

Jinue

Generated by Doxygen 1.5.5

Sun Mar 22 14:55:17 2009

Contents

1	Data Structure Index	1
1.1	Data Structures	1
2	File Index	3
2.1	File List	3
3	Data Structure Documentation	5
3.1	boot_t Struct Reference	5
3.2	e820_t Struct Reference	7
3.3	slab_cache_t Struct Reference	8
3.4	slab_header_t Struct Reference	10
3.5	vm_alloc_t Struct Reference	12
3.6	vm_link_t Struct Reference	14
4	File Documentation	15
4.1	/data/home/phil/svn/jinue/include/alloc.h File Reference	15
4.2	/data/home/phil/svn/jinue/include/ascii.h File Reference	19
4.3	/data/home/phil/svn/jinue/include/assert.h File Reference	20
4.4	/data/home/phil/svn/jinue/include/boot.h File Reference	22
4.5	/data/home/phil/svn/jinue/include/io.h File Reference	27
4.6	/data/home/phil/svn/jinue/include/jinue/vm.h File Reference . .	28
4.7	/data/home/phil/svn/jinue/include/vm.h File Reference	31
4.8	/data/home/phil/svn/jinue/include/kernel.h File Reference	41
4.9	/data/home/phil/svn/jinue/include/panic.h File Reference	45

4.10	/data/home/phil/svn/jinue/include/printk.h File Reference . . .	46
4.11	/data/home/phil/svn/jinue/include/slab.h File Reference	52
4.12	/data/home/phil/svn/jinue/include/startup.h File Reference . .	55
4.13	/data/home/phil/svn/jinue/include/stdarg.h File Reference . . .	56
4.14	/data/home/phil/svn/jinue/include/stdbool.h File Reference . .	58
4.15	/data/home/phil/svn/jinue/include/stddef.h File Reference . . .	59
4.16	/data/home/phil/svn/jinue/include/vga.h File Reference	61
4.17	/data/home/phil/svn/jinue/include/vm_alloc.h File Reference .	70
4.18	/data/home/phil/svn/jinue/kernel/alloc.c File Reference	75
4.19	/data/home/phil/svn/jinue/kernel/assert.c File Reference	79
4.20	/data/home/phil/svn/jinue/kernel/boot.c File Reference	81
4.21	/data/home/phil/svn/jinue/kernel/kernel.c File Reference	85
4.22	/data/home/phil/svn/jinue/kernel/panic.c File Reference	89
4.23	/data/home/phil/svn/jinue/kernel/printk.c File Reference	90
4.24	/data/home/phil/svn/jinue/kernel/slab.c File Reference	96
4.25	/data/home/phil/svn/jinue/kernel/vga.c File Reference	98
4.26	/data/home/phil/svn/jinue/kernel/vm.c File Reference	103
4.27	/data/home/phil/svn/jinue/kernel/vm_alloc.c File Reference . .	106

Chapter 1

Data Structure Index

1.1 Data Structures

Here are the data structures with brief descriptions:

boot_t	5
e820_t	7
slab_cache_t	8
slab_header_t	10
vm_alloc_t	12
vm_link_t	14

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

/data/home/phil/svn/jinue/include/ alloc.h	15
/data/home/phil/svn/jinue/include/ ascii.h	19
/data/home/phil/svn/jinue/include/ assert.h	20
/data/home/phil/svn/jinue/include/ boot.h	22
/data/home/phil/svn/jinue/include/ io.h	27
/data/home/phil/svn/jinue/include/ kernel.h	41
/data/home/phil/svn/jinue/include/ panic.h	45
/data/home/phil/svn/jinue/include/ printk.h	46
/data/home/phil/svn/jinue/include/ slab.h	52
/data/home/phil/svn/jinue/include/ startup.h	55
/data/home/phil/svn/jinue/include/ stdarg.h	56
/data/home/phil/svn/jinue/include/ stdbool.h	58
/data/home/phil/svn/jinue/include/ stddef.h	59
/data/home/phil/svn/jinue/include/ vga.h	61
/data/home/phil/svn/jinue/include/ vm.h	31
/data/home/phil/svn/jinue/include/ vm_alloc.h	70
/data/home/phil/svn/jinue/include/jinue/ vm.h	28
/data/home/phil/svn/jinue/kernel/ alloc.c	75
/data/home/phil/svn/jinue/kernel/ assert.c	79
/data/home/phil/svn/jinue/kernel/ boot.c	81
/data/home/phil/svn/jinue/kernel/ kernel.c	85
/data/home/phil/svn/jinue/kernel/ panic.c	89
/data/home/phil/svn/jinue/kernel/ printk.c	90
/data/home/phil/svn/jinue/kernel/ slab.c	96
/data/home/phil/svn/jinue/kernel/ vga.c	98

