219     boot_alloc->kernel_paddr_top = paddr + PAGE_SIZE;
220
221     /* Check bounds. */
222     if(boot_alloc->kernel_paddr_top > boot_alloc->kernel_paddr_limit) {
223         panic("pfalloc_boot(): available memory exhausted");
224     }
225
226     return paddr;
227 }

```

Here is the call graph for this function:



#### 4.80.1.9 addr\_t boot\_vmalloc ( boot\_alloc\_t \* boot\_alloc )

Allocate a page of address space.

No memory is mapped to the allocated page. The page is allocated from the image region of the kernel address space, just after the kernel image and other initialization-time page allocations. This function allocates pages sequentially.

##### Parameters

<i>boot_alloc</i>	the boot allocator state
-------------------	--------------------------

##### Returns

address of allocated page

Definition at line 241 of file boot.c.

References boot\_alloc\_t::its\_early, boot\_alloc\_t::kernel\_vm\_limit, boot\_alloc\_t::kernel\_vm\_top, NULL, PAGE\_SIZE, and panic().

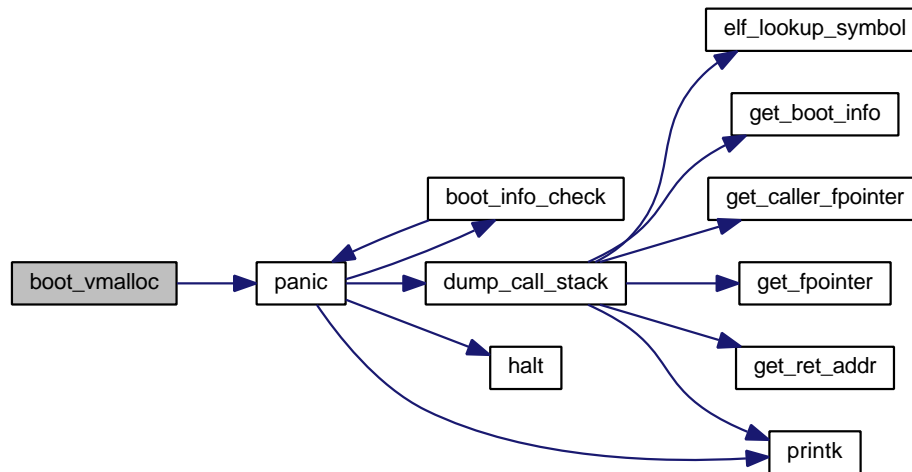
Referenced by boot\_page\_alloc\_image(), and vm\_boot\_init().

```

241     {
242     if(boot_alloc->its_early) {
243         panic("boot_vmalloc() called too soon");
244     }
245
246     if(boot_alloc->kernel_vm_top == NULL) {
247         panic("boot_pgalloc_early(): allocator is uninitialized");
248     }
249
250     /* address of allocated page */
251     addr_t page = boot_alloc->kernel_vm_top;
252
253     /* Update allocator state. */
254     boot_alloc->kernel_vm_top = page + PAGE_SIZE;
255
256     /* Check bounds. */
257     if(boot_alloc->kernel_vm_top > boot_alloc->kernel_vm_limit) {
258         panic("vmalloc_boot(): kernel address space exhausted");
259     }
260
261     return page;
262 }

```

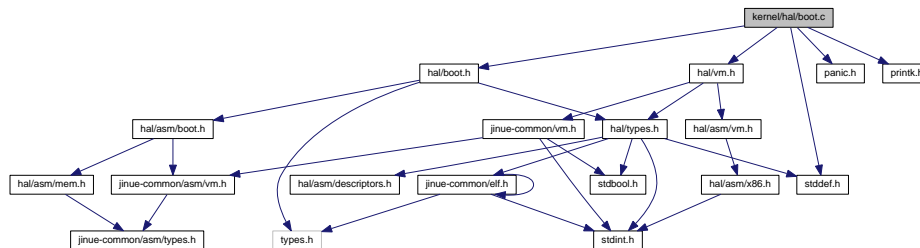
Here is the call graph for this function:



## 4.81 kernel/hal/boot.c File Reference

```
#include <hal/boot.h>
#include <hal/vm.h>
#include <panic.h>
#include <printk.h>
#include <stddef.h>
```

Include dependency graph for boot.c:



## Functions

- **bool** **boot\_info\_check**(bool panic\_on\_failure)
- const **boot\_info\_t** \* **get\_boot\_info**(void)
- void **boot\_info\_dump**(void)

## Variables

- const **boot\_info\_t** \* **boot\_info**

### 4.81.1 Function Documentation

#### 4.81.1.1 bool boot\_info\_check ( bool panic\_on\_failure )

Definition at line 42 of file boot.c.

References BOOT\_SETUP\_MAGIC, boot\_info\_t::image\_start, boot\_info\_t::kernel\_start, NULL, page\_offset\_of, panic(), and boot\_info\_t::setup\_signature.

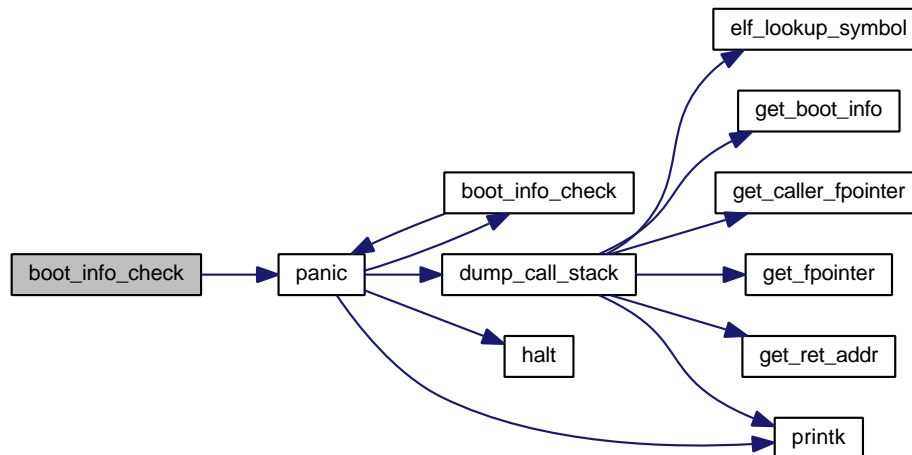
Referenced by kmain(), and panic().

```

42      {
43          const char *error_description = NULL;
44
45          /* This data structure is accessed early during the boot process, before
46           * paging is enabled. What this means is that, if boot_info is NULL and we
47           * dereference it, it does *not* cause a page fault or any other CPU
48           * exception. */
49          if(boot_info == NULL) {
50              error_description = "Boot information structure pointer is NULL.";
51          }
52          else if(boot_info->setup_signature != BOOT_SETUP_MAGIC) {
53              error_description = "Bad setup header signature.";
54          }
55          else if(page_offset_of(boot_info->image_start) != 0) {
56              error_description = "Bad image alignment.";
57          }
58          else if(page_offset_of(boot_info->kernel_start) != 0) {
59              error_description = "Bad kernel alignment.";
60          }
61          else {
62              return true;
63          }
64
65          if(panic_on_failure) {
66              panic(error_description);
67          }
68
69          return false;
70 }

```

Here is the call graph for this function:



#### 4.81.1.2 void boot\_info\_dump ( void )

Definition at line 76 of file boot.c.

References boot\_info\_t::boot\_end, boot\_info\_t::boot\_heap, boot\_info\_t::e820\_entries, boot\_info\_t::e820\_map, boot\_info\_t::image\_start, boot\_info\_t::image\_top, boot\_info\_t::kernel\_size, boot\_info\_t::kernel\_start, boot\_info\_t::page\_

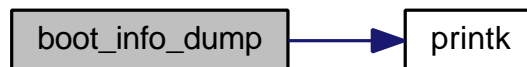


directory, boot\_info\_t::page\_table, printk(), boot\_info\_t::proc\_size, boot\_info\_t::proc\_start, and boot\_info\_t::setup\_signature.

```

76         {
77     printk("Boot information structure:\n");
78     printk("    kernel_start    %x  %u\n", boot_info->kernel_start    , boot_info->
kernel_start    );
79     printk("    kernel_size     %x  %u\n", boot_info->kernel_size     , boot_info->
kernel_size     );
80     printk("    proc_start      %x  %u\n", boot_info->proc_start      , boot_info->
proc_start      );
81     printk("    proc_size       %x  %u\n", boot_info->proc_size       , boot_info->
proc_size       );
82     printk("    image_start    %x  %u\n", boot_info->image_start    , boot_info->
image_start    );
83     printk("    image_top      %x  %u\n", boot_info->image_top      , boot_info->
image_top      );
84     printk("    e820_entries    %x  %u\n", boot_info->e820_entries    , boot_info->
e820_entries    );
85     printk("    e820_map       %x  %u\n", boot_info->e820_map       , boot_info->
e820_map       );
86     printk("    boot_heap      %x  %u\n", boot_info->boot_heap      , boot_info->
boot_heap      );
87     printk("    boot_end       %x  %u\n", boot_info->boot_end       , boot_info->
boot_end       );
88     printk("    page_table     %x  %u\n", boot_info->page_table     , boot_info->
page_table     );
89     printk("    page_directory %x  %u\n", boot_info->page_directory , boot_info->
page_directory );
90     printk("    setup_signature %x  %u\n", boot_info->setup_signature, boot_info->
setup_signature );
91 }
```

Here is the call graph for this function:



#### 4.81.1.3 const boot\_info\_t\* get\_boot\_info ( void )

Definition at line 72 of file boot.c.

References boot\_info.

Referenced by dispatch\_syscall(), dump\_call\_stack(), and kmain().

```

72         {
73     return boot_info;
74 }
```

### 4.81.2 Variable Documentation

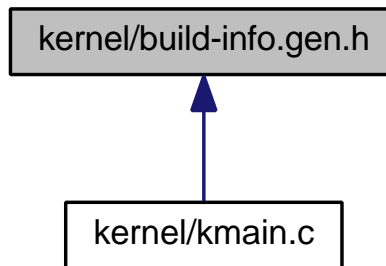
#### 4.81.2.1 const boot\_info\_t\* boot\_info

Definition at line 40 of file boot.c.

Referenced by dispatch\_syscall(), dump\_call\_stack(), and get\_boot\_info().

## 4.82 kernel/build-info.gen.h File Reference

This graph shows which files directly or indirectly include this file:



### Macros

- `#define GIT_REVISION "2a99f29"`
- `#define BUILD_TIME "Wed Mar 20 21:01:51 EDT 2019"`
- `#define BUILD_HOST "raskolnikov"`

### 4.82.1 Macro Definition Documentation

#### 4.82.1.1 `#define BUILD_HOST "raskolnikov"`

Definition at line 9 of file build-info.gen.h.

Referenced by kmain().

#### 4.82.1.2 `#define BUILD_TIME "Wed Mar 20 21:01:51 EDT 2019"`

Definition at line 8 of file build-info.gen.h.

Referenced by kmain().

#### 4.82.1.3 `#define GIT_REVISION "2a99f29"`

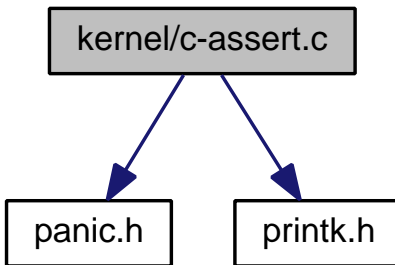
Definition at line 7 of file build-info.gen.h.

Referenced by kmain().

## 4.83 kernel/c-assert.c File Reference

```
#include <panic.h>
#include <printk.h>
```

Include dependency graph for c-assert.c:



## Functions

- void **\_\_assert\_failed** (const char \*expr, const char \*file, unsigned int line, const char \*func)

### 4.83.1 Function Documentation

4.83.1.1 void **\_\_assert\_failed** ( const char \* *expr*, const char \* *file*, unsigned int *line*, const char \* *func* )

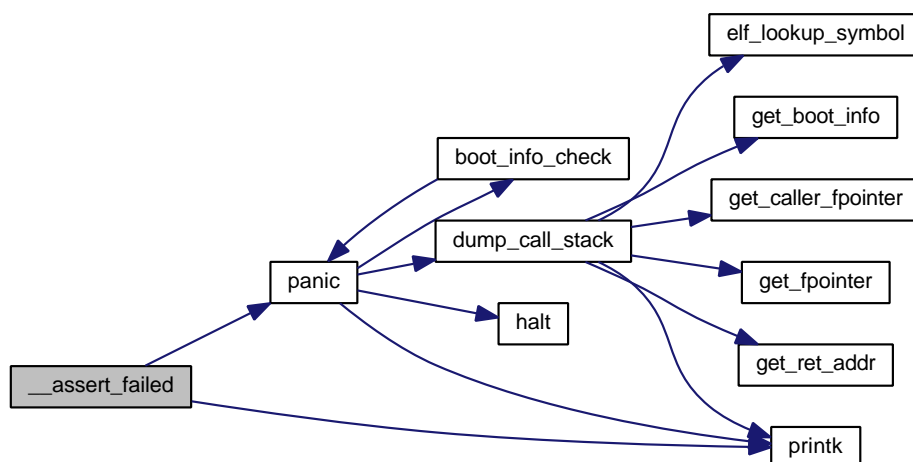
Definition at line 36 of file c-assert.c.

References panic(), and printk().

```

40         {
41
42     printk(
43         "ASSERTION FAILED [%s]: %s at line %u in function %s.\n",
44         expr, file, line, func );
45
46     panic("Assertion failed.");
47 }
  
```

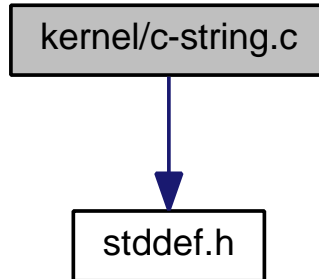
Here is the call graph for this function:



## 4.84 kernel/c-string.c File Reference

```
#include <stddef.h>
```

Include dependency graph for c-string.c:



### Functions

- void \* **memset** (void \*s, int c, **size\_t** n)
- void \* **memcpy** (void \*dest, const void \*src, **size\_t** n)
- **size\_t** **strlen** (const char \*s)

### 4.84.1 Function Documentation

#### 4.84.1.1 void\* memcpy ( void \* dest, const void \* src, size\_t n )

Definition at line 45 of file c-string.c.

Referenced by ipc\_receive(), ipc\_reply(), ipc\_send(), and process\_create\_initial().

```

45                                     {
46     size_t      idx;
47     char        *cdest  = dest;
48     const char  *csrc   = src;
49
50     for(idx = 0; idx < n; ++idx) {
51         cdest[idx] = csrc[idx];
52     }
53
54     return dest;
55 }
```

#### 4.84.1.2 void\* memset ( void \* s, int c, size\_t n )

Definition at line 34 of file c-string.c.

Referenced by boot\_alloc\_init(), clear\_page(), cpu\_init\_data(), thread\_page\_init(), and vm\_pae\_lookup\_page\_directory().

```

34                                     {
35     size_t      idx;
36     char        *cs = s;
37
38     for(idx = 0; idx < n; ++idx) {
39         cs[idx] = c;
40     }
```

```

41
42     return s;
43 }

```

#### 4.84.1.3 size\_t strlen ( const char \* s )

Definition at line 57 of file c-string.c.

Referenced by console\_print().

```

57     {
58         size_t count = 0;
59         while (*s != 0) {
60             ++s;
61             ++count;
62         }
63     }
64     return count;
65 }
66 }

```

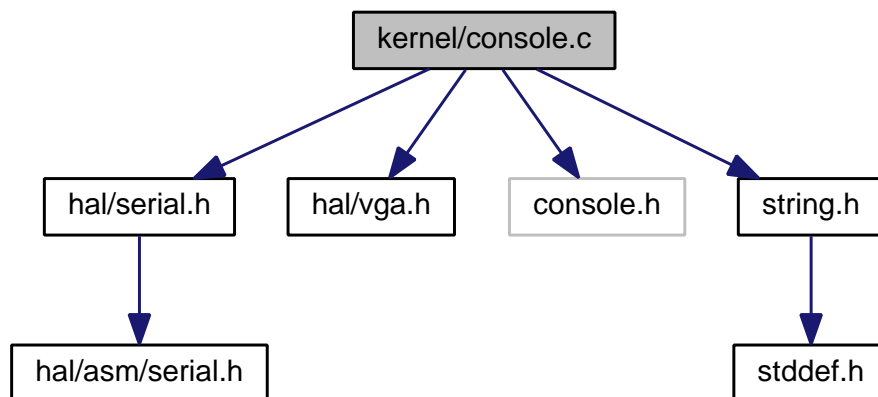
## 4.85 kernel/console.c File Reference

```

#include <hal/serial.h>
#include <hal/vga.h>
#include <console.h>
#include <string.h>

```

Include dependency graph for console.c:



### Functions

- void **console\_init** (void)
- void **console\_printn** (const char \*message, unsigned int n, int colour)
- void **console\_putc** (char c, int colour)
- void **console\_print** (const char \*message, int colour)

#### 4.85.1 Function Documentation

#### 4.85.1.1 void console\_init ( void )

Definition at line 38 of file console.c.

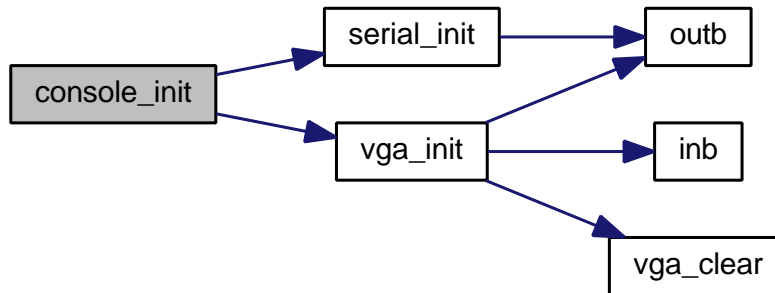
References `CONSOLE_SERIAL_BAUD_RATE`, `CONSOLE_SERIAL_IOPORT`, `serial_init()`, and `vga_init()`.

Referenced by `kmain()`.

```

38         {
39     vga_init();
40     serial_init(CONSOLE_SERIAL_IOPORT, CONSOLE_SERIAL_BAUD_RATE);
41 }
```

Here is the call graph for this function:



#### 4.85.1.2 void console\_print ( const char \* message, int colour )

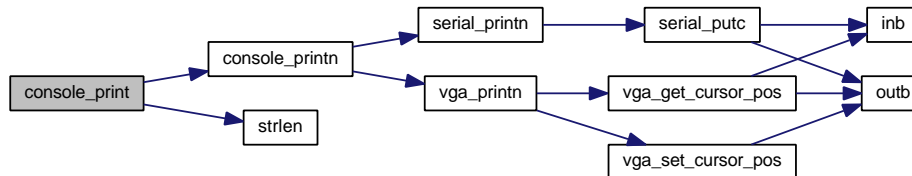
Definition at line 53 of file console.c.

References `console_printn()`, and `strlen()`.

```

53     {
54     console_printn(message, strlen(message), colour);
55 }
```

Here is the call graph for this function:



#### 4.85.1.3 void console\_printn ( const char \* message, unsigned int n, int colour )

Definition at line 43 of file console.c.

References `CONSOLE_SERIAL_IOPORT`, `serial_printn()`, and `vga_printn()`.

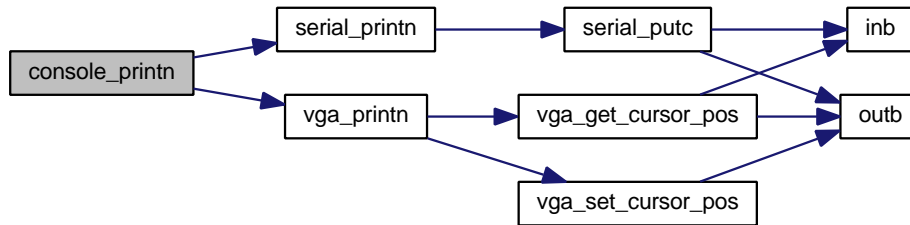
Referenced by `console_print()`, and `dispatch_syscall()`.

```

43                                     {
44     vga_printn(message, n, colour);
45     serial_printn(CONSOLE_SERIAL_IOPORT, message, n);
46 }

```

Here is the call graph for this function:



#### 4.85.1.4 void console\_putc ( char c, int colour )

Definition at line 48 of file console.c.

References CONSOLE\_SERIAL\_IOPORT, serial\_putc(), and vga\_putc().

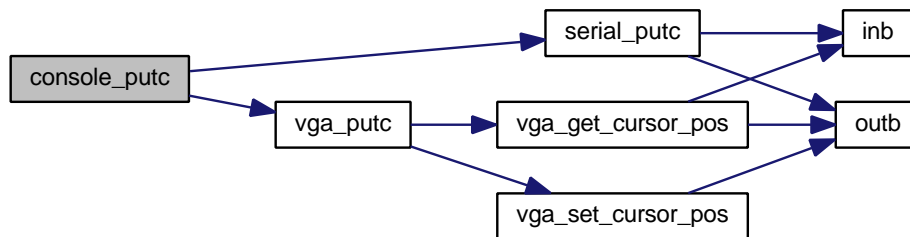
Referenced by dispatch\_syscall().

```

48                                     {
49     vga_putc(c, colour);
50     serial_putc(CONSOLE_SERIAL_IOPORT, c);
51 }

```

Here is the call graph for this function:



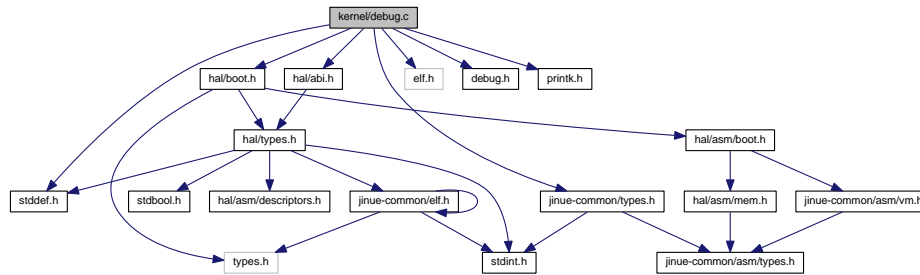
## 4.86 kernel/debug.c File Reference

```

#include <jinue-common/types.h>
#include <hal/abi.h>
#include <hal/boot.h>
#include <elf.h>
#include <stddef.h>
#include <debug.h>
#include <printk.h>

```

Include dependency graph for debug.c:



## Functions

- void **dump\_call\_stack** (void)

### 4.86.1 Function Documentation

#### 4.86.1.1 void dump\_call\_stack ( void )

Definition at line 41 of file debug.c.

References `elf_symbol_t::addr`, `boot_info`, `elf_lookup_symbol()`, `get_boot_info()`, `get_caller_fpointer()`, `get_fpointer()`, `get_ret_addr()`, `boot_info_t::kernel_start`, `elf_symbol_t::name`, `NULL`, `printk()`, and `STT_FUNCTION`.

Referenced by `panic()`.

```

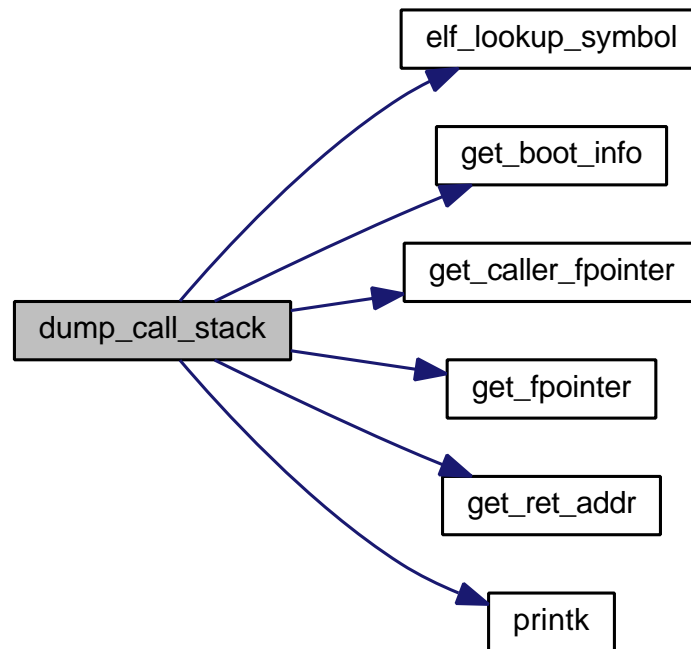
41      {
42      addr_t      fptr;
43
44      const boot_info_t *boot_info = get_boot_info();
45
46      printk("Call stack dump:\n");
47
48      fptr = get_fpointer();
49
50      while(fptr != NULL) {
51          addr_t return_addr = get_ret_addr(fptr);
52          if(return_addr == NULL) {
53              break;
54          }
55
56          /* assume e8 xx xx xx xx for call instruction encoding */
57          return_addr -= 5;
58
59          elf_symbol_t symbol;
60          int retval = elf_lookup_symbol(
61              boot_info->kernel_start,
62              (Elf32_Addr)return_addr,
63              STT_FUNCTION,
64              &symbol);
65
66          if(retval < 0) {
67              printk("\t0x%x (unknown)\n", return_addr);
68          }
69          else {
70              const char *name = symbol.name;
71
72              if(name == NULL) {
73                  name = "[unknown]";
74              }
75
76              printk(
77                  "\t0x%x (%s+u)\n",
78                  return_addr,

```



```
79         name,  
80         return_addr - symbol.addr);  
81     }  
82  
83     fptr = get_caller_fpointer(fptr);  
84 }  
85 }
```

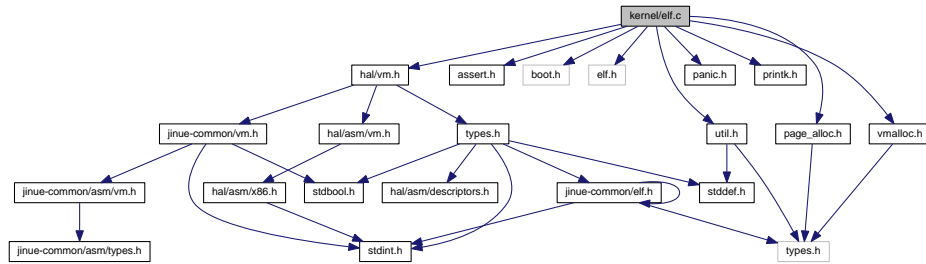
Here is the call graph for this function:



## 4.87 kernel/elf.c File Reference

```
#include <hal/vm.h>  
#include <assert.h>  
#include <boot.h>  
#include <elf.h>  
#include <page_alloc.h>  
#include <panic.h>  
#include <printk.h>  
#include <vmalloc.h>  
#include <util.h>
```

Include dependency graph for elf.c:



## Functions

- void **elf\_check** (**Elf32\_Ehdr** \*elf)
- void **elf\_load** (**elf\_info\_t** \*info, **Elf32\_Ehdr** \*elf, **addr\_space\_t** \*addr\_space, **boot\_alloc\_t** \*boot\_alloc)
- void **elf\_setup\_stack** (**elf\_info\_t** \*info, **boot\_alloc\_t** \*boot\_alloc)
- int **elf\_lookup\_symbol** (const **Elf32\_Ehdr** \*elf\_header, **Elf32\_Addr** addr, int type, **elf\_symbol\_t** \*result)

### 4.87.1 Function Documentation

#### 4.87.1.1 void elf\_check ( Elf32\_Ehdr \* elf )

Definition at line 43 of file elf.c.

References `Elf32_Ehdr::e_entry`, `Elf32_Ehdr::e_flags`, `Elf32_Ehdr::e_ident`, `Elf32_Ehdr::e_machine`, `Elf32_Ehdr::e_phentsize`, `Elf32_Ehdr::e_phnum`, `Elf32_Ehdr::e_phoff`, `Elf32_Ehdr::e_type`, `Elf32_Ehdr::e_version`, `EI_CLASS`, `EI_DATA`, `EI_MAG0`, `EI_MAG1`, `EI_MAG2`, `EI_MAG3`, `EI_VERSION`, `ELF_MAGIC0`, `ELF_MAGIC1`, `ELF_MAGIC2`, `ELF_MAGIC3`, `ELFCLASS32`, `ELFDATA2LSB`, `EM_386`, `ET_EXEC`, and `panic()`.

Referenced by `elf_load()`.

```

43         {
44     /* check: valid ELF binary magic number */
45     if (    elf->e_ident[EI_MAG0] != ELF_MAGIC0 ||
46         elf->e_ident[EI_MAG1] != ELF_MAGIC1 ||
47         elf->e_ident[EI_MAG2] != ELF_MAGIC2 ||
48         elf->e_ident[EI_MAG3] != ELF_MAGIC3 ) {
49         panic("Not an ELF binary");
50     }
51
52     /* check: 32-bit objects */
53     if (elf->e_ident[EI_CLASS] != ELFCLASS32) {
54         panic("Bad file class");
55     }
56
57     /* check: endianness */
58     if (elf->e_ident[EI_DATA] != ELFDATA2LSB) {
59         panic("Bad endianness");
60     }
61
62     /* check: version */
63     if (elf->e_version != 1 || elf->e_ident[EI_VERSION] != 1) {
64         panic("Not ELF version 1");
65     }
66
67     /* check: machine */
68     if (elf->e_machine != EM_386) {
69         panic("This process manager binary does not target the x86 architecture");
70     }
71
72     /* check: the 32-bit Intel architecture defines no flags */
73     if (elf->e_flags != 0) {

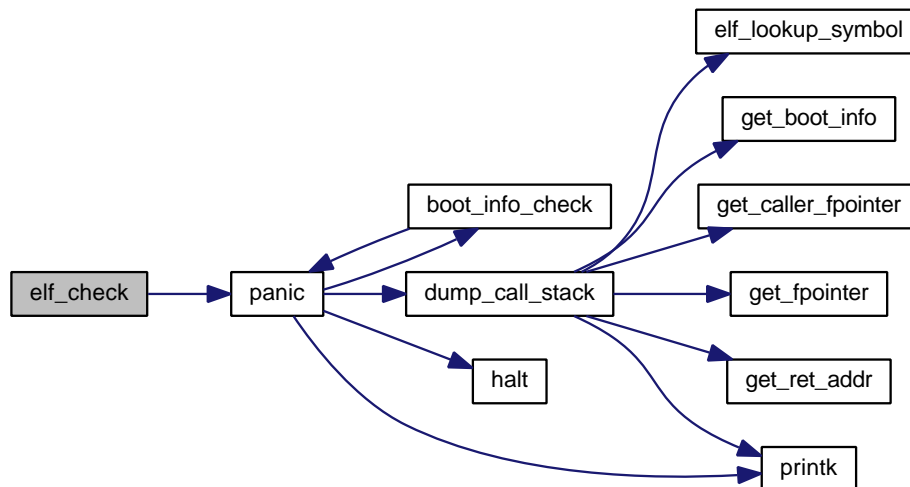
```

```

74     panic("Invalid flags specified");
75 }
76
77 /* check: file type is executable */
78 if(elf->e_type != ET_EXEC) {
79     panic("process manager binary is not an an executable");
80 }
81
82 /* check: must have a program header */
83 if(elf->e_phoff == 0 || elf->e_phnum == 0) {
84     panic("No program headers");
85 }
86
87 /* check: must have an entry point */
88 if(elf->e_entry == 0) {
89     panic("No entry point for process manager");
90 }
91
92 /* check: program header entry size */
93 if(elf->e_phentsize != sizeof(Elf32_Phdr)) {
94     panic("Unsupported program header size");
95 }
96 }

```

Here is the call graph for this function:



**4.87.1.2** void elf\_load ( elf\_info\_t \* info, Elf32\_Ehdr \* elf, addr\_space\_t \* addr\_space, boot\_alloc\_t \* boot\_alloc )

TODO: add exec flag once PAE is enabled

TODO add exec flag once PAE is enabled TODO lookup actual address of page frame

Definition at line 98 of file elf.c.

References elf\_info\_t::addr\_space, ALIGN\_END\_PTR, ALIGN\_START\_PTR, elf\_info\_t::at\_phdr, elf\_info\_t::at\_phent, elf\_info\_t::at\_phnum, boot\_page\_frame\_alloc(), Elf32\_Ehdr::e\_entry, Elf32\_Ehdr::e\_phentsize, Elf32\_Ehdr::e\_phnum, Elf32\_Ehdr::e\_phoff, EARLY\_PTR\_TO\_PHYS\_ADDR, elf\_check(), elf\_setup\_stack(), elf\_info\_t::entry, Elf32\_Phdr::p\_filesz, Elf32\_Phdr::p\_memsz, Elf32\_Phdr::p\_vaddr, PAGE\_SIZE, panic(), PF\_W, printk(), PT\_LOAD, VM\_FLAG\_READ\_ONLY, VM\_FLAG\_READ\_WRITE, and vm\_map\_user().

Referenced by kmain().

```

102     {
103
104     unsigned int idx;

```

```

105
106  /* check that ELF binary is valid */
107  elf_check(elf);
108
109  /* get the program header table */
110  Elf32_Phdr * phdr   = (Elf32_Phdr *) ((char *)elf + elf->e_phoff);
111
112  info->at_phdr        = (addr_t)phdr;
113  info->at_phnum        = elf->e_phnum;
114  info->at_phent        = elf->e_phentsize;
115  info->addr_space      = addr_space;
116  info->entry           = (addr_t)elf->e_entry;
117
118  for(idx = 0; idx < elf->e_phnum; ++idx) {
119      if(phdr[idx].p_type != PT_LOAD) {
120          continue;
121      }
122
123      /* check that the segment is not in the region reserved for kernel use */
124      if(! user_buffer_check((void *)phdr[idx].p_vaddr, phdr[idx].p_memsz)) {
125          panic("process manager memory layout -- address of segment too low");
126      }
127
128      /* set start and end addresses for mapping and copying */
129      char *file_ptr    = (char *)elf + phdr[idx].p_offset;
130      char *vptr        = (char *)phdr[idx].p_vaddr;
131      char *vend        = vptr + phdr[idx].p_memsz; /* limit for padding */
132      char *vfend       = vptr + phdr[idx].p_filesz; /* limit for copy */
133
134      /* align on page boundaries, be inclusive,
135       note that vfend is not aligned */
136      file_ptr          = ALIGN_START_PTR(file_ptr, PAGE_SIZE);
137      vptr              = ALIGN_START_PTR(vptr, PAGE_SIZE);
138      vend              = ALIGN_END_PTR(vend, PAGE_SIZE);
139
140      /* copy if we have to */
141      if( (phdr[idx].p_flags & PF_W) || (phdr[idx].p_filesz != phdr[idx].p_memsz) ) {
142          while(vptr < vend) {
143              unsigned long flags;
144              char *stop;
145
146              /* start of this page and next page */
147              char *vnext = vptr + PAGE_SIZE;
148
149              /* set flags */
150              if(phdr[idx].p_flags & PF_W) {
151                  flags = VM_FLAG_READ_WRITE;
152              }
153              else {
154                  flags = VM_FLAG_READ_ONLY;
155              }
156
157              /* allocate and map the new page */
158              kern_paddr_t page = boot_page_frame_alloc(boot_alloc);
159              vm_map_user(addr_space, (addr_t)vptr, page, flags);
160
161              /* copy */
162              if(vnext > vfend) {
163                  stop = vfend;
164              }
165              else {
166                  stop = vnext;
167              }
168
169              while(vptr < stop) {
170                  *(vptr++) = *(file_ptr++);
171              }
172
173              /* pad */
174              while(vptr < vnext) {
175                  *(vptr++) = 0;
176              }
177          }
178      }
179      else {
180          while(vptr < vend) {
181              /* perform mapping */
182              vm_map_user(addr_space, (addr_t)vptr, EARLY_PTR_TO_PHYS_ADDR(file_ptr),
183 VM_FLAG_READ_ONLY);
184
185              vptr += PAGE_SIZE;
186          }
187

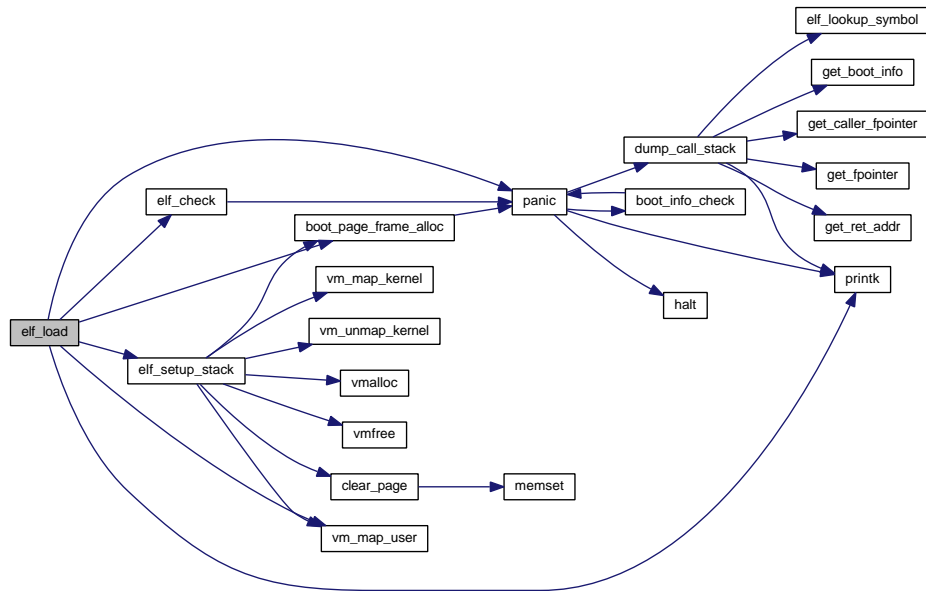
```

```

188         file_ptr += PAGE_SIZE;
189     }
190 }
191 }
192
193 elf_setup_stack(info, boot_alloc);
194 printk("ELF binary loaded.\n");
195
196 }

```

Here is the call graph for this function:



#### 4.87.1.3 int elf\_lookup\_symbol ( const Elf32\_Ehdr \* elf\_header, Elf32\_Addr addr, int type, elf\_symbol\_t \* result )

Definition at line 272 of file elf.c.

References `elf_symbol_t::addr`, `Elf32_Ehdr::e_shnum`, `ELF32_ST_TYPE`, `elf_symbol_t::name`, `NULL`, `Elf32_Shdr::sh_entsize`, `Elf32_Shdr::sh_link`, `Elf32_Shdr::sh_offset`, `Elf32_Shdr::sh_size`, `Elf32_Shdr::sh_type`, `SHT_SYMTAB`, `Elf32_Sym::st_info`, `Elf32_Sym::st_name`, `Elf32_Sym::st_size`, and `Elf32_Sym::st_value`.

Referenced by `dump_call_stack()`.

```

276                                     {
277
278     int      idx;
279     size_t   symbol_entry_size;
280     size_t   symbol_table_size;
281
282     const char *elf_file      = elf_file_bytes(elf_header);
283     const char *symbols_table = NULL;
284     const char *string_table  = NULL;
285
286     for(idx = 0; idx < elf_header->e_shnum; ++idx) {
287         const Elf32_Shdr *section_header = elf_get_section_header(elf_header, idx);
288
289         if(section_header->sh_type == SHT_SYMTAB) {
290             symbols_table = &elf_file[section_header->sh_offset];
291             symbol_entry_size = section_header->sh_entsize;
292             symbol_table_size = section_header->sh_size;
293
294             const Elf32_Shdr *string_section_header = elf_get_section_header(
295                 elf_header,

```

```

296         section_header->sh_link);
297
298     string_table = &elf_file[string_section_header->sh_offset];
299
300     break;
301 }
302 }
303
304 if(symbols_table == NULL) {
305     /* no symbol table */
306     return -1;
307 }
308
309 const char *symbol = symbols_table;
310
311 while(symbol < symbols_table + symbol_table_size) {
312     const Elf32_Sym *symbol_header = (const Elf32_Sym *)symbol;
313
314     if(ELF32_ST_TYPE(symbol_header->st_info) == type) {
315         Elf32_Addr lookup_addr = (Elf32_Addr)addr;
316         Elf32_Addr start      = symbol_header->st_value;
317         Elf32_Addr end        = start + symbol_header->st_size;
318
319         if(lookup_addr >= start && lookup_addr < end) {
320             result->addr = symbol_header->st_value;
321             result->name = &string_table[symbol_header->st_name];
322
323             return 0;
324         }
325     }
326
327     symbol += symbol_entry_size;
328 }
329
330 /* not found */
331 return -1;
332 }

```

#### 4.87.1.4 void elf\_setup\_stack ( elf\_info\_t\* info, boot\_alloc\_t\* boot\_alloc )

TODO: check for overlap of stack with loaded segments

Definition at line 198 of file elf.c.

References Elf32\_auxv\_t::a\_type, Elf32\_auxv\_t::a\_un, Elf32\_auxv\_t::a\_val, elf\_info\_t::addr\_space, AT\_ENTRY, AT\_NULL, AT\_PAGESZ, elf\_info\_t::at\_phdr, AT\_PHDR, elf\_info\_t::at\_phent, AT\_PHENT, elf\_info\_t::at\_phnum, AT\_PHNUM, AT\_STACKBASE, boot\_page\_frame\_alloc(), clear\_page(), elf\_info\_t::entry, PAGE\_SIZE, elf\_info\_t::stack\_addr, STACK\_BASE, STACK\_START, VM\_FLAG\_READ\_WRITE, vm\_map\_kernel(), vm\_map\_user(), vm\_unmap\_kernel(), vmalloc(), and vmfree().

Referenced by elf\_load().

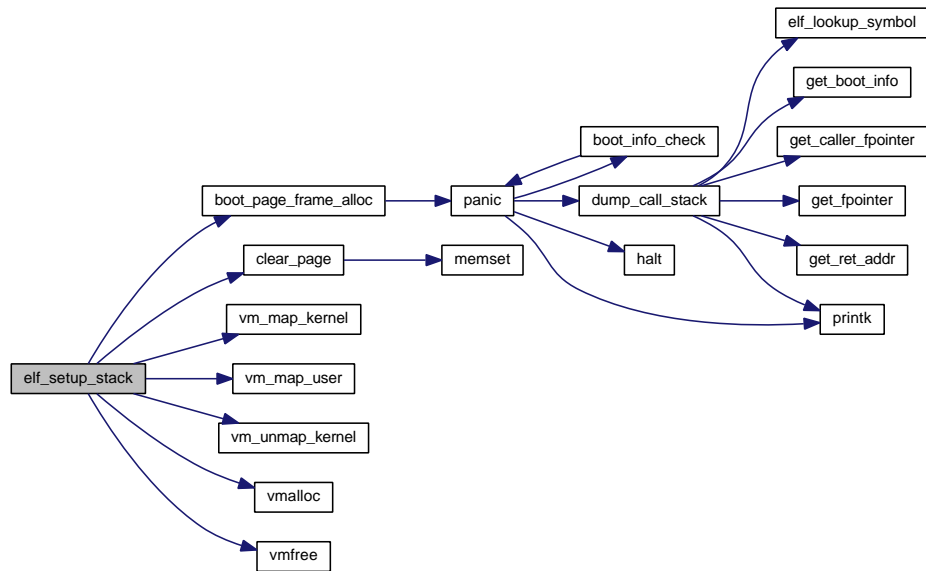
```

198
199     kern_paddr_t page;
200     addr_t vpage;
201
202     /* initial stack allocation */
203     for(vpage = (addr_t)STACK_START; vpage < (addr_t)STACK_BASE; vpage +=
204     PAGE_SIZE) {
205         page = boot_page_frame_alloc(boot_alloc);
206         vm_map_user(info->addr_space, vpage, page, VM_FLAG_READ_WRITE);
207
208         /* This newly allocated page may have data left from a previous boot which
209          * may contain sensitive information. Let's clear it. */
210         clear_page(vpage);
211     }
212
213     /* At this point, page has the address of the stack's top-most page frame,
214      * which is the one in which we are about to copy the auxiliary vectors. Map
215      * it temporarily in this address space so we can write to it. */
216     addr_t top_page = vmalloc();
217     vm_map_kernel(top_page, page, VM_FLAG_READ_WRITE);
218
219

```

```
220  /* start at the top */
221  uint32_t *sp = (uint32_t *) (top_page + PAGE_SIZE);
222
223  /* Program name string: "proc", null-terminated */
224  * (--sp) = 0;
225  * (--sp) = 0x636f7270;
226
227  char *argv0 = (char *) STACK_BASE - 2 * sizeof(uint32_t);
228
229  /* auxiliary vectors */
230  Elf32_auxv_t *auxvp = (Elf32_auxv_t *) sp - 7;
231
232  auxvp[0].a_type = AT_PHDR;
233  auxvp[0].a_un.a_val = (int32_t) info->at_phdr;
234
235  auxvp[1].a_type = AT_PHENT;
236  auxvp[1].a_un.a_val = (int32_t) info->at_phent;
237
238  auxvp[2].a_type = AT_PHNUM;
239  auxvp[2].a_un.a_val = (int32_t) info->at_phnum;
240
241  auxvp[3].a_type = AT_PAGESZ;
242  auxvp[3].a_un.a_val = PAGE_SIZE;
243
244  auxvp[4].a_type = AT_ENTRY;
245  auxvp[4].a_un.a_val = (int32_t) info->entry;
246
247  auxvp[5].a_type = AT_STACKBASE;
248  auxvp[5].a_un.a_val = STACK_BASE;
249
250  auxvp[6].a_type = AT_NULL;
251  auxvp[6].a_un.a_val = 0;
252
253  sp = (uint32_t *) auxvp;
254
255  /* empty environment variables */
256  * (--sp) = 0;
257
258  /* argv with only program name */
259  * (--sp) = 0;
260  * (--sp) = (uint32_t) argv0;
261
262  /* argc */
263  * (--sp) = 1;
264
265  info->stack_addr = (addr_t) STACK_BASE - PAGE_SIZE + ((addr_t) sp - top_page);
266
267  /* unmap and free temporary page */
268  vm_unmap_kernel(top_page);
269  vmfree(top_page);
270 }
```

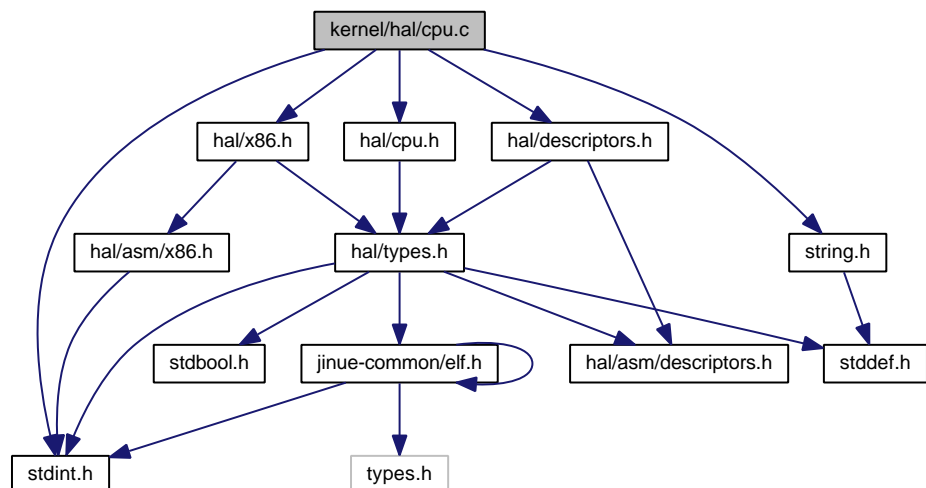
Here is the call graph for this function:



## 4.88 kernel/hal/cpu.c File Reference

```
#include <hal/cpu.h>
#include <hal/descriptors.h>
#include <hal/x86.h>
#include <stdint.h>
#include <string.h>
```

Include dependency graph for `cpu.c`:



## Functions

- void `cpu_init_data(cpu_data_t *data)`



- void **cpu\_detect\_features** (void)

## Variables

- **cpu\_info\_t** **cpu\_info**

### 4.88.1 Function Documentation

#### 4.88.1.1 void **cpu\_detect\_features** ( void )

Definition at line 87 of file `cpu.c`.

References `CPU_EFLAGS_ID`, `CPU_FEATURE_CPUID`, `CPU_FEATURE_LOCAL_APIC`, `CPU_FEATURE_PAE`, `CPU_FEATURE_SYSCALL`, `CPU_FEATURE_SYSENTER`, `CPU_VENDOR_AMD`, `CPU_VENDOR_AMD_DW0`, `CPU_VENDOR_AMD_DW1`, `CPU_VENDOR_AMD_DW2`, `CPU_VENDOR_GENERIC`, `CPU_VENDOR_INTEL`, `CPU_VENDOR_INTEL_DW0`, `CPU_VENDOR_INTEL_DW1`, `CPU_VENDOR_INTEL_DW2`, `cpuid()`, `CPUID_EXT_FEATURE_SYSCALL`, `CPUID_FEATURE_APIC`, `CPUID_FEATURE_CLFLUSH`, `CPUID_FEATURE_PAE`, `CPUID_FEATURE_SEP`, `cpu_info_t::dcache_alignment`, `x86_cpuid_regs_t::eax`, `x86_cpuid_regs_t::ebx`, `x86_cpuid_regs_t::ecx`, `x86_cpuid_regs_t::edx`, `cpu_info_t::family`, `cpu_info_t::features`, `get_eflags()`, `cpu_info_t::model`, `set_eflags()`, `cpu_info_t::stepping`, and `cpu_info_t::vendor`.

Referenced by `hal_init()`.

```

87     {
88         uint32_t temp_eflags;
89
90         /* default values */
91         cpu_info.dcache_alignment = 32;
92         cpu_info.features         = 0;
93         cpu_info.vendor           = CPU_VENDOR_GENERIC;
94         cpu_info.family           = 0;
95         cpu_info.model            = 0;
96         cpu_info.stepping         = 0;
97
98         /* The CPUID instruction is available if we can change the value of eflags
99          * bit 21 (ID) */
100        temp_eflags = get_eflags();
101        temp_eflags ^= CPU_EFLAGS_ID;
102        set_eflags(temp_eflags);
103
104        if(temp_eflags == get_eflags()) {
105            cpu_info.features |= CPU_FEATURE_CPUID;
106        }
107
108        if(cpu_has_feature(CPU_FEATURE_CPUID)) {
109            uint32_t signature;
110            uint32_t flags, ext_flags;
111            uint32_t vendor_dw0, vendor_dw1, vendor_dw2;
112            uint32_t cpuid_max;
113            uint32_t cpuid_ext_max;
114            x86_cpuid_regs_t regs;
115
116            /* default values */
117            flags = 0;
118            ext_flags = 0;
119
120            /* function 0: vendor ID string, max value of eax when calling CPUID */
121            regs.eax = 0;
122
123            /* call CPUID instruction */
124            cpuid_max = cpuid(&regs);
125            vendor_dw0 = regs.ebx;
126            vendor_dw1 = regs.edx;
127            vendor_dw2 = regs.ecx;
128
129            /* identify vendor */
130            if( vendor_dw0 == CPU_VENDOR_AMD_DW0
131               && vendor_dw1 == CPU_VENDOR_AMD_DW1

```

```

132         && vendor_dw2 == CPU_VENDOR_AMD_DW2) {
133
134         cpu_info.vendor = CPU_VENDOR_AMD;
135     }
136     else if (vendor_dw0 == CPU_VENDOR_INTEL_DW0
137         && vendor_dw1 == CPU_VENDOR_INTEL_DW1
138         && vendor_dw2 == CPU_VENDOR_INTEL_DW2) {
139
140         cpu_info.vendor = CPU_VENDOR_INTEL;
141     }
142
143     /* get processor signature (family/model/stepping) and feature flags */
144     if(cpuinfo_max >= 1) {
145         /* function 1: processor signature and feature flags */
146         regs.eax = 1;
147
148         /* call CPUID instruction */
149         signature = cpuid(&regs);
150
151         /* set processor signature */
152         cpu_info.stepping = signature & 0xf;
153         cpu_info.model = (signature>>4) & 0xf;
154         cpu_info.family = (signature>>8) & 0xf;
155
156         /* feature flags */
157         flags = regs.edx;
158
159         /* cache alignment */
160         if(flags & CPUID_FEATURE_CLFLUSH) {
161             cpu_info.dcache_alignment = ((regs.ebx >> 8) & 0xff) * 8;
162         }
163     }
164
165     /* extended function 0: max value of eax when calling CPUID (extended function) */
166     regs.eax = 0x80000000;
167     cpuid_ext_max = cpuid(&regs);
168
169     /* get extended feature flags */
170     if(cpuid_ext_max >= 0x80000001) {
171         /* extended function 1: extended feature flags */
172         regs.eax = 0x80000001;
173         (void)cpuid(&regs);
174
175         /* extended feature flags */
176         ext_flags = regs.edx;
177     }
178
179     /* support for SYSENTER/SYSEXIT instructions */
180     if(flags & CPUID_FEATURE_SEP) {
181         if(cpu_info.vendor == CPU_VENDOR_AMD) {
182             cpu_info.features |= CPU_FEATURE_SYSENTER;
183         }
184         else if(cpu_info.vendor == CPU_VENDOR_INTEL) {
185             if(cpu_info.family == 6 && cpu_info.model < 3 && cpu_info.
stepping < 3) {
186                 /* not supported */
187             }
188             else {
189                 cpu_info.features |= CPU_FEATURE_SYSENTER;
190             }
191         }
192     }
193
194     /* support for SYSCALL/SYSRET instructions */
195     if(cpu_info.vendor == CPU_VENDOR_AMD) {
196         if(ext_flags & CPUID_EXT_FEATURE_SYSCALL) {
197             cpu_info.features |= CPU_FEATURE_SYSCALL;
198         }
199     }
200
201     /* support for local APIC */
202     if(cpu_info.vendor == CPU_VENDOR_AMD || cpu_info.vendor ==
CPU_VENDOR_INTEL) {
203         if(flags & CPUID_FEATURE_APIC) {
204             cpu_info.features |= CPU_FEATURE_LOCAL_APIC;
205         }
206     }
207
208     /* support for physical address extension (PAE) */
209     if(cpu_info.vendor == CPU_VENDOR_AMD || cpu_info.vendor ==
CPU_VENDOR_INTEL) {

```

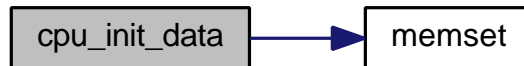


```

69     SEG_DESCRIPTOR( tss,    TSS_LIMIT-1,          SEG_TYPE_TSS    |
70     SEG_FLAG_KERNEL | SEG_FLAG_TSS);
71     data->gdt[GDT_PER_CPU_DATA] =
72     SEG_DESCRIPTOR( data,    sizeof(cpu_data_t)-1,  SEG_TYPE_DATA    |
73     SEG_FLAG_KERNEL | SEG_FLAG_32BIT | SEG_FLAG_IN_BYTES | SEG_FLAG_NOSYSTEM |
74     SEG_FLAG_PRESENT);
75
76     data->gdt[GDT_USER_TLS_DATA] = SEG_DESCRIPTOR(0, 0, 0);
77
78     /* setup kernel stack in TSS */
79     tss->ss0 = SEG_SELECTOR(GDT_KERNEL_DATA, RPL_KERNEL);
80     tss->ss1 = SEG_SELECTOR(GDT_KERNEL_DATA, RPL_KERNEL);
81     tss->ss2 = SEG_SELECTOR(GDT_KERNEL_DATA, RPL_KERNEL);
82
83     /* kernel stack address is updated by thread_context_switch() */
84     tss->esp0 = NULL;
85     tss->esp1 = NULL;
86     tss->esp2 = NULL;
87 }

```

Here is the call graph for this function:



## 4.88.2 Variable Documentation

### 4.88.2.1 cpu\_info\_t cpu\_info

Definition at line 39 of file cpu.c.

Referenced by slab\_cache\_init().

## 4.89 kernel/hal/hal.c File Reference

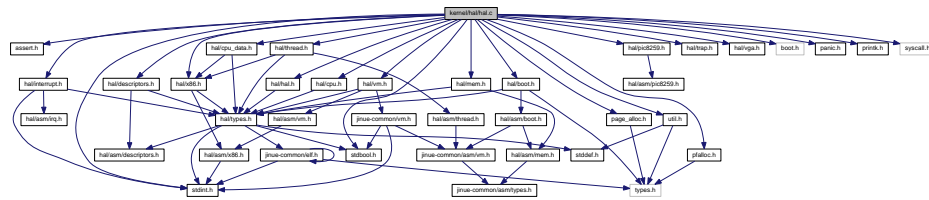
```
#include <assert.h>
```

```

#include <hal/boot.h>
#include <hal/cpu.h>
#include <hal/cpu_data.h>
#include <hal/descriptors.h>
#include <hal/hal.h>
#include <hal/interrupt.h>
#include <hal/mem.h>
#include <hal/pic8259.h>
#include <hal/thread.h>
#include <hal/trap.h>
#include <hal/vga.h>
#include <hal/vm.h>
#include <hal/x86.h>
#include <boot.h>
#include <panic.h>
#include <page_alloc.h>
#include <pfalloc.h>
#include <printk.h>
#include <stdbool.h>
#include <stdint.h>
#include <syscall.h>
#include <util.h>

```

Include dependency graph for hal.c:



## Functions

- void **hal\_init** (**boot\_alloc\_t** \*boot\_alloc, const **boot\_info\_t** \*boot\_info)

## Variables

- int **syscall\_method**  
*Specifies the entry point to use for system calls.*

### 4.89.1 Function Documentation

#### 4.89.1.1 void hal\_init ( boot\_alloc\_t \* boot\_alloc, const boot\_info\_t \* boot\_info )

Definition at line 156 of file hal.c.

References `assert`, `boot_heap_alloc`, `boot_page_alloc()`, `boot_page_alloc_image()`, `CPU_DATA_ALIGNMENT`, `cpu_detect_features()`, `CPU_FEATURE_PAE`, `cpu_init_data()`, `global_pfalloc_cache`, `IDT_PIC8259_BASE`, `init_pfalloc_cache()`, `KERNEL_PAGE_STACK_INIT`, `page_free()`, `pfalloc`, `pic8259_init()`, `vm_boot_init()`, `vm_boot_postinit()`, and `vm_lookup_kernel_paddr()`.

Referenced by `kmain()`.

```

156                                     {
157     int idx;
158
159     /* get cpu info */
160     cpu_detect_features();
161
162     /* allocate per-CPU data
163     *
164     * We need to ensure that the Task State Segment (TSS) contained in this
165     * memory block does not cross a page boundary. */
166     assert(sizeof(cpu_data_t) < CPU_DATA_ALIGNMENT);
167
168     cpu_data_t *cpu_data = boot_heap_alloc(boot_alloc, cpu_data_t,
169     CPU_DATA_ALIGNMENT);
170
171     /* initialize per-CPU data */
172     cpu_init_data(cpu_data);
173
174     /* Initialize interrupt descriptor table (IDT)
175     *
176     * This function modifies the IDT in-place (see trap.asm). This must be
177     * done before vm_boot_init() because the page protection bits set up by
178     * vm_boot_init() prevent this. */
179     hal_init_idt();
180
181     /* Initialize programmable interrupt_controller. */
182     pic8259_init(IDT_PIC8259_BASE);
183
184     /* initialize virtual memory management, enable paging
185     *
186     * below this point, it is no longer safe to call pfalloc_early() */
187     bool use_pae = cpu_has_feature(CPU_FEATURE_PAE);
188     vm_boot_init(boot_info, use_pae, cpu_data, boot_alloc);
189
190     /* Initialize GDT and TSS */
191     hal_init_descriptors(cpu_data, boot_alloc);
192
193     /* initialize the page frame allocator */
194     kern_paddr_t *page_stack_buffer = (kern_paddr_t *)boot_page_alloc_image(boot_alloc);
195     init_pfalloc_cache(&global_pfalloc_cache, page_stack_buffer);
196
197     /* TODO Remove this once vm.c rework is done. */
198     for(idx = 0; idx < KERNEL_PAGE_STACK_INIT; ++idx) {
199         pffree(
200             vm_lookup_kernel_paddr(
201                 boot_page_alloc_image(boot_alloc)));
202     }
203
204     /* Initialize virtual memory allocator and VM management caches. */
205     vm_boot_postinit(boot_info, boot_alloc, use_pae);
206
207     /* TODO Remove this once add page frame system call is implemented.
208     *
209     * The test user space program needs one page to create a new thread. */
210     page_free(boot_page_alloc(boot_alloc));
211
212     /* choose system call method */
213     hal_select_syscall_method();
214 }

```

[illegible]

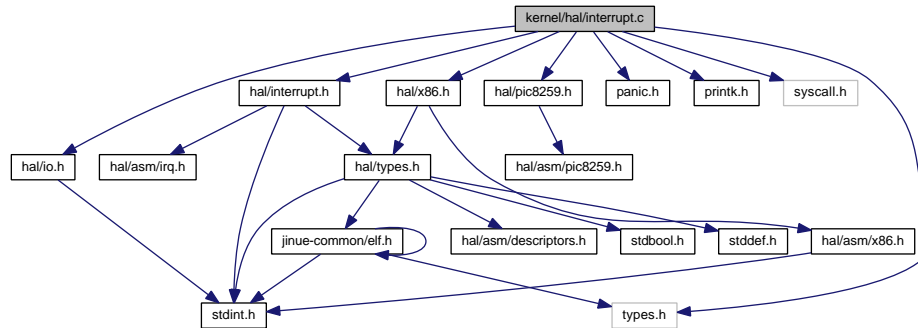
#### 4.89.2.1 int syscall\_method

Definition at line 58 of file hal.c.

## 4.90 kernel/hal/interrupt.c File Reference

Generated on Wed Mar 20 2019 21:26:14 for Jinue by Doxygen

Include dependency graph for interrupt.c:



## Functions

- void **dispatch\_interrupt** (trapframe\_t \*trapframe)

### 4.90.1 Function Documentation

#### 4.90.1.1 void dispatch\_interrupt ( trapframe\_t \* trapframe )

Definition at line 42 of file interrupt.c.

References `dispatch_syscall()`, `trapframe_t::eip`, `trapframe_t::errcode`, `get_cr2()`, `IDT_LAST_EXCEPTION`, `IDT_PIC8259_BASE`, `trapframe_t::ivt`, `panic()`, `pic8259_eoi()`, `PIC8259_IRQ_COUNT`, `printk()`, and `SYSCALL_IRQ`.

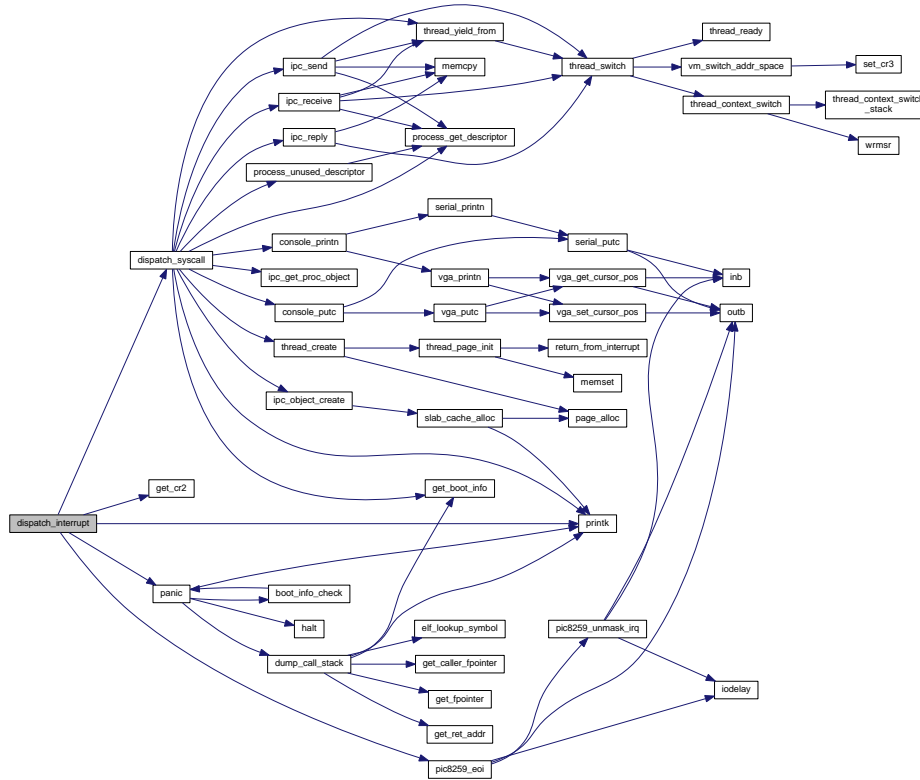
```

42      {
43      unsigned int    ivt      = trapframe->ivt;
44      uintptr_t      eip      = trapframe->eip;
45      uint32_t        errcode  = trapframe->errcode;
46
47      /* exceptions */
48      if(ivt <= IDT_LAST_EXCEPTION) {
49          printk("EXCEPT: %u cr2=0x%x errcode=0x%x eip=0x%x\n", ivt, get_cr2(), errcode, eip);
50
51          /* never returns */
52          panic("caught exception");
53      }
54
55      if(ivt == SYSCALL_IRQ) {
56          /* slow system call method */
57          dispatch_syscall(trapframe);
58      }
59      else if(ivt >= IDT_PIC8259_BASE && ivt < IDT_PIC8259_BASE +
60      PIC8259_IRQ_COUNT) {
61          int irq = ivt - IDT_PIC8259_BASE;
62          printk("IRQ: %u (vector %u)\n", irq, ivt);
63          pic8259_eoi(irq);
64      }
65      else {
66          printk("INTR: vector %u\n", ivt);
67      }
68  }

```



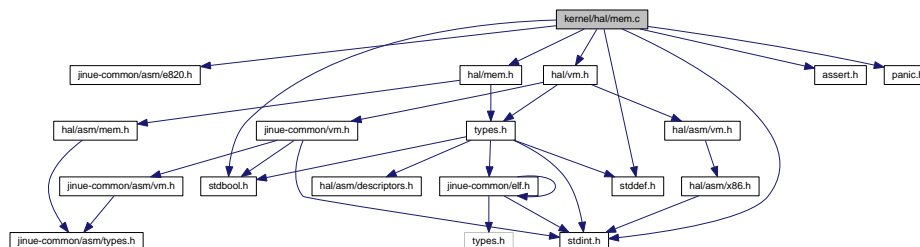
Here is the call graph for this function:



## 4.91 kernel/hal/mem.c File Reference

```
#include <jinue-common/asm/e820.h>
#include <hal/mem.h>
#include <hal/vm.h>
#include <assert.h>
#include <panic.h>
#include <stdbool.h>
#include <stddef.h>
#include <stdint.h>
```

Include dependency graph for mem.c:



## Functions

- void **mem\_check\_memory** (**boot\_alloc\_t** \*boot\_alloc, const **boot\_info\_t** \*boot\_info)

### 4.91.1 Function Documentation

#### 4.91.1.1 void mem\_check\_memory ( boot\_alloc\_t \* boot\_alloc, const boot\_info\_t \* boot\_info )

ASSERTION: Any unsigned value less than MEM\_ZONE\_MEM32\_END can be stored in 32 bits.

Definition at line 57 of file mem.c.

References e820\_t::addr, assert, boot\_info\_t::boot\_end, boot\_info\_t::e820\_entries, boot\_info\_t::e820\_map, E820\_RAM, EARLY\_VIRT\_TO\_PHYS, GB, KERNEL\_EARLY\_LIMIT, boot\_alloc\_t::kernel\_paddr\_limit, boot\_alloc\_t::kernel\_paddr\_top, boot\_alloc\_t::kernel\_vm\_limit, boot\_alloc\_t::kernel\_vm\_top, MEM\_ZONE\_DMA16\_END, MEM\_ZONE\_DMA16\_START, MEM\_ZONE\_MEM32\_END, MEM\_ZONE\_MEM32\_START, panic(), boot\_info\_t::ramdisk\_size, boot\_info\_t::ramdisk\_start, and e820\_t::type.

Referenced by kmain().

```

57                                     {
58     int idx;
59
60     #if 0
61     if((uint64_t)boot_info->ramdisk_start + boot_info->ramdisk_size >
        MEM_ZONE_MEM32_END) {
62         panic("Initial RAM disk loaded too high in memory.");
63     }
64
65     uint32_t ramdisk_end = boot_info->ramdisk_start + boot_info->ramdisk_size;
66 #endif
67
68     /* -----
69     * We consult the memory map provided by the BIOS to figure out how much
70     * memory is available in both zones usable by the kernel. We also want to
71     * make sure the initial RAM disk image is in available RAM.
72     *
73     * The first step in accomplishing this is to iterate over all entries that
74     * are reported as available RAM and confirm at least one of them covers
75     * each of both zones (at least the start) and the initial RAM disk.
76     * ----- */
77     uint32_t zone_dmal6_top = 0;
78     uint32_t zone_mem32_top = 0;
79     #if 0
80     bool ramdisk_ok          = false;
81 #endif
82
83     assert(MEM_ZONE_MEM32_END < (uint64_t)4 * GB);
84
85     for(idx = 0; idx < boot_info->e820_entries; ++idx) {
86         const e820_t *entry = &boot_info->e820_map[idx];
87
88         /* Consider only usable RAM entries. */
89         if(entry->type != E820_RAM) {
90             continue;
91         }
92
93         /* Ignore entries that start past MEM_ZONE_MEM32_END since the kernel
94          * cannot use them. Past this check, entry->addr is assumed to be
95          * representable in 32 bits. */
96         if(entry->addr >= MEM_ZONE_MEM32_END) {
97             continue;
98         }
99
100         uint32_t entry_end = clip_e820_entry_end(entry);
101
102         /* If this entry covers the start the DMA16 zone, adjust the top pointer
103          * accordingly. Overlapping entries are resolved in favor of the largest
104          * entry. */
105         if(entry->addr <= MEM_ZONE_DMA16_START && entry_end >
            MEM_ZONE_DMA16_START) {
106             /* This condition covers the initial case where zone_dmal6_top is zero. */

```

```

108         if(entry_end > zone_dma16_top) {
109             zone_dma16_top = entry_end;
110         }
111     }
112
113     /* Do the same for the MEM32 zone. */
114     if(entry->addr <= MEM_ZONE_MEM32_START && entry_end >
MEM_ZONE_MEM32_START) {
115         if(entry_end > zone_mem32_top) {
116             zone_mem32_top = entry_end;
117         }
118     }
119
120     /* If this entry covers the initial RAM disk, this is good argument in
121      * favor of it being in available RAM (one more check below). Unlike the
122      * above, the entry must cover the initial RAM disk image completely,
123      * not just the start. */
124 #if 0
125     if(entry->addr <= boot_info->ramdisk_start && entry_end >= ramdisk_end) {
126         ramdisk_ok = true;
127     }
128 #endif
129 }
130
131 /* -----
132  * Next, iterate over non-available RAM entries of the map to ensure nothing
133  * is relevant there.
134  * ----- */
135 for(idx = 0; idx < boot_info->e820_entries; ++idx) {
136     const e820_t *entry = &boot_info->e820_map[idx];
137
138     /* Consider only non-usable RAM entries. */
139     if(entry->type == E820_RAM) {
140         continue;
141     }
142
143     if(entry->addr >= MEM_ZONE_MEM32_END) {
144         continue;
145     }
146
147     uint32_t entry_end = clip_e820_entry_end(entry);
148
149     if(ranges_overlap(MEM_ZONE_DMA16_START, MEM_ZONE_DMA16_END, entry->addr, entry_end)) {
150         if(entry->addr > MEM_ZONE_DMA16_START) {
151             if(entry->addr < zone_dma16_top) {
152                 zone_dma16_top = entry->addr;
153             }
154         }
155         else {
156             /* This reserved entry covers the start of the zone. */
157             zone_dma16_top = 0;
158         }
159     }
160
161     if(ranges_overlap(MEM_ZONE_MEM32_START, MEM_ZONE_MEM32_END, entry->addr, entry_end)) {
162         if(entry->addr > MEM_ZONE_MEM32_START) {
163             if(entry->addr < zone_mem32_top) {
164                 zone_mem32_top = entry->addr;
165             }
166         }
167         else {
168             zone_mem32_top = 0;
169         }
170     }
171
172     /* Check for overlap with the initial RAM disk. */
173 #if 0
174     if(ranges_overlap(boot_info->ramdisk_start, ramdisk_end, entry->addr, entry_end)) {
175         ramdisk_ok = false;
176     }
177 #endif
178 }
179
180 /* -----
181  * Now that we are done, let's look at the results.
182  * ----- */
183 #if 0
184     if(! ramdisk_ok) {
185         panic("Initial RAM disk was loaded in reserved memory.");
186     }
187 #endif

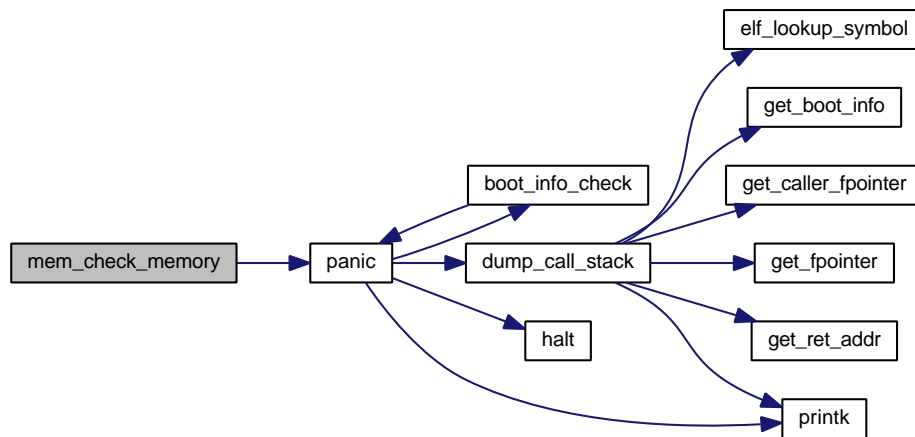
```

```

188
189  /* It is early during the boot process and the page table set up by the
190   * setup code is still being used. This (single) page table maps the first
191   * two megabytes of RAM linearly starting at KLIMIT in the virtual address
192   * space. */
193  boot_alloc->kernel_vm_top      = boot_info->boot_end;
194  boot_alloc->kernel_vm_limit    = (addr_t)KERNEL_EARLY_LIMIT;
195  boot_alloc->kernel_paddr_top   = EARLY_VIRT_TO_PHYS(boot_alloc->
kernel_vm_top);
196  boot_alloc->kernel_paddr_limit = zone_dma16_top;
197
198  if(boot_alloc->kernel_paddr_top > boot_alloc->kernel_paddr_limit) {
199      panic("Kernel image was loaded in reserved memory.");
200  }
201
202  /* TODO Compute sequential allocation limit taking initrd into account */
203  /* TODO Report zone limits */
204 }

```

Here is the call graph for this function:



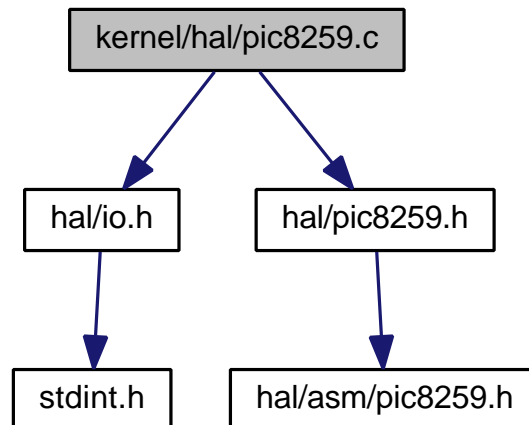
## 4.92 kernel/hal/pic8259.c File Reference

```

#include <hal/io.h>
#include <hal/pic8259.h>

```

Include dependency graph for pic8259.c:



## Functions

- void **pic8259\_init** (int intrvect\_base)
- void **pic8259\_mask\_irq** (int irq)
- void **pic8259\_unmask\_irq** (int irq)
- void **pic8259\_eoi** (int irq)

### 4.92.1 Function Documentation

#### 4.92.1.1 void pic8259\_eoi ( int irq )

Definition at line 124 of file pic8259.c.

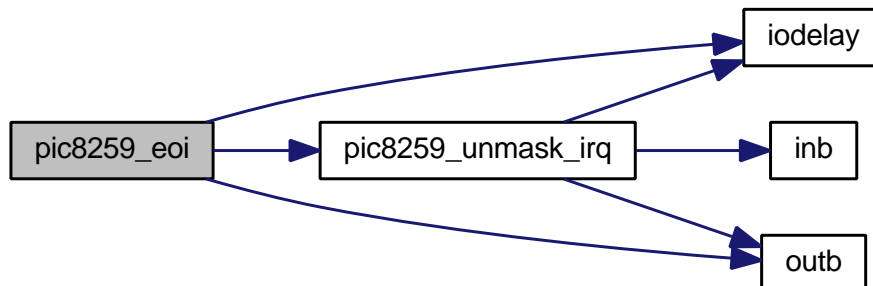
References iodelay(), outb(), PIC8259\_EOI, PIC8259\_IRQ\_COUNT, PIC8259\_MASTER\_BASE, PIC8259\_SLAVE\_BASE, and pic8259\_unmask\_irq().

Referenced by dispatch\_interrupt().

```

124         {
125     if(irq < PIC8259_IRQ_COUNT) {
126         if(irq >= 8) {
127             outb(PIC8259_SLAVE_BASE+0, PIC8259_EOI);
128             iodelay();
129         }
130
131         outb(PIC8259_MASTER_BASE+0, PIC8259_EOI);
132         iodelay();
133
134         pic8259_unmask_irq(irq);
135     }
136 }
```

Here is the call graph for this function:



#### 4.92.1.2 void pic8259\_init ( int intrvect\_base )

Definition at line 36 of file pic8259.c.

References iodelay(), outb(), PIC8259\_CASCADE\_INPUT, PIC8259\_ICW1\_1, PIC8259\_ICW1\_IC4, PIC8259\_ICW4\_UPM, PIC8259\_MASTER\_BASE, and PIC8259\_SLAVE\_BASE.

Referenced by hal\_init().

```

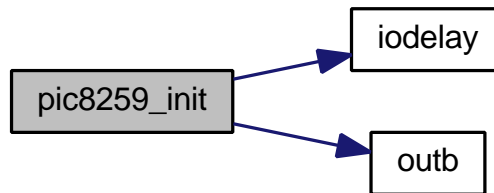
36         {
37     /* Issue ICW1 to start initialization sequence of both interrupt controllers.
38     * Specify there will be an ICW4 in the sequence. Specify the interrupts are
```

```

39      * edge-triggered and that the PICs are in a cascaded configuration by
40      * leaving the relevant flags cleared. */
41      outb(PIC8259_MASTER_BASE+0, PIC8259_ICW1_1 | PIC8259_ICW1_IC4);
42      iodelay();
43      outb(PIC8259_SLAVE_BASE+0, PIC8259_ICW1_1 | PIC8259_ICW1_IC4);
44      iodelay();
45
46      /* ICW2: base interrupt vector */
47      outb(PIC8259_MASTER_BASE+1, intrvect_base);
48      iodelay();
49      outb(PIC8259_SLAVE_BASE+1, intrvect_base + 8);
50      iodelay();
51
52      /* ICW3: master-slave connections */
53      outb(PIC8259_MASTER_BASE+1, (1<<PIC8259_CASCADE_INPUT));
54      iodelay();
55      outb(PIC8259_SLAVE_BASE+1, PIC8259_CASCADE_INPUT);
56      iodelay();
57
58      /* ICW4: Use 8088/8086 mode */
59      outb(PIC8259_MASTER_BASE+1, PIC8259_ICW4_UPM);
60      iodelay();
61      outb(PIC8259_SLAVE_BASE+1, PIC8259_ICW4_UPM);
62      iodelay();
63
64      /* Set interrupt mask: all masked */
65      outb(PIC8259_MASTER_BASE+1, 0xff & ~(1<<PIC8259_CASCADE_INPUT));
66      iodelay();
67      outb(PIC8259_SLAVE_BASE+1, 0xff);
68      iodelay();
69 }

```

Here is the call graph for this function:



#### 4.92.1.3 void pic8259\_mask\_irq ( int irq )

Definition at line 71 of file pic8259.c.

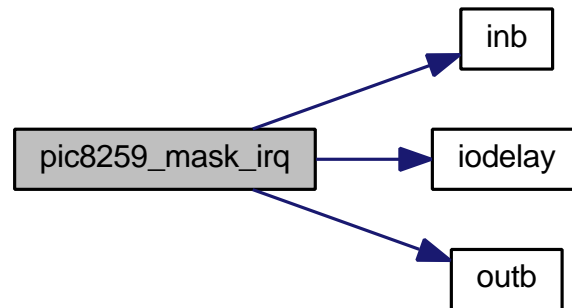
References inb(), iodelay(), outb(), PIC8259\_CASCADE\_INPUT, PIC8259\_IRQ\_COUNT, PIC8259\_MASTER\_BASE, and PIC8259\_SLAVE\_BASE.

```

71      {
72      if(irq < PIC8259_IRQ_COUNT) {
73          if(irq < 8 && irq != PIC8259_CASCADE_INPUT) {
74              int mask = inb(PIC8259_MASTER_BASE+1);
75              iodelay();
76
77              mask |= (1<<irq);
78              outb(PIC8259_MASTER_BASE+1, mask);
79              iodelay();
80          }
81          else {
82              int mask = inb(PIC8259_SLAVE_BASE+1);
83              iodelay();
84
85              mask |= (1<<(irq - 8));
86              outb(PIC8259_SLAVE_BASE+1, mask);
87              iodelay();
88          }
89      }
90 }

```

Here is the call graph for this function:



#### 4.92.1.4 void pic8259\_unmask\_irq ( int irq )

Definition at line 92 of file pic8259.c.

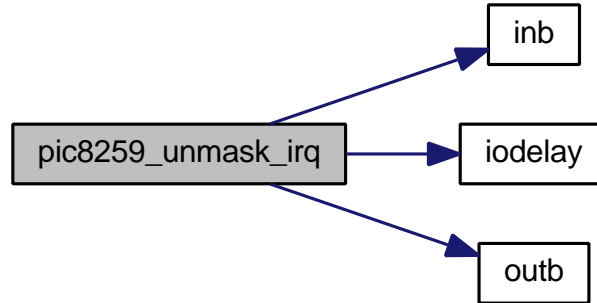
References `inb()`, `iodelay()`, `outb()`, `PIC8259_CASCADE_INPUT`, `PIC8259_IRQ_COUNT`, `PIC8259_MASTER_BASE`, and `PIC8259_SLAVE_BASE`.

Referenced by `pic8259_eoi()`.

```

92     {
93     if(irq < PIC8259_IRQ_COUNT) {
94         int master_irq;
95
96         if(irq < 8) {
97             master_irq = irq;
98         }
99         else {
100             /* Unmask interrupt in slave PIC. */
101             int mask = inb(PIC8259_SLAVE_BASE+1);
102             iodelay();
103
104             mask &= ~(1<<(irq - 8));
105             outb(PIC8259_SLAVE_BASE+1, mask);
106             iodelay();
107
108             /* We will also want to unmask the cascaded interrupt line in the
109              * master PIC. */
110             master_irq = PIC8259_CASCADE_INPUT;
111         }
112
113         if(irq != PIC8259_CASCADE_INPUT) {
114             int mask = inb(PIC8259_MASTER_BASE+1);
115             iodelay();
116
117             mask &= ~(1<<master_irq);
118             outb(PIC8259_MASTER_BASE+1, mask);
119             iodelay();
120         }
121     }
122 }
```

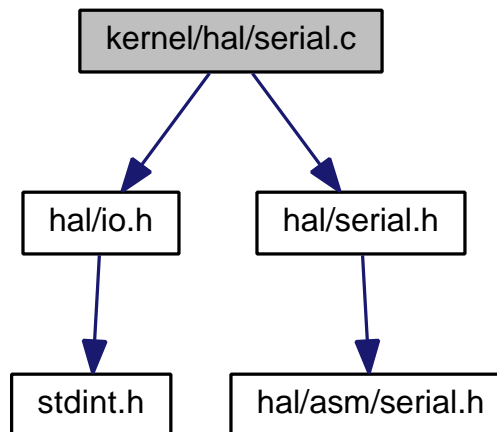
Here is the call graph for this function:



### 4.93 kernel/hal/serial.c File Reference

```
#include <hal/io.h>
#include <hal/serial.h>
```

Include dependency graph for serial.c:



#### Functions

- void **serial\_init** (int base\_ioport, unsigned int baud\_rate)
- void **serial\_printn** (int base\_ioport, const char \*message, unsigned int n)
- void **serial\_putc** (int base\_ioport, char c)

#### 4.93.1 Function Documentation

4.93.1.1 void `serial_init` ( int *base\_ioport*, unsigned int *baud\_rate* )

Definition at line 4 of file `serial.c`.

References `outb()`, `SERIAL_REG_DIVISOR_HIGH`, `SERIAL_REG_DIVISOR_LOW`, `SERIAL_REG_FIFO_CTRL`, `SERIAL_REG_INTR_ENABLE`, `SERIAL_REG_LINE_CTRL`, and `SERIAL_REG_MODEM_CTRL`.

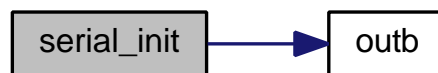


Referenced by console\_init().

```

4      {
5      unsigned int divisor = 115200 / baud_rate;
6
7      /* disable interrupts */
8      outb(base_ioport + SERIAL_REG_INTR_ENABLE, 0);
9
10     /* 8N1, enable DLAB to allow setting baud rate */
11     outb(base_ioport + SERIAL_REG_LINE_CTRL, 0x83);
12
13     /* set baud rate */
14     outb(base_ioport + SERIAL_REG_DIVISOR_LOW, (divisor & 0xff));
15     outb(base_ioport + SERIAL_REG_DIVISOR_HIGH, ((divisor >> 8) & 0xff));
16
17     /* 8N1, disable DLAB */
18     outb(base_ioport + SERIAL_REG_LINE_CTRL, 0x03);
19
20     /* enable and clear FIFO
21     *
22     * Receive FIFO trigger level is not relevant for us as we are only
23     * transmitting. */
24     outb(base_ioport + SERIAL_REG_FIFO_CTRL, 0x07);
25
26     /* assert DTR and RTS */
27     outb(base_ioport + SERIAL_REG_MODEM_CTRL, 0x03);
28 }
```

Here is the call graph for this function:



#### 4.93.1.2 void serial\_printn ( int base\_ioport, const char \* message, unsigned int n )

Definition at line 30 of file serial.c.

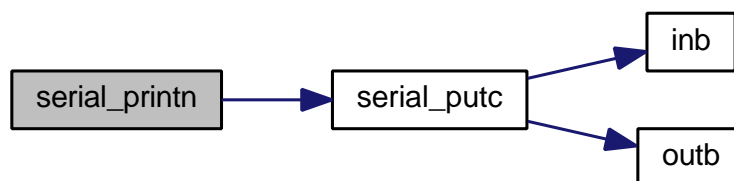
References serial\_putc().

Referenced by console\_printn().

```

30      {
31      int idx;
32
33      for(idx = 0; idx < n; ++idx) {
34          serial_putc(base_ioport, message[idx]);
35      }
36 }
```

Here is the call graph for this function:



#### 4.93.1.3 void serial\_putc ( int base\_ioport, char c )

Definition at line 38 of file serial.c.

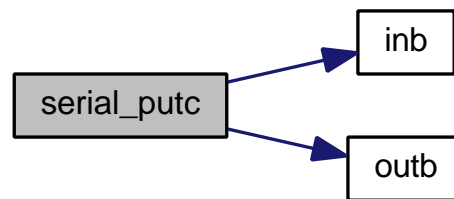
References inb(), outb(), SERIAL\_REG\_DATA\_BUFFER, and SERIAL\_REG\_LINE\_STATUS.

Referenced by console\_putc(), and serial\_printn().

```

38                                     {
39     /* wait for the UART to be ready to accept a new character */
40     while( (inb(base_ioport + SERIAL_REG_LINE_STATUS) & 0x20) == 0) {}
41
42     outb(base_ioport + SERIAL_REG_DATA_BUFFER, c);
43 }
```

Here is the call graph for this function:

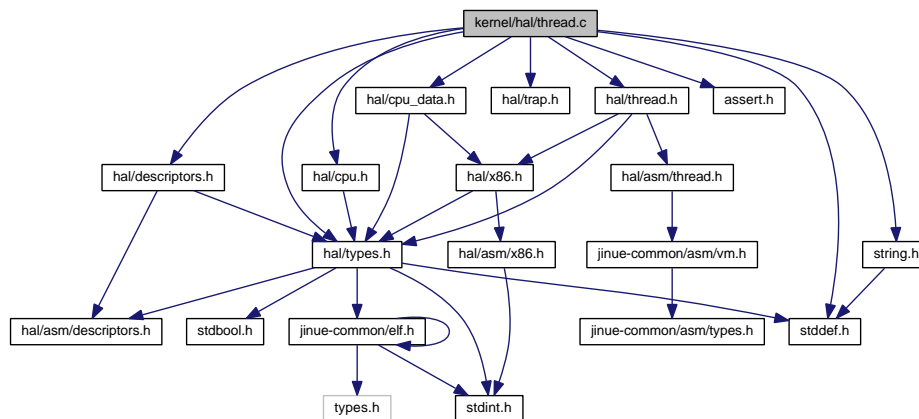


## 4.94 kernel/hal/thread.c File Reference

```

#include <hal/cpu.h>
#include <hal/cpu_data.h>
#include <hal/descriptors.h>
#include <hal/thread.h>
#include <hal/trap.h>
#include <hal/types.h>
#include <assert.h>
#include <stddef.h>
#include <string.h>
```

Include dependency graph for thread.c:



## Functions

- void **thread\_context\_switch\_stack** (**thread\_context\_t** \*from\_ctx, **thread\_context\_t** \*to\_ctx, **bool** destroy\_from)
- **thread\_t** \* **thread\_page\_init** (**addr\_t** thread\_page, **addr\_t** entry, **addr\_t** user\_stack)
- void **thread\_context\_switch** (**thread\_context\_t** \*from\_ctx, **thread\_context\_t** \*to\_ctx, **bool** destroy\_from)

### 4.94.1 Function Documentation

4.94.1.1 void **thread\_context\_switch** ( **thread\_context\_t** \* *from\_ctx*, **thread\_context\_t** \* *to\_ctx*, **bool** *destroy\_from* )

ASSERTION: to\_ctx argument must not be NULL

ASSERTION: from\_ctx argument must not be NULL if destroy\_from is true

Definition at line 122 of file thread.c.

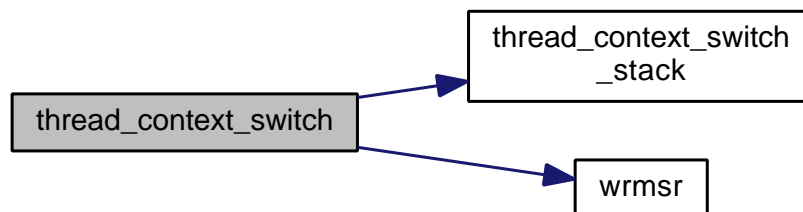
References `assert`, `CPU_FEATURE_SYSENTER`, `tss_t::esp0`, `tss_t::esp1`, `tss_t::esp2`, `MSR_IA32_SYSENTER_ESP`, `NULL`, `thread_context_switch_stack()`, and `wrmsr()`.

Referenced by `thread_switch()`.

```

125                                     {
126
128     assert(to_ctx != NULL);
129
131     assert(from_ctx != NULL || ! destroy_from);
132
133     /* nothing to do if this is already the current thread */
134     if(from_ctx != to_ctx) {
135         /* setup TSS with kernel stack base for this thread context */
136         addr_t kernel_stack_base = get_kernel_stack_base(to_ctx);
137         tss_t *tss = get_tss();
138
139         tss->esp0 = kernel_stack_base;
140         tss->esp1 = kernel_stack_base;
141         tss->esp2 = kernel_stack_base;
142
143         /* update kernel stack address for SYSENTER instruction */
144         if(cpu_has_feature(CPU_FEATURE_SYSENTER)) {
145             wrmsr(MSR_IA32_SYSENTER_ESP, (uint64_t) (uintptr_t) kernel_stack_base);
146         }
147
148         /* switch thread context stack */
149         thread_context_switch_stack(from_ctx, to_ctx, destroy_from);
150     }
151 }
```

Here is the call graph for this function:



4.94.1.2 void **thread\_context\_switch\_stack** ( **thread\_context\_t** \* *from\_ctx*, **thread\_context\_t** \* *to\_ctx*, **bool** *destroy\_from* )

Referenced by `thread_context_switch()`.

#### 4.94.1.3 thread\_t\* thread\_page\_init ( addr\_t thread\_page, addr\_t entry, addr\_t user\_stack )

Definition at line 81 of file thread.c.

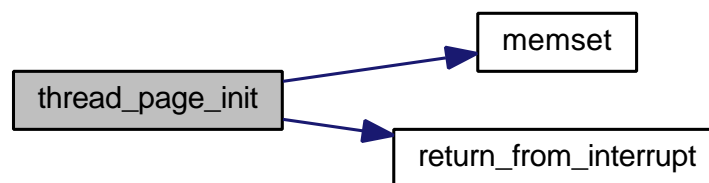
References trapframe\_t::cs, trapframe\_t::ds, trapframe\_t::eflags, trapframe\_t::eip, kernel\_context\_t::eip, trapframe\_t::es, trapframe\_t::esp, trapframe\_t::fs, GDT\_USER\_CODE, GDT\_USER\_DATA, trapframe\_t::gs, thread\_context\_t::local\_storage\_addr, memset(), NULL, return\_from\_interrupt(), RPL\_USER, thread\_context\_t::saved\_stack\_pointer, SEG\_SELECTOR, trapframe\_t::ss, and thread\_t::thread\_ctx.

Referenced by thread\_create(), and thread\_create\_boot().