/data/home/phil/svn/jinue/kernel/ vm.c	103
/data/home/phil/svn/jinue/kernel/ vm_alloc.c	106

Chapter 3

Data Structure Documentation

3.1 `boot_t` Struct Reference

```
#include <boot.h>
```

3.1.1 Detailed Description

Definition at line 26 of file `boot.h`.

Data Fields

- unsigned long **magic**
- unsigned char **setup_sects**
- unsigned short **root_flags**
- unsigned long **sysize**
- unsigned short **ram_size**
- unsigned short **vid_mode**
- unsigned short **root_dev**
- unsigned short **signature**

3.1.2 Field Documentation

3.1.2.1 unsigned long `boot_t::magic`

Definition at line 27 of file `boot.h`.

Referenced by `get__boot__data()`.

3.1.2.2 unsigned char boot__t::setup_sects

Definition at line 28 of file `boot.h`.

3.1.2.3 unsigned short boot__t::root_flags

Definition at line 29 of file `boot.h`.

3.1.2.4 unsigned long boot__t::sysize

Definition at line 30 of file `boot.h`.

Referenced by `kinit()`.

3.1.2.5 unsigned short boot__t::ram_size

Definition at line 31 of file `boot.h`.

3.1.2.6 unsigned short boot__t::vid_mode

Definition at line 32 of file `boot.h`.

3.1.2.7 unsigned short boot__t::root_dev

Definition at line 33 of file `boot.h`.

3.1.2.8 unsigned short boot__t::signature

Definition at line 34 of file `boot.h`.

Referenced by `get__boot__data()`.

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

- `/data/home/phil/svn/jinue/include/boot.h`

3.2 e820__t Struct Reference

```
#include <boot.h>
```

3.2.1 Detailed Description

Definition at line 19 of file boot.h.

Data Fields

- **e820__addr__t addr**
- **e820__size__t size**
- **e820__type__t type**

3.2.2 Field Documentation

3.2.2.1 e820__addr__t e820__t::addr

Definition at line 20 of file boot.h.

3.2.2.2 e820__size__t e820__t::size

Definition at line 21 of file boot.h.

Referenced by e820__get__size().

3.2.2.3 e820__type__t e820__t::type

Definition at line 22 of file boot.h.

Referenced by e820__get__type().

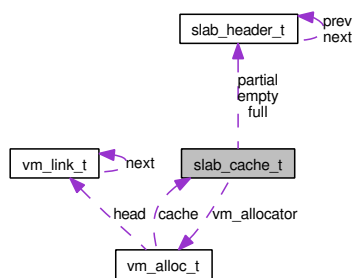
The documentation for this struct was generated from the following file:

- /data/home/phil/svn/jinue/include/**boot.h**

3.3 slab_cache_t Struct Reference

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

Collaboration diagram for slab_cache_t:



3.3.1 Detailed Description

Definition at line 15 of file slab.h.

Data Fields

- `size_t obj_size`
- `count_t per_slab`
- `slab_header_t * empty`
- `slab_header_t * partial`
- `slab_header_t * full`
- `struct vm_alloc_t * vm_allocator`

3.3.2 Field Documentation

3.3.2.1 `size_t slab_cache_t::obj_size`

Definition at line 16 of file slab.h.