```

84                                     {
85
86     /* initialize fields */
87     thread_t *thread                = (thread_t *)thread_page;
88     thread_context_t *thread_ctx    = &thread->thread_ctx;
89
90     thread_ctx->local_storage_addr = NULL;
91
92     /* setup stack for initial return to user space */
93     void *kernel_stack_base = get_kernel_stack_base(thread_ctx);
94
95     trapframe_t *trapframe = (trapframe_t *)kernel_stack_base - 1;
96
97     memset(trapframe, 0, sizeof(trapframe_t));
98
99     trapframe->eip      = (uint32_t)entry;
100    trapframe->esp       = (uint32_t)user_stack;
101    trapframe->eflags    = 2;
102    trapframe->cs        = SEG_SELECTOR(GDT_USER_CODE, RPL_USER);
103    trapframe->ss        = SEG_SELECTOR(GDT_USER_DATA, RPL_USER);
104    trapframe->ds        = SEG_SELECTOR(GDT_USER_DATA, RPL_USER);
105    trapframe->es        = SEG_SELECTOR(GDT_USER_DATA, RPL_USER);
106    trapframe->fs        = SEG_SELECTOR(GDT_USER_DATA, RPL_USER);
107    trapframe->gs        = SEG_SELECTOR(GDT_USER_DATA, RPL_USER);
108
109    kernel_context_t *kernel_context = (kernel_context_t *)trapframe - 1;
110
111    memset(kernel_context, 0, sizeof(kernel_context_t));
112
113    /* This is the address to which thread_context_switch_stack() will return. */
114    kernel_context->eip = (uint32_t)return_from_interrupt;
115
116    /* set thread stack pointer */
117    thread_ctx->savd_stack_pointer = (addr_t)kernel_context;
118
119    return thread;
120 }
```

Here is the call graph for this function:

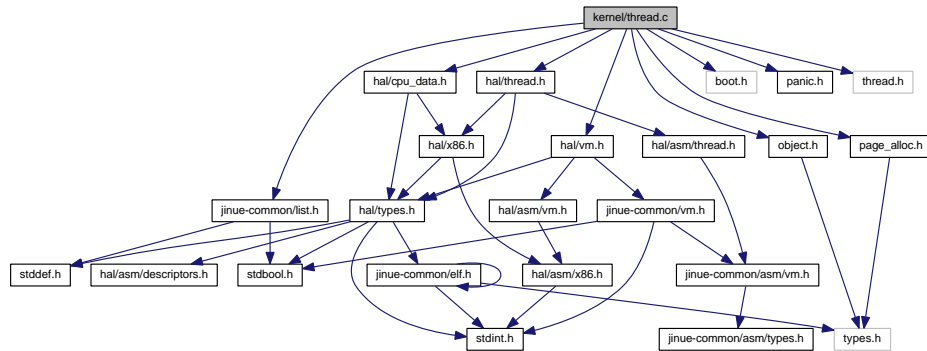


## 4.95 kernel/thread.c File Reference

```
#include <jinue-common/list.h>
```

```
#include <hal/cpu_data.h>
#include <hal/thread.h>
#include <hal/vm.h>
#include <boot.h>
#include <object.h>
#include <page_alloc.h>
#include <panic.h>
#include <thread.h>
```

Include dependency graph for thread.c:



## Functions

- **thread\_t \* thread\_create** (**process\_t** \*process, **addr\_t** entry, **addr\_t** user\_stack)
- **thread\_t \* thread\_create\_boot** (**process\_t** \*process, **addr\_t** entry, **addr\_t** user\_stack, **boot\_alloc\_t** \*boot\_alloc)
- void **thread\_destroy** (**thread\_t** \*thread)
- void **thread\_ready** (**thread\_t** \*thread)
- void **thread\_switch** (**thread\_t** \*from\_thread, **thread\_t** \*to\_thread, **bool** blocked, **bool** do\_destroy)
- void **thread\_yield\_from** (**thread\_t** \*from\_thread, **bool** blocked, **bool** do\_destroy)

### 4.95.1 Function Documentation

#### 4.95.1.1 thread\_t\* thread\_create ( process\_t\* process, addr\_t entry, addr\_t user\_stack )

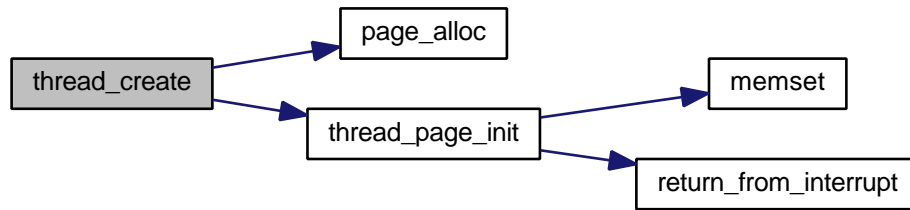
Definition at line 56 of file thread.c.

References NULL, page\_alloc(), and thread\_page\_init().

Referenced by dispatch\_syscall().

```
59                                     {
60
61     void *thread_page = page_alloc();
62
63     if(thread_page == NULL) {
64         return NULL;
65     }
66
67     thread_t *thread = thread_page_init(thread_page, entry, user_stack);
68     thread_init(thread, process);
69
70     return thread;
71 }
```

Here is the call graph for this function:



#### 4.95.1.2 thread\_t\*thread\_create\_boot ( process\_t\*process, addr\_t entry, addr\_t user\_stack, boot\_alloc\_t\* boot\_alloc )

Definition at line 73 of file thread.c.

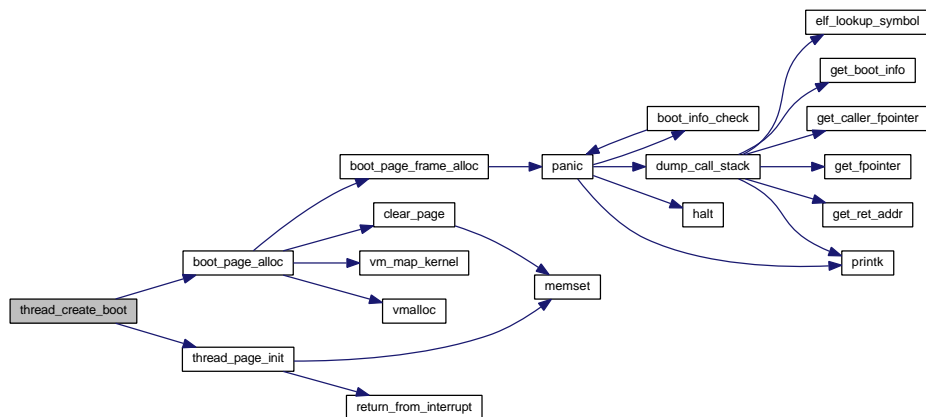
References boot\_page\_alloc(), and thread\_page\_init().

Referenced by kmain().

```

77         {
78
79     /* The kernel panics if this allocation fails. */
80     void *thread_page = boot_page_alloc(boot_alloc);
81     thread_t *thread = thread_page_init(thread_page, entry, user_stack);
82     thread_init(thread, process);
83
84     return thread;
85 }
  
```

Here is the call graph for this function:



#### 4.95.1.3 void thread\_destroy ( thread\_t\* thread )

Definition at line 88 of file thread.c.

References page\_free(), and PAGE\_MASK.

```

88     {
89     void * thread_page = (void *) ((uintptr_t)thread & ~PAGE_MASK);
90     page_free(thread_page);
91 }
  
```

Here is the call graph for this function:



#### 4.95.1.4 void thread\_ready ( thread\_t \* thread )

Definition at line 93 of file thread.c.

References thread\_t::thread\_list.

Referenced by thread\_switch().

```

93      {
94      /* add thread to the tail of the ready list to give other threads a chance
95       * to run */
96      jinue_list_enqueue(&ready_list, &thread->thread_list);
97  }
```

#### 4.95.1.5 void thread\_switch ( thread\_t \* from\_thread, thread\_t \* to\_thread, bool blocked, bool do\_destroy )

Definition at line 99 of file thread.c.

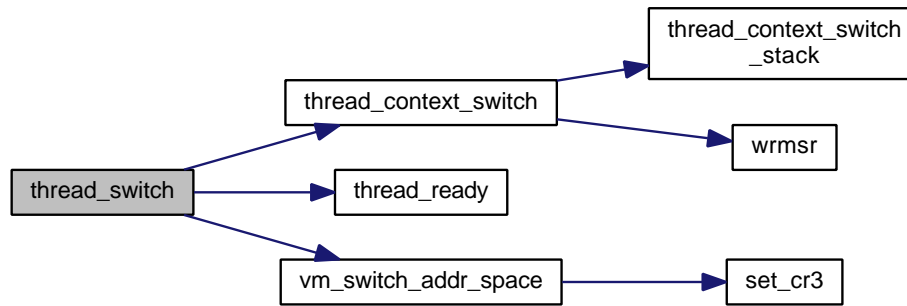
References process\_t::addr\_space, NULL, thread\_t::process, thread\_context\_switch(), thread\_t::thread\_ctx, thread\_ready(), and vm\_switch\_addr\_space().

Referenced by ipc\_receive(), ipc\_reply(), ipc\_send(), and thread\_yield\_from().

```

103      {
104
105      if(to_thread != from_thread) {
106          thread_context_t    *from_context;
107          process_t           *from_process;
108
109          if(from_thread == NULL) {
110              from_context = NULL;
111              from_process = NULL;
112          }
113          else {
114              from_context = &from_thread->thread_ctx;
115              from_process = from_thread->process;
116
117              /* Put the the thread we are switching away from (the current thread)
118               * back into the ready list, unless it just blocked or it is being
119               * destroyed. */
120              if(!(do_destroy || blocked)) {
121                  thread_ready(from_thread);
122              }
123          }
124
125          if(from_process != to_thread->process) {
126              vm_switch_addr_space(
127                  &to_thread->process->addr_space,
128                  get_cpu_local_data()
129              );
130          }
131
132          thread_context_switch(
133              from_context,
134              &to_thread->thread_ctx,
135              do_destroy);
136      }
137  }
```

Here is the call graph for this function:



#### 4.95.1.6 void thread\_yield\_from ( thread\_t \* from\_thread, bool blocked, bool do\_destroy )

Definition at line 163 of file thread.c.

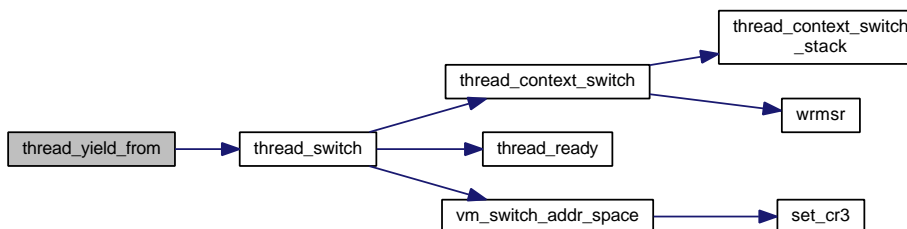
References thread\_switch().

Referenced by dispatch\_syscall(), ipc\_receive(), ipc\_send(), and kmain().

```

163                                     {
164     bool from_can_run = ! (blocked || do_destroy);
165
166     thread_switch(
167         from_thread,
168         reschedule(from_thread, from_can_run),
169         blocked,
170         do_destroy);
171 }
  
```

Here is the call graph for this function:



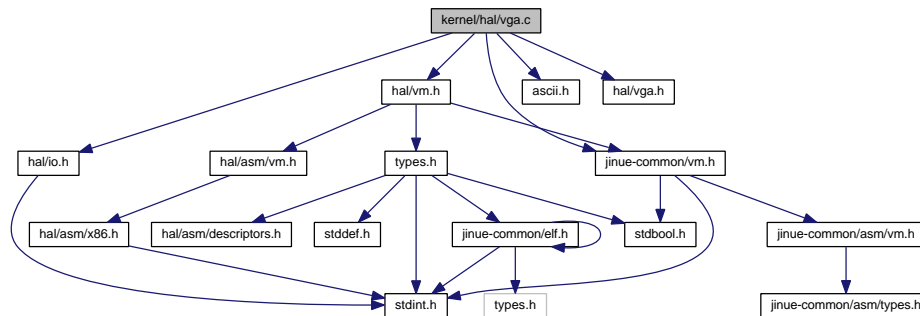
## 4.96 kernel/hal/vga.c File Reference

```

#include <jinue-common/vm.h>
#include <ascii.h>
#include <hal/io.h>
#include <hal/vga.h>
#include <hal/vm.h>
  
```



Include dependency graph for vga.c:



## Functions

- void **vga\_init** (void)
- void **vga\_set\_base\_addr** (void \*base\_addr)
- void **vga\_clear** (void)
- void **vga\_scroll** (void)
- **vga\_pos\_t** **vga\_get\_cursor\_pos** (void)
- void **vga\_set\_cursor\_pos** (**vga\_pos\_t** pos)
- void **vga\_print** (const char \*message, int colour)
- void **vga\_printn** (const char \*message, unsigned int n, int colour)
- void **vga\_putc** (char c, int colour)

### 4.96.1 Function Documentation

#### 4.96.1.1 void vga\_clear ( void )

Definition at line 65 of file vga.c.

References VGA\_COLOR\_ERASE, VGA\_LINES, and VGA\_WIDTH.

Referenced by vga\_init().

```

65     {
66         unsigned int idx = 0;
67
68         while( idx < (VGA_LINES * VGA_WIDTH * 2) ) {
69             video_base_addr[idx++] = 0x20;
70             video_base_addr[idx++] = VGA_COLOR_ERASE;
71         }
72     }

```

#### 4.96.1.2 vga\_pos\_t vga\_get\_cursor\_pos ( void )

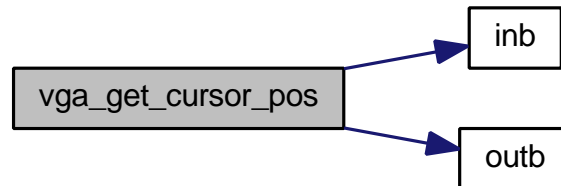
Definition at line 89 of file vga.c.

References inb(), outb(), VGA\_CRTC\_ADDR, and VGA\_CRTC\_DATA.

Referenced by vga\_print(), vga\_printn(), and vga\_putc().

```
89     {
90     unsigned char h, l;
91
92     outb(VGA_CRTC_ADDR, 0x0e);
93     h = inb(VGA_CRTC_DATA);
94     outb(VGA_CRTC_ADDR, 0x0f);
95     l = inb(VGA_CRTC_DATA);
96
97     return (h << 8) | l;
98 }
```

Here is the call graph for this function:



#### 4.96.1.3 void vga\_init ( void )

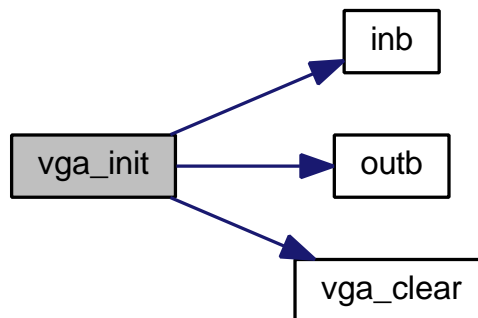
Definition at line 43 of file vga.c.

References `inb()`, `outb()`, `vga_clear()`, `VGA_CRTC_ADDR`, `VGA_CRTC_DATA`, `VGA_MISC_OUT_RD`, and `VGA_MISC_OUT_WR`.

Referenced by `console_init()`.

```
43     {
44     unsigned char data;
45
46     /* Set address select bit in a known state: CRTC regs at 0x3dx */
47     data = inb(VGA_MISC_OUT_RD);
48     data |= 1;
49     outb(VGA_MISC_OUT_WR, data);
50
51     /* Move cursor to line 0 col 0 */
52     outb(VGA_CRTC_ADDR, 0x0e);
53     outb(VGA_CRTC_DATA, 0x0);
54     outb(VGA_CRTC_ADDR, 0x0f);
55     outb(VGA_CRTC_DATA, 0x0);
56
57     /* Clear the screen */
58     vga_clear();
59 }
```

Here is the call graph for this function:



4.96.1.4 void vga\_print ( const char \* *message*, int *colour* )

Definition at line 111 of file vga.c.

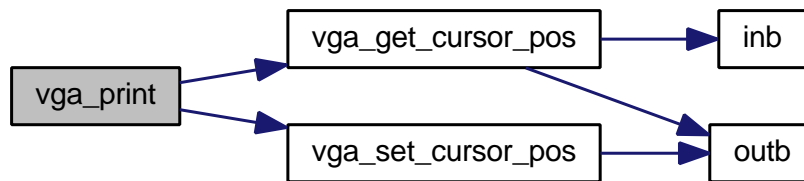
References vga\_get\_cursor\_pos(), and vga\_set\_cursor\_pos().

```

111     {
112         unsigned short int pos = vga_get_cursor_pos();
113         char c;
114
115         while( (c = *(message++)) ) {
116             pos = vga_raw_putc(c, pos, colour);
117         }
118
119         vga_set_cursor_pos(pos);
120     }

```

Here is the call graph for this function:

4.96.1.5 void vga\_printn ( const char \* *message*, unsigned int *n*, int *colour* )

Definition at line 122 of file vga.c.

References vga\_get\_cursor\_pos(), and vga\_set\_cursor\_pos().

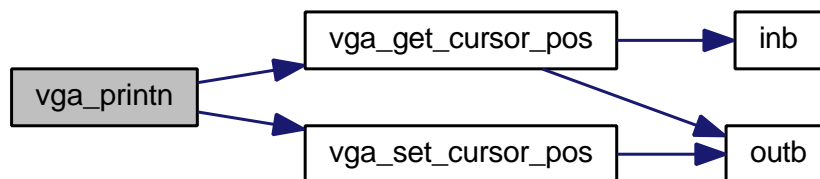
Referenced by console\_printn().

```

122     {
123         vga_pos_t pos = vga_get_cursor_pos();
124         char c;
125
126         while(n) {
127             c = *(message++);
128             pos = vga_raw_putc(c, pos, colour);
129             --n;
130         }
131
132         vga_set_cursor_pos(pos);
133     }

```

Here is the call graph for this function:



#### 4.96.1.6 void vga\_putc ( char *c*, int *colour* )

Definition at line 135 of file vga.c.

References vga\_get\_cursor\_pos(), and vga\_set\_cursor\_pos().

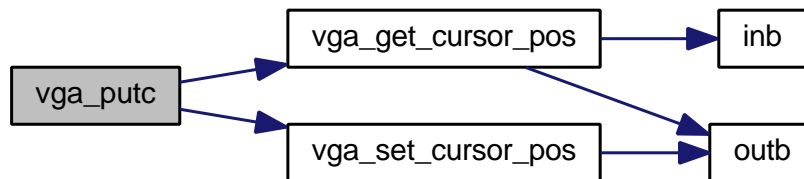
Referenced by console\_putc().

```

135     {
136     vga_pos_t pos = vga_get_cursor_pos();
137
138     pos = vga_raw_putc(c, pos, colour);
139
140     vga_set_cursor_pos(pos);
141 }

```

Here is the call graph for this function:



#### 4.96.1.7 void vga\_scroll ( void )

Definition at line 74 of file vga.c.

References VGA\_COLOR\_ERASE, VGA\_LINES, and VGA\_WIDTH.

```

74     {
75     unsigned char *di = video_base_addr;
76     unsigned char *si = video_base_addr + 2 * VGA_WIDTH;
77     unsigned int idx;
78
79     for(idx = 0; idx < 2 * VGA_WIDTH * (VGA_LINES - 1); ++idx) {
80         *(di++) = *(si++);
81     }
82
83     for(idx = 0; idx < VGA_WIDTH; ++idx) {
84         *(di++) = 0x20;
85         *(di++) = VGA_COLOR_ERASE;
86     }
87 }

```

#### 4.96.1.8 void vga\_set\_base\_addr ( void \* *base\_addr* )

Definition at line 61 of file vga.c.

Referenced by vm\_boot\_init().

```

61     {
62     video_base_addr = base_addr;
63 }

```

## Definition at line 100 of file vga.c.

Referenced by `vga_print()`, `vga_printn()`, and `vga_putc()`.

```
100
101     unsigned char h = pos >> 8;
102     unsigned char l = pos;
103
104     outb(VGA_CRTC_ADDR, 0x0e);
105     outb(VGA_CRTC_DATA, h);
106     outb(VGA_CRTC_ADDR, 0x0f);
107     outb(VGA_CRTC_DATA, l);
108 }
```

```
graph LR; vga_set_cursor_pos --> outb
```

```
#include <hal/boot.h>
#include <hal/cpu_data.h>
#include <hal/vga.h>
#include <hal/vm.h>
#include <hal/vm_private.h>
#include <hal/x86.h>
#include <assert.h>
#include <boot.h>
#include <page_alloc.h>
#include <pfalloc.h>
#include <printk.h>
#include <stdbool.h>
#include <stdint.h>
#include <string.h>
#include <vmalloc.h>
```

## Functions

- void **vm\_boot\_init** (const **boot\_info\_t** \***boot\_info**, **bool** use\_pae, **cpu\_data\_t** \***cpu\_data**, **boot\_alloc\_t** \***boot\_alloc**)
- void **vm\_boot\_postinit** (const **boot\_info\_t** \***boot\_info**, **boot\_alloc\_t** \***boot\_alloc**, **bool** use\_pae)
- void **vm\_map\_kernel** (**addr\_t** vaddr, **kern\_paddr\_t** paddr, int flags)
- void **vm\_map\_user** (**addr\_space\_t** \***addr\_space**, **addr\_t** vaddr, **user\_paddr\_t** paddr, int flags)
- void **vm\_unmap\_kernel** (**addr\_t** addr)
- void **vm\_unmap\_user** (**addr\_space\_t** \***addr\_space**, **addr\_t** addr)
- **kern\_paddr\_t** **vm\_lookup\_kernel\_paddr** (**addr\_t** addr)
- void **vm\_change\_flags** (**addr\_space\_t** \***addr\_space**, **addr\_t** addr, int flags)
- void **vm\_map\_early** (**addr\_t** vaddr, **kern\_paddr\_t** paddr, int flags)
- **kern\_paddr\_t** **vm\_clone\_page\_directory** (**kern\_paddr\_t** template\_paddr, unsigned int start\_index)
- **addr\_space\_t** \* **vm\_create\_addr\_space** (**addr\_space\_t** \***addr\_space**)
- void **vm\_init\_initial\_page\_directory** (**pte\_t** \***page\_directory**, **boot\_alloc\_t** \***boot\_alloc**, unsigned int start\_index, unsigned int end\_index, **bool** first\_directory)
- **addr\_space\_t** \* **vm\_create\_initial\_addr\_space** (**bool** use\_pae, **boot\_alloc\_t** \***boot\_alloc**)
- void **vm\_destroy\_page\_directory** (**kern\_paddr\_t** pgdir\_paddr, unsigned int from\_index, unsigned int to\_index)
- void **vm\_destroy\_addr\_space** (**addr\_space\_t** \***addr\_space**)
- void **vm\_switch\_addr\_space** (**addr\_space\_t** \***addr\_space**, **cpu\_data\_t** \***cpu\_data**)

## Variables

- **pte\_t** \* **global\_page\_tables**
- **addr\_space\_t** **initial\_addr\_space**
- **size\_t** **page\_table\_entries**

### 4.97.1 Function Documentation

4.97.1.1 void **vm\_boot\_init** ( const **boot\_info\_t** \* *boot\_info*, **bool** *use\_pae*, **cpu\_data\_t** \* *cpu\_data*, **boot\_alloc\_t** \* *boot\_alloc* )

below this point, it is no longer safe to call **pfalloc\_early**()

Definition at line 57 of file **vm.c**.

References **boot\_vmalloc**(), **addr\_space\_t::cr3**, **EARLY\_PTR\_TO\_PHYS\_ADDR**, **enable\_pae**(), **boot\_info\_t::image\_start**, **boot\_alloc\_t::its\_early**, **PAGE\_SIZE**, **printk**(), **vga\_set\_base\_addr**(), **VGA\_TEXT\_VID\_BASE**, **VGA\_TEXT\_VID\_TOP**, **vm\_create\_initial\_addr\_space**(), **VM\_FLAG\_KERNEL**, **VM\_FLAG\_READ\_WRITE**, **vm\_map\_early**(), **vm\_pae\_boot\_init**(), **vm\_pae\_unmap\_low\_alias**(), **vm\_switch\_addr\_space**(), and **vm\_x86\_boot\_init**().

Referenced by **hal\_init**().

```

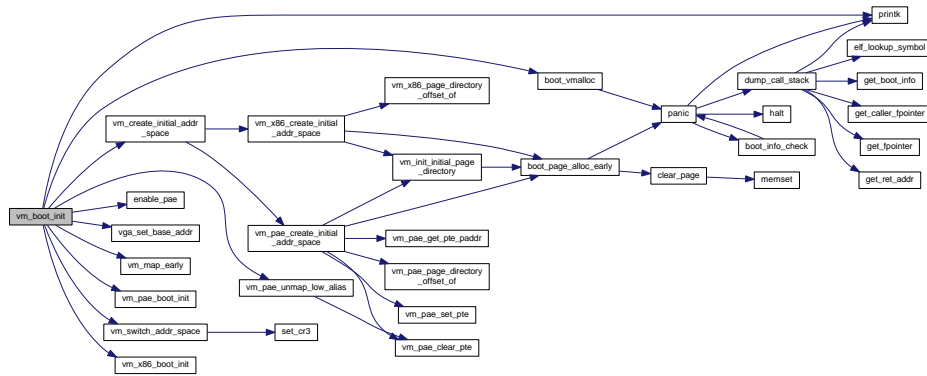
61                                     {
62
63     addr_t addr;
64
65     if (use_pae) {
66         printk("Enabling Physical Address Extension (PAE).\n");
67         vm_pae_boot_init();
68     }
69     else {
70         vm_x86_boot_init();
71     }
72
73     pgtable_format_pae = use_pae;
74
```

```

75  /* create initial address space */
76  addr_space_t *addr_space = vm_create_initial_addr_space(use_pae, boot_alloc);
77
78  boot_alloc->its_early = false;
79
80  /* perform 1:1 mapping of kernel image and data */
81  for(addr = (addr_t)boot_info->image_start; addr < boot_alloc->kernel_vm_top; addr +=
82  PAGE_SIZE) {
83      vm_map_early(addr, EARLY_PTR_TO_PHYS_ADDR(addr), VM_FLAG_KERNEL |
84      VM_FLAG_READ_WRITE);
85  }
86
87  /* map VGA text buffer in the new address space
88  *
89  * This is a good place to do this because:
90  *
91  * 1) It is our last chance to allocate a continuous region of virtual memory.
92  *    Once the page allocator is initialized (see call to vm_alloc_init_allocator()
93  *    below) and we start using vm_alloc() to allocate memory, pages can only
94  *    be allocated one at a time.
95  *
96  * 2) Doing this last makes things simpler because this is the only place where
97  *    we have to allocate a continuous region of virtual memory but no physical
98  *    memory to back it. To allocate it, we just have to increase kernel_vm_top,
99  *    which represents the end of the virtual memory region that is used by the
100  *    kernel. */
101  addr_t vga_text_base;
102  kern_paddr_t paddr = VGA_TEXT_VID_BASE;
103
104  while(paddr < VGA_TEXT_VID_TOP) {
105      /* Pages allocated by successive calls to vmalloc_boot() are guaranteed
106      * to be contiguous. */
107      addr = boot_vmalloc(boot_alloc);
108
109      if(paddr == VGA_TEXT_VID_BASE) {
110          /* First iteration */
111          vga_text_base = addr;
112      }
113
114      vm_map_early(addr, paddr, VM_FLAG_KERNEL | VM_FLAG_READ_WRITE);
115      paddr += PAGE_SIZE;
116  }
117
118  /* remap VGA text buffer
119  *
120  * Note: after the call to vga_set_base_addr() below until we switch to the
121  * new address space, VGA output is not possible. Calling printk() will cause
122  * a kernel panic due to a page fault (and the panic handler calls printk()). */
123  printk("Remapping text video memory at 0x%x\n", vga_text_base);
124
125  vga_set_base_addr(vga_text_base);
126
127  if(use_pae) {
128      /* If we are enabling PAE, this is where the switch to the new page
129      * tables actually happens instead of at the call to vm_switch_addr_space()
130      * as would be expected. */
131      enable_pae(addr_space->cr3);
132
133      /* Now that PAE has been enabled, there is no need to ever disable paging
134      * again, so the low alias for the first 2MB of RAM can be unmapped. This
135      * is only relevant for PAE because, for the non-PAE case, this low alias
136      * is just never set up in the initial address space in the first place,
137      * which means there is no longer a low alias once vm_switch_addr_space()
138      * is called below.
139      *
140      * This call to vm_pae_unmap_low_alias() does not do any TLB invalidation
141      * but this is fine because the call to vm_switch_addr_space() below
142      * reloads CR3.*/
143      vm_pae_unmap_low_alias(addr_space);
144  }
145
146  /* switch to new address space */
147  vm_switch_addr_space(addr_space, cpu_data);
148 }

```

Here is the call graph for this function:



#### 4.97.1.2 void vm\_boot\_postinit ( const boot\_info\_t \* boot\_info, boot\_alloc\_t \* boot\_alloc, bool use\_pae )

Definition at line 149 of file vm.c.

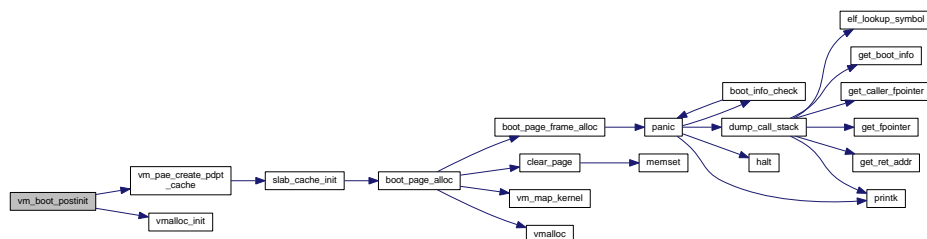
References `KERNEL_IMAGE_END`, `KERNEL_PREALLOC_LIMIT`, `MB`, `vm_pae_create_pdpt_cache()`, and `vmalloc_init()`.

Referenced by `hal_init()`.

```

149
150 /* initialize global page allocator (region starting at KLIMIT)
151  *
152  * TODO Some work needs to be done in the page allocator to support allocating
153  * up to the top of memory (i.e. 0x100000000, which cannot be represented on
154  * 32 bits). In the mean time, we leave a 4MB (one block) gap. */
155 vmalloc_init(
156     (addr_t) KERNEL_IMAGE_END,
157     (addr_t) 0 - 4 * MB,
158     (addr_t) KERNEL_PREALLOC_LIMIT,
159     boot_alloc);
160
161 /* create slab cache to allocate PDPTs
162  *
163  * This must be done after the global page allocator has been initialized
164  * because the slab allocator needs to allocate a slab to allocate the new
165  * slab cache on the slab cache cache.
166  *
167  * This must be done before the first time vm_create_addr_space() is called. */
168 if(use_pae) {
169     vm_pae_create_pdpt_cache(boot_alloc);
170 }
171 }
```

Here is the call graph for this function:





## 4.97.1.3 void vm\_change\_flags ( addr\_space\_t\* addr\_space, addr\_t addr, int flags )

ASSERTION: there is a page table entry marked present for this address

Definition at line 463 of file vm.c.

References `assert`, `invalidate_tlb()`, and `NULL`.

```

463
464     pte_t *pte = vm_lookup_page_table_entry(addr_space, addr, false);
465
466     assert(pte != NULL && (get_pte_flags(pte) & VM_FLAG_PRESENT));
467
468     /* perform the flags change */
469     set_pte_flags(pte, flags | VM_FLAG_PRESENT);
470
471     vm_free_page_table_entry(addr, pte);
472
473     /* invalidate TLB entry for the affected page */
474     invalidate_tlb(addr);
475
476 }
```

Here is the call graph for this function:



## 4.97.1.4 kern\_paddr\_t vm\_clone\_page\_directory ( kern\_paddr\_t template\_paddr, unsigned int start\_index )

Definition at line 489 of file vm.c.

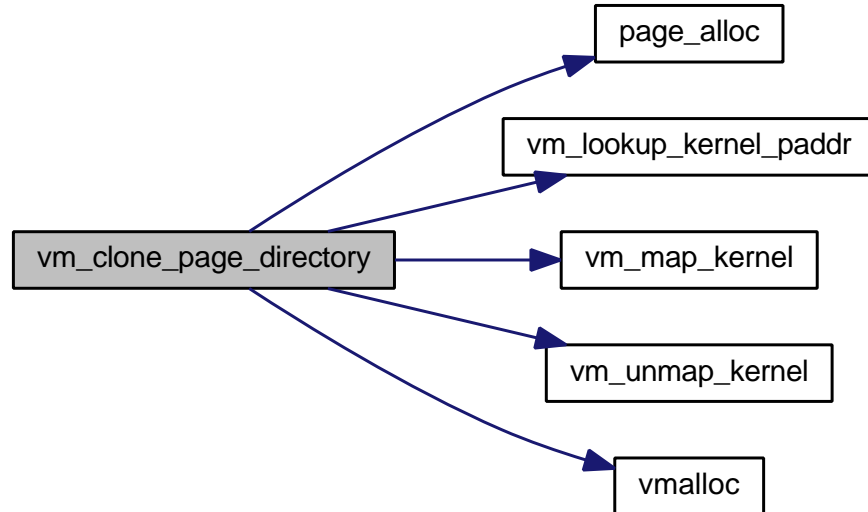
References `page_alloc()`, `page_table_entries`, `VM_FLAG_READ_WRITE`, `vm_lookup_kernel_paddr()`, `vm_map_kernel()`, `vm_unmap_kernel()`, and `vmalloc()`.

Referenced by `vm_x86_create_addr_space()`.

```

489
490     unsigned int    idx;
491
492     /* Allocate new page directory.
493      *
494      * TODO handle allocation failure */
495     pte_t *page_directory = (pte_t *)page_alloc();
496
497     /* map page directory template */
498     pte_t *template = (pte_t *)vmalloc();
499     vm_map_kernel((addr_t)template, template_paddr, VM_FLAG_READ_WRITE);
500
501     /* clear all entries below index start_index */
502     for(idx = 0; idx < start_index; ++idx) {
503         clear_pte( get_pte_with_offset(page_directory, idx) );
504     }
505
506     /* copy entries from template for indexes start_index and above */
507     for(idx = start_index; idx < page_table_entries; ++idx) {
508         copy_pte(
509             get_pte_with_offset(page_directory, idx),
510             get_pte_with_offset(template, idx)
511         );
512     }
513
514     vm_unmap_kernel((addr_t)page_directory);
515     vm_unmap_kernel((addr_t)template);
516
517     return vm_lookup_kernel_paddr((addr_t)page_directory);
518 }
```

Here is the call graph for this function:



#### 4.97.1.5 `addr_space_t* vm_create_addr_space ( addr_space_t* addr_space )`

Definition at line 520 of file `vm.c`.

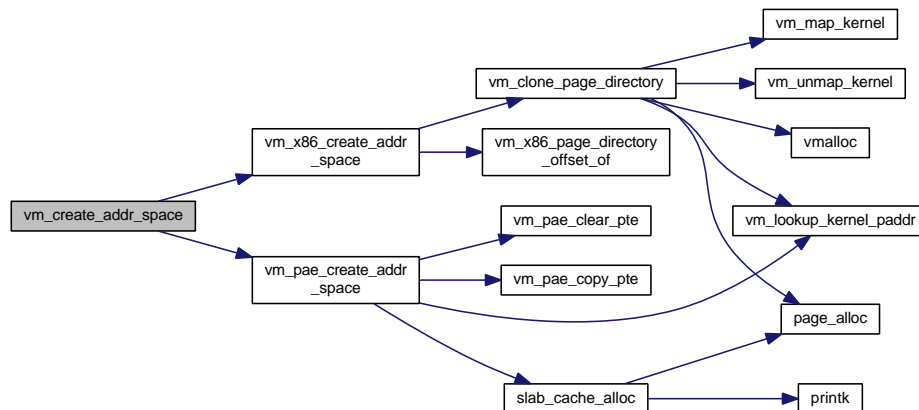
References `vm_pae_create_addr_space()`, and `vm_x86_create_addr_space()`.

Referenced by `process_create()`.

```

520                                     {
521     if(pgtable_format_pae) {
522         return vm_pae_create_addr_space(addr_space);
523     }
524     else {
525         return vm_x86_create_addr_space(addr_space);
526     }
527 }
  
```

Here is the call graph for this function:



4.97.1.6 `addr_space_t* vm_create_initial_addr_space ( bool use_pae, boot_alloc_t* boot_alloc )`

Definition at line 572 of file vm.c.

References `vm_pae_create_initial_addr_space()`, and `vm_x86_create_initial_addr_space()`.

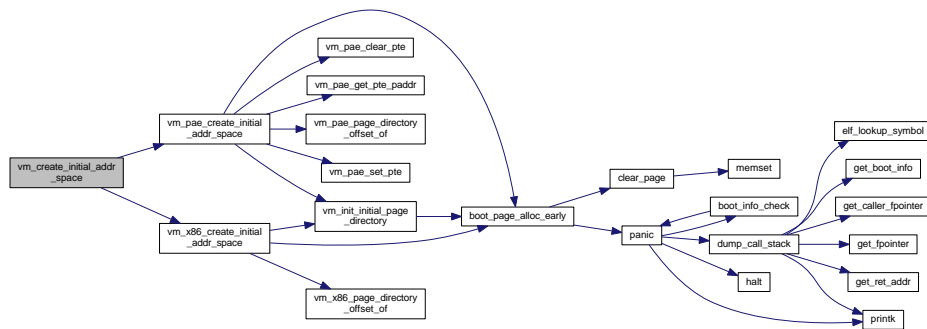
Referenced by `vm_boot_init()`.

```

574                                     {
575
576     if (use_pae) {
577         return vm_pae_create_initial_addr_space (boot_alloc);
578     }
579     else {
580         return vm_x86_create_initial_addr_space (boot_alloc);
581     }
582 }

```

Here is the call graph for this function:

4.97.1.7 `void vm_destroy_addr_space ( addr_space_t* addr_space )`

ASSERTION: address space must not be NULL

ASSERTION: the initial address space should not be destroyed

ASSERTION: the current address space should not be destroyed

Definition at line 603 of file vm.c.

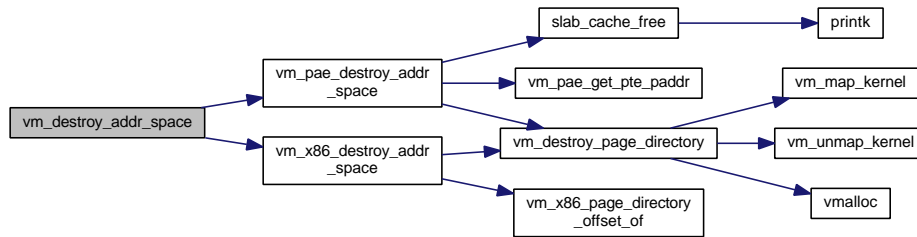
References `assert`, `NULL`, `vm_pae_destroy_addr_space()`, and `vm_x86_destroy_addr_space()`.

```

603                                     {
604
605     assert (addr_space != NULL);
606
607     assert (addr_space != &initial_addr_space);
608
609     assert ( addr_space != get_current_addr_space() );
610
611     if (pgtable_format_pae) {
612         vm_pae_destroy_addr_space (addr_space);
613     }
614     else {
615         vm_x86_destroy_addr_space (addr_space);
616     }
617 }
618
619 }

```

Here is the call graph for this function:



#### 4.97.1.8 void vm\_destroy\_page\_directory ( kern\_paddr\_t pgdir\_paddr, unsigned int from\_index, unsigned int to\_index )

Definition at line 584 of file vm.c.

References pffree, VM\_FLAG\_READ\_WRITE, vm\_map\_kernel(), vm\_unmap\_kernel(), and vmalloc().

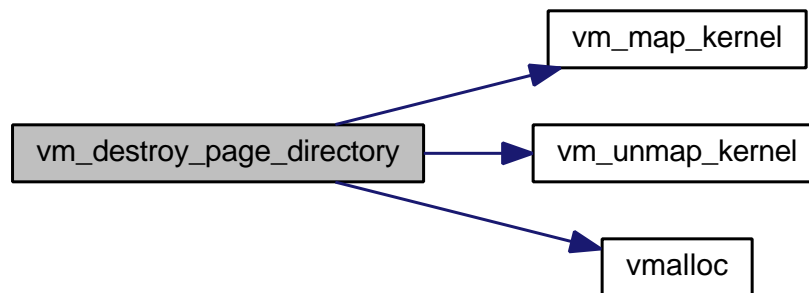
Referenced by vm\_pae\_destroy\_addr\_space(), and vm\_x86\_destroy\_addr\_space().

```

584
585     unsigned int idx;
586
587     pte_t *page_directory = (pte_t *)vmalloc();
588     vm_map_kernel((addr_t)page_directory, pgdir_paddr, VM_FLAG_READ_WRITE);
589
590     /* be careful not to free the kernel page tables */
591     for(idx = from_index; idx < to_index; ++idx) {
592         pte_t *pte = get_pte_with_offset(page_directory, idx);
593
594         if(get_pte_flags(pte) & VM_FLAG_PRESENT) {
595             pffree( get_pte_paddr(pte) );
596         }
597     }
598
599     vm_unmap_kernel((addr_t)page_directory);
600     pffree(pgdir_paddr);
601 }

```

Here is the call graph for this function:



#### 4.97.1.9 void vm\_init\_initial\_page\_directory ( pte\_t \* page\_directory, boot\_alloc\_t \* boot\_alloc, unsigned int start\_index, unsigned int end\_index, bool first\_directory )

Definition at line 529 of file vm.c.

References `boot_page_alloc_early()`, `EARLY_PTR_TO_PHYS_ADDR`, `page_table_entries`, and `VM_FLAG_READ_WRITE`.

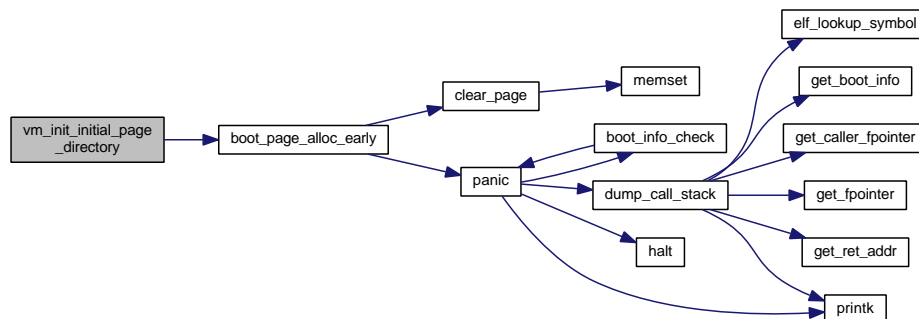
Referenced by `vm_pae_create_initial_addr_space()`, and `vm_x86_create_initial_addr_space()`.

```

534                                     {
535
536     unsigned int idx, idy;
537
538     /* Allocate page tables and initialize page directory entries. */
539     for(idx = 0; idx < page_table_entries; ++idx) {
540         if(idx < start_index || idx >= end_index) {
541             /* Clear page directory entries for user space and non-preallocated
542              * kernel page tables. */
543             clear_pte( get_pte_with_offset(page_directory, idx) );
544         }
545         else {
546             /* Allocate page tables for kernel data/code region.
547              *
548              * Note that the use of pfalloc_early() here guarantees that the
549              * page tables are allocated contiguously, and that they keep the
550              * same address once paging is enabled. */
551             pte_t *page_table = (pte_t *)boot_page_alloc_early(boot_alloc);
552
553             if(first_directory && idx == start_index) {
554                 /* remember the address of the first page table for use by
555                  * vm_map() later */
556                 global_page_tables = page_table;
557             }
558
559             set_pte(
560                 get_pte_with_offset(page_directory, idx),
561                 EARLY_PTR_TO_PHYS_ADDR(page_table),
562                 VM_FLAG_PRESENT | VM_FLAG_READ_WRITE);
563
564             /* clear page table */
565             for(idy = 0; idy < page_table_entries; ++idy) {
566                 clear_pte( get_pte_with_offset(page_table, idy) );
567             }
568         }
569     }
570 }

```

Here is the call graph for this function:



#### 4.97.1.10 kern\_paddr\_t vm\_lookup\_kernel\_paddr ( addr\_t addr )

ASSERTION: there is a page table entry marked present for this address

Definition at line 450 of file `vm.c`.

References `assert`, and `NULL`.

Referenced by `hal_init()`, `remove_page_frame()`, `vm_clone_page_directory()`, and `vm_pae_create_addr_space()`.

```

450                                     {
451     pte_t *pte = vm_lookup_page_table_entry(NULL, addr, false);
452
453     assert(pte != NULL && (get_pte_flags(pte) & VM_FLAG_PRESENT));
454
455     kern_paddr_t paddr = (kern_paddr_t) get_pte_paddr(pte);
456
457     vm_free_page_table_entry(addr, pte);
458
459     return paddr;
460 }
461

```

#### 4.97.1.11 void vm\_map\_early ( addr\_t vaddr, kern\_paddr\_t paddr, int flags )

ASSERTION: we are within the mapping set up by the setup code

ASSERTION: we assume vaddr is aligned on a page boundary

Definition at line 478 of file vm.c.

References assert, EARLY\_VIRT\_TO\_PHYS, page\_number\_of, and page\_offset\_of.

Referenced by vm\_boot\_init().

```

478                                     {
479     assert( is_early_pointer(vaddr) );
480
481     assert( page_offset_of(vaddr) == 0 );
482
483     pte_t *pte = get_pte_with_offset(global_page_tables, page_number_of(
484     EARLY_VIRT_TO_PHYS((uintptr_t)vaddr) ));
485     set_pte(pte, paddr, flags | VM_FLAG_PRESENT);
486 }
487

```

#### 4.97.1.12 void vm\_map\_kernel ( addr\_t vaddr, kern\_paddr\_t paddr, int flags )

Definition at line 434 of file vm.c.

References NULL, and VM\_FLAG\_KERNEL.

Referenced by add\_page\_frame(), boot\_page\_alloc(), boot\_page\_alloc\_image(), elf\_setup\_stack(), vm\_clone\_page\_directory(), vm\_destroy\_page\_directory(), vm\_pae\_lookup\_page\_directory(), and vm\_x86\_lookup\_page\_directory().

```

434                                     {
435     vm_map(NULL, vaddr, paddr, flags | VM_FLAG_KERNEL);
436 }
437

```

#### 4.97.1.13 void vm\_map\_user ( addr\_space\_t \* addr\_space, addr\_t vaddr, user\_paddr\_t paddr, int flags )

Definition at line 438 of file vm.c.

References VM\_FLAG\_USER.

Referenced by elf\_load(), and elf\_setup\_stack().

```

438                                     {
439     vm_map(addr_space, vaddr, paddr, flags | VM_FLAG_USER);
440 }
441

```

**4.97.1.14 void vm\_switch\_addr\_space ( addr\_space\_t\* addr\_space, cpu\_data\_t\* cpu\_data )**

Definition at line 621 of file vm.c.

References `addr_space_t::cr3`, `cpu_data_t::current_addr_space`, and `set_cr3()`.

Referenced by `thread_switch()`, and `vm_boot_init()`.

```

621                                     {
622     set_cr3(addr_space->cr3);
623
624     cpu_data->current_addr_space = addr_space;
625 }
```

Here is the call graph for this function:

**4.97.1.15 void vm\_unmap\_kernel ( addr\_t addr )**

Definition at line 442 of file vm.c.

References `NULL`.

Referenced by `elf_setup_stack()`, `remove_page_frame()`, `vm_clone_page_directory()`, and `vm_destroy_page_directory()`.

```

442                                     {
443     vm_unmap(NULL, addr);
444 }
```

**4.97.1.16 void vm\_unmap\_user ( addr\_space\_t\* addr\_space, addr\_t addr )**

Definition at line 446 of file vm.c.

```

446                                     {
447     vm_unmap(addr_space, addr);
448 }
```

**4.97.2 Variable Documentation****4.97.2.1 pte\_t\* global\_page\_tables**

Definition at line 49 of file vm.c.

**4.97.2.2 addr\_space\_t initial\_addr\_space**

Definition at line 51 of file vm.c.

Referenced by `process_create_initial()`, `vm_pae_create_addr_space()`, `vm_pae_create_initial_addr_space()`, `vm_x86_create_addr_space()`, and `vm_x86_create_initial_addr_space()`.

### 4.97.2.3 size\_t page\_table\_entries

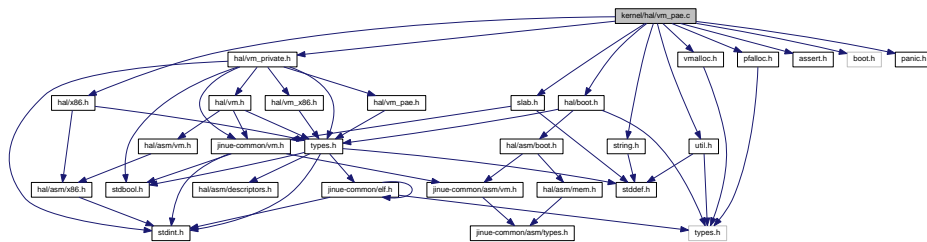
Definition at line 53 of file vm.c.

Referenced by vm\_clone\_page\_directory(), vm\_init\_initial\_page\_directory(), vm\_pae\_boot\_init(), vm\_pae\_create\_initial\_addr\_space(), and vm\_x86\_boot\_init().

## 4.98 kernel/hal/vm\_pae.c File Reference

```
#include <hal/vm_private.h>
#include <hal/boot.h>
#include <hal/x86.h>
#include <assert.h>
#include <boot.h>
#include <panic.h>
#include <pfalloc.h>
#include <slab.h>
#include <string.h>
#include <util.h>
#include <vmalloc.h>
```

Include dependency graph for vm\_pae.c:



## Data Structures

- struct **pte\_t**
- struct **pdpt\_t**

## Macros

- #define **PDPT\_BITS** 2  
*number of address bits that encode the PDPT offset*
- #define **PDPT\_ENTRIES** (1 << PDPT\_BITS)  
*number of entries in a Page Directory Pointer Table (PDPT)*

## Functions

- void **vm\_pae\_boot\_init** (void)  
*This header file contains declarations for the PAE functions defined in **hal/vm\_pae.c** (p. 342).*
- **pte\_t \* vm\_pae\_lookup\_page\_directory** (**addr\_space\_t** \*addr\_space, void \*addr, **bool** create\_as\_needed)  
*Lookup and map the page directory for a specified address and address space.*



- unsigned int **vm\_pae\_page\_table\_offset\_of**(addr\_t addr)
- unsigned int **vm\_pae\_page\_directory\_offset\_of**(addr\_t addr)
- **pte\_t \* vm\_pae\_get\_pte\_with\_offset**(pte\_t \*pte, unsigned int offset)
- void **vm\_pae\_set\_pte**(pte\_t \*pte, uint64\_t paddr, int flags)  
*TODO handle flag bit position > 31 for NX bit support.*
- void **vm\_pae\_set\_pte\_flags**(pte\_t \*pte, int flags)  
*TODO handle flag bit position > 31 for NX bit support.*
- int **vm\_pae\_get\_pte\_flags**(const pte\_t \*pte)
- **uint64\_t vm\_pae\_get\_pte\_paddr**(const pte\_t \*pte)  
*TODO mask NX bit as well, maximum 52 bits supported.*
- void **vm\_pae\_clear\_pte**(pte\_t \*pte)
- void **vm\_pae\_copy\_pte**(pte\_t \*dest, const pte\_t \*src)
- void **vm\_pae\_create\_pdpt\_cache**(boot\_alloc\_t \*boot\_alloc)
- **addr\_space\_t \* vm\_pae\_create\_addr\_space**(addr\_space\_t \*addr\_space)
- **addr\_space\_t \* vm\_pae\_create\_initial\_addr\_space**(boot\_alloc\_t \*boot\_alloc)
- void **vm\_pae\_destroy\_addr\_space**(addr\_space\_t \*addr\_space)
- void **vm\_pae\_unmap\_low\_alias**(addr\_space\_t \*addr\_space)

## Variables

- **pdpt\_t \* initial\_pdpt**

### 4.98.1 Macro Definition Documentation

#### 4.98.1.1 #define PDPT\_BITS 2

number of address bits that encode the PDPT offset

Definition at line 46 of file vm\_pae.c.

#### 4.98.1.2 #define PDPT\_ENTRIES (1 << PDPT\_BITS)

number of entries in a Page Directory Pointer Table (PDPT)

Definition at line 49 of file vm\_pae.c.

Referenced by `vm_pae_create_addr_space()`, `vm_pae_create_initial_addr_space()`, and `vm_pae_destroy_addr_space()`.

### 4.98.2 Function Documentation

#### 4.98.2.1 void vm\_pae\_boot\_init ( void )

This header file contains declarations for the PAE functions defined in **hal/vm\_pae.c** (p. 342).

It is intended to be included by **hal/vm.c** (p. 331) and **hal/vm\_pae.c** (p. 342). There should be no reason to include it anywhere else.

Definition at line 72 of file vm\_pae.c.

References `PAGE_TABLE_ENTRIES`, and `page_table_entries`.

Referenced by `vm_boot_init()`.

```

72         {
73     page_table_entries = (size_t)PAGE_TABLE_ENTRIES;
74 }

```

#### 4.98.2.2 void vm\_pae\_clear\_pte ( pte\_t \* pte )

Definition at line 150 of file vm\_pae.c.

References pte\_t::entry.

Referenced by vm\_pae\_create\_addr\_space(), vm\_pae\_create\_initial\_addr\_space(), and vm\_pae\_unmap\_low\_alias().

```

150         {
151     pte->entry = 0;
152 }

```

#### 4.98.2.3 void vm\_pae\_copy\_pte ( pte\_t \* dest, const pte\_t \* src )

Definition at line 154 of file vm\_pae.c.

References pte\_t::entry.

Referenced by vm\_pae\_create\_addr\_space().

```

154         {
155     dest->entry = src->entry;
156 }

```

#### 4.98.2.4 addr\_space\_t \* vm\_pae\_create\_addr\_space ( addr\_space\_t \* addr\_space )

Definition at line 170 of file vm\_pae.c.

References addr\_space\_t::cr3, initial\_addr\_space, KLIMIT, NULL, page\_address\_of, page\_offset\_of, pdpt\_t::pd, addr\_space\_t::pdpt, PDPT\_ENTRIES, slab\_cache\_alloc(), addr\_space\_t::top\_level, vm\_lookup\_kernel\_paddr(), vm\_pae\_clear\_pte(), and vm\_pae\_copy\_pte().

Referenced by vm\_create\_addr\_space().

```

170         {
171     unsigned int idx;
172     pte_t *pdpte;
173
174     /* Create a PDPT for the new address space */
175     pdpt_t *pdpt = slab_cache_alloc(&pdpt_cache);
176
177     if(pdpt == NULL) {
178         return NULL;
179     }
180
181     /* Use the initial address space as a template for the kernel address range
182      * (address KLIMIT and above). The page tables for that range are shared by
183      * all address spaces. */
184     pdpt_t *template_pdpt = initial_addr_space.top_level.pdpt;
185
186     for(idx = 0; idx < PDPT_ENTRIES; ++idx) {
187         pdpte = &pdpt->pd[idx];
188
189         if(idx < pdpt_offset_of((addr_t)KLIMIT)) {
190             /* This PDPT entry describes an address range entirely under KLIMIT
191              * so it is all user space: do not create a page directory at this
192              * time. */
193             vm_pae_clear_pte(pdpte);
194         }

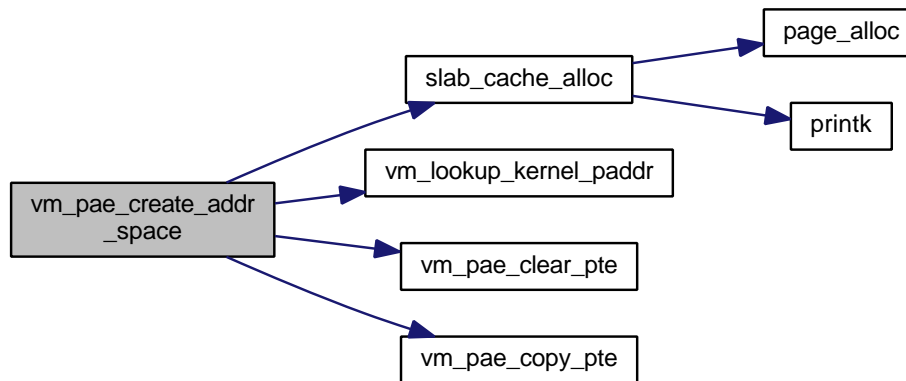
```

```

195     else {
196         /* This page directory describes an address range entirely above
197          * KLIMIT: share the template's page directory. */
198         vm_pae_copy_pte(pdpte, &template_pdpt->pd[idx]);
199     }
200 }
201
202 /* Lookup the physical address of the page where the PDPT resides. */
203 kern_paddr_t pdpt_page_paddr = vm_lookup_kernel_paddr((addr_t)
page_address_of(pdpt));
204
205 /* physical address of PDPT */
206 kern_paddr_t pdpt_paddr = pdpt_page_paddr | page_offset_of(pdpt);
207
208 addr_space->top_level.pdpt = pdpt;
209 addr_space->cr3 = pdpt_paddr;
210
211 return addr_space;
212 }

```

Here is the call graph for this function:



#### 4.98.2.5 addr\_space\_t\* vm\_pae\_create\_initial\_addr\_space ( boot\_alloc\_t\* boot\_alloc )

Definition at line 258 of file vm\_pae.c.

References boot\_heap\_alloc, boot\_page\_alloc\_early(), addr\_space\_t::cr3, EARLY\_PHYS\_TO\_VIRT, EARLY\_PTR\_TO\_PHYS\_ADDR, EARLY\_VIRT\_TO\_PHYS, initial\_addr\_space, initial\_pdpt, KERNEL\_PREALLOC\_LIMIT, KLIMIT, page\_table\_entries, pdpt\_t::pd, addr\_space\_t::pdpt, PDPT\_ENTRIES, addr\_space\_t::top\_level, VM\_FLAG\_PRESENT, vm\_init\_initial\_page\_directory(), vm\_pae\_clear\_pte(), vm\_pae\_get\_pte\_paddr(), vm\_pae\_page\_directory\_offset\_of(), and vm\_pae\_set\_pte().

Referenced by vm\_create\_initial\_addr\_space().

```

258 {
259
260     unsigned int idx;
261
262     /* Allocate initial PDPT. PDPT must be 32-byte aligned. */
263     initial_pdpt = boot_heap_alloc(boot_alloc, pdpt_t, 32);
264
265     /* We want the pre-allocated kernel page tables to be contiguous. For this
266      * reason, we allocate the page directories first, and then the page tables.
267      *
268      * This function allocates pages in this order:
269      * +-----+-----+-----+-----+
270      * | Low alias | pre-allocated | pre-allocated |
271      * | page directory | kernel | kernel |
272      * | and page table | page directories | page tables |
273      * +-----+-----+-----+-----+

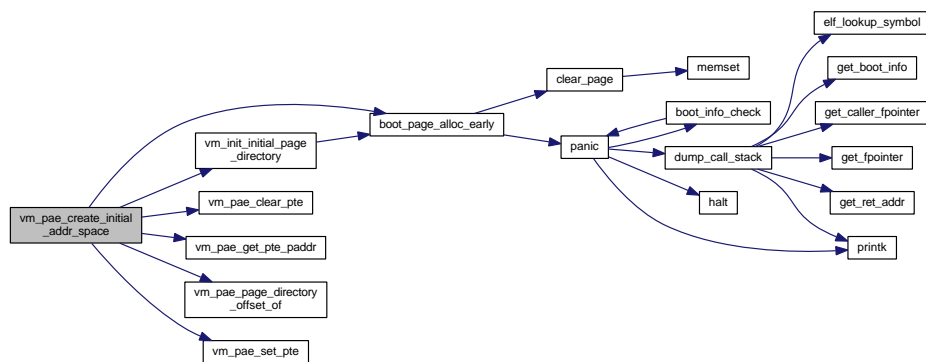
```

```

274  * */
275
276  for(idx = 0; idx < PDPT_ENTRIES; ++idx) {
277      vm_pae_clear_pte(&initial_pdpt->pd[idx]);
278  }
279
280  vm_pae_init_low_alias(initial_pdpt, boot_alloc);
281
282  const unsigned int last_idx = pdpt_offset_of((addr_t)KERNEL_PREALLOC_LIMIT - 1);
283
284  for(idx = pdpt_offset_of((addr_t)KLIMIT); idx <= last_idx; ++idx) {
285      pte_t *const pdpte = &initial_pdpt->pd[idx];
286      pte_t *page_directory = (pte_t *)boot_page_alloc_early(boot_alloc);
287
288      vm_pae_set_pte(
289          pdpte,
290          EARLY_PTR_TO_PHYS_ADDR(page_directory),
291          VM_FLAG_PRESENT);
292  }
293
294  for(idx = pdpt_offset_of((addr_t)KLIMIT); idx <= last_idx; ++idx) {
295      unsigned int end_index;
296
297      pte_t *const pdpte = &initial_pdpt->pd[idx];
298      pte_t *const page_directory = (pte_t *)EARLY_PHYS_TO_VIRT(
299  vm_pae_get_pte_paddr(pdpte));
300
301      if(idx < pdpt_offset_of((addr_t)KERNEL_PREALLOC_LIMIT)) {
302          end_index = page_table_entries;
303      }
304      else {
305          end_index = vm_pae_page_directory_offset_of((addr_t)KERNEL_PREALLOC_LIMIT);
306      }
307
308      vm_init_initial_page_directory(
309          page_directory,
310          boot_alloc,
311          0,
312          end_index,
313          idx == pdpt_offset_of((addr_t)KLIMIT));
314  }
315
316  initial_addr_space.top_level.pdpt = initial_pdpt;
317  initial_addr_space.cr3 = EARLY_VIRT_TO_PHYS(initial_pdpt);
318
319  return &initial_addr_space;
320 }

```

Here is the call graph for this function:



#### 4.98.2.6 void vm\_pae\_create\_pdpt\_cache ( boot\_alloc\_t \* boot\_alloc )

Definition at line 158 of file vm\_pae.c.

References NULL, slab\_cache\_init(), and SLAB\_DEFAULTS.

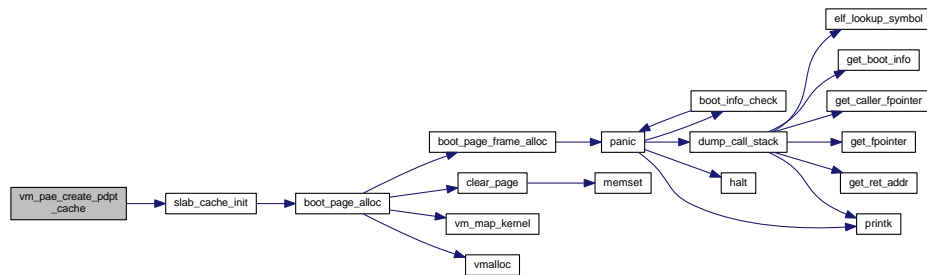
Referenced by `vm_boot_postinit()`.

```

158                                     {
159     slab_cache_init(
160         &pdpt_cache,
161         "vm_pae_pdpt_cache",
162         sizeof(pdpt_t),
163         sizeof(pdpt_t),
164         NULL,
165         NULL,
166         SLAB_DEFAULTS,
167         boot_alloc);
168 }

```

Here is the call graph for this function:



#### 4.98.2.7 void vm\_pae\_destroy\_addr\_space ( addr\_space\_t \* addr\_space )

Definition at line 321 of file `vm_pae.c`.

References `pte_t::entry`, `KLIMIT`, `pdpt_t::pd`, `addr_space_t::pdpt`, `PDPT_ENTRIES`, `slab_cache_free()`, `addr_space_t::top_level`, `vm_destroy_page_directory()`, `VM_FLAG_PRESENT`, and `vm_pae_get_pte_paddr()`.

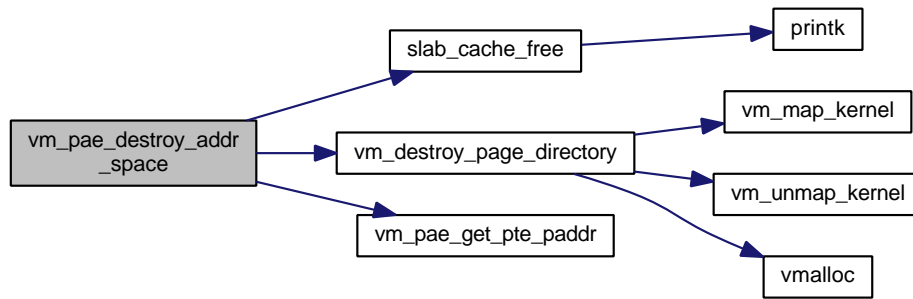
Referenced by `vm_destroy_addr_space()`.

```

321                                     {
322     unsigned int idx;
323     pte_t pdpte;
324
325     pdpt_t *pdpt = addr_space->top_level.pdpt;
326
327     for(idx = 0; idx < PDPT_ENTRIES; ++idx) {
328         pdpte.entry = pdpt->pd[idx].entry;
329
330         if(idx < pdpt_offset_of((addr_t)KLIMIT)) {
331             /* This page directory describes an address range entirely under
332              * KLIMIT so it is all user space: free all page tables in this
333              * page directory as well as the page directory itself. */
334             if(pdpte.entry & VM_FLAG_PRESENT) {
335                 vm_destroy_page_directory(
336                     vm_pae_get_pte_paddr(&pdpte),
337                     0,
338                     page_table_entries);
339             }
340         }
341         else {
342             /* This page directory describes an address range entirely above
343              * KLIMIT: do nothing.
344              *
345              * The page directory must not be freed because it is shared by all
346              * address spaces. */
347         }
348     }
349     slab_cache_free(pdpt);
350 }
351 }

```

Here is the call graph for this function:



#### 4.98.2.8 int vm\_pae\_get\_pte\_flags ( const pte\_t \* pte )

Definition at line 141 of file vm\_pae.c.

References pte\_t::entry, and PAGE\_MASK.

Referenced by vm\_pae\_lookup\_page\_directory().

```

141                                     {
142     return pte->entry & PAGE_MASK;
143 }
```

#### 4.98.2.9 uint64\_t vm\_pae\_get\_pte\_paddr ( const pte\_t \* pte )

TODO mask NX bit as well, maximum 52 bits supported.

Definition at line 146 of file vm\_pae.c.

References pte\_t::entry, and PAGE\_MASK.

Referenced by vm\_pae\_create\_initial\_addr\_space(), vm\_pae\_destroy\_addr\_space(), and vm\_pae\_lookup\_page\_directory().

```

146                                     {
147     return (pte->entry & ~(uint64_t)PAGE_MASK);
148 }
```

#### 4.98.2.10 pte\_t\* vm\_pae\_get\_pte\_with\_offset ( pte\_t \* pte, unsigned int offset )

Definition at line 127 of file vm\_pae.c.

```

127                                     {
128     return &pte[offset];
129 }
```

#### 4.98.2.11 pte\_t\* vm\_pae\_lookup\_page\_directory ( addr\_space\_t \* addr\_space, void \* addr, bool create\_as\_needed )

Lookup and map the page directory for a specified address and address space.

Important note: it is the caller's responsibility to unmap and free the returned page directory when it is done with it.

## Parameters

<i>addr_space</i>	address space in which the address is looked up.
<i>addr</i>	address to look up
<i>create_as_need</i>	Whether a page table is allocated if it does not exist

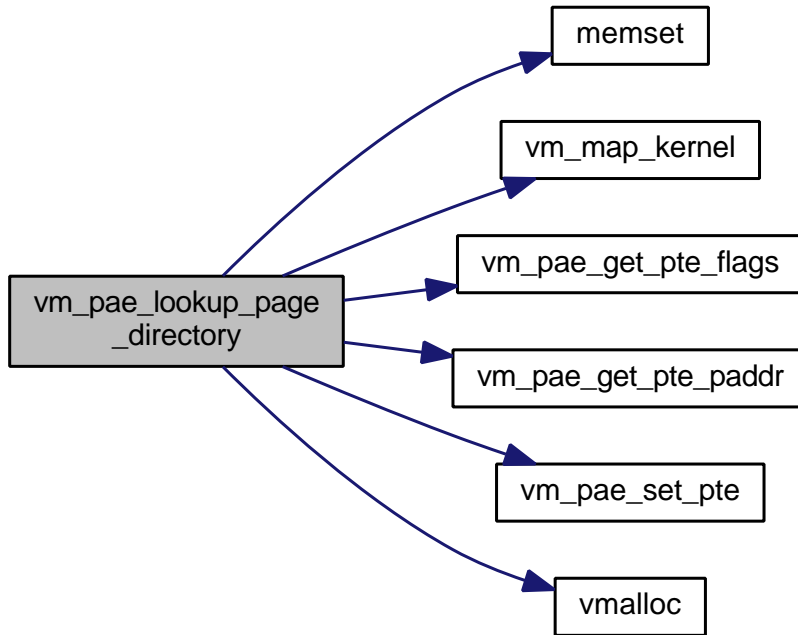
Definition at line 86 of file vm\_pae.c.

References `memset()`, `NULL`, `PAGE_SIZE`, `pdpt_t::pd`, `addr_space_t::pdpt`, `pfalloc`, `addr_space_t::top_level`, `VM_FLAG_PRESENT`, `VM_FLAG_READ_WRITE`, `vm_map_kernel()`, `vm_pae_get_pte_flags()`, `vm_pae_get_pte_paddr()`, `vm_pae_set_pte()`, and `vmalloc()`.

```

86                                     {
87     pdpt_t *pdpt    = addr_space->top_level.pdpt;
88     pte_t *pdpte    = &pdpt->pd[pdpt_offset_of(addr)];
89
90     if (vm_pae_get_pte_flags(pdpte) & VM_FLAG_PRESENT) {
91         /* map page directory */
92         pte_t *page_directory = (pte_t *)vmalloc();
93         vm_map_kernel((addr_t)page_directory, vm_pae_get_pte_paddr(pdpte),
VM_FLAG_READ_WRITE);
94
95         return page_directory;
96     }
97     else {
98         if (create_as_needed) {
99             /* allocate a new page directory and map it */
100             pte_t *page_directory = (pte_t *)vmalloc();
101             kern_paddr_t pgdir_paddr = pfalloc();
102
103             vm_map_kernel((addr_t)page_directory, pgdir_paddr,
VM_FLAG_READ_WRITE);
104
105             /* zero content of page directory */
106             memset(page_directory, 0, PAGE_SIZE);
107
108             /* link page directory in PDPT */
109             vm_pae_set_pte(pdpte, pgdir_paddr, VM_FLAG_PRESENT);
110
111             return page_directory;
112         }
113         else {
114             return NULL;
115         }
116     }
117 }
```

Here is the call graph for this function:



#### 4.98.2.12 unsigned int vm\_pae\_page\_directory\_offset\_of ( addr\_t addr )

Definition at line 123 of file `vm_pae.c`.

References `PAGE_DIRECTORY_OFFSET_OF`.

Referenced by `vm_pae_create_initial_addr_space()`.

```

123                                     {
124     return PAGE_DIRECTORY_OFFSET_OF(addr);
125 }
```

#### 4.98.2.13 unsigned int vm\_pae\_page\_table\_offset\_of ( addr\_t addr )

Definition at line 119 of file `vm_pae.c`.

References `PAGE_TABLE_OFFSET_OF`.

```

119                                     {
120     return PAGE_TABLE_OFFSET_OF(addr);
121 }
```

#### 4.98.2.14 void vm\_pae\_set\_pte ( pte\_t \*pte, uint64\_t paddr, int flags )

TODO handle flag bit position > 31 for NX bit support.

Definition at line 132 of file `vm_pae.c`.

References `pte_t::entry`.

Referenced by `vm_pae_create_initial_addr_space()`, and `vm_pae_lookup_page_directory()`.



```

132                                     {
133     pte->entry = paddr | flags;
134 }

```

#### 4.98.2.15 void vm\_pae\_set\_pte\_flags ( pte\_t \* pte, int flags )

TODO handle flag bit position > 31 for NX bit support.

Definition at line 137 of file vm\_pae.c.

References pte\_t::entry, and PAGE\_MASK.

```

137                                     {
138     pte->entry = (pte->entry & ~(uint64_t)PAGE_MASK) | flags;
139 }

```

#### 4.98.2.16 void vm\_pae\_unmap\_low\_alias ( addr\_space\_t \* addr\_space )

Definition at line 353 of file vm\_pae.c.

References pdpt\_t::pd, addr\_space\_t::pdpt, addr\_space\_t::top\_level, and vm\_pae\_clear\_pte().

Referenced by vm\_boot\_init().

```

353                                     {
354     /* Enabling PAE requires disabling paging temporarily, which in turn requires
355      * an alias of the kernel image region at address 0 to match its physical
356      * address. This function gets rid of this alias once PAE is enabled.
357      *
358      * There is no need for TLB invalidation because the caller reloads CR3 just
359      * after calling this function. */
360     vm_pae_clear_pte(&addr_space->top_level.pdpt->pd[0]);
361 }

```

Here is the call graph for this function:



### 4.98.3 Variable Documentation

#### 4.98.3.1 pdpt\_t \* initial\_pdpt

Definition at line 63 of file vm\_pae.c.

Referenced by vm\_pae\_create\_initial\_addr\_space().

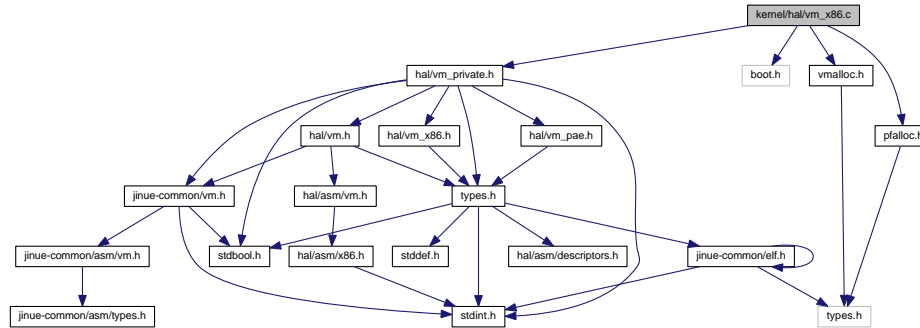
## 4.99 kernel/hal/vm\_x86.c File Reference

```

#include <hal/vm_private.h>
#include <boot.h>
#include <pfalloc.h>
#include <vmalloc.h>

```

Include dependency graph for `vm_x86.c`:



## Data Structures

- struct `pte_t`

## Functions

- void `vm_x86_boot_init` (void)  
*This header file contains declarations for the non-PAE functions defined in `hal/vm_x86.c` (p. 351).*
- `addr_space_t * vm_x86_create_addr_space` (`addr_space_t *addr_space`)
- `addr_space_t * vm_x86_create_initial_addr_space` (`boot_alloc_t *boot_alloc`)
- void `vm_x86_destroy_addr_space` (`addr_space_t *addr_space`)
- unsigned int `vm_x86_page_table_offset_of` (`addr_t addr`)
- unsigned int `vm_x86_page_directory_offset_of` (`addr_t addr`)
- `pte_t * vm_x86_lookup_page_directory` (`addr_space_t *addr_space`)  
*Lookup and map the page directory for a specified address and address space.*
- `pte_t * vm_x86_get_pte_with_offset` (`pte_t *pte`, unsigned int offset)
- void `vm_x86_set_pte` (`pte_t *pte`, `uint32_t paddr`, int flags)
- void `vm_x86_set_pte_flags` (`pte_t *pte`, int flags)
- int `vm_x86_get_pte_flags` (const `pte_t *pte`)
- `uint32_t vm_x86_get_pte_paddr` (const `pte_t *pte`)
- void `vm_x86_clear_pte` (`pte_t *pte`)
- void `vm_x86_copy_pte` (`pte_t *dest`, const `pte_t *src`)

### 4.99.1 Function Documentation

#### 4.99.1.1 void vm\_x86\_boot\_init ( void )

This header file contains declarations for the non-PAE functions defined in `hal/vm_x86.c` (p. 351).

It is intended to be included by `hal/vm.c` (p. 331) and `hal/vm_x86.c` (p. 351). There should be no reason to include it anywhere else.

Definition at line 41 of file `vm_x86.c`.

References `PAGE_TABLE_ENTRIES`, and `page_table_entries`.

Referenced by `vm_boot_init()`.

```

41         {
42     page_table_entries = (size_t)PAGE_TABLE_ENTRIES;
43 }

```

#### 4.99.1.2 void vm\_x86\_clear\_pte ( pte\_t \* pte )

Definition at line 133 of file vm\_x86.c.

References pte\_t::entry.

```

133         {
134     pte->entry = 0;
135 }

```

#### 4.99.1.3 void vm\_x86\_copy\_pte ( pte\_t \* dest, const pte\_t \* src )

Definition at line 137 of file vm\_x86.c.

References pte\_t::entry.

```

137         {
138     dest->entry = src->entry;
139 }

```

#### 4.99.1.4 addr\_space\_t \* vm\_x86\_create\_addr\_space ( addr\_space\_t \* addr\_space )

Definition at line 45 of file vm\_x86.c.

References addr\_space\_t::cr3, initial\_addr\_space, KLIMIT, addr\_space\_t::pd, addr\_space\_t::top\_level, vm\_clone\_page\_directory(), and vm\_x86\_page\_directory\_offset\_of().

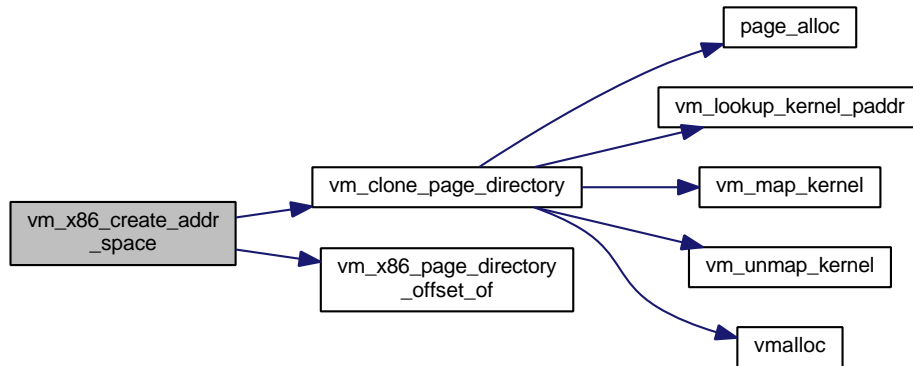
Referenced by vm\_create\_addr\_space().

```

45         {
46     /* Create a new page directory where entries for the address range starting
47     * at KLIMIT are copied from the initial address space. The mappings starting
48     * at KLIMIT belong to the kernel and are identical in all address spaces. */
49     kern_paddr_t paddr = vm_clone_page_directory(
50         initial_addr_space.top_level.pd,
51         vm_x86_page_directory_offset_of((addr_t)KLIMIT));
52
53     addr_space->top_level.pd = paddr;
54     addr_space->cr3 = paddr;
55
56     return addr_space;
57 }

```

Here is the call graph for this function:



#### 4.99.1.5 `addr_space_t* vm_x86_create_initial_addr_space ( boot_alloc_t* boot_alloc )`

Definition at line 59 of file `vm_x86.c`.

References `boot_page_alloc_early()`, `addr_space_t::cr3`, `EARLY_PTR_TO_PHYS_ADDR`, `EARLY_VIRT_TO_PHYS`, `initial_addr_space`, `KERNEL_PREALLOC_LIMIT`, `KLIMIT`, `addr_space_t::pd`, `addr_space_t::top_level`, `vm_init_initial_page_directory()`, and `vm_x86_page_directory_offset_of()`.

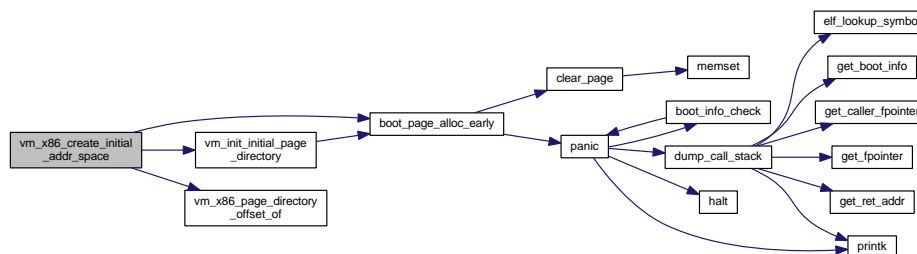
Referenced by `vm_create_initial_addr_space()`.

```

59      {
60      pte_t *page_directory = (pte_t *)boot_page_alloc_early(boot_alloc);
61
62      vm_init_initial_page_directory(
63          page_directory,
64          boot_alloc,
65          vm_x86_page_directory_offset_of((addr_t)KLIMIT),
66          vm_x86_page_directory_offset_of((addr_t)KERNEL_PREALLOC_LIMIT),
67          true);
68
69      initial_addr_space.top_level.pd = EARLY_PTR_TO_PHYS_ADDR(page_directory);
70      initial_addr_space.cr3          = EARLY_VIRT_TO_PHYS((uintptr_t)page_directory);
71
72      return &initial_addr_space;
73  }

```

Here is the call graph for this function:



#### 4.99.1.6 `void vm_x86_destroy_addr_space ( addr_space_t* addr_space )`

Definition at line 75 of file `vm_x86.c`.

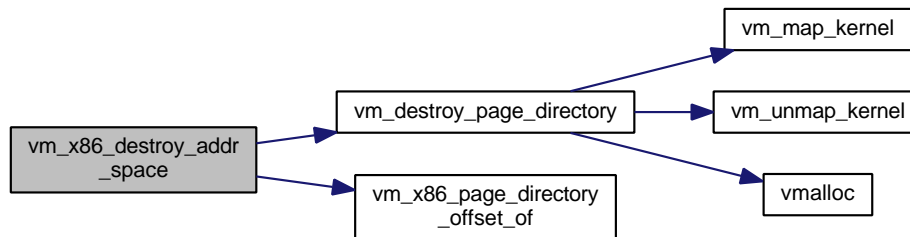
References KLIMIT, addr\_space\_t::pd, addr\_space\_t::top\_level, vm\_destroy\_page\_directory(), and vm\_x86\_page\_directory\_offset\_of().

Referenced by vm\_destroy\_addr\_space().

```

75                                     {
76     vm_destroy_page_directory(
77         addr_space->top_level.pd,
78         /* Free page tables for addresses 0..KLIMIT, be careful not to free
79          * the kernel page tables starting at KLIMIT. */
80         0,
81         vm_x86_page_directory_offset_of((addr_t)KLIMIT));
82 }
```

Here is the call graph for this function:



#### 4.99.1.7 int vm\_x86\_get\_pte\_flags ( const pte\_t \* pte )

Definition at line 125 of file vm\_x86.c.

References pte\_t::entry, and PAGE\_MASK.

```

125                                     {
126     return pte->entry & PAGE_MASK;
127 }
```

#### 4.99.1.8 uint32\_t vm\_x86\_get\_pte\_paddr ( const pte\_t \* pte )

Definition at line 129 of file vm\_x86.c.

References pte\_t::entry, and PAGE\_MASK.

```

129                                     {
130     return pte->entry & ~PAGE_MASK;
131 }
```

#### 4.99.1.9 pte\_t\* vm\_x86\_get\_pte\_with\_offset ( pte\_t \* pte, unsigned int offset )

Definition at line 113 of file vm\_x86.c.

```

113                                     {
114     return &pte[offset];
115 }
```

#### 4.99.1.10 `pte_t* vm_x86_lookup_page_directory ( addr_space_t* addr_space )`

Lookup and map the page directory for a specified address and address space.

This is the implementation for standard 32-bit (i.e. non-PAE) paging. This means that there is only one preallocated page directory, so the `addr` and `create_as_needed` arguments are both irrelevant.

Important note: it is the caller's responsibility to unmap and free the returned page directory when it is done with it.

##### Parameters

<i>addr_space</i>	address space in which the address is looked up.
<i>addr</i>	address to look up
<i>create_as_need</i>	Whether a page table is allocated if it does not exist

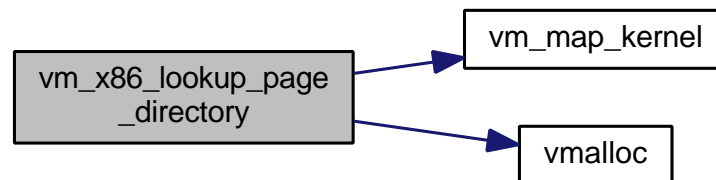
Definition at line 106 of file `vm_x86.c`.

References `addr_space_t::pd`, `addr_space_t::top_level`, `VM_FLAG_READ_WRITE`, `vm_map_kernel()`, and `vmalloc()`.

```

106                                     {
107     pte_t *page_directory = (pte_t *)vmalloc();
108     vm_map_kernel((addr_t)page_directory, addr_space->top_level.pd,
109                 VM_FLAG_READ_WRITE);
110     return page_directory;
111 }
```

Here is the call graph for this function:



#### 4.99.1.11 `unsigned int vm_x86_page_directory_offset_of ( addr_t addr )`

Definition at line 88 of file `vm_x86.c`.

References `PAGE_DIRECTORY_OFFSET_OF`.

Referenced by `vm_x86_create_addr_space()`, `vm_x86_create_initial_addr_space()`, and `vm_x86_destroy_addr_space()`.

```

88                                     {
89     return PAGE_DIRECTORY_OFFSET_OF(addr);
90 }
```

#### 4.99.1.12 `unsigned int vm_x86_page_table_offset_of ( addr_t addr )`

Definition at line 84 of file `vm_x86.c`.

References `PAGE_TABLE_OFFSET_OF`.

```

84                                     {
85     return PAGE_TABLE_OFFSET_OF(addr);
86 }
```

Definition at line 117 of file vm\_x86.c.

```

117                                     {
118     pte->entry = paddr | flags;
119 }

```

Definition at line 121 of file vm\_x86.c.

```

121                                     {
122     pte->entry = (pte->entry & ~PAGE_MASK) | flags;
123 }

```

```
#include <jinue-common/errno.h>
#include <jinue-common/ipc.h>
#include <hal/thread.h>
#include <ipc.h>
#include <object.h>
#include <panic.h>
#include <process.h>
#include <slab.h>
#include <stddef.h>
#include <string.h>
#include <syscall.h>
#include <thread.h>
```

- void **ipc\_boot\_init**(boot\_alloc\_t \*boot\_alloc)
- **ipc\_t** \* **ipc\_object\_create**(int flags)
- **ipc\_t** \* **ipc\_get\_proc\_object**(void)
- void **ipc\_send**(jinue\_syscall\_args\_t \*args)
- void **ipc\_receive**(jinue\_syscall\_args\_t \*args)
- void **ipc\_reply**(jinue\_syscall\_args\_t \*args)

### 4.100.1 Function Documentation

#### 4.100.1.1 void ipc\_boot\_init ( boot\_alloc\_t\* boot\_alloc )

Definition at line 58 of file ipc.c.

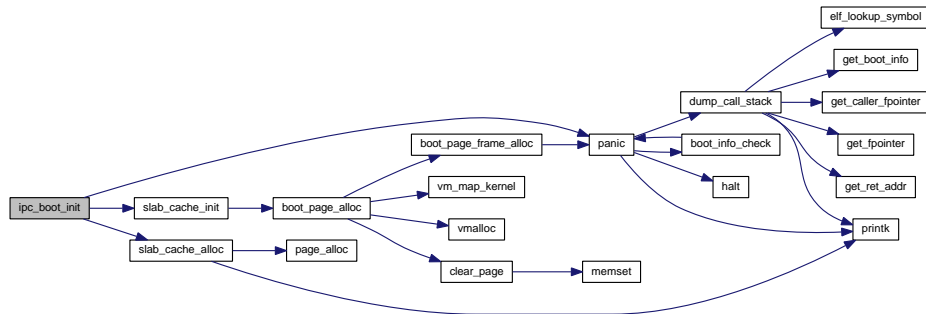
References NULL, panic(), slab\_cache\_alloc(), slab\_cache\_init(), and SLAB\_DEFAULTS.

Referenced by kmain().

```

58                                     {
59     slab_cache_init (
60         &ipc_object_cache,
61         "ipc_object_cache",
62         sizeof(ipc_t),
63         0,
64         ipc_object_ctor,
65         NULL,
66         SLAB_DEFAULTS,
67         boot_alloc);
68
69     proc_ipc = slab_cache_alloc(&ipc_object_cache);
70
71     if(proc_ipc == NULL) {
72         panic("Cannot create process manager IPC object.");
73     }
74 }
```

Here is the call graph for this function:



#### 4.100.1.2 ipc\_t\* ipc\_get\_proc\_object ( void )

Definition at line 86 of file ipc.c.

Referenced by dispatch\_syscall().

```

86                                     {
87     return proc_ipc;
88 }
```

#### 4.100.1.3 ipc\_t\* ipc\_object\_create ( int flags )

Definition at line 76 of file ipc.c.

References object\_header\_t::flags, ipc\_t::header, NULL, and slab\_cache\_alloc().

Referenced by dispatch\_syscall().

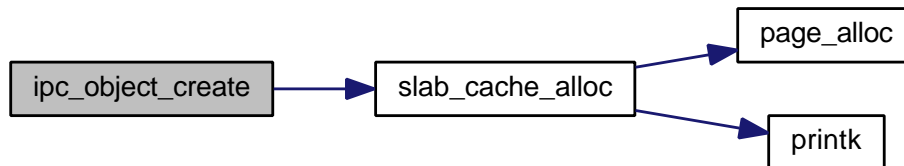


```

76     {
77     ipc_t *ipc = slab_cache_alloc(&ipc_object_cache);
78
79     if(ipc != NULL) {
80         ipc->header.flags = flags;
81     }
82
83     return ipc;
84 }

```

Here is the call graph for this function:



#### 4.100.1.4 void ipc\_receive ( jinue\_syscall\_args\_t\* args )

Definition at line 205 of file ipc.c.

References jinue\_syscall\_args\_t::arg0, jinue\_syscall\_args\_t::arg1, jinue\_syscall\_args\_t::arg2, jinue\_syscall\_args\_t::arg3, message\_info\_t::data\_size, object\_header\_t::flags, thread\_t::header, JINUE\_E2BIG, JINUE\_EBADF, JINUE\_EINVAL, JINUE\_EIO, JINUE\_EPERM, jinue\_node\_entry, memcpy(), thread\_t::message\_args, thread\_t::message\_buffer, thread\_t::message\_info, NULL, OBJECT\_REF\_FLAG\_CLOSED, OBJECT\_TYPE\_IPC, thread\_t::process, process\_get\_descriptor(), ipc\_t::recv\_list, ipc\_t::send\_list, thread\_t::sender, thread\_t::thread\_list, thread\_switch(), thread\_yield\_from(), message\_info\_t::total\_size, and object\_header\_t::type.

Referenced by dispatch\_syscall().

```

205     {
206     thread_t *thread = get_current_thread();
207
208     int fd = (int)args->arg1;
209
210     object_ref_t *ref = process_get_descriptor(thread->process, fd);
211
212     if(! object_ref_is_valid(ref)) {
213         syscall_args_set_error(args, JINUE_EBADF);
214         return;
215     }
216
217     if(object_ref_is_closed(ref)) {
218         syscall_args_set_error(args, JINUE_EIO);
219         return;
220     }
221
222     if(! object_ref_is_owner(ref)) {
223         syscall_args_set_error(args, JINUE_EPERM);
224         return;
225     }
226
227     object_header_t *header = ref->object;
228
229     if(object_is_destroyed(header)) {
230         ref->flags |= OBJECT_REF_FLAG_CLOSED;
231         object_subref(header);
232
233         syscall_args_set_error(args, JINUE_EIO);
234         return;
235     }
236
237     if(header->type != OBJECT_TYPE_IPC) {
238         syscall_args_set_error(args, JINUE_EBADF);

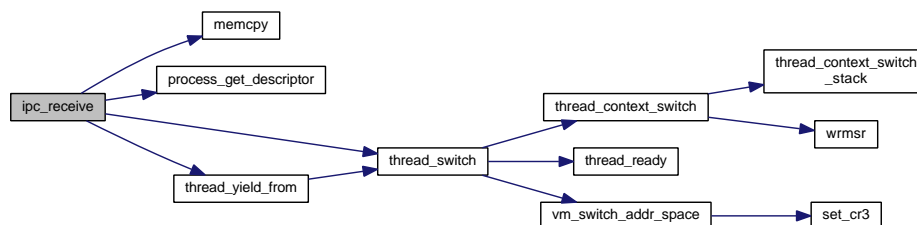
```

```

239     return;
240 }
241
242 ipc_t *ipc = (ipc_t *)header;
243
244 char *user_ptr = (char *)args->arg2;
245 size_t buffer_size = jinue_args_get_buffer_size(args);
246
247 if(! user_buffer_check(user_ptr, buffer_size)) {
248     syscall_args_set_error(args, JINUE_EINVAL);
249     return;
250 }
251
252 thread_t *send_thread = jinue_node_entry(
253     jinue_list_dequeue(&ipc->send_list),
254     thread_t,
255     thread_list);
256
257 if(send_thread == NULL) {
258     /* No thread is waiting to send a message, so we must wait on the receive
259      * list. */
260     jinue_list_enqueue(&ipc->recv_list, &thread->thread_list);
261
262     thread_yield_from(
263         thread,
264         true,      /* make thread block */
265         false);    /* don't destroy */
266
267     /* set by sending thread */
268     send_thread = thread->sender;
269 }
270 else {
271     object_addrref(&send_thread->header);
272     thread->sender = send_thread;
273 }
274
275 if(send_thread->message_info.total_size > buffer_size) {
276     /* message is too big for receive buffer */
277     object_subref(&send_thread->header);
278     thread->sender = NULL;
279
280     syscall_args_set_error(send_thread->message_args, JINUE_E2BIG);
281     syscall_args_set_error(args, JINUE_E2BIG);
282
283     /* switch back to sender thread to return from call immediately */
284     thread_switch(
285         thread,
286         send_thread,
287         false,      /* don't block (put this thread back in ready queue) */
288         false);    /* don't destroy */
289
290     return;
291 }
292
293 memcpy(
294     user_ptr,
295     send_thread->message_buffer,
296     send_thread->message_info.data_size);
297
298 args->arg0 = send_thread->message_args->arg0;
299 args->arg1 = ref->cookie;
300 /* argument 2 is left intact (buffer pointer) */
301 args->arg3 = send_thread->message_args->arg3;
302 }

```

Here is the call graph for this function:



## 4.100.1.5 void ipc\_reply ( jinue\_syscall\_args\_t\* args )

TODO is there a better error number for this situation?

TODO remove this check when descriptor passing is implemented

TODO copy descriptors

TODO set return value and error number

Definition at line 304 of file ipc.c.

References jinue\_syscall\_args\_t::arg2, jinue\_syscall\_args\_t::arg3, message\_info\_t::buffer\_size, message\_info\_t::data\_size, message\_info\_t::desc\_n, thread\_t::header, JINUE\_EINVAL, JINUE\_ENOSYS, JINUE\_SEND\_BUFFER\_SIZE\_OFFSET, JINUE\_SEND\_MAX\_N\_DESC, JINUE\_SEND\_MAX\_SIZE, JINUE\_SEND\_SIZE\_MASK, memcpy(), thread\_t::message\_args, thread\_t::message\_buffer, thread\_t::message\_info, NULL, thread\_t::sender, and thread\_switch().

Referenced by dispatch\_syscall().