Referenced by `slab_prepare_page()`.

3.3.2.2 `count_t slab_cache_t::per_slab`

Definition at line 17 of file slab.h.

Referenced by `slab_prepare_page()`.

3.3.2.3 slab_header_t* slab_cache_t::empty

Definition at line 18 of file slab.h.

Referenced by slab_prepare_page(), and vm_vfree_block().

3.3.2.4 slab_header_t* slab_cache_t::partial

Definition at line 19 of file slab.h.

Referenced by vm_vfree_block().

3.3.2.5 slab_header_t* slab_cache_t::full

Definition at line 20 of file slab.h.

3.3.2.6 struct vm_alloc_t* slab_cache_t::vm_allocator [read]

Definition at line 21 of file slab.h.

Referenced by vm_vfree_block().

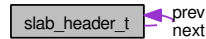
The documentation for this struct was generated from the following file:

- /data/home/phil/svn/jinue/include/**slab.h**

3.4 slab_header_t Struct Reference

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

Collaboration diagram for slab_header_t:



3.4.1 Detailed Description

Definition at line 6 of file slab.h.

Data Fields

- **count_t** available
- **addr_t** free_list
- struct **slab_header_t** * next
- struct **slab_header_t** * prev

3.4.2 Field Documentation

3.4.2.1 count_t slab_header_t::available

Definition at line 7 of file slab.h.

Referenced by slab_prepare_page().

3.4.2.2 addr_t slab_header_t::free_list

Definition at line 8 of file slab.h.

Referenced by slab_prepare_page().

3.4.2.3 struct slab_header_t* slab_header_t::next [read]

Definition at line 9 of file slab.h.

Referenced by slab_prepare_page().

3.4.2.4 struct slab_header_t* slab_header_t::prev [read]

Definition at line 10 of file slab.h.

Referenced by slab_prepare_page().

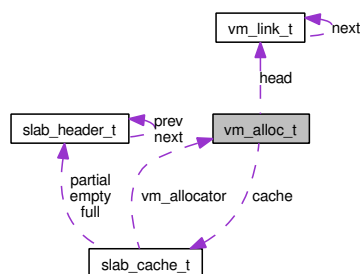
The documentation for this struct was generated from the following file:

- /data/home/phil/svn/jinue/include/**slab.h**

3.5 vm_alloc_t Struct Reference

```
#include <vm_alloc.h>
```

Collaboration diagram for vm_alloc_t:



3.5.1 Detailed Description

Definition at line 15 of file `vm_alloc.h`.

Data Fields

- `size_t size`
- `vm_link_t * head`
- `struct slab_cache_t * cache`

3.5.2 Field Documentation

3.5.2.1 `size_t vm_alloc_t::size`

Definition at line 16 of file `vm_alloc.h`.

Referenced by `alloc_init()`, `e820_is_valid()`, and `printk()`.

3.5.2.2 `vm_link_t* vm_alloc_t::head`

Definition at line 17 of file `vm_alloc.h`.

Referenced by `vm_valloc()`, and `vm_vfree_block()`.

3.5.2.3 `struct slab_cache_t* vm_alloc_t::cache` [read]

Definition at line 18 of file `vm_alloc.h`.

Referenced by `vm__valloc()`, and `vm__vfree__block()`.

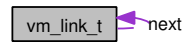
The documentation for this struct was generated from the following file:

- `/data/home/phil/svn/jinue/include/vm__alloc.h`

3.6 vm_link_t Struct Reference

```
#include <vm_alloc.h>
```

Collaboration diagram for vm_link_t:



3.6.1 Detailed Description

Definition at line 7 of file vm_alloc.h.

Data Fields

- struct **vm_link_t** * **next**
- **size_t** **size**
- **addr_t** **addr**

3.6.2 Field Documentation

3.6.2.1 struct vm_link_t* vm_link_t::next [read]

Definition at line 8 of file vm_alloc.h.

Referenced by vm_valloc(), and vm_vfree_block().

3.6.2.2 size_t vm_link_t::size

Definition at line 9 of file vm_alloc.h.

Referenced by vm_valloc(), and vm_vfree_block().

3.6.2.3 addr_t vm_link_t::addr

Definition at line 10 of file vm_alloc.h.

Referenced by vm_valloc(), and vm_vfree_block().

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

- /data/home/phil/svn/jinue/include/**vm_alloc.h**

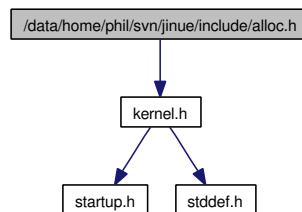
Chapter 4

File Documentation

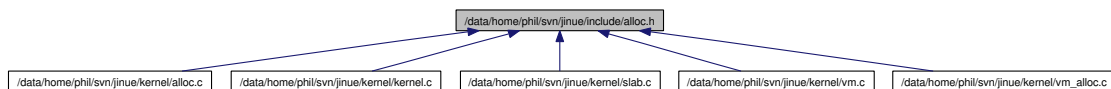
4.1 /data/home/phil/svn/jinue/include/alloc.h File Reference

```
#include <kernel.h>
```

Include dependency graph for alloc.h:



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



Functions

- void `alloc_init` (void)
- `addr_t alloc` (`size_t` size)

4.1.1 Function Documentation

4.1.1.1 `addr_t alloc (size_t size)`

Definition at line 96 of file `alloc.c`.

References `assert`, `PAGE_BITS`, `PAGE_MASK`, `PAGE_SIZE`, and `panic()`.

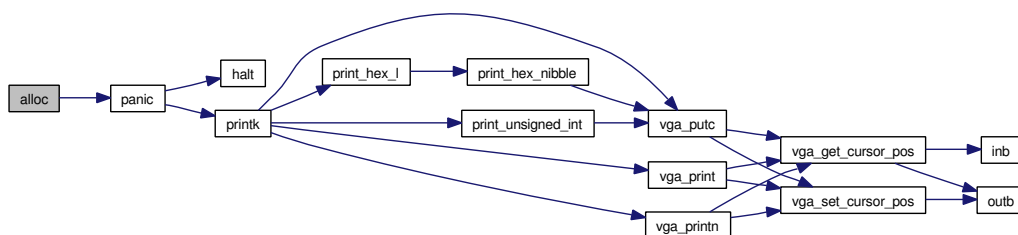
Referenced by `vm_alloc()`, `vm_map()`, and `vm_vfree_block()`.

```

96      {
97          addr_t addr;
98          size_t pages;
99
100         pages = size >> PAGE_BITS;
101
102         if( (size & PAGE_MASK) != 0 ) {
103             ++pages;
104         }
105
106         if(_alloc_size < pages) {
107             panic("out of memory.");
108         }
109
110         addr = _alloc_addr;
111         _alloc_addr += pages * PAGE_SIZE;
112         _alloc_size -= pages;
113
114         /* returned address should be aligned on a page boundary */
115         assert( ((unsigned long)addr & PAGE_MASK) == 0 );
116
117         return addr;
118     }

```

Here is the call graph for this function:



4.1.1.2 `void alloc_init (void)`

Definition at line 12 of file `alloc.c`.

References e820_get_addr(), e820_get_size(), e820_get_type(), e820_is_available(), e820_is_valid(), e820_type_description(), kernel_start, kernel_top, PAGE_SIZE, panic(), printk(), and vm_alloc_t::size.