```

304
305     thread_t *thread      = get_current_thread();
306     thread_t *send_thread = thread->sender;
307
308     if(send_thread == NULL) {
309         syscall_args_set_error(args, JINUE_EINVAL);
310         return;
311     }
312
313
314     size_t buffer_size = jinue_args_get_buffer_size(args);
315     size_t data_size   = jinue_args_get_data_size(args);
316     size_t desc_n      = jinue_args_get_n_desc(args);
317     size_t total_size  =
318         data_size +
319         desc_n * sizeof(jinue_ipc_descriptor_t);
320
321     if(buffer_size > JINUE_SEND_MAX_SIZE) {
322         syscall_args_set_error(args, JINUE_EINVAL);
323         return;
324     }
325
326     if(total_size > buffer_size) {
327         syscall_args_set_error(args, JINUE_EINVAL);
328         return;
329     }
330
331     if(desc_n > JINUE_SEND_MAX_N_DESC) {
332         syscall_args_set_error(args, JINUE_EINVAL);
333         return;
334     }
335
336     /* the reply must fit in the sender's buffer */
337     if(total_size > send_thread->message_info.buffer_size) {
338         syscall_args_set_error(args, JINUE_EINVAL);
339         return;
340     }
341
342     if(desc_n > 0) {
343         syscall_args_set_error(args, JINUE_ENOSYS);
344         return;
345     }
346
347     const char *user_ptr = (const char *)args->arg2;
348
349     if(! user_buffer_check(user_ptr, buffer_size)) {
350         syscall_args_set_error(args, JINUE_EINVAL);
351         return;
352     }
353
354     memcpy(&send_thread->message_buffer, user_ptr, data_size);
355
356     syscall_args_set_return(send_thread->message_args, 0);
357     send_thread->message_args->arg3 =
358         args->arg3 & ~(JINUE_SEND_SIZE_MASK << JINUE_SEND_BUFFER_SIZE_OFFSET);
359
360     send_thread->message_info.data_size = data_size;
361     send_thread->message_info.desc_n   = desc_n;
362
363

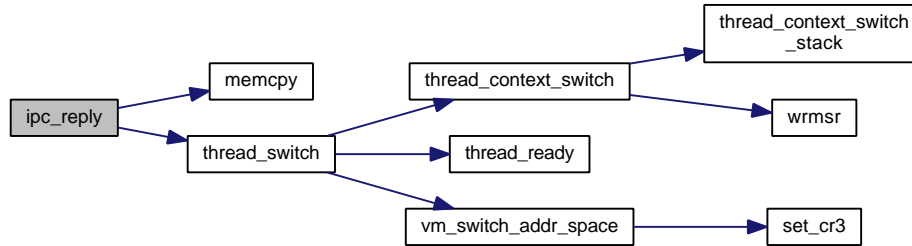
```

```

367     object_subref(&send_thread->header);
368     thread->sender = NULL;
369
370     syscall_args_set_return(args, 0);
371
372     /* switch back to sender thread to return from call immediately */
373     thread_switch(
374         thread,
375         send_thread,
376         false, /* don't block (put this thread back in ready queue) */
377         false); /* don't destroy */
378 }

```

Here is the call graph for this function:



#### 4.100.1.6 void ipc\_send ( jinue\_syscall\_args\_t\* args )

TODO remove this check when descriptor passing is implemented

TODO copy descriptors

TODO copy descriptors

Definition at line 90 of file ipc.c.

References jinue\_syscall\_args\_t::arg0, jinue\_syscall\_args\_t::arg1, jinue\_syscall\_args\_t::arg2, message\_info\_t::buffer\_size, message\_info\_t::cookie, message\_info\_t::data\_size, message\_info\_t::desc\_n, object\_header\_t::flags, message\_info\_t::function, thread\_t::header, JINUE\_EBADF, JINUE\_EINVAL, JINUE\_EIO, JINUE\_ENOSYS, jinue\_node\_entry, JINUE\_SEND\_MAX\_N\_DESC, JINUE\_SEND\_MAX\_SIZE, memcpy(), thread\_t::message\_args, thread\_t::message\_buffer, thread\_t::message\_info, NULL, OBJECT\_REF\_FLAG\_CLOSED, OBJECT\_TYPE\_IPC, thread\_t::process, process\_get\_descriptor(), ipc\_t::rcv\_list, ipc\_t::send\_list, thread\_t::sender, thread\_t::thread\_list, thread\_switch(), thread\_yield\_from(), message\_info\_t::total\_size, and object\_header\_t::type.

Referenced by dispatch\_syscall().

```

90     {
91         thread_t *thread = get_current_thread();
92
93         message_info_t *message_info = &thread->message_info;
94
95         message_info->function      = args->arg0;
96         message_info->buffer_size   = jinue_args_get_buffer_size(args);
97         message_info->data_size     = jinue_args_get_data_size(args);
98         message_info->desc_n        = jinue_args_get_n_desc(args);
99         message_info->total_size    =
100             message_info->data_size +
101             message_info->desc_n * sizeof(jinue_ipc_descriptor_t);
102
103         if(message_info->buffer_size > JINUE_SEND_MAX_SIZE) {
104             syscall_args_set_error(args, JINUE_EINVAL);
105             return;
106         }
107
108         if(message_info->total_size > message_info->buffer_size) {
109             syscall_args_set_error(args, JINUE_EINVAL);

```

```

110     return;
111 }
112
113 if(message_info->desc_n > JINUE_SEND_MAX_N_DESC) {
114     syscall_args_set_error(args, JINUE_EINVAL);
115     return;
116 }
117
118 if(message_info->desc_n > 0) {
119     syscall_args_set_error(args, JINUE_ENOSYS);
120     return;
121 }
122
123 int fd = (int)args->arg1;
124
125 object_ref_t *ref = process_get_descriptor(thread->process, fd);
126
127 if(! object_ref_is_valid(ref)) {
128     syscall_args_set_error(args, JINUE_EBADF);
129     return;
130 }
131
132 if(object_ref_is_closed(ref)) {
133     syscall_args_set_error(args, JINUE_EIO);
134     return;
135 }
136
137 message_info->cookie = ref->cookie;
138
139 object_header_t *header = ref->object;
140
141 if(object_is_destroyed(header)) {
142     ref->flags |= OBJECT_REF_FLAG_CLOSED;
143     object_subref(header);
144
145     syscall_args_set_error(args, JINUE_EIO);
146     return;
147 }
148
149 if(header->type != OBJECT_TYPE_IPC) {
150     syscall_args_set_error(args, JINUE_EBADF);
151     return;
152 }
153
154 ipc_t *ipc = (ipc_t *)header;
155
156 char *user_ptr = (char *)args->arg2;
157
158 if(! user_buffer_check(user_ptr, message_info->buffer_size)) {
159     syscall_args_set_error(args, JINUE_EINVAL);
160     return;
161 }
162
163 memcpy(&thread->message_buffer, user_ptr, message_info->data_size);
164
165 /* return values are set by ipc_reply() (or by ipc_receive() if the call
166  * fails because the message is too big for the receiver's buffer) */
167 thread->message_args = args;
168
169 thread_t *recv_thread = jinue_node_entry(
170     jinue_list_dequeue(&ipc->recv_list),
171     thread_t,
172     thread_list);
173
174 if(recv_thread == NULL) {
175     /* No thread is waiting to receive this message, so we must wait on the
176      * sender list. */
177     jinue_list_enqueue(&ipc->send_list, &thread->thread_list);
178
179     thread_yield_from(
180         thread,
181         true,          /* make thread block */
182         false);       /* don't destroy */
183 }
184 else {
185     object_addr(&thread->header);
186     recv_thread->sender = thread;
187
188     /* switch to receiver thread, which will resume inside syscall_receive() */
189     thread_switch(
190         thread,

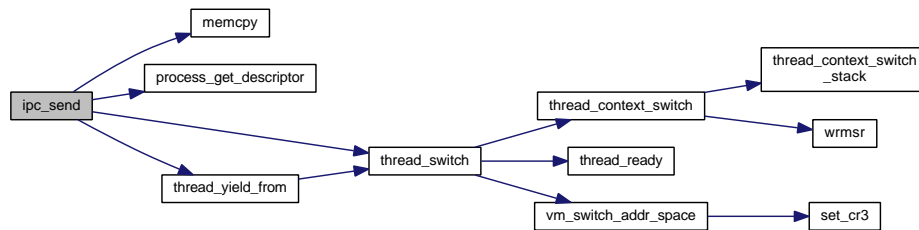
```

```

194         rcv_thread,
195         true,      /* block sender thread */
196         false);    /* don't destroy sender */
197     }
198
199     /* copy reply to user space buffer */
200     memcpy(user_ptr, &thread->message_buffer, message_info->data_size);
201
202 }
203

```

Here is the call graph for this function:



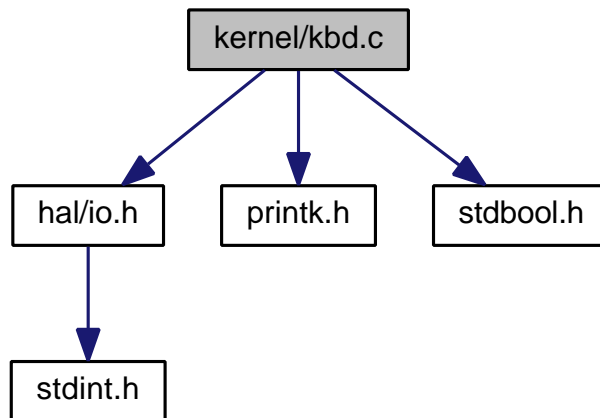
## 4.101 kernel/kbd.c File Reference

```

#include <hal/io.h>
#include <printk.h>
#include <stdbool.h>

```

Include dependency graph for kbd.c:



### Functions

- void **any\_key** (void)

#### 4.101.1 Function Documentation

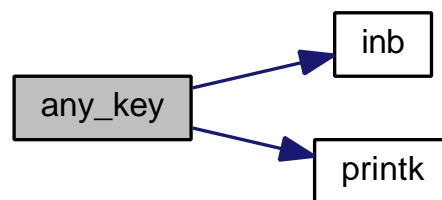
##### 4.101.1.1 void any\_key ( void )

Definition at line 36 of file kbd.c.

References inb(), and printk().

```
36     {
37     unsigned char buffer;
38     bool ignore;
39
40     /* prompt */
41     printk("(press enter)");
42
43     /* wait for key, ignore break codes */
44     ignore = false;
45     while(1) {
46         do {
47             buffer = inb(0x64);
48         } while( (buffer & 1) == 0 );
49
50         buffer = inb(0x60);
51
52         if(buffer == 0x0e || buffer == 0x0f) {
53             ignore = true;
54             continue;
55         }
56
57         if(ignore) {
58             ignore = false;
59             continue;
60         }
61
62         if(buffer == 0x1c || buffer == 0x5a) {
63             break;
64         }
65     }
66
67     /* advance cursor */
68     printk("\n");
69 }
```

Here is the call graph for this function:



## 4.102 kernel/kmain.c File Reference

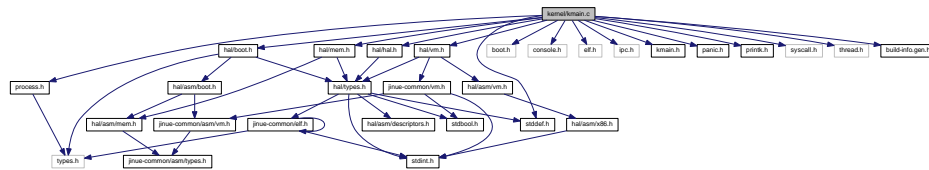
```
#include <hal/boot.h>
```

```

#include <hal/hal.h>
#include <hal/mem.h>
#include <hal/vm.h>
#include <boot.h>
#include <console.h>
#include <elf.h>
#include <ipc.h>
#include <kmain.h>
#include <panic.h>
#include <printk.h>
#include <process.h>
#include <stddef.h>
#include <syscall.h>
#include <thread.h>
#include "build-info.gen.h"

```

Include dependency graph for kmain.c:



## Functions

- void **kmain** (void)

### 4.102.1 Function Documentation

#### 4.102.1.1 void kmain ( void )

Definition at line 67 of file kmain.c.

References `process_t::addr_space`, `boot_alloc_init()`, `boot_info_t::boot_heap`, `boot_info_check()`, `BUILD_HOST`, `BUILD_TIME`, `boot_info_t::cmdline`, `console_init()`, `elf_load()`, `elf_info_t::entry`, `get_boot_info()`, `GIT_REVISION`, `hal_init()`, `ipc_boot_init()`, `boot_info_t::kernel_size`, `mem_check_memory()`, `NULL`, `panic()`, `printk()`, `process_boot_init()`, `process_create_initial()`, `boot_info_t::ramdisk_size`, `boot_info_t::ramdisk_start`, `elf_info_t::stack_addr`, `thread_create_boot()`, `thread_yield_from()`, and `VGA_COLOR_YELLOW`.

```

67     {
68         elf_info_t elf_info;
69
70         /* initialize console and say hello */
71         console_init();
72
73         /* Say hello. */
74         printk("Kernel revision " GIT_REVISION " built " BUILD_TIME " on "
75             BUILD_HOST "\n");
76
77         const boot_info_t *boot_info = get_boot_info();
78         (void)boot_info_check(true);
79
80         printk("Kernel size is %u bytes.\n", boot_info->kernel_size);
81
82         if(boot_info->ramdisk_start == 0 || boot_info->ramdisk_size == 0) {
83             printk("%kWarning: no initial RAM disk loaded.\n", VGA_COLOR_YELLOW);
84         }
85     }

```

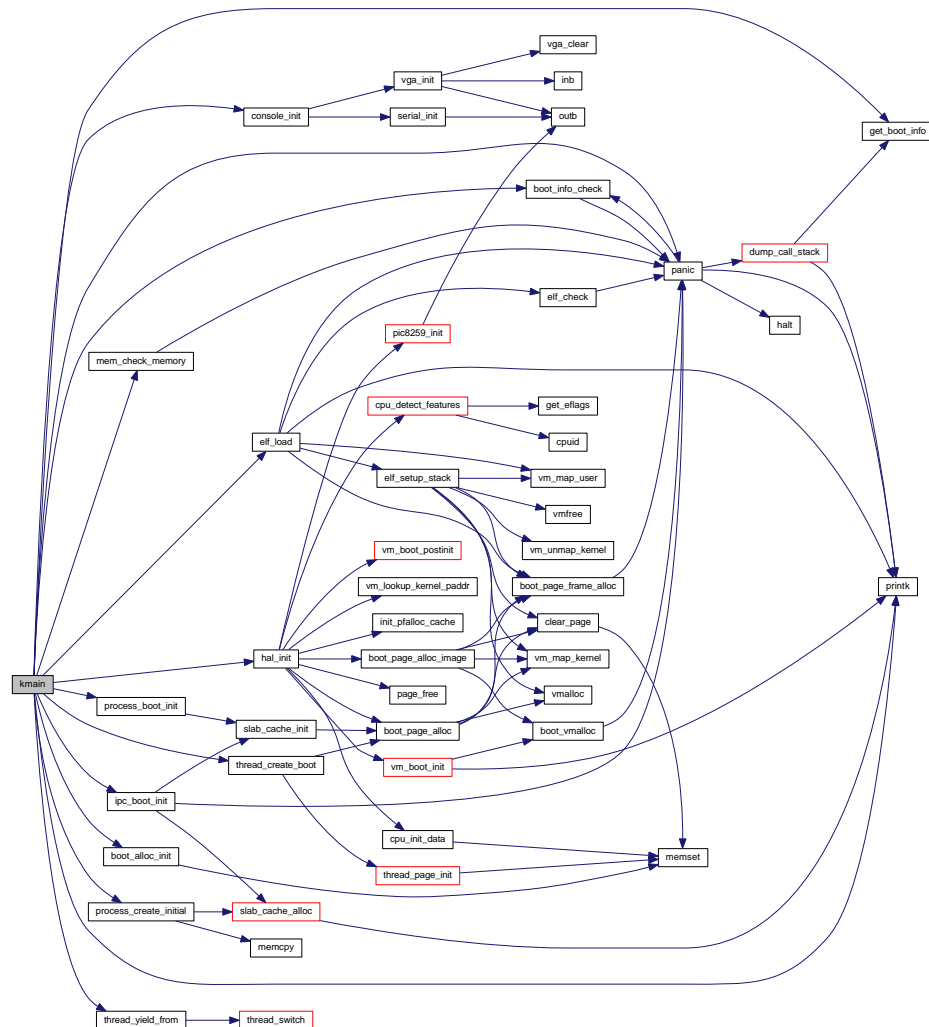


```

85     printk("RAM disk with size %u bytes loaded at address %x.\n", boot_info->
ramdisk_size, boot_info->ramdisk_start);
86 }
87
88     printk("Kernel command line:\n", boot_info->kernel_size);
89     printk("    %s\n", boot_info->cmdline);
90
91     /* Initialize the boot allocator. */
92     boot_alloc_t boot_alloc;
93     boot_alloc_init(&boot_alloc, boot_info->boot_heap);
94     mem_check_memory(&boot_alloc, boot_info);
95
96     /* initialize hardware abstraction layer */
97     hal_init(&boot_alloc, boot_info);
98
99     /* initialize caches */
100     ipc_boot_init(&boot_alloc);
101     process_boot_init(&boot_alloc);
102
103     /* create process for process manager */
104     process_t *process = process_create_initial();
105
106     if(process == NULL) {
107         panic("Could not create initial process.");
108     }
109
110     /* load process manager binary */
111     Elf32_Ehdr *elf = find_process_manager();
112     elf_load(&elf_info, elf, &process->addr_space, &boot_alloc);
113
114     /* create initial thread */
115     thread_t *thread = thread_create_boot(
116         process,
117         elf_info.entry,
118         elf_info.stack_addr,
119         &boot_alloc);
120
121     if(thread == NULL) {
122         panic("Could not create initial thread.");
123     }
124
125     /* start process manager
126     *
127     * We switch from NULL since this is the first thread. */
128     thread_yield_from(
129         NULL,
130         false,      /* don't block */
131         false);     /* don't destroy */
132                 /* just be nice */
133
134     /* should never happen */
135     panic("thread_yield_from() returned in kmain()");
136 }

```

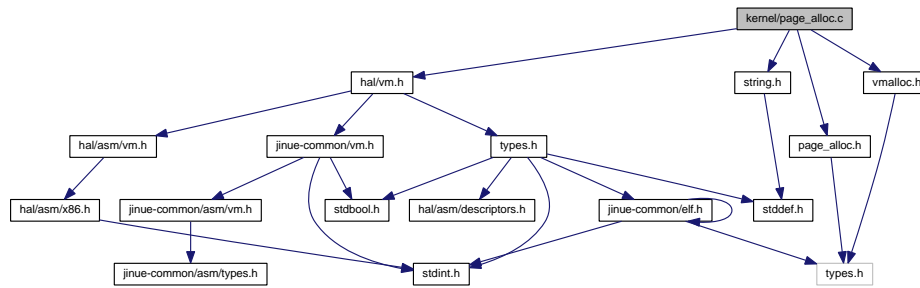
Here is the call graph for this function:



#### 4.103 kernel/page\_alloc.c File Reference

```
#include <hal/vm.h>
#include <page_alloc.h>
#include <string.h>
#include <vmalloc.h>
```

Include dependency graph for page\_alloc.c:



## Data Structures

- struct **alloc\_page**

## Functions

- void \* **page\_alloc** (void)  
*Allocate a page of kernel memory.*
- void **page\_free** (void \*page)  
*Free a page of kernel memory.*
- bool **page\_alloc\_is\_empty** (void)  
*Check that pages are available to be allocated.*
- bool **add\_page\_frame** (kern\_paddr\_t paddr)  
*Map a page frame and add it to the page allocator.*
- kern\_paddr\_t **remove\_page\_frame** (void)  
*Remove a page frame from the allocator.*
- void **clear\_page** (void \*page)  
*Clear a page by writing all bytes to zero.*

### 4.103.1 Function Documentation

#### 4.103.1.1 bool add\_page\_frame ( kern\_paddr\_t paddr )

Map a page frame and add it to the page allocator.

This function is used to implement a system call that allows userspace to provide additional page frames to the kernel. This function fails when no more pages of kernel address space can be allocated with **vmalloc()** (p. 275) to map the provided page frame.

#### Parameters

<i>paddr</i>	physical address of the provided page frame
--------------	---

**Returns**

true if the function succeeded

Definition at line 111 of file page\_alloc.c.

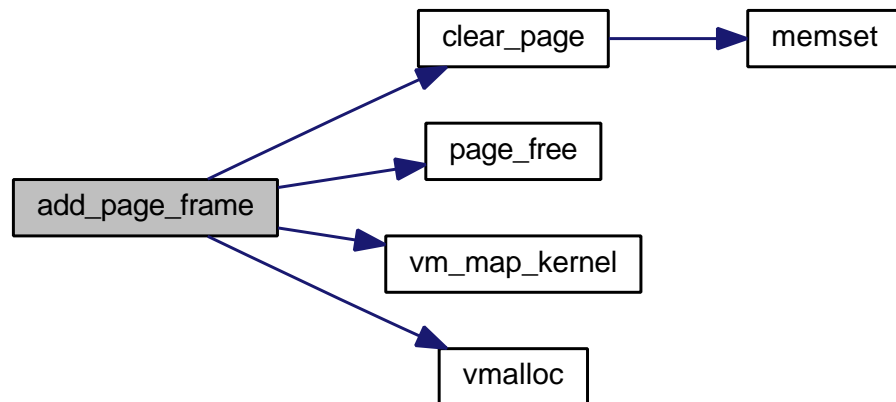
References `clear_page()`, `NULL`, `page_free()`, `VM_FLAG_READ_WRITE`, `vm_map_kernel()`, and `vmalloc()`.

```

111     {
112         void *page = vmalloc();
113
114         if (page == NULL) {
115             return false;
116         }
117
118         vm_map_kernel(page, paddr, VM_FLAG_READ_WRITE);
119
120         /* Since this page is coming from userspace, it is important to clear it:
121          * 1) The page may contain sensitive information, which we don't want to
122          *    leak through Meltdown-like vulnerabilities; and
123          * 2) Since the content is userspace-chosen, it could be used for kernel
124          *    vulnerability exploits. */
125         clear_page(page);
126         page_free(page);
127
128         return true;
129     }

```

Here is the call graph for this function:

**4.103.1.2 void clear\_page ( void \* page )**

Clear a page by writing all bytes to zero.

**Parameters**

<i>page</i>	the page to clear
-------------	-------------------

Definition at line 173 of file page\_alloc.c.

References `memset()`, and `PAGE_SIZE`.

Referenced by `add_page_frame()`, `boot_page_alloc()`, `boot_page_alloc_early()`, `boot_page_alloc_image()`, `elf_setup_stack()`, and `remove_page_frame()`.

```

173     {
174         memset(page, 0, PAGE_SIZE);
175     }

```

Here is the call graph for this function:



#### 4.103.1.3 void\* page\_alloc ( void )

Allocate a page of kernel memory.

Pages allocated by this function can be used for any purpose in the kernel, e.g. as slabs for the slab allocator or as page tables.

Pages allocated by this function are not guaranteed to be mapped in the allocations region of the kernel address space (that is, the region managed by **vmalloc()** (p. 275)). While most will be, pages originally allocated in the image region during initialization by calling **boot\_page\_alloc\_image()** (p. 72) can be reclaimed with **page\_free()** (p. 252) and then re-allocated by this function.

##### Returns

allocated page

Definition at line 59 of file page\_alloc.c.

References `alloc_page::next`, and `NULL`.

Referenced by `remove_page_frame()`, `slab_cache_alloc()`, `thread_create()`, and `vm_clone_page_directory()`.

```
59      {
60      struct alloc_page *alloc_page = head_page;
61
62      if(alloc_page != NULL) {
63          head_page = alloc_page->next;
64      }
65
66      return alloc_page;
67 }
```

#### 4.103.1.4 bool page\_alloc\_is\_empty ( void )

Check that pages are available to be allocated.

Page availability can be checked with this function before calling either **page\_alloc()** (p. 251) or **remove\_page\_frame()** (p. 253).

##### Returns

true if pages are available (one or more)

Definition at line 95 of file page\_alloc.c.

References `NULL`.

```
95      {
96      return head_page == NULL;
97 }
```

#### 4.103.1.5 void page\_free ( void \* page )

Free a page of kernel memory.

Pages freed by calling this function are available to be re-allocated by the **page\_alloc()** (p. 251) function. This function can be used to free pages allocated by **page\_alloc()** (p. 251) or to reclaim pages allocated during kernel initialization by **boot\_page\_alloc()** (p. 70) or **boot\_page\_alloc\_image()** (p. 72).

##### Parameters

<i>page</i>	the page to free
-------------	------------------

Definition at line 80 of file page\_alloc.c.

References alloc\_page::next.

Referenced by add\_page\_frame(), hal\_init(), and thread\_destroy().

```

80      {
81      struct alloc_page *alloc_page = page;
82      alloc_page->next      = head_page;
83      head_page            = alloc_page;
84  }
```

#### 4.103.1.6 kern\_paddr\_t remove\_page\_frame ( void )

Remove a page frame from the allocator.

This function is used implement a system call that allows userspace to reclaim free kernel memory for its own use. The address space page is freed with **vmfree()** (p. 276) and the physical address of the underlying page frame is returned.

##### Returns

physical address of the freed page frame, or PFNULL if none is available

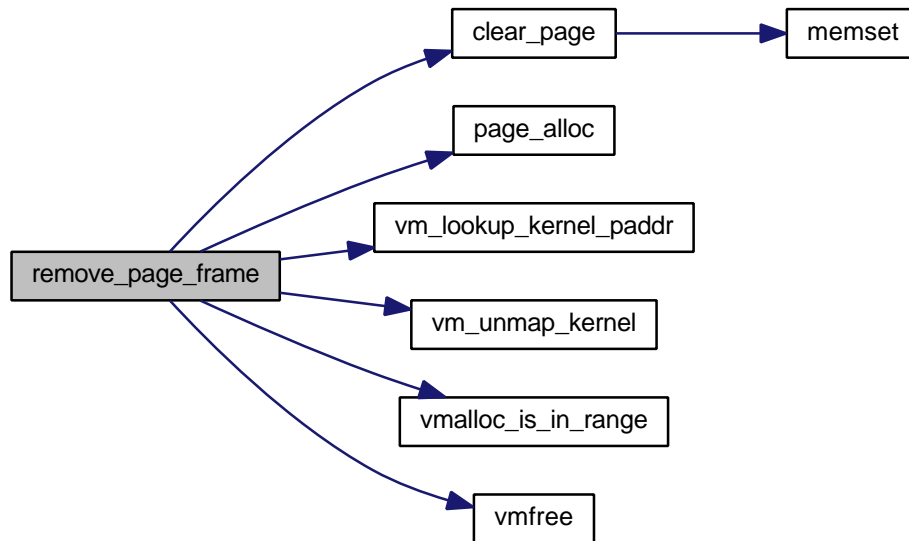
Definition at line 142 of file page\_alloc.c.

References clear\_page(), NULL, page\_alloc(), PFNULL, vm\_lookup\_kernel\_paddr(), vm\_unmap\_kernel(), vmalloc\_is\_in\_range(), and vmfree().

```

142      {
143      void *page = page_alloc();
144
145      if(page == NULL) {
146          return PFNULL;
147      }
148
149      /* This page is going to userspace. Let's clear its content so we don't
150       * leak information about the kernel's internal state that could be useful
151       * for exploiting vulnerabilities. */
152      clear_page(page);
153
154      kern_paddr_t paddr = vm_lookup_kernel_paddr(page);
155
156      vm_unmap_kernel(page);
157
158      /* The page may be in the image region instead of the allocations region if
159       * it was allocated during kernel initialization. */
160      if(vmalloc_is_in_range(page)) {
161          vmfree(page);
162      }
163
164      return paddr;
165  }
```

Here is the call graph for this function:



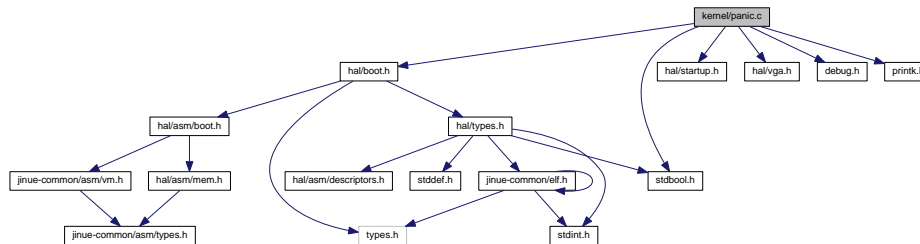
## 4.104 kernel/panic.c File Reference

```

#include <hal/boot.h>
#include <hal/startup.h>
#include <hal/vga.h>
#include <debug.h>
#include <printk.h>
#include <stdbool.h>

```

Include dependency graph for panic.c:



## Functions

- void **panic** (const char \*message)

### 4.104.1 Function Documentation

#### 4.104.1.1 void panic ( const char \* *message* )

Definition at line 40 of file `panic.c`.

References `boot_info_check()`, `dump_call_stack()`, `halt()`, `printk()`, and `VGA_COLOR_RED`.

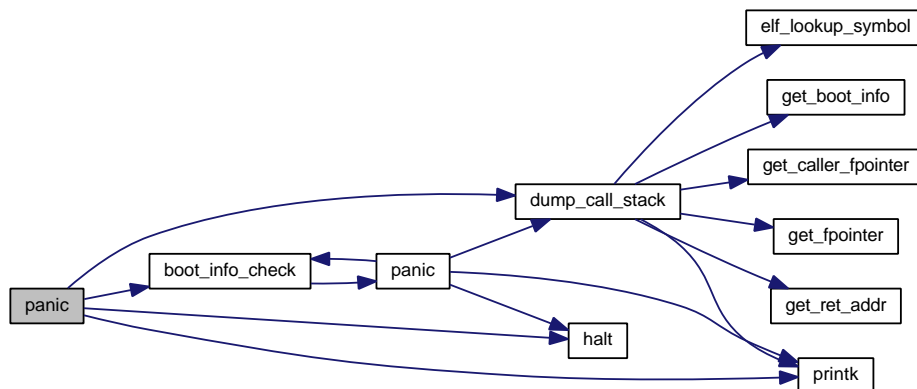
Referenced by `__assert_failed()`, `boot_heap_pop()`, `boot_info_check()`, `boot_page_alloc_early()`, `boot_page_frame_alloc()`, `boot_vmalloc()`, `dispatch_interrupt()`, `elf_check()`, `elf_load()`, `ipc_boot_init()`, `kmain()`, `mem_check_memory()`, and `pfalloc_from()`.

```

40
41     static int enter_count = 0;
42
43     ++enter_count;
44
45     /* When things go seriously wrong, things that panic does itself can create
46      * a further panic, for example by triggering a hardware exception. The
47      * enter_count static variable keeps count of the number of times panic()
48      * is entered. */
49     switch(enter_count) {
50     case 1:
51     case 2:
52         /* The first two times panic() is entered, a panic message is displayed
53          * along with a full call sack dump. */
54         printk( "%kKERNEL PANIC%s: %s\n",
55                VGA_COLOR_RED,
56                enter_count==1?"": " (recursive)",
57                message);
58
59         if( boot_info_check(false) ) {
60             dump_call_stack();
61         }
62         else {
63             printk("Cannot dump call stack because boot information structure is invalid.\n");
64         }
65         break;
66     case 3:
67         /* The third time, a "recursive count exceeded" message is displayed. We
68          * try to limit the number of actions we take to limit the chances of a
69          * further panic. */
70         printk("%kKERNEL PANIC (recursive count exceeded)\n", VGA_COLOR_RED);
71         break;
72     default:
73         /* The fourth time, we do nothing but halt the CPU. */
74         break;
75     }
76
77     halt();
78 }

```

Here is the call graph for this function:

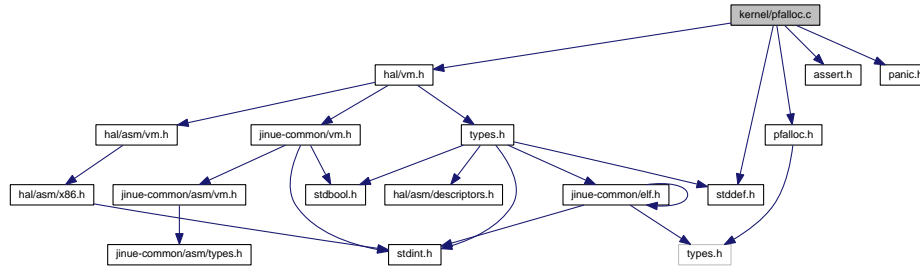


## 4.105 kernel/pfalloc.c File Reference

```
#include <hal/vm.h>
```



```
#include <assert.h>
#include <panic.h>
#include <pfalloc.h>
#include <stddef.h>
Include dependency graph for pfalloc.c:
```



## Functions

- void **init\_pfalloc\_cache** (pfalloc\_cache\_t \*pfcache, kern\_paddr\_t \*stack\_page)
- kern\_paddr\_t **pfalloc\_from** (pfalloc\_cache\_t \*pfcache)
- void **pffree\_to** (pfalloc\_cache\_t \*pfcache, kern\_paddr\_t paddr)

## Variables

- pfalloc\_cache\_t **global\_pfalloc\_cache**

### 4.105.1 Function Documentation

#### 4.105.1.1 void init\_pfalloc\_cache ( pfalloc\_cache\_t \* pfcache, kern\_paddr\_t \* stack\_page )

Definition at line 40 of file pfalloc.c.

References pfalloc\_cache\_t::count, KERNEL\_PAGE\_STACK\_SIZE, PFNULL, and pfalloc\_cache\_t::ptr.

Referenced by hal\_init().

```
40                                     {
41     kern_paddr_t    *ptr;
42     unsigned int    idx;
43
44     ptr = stack_page;
45
46     for(idx = 0; idx < KERNEL_PAGE_STACK_SIZE; ++idx) {
47         ptr[idx] = PFNULL;
48     }
49
50     pfcache->ptr    = stack_page;
51     pfcache->count = 0;
52 }
```

#### 4.105.1.2 kern\_paddr\_t pfalloc\_from ( pfalloc\_cache\_t \* pfcache )

Definition at line 54 of file pfalloc.c.

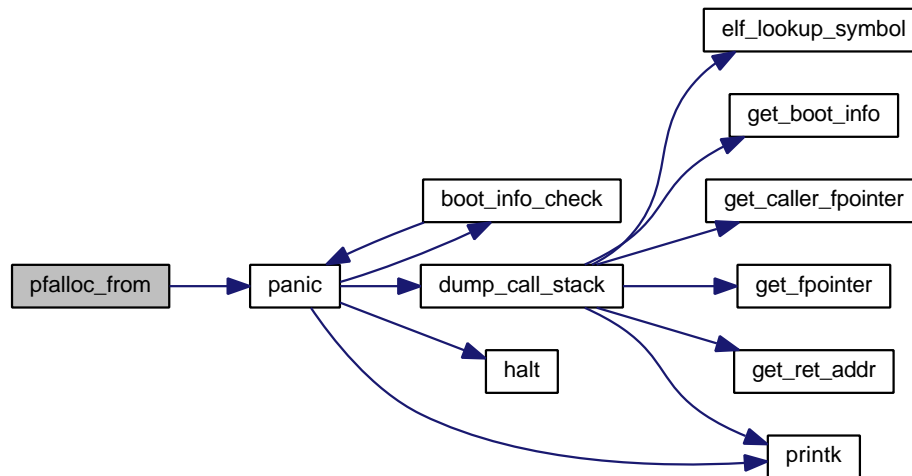
References pfalloc\_cache\_t::count, panic(), and pfalloc\_cache\_t::ptr.

```

54                                     {
55     if(pfcache->count == 0) {
56         panic("pfalloc_from(): no more pages to allocate");
57     }
58
59     --pfcache->count;
60
61     return * (--pfcache->ptr);
62 }

```

Here is the call graph for this function:



#### 4.105.1.3 void pffree\_to ( pfalloc\_cache\_t\* pfcache, kern\_paddr\_t paddr )

We are leaking memory here. Should we panic instead?

Definition at line 64 of file pfalloc.c.

References pfalloc\_cache\_t::count, KERNEL\_PAGE\_STACK\_SIZE, and pfalloc\_cache\_t::ptr.

```

64                                     {
65     if(pfcache->count >= KERNEL_PAGE_STACK_SIZE) {
66         return;
67     }
68
69     ++pfcache->count;
70
71     (pfcache->ptr++)[0] = paddr;
72 }
73

```

### 4.105.2 Variable Documentation

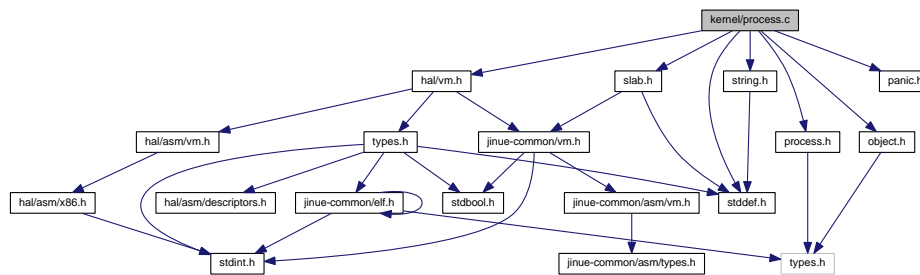
#### 4.105.2.1 pfalloc\_cache\_t global\_pfalloc\_cache

Definition at line 38 of file pfalloc.c.

Referenced by hal\_init().

## 4.106 kernel/process.c File Reference

```
#include <hal/vm.h>
#include <panic.h>
#include <process.h>
#include <object.h>
#include <slab.h>
#include <stddef.h>
#include <string.h>
Include dependency graph for process.c:
```



### Functions

- void **process\_boot\_init** (boot\_alloc\_t \*boot\_alloc)
- process\_t \* **process\_create** (void)
- process\_t \* **process\_create\_initial** (void)
- object\_ref\_t \* **process\_get\_descriptor** (process\_t \*process, int fd)
- int **process\_unused\_descriptor** (process\_t \*process)

### 4.106.1 Function Documentation

#### 4.106.1.1 void process\_boot\_init ( boot\_alloc\_t \* boot\_alloc )

Definition at line 49 of file process.c.

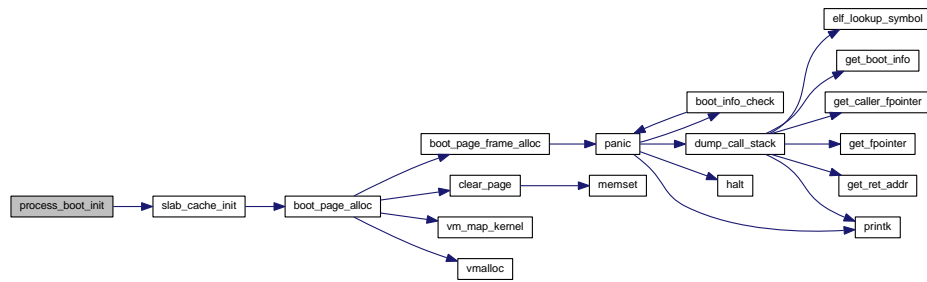
References NULL, slab\_cache\_init(), and SLAB\_DEFAULTS.

Referenced by kmain().

```

49      {
50          slab_cache_init(
51              &process_cache,
52              "process_cache",
53              sizeof(process_t),
54              0,
55              process_ctor,
56              NULL,
57              SLAB_DEFAULTS,
58              boot_alloc);
59      }
```

Here is the call graph for this function:



#### 4.106.1.2 process\_t\*process\_create ( void )

Definition at line 65 of file process.c.

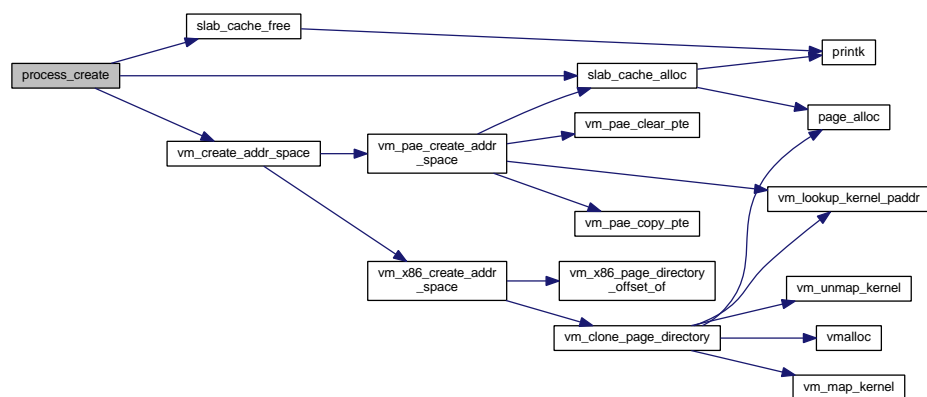
References process\_t::addr\_space, NULL, slab\_cache\_alloc(), slab\_cache\_free(), and vm\_create\_addr\_space().

```

65         {
66     process_t *process = slab_cache_alloc(&process_cache);
67
68     if(process != NULL) {
69         addr_space_t *addr_space = vm_create_addr_space(&process->addr_space);
70
71         /* The address space object is located inside the process object but the
72          * call to vm_create_addr_space() above can still fail if we cannot
73          * allocate the initial page directory/tables or, when PAE is enabled,
74          * if we cannot allocate a PDPT. */
75         if(addr_space == NULL) {
76             slab_cache_free(process);
77             return NULL;
78         }
79
80         process_init(process);
81     }
82
83     return process;
84 }

```

Here is the call graph for this function:



4.106.1.3 `process_t* process_create_initial ( void )`

Definition at line 86 of file process.c.

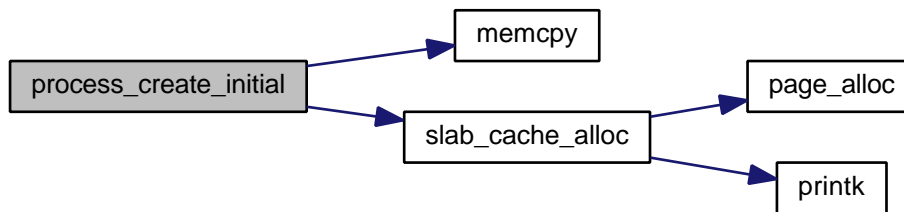
References `process_t::addr_space`, `initial_addr_space`, `memcpy()`, `NULL`, and `slab_cache_alloc()`.

Referenced by `kmain()`.

```

86      {
87      process_t *process = slab_cache_alloc(&process_cache);
88
89      if(process != NULL) {
90          memcpy(&process->addr_space, &initial_addr_space, sizeof(addr_space_t));
91          process_init(process);
92      }
93
94      return process;
95  }
```

Here is the call graph for this function:

4.106.1.4 `object_ref_t* process_get_descriptor ( process_t* process, int fd )`

Definition at line 97 of file process.c.

References `process_t::descriptors`, `NULL`, and `PROCESS_MAX_DESCRIPTOR`.

Referenced by `dispatch_syscall()`, `ipc_receive()`, `ipc_send()`, and `process_unused_descriptor()`.

```

97      {
98      if(fd < 0 || fd > PROCESS_MAX_DESCRIPTOR) {
99          return NULL;
100      }
101
102      return &process->descriptors[fd];
103  }
```

4.106.1.5 `int process_unused_descriptor ( process_t* process )`

Definition at line 105 of file process.c.

References `process_get_descriptor()`, and `PROCESS_MAX_DESCRIPTOR`.

Referenced by `dispatch_syscall()`.

```

105      {
106      int idx;
107
108      for(idx = 0; idx < PROCESS_MAX_DESCRIPTOR; ++idx) {
109          object_ref_t *ref = process_get_descriptor(process, idx);
110
111          if(! object_ref_is_valid(ref)) {
```

```

112         return idx;
113     }
114 }
115
116     return -1;
117 }

```

Here is the call graph for this function:



## 4.107 kernel/slab.c File Reference

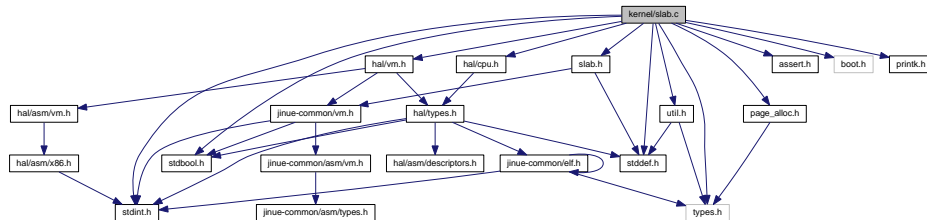
Kernel object allocator.