Referenced by kinit().

```

12         {
13             unsigned int idx;
14             unsigned int remainder;
15             bool avail;
16             size_t size;
17             e820_type_t type;
18             addr_t addr, fixed_addr, best_addr;
19             size_t fixed_size, best_size;
20
21             idx = 0;
22             best_size = 0;
23
24             printk("Dump of the BIOS memory map:\n");
25             printk(" address size      type\n");
26             while( e820_is_valid(idx) ) {
27                 addr = e820_get_addr(idx);
28                 size = e820_get_size(idx);
29                 type = e820_get_type(idx);
30                 avail = e820_is_available(idx);
31
32                 ++idx;
33
34                 printk("%c %x %x %s\n",
35                     avail?'*':' ',
36                     addr,
37                     size,
38                     e820_type_description(type) );
39
40                 if( !avail ) {
41                     continue;
42                 }
43
44                 fixed_addr = addr;
45                 fixed_size = size;
46
47                 /* is the region completely under the kernel ? */
48                 if(addr + size > kernel_start) {
49                     /* is the region completely above the kernel ? */
50                     if(addr < kernel_top) {
51                         /* if the region touches the kernel, we take only
52                          * the part above the kernel, if there is one... */
53                         if(addr + size <= kernel_top) {
54                             /* ... and apparently, there is none */
55                             continue;
56                         }
57
58                         fixed_addr = kernel_top;
59                         fixed_size -= fixed_addr - addr;
60                     }
61                 }
62             }

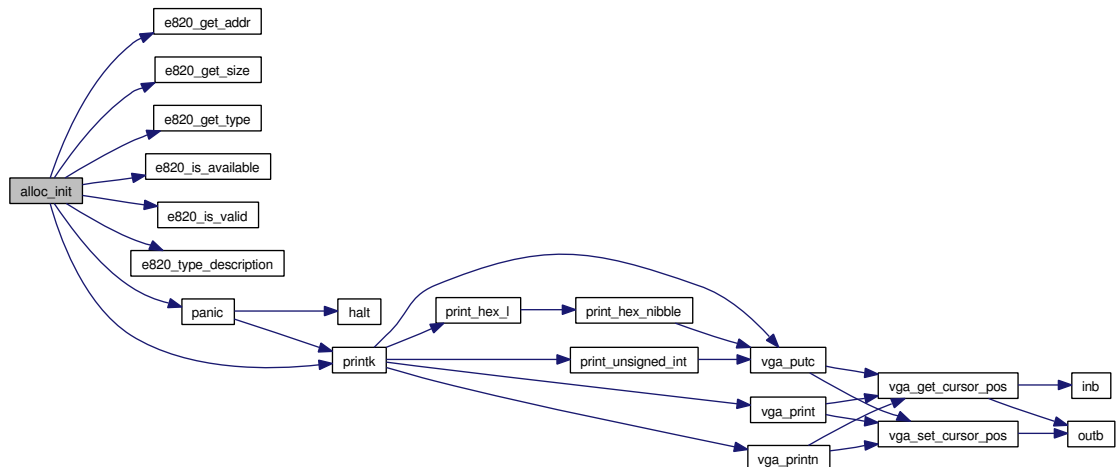
```

```

63      /* we must make sure the starting address is aligned on a
64       * page boundary. The size will eventually be divided
65       * by the page size, and thus need not be aligned. */
66      remainder = (unsigned int)fixed_addr % PAGE_SIZE;
67      if(remainder != 0) {
68          remainder = PAGE_SIZE - remainder;
69          if(fixed_size < remainder) {
70              continue;
71          }
72
73          fixed_addr += remainder;
74          fixed_size -= remainder;
75      }
76
77      if(fixed_size > best_size) {
78          best_addr = fixed_addr;
79          best_size = fixed_size;
80      }
81  }
82
83  _alloc_addr = (addr_t)best_addr;
84  _alloc_size = best_size / PAGE_SIZE;
85
86  if(_alloc_size == 0) {
87      panic("no memory to allocate.");
88  }
89
90  printk("%u kilobytes (%u pages) available starting at %xh.\n",
91         _alloc_size * PAGE_SIZE / 1024,
92         _alloc_size,
93         _alloc_addr );
94 }

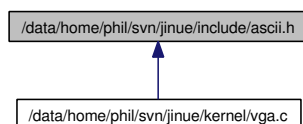
```

Here is the call graph for this function:



4.2 /data/home/phil/svn/jinue/include/ascii.h File Reference

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



Defines

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

4.2.1 Define Documentation

4.2.1.1 `#define CHAR_BS 0x08`

Definition at line 4 of file `ascii.h`.