```

#include <hal/cpu.h>
#include <hal/vm.h>
#include <assert.h>
#include <boot.h>
#include <page_alloc.h>
#include <printk.h>
#include <slab.h>
#include <stdbool.h>
#include <stddef.h>
#include <stdint.h>
#include <types.h>
#include <util.h>

```

Include dependency graph for slab.c:



## Functions

- void **slab\_cache\_init**(**slab\_cache\_t** \*cache, char \*name, **size\_t** size, **size\_t** alignment, **slab\_ctor\_t** ctor, **slab\_ctor\_t** dtor, int flags, **boot\_alloc\_t** \*boot\_alloc)
 

Initialize an object cache.
- void \* **slab\_cache\_alloc**(**slab\_cache\_t** \*cache)
 

Allocate an object from the specified cache.
- void **slab\_cache\_free**(void \*buffer)
 

Free an object.
- void **slab\_cache\_reap**(**slab\_cache\_t** \*cache)
 

Return memory to the page allocator.
- void **slab\_cache\_set\_working\_set**(**slab\_cache\_t** \*cache, unsigned int n)
 

Set a cache's working set.

### 4.107.1 Detailed Description

Kernel object allocator. This file implements a slab allocator as described in Jeff Bonwick's paper "The Slab Allocator: An Object-Caching Kernel Memory Allocator":

[https://www.usenix.org/publications/library/proceedings/bos94/full\\_papers/bonwick.-ps](https://www.usenix.org/publications/library/proceedings/bos94/full_papers/bonwick.-ps)

This is the main object allocator for the kernel. (Some early allocations performed during kernel initialization use the boot heap instead - see boot.c.)

Definition in file **slab.c**.

### 4.107.2 Function Documentation

#### 4.107.2.1 void\* slab\_cache\_alloc ( slab\_cache\_t\* cache )

Allocate an object from the specified cache.

The cache must have been initialized with **slab\_cache\_init()** (p. 270). If no more space is available on existing slabs, this function tries to allocate a new slab using the kernel's page allocator (i.e. **page\_alloc()** (p. 251)). If page allocation fails, this function fails by returning NULL.

##### Parameters

<i>cache</i>	the cache from which to allocate an object
--------------	--

##### Returns

the address of the allocated object, or NULL if allocation failed

ASSERTION: now that slab\_cache\_grow() has run, we should have found at least one empty slab

Important note regarding the slab lists: The empty, partial and full slab lists are doubly-linked lists. This is done to allow the deletion of an arbitrary link given a pointer to it. We do not allow reverse traversal: we do not maintain a tail pointer and, more importantly: we do *NOT* maintain the previous pointer of the first link in the list (i.e. it is garbage data, not NULL).

ASSERTION: there is at least one buffer on the free list

ASSERT: the slab is the head of the partial list

Definition at line 232 of file slab.c.

References `assert`, `slab_cache_t::bufctl_offset`, `slab_cache_t::ctor`, `slab_cache_t::empty_count`, `slab_cache_t::flags`, `slab_t::free_list`, `slab_cache_t::name`, `slab_bufctl_t::next`, `slab_t::next`, `NULL`, `slab_t::obj_count`, `slab_cache_t::obj_size`, `page_alloc()`, `slab_t::prev`, `printf()`, `SLAB_POISON`, `SLAB_POISON_ALIVE_VALUE`, `SLAB_POISON_DEAD_VALUE`, `SLAB_RED_ZONE`, `SLAB_RED_ZONE_VALUE`, `slab_cache_t::slabs_empty`, `slab_cache_t::slabs_full`, and `slab_cache_t::slabs_partial`.

Referenced by `ipc_boot_init()`, `ipc_object_create()`, `process_create()`, `process_create_initial()`, and `vm_pae_create_addr_space()`.

```

232                                     {
233     slab_t          *slab;
234
235     if(cache->slabs_partial != NULL) {
236         slab = cache->slabs_partial;
237     }
238     else {
239         if(cache->slabs_empty == NULL) {
240             void *slab_addr = page_alloc();

```

```

241
242     if(slab_addr == NULL) {
243         return NULL;
244     }
245
246     init_and_add_slab(cache, slab_addr);
247 }
248
249 slab = cache->slabs_empty;
250
251 assert(slab != NULL);
252
253
254 /* We are about to allocate one object from this slab, so it will
255  * not be empty anymore...*/
256 cache->slabs_empty = slab->next;
257
258 --(cache->empty_count);
259
260 slab->next = cache->slabs_partial;
261 if(slab->next != NULL) {
262     slab->next->prev = slab;
263 }
264 cache->slabs_partial = slab;
265 }
266
267 slab_bufctl_t *bufctl = slab->free_list;
268
269 assert(bufctl != NULL);
270
271 slab->free_list = bufctl->next;
272 slab->obj_count += 1;
273
274 /* If we just allocated the last buffer, move the slab to the full
275  * list */
276 if(slab->free_list == NULL) {
277     /* remove from the partial slabs list */
278
279     assert(cache->slabs_partial == slab);
280
281     cache->slabs_partial = slab->next;
282
283     if(slab->next != NULL) {
284         slab->next->prev = slab->prev;
285     }
286
287     /* add to the full slabs list */
288     slab->next = cache->slabs_full;
289     cache->slabs_full = slab;
290
291     if(slab->next != NULL) {
292         slab->next->prev = slab;
293     }
294 }
295
296 uint32_t *buffer = (uint32_t *) ( (char *)bufctl - cache->bufctl_offset );
297
298 if(cache->flags & SLAB_POISON) {
299     unsigned int idx;
300     unsigned int dump_lines = 0;
301
302     for(idx = 0; idx < cache->obj_size / sizeof(uint32_t); ++idx) {
303         if(buffer[idx] != SLAB_POISON_DEAD_VALUE) {
304             if(dump_lines == 0) {
305                 printk("detected write to freed object, cache: %s buffer: 0x%x:\n",
306                     cache->name,
307                     (unsigned int)buffer
308                 );
309             }
310
311             if(dump_lines < 4) {
312                 printk(" value 0x%x at byte offset %u\n", buffer[idx], idx * sizeof(
313 uint32_t));
314             }
315
316             ++dump_lines;
317         }
318     }
319
320     buffer[idx] = SLAB_POISON_ALIVE_VALUE;
321 }
322
323 /* If both SLAB_POISON and SLAB_RED_ZONE are enabled, we perform

```

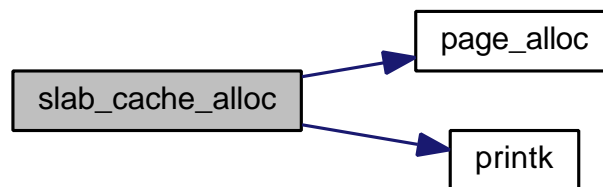


```

333     * redzone checking even on freed objects. */
334     if(cache->flags & SLAB_RED_ZONE) {
335         if(buffer[idx] != SLAB_RED_ZONE_VALUE) {
336             printk("detected write past the end of freed object, cache: %s buffer: 0x%x value: 0x%x\n",
337                 cache->name,
338                 (unsigned int)buffer,
339                 buffer[idx]
340             );
341         }
342         buffer[idx] = SLAB_RED_ZONE_VALUE;
343     }
344 }
345
346 if(cache->ctor != NULL) {
347     cache->ctor((void *)buffer, cache->obj_size);
348 }
349 }
350 else if(cache->flags & SLAB_RED_ZONE) {
351     buffer[cache->obj_size / sizeof(uint32_t)] = SLAB_RED_ZONE_VALUE;
352 }
353 }
354 return (void *)buffer;
355 }

```

Here is the call graph for this function:



#### 4.107.2.2 void slab\_cache\_free ( void \* *buffer* )

Free an object.

##### Parameters

<i>buffer</i>	the object to free
---------------	--------------------

Definition at line 363 of file slab.c.

References ALIGN\_START\_PTR, slab\_cache\_t::bufctl\_offset, slab\_t::cache, slab\_cache\_t::dtor, slab\_cache\_t::empty\_count, slab\_cache\_t::flags, slab\_t::free\_list, slab\_cache\_t::name, slab\_bufctl\_t::next, slab\_t::next, NULL, slab\_t::obj\_count, slab\_cache\_t::obj\_size, slab\_t::prev, printk(), SLAB\_POISON, SLAB\_POISON\_DEAD\_VALUE, SLAB\_RED\_ZONE, SLAB\_RED\_ZONE\_VALUE, SLAB\_SIZE, slab\_cache\_t::slabs\_empty, slab\_cache\_t::slabs\_full, and slab\_cache\_t::slabs\_partial.

Referenced by process\_create(), and vm\_pae\_destroy\_addr\_space().

```

363     {
364         /* compute address of slab data structure */
365         addr_t slab_start = ALIGN_START_PTR(buffer, SLAB_SIZE);
366         slab_t *slab = (slab_t *) (slab_start + SLAB_SIZE - sizeof(slab_t) );
367
368         /* obtain address of cache and bufctl */
369         slab_cache_t *cache = slab->cache;
370         slab_bufctl_t *bufctl = (slab_bufctl_t *) ((char *)buffer + cache->
371             bufctl_offset);
372
373         /* If slab is on the full slabs list, move it to the partial list
374          * since we are about to return a buffer to it. */
375         if(slab->free_list == NULL) {
376             /* remove from full slabs list */

```

```

376     if(cache->slabs_full == slab) {
377         cache->slabs_full = slab->next;
378     }
379     else {
380         slab->prev->next = slab->next;
381     }
382
383     if(slab->next != NULL) {
384         slab->next->prev = slab->prev;
385     }
386
387     /* add to partial slabs list */
388     slab->next = cache->slabs_partial;
389     cache->slabs_partial = slab;
390
391     if(slab->next != NULL) {
392         slab->next->prev = slab;
393     }
394 }
395
396 if(cache->flags & SLAB_RED_ZONE) {
397     uint32_t *rz_word = (uint32_t *)((char *)buffer + cache->obj_size);
398
399     if(*rz_word != SLAB_RED_ZONE_VALUE) {
400         printk("detected write past the end of object, cache: %s buffer: 0x%x value: 0x%x\n",
401             cache->name,
402             (unsigned int)buffer,
403             *rz_word
404         );
405     }
406
407     *rz_word = SLAB_RED_ZONE_VALUE;
408 }
409
410 if(cache->flags & SLAB_POISON) {
411     unsigned int idx;
412
413     if(cache->dtor != NULL) {
414         cache->dtor(buffer, cache->obj_size);
415     }
416
417     uint32_t *buffer32 = (uint32_t *)buffer;
418
419     for(idx = 0; idx < cache->obj_size / sizeof(uint32_t); ++idx) {
420         buffer32[idx] = SLAB_POISON_DEAD_VALUE;
421     }
422 }
423
424 /* link buffer into slab free list */
425 bufctl->next = slab->free_list;
426 slab->free_list = bufctl;
427 slab->obj_count -= 1;
428
429 /* If we just returned the last object to the slab, move the slab to
430  * the empty list. */
431 if(slab->obj_count == 0) {
432     /* remove from partial slabs list */
433     if(cache->slabs_partial == slab) {
434         cache->slabs_partial = slab->next;
435     }
436     else {
437         slab->prev->next = slab->next;
438     }
439
440     if(slab->next != NULL) {
441         slab->next->prev = slab->prev;
442     }
443
444     /* add to empty slabs list */
445     slab->next = cache->slabs_empty;
446     cache->slabs_empty = slab;
447
448     if(slab->next != NULL) {
449         slab->next->prev = slab;
450     }
451
452     ++(cache->empty_count);
453 }
454 }

```

Here is the call graph for this function:



**4.107.2.3** void slab\_cache\_init ( slab\_cache\_t \* cache, char \* name, size\_t size, size\_t alignment, slab\_ctor\_t ctor, slab\_dtor\_t dtor, int flags, boot\_alloc\_t \* boot\_alloc )

Initialize an object cache.

The following flags are supported:

- **SLAB\_HWCACHE\_ALIGN** Align objects on at least the line size of the CPU's data cache.
- **SLAB\_COMPACT** the bufctl can safely be put inside the object without destroying the constructed state. If not set, additional space is reserved specifically for the bufctl to prevent corruption of the constructed state.
- **SLAB\_RED\_ZONE** (redzone checking - debugging) Add a guard word at the end of each object and use this to detect writes past the end of the object.
- **SLAB\_POISON** (debugging) Fill uninitialized objects with a recognizable pattern before calling the constructor function to help identify members that do not get initialized. Do the same when freeing objects and use this to detect writes to freed objects.

This function uses the kernel's boot-time page allocator to allocate an initial slab. This helps with bootstrapping because it allows a few objects (up to a slab's worth) to be allocated before the main page allocator has been initialized and then replenished by user space. It also means this function can only be called during kernel initialization (it would not make sense to call it later).

#### Parameters

<i>cache</i>	the cache to initialize
<i>name</i>	a human-readable name for the cache, used in debugging messages
<i>size</i>	the size of objects allocated on this cache
<i>alignment</i>	the minimum object alignment, or zero for no constraint
<i>ctor</i>	the object constructor function
<i>dtor</i>	the object destructor function
<i>flags</i>	see description
<i>boot_alloc</i>	the kernel boot-time page allocator structure

ASSERTION: buffer size is at least the size of a pointer

ASSERTION: name is not NULL string

Definition at line 137 of file slab.c.

References ALIGN\_END, slab\_cache\_t::alignment, slab\_cache\_t::alloc\_size, assert, boot\_page\_alloc(), slab\_cache\_t::bufctl\_offset, cpu\_info, slab\_cache\_t::ctor, cpu\_info\_t::dcache\_alignment, slab\_cache\_t::dtor, slab\_cache\_t::empty\_count, slab\_cache\_t::flags, slab\_cache\_t::max\_colour, slab\_cache\_t::name, slab\_cache\_t::next\_colour, NULL, slab\_cache\_t::obj\_size, SLAB\_COMPACT, SLAB\_DEFAULT\_WORKING\_SET, SLAB\_HWCACHE\_ALIGN, SLAB\_POISON, SLAB\_RED\_ZONE, SLAB\_SIZE, slab\_cache\_t::slabs\_empty, slab\_cache\_t::slabs\_full, slab\_cache\_t::slabs\_partial, and slab\_cache\_t::working\_set.

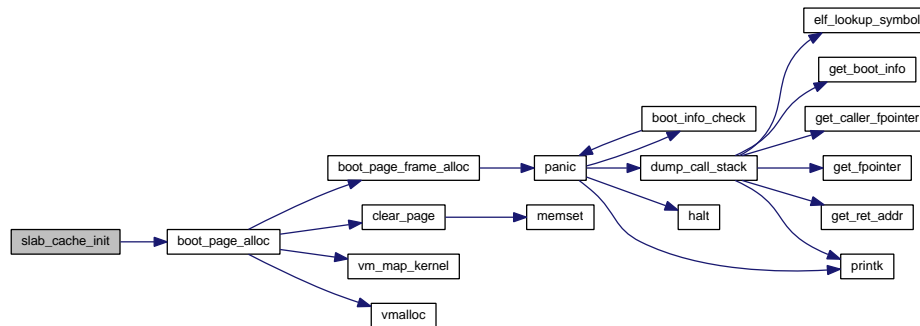
Referenced by ipc\_boot\_init(), process\_boot\_init(), and vm\_pae\_create\_pdpt\_cache().

```

145                                     {
146
147     assert(size >= sizeof(void *));
148
149     assert(name != NULL);
150
151     cache->name          = name;
152     cache->ctor           = ctor;
153     cache->dtor           = dtor;
154     cache->slabs_empty    = NULL;
155     cache->slabs_partial  = NULL;
156     cache->slabs_full     = NULL;
157     cache->empty_count    = 0;
158     cache->flags          = flags;
159     cache->next_colour    = 0;
160     cache->working_set    = SLAB_DEFAULT_WORKING_SET;
161
162     /* Compute actual alignment. */
163     if(alignment == 0) {
164         cache->alignment = sizeof(uint32_t);
165     }
166     else {
167         cache->alignment = alignment;
168     }
169
170     if((flags & SLAB_HWCACHE_ALIGN) && cache->alignment < cpu_info.
171     dcache_alignment) {
172         cache->alignment = cpu_info.dcache_alignment;
173     }
174
175     cache->alignment = ALIGN_END(cache->alignment, sizeof(uint32_t));
176
177     /* Reserve space for bufctl and/or redzone word. */
178     cache->obj_size = ALIGN_END(size, sizeof(uint32_t));
179
180     if((flags & SLAB_POISON) && (flags & SLAB_RED_ZONE)) {
181         /* bufctl and redzone word appended to buffer */
182         cache->alloc_size = cache->obj_size + sizeof(uint32_t) + sizeof(
183     slab_bufctl_t);
184     }
185     else if((flags & SLAB_POISON) || (flags & SLAB_RED_ZONE)) {
186         /* bufctl or redzone word appended to buffer (can be shared) */
187         cache->alloc_size = cache->obj_size + sizeof(uint32_t);
188     }
189     else if(ctor != NULL && ! (flags & SLAB_COMPACT)) {
190         /* If a constructor is defined, we cannot put the bufctl inside
191         * the object because that could overwrite constructed state,
192         * unless client explicitly says it's ok (SLAB_COMPACT flag). */
193         cache->alloc_size = cache->obj_size + sizeof(slab_bufctl_t);
194     }
195     else {
196         cache->alloc_size = cache->obj_size;
197     }
198
199     if(cache->alloc_size % cache->alignment != 0) {
200         cache->alloc_size += cache->alignment - cache->alloc_size % cache->
201     alignment;
202     }
203
204     size_t avail_space = SLAB_SIZE - sizeof(slab_t);
205
206     unsigned int buffers_per_slab = avail_space / cache->alloc_size;
207
208     size_t wasted_space = avail_space - buffers_per_slab * cache->alloc_size;
209
210     cache->max_colour = (wasted_space / cache->alignment) * cache->alignment;
211
212     cache->bufctl_offset = cache->alloc_size - sizeof(slab_bufctl_t);
213
214     /* Allocate first slab.
215     *
216     * This is needed to allow a few objects to be allocated during kernel
217     * initialization. */
218     init_and_add_slab(cache, boot_page_alloc(boot_alloc));
219 }

```

Here is the call graph for this function:



#### 4.107.2.4 void slab\_cache\_reap ( slab\_cache\_t \* cache )

Return memory to the page allocator.

Free slabs in excess to the cache's working set are finalized and freed.

##### Parameters

<i>cache</i>	the cache from which to reclaim memory
--------------	--

Definition at line 544 of file slab.c.

References slab\_cache\_t::empty\_count, slab\_t::next, slab\_cache\_t::slabs\_empty, and slab\_cache\_t::working\_set.

```

544
545     while(cache->empty_count > cache->working_set) {
546         /* select the first empty slab */
547         slab_t *slab = cache->slabs_empty;
548
549         /* unlink it and update count */
550         cache->slabs_empty = slab->next;
551         cache->empty_count -= 1;
552
553         /* destroy slab */
554         destroy_slab(cache, slab);
555     }
556 }

```

#### 4.107.2.5 void slab\_cache\_set\_working\_set ( slab\_cache\_t \* cache, unsigned int n )

Set a cache's working set.

The working set is defined as the number of free slabs the cache keeps for itself when pages are reclaimed from it. (This is terminology used in the Bonwick paper.) This provides some hysteresis to prevent slabs from being continuously created and destroyed, which requires calling the constructor and destructor functions on individual objects on the slabs.

##### Parameters

<i>cache</i>	the cache for which to set the working set
<i>n</i>	the size of the working set (number of pages)

Definition at line 571 of file slab.c.

References slab\_cache\_t::working\_set.



\_t::flags, get\_boot\_info(), ipc\_t::header, IPC\_FLAG\_NONE, IPC\_FLAG\_SYSTEM, ipc\_get\_proc\_object(), ipc\_object\_create(), ipc\_receive(), ipc\_reply(), ipc\_send(), JINUE\_EAGAIN, JINUE\_EINVAL, JINUE\_ENOSYS, JINUE\_IPC\_PROC, JINUE\_IPC\_SYSTEM, NULL, jinue\_mem\_map\_t::num\_entries, object\_ref\_t::object, OBJECT\_REF\_FLAG\_OWNED, OBJECT\_REF\_FLAG\_VALID, printk(), thread\_t::process, process\_get\_descriptor(), process\_unused\_descriptor(), jinue\_mem\_entry\_t::size, e820\_t::size, SYSCALL\_FUNCT\_CONSOLE\_PUTC, SYSCALL\_FUNCT\_CONSOLE\_PUTS, SYSCALL\_FUNCT\_CREATE\_IPC, SYSCALL\_FUNCT\_GET\_PHYS\_MEMORY, SYSCALL\_FUNCT\_GET\_THREAD\_LOCAL\_ADDR, SYSCALL\_FUNCT\_PROC\_BASE, SYSCALL\_FUNCT\_RECEIVE, SYSCALL\_FUNCT\_REPLY, SYSCALL\_FUNCT\_SET\_THREAD\_LOCAL\_ADDR, SYSCALL\_FUNCT\_SYSCALL\_METHOD, SYSCALL\_FUNCT\_SYSTEM\_BASE, SYSCALL\_FUNCT\_THREAD\_CREATE, SYSCALL\_FUNCT\_THREAD\_YIELD, syscall\_method, thread\_create(), thread\_yield\_from(), jinue\_mem\_entry\_t::type, and e820\_t::type.

Referenced by dispatch\_interrupt().

```

48                                     {
49     jinue_syscall_args_t *args = (jinue_syscall_args_t *)&trapframe->msg_arg0;
50
51     uintptr_t function_number = args->arg0;
52
53     if(function_number < SYSCALL_FUNCT_PROC_BASE) {
54         /* microkernel system calls */
55         switch(function_number) {
56
57             case SYSCALL_FUNCT_SYSCALL_METHOD:
58                 syscall_args_set_return(args, syscall_method);
59                 break;
60
61             case SYSCALL_FUNCT_CONSOLE_PUTC:
62                 console_putc(
63                     (char)args->arg1,
64                     CONSOLE_DEFAULT_COLOR);
65                 syscall_args_set_return(args, 0);
66                 break;
67
68             case SYSCALL_FUNCT_CONSOLE_PUTS:
69                 console_printn(
70                     (char *)args->arg2,
71                     jinue_args_get_data_size(args),
72                     CONSOLE_DEFAULT_COLOR);
73                 syscall_args_set_return(args, 0);
74                 break;
75
76             case SYSCALL_FUNCT_THREAD_CREATE:
77             {
78                 thread_t *thread = thread_create(
79                     /* TODO use arg1 as an address space reference if specified */
80                     get_current_thread()->process,
81                     (addr_t)args->arg2,
82                     (addr_t)args->arg3);
83
84                 if(thread == NULL) {
85                     syscall_args_set_error(args, JINUE_EAGAIN);
86                 }
87                 else {
88                     syscall_args_set_return(args, 0);
89                 }
90             }
91             break;
92
93             case SYSCALL_FUNCT_THREAD_YIELD:
94                 thread_yield_from(
95                     get_current_thread(),
96                     false, /* don't block */
97                     args->arg1); /* destroy (aka. exit) thread if true */
98                 syscall_args_set_return(args, 0);
99                 break;
100
101             case SYSCALL_FUNCT_SET_THREAD_LOCAL_ADDR:
102                 thread_context_set_local_storage(
103                     &get_current_thread()->thread_ctx,
104                     (addr_t)args->arg1,
105                     (size_t)args->arg2);
106                 syscall_args_set_return(args, 0);
107                 break;
108
109             case SYSCALL_FUNCT_GET_THREAD_LOCAL_ADDR:
110                 syscall_args_set_return_ptr(

```

```

114         args,
115         thread_context_get_local_storage(
116             &get_current_thread()->thread_ctx));
117     break;
118
119     case SYSCALL_FUNCT_GET_PHYS_MEMORY:
120     {
121         unsigned int idx;
122
123         size_t buffer_size = jinue_args_get_buffer_size(args);
124         jinue_mem_map_t *map = (jinue_mem_map_t *)jinue_args_get_buffer_ptr(args);
125         const boot_info_t *boot_info = get_boot_info();
126
127         if(buffer_size < sizeof(jinue_mem_map_t) + boot_info->e820_entries * sizeof(
128             jinue_mem_entry_t) ) {
129             syscall_args_set_error(args, JINUE_EINVAL);
130         }
131         else {
132             map->num_entries = boot_info->e820_entries;
133
134             for(idx = 0; idx < map->num_entries; ++idx) {
135                 map->entry[idx].addr = boot_info->e820_map[idx].addr;
136                 map->entry[idx].size = boot_info->e820_map[idx].size;
137                 map->entry[idx].type = boot_info->e820_map[idx].type;
138             }
139
140             syscall_args_set_return(args, 0);
141         }
142     }
143     break;
144
145     case SYSCALL_FUNCT_CREATE_IPC:
146     {
147         ipc_t *ipc;
148
149         thread_t *thread = get_current_thread();
150
151         int fd = process_unused_descriptor(thread->process);
152
153         if(fd < 0) {
154             syscall_args_set_error(args, JINUE_EAGAIN);
155             break;
156         }
157
158         if(args->arg1 & JINUE_IPC_PROC) {
159             ipc = ipc_get_proc_object();
160         }
161         else {
162             int flags = IPC_FLAG_NONE;
163
164             if(args->arg1 & JINUE_IPC_SYSTEM) {
165                 flags |= IPC_FLAG_SYSTEM;
166             }
167
168             ipc = ipc_object_create(flags);
169
170             if(ipc == NULL) {
171                 syscall_args_set_error(args, JINUE_EAGAIN);
172                 break;
173             }
174         }
175
176         object_ref_t *ref = process_get_descriptor(thread->process, fd);
177
178         object_addrref(&ipc->header);
179
180         ref->object = &ipc->header;
181         ref->flags = OBJECT_REF_FLAG_VALID | OBJECT_REF_FLAG_OWNER;
182         ref->cookie = 0;
183
184         syscall_args_set_return(args, fd);
185     }
186     break;
187
188     case SYSCALL_FUNCT_RECEIVE:
189         ipc_receive(args);
190         break;
191
192     case SYSCALL_FUNCT_REPLY:
193         ipc_reply(args);
194         break;

```

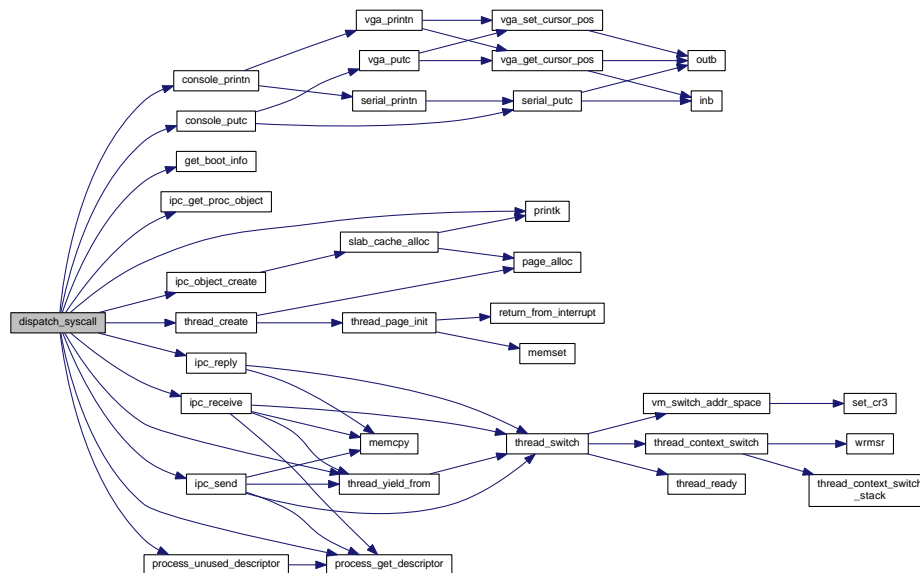


```

195
196     default:
197         printk("SYSCALL: function %u arg1=%u(0x%x) arg2=%u(0x%x) arg3=%u(0x%x)\n",
198             function_number,
199             args->arg1, args->arg1,
200             args->arg2, args->arg2,
201             args->arg3, args->arg3 );
202
203         syscall_args_set_error(args, JINUE_ENOSYS);
204     }
205 }
206 else if(function_number < SYSCALL_FUNCT_SYSTEM_BASE) {
207     /* process manager system calls */
208     printk("PROC SYSCALL: function %u arg1=%u(0x%x) arg2=%u(0x%x) arg3=%u(0x%x)\n",
209         function_number,
210         args->arg1, args->arg1,
211         args->arg2, args->arg2,
212         args->arg3, args->arg3 );
213
214     syscall_args_set_error(args, JINUE_ENOSYS);
215 }
216 else {
217     /* inter-process message */
218     ipc_send(args);
219 }
220 }

```

Here is the call graph for this function:

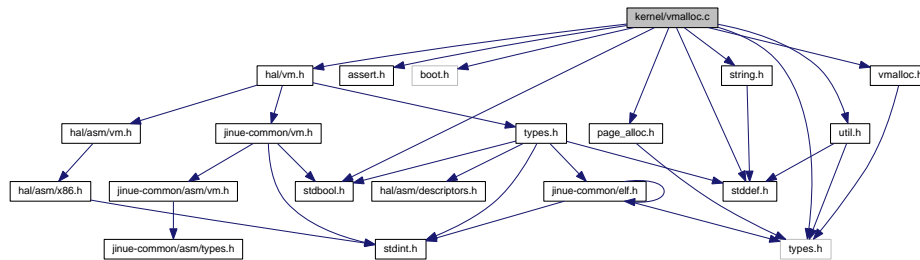


## 4.109 kernel/vmalloc.c File Reference

Virtual address space allocator.

```
#include <hal/vm.h>
#include <assert.h>
#include <boot.h>
#include <page_alloc.h>
#include <stdbool.h>
#include <stddef.h>
#include <string.h>
#include <types.h>
#include <util.h>
#include <vmalloc.h>
```

Include dependency graph for vmalloc.c:



## Data Structures

- struct **vmalloc\_t**
- struct **vmalloc\_block\_t**

## Macros

- #define **VMALLOC\_STACK\_ENTRIES** (PAGE\_SIZE / sizeof(addr\_t))
- #define **VMALLOC\_BLOCK\_SIZE** (VMALLOC\_STACK\_ENTRIES \* PAGE\_SIZE)

## Typedefs

- typedef struct **vmalloc\_block\_t** vmalloc\_block\_t

## Functions

- **addr\_t vmalloc** (void)  
*Allocate a page of virtual address space.*
- **void vmfree** (addr\_t page)  
*Free a page of virtual address space.*
- **void vmalloc\_init** (addr\_t start\_addr, addr\_t end\_addr, addr\_t preinit\_limit, boot\_alloc\_t \*boot\_alloc)  
*Basic initialization of the virtual memory allocator.*
- **bool vmalloc\_is\_in\_range** (addr\_t page)  
*Check whether the specified page is in the region managed by the allocator.*

### 4.109.1 Detailed Description

Virtual address space allocator. Functions in this file are used to manage the allocation of pages in a virtual address space region.

The kernel address space has only one region where allocations are managed by this allocator, the so-called allocations region. A different allocator is used during initialization - see boot.c. That other allocator manages a different, smaller region of the address space, the so-called image region.

While this allocation manages a single region of the address space, it is structured in such a way that it can easily be used to manage multiple address space regions independently, with each region being represented by a **vmalloc\_t** (p. 56) structure.

This allocator allocates pages one at a time. There is no way to allocate multiple contiguous pages after kernel initialization. (However, this is something the initialization-time allocator can do.

The allocator manages its address space region in identically-sized blocks of a few megabytes aligned on their size. Each block is represented by a **vmalloc\_block\_t** (p. 55) structure and has a stack to record available pages. The size of this stack is exactly one page, which is what determines the block size.

vm\_block\_t structures for an allocator are in an array that ensures the right vm\_block\_t structure can be found quickly during de-allocation. Non-depleted blocks are linked to a free list (a circular, doubly-linked list) that allows the allocator to quickly find a block with free pages during allocations.

Warning: the prev member (back pointer) of **vmalloc\_block\_t** (p. 55) is only meaningful while the block is linked to the free list. If there is a need to check whether the block is linked or not and vmalloc\_stack\_is\_empty() isn't appropriate, check if the next member (not prev) is NULL.

Definition in file **vmalloc.c**.

### 4.109.2 Macro Definition Documentation

#### 4.109.2.1 #define VMALLOC\_BLOCK\_SIZE (VMALLOC\_STACK\_ENTRIES \* PAGE\_SIZE)

Definition at line 84 of file vmalloc.c.

#### 4.109.2.2 #define VMALLOC\_STACK\_ENTRIES (PAGE\_SIZE / sizeof(addr\_t))

Definition at line 82 of file vmalloc.c.

### 4.109.3 Typedef Documentation

#### 4.109.3.1 typedef struct vmalloc\_block\_t vmalloc\_block\_t

Definition at line 124 of file vmalloc.c.

### 4.109.4 Function Documentation

#### 4.109.4.1 addr\_t vmalloc ( void )

Allocate a page of virtual address space.

**Returns**

address of allocated page or NULL if allocation failed

Definition at line 150 of file vmalloc.c.

Referenced by `add_page_frame()`, `boot_page_alloc()`, `elf_setup_stack()`, `vm_clone_page_directory()`, `vm_destroy_page_directory()`, `vm_pae_lookup_page_directory()`, and `vm_x86_lookup_page_directory()`.

```
150         {
151     return vmalloc_from(&kernel_vmallocator);
152 }
```

**4.109.4.2 void vmalloc\_init ( addr\_t start\_addr, addr\_t end\_addr, addr\_t preinit\_limit, boot\_alloc\_t \* boot\_alloc )**

Basic initialization of the virtual memory allocator.

This function initializes the allocator structure, and then initializes the first few blocks up to the limit set by the `preinit_limit` argument (more precisely, up to and including the block that contains `preinit_limit - 1`). TODO mention how to initialize the rest once this is implemented.

**Parameters**

<i>start_addr</i>	the start address of the region managed by the allocator
<i>end_addr</i>	the end address of the region managed by the allocator
<i>preinit_limit</i>	the limit address for preinitialized blocks
<i>boot_alloc</i>	the initialization-time page allocator structure

Definition at line 178 of file vmalloc.c.

Referenced by `vm_boot_postinit()`.

```
182         {
183
184     init_allocator(
185         &kernel_vmallocator,
186         start_addr,
187         end_addr,
188         preinit_limit,
189         boot_alloc);
190 }
```

**4.109.4.3 bool vmalloc\_is\_in\_range ( addr\_t page )**

Check whether the specified page is in the region managed by the allocator.

**Parameters**

<i>page</i>	the address of the page
-------------	-------------------------

**Returns**

true if it is in the region, false otherwise

Definition at line 199 of file vmalloc.c.

Referenced by `remove_page_frame()`.

```
199         {
200     return addr_is_in_initialized_range(&kernel_vmallocator, page);
201 }
```

4.109.4.4 void vmfree ( *addr\_t page* )

Free a page of virtual address space.

**Parameters**

<i>page</i>	the address of the page to free
-------------	---------------------------------

Definition at line 160 of file vmalloc.c.

Referenced by `elf_setup_stack()`, and `remove_page_frame()`.