4.2.1.2 `#define CHAR_CR 0x0d`

Definition at line 7 of file `ascii.h`.

4.2.1.3 `#define CHAR_HT 0x09`

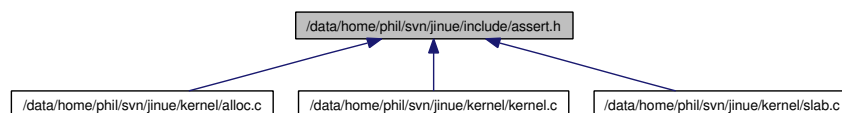
Definition at line 5 of file `ascii.h`.

4.2.1.4 `#define CHAR_LF 0x0a`

Definition at line 6 of file `ascii.h`.

4.3 /data/home/phil/svn/jinue/include/assert.h File Reference

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



Defines

- `#define assert(expr)`

Functions

- `void __assert_failed (const char *expr, const char *file, unsigned int line, const char *func)`

4.3.1 Define Documentation

4.3.1.1 `#define assert(expr)`

Value:

```
( (void) ( \
                (expr)?0:( __assert_failed(#expr, __FILE__, __LINE__, __func__) ) \
    ) )
```

Definition at line 12 of file `assert.h`.

Referenced by `alloc()`, `kinit()`, and `slab_prepare_page()`.

4.3.2 Function Documentation

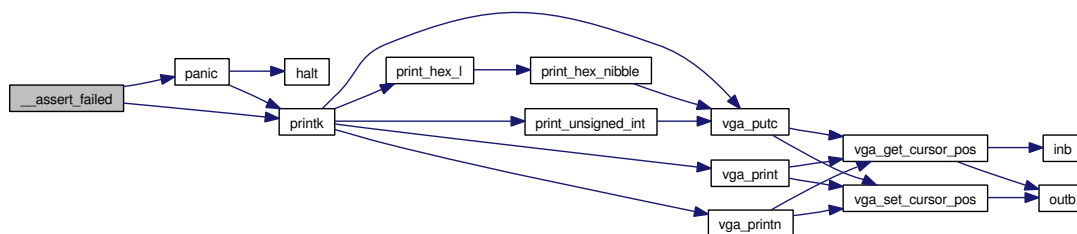
4.3.2.1 `void __assert_failed (const char * expr, const char * file, unsigned int line, const char * func)`

Definition at line 5 of file `assert.c`.

References `panic()`, and `printk()`.


```
9             {
10
11         printk(
12             "ASSERTION FAILED [%s]: %s at line %u in function %s.\n",
13             expr, file, line, func );
14
15         panic("Assertion failed.");
16 }
```

Here is the call graph for this function:

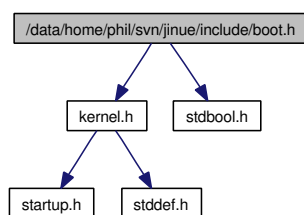


4.4 /data/home/phil/svn/jinue/include/boot.h File Reference

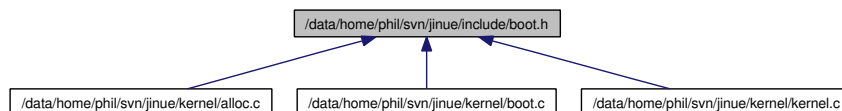
```
#include <kernel.h>
```

```
#include <stdbool.h>
```

Include dependency graph for boot.h:



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



Data Structures

- struct **e820_t**
- struct **boot_t**

Defines

- #define **BOOT_SIGNATURE** 0xaa55
- #define **BOOT_MAGIC** 0xcafef00d
- #define **SETUP_HEADER** 0x53726448
- #define **E820_RAM** 1
- #define **E820_RESERVED** 2
- #define **E820_ACPI** 3