```
160         {  
161     vmfree_to(&kernel_vmallocator, page);  
162 }
```

# Index

- \_\_assert\_failed
    - assert.h, 240
    - c-assert.c, 289
  - \_\_bool\_true\_false\_are\_defined
    - stdbool.h, 242
- a\_type
  - Elf32\_auxv\_t, 17
- a\_un
  - Elf32\_auxv\_t, 18
- a\_val
  - Elf32\_auxv\_t, 18
- ADDR\_4GB
  - hal/vm.h, 142
- ALIGN\_END
  - util.h, 273
- ALIGN\_END\_PTR
  - util.h, 273
- ALIGN\_START
  - util.h, 274
- ALIGN\_START\_PTR
  - util.h, 274
- AT\_BASE
  - jinue-common/elf.h, 86
- AT\_DCACHEBSIZE
  - jinue-common/elf.h, 86
- AT\_ENTRY
  - jinue-common/elf.h, 86
- AT\_EXECFD
  - jinue-common/elf.h, 86
- AT\_FLAGS
  - jinue-common/elf.h, 87
- AT\_HWCAP
  - jinue-common/elf.h, 87
- AT\_HWCAP2
  - jinue-common/elf.h, 87
- AT\_ICACHEBSIZE
  - jinue-common/elf.h, 87
- AT\_IGNORE
  - jinue-common/elf.h, 87
- AT\_NULL
  - jinue-common/elf.h, 87
- AT\_PAGESZ
  - jinue-common/elf.h, 87
- AT\_PHDR
  - jinue-common/elf.h, 87
- AT\_PHENT
  - jinue-common/elf.h, 88
- AT\_PHNUM
  - jinue-common/elf.h, 88
- AT\_STACKBASE
  - jinue-common/elf.h, 88
- AT\_SYSINFO\_EHDR
  - jinue-common/elf.h, 88
- AT\_UCACHEBSIZE
  - jinue-common/elf.h, 88
- abi.h
  - get\_caller\_fpointer, 104
  - get\_fpointer, 104
  - get\_program\_counter, 104
  - get\_ret\_addr, 104
- add\_page\_frame
  - page\_alloc.c, 369
  - page\_alloc.h, 250
- addr
  - e820\_t, 17
  - elf\_symbol\_t, 27
  - jinue\_mem\_entry\_t, 29
  - pseudo\_descriptor\_t, 41
- addr\_space
  - elf\_info\_t, 25
  - process\_t, 40
- addr\_space\_t, 7
  - cr3, 8
  - pd, 8
  - pdpt, 8
  - top\_level, 8
- addr\_t
  - hal/types.h, 178
- alignment
  - slab\_cache\_t, 43
- alloc\_backward
  - util.h, 274
- alloc\_forward
  - util.h, 274
- alloc\_page, 8
  - next, 9
- alloc\_size
  - slab\_cache\_t, 43
- any\_key
  - kbd.c, 364

- kbd.h, 236
- arg0
  - jinue\_syscall\_args\_t, 33
- arg1
  - jinue\_syscall\_args\_t, 33
- arg2
  - jinue\_syscall\_args\_t, 33
- arg3
  - jinue\_syscall\_args\_t, 33
- ascii.h
  - CHAR\_BS, 61
  - CHAR\_CR, 61
  - CHAR\_HT, 61
  - CHAR\_LF, 62
- asm/descriptors.h
  - GDT\_KERNEL\_CODE, 106
  - GDT\_KERNEL\_DATA, 106
  - GDT\_LENGTH, 106
  - GDT\_NULL, 106
  - GDT\_PER\_CPU\_DATA, 106
  - GDT\_TSS, 106
  - GDT\_USER\_CODE, 107
  - GDT\_USER\_DATA, 107
  - GDT\_USER\_TLS\_DATA, 107
  - RPL\_KERNEL, 107
  - RPL\_USER, 107
  - SEG\_FLAG\_16BIT, 107
  - SEG\_FLAG\_16BIT\_GATE, 107
  - SEG\_FLAG\_32BIT, 107
  - SEG\_FLAG\_32BIT\_GATE, 108
  - SEG\_FLAG\_BUSY, 108
  - SEG\_FLAG\_IN\_BYTES, 108
  - SEG\_FLAG\_IN\_PAGES, 108
  - SEG\_FLAG\_KERNEL, 108
  - SEG\_FLAG\_NORMAL, 108
  - SEG\_FLAG\_NORMAL\_GATE, 108
  - SEG\_FLAG\_NOSYSTEM, 109
  - SEG\_FLAG\_PRESENT, 109
  - SEG\_FLAG\_SYSTEM, 109
  - SEG\_FLAG\_TSS, 109
  - SEG\_FLAG\_USER, 109
  - SEG\_FLAGS\_OFFSET, 109
  - SEG\_SELECTOR, 109
  - SEG\_TYPE\_CALL\_GATE, 109
  - SEG\_TYPE\_CODE, 110
  - SEG\_TYPE\_DATA, 110
  - SEG\_TYPE\_INTERRUPT\_GATE, 110
  - SEG\_TYPE\_READ\_ONLY, 110
  - SEG\_TYPE\_TASK\_GATE, 110
  - SEG\_TYPE\_TRAP\_GATE, 110
  - SEG\_TYPE\_TSS, 110
  - TSS\_LIMIT, 110
- asm/mem.h
  - MEM\_ZONE\_DMA16\_END, 116
  - MEM\_ZONE\_DMA16\_START, 116
  - MEM\_ZONE\_MEM32\_END, 117
  - MEM\_ZONE\_MEM32\_START, 117
- asm/pic8259.h
  - PIC8259\_CASCADE\_INPUT, 121
  - PIC8259\_EOI, 121
  - PIC8259\_ICW1\_1, 121
  - PIC8259\_ICW1\_IC4, 121
  - PIC8259\_ICW1\_LTIM, 121
  - PIC8259\_ICW1\_SNGL, 122
  - PIC8259\_ICW4\_AEOI, 122
  - PIC8259\_ICW4\_UPM, 122
  - PIC8259\_IRQ\_COUNT, 122
  - PIC8259\_MASTER\_BASE, 122
  - PIC8259\_SLAVE\_BASE, 122
- asm/serial.h
  - SERIAL\_COM1\_IOPORT, 127
  - SERIAL\_COM2\_IOPORT, 127
  - SERIAL\_COM3\_IOPORT, 127
  - SERIAL\_COM4\_IOPORT, 127
  - SERIAL\_REG\_DATA\_BUFFER, 128
  - SERIAL\_REG\_DIVISOR\_HIGH, 128
  - SERIAL\_REG\_DIVISOR\_LOW, 128
  - SERIAL\_REG\_FIFO\_CTRL, 128
  - SERIAL\_REG\_INTR\_ENABLE, 128
  - SERIAL\_REG\_INTR\_ID, 128
  - SERIAL\_REG\_LINE\_CTRL, 128
  - SERIAL\_REG\_LINE\_STATUS, 128
  - SERIAL\_REG\_MODEM\_CTRL, 128
  - SERIAL\_REG\_MODEM\_STATUS, 129
  - SERIAL\_REG\_SCRATCH, 129
- asm/x86.h
  - X86\_CR0\_PG, 155
  - X86\_CR0\_WP, 155
  - X86\_CR4\_PAE, 155
  - X86\_CR4\_PGE, 155
  - X86\_CR4\_PSE, 155
  - X86\_PDE\_PAGE\_SIZE, 155
  - X86\_PTE\_ACCESSED, 156
  - X86\_PTE\_CACHE\_DISABLE, 156
  - X86\_PTE\_DIRTY, 156
  - X86\_PTE\_GLOBAL, 156
  - X86\_PTE\_NX, 156
  - X86\_PTE\_PRESENT, 156
  - X86\_PTE\_READ\_WRITE, 156
  - X86\_PTE\_USER, 156
  - X86\_PTE\_WRITE\_THROUGH, 156
- assert
  - assert.h, 240
- assert.h
  - \_\_assert\_failed, 240
  - assert, 240
- at\_phdr
  - elf\_info\_t, 25



at\_phent  
    elf\_info\_t, 26  
at\_phnum  
    elf\_info\_t, 26  
auxv\_t  
    jinue-common/elf.h, 97  
  
BOOT\_CMD\_LINE\_PTR  
    hal/asm/boot.h, 63  
BOOT\_DATA\_STRUCT  
    hal/asm/boot.h, 63  
BOOT\_E820\_ENTRIES  
    hal/asm/boot.h, 63  
BOOT\_E820\_MAP  
    hal/asm/boot.h, 63  
BOOT\_E820\_MAP\_END  
    hal/asm/boot.h, 63  
BOOT\_E820\_MAP\_SIZE  
    hal/asm/boot.h, 63  
BOOT\_MAGIC  
    hal/asm/boot.h, 63  
BOOT\_RAMDISK\_IMAGE  
    hal/asm/boot.h, 63  
BOOT\_RAMDISK\_SIZE  
    hal/asm/boot.h, 63  
BOOT\_SETUP  
    hal/asm/boot.h, 64  
BOOT\_SETUP32\_ADDR  
    hal/asm/boot.h, 64  
BOOT\_SETUP32\_SIZE  
    hal/asm/boot.h, 64  
BOOT\_SETUP\_HEADER  
    hal/asm/boot.h, 64  
BOOT\_SETUP\_MAGIC  
    hal/asm/boot.h, 64  
BOOT\_SETUP\_SECTS  
    hal/asm/boot.h, 64  
BOOT\_SIGNATURE  
    hal/asm/boot.h, 64  
BOOT\_STACK\_HEAP\_SIZE  
    hal/asm/boot.h, 64  
BOOT\_SYSIZE  
    hal/asm/boot.h, 64  
BUILD\_HOST  
    build-info.gen.h, 288  
BUILD\_TIME  
    build-info.gen.h, 288  
base\_addr  
    vmalloc\_t, 57  
block\_array  
    vmalloc\_t, 57  
block\_count  
    vmalloc\_t, 57  
bool  
    stdbool.h, 242  
boot.c  
    boot\_alloc\_init, 277  
    boot\_heap\_alloc\_size, 278  
    boot\_heap\_pop, 278  
    boot\_heap\_push, 279  
    boot\_page\_alloc, 279  
    boot\_page\_alloc\_early, 280  
    boot\_page\_alloc\_image, 282  
    boot\_page\_frame\_alloc, 283  
    boot\_vmalloc, 284  
boot\_alloc\_init  
    boot.c, 277  
    hal/boot.h, 66  
boot\_alloc\_t, 9  
    heap\_ptr, 10  
    heap\_pushed\_state, 10  
    its\_early, 10  
    kernel\_paddr\_limit, 10  
    kernel\_paddr\_top, 10  
    kernel\_vm\_limit, 10  
    kernel\_vm\_top, 10  
boot\_end  
    boot\_info\_t, 12  
boot\_heap  
    boot\_info\_t, 12  
boot\_heap\_alloc  
    hal/boot.h, 66  
boot\_heap\_alloc\_size  
    boot.c, 278  
    hal/boot.h, 66  
boot\_heap\_pop  
    boot.c, 278  
    hal/boot.h, 67  
boot\_heap\_push  
    boot.c, 279  
    hal/boot.h, 68  
boot\_heap\_pushed\_state, 10  
    next, 11  
boot\_info  
    hal/boot.c, 287  
boot\_info\_check  
    hal/boot.c, 285  
    hal/boot.h, 68  
boot\_info\_dump  
    hal/boot.c, 286  
    hal/boot.h, 69  
boot\_info\_t, 11  
    boot\_end, 12  
    boot\_heap, 12  
    cmdline, 12  
    e820\_entries, 12  
    e820\_map, 12  
    image\_start, 12

- image\_top, 13
- kernel\_size, 13
- kernel\_start, 13
- page\_directory, 13
- page\_table, 13
- proc\_size, 13
- proc\_start, 13
- ramdisk\_size, 13
- ramdisk\_start, 13
- setup\_signature, 14
- boot\_page\_alloc
  - boot.c, 279
  - hal/boot.h, 70
- boot\_page\_alloc\_early
  - boot.c, 280
  - hal/boot.h, 71
- boot\_page\_alloc\_image
  - boot.c, 282
  - hal/boot.h, 72
- boot\_page\_frame\_alloc
  - boot.c, 283
  - hal/boot.h, 73
- boot\_vmalloc
  - boot.c, 284
  - hal/boot.h, 74
- bufctl\_offset
  - slab\_cache\_t, 43
- buffer\_size
  - jinue\_message\_t, 31
  - message\_info\_t, 35
- build-info.gen.h
  - BUILD\_HOST, 288
  - BUILD\_TIME, 288
  - GIT\_REVISION, 288
- c-assert.c
  - \_\_assert\_failed, 289
- c-string.c
  - memcpy, 290
  - memset, 290
  - strlen, 291
- CHAR\_BS
  - ascii.h, 61
- CHAR\_CR
  - ascii.h, 61
- CHAR\_HT
  - ascii.h, 61
- CHAR\_LF
  - ascii.h, 62
- CONSOLE\_DEFAULT\_COLOR
  - jinue-common/console.h, 77
- CONSOLE\_SERIAL\_BAUD\_RATE
  - jinue-common/console.h, 77
- CONSOLE\_SERIAL\_IOPORT
  - jinue-common/console.h, 77
- CPU\_DATA\_ALIGNMENT
  - cpu\_data.h, 168
- CPU\_EFLAGS\_ID
  - cpu.h, 161
- CPU\_FEATURE\_CPUID
  - cpu.h, 161
- CPU\_FEATURE\_LOCAL\_APIC
  - cpu.h, 161
- CPU\_FEATURE\_PAE
  - cpu.h, 161
- CPU\_FEATURE\_SYSCALL
  - cpu.h, 162
- CPU\_FEATURE\_SYSENTER
  - cpu.h, 162
- CPU\_VENDOR\_AMD
  - cpu.h, 162
- CPU\_VENDOR\_AMD\_DW0
  - cpu.h, 162
- CPU\_VENDOR\_AMD\_DW1
  - cpu.h, 162
- CPU\_VENDOR\_AMD\_DW2
  - cpu.h, 162
- CPU\_VENDOR\_GENERIC
  - cpu.h, 162
- CPU\_VENDOR\_INTEL
  - cpu.h, 162
- CPU\_VENDOR\_INTEL\_DW0
  - cpu.h, 162
- CPU\_VENDOR\_INTEL\_DW1
  - cpu.h, 163
- CPU\_VENDOR\_INTEL\_DW2
  - cpu.h, 163
- CPUID\_EXT\_FEATURE\_SYSCALL
  - cpu.h, 163
- CPUID\_FEATURE\_APIC
  - cpu.h, 163
- CPUID\_FEATURE\_CLFLUSH
  - cpu.h, 163
- CPUID\_FEATURE\_FPU
  - cpu.h, 163
- CPUID\_FEATURE\_HTTP
  - cpu.h, 163
- CPUID\_FEATURE\_PAE
  - cpu.h, 163
- CPUID\_FEATURE\_SEP
  - cpu.h, 163
- cache
  - slab\_t, 46
- clear\_page
  - page\_alloc.c, 370
  - page\_alloc.h, 251
- cli
  - x86.h, 158

- cmdline
  - boot\_info\_t, 12
- colour
  - slab\_t, 46
- common/errno.h
  - JINUE\_E2BIG, 223
  - JINUE\_EAGAIN, 223
  - JINUE\_EBADF, 223
  - JINUE\_EINVAL, 223
  - JINUE\_EIO, 223
  - JINUE\_EMORE, 224
  - JINUE\_ENOMEM, 224
  - JINUE\_ENOSYS, 224
  - JINUE\_EPERM, 224
- common/list.h
  - JINUE\_LIST\_STATIC, 226
  - JINUE\_OFFSETOF, 226
  - jinue\_cursor\_entry, 226
  - jinue\_cursor\_t, 226
  - jinue\_list\_pop, 226
  - jinue\_node\_entry, 226
  - jinue\_node\_t, 226
- console.c
  - console\_init, 291
  - console\_print, 292
  - console\_printn, 292
  - console\_putc, 293
- console\_init
  - console.c, 291
  - jinue-common/console.h, 77
- console\_print
  - console.c, 292
  - jinue-common/console.h, 78
- console\_printn
  - console.c, 292
  - jinue-common/console.h, 78
- console\_putc
  - console.c, 293
  - jinue-common/console.h, 79
- cookie
  - jinue\_message\_t, 31
  - message\_info\_t, 35
  - object\_ref\_t, 37
- count
  - pfalloc\_cache\_t, 39
- cpu.c
  - cpu\_detect\_features, 303
  - cpu\_info, 306
  - cpu\_init\_data, 305
- cpu.h
  - CPU\_EFLAGS\_ID, 161
  - CPU\_FEATURE\_CPUID, 161
  - CPU\_FEATURE\_LOCAL\_APIC, 161
  - CPU\_FEATURE\_PAE, 161
  - CPU\_FEATURE\_SYSCALL, 162
  - CPU\_FEATURE\_SYSENTER, 162
  - CPU\_VENDOR\_AMD, 162
  - CPU\_VENDOR\_AMD\_DW0, 162
  - CPU\_VENDOR\_AMD\_DW1, 162
  - CPU\_VENDOR\_AMD\_DW2, 162
  - CPU\_VENDOR\_GENERIC, 162
  - CPU\_VENDOR\_INTEL, 162
  - CPU\_VENDOR\_INTEL\_DW0, 162
  - CPU\_VENDOR\_INTEL\_DW1, 163
  - CPU\_VENDOR\_INTEL\_DW2, 163
  - CPUID\_EXT\_FEATURE\_SYSCALL, 163
  - CPUID\_FEATURE\_APIC, 163
  - CPUID\_FEATURE\_CLFLUSH, 163
  - CPUID\_FEATURE\_FPU, 163
  - CPUID\_FEATURE\_HTT, 163
  - CPUID\_FEATURE\_PAE, 163
  - CPUID\_FEATURE\_SEP, 163
  - cpu\_detect\_features, 164
  - cpu\_info, 167
  - cpu\_init\_data, 166
  - MSR\_EFER, 164
  - MSR\_FLAG\_STAR\_SCE, 164
  - MSR\_IA32\_SYSENTER\_CS, 164
  - MSR\_IA32\_SYSENTER\_EIP, 164
  - MSR\_IA32\_SYSENTER\_ESP, 164
  - MSR\_STAR, 164
- cpu\_data.h
  - CPU\_DATA\_ALIGNMENT, 168
- cpu\_data\_t, 14
  - current\_addr\_space, 15
  - gdt, 15
  - hal/types.h, 178
  - self, 15
  - tss, 15
- cpu\_detect\_features
  - cpu.c, 303
  - cpu.h, 164
- cpu\_info
  - cpu.c, 306
  - cpu.h, 167
- cpu\_info\_t, 15
  - dcache\_alignment, 16
  - family, 16
  - features, 16
  - model, 16
  - stepping, 16
  - vendor, 16
- cpu\_init\_data
  - cpu.c, 305
  - cpu.h, 166
- cpuid
  - x86.h, 158
- cr3

- addr\_space\_t, 8
- tss\_t, 52
- cs
  - trapframe\_t, 50
  - tss\_t, 52
- ctor
  - slab\_cache\_t, 43
- current\_addr\_space
  - cpu\_data\_t, 15
- data\_size
  - jinue\_message\_t, 31
  - jinue\_reply\_t, 32
  - message\_info\_t, 35
- dcache\_alignment
  - cpu\_info\_t, 16
- debug
  - tss\_t, 52
- debug.c
  - dump\_call\_stack, 294
- debug.h
  - dump\_call\_stack, 80
- desc\_n
  - jinue\_message\_t, 31
  - jinue\_reply\_t, 32
  - message\_info\_t, 35
- descriptors
  - process\_t, 40
- descriptors.h
  - GATE\_DESCRIPTOR, 111
  - PACK\_DESCRIPTOR, 112
  - SEG\_DESCRIPTOR, 112
- dispatch\_interrupt
  - interrupt.c, 310
  - interrupt.h, 172
- dispatch\_syscall
  - syscall.c, 388
  - syscall.h, 233
- ds
  - trapframe\_t, 50
  - tss\_t, 53
- dtor
  - slab\_cache\_t, 43
- dump\_call\_stack
  - debug.c, 294
  - debug.h, 80
- e820.h
  - E820\_ACPI, 222
  - E820\_RAM, 222
  - E820\_RESERVED, 222
  - E820\_SMAP, 222
- E820\_ACPI
  - e820.h, 222
- E820\_RAM
  - e820.h, 222
- E820\_RESERVED
  - e820.h, 222
- E820\_SMAP
  - e820.h, 222
- e820\_entries
  - boot\_info\_t, 12
- e820\_map
  - boot\_info\_t, 12
- e820\_t, 16
  - addr, 17
  - size, 17
  - type, 17
- e\_ehsize
  - Elf32\_Ehdr, 18
- e\_entry
  - Elf32\_Ehdr, 18
- e\_flags
  - Elf32\_Ehdr, 19
- e\_ident
  - Elf32\_Ehdr, 19
- e\_machine
  - Elf32\_Ehdr, 19
- e\_phentsize
  - Elf32\_Ehdr, 19
- e\_phnum
  - Elf32\_Ehdr, 19
- e\_phoff
  - Elf32\_Ehdr, 19
- e\_shentsize
  - Elf32\_Ehdr, 19
- e\_shnum
  - Elf32\_Ehdr, 19
- e\_shoff
  - Elf32\_Ehdr, 19
- e\_shstrndx
  - Elf32\_Ehdr, 20
- e\_type
  - Elf32\_Ehdr, 20
- e\_version
  - Elf32\_Ehdr, 20
- EARLY\_PHYS\_TO\_VIRT
  - hal/vm.h, 142
- EARLY\_PTR\_TO\_PHYS\_ADDR
  - hal/vm.h, 142
- EARLY\_VIRT\_TO\_PHYS
  - hal/vm.h, 142
- EI\_CLASS
  - jinue-common/elf.h, 88
- EI\_DATA
  - jinue-common/elf.h, 88
- EI\_MAG0
  - jinue-common/elf.h, 89
- EI\_MAG1

- jinue-common/elf.h, 89
- EI\_MAG2
  - jinue-common/elf.h, 89
- EI\_MAG3
  - jinue-common/elf.h, 89
- EI\_NIDENT
  - jinue-common/elf.h, 89
- EI\_PAD
  - jinue-common/elf.h, 89
- EI\_VERSION
  - jinue-common/elf.h, 89
- ELF32\_ST\_BIND
  - jinue-common/elf.h, 89
- ELF32\_ST\_TYPE
  - jinue-common/elf.h, 90
- ELF\_MAGIC0
  - jinue-common/elf.h, 90
- ELF\_MAGIC1
  - jinue-common/elf.h, 90
- ELF\_MAGIC2
  - jinue-common/elf.h, 90
- ELF\_MAGIC3
  - jinue-common/elf.h, 90
- ELFCLASS32
  - jinue-common/elf.h, 90
- ELFCLASS64
  - jinue-common/elf.h, 90
- ELFCLASSNONE
  - jinue-common/elf.h, 90
- ELFDATA2LSB
  - jinue-common/elf.h, 91
- ELFDATA2MSB
  - jinue-common/elf.h, 91
- ELFDATANONE
  - jinue-common/elf.h, 91
- EM\_386
  - jinue-common/elf.h, 91
- EM\_AARCH64
  - jinue-common/elf.h, 91
- EM\_ALTERA\_NIOS2
  - jinue-common/elf.h, 91
- EM\_ARM
  - jinue-common/elf.h, 91
- EM\_MICROBLAZE
  - jinue-common/elf.h, 91
- EM\_MIPS
  - jinue-common/elf.h, 92
- EM\_NONE
  - jinue-common/elf.h, 92
- EM\_OPENRISC
  - jinue-common/elf.h, 92
- EM\_SPARC
  - jinue-common/elf.h, 92
- EM\_SPARC32PLUS
  - jinue-common/elf.h, 92
- EM\_X86\_64
  - jinue-common/elf.h, 92
- ET\_CORE
  - jinue-common/elf.h, 92
- ET\_DYN
  - jinue-common/elf.h, 92
- ET\_EXEC
  - jinue-common/elf.h, 92
- ET\_NONE
  - jinue-common/elf.h, 93
- ET\_REL
  - jinue-common/elf.h, 93
- EXCEPTION\_ALIGNMENT
  - irq.h, 113
- EXCEPTION\_BOUND
  - irq.h, 113
- EXCEPTION\_BREAK
  - irq.h, 114
- EXCEPTION\_DIV\_ZERO
  - irq.h, 114
- EXCEPTION\_DOUBLE\_FAULT
  - irq.h, 114
- EXCEPTION\_GENERAL\_PROTECTION
  - irq.h, 114
- EXCEPTION\_INVALID\_OP
  - irq.h, 114
- EXCEPTION\_INVALID\_TSS
  - irq.h, 114
- EXCEPTION\_MACHINE\_CHECK
  - irq.h, 114
- EXCEPTION\_MATH
  - irq.h, 114
- EXCEPTION\_NMI
  - irq.h, 114
- EXCEPTION\_NO\_COPROC
  - irq.h, 115
- EXCEPTION\_OVERFLOW
  - irq.h, 115
- EXCEPTION\_PAGE\_FAULT
  - irq.h, 115
- EXCEPTION\_SEGMENT\_NOT\_PRESENT
  - irq.h, 115
- EXCEPTION\_SIMD
  - irq.h, 115
- EXCEPTION\_STACK\_SEGMENT
  - irq.h, 115
- EXIT\_FAILURE
  - stdlib.h, 245
- EXIT\_SUCCESS
  - stdlib.h, 245
- eax
  - trapframe\_t, 50
  - tss\_t, 53

- x86\_cpuid\_regs\_t, 58
- ebp
  - kernel\_context\_t, 34
  - trapframe\_t, 50
  - tss\_t, 53
- ebx
  - kernel\_context\_t, 34
  - trapframe\_t, 50
  - tss\_t, 53
  - x86\_cpuid\_regs\_t, 58
- ecx
  - trapframe\_t, 50
  - tss\_t, 53
  - x86\_cpuid\_regs\_t, 58
- edi
  - kernel\_context\_t, 34
  - trapframe\_t, 50
  - tss\_t, 53
- edx
  - trapframe\_t, 50
  - tss\_t, 53
  - x86\_cpuid\_regs\_t, 58
- eflags
  - trapframe\_t, 50
  - tss\_t, 53
- eip
  - kernel\_context\_t, 34
  - trapframe\_t, 50
  - tss\_t, 53
- elf.c
  - elf\_check, 296
  - elf\_load, 297
  - elf\_lookup\_symbol, 299
  - elf\_setup\_stack, 300
- Elf32\_Addr
  - jinue-common/elf.h, 97
- Elf32\_Ehdr, 18
  - e\_ehsize, 18
  - e\_entry, 18
  - e\_flags, 19
  - e\_ident, 19
  - e\_machine, 19
  - e\_phentsize, 19
  - e\_phnum, 19
  - e\_phoff, 19
  - e\_shentsize, 19
  - e\_shnum, 19
  - e\_shoff, 19
  - e\_shstrndx, 20
  - e\_type, 20
  - e\_version, 20
- Elf32\_Half
  - jinue-common/elf.h, 97
- Elf32\_Off
  - jinue-common/elf.h, 97
- Elf32\_Phdr, 20
  - p\_align, 20
  - p\_filesz, 20
  - p\_flags, 21
  - p\_memsz, 21
  - p\_offset, 21
  - p\_paddr, 21
  - p\_type, 21
  - p\_vaddr, 21
- Elf32\_Shdr, 21
  - sh\_addr, 22
  - sh\_addralign, 22
  - sh\_entsize, 22
  - sh\_flags, 22
  - sh\_info, 22
  - sh\_link, 22
  - sh\_name, 22
  - sh\_offset, 22
  - sh\_size, 23
  - sh\_type, 23
- Elf32\_Sword
  - jinue-common/elf.h, 97
- Elf32\_Sym, 23
  - st\_info, 23
  - st\_name, 23
  - st\_other, 24
  - st\_shndx, 24
  - st\_size, 24
  - st\_value, 24
- Elf32\_Word
  - jinue-common/elf.h, 97
- Elf32\_auxv\_t, 17
  - a\_type, 17
  - a\_un, 18
  - a\_val, 18
- elf\_check
  - elf.c, 296
  - jinue-common/elf.h, 97
- elf\_info\_t, 24
  - addr\_space, 25
  - at\_phdr, 25
  - at\_phent, 26
  - at\_phnum, 26
  - entry, 26
  - stack\_addr, 26
- elf\_load
  - elf.c, 297
  - jinue-common/elf.h, 98
- elf\_lookup\_symbol
  - elf.c, 299
  - jinue-common/elf.h, 100
- elf\_setup\_stack
  - elf.c, 300

- jinue-common/elf.h, 101
- elf\_symbol\_t, 26
  - addr, 27
  - name, 27
- empty\_count
  - slab\_cache\_t, 44
- enable\_pae
  - x86.h, 158
- end\_addr
  - vmalloc\_t, 57
- entry
  - elf\_info\_t, 26
  - jinue\_mem\_map\_t, 30
  - pte\_t, 41
- errcode
  - trapframe\_t, 50
- es
  - trapframe\_t, 51
  - tss\_t, 53
- esi
  - kernel\_context\_t, 34
  - trapframe\_t, 51
  - tss\_t, 53
- esp
  - trapframe\_t, 51
  - tss\_t, 54
- esp0
  - tss\_t, 54
- esp1
  - tss\_t, 54
- esp2
  - tss\_t, 54
- false
  - stdbool.h, 242
- family
  - cpu\_info\_t, 16
- fast\_amd\_entry
  - trap.h, 176
- fast\_intel\_entry
  - trap.h, 176
- features
  - cpu\_info\_t, 16
- flags
  - object\_header\_t, 36
  - object\_ref\_t, 37
  - slab\_cache\_t, 44
- free\_list
  - slab\_t, 46
  - vmalloc\_t, 57
- fs
  - trapframe\_t, 51
  - tss\_t, 54
- function
  - jinue\_message\_t, 31
  - message\_info\_t, 35
- GATE\_DESCRIPTOR
  - descriptors.h, 111
- GB
  - jinue-common/asm/types.h, 180
- GDT\_KERNEL\_CODE
  - asm/descriptors.h, 106
- GDT\_KERNEL\_DATA
  - asm/descriptors.h, 106
- GDT\_LENGTH
  - asm/descriptors.h, 106
- GDT\_NULL
  - asm/descriptors.h, 106
- GDT\_PER\_CPU\_DATA
  - asm/descriptors.h, 106
- GDT\_TSS
  - asm/descriptors.h, 106
- GDT\_USER\_CODE
  - asm/descriptors.h, 107
- GDT\_USER\_DATA
  - asm/descriptors.h, 107
- GDT\_USER\_TLS\_DATA
  - asm/descriptors.h, 107
- GIT\_REVISION
  - build-info.gen.h, 288
- gdt
  - cpu\_data\_t, 15
- get\_boot\_info
  - hal/boot.c, 287
  - hal/boot.h, 75
- get\_caller\_fpointer
  - abi.h, 104
- get\_cr0
  - x86.h, 158
- get\_cr2
  - x86.h, 158
- get\_cr3
  - x86.h, 158
- get\_cr4
  - x86.h, 159
- get\_eflags
  - x86.h, 159
- get\_esp
  - x86.h, 159
- get\_fpointer
  - abi.h, 104
- get\_gs\_ptr
  - x86.h, 159
- get\_program\_counter
  - abi.h, 104
- get\_ret\_addr
  - abi.h, 104

- global\_page\_tables
  - vm.c, 341
  - vm\_private.h, 204
- global\_pfallloc\_cache
  - pfalloc.c, 376
  - pfalloc.h, 258
- gs
  - trapframe\_t, 51
  - tss\_t, 54
- HAS\_ERRCODE
  - irq.h, 115
- hal.c
  - hal\_init, 307
  - syscall\_method, 309
- hal.h
  - hal\_init, 169
- hal/asm/boot.h
  - BOOT\_CMD\_LINE\_PTR, 63
  - BOOT\_DATA\_STRUCT, 63
  - BOOT\_E820\_ENTRIES, 63
  - BOOT\_E820\_MAP, 63
  - BOOT\_E820\_MAP\_END, 63
  - BOOT\_E820\_MAP\_SIZE, 63
  - BOOT\_MAGIC, 63
  - BOOT\_RAMDISK\_IMAGE, 63
  - BOOT\_RAMDISK\_SIZE, 63
  - BOOT\_SETUP, 64
  - BOOT\_SETUP32\_ADDR, 64
  - BOOT\_SETUP32\_SIZE, 64
  - BOOT\_SETUP\_HEADER, 64
  - BOOT\_SETUP\_MAGIC, 64
  - BOOT\_SETUP\_SECTS, 64
  - BOOT\_SIGNATURE, 64
  - BOOT\_STACK\_HEAP\_SIZE, 64
  - BOOT\_SYSIZE, 64
- hal/asm/thread.h
  - THREAD\_CONTEXT\_MASK, 132
  - THREAD\_CONTEXT\_SIZE, 132
- hal/asm/vm.h
  - VM\_FLAG\_ACCESSED, 140
  - VM\_FLAG\_DIRTY, 140
  - VM\_FLAG\_KERNEL, 140
  - VM\_FLAG\_PRESENT, 140
  - VM\_FLAG\_READ\_ONLY, 140
  - VM\_FLAG\_READ\_WRITE, 140
  - VM\_FLAG\_USER, 141
- hal/boot.c
  - boot\_info, 287
  - boot\_info\_check, 285
  - boot\_info\_dump, 286
  - get\_boot\_info, 287
- hal/boot.h
  - boot\_alloc\_init, 66
  - boot\_heap\_alloc, 66
  - boot\_heap\_alloc\_size, 66
  - boot\_heap\_pop, 67
  - boot\_heap\_push, 68
  - boot\_info\_check, 68
  - boot\_info\_dump, 69
  - boot\_page\_alloc, 70
  - boot\_page\_alloc\_early, 71
  - boot\_page\_alloc\_image, 72
  - boot\_page\_frame\_alloc, 73
  - boot\_vmalloc, 74
  - get\_boot\_info, 75
- hal/thread.c
  - thread\_context\_switch, 321
  - thread\_context\_switch\_stack, 321
  - thread\_page\_init, 321
- hal/thread.h
  - thread\_context\_switch, 133
  - thread\_page\_init, 134
- hal/types.h
  - addr\_t, 178
  - cpu\_data\_t, 178
  - kern\_paddr\_t, 178
  - msg\_arg0, 177
  - msg\_arg1, 177
  - msg\_arg2, 178
  - msg\_arg3, 178
  - PFNULL, 178
  - pdpt\_t, 178
  - pte\_t, 178
  - seg\_descriptor\_t, 178
  - seg\_selector\_t, 179
  - user\_paddr\_t, 179
- hal/vm.h
  - ADDR\_4GB, 142
  - EARLY\_PHYS\_TO\_VIRT, 142
  - EARLY\_PTR\_TO\_PHYS\_ADDR, 142
  - EARLY\_VIRT\_TO\_PHYS, 142
  - initial\_addr\_space, 149
  - vm\_boot\_init, 143
  - vm\_boot\_postinit, 144
  - vm\_change\_flags, 145
  - vm\_create\_addr\_space, 145
  - vm\_create\_initial\_addr\_space, 146
  - vm\_destroy\_addr\_space, 147
  - vm\_lookup\_kernel\_paddr, 147
  - vm\_map\_early, 147
  - vm\_map\_kernel, 148
  - vm\_map\_user, 148
  - vm\_switch\_addr\_space, 148
  - vm\_unmap\_kernel, 149
  - vm\_unmap\_user, 149
- hal\_init
  - hal.c, 307



hal.h, 169  
halt  
  startup.h, 175  
head  
  jinue\_list\_t, 29  
header  
  ipc\_t, 28  
  process\_t, 40  
  thread\_t, 48  
heap\_ptr  
  boot\_alloc\_t, 10  
heap\_pushed\_state  
  boot\_alloc\_t, 10  
  
IDT\_LAST\_EXCEPTION  
  irq.h, 115  
IDT\_PIC8259\_BASE  
  irq.h, 115  
IDT\_VECTOR\_COUNT  
  irq.h, 116  
INT64\_C  
  stdint.h, 244  
IPC\_FLAG\_NONE  
  jinue-common/ipc.h, 214  
IPC\_FLAG\_SYSTEM  
  jinue-common/ipc.h, 214  
idt  
  interrupt.h, 173  
image\_start  
  boot\_info\_t, 12  
image\_top  
  boot\_info\_t, 13  
inb  
  io.h, 174  
include/ascii.h, 61  
include/boot.h, 62  
include/console.h, 76  
include/debug.h, 79  
include/elf.h, 81  
include/hal/abi.h, 103  
include/hal/asm/boot.h, 62  
include/hal/asm/descriptors.h, 104  
include/hal/asm/irq.h, 112  
include/hal/asm/mem.h, 116  
include/hal/asm/pic8259.h, 120  
include/hal/asm/serial.h, 127  
include/hal/asm/thread.h, 131  
include/hal/asm/vm.h, 139  
include/hal/asm/x86.h, 154  
include/hal/boot.h, 65  
include/hal/cpu.h, 160  
include/hal/cpu\_data.h, 168  
include/hal/descriptors.h, 111  
include/hal/hal.h, 169  
include/hal/interrupt.h, 171  
include/hal/io.h, 173  
include/hal/mem.h, 117  
include/hal/pic8259.h, 122  
include/hal/serial.h, 129  
include/hal/startup.h, 175  
include/hal/thread.h, 133  
include/hal/trap.h, 175  
include/hal/types.h, 176  
include/hal/vga.h, 182  
include/hal/vm.h, 141  
include/hal/vm\_pae.h, 190  
include/hal/vm\_private.h, 199  
include/hal/vm\_x86.h, 204  
include/hal/x86.h, 157  
include/ipc.h, 210  
include/jinue-common/asm/e820.h, 221  
include/jinue-common/asm/ipc.h, 211  
include/jinue-common/asm/syscall.h, 228  
include/jinue-common/asm/types.h, 179  
include/jinue-common/asm/vm.h, 150  
include/jinue-common/console.h, 76  
include/jinue-common/elf.h, 82  
include/jinue-common/errno.h, 223  
include/jinue-common/ipc.h, 213  
include/jinue-common/list.h, 225  
include/jinue-common/syscall.h, 232  
include/jinue-common/types.h, 180  
include/jinue-common/vm.h, 152  
include/jinue/console.h, 76  
include/jinue/elf.h, 81  
include/jinue/errno.h, 222  
include/jinue/ipc.h, 210  
include/jinue/list.h, 224  
include/jinue/memory.h, 226  
include/jinue/syscall.h, 227  
include/jinue/types.h, 179  
include/jinue/vm.h, 149  
include/kbd.h, 236  
include/kmain.h, 237  
include/kstdc/assert.h, 239  
include/kstdc/stdarg.h, 241  
include/kstdc/stdbool.h, 241  
include/kstdc/stddef.h, 242  
include/kstdc/stdint.h, 243  
include/kstdc/stdlib.h, 245  
include/kstdc/string.h, 246  
include/object.h, 247  
include/page\_alloc.h, 249  
include/panic.h, 254  
include/pfalloc.h, 255  
include/printk.h, 258  
include/process.h, 259  
include/slab.h, 262

- include/syscall.h, 232
- include/thread.h, 135
- include/types.h, 181
- include/util.h, 273
- include/vmalloc.h, 274
- init\_limit
  - vmalloc\_t, 57
- init\_pfallloc\_cache
  - pfalloc.c, 375
  - pfalloc.h, 257
- initial\_addr\_space
  - hal/vm.h, 149
  - vm.c, 341
- initial\_pdpt
  - vm\_pae.c, 351
- inl
  - io.h, 174
- int16\_t
  - stdint.h, 244
- int32\_t
  - stdint.h, 244
- int64\_t
  - stdint.h, 244
- int8\_t
  - stdint.h, 244
- interrupt.c
  - dispatch\_interrupt, 310
- interrupt.h
  - dispatch\_interrupt, 172
  - idt, 173
- intptr\_t
  - stdint.h, 244
- invalidate\_tlb
  - x86.h, 159
- inw
  - io.h, 174
- io.h
  - inb, 174
  - inl, 174
  - inw, 174
  - iodelay, 174
  - outb, 174
  - outl, 174
  - outw, 174
- iodelay
  - io.h, 174
- iomap
  - tss\_t, 54
- ipc.c
  - ipc\_boot\_init, 358
  - ipc\_get\_proc\_object, 358
  - ipc\_object\_create, 358
  - ipc\_receive, 359
  - ipc\_reply, 361
  - ipc\_send, 362
- ipc\_boot\_init
  - ipc.c, 358
  - jinue-common/ipc.h, 215
- ipc\_get\_proc\_object
  - ipc.c, 358
  - jinue-common/ipc.h, 215
- ipc\_object\_create
  - ipc.c, 358
  - jinue-common/ipc.h, 215
- ipc\_receive
  - ipc.c, 359
  - jinue-common/ipc.h, 216
- ipc\_reply
  - ipc.c, 361
  - jinue-common/ipc.h, 218
- ipc\_send
  - ipc.c, 362
  - jinue-common/ipc.h, 219
- ipc\_t, 27
  - header, 28
  - recv\_list, 28
  - send\_list, 28
- irq.h
  - EXCEPTION\_ALIGNMENT, 113
  - EXCEPTION\_BOUND, 113
  - EXCEPTION\_BREAK, 114
  - EXCEPTION\_DIV\_ZERO, 114
  - EXCEPTION\_DOUBLE\_FAULT, 114
  - EXCEPTION\_GENERAL\_PROTECTION, 114
  - EXCEPTION\_INVALID\_OP, 114
  - EXCEPTION\_INVALID\_TSS, 114
  - EXCEPTION\_MACHINE\_CHECK, 114
  - EXCEPTION\_MATH, 114
  - EXCEPTION\_NMI, 114
  - EXCEPTION\_NO\_COPROC, 115
  - EXCEPTION\_OVERFLOW, 115
  - EXCEPTION\_PAGE\_FAULT, 115
  - EXCEPTION\_SEGMENT\_NOT\_PRESENT, 115
  - EXCEPTION\_SIMD, 115
  - EXCEPTION\_STACK\_SEGMENT, 115
  - HAS\_ERRCODE, 115
  - IDT\_LAST\_EXCEPTION, 115
  - IDT\_PIC8259\_BASE, 115
  - IDT\_VECTOR\_COUNT, 116
- its\_early
  - boot\_alloc\_t, 10
- ivt
  - trapframe\_t, 51
- JINUE\_ARGS\_PACK\_BUFFER\_SIZE
  - jinue-common/asm/ipc.h, 211
- JINUE\_ARGS\_PACK\_DATA\_SIZE
  - jinue-common/asm/ipc.h, 211

- JINUE\_ARGS\_PACK\_N\_DESC
  - jinue-common/asm/ipc.h, 212
- JINUE\_E2BIG
  - common/errno.h, 223
- JINUE\_EAGAIN
  - common/errno.h, 223
- JINUE\_EBADF
  - common/errno.h, 223
- JINUE\_EINVAL
  - common/errno.h, 223
- JINUE\_EIO
  - common/errno.h, 223
- JINUE\_EMORE
  - common/errno.h, 224
- JINUE\_ENOMEM
  - common/errno.h, 224
- JINUE\_ENOSYS
  - common/errno.h, 224
- JINUE\_EPERM
  - common/errno.h, 224
- JINUE\_IPC\_NONE
  - jinue-common/ipc.h, 214
- JINUE\_IPC\_PROC
  - jinue-common/ipc.h, 214
- JINUE\_IPC\_SYSTEM
  - jinue-common/ipc.h, 214
- JINUE\_LIST\_STATIC
  - common/list.h, 226
- JINUE\_OFFSETOF
  - common/list.h, 226
- JINUE\_SEND\_BUFFER\_SIZE\_OFFSET
  - jinue-common/asm/ipc.h, 212
- JINUE\_SEND\_DATA\_SIZE\_OFFSET
  - jinue-common/asm/ipc.h, 212
- JINUE\_SEND\_MAX\_N\_DESC
  - jinue-common/asm/ipc.h, 212
- JINUE\_SEND\_MAX\_SIZE
  - jinue-common/asm/ipc.h, 212
- JINUE\_SEND\_N\_DESC\_BITS
  - jinue-common/asm/ipc.h, 212
- JINUE\_SEND\_N\_DESC\_MASK
  - jinue-common/asm/ipc.h, 212
- JINUE\_SEND\_N\_DESC\_OFFSET
  - jinue-common/asm/ipc.h, 212
- JINUE\_SEND\_SIZE\_BITS
  - jinue-common/asm/ipc.h, 213
- JINUE\_SEND\_SIZE\_MASK
  - jinue-common/asm/ipc.h, 213
- jinue-common/asm/ipc.h
  - JINUE\_ARGS\_PACK\_BUFFER\_SIZE, 211
  - JINUE\_ARGS\_PACK\_DATA\_SIZE, 211
  - JINUE\_ARGS\_PACK\_N\_DESC, 212
  - JINUE\_SEND\_BUFFER\_SIZE\_OFFSET, 212
  - JINUE\_SEND\_DATA\_SIZE\_OFFSET, 212
  - JINUE\_SEND\_MAX\_N\_DESC, 212
  - JINUE\_SEND\_MAX\_SIZE, 212
  - JINUE\_SEND\_N\_DESC\_BITS, 212
  - JINUE\_SEND\_N\_DESC\_MASK, 212
  - JINUE\_SEND\_N\_DESC\_OFFSET, 212
  - JINUE\_SEND\_SIZE\_BITS, 213
  - JINUE\_SEND\_SIZE\_MASK, 213
- jinue-common/asm/syscall.h
  - SYSCALL\_FUNCT\_CONSOLE\_PUTC, 229
  - SYSCALL\_FUNCT\_CONSOLE\_PUTS, 229
  - SYSCALL\_FUNCT\_CREATE\_IPC, 229
  - SYSCALL\_FUNCT\_GET\_PHYS\_MEMORY, 230
  - SYSCALL\_FUNCT\_GET\_THREAD\_LOCAL\_ADDR, 230
  - SYSCALL\_FUNCT\_PROC\_BASE, 230
  - SYSCALL\_FUNCT\_RECEIVE, 230
  - SYSCALL\_FUNCT\_REPLY, 230
  - SYSCALL\_FUNCT\_SET\_THREAD\_LOCAL\_ADDR, 230
  - SYSCALL\_FUNCT\_SYSCALL\_METHOD, 230
  - SYSCALL\_FUNCT\_SYSTEM\_BASE, 231
  - SYSCALL\_FUNCT\_THREAD\_CREATE, 231
  - SYSCALL\_FUNCT\_THREAD\_YIELD, 231
  - SYSCALL\_FUNCT\_USER\_BASE, 231
  - SYSCALL\_IRQ, 231
  - SYSCALL\_METHOD\_FAST\_AMD, 231
  - SYSCALL\_METHOD\_FAST\_INTEL, 231
  - SYSCALL\_METHOD\_INTR, 232
- jinue-common/asm/types.h
  - GB, 180
  - KB, 180
  - MB, 180
- jinue-common/asm/vm.h
  - KERNEL\_EARLY\_LIMIT, 151
  - KERNEL\_IMAGE\_END, 151
  - KERNEL\_PREALLOC\_LIMIT, 151
  - KLIMIT, 151
  - PAGE\_BITS, 152
  - PAGE\_MASK, 152
  - PAGE\_SIZE, 152
  - STACK\_BASE, 152
  - STACK\_SIZE, 152
  - STACK\_START, 152
- jinue-common/console.h
  - CONSOLE\_DEFAULT\_COLOR, 77
  - CONSOLE\_SERIAL\_BAUD\_RATE, 77
  - CONSOLE\_SERIAL\_IOPORT, 77
  - console\_init, 77
  - console\_print, 78
  - console\_printn, 78
  - console\_putc, 79
- jinue-common/elf.h
  - AT\_BASE, 86
  - AT\_DCACHEBSIZE, 86

- AT\_ENTRY, 86
- AT\_EXECFD, 86
- AT\_FLAGS, 87
- AT\_HWCAP, 87
- AT\_HWCAP2, 87
- AT\_ICACHEBSIZE, 87
- AT\_IGNORE, 87
- AT\_NULL, 87
- AT\_PAGESZ, 87
- AT\_PHDR, 87
- AT\_PHEMT, 88
- AT\_PHNUM, 88
- AT\_STACKBASE, 88
- AT\_SYSINFO\_EHDR, 88
- AT\_UCACHEBSIZE, 88
- auxv\_t, 97
- EI\_CLASS, 88
- EI\_DATA, 88
- EI\_MAG0, 89
- EI\_MAG1, 89
- EI\_MAG2, 89
- EI\_MAG3, 89
- EI\_NIDENT, 89
- EI\_PAD, 89
- EI\_VERSION, 89
- ELF32\_ST\_BIND, 89
- ELF32\_ST\_TYPE, 90
- ELF\_MAGIC0, 90
- ELF\_MAGIC1, 90
- ELF\_MAGIC2, 90
- ELF\_MAGIC3, 90
- ELFCLASS32, 90
- ELFCLASS64, 90
- ELFCLASSNONE, 90
- ELFDATA2LSB, 91
- ELFDATA2MSB, 91
- ELFDATANONE, 91
- EM\_386, 91
- EM\_AARCH64, 91
- EM\_ALTERA\_NIOS2, 91
- EM\_ARM, 91
- EM\_MICROBLAZE, 91
- EM\_MIPS, 92
- EM\_NONE, 92
- EM\_OPENRISC, 92
- EM\_SPARC, 92
- EM\_SPARC32PLUS, 92
- EM\_X86\_64, 92
- ET\_CORE, 92
- ET\_DYN, 92
- ET\_EXEC, 92
- ET\_NONE, 93
- ET\_REL, 93
- Elf32\_Addr, 97
- Elf32\_Half, 97
- Elf32\_Off, 97
- Elf32\_Sword, 97
- Elf32\_Word, 97
- elf\_check, 97
- elf\_load, 98
- elf\_lookup\_symbol, 100
- elf\_setup\_stack, 101
- PF\_R, 93
- PF\_W, 93
- PF\_X, 93
- PT\_DYNAMIC, 93
- PT\_INTERP, 93
- PT\_LOAD, 93
- PT\_NOTE, 93
- PT\_NULL, 94
- PT\_PHDR, 94
- PT\_SHLIB, 94
- SHT\_DYNAMIC, 94
- SHT\_DYNSYM, 94
- SHT\_HASH, 94
- SHT\_NOBITS, 94
- SHT\_NOTE, 94
- SHT\_NULL, 94
- SHT\_PROGBITS, 95
- SHT\_REL, 95
- SHT\_RELA, 95
- SHT\_SHLIB, 95
- SHT\_STRTAB, 95
- SHT\_SYMTAB, 95
- STB\_GLOBAL, 95
- STB\_LOCAL, 95
- STB\_WEAK, 96
- STN\_UNDEF, 96
- STT\_FILE, 96
- STT\_FUNCTION, 96
- STT\_NOTYPE, 96
- STT\_OBJECT, 96
- STT\_SECTION, 96
- jinue-common/ipc.h
  - IPC\_FLAG\_NONE, 214
  - IPC\_FLAG\_SYSTEM, 214
  - ipc\_boot\_init, 215
  - ipc\_get\_proc\_object, 215
  - ipc\_object\_create, 215
  - ipc\_receive, 216
  - ipc\_reply, 218
  - ipc\_send, 219
  - JINUE\_IPC\_NONE, 214
  - JINUE\_IPC\_PROC, 214
  - JINUE\_IPC\_SYSTEM, 214
  - jinue\_ipc\_descriptor\_t, 215
- jinue-common/vm.h
  - page\_address\_of, 153

- page\_number\_of, 153
- page\_offset\_of, 153
- jinue/ipc.h
  - jinue\_create\_ipc, 211
  - jinue\_receive, 211
  - jinue\_reply, 211
  - jinue\_send, 211
- jinue/syscall.h
  - jinue\_call, 228
  - jinue\_call\_raw, 228
  - jinue\_get\_syscall\_implementation, 228
  - jinue\_get\_syscall\_implementation\_name, 228
  - jinue\_get\_thread\_local\_storage, 228
  - jinue\_set\_thread\_local\_storage, 228
  - jinue\_thread\_create, 228
  - jinue\_thread\_exit, 228
  - jinue\_yield, 228
- jinue\_call
  - jinue/syscall.h, 228
- jinue\_call\_raw
  - jinue/syscall.h, 228
- jinue\_create\_ipc
  - jinue/ipc.h, 211
- jinue\_cursor\_entry
  - common/list.h, 226
- jinue\_cursor\_t
  - common/list.h, 226
- jinue\_get\_phys\_memory
  - memory.h, 227
- jinue\_get\_syscall\_implementation
  - jinue/syscall.h, 228
- jinue\_get\_syscall\_implementation\_name
  - jinue/syscall.h, 228
- jinue\_get\_thread\_local\_storage
  - jinue/syscall.h, 228
- jinue\_ipc\_descriptor\_t
  - jinue-common/ipc.h, 215
- jinue\_list\_pop
  - common/list.h, 226
- jinue\_list\_t, 28
  - head, 29
  - tail, 29
- jinue\_mem\_entry\_t, 29
  - addr, 29
  - size, 29
  - type, 29
- jinue\_mem\_map\_t, 30
  - entry, 30
  - num\_entries, 30
- jinue\_message\_t, 31
  - buffer\_size, 31
  - cookie, 31
  - data\_size, 31
  - desc\_n, 31
  - function, 31
- jinue\_node\_entry
  - common/list.h, 226
- jinue\_node\_t, 31
  - common/list.h, 226
  - next, 32
- jinue\_pys\_mem\_type\_description
  - memory.h, 227
- jinue\_receive
  - jinue/ipc.h, 211
- jinue\_reply
  - jinue/ipc.h, 211