Typedefs

- typedef unsigned long long **e820_addr_t**

- typedef unsigned long long **e820_size_t**
- typedef unsigned long **e820_type_t**

Functions

- **addr_t e820_get_addr** (unsigned int idx)
- **size_t e820_get_size** (unsigned int idx)
- **e820_type_t e820_get_type** (unsigned int idx)
- **bool e820_is_valid** (unsigned int idx)
- **bool e820_is_available** (unsigned int idx)
- **const char * e820_type_description** (e820_type_t type)
- **boot_t * get_boot_data** (void)

4.4.1 Define Documentation

4.4.1.1 **#define BOOT_MAGIC 0xcafef00d**

Definition at line 8 of file boot.h.

Referenced by `get_boot_data()`.

4.4.1.2 **#define BOOT_SIGNATURE 0xaa55**

Definition at line 7 of file boot.h.

Referenced by `get_boot_data()`.

4.4.1.3 **#define E820 ACPI 3**

Definition at line 13 of file boot.h.

Referenced by `e820_type_description()`.

4.4.1.4 **#define E820_RAM 1**

Definition at line 11 of file boot.h.

Referenced by `e820_is_available()`, and `e820_type_description()`.

4.4.1.5 **#define E820_RESERVED 2**

Definition at line 12 of file boot.h.

Referenced by `e820_type_description()`.

4.4.1.6 `#define SETUP_HEADER 0x53726448`

Definition at line 9 of file boot.h.

4.4.2 Typedef Documentation

4.4.2.1 `typedef unsigned long long e820_addr_t`

Definition at line 15 of file boot.h.

4.4.2.2 `typedef unsigned long long e820_size_t`

Definition at line 16 of file boot.h.

4.4.2.3 `typedef unsigned long e820_type_t`

Definition at line 17 of file boot.h.

4.4.3 Function Documentation

4.4.3.1 `addr_t e820_get_addr (unsigned int idx)`

Definition at line 8 of file boot.c.

Referenced by `alloc_init()`.

```

8                                     {
9         return (addr_t)(unsigned long)e820_map[idx].addr;
10 }
```

4.4.3.2 `size_t e820_get_size (unsigned int idx)`

Definition at line 12 of file boot.c.

References `e820_t::size`.

Referenced by `alloc_init()`.

```

12                                     {
13         return (size_t)e820_map[idx].size;
14 }
```

4.4.3.3 e820_type_t e820_get_type (unsigned int *idx*)

Definition at line 16 of file boot.c.

References e820_t::type.

Referenced by alloc_init().

```
16                                     {
17     return e820_map[idx].type;
18 }
```

4.4.3.4 bool e820_is_available (unsigned int *idx*)

Definition at line 24 of file boot.c.

References E820_RAM.

Referenced by alloc_init().

```
24                                     {
25     return (e820_map[idx].type == E820_RAM);
26 }
```

4.4.3.5 bool e820_is_valid (unsigned int *idx*)

Definition at line 20 of file boot.c.

References vm_alloc_t::size.

Referenced by alloc_init().

```
20                                     {
21     return (e820_map[idx].size != 0);
22 }
```

4.4.3.6 const char* e820_type_description (e820_type_t *type*)

Definition at line 28 of file boot.c.

References E820_ACPI, E820_RAM, and E820_RESERVED.

Referenced by alloc_init().

```
28                                     {
29     switch(type) {
30
```

```

31     case E820_RAM:
32         return "available";
33
34     case E820_RESERVED:
35         return "unavailable/reserved";
36
37     case E820_ACPI:
38         return "unavailable/acpi";
39
40     default:
41         return "unavailable/other";
42 }
43 }

```

4.4.3.7 boot_t* get_boot_data (void)

Definition at line 45 of file boot.c.

References `BOOT_MAGIC`, `boot_setup_addr`, `BOOT_SIGNATURE`, `boot_t::magic`, `panic()`, and `boot_t::signature`.