- jinue\_reply\_t, 32
  - data\_size, 32
  - desc\_n, 32
- jinue\_send
  - jinue/ipc.h, 211
- jinue\_set\_thread\_local\_storage
  - jinue/syscall.h, 228
- jinue\_syscall\_args\_t, 33
  - arg0, 33
  - arg1, 33
  - arg2, 33
  - arg3, 33
- jinue\_thread\_create
  - jinue/syscall.h, 228
- jinue\_thread\_exit
  - jinue/syscall.h, 228
- jinue\_yield
  - jinue/syscall.h, 228
- KB
  - jinue-common/asm/types.h, 180
- KERNEL\_EARLY\_LIMIT
  - jinue-common/asm/vm.h, 151
- KERNEL\_IMAGE\_END
  - jinue-common/asm/vm.h, 151
- KERNEL\_PAGE\_STACK\_INIT
  - pfalloc.h, 256
- KERNEL\_PAGE\_STACK\_SIZE
  - pfalloc.h, 256
- KERNEL\_PREALLOC\_LIMIT
  - jinue-common/asm/vm.h, 151
- KLIMIT
  - jinue-common/asm/vm.h, 151
- kbd.c
  - any\_key, 364
- kbd.h
  - any\_key, 236
- kern\_paddr\_t
  - hal/types.h, 178
- kernel/boot.c, 276
- kernel/build-info.gen.h, 288
- kernel/c-assert.c, 288

- kernel/c-string.c, 290
- kernel/console.c, 291
- kernel/debug.c, 293
- kernel/elf.c, 295
- kernel/hal/boot.c, 285
- kernel/hal/cpu.c, 302
- kernel/hal/hal.c, 306
- kernel/hal/interrupt.c, 309
- kernel/hal/mem.c, 311
- kernel/hal/pic8259.c, 314
- kernel/hal/serial.c, 318
- kernel/hal/thread.c, 320
- kernel/hal/vga.c, 326
- kernel/hal/vm.c, 331
- kernel/hal/vm\_pae.c, 342
- kernel/hal/vm\_x86.c, 351
- kernel/ipc.c, 357
- kernel/kbd.c, 364
- kernel/kmain.c, 365
- kernel/page\_alloc.c, 368
- kernel/panic.c, 373
- kernel/pfalloc.c, 374
- kernel/process.c, 377
- kernel/slab.c, 380
- kernel/syscall.c, 388
- kernel/thread.c, 322
- kernel/vmalloc.c, 391
- kernel\_context\_t, 34
  - ebp, 34
  - ebx, 34
  - edi, 34
  - eip, 34
  - esi, 34
- kernel\_paddr\_limit
  - boot\_alloc\_t, 10
- kernel\_paddr\_top
  - boot\_alloc\_t, 10
- kernel\_size
  - boot\_info\_t, 13
- kernel\_start
  - boot\_info\_t, 13
- kernel\_vm\_limit
  - boot\_alloc\_t, 10
- kernel\_vm\_top
  - boot\_alloc\_t, 10
- kmain
  - kmain.c, 366
  - kmain.h, 237
- kmain.c
  - kmain, 366
- kmain.h
  - kmain, 237
- ldt
  - tss\_t, 54
- lgdt
  - x86.h, 159
- lidt
  - x86.h, 159
- limit
  - pseudo\_descriptor\_t, 41
- local\_storage\_addr
  - thread\_context\_t, 47
- local\_storage\_size
  - thread\_context\_t, 47
- ltr
  - x86.h, 159
- MB
  - jinue-common/asm/types.h, 180
- MEM\_ZONE\_DMA16\_END
  - asm/mem.h, 116
- MEM\_ZONE\_DMA16\_START
  - asm/mem.h, 116
- MEM\_ZONE\_MEM32\_END
  - asm/mem.h, 117
- MEM\_ZONE\_MEM32\_START
  - asm/mem.h, 117
- MSR\_EFER
  - cpu.h, 164
- MSR\_FLAG\_STAR\_SCE
  - cpu.h, 164
- MSR\_IA32\_SYSENTER\_CS
  - cpu.h, 164
- MSR\_IA32\_SYSENTER\_EIP
  - cpu.h, 164
- MSR\_IA32\_SYSENTER\_ESP
  - cpu.h, 164
- MSR\_STAR
  - cpu.h, 164
- max\_colour
  - slab\_cache\_t, 44
- mem.c
  - mem\_check\_memory, 312
- mem.h
  - mem\_check\_memory, 118
- mem\_check\_memory
  - mem.c, 312
  - mem.h, 118
- memcpy
  - c-string.c, 290
  - string.h, 246
- memory.h
  - jinue\_get\_phys\_memory, 227
  - jinue\_pys\_mem\_type\_description, 227
- memset
  - c-string.c, 290
  - string.h, 246

- message\_args
  - thread\_t, 48
- message\_buffer
  - thread\_t, 48
- message\_info
  - thread\_t, 48
- message\_info\_t, 35
  - buffer\_size, 35
  - cookie, 35
  - data\_size, 35
  - desc\_n, 35
  - function, 35
  - total\_size, 35
- model
  - cpu\_info\_t, 16
- msg\_arg0
  - hal/types.h, 177
- msg\_arg1
  - hal/types.h, 177
- msg\_arg2
  - hal/types.h, 178
- msg\_arg3
  - hal/types.h, 178
- msr\_addr\_t
  - x86.h, 158
- NULL
  - stddef.h, 243
- name
  - elf\_symbol\_t, 27
  - slab\_cache\_t, 44
- next
  - alloc\_page, 9
  - boot\_heap\_pushed\_state, 11
  - jinue\_node\_t, 32
  - slab\_bufctl\_t, 42
  - slab\_t, 46
  - vmalloc\_block\_t, 56
- next\_colour
  - slab\_cache\_t, 44
- num\_entries
  - jinue\_mem\_map\_t, 30
- OBJECT\_FLAG\_DESTROYED
  - object.h, 248
- OBJECT\_FLAG\_NONE
  - object.h, 248
- OBJECT\_REF\_FLAG\_CLOSED
  - object.h, 248
- OBJECT\_REF\_FLAG\_NONE
  - object.h, 248
- OBJECT\_REF\_FLAG\_OWNER
  - object.h, 248
- OBJECT\_REF\_FLAG\_VALID
  - object.h, 248
- OBJECT\_TYPE\_IPC
  - object.h, 249
- OBJECT\_TYPE\_PROCESS
  - object.h, 249
- OBJECT\_TYPE\_THREAD
  - object.h, 249
- OFFSET\_OF\_PTR
  - util.h, 274
- obj\_count
  - slab\_t, 46
- obj\_size
  - slab\_cache\_t, 44
- object
  - object\_ref\_t, 37
- object.h
  - OBJECT\_FLAG\_DESTROYED, 248
  - OBJECT\_FLAG\_NONE, 248
  - OBJECT\_REF\_FLAG\_CLOSED, 248
  - OBJECT\_REF\_FLAG\_NONE, 248
  - OBJECT\_REF\_FLAG\_OWNER, 248
  - OBJECT\_REF\_FLAG\_VALID, 248
  - OBJECT\_TYPE\_IPC, 249
  - OBJECT\_TYPE\_PROCESS, 249
  - OBJECT\_TYPE\_THREAD, 249
- object\_header\_t, 36
  - flags, 36
  - ref\_count, 36
  - type, 36
- object\_ref\_t, 37
  - cookie, 37
  - flags, 37
  - object, 37
- offsetof
  - stddef.h, 243
- outb
  - io.h, 174
- outl
  - io.h, 174
- outw
  - io.h, 174
- p\_align
  - Elf32\_Phdr, 20
- p\_filesz
  - Elf32\_Phdr, 20
- p\_flags
  - Elf32\_Phdr, 21
- p\_memsz
  - Elf32\_Phdr, 21
- p\_offset
  - Elf32\_Phdr, 21
- p\_paddr
  - Elf32\_Phdr, 21
- p\_type
  - Elf32\_Phdr, 21

- Elf32\_Phdr, 21
- p\_vaddr
  - Elf32\_Phdr, 21
- PACK\_DESCRIPTOR
  - descriptors.h, 112
- PAGE\_BITS
  - jinue-common/asm/vm.h, 152
- PAGE\_DIRECTORY\_OFFSET\_OF
  - vm\_private.h, 201
- PAGE\_MASK
  - jinue-common/asm/vm.h, 152
- PAGE\_SIZE
  - jinue-common/asm/vm.h, 152
- PAGE\_TABLE\_ENTRIES
  - vm\_private.h, 201
- PAGE\_TABLE\_MASK
  - vm\_private.h, 201
- PAGE\_TABLE\_OFFSET\_OF
  - vm\_private.h, 201
- PDPT\_BITS
  - vm\_pae.c, 343
- PDPT\_ENTRIES
  - vm\_pae.c, 343
- PF\_R
  - jinue-common/elf.h, 93
- PF\_W
  - jinue-common/elf.h, 93
- PF\_X
  - jinue-common/elf.h, 93
- PFNULL
  - hal/types.h, 178
- PIC8259\_CASCADE\_INPUT
  - asm/pic8259.h, 121
- PIC8259\_EOI
  - asm/pic8259.h, 121
- PIC8259\_ICW1\_1
  - asm/pic8259.h, 121
- PIC8259\_ICW1\_IC4
  - asm/pic8259.h, 121
- PIC8259\_ICW1\_LTIM
  - asm/pic8259.h, 121
- PIC8259\_ICW1\_SINGL
  - asm/pic8259.h, 122
- PIC8259\_ICW4\_AEOI
  - asm/pic8259.h, 122
- PIC8259\_ICW4\_UPM
  - asm/pic8259.h, 122
- PIC8259\_IRQ\_COUNT
  - asm/pic8259.h, 122
- PIC8259\_MASTER\_BASE
  - asm/pic8259.h, 122
- PIC8259\_SLAVE\_BASE
  - asm/pic8259.h, 122
- PROCESS\_MAX\_DESCRIPTOR
  - types.h, 181
- PT\_DYNAMIC
  - jinue-common/elf.h, 93
- PT\_INTERP
  - jinue-common/elf.h, 93
- PT\_LOAD
  - jinue-common/elf.h, 93
- PT\_NOTE
  - jinue-common/elf.h, 93
- PT\_NULL
  - jinue-common/elf.h, 94
- PT\_PHDR
  - jinue-common/elf.h, 94
- PT\_SHLIB
  - jinue-common/elf.h, 94
- padding
  - pseudo\_descriptor\_t, 41
- page\_address\_of
  - jinue-common/vm.h, 153
- page\_alloc
  - page\_alloc.c, 371
  - page\_alloc.h, 251
- page\_alloc.c
  - add\_page\_frame, 369
  - clear\_page, 370
  - page\_alloc, 371
  - page\_alloc\_is\_empty, 371
  - page\_free, 371
  - remove\_page\_frame, 372
- page\_alloc.h
  - add\_page\_frame, 250
  - clear\_page, 251
  - page\_alloc, 251
  - page\_alloc\_is\_empty, 252
  - page\_free, 252
  - remove\_page\_frame, 252
- page\_alloc\_is\_empty
  - page\_alloc.c, 371
  - page\_alloc.h, 252
- page\_directory
  - boot\_info\_t, 13
- page\_free
  - page\_alloc.c, 371
  - page\_alloc.h, 252
- page\_number\_of
  - jinue-common/vm.h, 153
- page\_offset\_of
  - jinue-common/vm.h, 153
- page\_table
  - boot\_info\_t, 13
- page\_table\_entries
  - vm.c, 341
  - vm\_private.h, 204
- panic



- panic.c, 373
- panic.h, 254
- panic.c
  - panic, 373
- panic.h
  - panic, 254
- pd
  - addr\_space\_t, 8
  - pdpt\_t, 38
- pdpt
  - addr\_space\_t, 8
- pdpt\_t, 38
  - hal/types.h, 178
  - pd, 38
- pfalloc
  - pfalloc.h, 256
- pfalloc.c
  - global\_pfalloc\_cache, 376
  - init\_pfalloc\_cache, 375
  - pfalloc\_from, 375
  - pffree\_to, 376
- pfalloc.h
  - global\_pfalloc\_cache, 258
  - init\_pfalloc\_cache, 257
  - KERNEL\_PAGE\_STACK\_INIT, 256
  - KERNEL\_PAGE\_STACK\_SIZE, 256
  - pfalloc, 256
  - pfalloc\_from, 257
  - pffree, 256
  - pffree\_to, 257
- pfalloc\_cache\_t, 38
  - count, 39
  - ptr, 39
- pfalloc\_from
  - pfalloc.c, 375
  - pfalloc.h, 257
- pffree
  - pfalloc.h, 256
- pffree\_to
  - pfalloc.c, 376
  - pfalloc.h, 257
- pic8259.c
  - pic8259\_eoi, 315
  - pic8259\_init, 315
  - pic8259\_mask\_irq, 316
  - pic8259\_unmask\_irq, 317
- pic8259.h
  - pic8259\_eoi, 123
  - pic8259\_init, 124
  - pic8259\_mask\_irq, 125
  - pic8259\_unmask\_irq, 125
- pic8259\_eoi
  - pic8259.c, 315
  - pic8259.h, 123
- pic8259\_init
  - pic8259.c, 315
  - pic8259.h, 124
- pic8259\_mask\_irq
  - pic8259.c, 316
  - pic8259.h, 125
- pic8259\_unmask\_irq
  - pic8259.c, 317
  - pic8259.h, 125
- prev
  - slab\_t, 46
  - tss\_t, 54
  - vmalloc\_block\_t, 56
- print\_hex\_b
  - printk.h, 258
- print\_hex\_l
  - printk.h, 258
- print\_hex\_nibble
  - printk.h, 258
- print\_hex\_q
  - printk.h, 258
- print\_hex\_w
  - printk.h, 258
- print\_unsigned\_int
  - printk.h, 259
- printk
  - printk.h, 259
- printk.h
  - print\_hex\_b, 258
  - print\_hex\_l, 258
  - print\_hex\_nibble, 258
  - print\_hex\_q, 258
  - print\_hex\_w, 258
  - print\_unsigned\_int, 259
  - printk, 259
- proc\_size
  - boot\_info\_t, 13
- proc\_start
  - boot\_info\_t, 13
- process
  - thread\_t, 48
- process.c
  - process\_boot\_init, 377
  - process\_create, 378
  - process\_create\_initial, 378
  - process\_get\_descriptor, 379
  - process\_unused\_descriptor, 379
- process.h
  - process\_boot\_init, 259
  - process\_create, 260
  - process\_create\_initial, 261
  - process\_get\_descriptor, 261
  - process\_unused\_descriptor, 262
- process\_boot\_init

- process.c, 377
- process.h, 259
- process\_create
  - process.c, 378
  - process.h, 260
- process\_create\_initial
  - process.c, 378
  - process.h, 261
- process\_get\_descriptor
  - process.c, 379
  - process.h, 261
- process\_t, 39
  - addr\_space, 40
  - descriptors, 40
  - header, 40
- process\_unused\_descriptor
  - process.c, 379
  - process.h, 262
- pseudo\_descriptor\_t, 40
  - addr, 41
  - limit, 41
  - padding, 41
- pte\_t, 41
  - entry, 41
  - hal/types.h, 178
- ptr
  - pfalloc\_cache\_t, 39
- ptrdiff\_t
  - stddef.h, 243
- RPL\_KERNEL
  - asm/descriptors.h, 107
- RPL\_USER
  - asm/descriptors.h, 107
- ramdisk\_size
  - boot\_info\_t, 13
- ramdisk\_start
  - boot\_info\_t, 13
- rdmsr
  - x86.h, 159
- rdtsc
  - x86.h, 159
- recv\_list
  - ipc\_t, 28
- ref\_count
  - object\_header\_t, 36
- remove\_page\_frame
  - page\_alloc.c, 372
  - page\_alloc.h, 252
- return\_from\_interrupt
  - trap.h, 176
- SEG\_DESCRIPTOR
  - descriptors.h, 112
- SEG\_FLAG\_16BIT
  - asm/descriptors.h, 107
- SEG\_FLAG\_16BIT\_GATE
  - asm/descriptors.h, 107
- SEG\_FLAG\_32BIT
  - asm/descriptors.h, 107
- SEG\_FLAG\_32BIT\_GATE
  - asm/descriptors.h, 108
- SEG\_FLAG\_BUSY
  - asm/descriptors.h, 108
- SEG\_FLAG\_IN\_BYTES
  - asm/descriptors.h, 108
- SEG\_FLAG\_IN\_PAGES
  - asm/descriptors.h, 108
- SEG\_FLAG\_KERNEL
  - asm/descriptors.h, 108
- SEG\_FLAG\_NORMAL
  - asm/descriptors.h, 108
- SEG\_FLAG\_NORMAL\_GATE
  - asm/descriptors.h, 108
- SEG\_FLAG\_NOSYSTEM
  - asm/descriptors.h, 109
- SEG\_FLAG\_PRESENT
  - asm/descriptors.h, 109
- SEG\_FLAG\_SYSTEM
  - asm/descriptors.h, 109
- SEG\_FLAG\_TSS
  - asm/descriptors.h, 109
- SEG\_FLAG\_USER
  - asm/descriptors.h, 109
- SEG\_FLAGS\_OFFSET
  - asm/descriptors.h, 109
- SEG\_SELECTOR
  - asm/descriptors.h, 109
- SEG\_TYPE\_CALL\_GATE
  - asm/descriptors.h, 109
- SEG\_TYPE\_CODE
  - asm/descriptors.h, 110
- SEG\_TYPE\_DATA
  - asm/descriptors.h, 110
- SEG\_TYPE\_INTERRUPT\_GATE
  - asm/descriptors.h, 110
- SEG\_TYPE\_READ\_ONLY
  - asm/descriptors.h, 110
- SEG\_TYPE\_TASK\_GATE
  - asm/descriptors.h, 110
- SEG\_TYPE\_TRAP\_GATE
  - asm/descriptors.h, 110
- SEG\_TYPE\_TSS
  - asm/descriptors.h, 110
- SERIAL\_COM1\_IOPORT
  - asm/serial.h, 127
- SERIAL\_COM2\_IOPORT
  - asm/serial.h, 127
- SERIAL\_COM3\_IOPORT

- asm/serial.h, 127
- SERIAL\_COM4\_IOPORT
  - asm/serial.h, 127
- SERIAL\_REG\_DATA\_BUFFER
  - asm/serial.h, 128
- SERIAL\_REG\_DIVISOR\_HIGH
  - asm/serial.h, 128
- SERIAL\_REG\_DIVISOR\_LOW
  - asm/serial.h, 128
- SERIAL\_REG\_FIFO\_CTRL
  - asm/serial.h, 128
- SERIAL\_REG\_INTR\_ENABLE
  - asm/serial.h, 128
- SERIAL\_REG\_INTR\_ID
  - asm/serial.h, 128
- SERIAL\_REG\_LINE\_CTRL
  - asm/serial.h, 128
- SERIAL\_REG\_LINE\_STATUS
  - asm/serial.h, 128
- SERIAL\_REG\_MODEM\_CTRL
  - asm/serial.h, 128
- SERIAL\_REG\_MODEM\_STATUS
  - asm/serial.h, 129
- SERIAL\_REG\_SCRATCH
  - asm/serial.h, 129
- SHT\_DYNAMIC
  - jinue-common/elf.h, 94
- SHT\_DYNSYM
  - jinue-common/elf.h, 94
- SHT\_HASH
  - jinue-common/elf.h, 94
- SHT\_NOBITS
  - jinue-common/elf.h, 94
- SHT\_NOTE
  - jinue-common/elf.h, 94
- SHT\_NULL
  - jinue-common/elf.h, 94
- SHT\_PROGBITS
  - jinue-common/elf.h, 95
- SHT\_REL
  - jinue-common/elf.h, 95
- SHT\_RELA
  - jinue-common/elf.h, 95
- SHT\_SHLIB
  - jinue-common/elf.h, 95
- SHT\_STRTAB
  - jinue-common/elf.h, 95
- SHT\_SYMTAB
  - jinue-common/elf.h, 95
- SLAB\_COMPACT
  - slab.h, 264
- SLAB\_DEFAULT\_WORKING\_SET
  - slab.h, 264
- SLAB\_DEFAULTS
  - slab.h, 264
- SLAB\_HWCACHE\_ALIGN
  - slab.h, 264
- SLAB\_POISON
  - slab.h, 264
- SLAB\_POISON\_ALIVE\_VALUE
  - slab.h, 265
- SLAB\_POISON\_DEAD\_VALUE
  - slab.h, 265
- SLAB\_RED\_ZONE
  - slab.h, 265
- SLAB\_RED\_ZONE\_VALUE
  - slab.h, 265
- SLAB\_SIZE
  - slab.h, 265
- STACK\_BASE
  - jinue-common/asm/vm.h, 152
- STACK\_SIZE
  - jinue-common/asm/vm.h, 152
- STACK\_START
  - jinue-common/asm/vm.h, 152
- STB\_GLOBAL
  - jinue-common/elf.h, 95
- STB\_LOCAL
  - jinue-common/elf.h, 95
- STB\_WEAK
  - jinue-common/elf.h, 96
- STN\_UNDEF
  - jinue-common/elf.h, 96
- STT\_FILE
  - jinue-common/elf.h, 96
- STT\_FUNCTION
  - jinue-common/elf.h, 96
- STT\_NOTYPE
  - jinue-common/elf.h, 96
- STT\_OBJECT
  - jinue-common/elf.h, 96
- STT\_SECTION
  - jinue-common/elf.h, 96
- SYSCALL\_FUNCT\_CONSOLE\_PUTC
  - jinue-common/asm/syscall.h, 229
- SYSCALL\_FUNCT\_CONSOLE\_PUTS
  - jinue-common/asm/syscall.h, 229
- SYSCALL\_FUNCT\_CREATE\_IPC
  - jinue-common/asm/syscall.h, 229
- SYSCALL\_FUNCT\_GET\_PHYS\_MEMORY
  - jinue-common/asm/syscall.h, 230
- SYSCALL\_FUNCT\_GET\_THREAD\_LOCAL\_ADDR
  - jinue-common/asm/syscall.h, 230
- SYSCALL\_FUNCT\_PROC\_BASE
  - jinue-common/asm/syscall.h, 230
- SYSCALL\_FUNCT\_RECEIVE
  - jinue-common/asm/syscall.h, 230
- SYSCALL\_FUNCT\_REPLY

- jinue-common/asm/syscall.h, 230
- SYSCALL\_FUNCT\_SET\_THREAD\_LOCAL\_ADDR
  - jinue-common/asm/syscall.h, 230
- SYSCALL\_FUNCT\_SYSCALL\_METHOD
  - jinue-common/asm/syscall.h, 230
- SYSCALL\_FUNCT\_SYSTEM\_BASE
  - jinue-common/asm/syscall.h, 231
- SYSCALL\_FUNCT\_THREAD\_CREATE
  - jinue-common/asm/syscall.h, 231
- SYSCALL\_FUNCT\_THREAD\_YIELD
  - jinue-common/asm/syscall.h, 231
- SYSCALL\_FUNCT\_USER\_BASE
  - jinue-common/asm/syscall.h, 231
- SYSCALL\_IRQ
  - jinue-common/asm/syscall.h, 231
- SYSCALL\_METHOD\_FAST\_AMD
  - jinue-common/asm/syscall.h, 231
- SYSCALL\_METHOD\_FAST\_INTEL
  - jinue-common/asm/syscall.h, 231
- SYSCALL\_METHOD\_INTR
  - jinue-common/asm/syscall.h, 232
- saved\_stack\_pointer
  - thread\_context\_t, 47
- seg\_descriptor\_t
  - hal/types.h, 178
- seg\_selector\_t
  - hal/types.h, 179
- self
  - cpu\_data\_t, 15
- send\_list
  - ipc\_t, 28
- sender
  - thread\_t, 48
- serial.c
  - serial\_init, 318
  - serial\_printn, 319
  - serial\_putc, 319
- serial.h
  - serial\_init, 130
  - serial\_printn, 130
  - serial\_putc, 131
- serial\_init
  - serial.c, 318
  - serial.h, 130
- serial\_printn
  - serial.c, 319
  - serial.h, 130
- serial\_putc
  - serial.c, 319
  - serial.h, 131
- set\_cr0
  - x86.h, 159
- set\_cr3
  - x86.h, 159
- set\_cr4
  - x86.h, 159
- set\_cs
  - x86.h, 159
- set\_ds
  - x86.h, 159
- set\_eflags
  - x86.h, 159
- set\_es
  - x86.h, 159
- set\_fs
  - x86.h, 159
- set\_gs
  - x86.h, 159
- set\_ss
  - x86.h, 159
- setup\_signature
  - boot\_info\_t, 14
- sh\_addr
  - Elf32\_Shdr, 22
- sh\_addralign
  - Elf32\_Shdr, 22
- sh\_entsize
  - Elf32\_Shdr, 22
- sh\_flags
  - Elf32\_Shdr, 22
- sh\_info
  - Elf32\_Shdr, 22
- sh\_link
  - Elf32\_Shdr, 22
- sh\_name
  - Elf32\_Shdr, 22
- sh\_offset
  - Elf32\_Shdr, 22
- sh\_size
  - Elf32\_Shdr, 23
- sh\_type
  - Elf32\_Shdr, 23
- size
  - e820\_t, 17
  - jinue\_mem\_entry\_t, 29
- size\_t
  - stddef.h, 243
- slab.c
  - slab\_cache\_alloc, 381
  - slab\_cache\_free, 383
  - slab\_cache\_init, 385
  - slab\_cache\_reap, 387
  - slab\_cache\_set\_working\_set, 387
- slab.h
  - SLAB\_COMPACT, 264
  - SLAB\_DEFAULT\_WORKING\_SET, 264
  - SLAB\_DEFAULTS, 264
  - SLAB\_HWCACHE\_ALIGN, 264

- SLAB\_POISON, 264
- SLAB\_POISON\_ALIVE\_VALUE, 265
- SLAB\_POISON\_DEAD\_VALUE, 265
- SLAB\_RED\_ZONE, 265
- SLAB\_RED\_ZONE\_VALUE, 265
- SLAB\_SIZE, 265
- slab\_bufctl\_t, 265
- slab\_cache\_alloc, 266
- slab\_cache\_free, 268
- slab\_cache\_init, 269
- slab\_cache\_list, 273
- slab\_cache\_reap, 272
- slab\_cache\_set\_working\_set, 272
- slab\_cache\_t, 265
- slab\_ctor\_t, 265
- slab\_t, 265
- slab\_bufctl\_t, 42
  - next, 42
  - slab.h, 265
- slab\_cache\_alloc
  - slab.c, 381
  - slab.h, 266
- slab\_cache\_free
  - slab.c, 383
  - slab.h, 268
- slab\_cache\_init
  - slab.c, 385
  - slab.h, 269
- slab\_cache\_list
  - slab.h, 273
- slab\_cache\_reap
  - slab.c, 387
  - slab.h, 272
- slab\_cache\_set\_working\_set
  - slab.c, 387
  - slab.h, 272
- slab\_cache\_t, 42
  - alignment, 43
  - alloc\_size, 43
  - bufctl\_offset, 43
  - ctor, 43
  - dtor, 43
  - empty\_count, 44
  - flags, 44
  - max\_colour, 44
  - name, 44
  - next\_colour, 44
  - obj\_size, 44
  - slab.h, 265
  - slabs\_empty, 44
  - slabs\_full, 44
  - slabs\_partial, 44
  - working\_set, 45
- slab\_ctor\_t
  - slab.h, 265
- slab\_t, 45
  - cache, 46
  - colour, 46
  - free\_list, 46
  - next, 46
  - obj\_count, 46
  - prev, 46
  - slab.h, 265
- slabs\_empty
  - slab\_cache\_t, 44
- slabs\_full
  - slab\_cache\_t, 44
- slabs\_partial
  - slab\_cache\_t, 44
- ss
  - trapframe\_t, 51
  - tss\_t, 54
- ss0
  - tss\_t, 55
- ss1
  - tss\_t, 55
- ss2
  - tss\_t, 55
- st\_info
  - Elf32\_Sym, 23
- st\_name
  - Elf32\_Sym, 23
- st\_other
  - Elf32\_Sym, 24
- st\_shndx
  - Elf32\_Sym, 24
- st\_size
  - Elf32\_Sym, 24
- st\_value
  - Elf32\_Sym, 24
- stack\_addr
  - elf\_info\_t, 26
- stack\_base
  - vmalloc\_block\_t, 56
- stack\_ptr
  - vmalloc\_block\_t, 56
- start\_addr
  - vmalloc\_t, 58
- startup.h
  - halt, 175
- stdarg.h
  - va\_arg, 241
  - va\_copy, 241
  - va\_end, 241
  - va\_list, 241
  - va\_start, 241
- stdbool.h
  - \_\_bool\_true\_false\_are\_defined, 242

- bool, 242
- false, 242
- true, 242
- stddef.h
  - NULL, 243
  - offsetof, 243
  - ptrdiff\_t, 243
  - size\_t, 243
  - wchar\_t, 243
- stdint.h
  - INT64\_C, 244
  - int16\_t, 244
  - int32\_t, 244
  - int64\_t, 244
  - int8\_t, 244
  - intptr\_t, 244
  - UINT64\_C, 244
  - uint16\_t, 244
  - uint32\_t, 245
  - uint64\_t, 245
  - uint8\_t, 245
  - uintptr\_t, 245
- stdlib.h
  - EXIT\_FAILURE, 245
  - EXIT\_SUCCESS, 245
- stepping
  - cpu\_info\_t, 16
- sti
  - x86.h, 159
- string.h
  - memcpy, 246
  - memset, 246
  - strlen, 247
- strlen
  - c-string.c, 291
  - string.h, 247
- syscall.c
  - dispatch\_syscall, 388
- syscall.h
  - dispatch\_syscall, 233
- syscall\_method
  - hal.c, 309
  - trap.h, 176
- THREAD\_CONTEXT\_MASK
  - hal/asm/thread.h, 132
- THREAD\_CONTEXT\_SIZE
  - hal/asm/thread.h, 132
- TSS\_LIMIT
  - asm/descriptors.h, 110
- tail
  - jinue\_list\_t, 29
- thread.c
  - thread\_create, 323
  - thread\_create\_boot, 324
  - thread\_destroy, 324
  - thread\_ready, 325
  - thread\_switch, 325
  - thread\_yield\_from, 326
- thread.h
  - thread\_create, 136
  - thread\_create\_boot, 136
  - thread\_destroy, 137
  - thread\_ready, 137
  - thread\_switch, 137
  - thread\_yield\_from, 138
- thread\_context\_switch
  - hal/thread.c, 321
  - hal/thread.h, 133
- thread\_context\_switch\_stack
  - hal/thread.c, 321
- thread\_context\_t, 46
  - local\_storage\_addr, 47
  - local\_storage\_size, 47
  - saved\_stack\_pointer, 47
- thread\_create
  - thread.c, 323
  - thread.h, 136
- thread\_create\_boot
  - thread.c, 324
  - thread.h, 136
- thread\_ctx
  - thread\_t, 48
- thread\_destroy
  - thread.c, 324
  - thread.h, 137
- thread\_list
  - thread\_t, 49
- thread\_page\_init
  - hal/thread.c, 321
  - hal/thread.h, 134
- thread\_ready
  - thread.c, 325
  - thread.h, 137
- thread\_switch
  - thread.c, 325
  - thread.h, 137
- thread\_t, 47
  - header, 48
  - message\_args, 48
  - message\_buffer, 48
  - message\_info, 48
  - process, 48
  - sender, 48
  - thread\_ctx, 48
  - thread\_list, 49
  - types.h, 182
- thread\_yield\_from

- thread.c, 326
- thread.h, 138
- top\_level
  - addr\_space\_t, 8
- total\_size
  - message\_info\_t, 35
- trap.h
  - fast\_amd\_entry, 176
  - fast\_intel\_entry, 176
  - return\_from\_interrupt, 176
  - syscall\_method, 176
- trapframe\_t, 49
  - cs, 50
  - ds, 50
  - eax, 50
  - ebp, 50
  - ebx, 50
  - ecx, 50
  - edi, 50
  - edx, 50
  - eflags, 50
  - eip, 50
  - errcode, 50
  - es, 51
  - esi, 51
  - esp, 51
  - fs, 51
  - gs, 51
  - ivt, 51
  - ss, 51
- true
  - stdbool.h, 242
- tss
  - cpu\_data\_t, 15
- tss\_t, 52
  - cr3, 52
  - cs, 52
  - debug, 52
  - ds, 53
  - eax, 53
  - ebp, 53
  - ebx, 53
  - ecx, 53
  - edi, 53
  - edx, 53
  - eflags, 53
  - eip, 53
  - es, 53
  - esi, 53
  - esp, 54
  - esp0, 54
  - esp1, 54
  - esp2, 54
  - fs, 54
  - gs, 54
  - iomap, 54
  - ldt, 54
  - prev, 54
  - ss, 54
  - ss0, 55
  - ss1, 55
  - ss2, 55
- type
  - e820\_t, 17
  - jinue\_mem\_entry\_t, 29
  - object\_header\_t, 36
- types.h
  - PROCESS\_MAX\_DESCRIPTOR, 181
  - thread\_t, 182
- UINT64\_C
  - stdint.h, 244
- uint16\_t
  - stdint.h, 244
- uint32\_t
  - stdint.h, 245
- uint64\_t
  - stdint.h, 245
- uint8\_t
  - stdint.h, 245
- uintptr\_t
  - stdint.h, 245
- user\_paddr\_t
  - hal/types.h, 179
- util.h
  - ALIGN\_END, 273
  - ALIGN\_END\_PTR, 273
  - ALIGN\_START, 274
  - ALIGN\_START\_PTR, 274
  - alloc\_backward, 274
  - alloc\_forward, 274
  - OFFSET\_OF\_PTR, 274
- VGA\_COL
  - vga.h, 183
- VGA\_COLOR\_BLACK
  - vga.h, 183
- VGA\_COLOR\_BLUE
  - vga.h, 183
- VGA\_COLOR\_BRIGHTBLUE
  - vga.h, 183
- VGA\_COLOR\_BRIGHTCYAN
  - vga.h, 183
- VGA\_COLOR\_BRIGHTGREEN
  - vga.h, 183
- VGA\_COLOR\_BRIGHTMAGENTA
  - vga.h, 184
- VGA\_COLOR\_BRIGHTRED
  - vga.h, 184

- VGA\_COLOR\_BRIGHTWHITE
  - vga.h, 184
- VGA\_COLOR\_BROWN
  - vga.h, 184
- VGA\_COLOR\_CYAN
  - vga.h, 184
- VGA\_COLOR\_ERASE
  - vga.h, 184
- VGA\_COLOR\_GRAY
  - vga.h, 184
- VGA\_COLOR\_GREEN
  - vga.h, 184
- VGA\_COLOR\_MAGENTA
  - vga.h, 184
- VGA\_COLOR\_RED
  - vga.h, 184
- VGA\_COLOR\_WHITE
  - vga.h, 184
- VGA\_COLOR\_YELLOW
  - vga.h, 185
- VGA\_CRTC\_ADDR
  - vga.h, 185
- VGA\_CRTC\_DATA
  - vga.h, 185
- VGA\_FB\_FLAG\_ACTIVE
  - vga.h, 185
- VGA\_LINE
  - vga.h, 185
- VGA\_LINES
  - vga.h, 185
- VGA\_MISC\_OUT\_RD
  - vga.h, 185
- VGA\_MISC\_OUT\_WR
  - vga.h, 185
- VGA\_TAB\_WIDTH
  - vga.h, 185
- VGA\_TEXT\_VID\_BASE
  - vga.h, 185
- VGA\_TEXT\_VID\_SIZE
  - vga.h, 186
- VGA\_TEXT\_VID\_TOP
  - vga.h, 186
- VGA\_WIDTH
  - vga.h, 186
- VM\_FLAG\_ACCESSED
  - hal/asm/vm.h, 140
- VM\_FLAG\_DIRTY
  - hal/asm/vm.h, 140
- VM\_FLAG\_KERNEL
  - hal/asm/vm.h, 140
- VM\_FLAG\_PRESENT
  - hal/asm/vm.h, 140
- VM\_FLAG\_READ\_ONLY
  - hal/asm/vm.h, 140
- VM\_FLAG\_READ\_WRITE
  - hal/asm/vm.h, 140
- VM\_FLAG\_USER
  - hal/asm/vm.h, 141
- VMALLOC\_BLOCK\_SIZE
  - vmalloc.c, 393
- VMALLOC\_STACK\_ENTRIES
  - vmalloc.c, 393
- va\_arg
  - stdarg.h, 241
- va\_copy
  - stdarg.h, 241
- va\_end
  - stdarg.h, 241
- va\_list
  - stdarg.h, 241
- va\_start
  - stdarg.h, 241
- vendor
  - cpu\_info\_t, 16
- vga.c
  - vga\_clear, 327
  - vga\_get\_cursor\_pos, 327
  - vga\_init, 328
  - vga\_print, 328
  - vga\_printn, 329
  - vga\_putc, 329
  - vga\_scroll, 330
  - vga\_set\_base\_addr, 330
  - vga\_set\_cursor\_pos, 330
- vga.h
  - VGA\_COL, 183
  - VGA\_COLOR\_BLACK, 183
  - VGA\_COLOR\_BLUE, 183
  - VGA\_COLOR\_BRIGHTBLUE, 183
  - VGA\_COLOR\_BRIGHTCYAN, 183
  - VGA\_COLOR\_BRIGHTGREEN, 183
  - VGA\_COLOR\_BRIGHTMAGENTA, 184
  - VGA\_COLOR\_BRIGHTRED, 184
  - VGA\_COLOR\_BRIGHTWHITE, 184
  - VGA\_COLOR\_BROWN, 184
  - VGA\_COLOR\_CYAN, 184
  - VGA\_COLOR\_ERASE, 184
  - VGA\_COLOR\_GRAY, 184
  - VGA\_COLOR\_GREEN, 184
  - VGA\_COLOR\_MAGENTA, 184
  - VGA\_COLOR\_RED, 184
  - VGA\_COLOR\_WHITE, 184
  - VGA\_COLOR\_YELLOW, 185
  - VGA\_CRTC\_ADDR, 185
  - VGA\_CRTC\_DATA, 185
  - VGA\_FB\_FLAG\_ACTIVE, 185
  - VGA\_LINE, 185
  - VGA\_LINES, 185



- VGA\_MISC\_OUT\_RD, 185
- VGA\_MISC\_OUT\_WR, 185
- VGA\_TAB\_WIDTH, 185
- VGA\_TEXT\_VID\_BASE, 185
- VGA\_TEXT\_VID\_SIZE, 186
- VGA\_TEXT\_VID\_TOP, 186
- VGA\_WIDTH, 186
- vga\_clear, 186
- vga\_get\_cursor\_pos, 186
- vga\_init, 187
- vga\_pos\_t, 186
- vga\_print, 187
- vga\_printn, 188
- vga\_putc, 188
- vga\_scroll, 189
- vga\_set\_base\_addr, 189
- vga\_set\_cursor\_pos, 189
- vga\_clear
  - vga.c, 327
  - vga.h, 186
- vga\_get\_cursor\_pos
  - vga.c, 327
  - vga.h, 186
- vga\_init
  - vga.c, 328
  - vga.h, 187
- vga\_pos\_t
  - vga.h, 186
- vga\_print
  - vga.c, 328
  - vga.h, 187
- vga\_printn
  - vga.c, 329
  - vga.h, 188
- vga\_putc
  - vga.c, 329
  - vga.h, 188
- vga\_scroll
  - vga.c, 330
  - vga.h, 189
- vga\_set\_base\_addr
  - vga.c, 330
  - vga.h, 189
- vga\_set\_cursor\_pos
  - vga.c, 330
  - vga.h, 189
- vm.c
  - global\_page\_tables, 341
  - initial\_addr\_space, 341
  - page\_table\_entries, 341
  - vm\_boot\_init, 332
  - vm\_boot\_postinit, 334
  - vm\_change\_flags, 334
  - vm\_clone\_page\_directory, 335
  - vm\_create\_addr\_space, 336
  - vm\_create\_initial\_addr\_space, 336
  - vm\_destroy\_addr\_space, 337
  - vm\_destroy\_page\_directory, 338
  - vm\_init\_initial\_page\_directory, 338
  - vm\_lookup\_kernel\_paddr, 339
  - vm\_map\_early, 340
  - vm\_map\_kernel, 340
  - vm\_map\_user, 340
  - vm\_switch\_addr\_space, 340
  - vm\_unmap\_kernel, 341
  - vm\_unmap\_user, 341
- vm\_boot\_init
  - hal/vm.h, 143
  - vm.c, 332
- vm\_boot\_postinit
  - hal/vm.h, 144
  - vm.c, 334
- vm\_change\_flags
  - hal/vm.h, 145
  - vm.c, 334
- vm\_clone\_page\_directory
  - vm.c, 335
  - vm\_private.h, 201
- vm\_create\_addr\_space
  - hal/vm.h, 145
  - vm.c, 336
- vm\_create\_initial\_addr\_space
  - hal/vm.h, 146
  - vm.c, 336
- vm\_destroy\_addr\_space
  - hal/vm.h, 147
  - vm.c, 337
- vm\_destroy\_page\_directory
  - vm.c, 338
  - vm\_private.h, 202
- vm\_init\_initial\_page\_directory
  - vm.c, 338
  - vm\_private.h, 203
- vm\_lookup\_kernel\_paddr
  - hal/vm.h, 147
  - vm.c, 339
- vm\_map\_early
  - hal/vm.h, 147
  - vm.c, 340
- vm\_map\_kernel
  - hal/vm.h, 148
  - vm.c, 340
- vm\_map\_user
  - hal/vm.h, 148
  - vm.c, 340
- vm\_pae.c
  - initial\_pdpt, 351
  - PDPT\_BITS, 343

- PDPT\_ENTRIES, 343
- vm\_pae\_boot\_init, 343
- vm\_pae\_clear\_pte, 344
- vm\_pae\_copy\_pte, 344
- vm\_pae\_create\_addr\_space, 344
- vm\_pae\_create\_initial\_addr\_space, 345
- vm\_pae\_create\_pdpt\_cache, 346
- vm\_pae\_destroy\_addr\_space, 347
- vm\_pae\_get\_pte\_flags, 348
- vm\_pae\_get\_pte\_paddr, 348
- vm\_pae\_get\_pte\_with\_offset, 348
- vm\_pae\_lookup\_page\_directory, 348
- vm\_pae\_page\_directory\_offset\_of, 350
- vm\_pae\_page\_table\_offset\_of, 350
- vm\_pae\_set\_pte, 350
- vm\_pae\_set\_pte\_flags, 351
- vm\_pae\_unmap\_low\_alias, 351
- vm\_pae.h
  - vm\_pae\_boot\_init, 191
  - vm\_pae\_clear\_pte, 192
  - vm\_pae\_copy\_pte, 192
  - vm\_pae\_create\_addr\_space, 192
  - vm\_pae\_create\_initial\_addr\_space, 193
  - vm\_pae\_create\_pdpt\_cache, 194
  - vm\_pae\_destroy\_addr\_space, 195
  - vm\_pae\_get\_pte\_flags, 196
  - vm\_pae\_get\_pte\_paddr, 196
  - vm\_pae\_get\_pte\_with\_offset, 196
  - vm\_pae\_lookup\_page\_directory, 196
  - vm\_pae\_page\_directory\_offset\_of, 198
  - vm\_pae\_page\_table\_offset\_of, 198
  - vm\_pae\_set\_pte, 198
  - vm\_pae\_set\_pte\_flags, 199
  - vm\_pae\_unmap\_low\_alias, 199
- vm\_pae\_boot\_init
  - vm\_pae.c, 343
  - vm\_pae.h, 191
- vm\_pae\_clear\_pte
  - vm\_pae.c, 344
  - vm\_pae.h, 192
- vm\_pae\_copy\_pte
  - vm\_pae.c, 344
  - vm\_pae.h, 192
- vm\_pae\_create\_addr\_space
  - vm\_pae.c, 344
  - vm\_pae.h, 192
- vm\_pae\_create\_initial\_addr\_space
  - vm\_pae.c, 345
  - vm\_pae.h, 193
- vm\_pae\_create\_pdpt\_cache
  - vm\_pae.c, 346
  - vm\_pae.h, 194
- vm\_pae\_destroy\_addr\_space
  - vm\_pae.c, 347
  - vm\_pae.h, 195
- vm\_pae\_get\_pte\_flags
  - vm\_pae.c, 348
  - vm\_pae.h, 196
- vm\_pae\_get\_pte\_paddr
  - vm\_pae.c, 348
  - vm\_pae.h, 196
- vm\_pae\_get\_pte\_with\_offset
  - vm\_pae.c, 348
  - vm\_pae.h, 196
- vm\_pae\_lookup\_page\_directory
  - vm\_pae.c, 348
  - vm\_pae.h, 196
- vm\_pae\_page\_directory\_offset\_of
  - vm\_pae.c, 350
  - vm\_pae.h, 198
- vm\_pae\_page\_table\_offset\_of
  - vm\_pae.c, 350
  - vm\_pae.h, 198
- vm\_pae\_set\_pte
  - vm\_pae.c, 350
  - vm\_pae.h, 198
- vm\_pae\_set\_pte\_flags
  - vm\_pae.c, 351
  - vm\_pae.h, 199
- vm\_pae\_unmap\_low\_alias
  - vm\_pae.c, 351
  - vm\_pae.h, 199
- vm\_private.h
  - global\_page\_tables, 204
  - PAGE\_DIRECTORY\_OFFSET\_OF, 201
  - PAGE\_TABLE\_ENTRIES, 201
  - PAGE\_TABLE\_MASK, 201
  - PAGE\_TABLE\_OFFSET\_OF, 201
  - page\_table\_entries, 204
  - vm\_clone\_page\_directory, 201
  - vm\_destroy\_page\_directory, 202
  - vm\_init\_initial\_page\_directory, 203
- vm\_switch\_addr\_space
  - hal/vm.h, 148
  - vm.c, 340
- vm\_unmap\_kernel
  - hal/vm.h, 149
  - vm.c, 341
- vm\_unmap\_user
  - hal/vm.h, 149
  - vm.c, 341
- vm\_x86.c
  - vm\_x86\_boot\_init, 352
  - vm\_x86\_clear\_pte, 353
  - vm\_x86\_copy\_pte, 353
  - vm\_x86\_create\_addr\_space, 353
  - vm\_x86\_create\_initial\_addr\_space, 354
  - vm\_x86\_destroy\_addr\_space, 354

- vm\_x86\_get\_pte\_flags, 355
- vm\_x86\_get\_pte\_paddr, 355
- vm\_x86\_get\_pte\_with\_offset, 355
- vm\_x86\_lookup\_page\_directory, 355
- vm\_x86\_page\_directory\_offset\_of, 356
- vm\_x86\_page\_table\_offset\_of, 356
- vm\_x86\_set\_pte, 356
- vm\_x86\_set\_pte\_flags, 357
- vm\_x86.h
  - vm\_x86\_boot\_init, 205
  - vm\_x86\_clear\_pte, 205
  - vm\_x86\_copy\_pte, 206
  - vm\_x86\_create\_addr\_space, 206
  - vm\_x86\_create\_initial\_addr\_space, 206
  - vm\_x86\_destroy\_addr\_space, 207
  - vm\_x86\_get\_pte\_flags, 208
  - vm\_x86\_get\_pte\_paddr, 208
  - vm\_x86\_get\_pte\_with\_offset, 208
  - vm\_x86\_lookup\_page\_directory, 208
  - vm\_x86\_page\_directory\_offset\_of, 209
  - vm\_x86\_page\_table\_offset\_of, 209
  - vm\_x86\_set\_pte, 209
  - vm\_x86\_set\_pte\_flags, 210
- vm\_x86\_boot\_init
  - vm\_x86.c, 352
  - vm\_x86.h, 205
- vm\_x86\_clear\_pte
  - vm\_x86.c, 353
  - vm\_x86.h, 205
- vm\_x86\_copy\_pte
  - vm\_x86.c, 353
  - vm\_x86.h, 206
- vm\_x86\_create\_addr\_space
  - vm\_x86.c, 353
  - vm\_x86.h, 206
- vm\_x86\_create\_initial\_addr\_space
  - vm\_x86.c, 354
  - vm\_x86.h, 206
- vm\_x86\_destroy\_addr\_space
  - vm\_x86.c, 354
  - vm\_x86.h, 207
- vm\_x86\_get\_pte\_flags
  - vm\_x86.c, 355
  - vm\_x86.h, 208
- vm\_x86\_get\_pte\_paddr
  - vm\_x86.c, 355
  - vm\_x86.h, 208
- vm\_x86\_get\_pte\_with\_offset
  - vm\_x86.c, 355
  - vm\_x86.h, 208
- vm\_x86\_lookup\_page\_directory
  - vm\_x86.c, 355
  - vm\_x86.h, 208
- vm\_x86\_page\_directory\_offset\_of
  - vm\_x86.c, 356
  - vm\_x86.h, 209
- vm\_x86\_page\_table\_offset\_of
  - vm\_x86.c, 356
  - vm\_x86.h, 209
- vm\_x86\_set\_pte
  - vm\_x86.c, 356
  - vm\_x86.h, 209
- vm\_x86\_set\_pte\_flags
  - vm\_x86.c, 357
  - vm\_x86.h, 210
- vmalloc
  - vmalloc.c, 393
  - vmalloc.h, 275
- vmalloc.c
  - VMALLOC\_BLOCK\_SIZE, 393
  - VMALLOC\_STACK\_ENTRIES, 393
  - vmalloc, 393
  - vmalloc\_block\_t, 393
  - vmalloc\_init, 394
  - vmalloc\_is\_in\_range, 394
  - vmfree, 394
- vmalloc.h
  - vmalloc, 275
  - vmalloc\_init, 275
  - vmalloc\_is\_in\_range, 276
  - vmfree, 276
- vmalloc\_block\_t, 55
  - next, 56
  - prev, 56
  - stack\_base, 56
  - stack\_ptr, 56
  - vmalloc.c, 393
- vmalloc\_init
  - vmalloc.c, 394
  - vmalloc.h, 275
- vmalloc\_is\_in\_range
  - vmalloc.c, 394
  - vmalloc.h, 276
- vmalloc\_t, 56
  - base\_addr, 57
  - block\_array, 57
  - block\_count, 57
  - end\_addr, 57
  - free\_list, 57
  - init\_limit, 57
  - start\_addr, 58
- vmfree
  - vmalloc.c, 394
  - vmalloc.h, 276
- wchar\_t
  - stddef.h, 243
- working\_set

- slab\_cache\_t, 45
- wrmsr
  - x86.h, 160
- x86.h
  - cli, 158
  - cpuid, 158
  - enable\_pae, 158
  - get\_cr0, 158
  - get\_cr2, 158
  - get\_cr3, 158
  - get\_cr4, 159
  - get\_eflags, 159
  - get\_esp, 159
  - get\_gs\_ptr, 159
  - invalidate\_tlb, 159
  - lgdt, 159
  - lidt, 159
  - ltr, 159
  - msr\_addr\_t, 158
  - rdmsr, 159
  - rdtsc, 159
  - set\_cr0, 159
  - set\_cr3, 159
  - set\_cr4, 159
  - set\_cs, 159
  - set\_ds, 159
  - set\_eflags, 159
  - set\_es, 159
  - set\_fs, 159
  - set\_gs, 159
  - set\_ss, 159
  - sti, 159
  - wrmsr, 160
- X86\_CR0\_PG
  - asm/x86.h, 155
- X86\_CR0\_WP
  - asm/x86.h, 155
- X86\_CR4\_PAE
  - asm/x86.h, 155
- X86\_CR4\_PGE
  - asm/x86.h, 155
- X86\_CR4\_PSE
  - asm/x86.h, 155
- X86\_PDE\_PAGE\_SIZE
  - asm/x86.h, 155
- X86\_PTE\_ACCESSED
  - asm/x86.h, 156
- X86\_PTE\_CACHE\_DISABLE
  - asm/x86.h, 156
- X86\_PTE\_DIRTY
  - asm/x86.h, 156
- X86\_PTE\_GLOBAL
  - asm/x86.h, 156
- X86\_PTE\_NX
  - asm/x86.h, 156
- X86\_PTE\_PRESENT
  - asm/x86.h, 156
- X86\_PTE\_READ\_WRITE
  - asm/x86.h, 156
- X86\_PTE\_USER
  - asm/x86.h, 156
- X86\_PTE\_WRITE\_THROUGH
  - asm/x86.h, 156
- x86\_cpuid\_regs\_t, 58
  - eax, 58
  - ebx, 58
  - ecx, 58
  - edx, 58