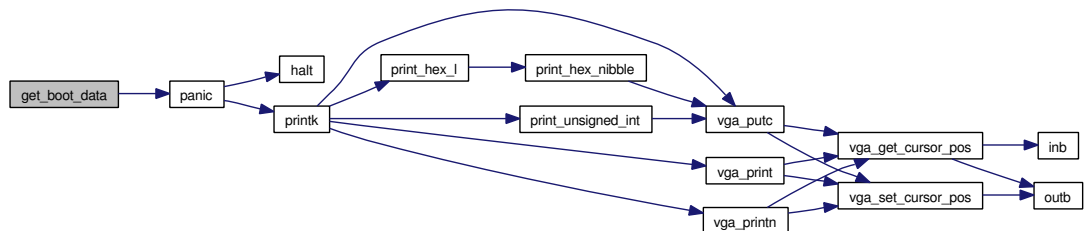
Referenced by `kinit()`.

```

45     {
46     boot_t *boot;
47
48     boot = (boot_t *) ( boot_setup_addr - sizeof(boot_t) );
49
50     if(boot->signature != BOOT_SIGNATURE) {
51         panic("bad boot sector signature.");
52     }
53
54     if(boot->magic != BOOT_MAGIC) {
55         panic("bad boot sector magic.");
56     }
57
58     return boot;
59 }

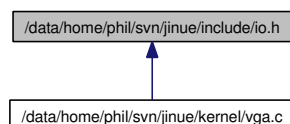
```

Here is the call graph for this function:



4.5 /data/home/phil/svn/jinue/include/io.h File Reference

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



Functions

- unsigned char **inb** (unsigned short int *port*)
- unsigned short int **inw** (unsigned short int *port*)
- unsigned int **inl** (unsigned short int *port*)
- void **outb** (unsigned short int *port*, unsigned char *value*)
- void **outw** (unsigned short int *port*, unsigned short int *value*)
- void **outl** (unsigned short int *port*, unsigned int *value*)

4.5.1 Function Documentation

4.5.1.1 unsigned char inb (unsigned short int *port*)

Referenced by vga_get_cursor_pos(), and vga_init().

4.5.1.2 unsigned int inl (unsigned short int *port*)

4.5.1.3 unsigned short int inw (unsigned short int *port*)

4.5.1.4 void outb (unsigned short int *port*, unsigned char *value*)

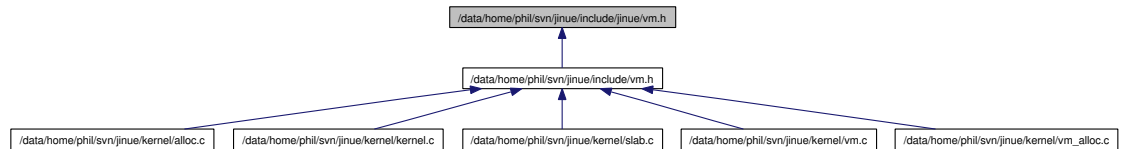
Referenced by vga_get_cursor_pos(), vga_init(), and vga_set_cursor_pos().

4.5.1.5 void outl (unsigned short int *port*, unsigned int *value*)

4.5.1.6 void outw (unsigned short int *port*, unsigned short int *value*)

4.6 /data/home/phil/svn/jinue/include/jinue/vm.h File Reference

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



Defines

- **#define PAGE_BITS** 12
number of bits in virtual address for offset inside page
- **#define PAGE_SIZE** (1<<PAGE_BITS)
size of page
- **#define PAGE_MASK** (PAGE_SIZE - 1)
bit mask for offset in page
- **#define PAGE_OFFSET_OF(x)** ((unsigned long)(x) & PAGE_MASK)
offset in page of virtual address
- **#define KLIMIT** (1<<24)
Virtual address range 0 to KLIMIT is reserved by kernel to store global data structures.
- **#define PLIMIT** (KLIMIT + (1<<24))
Virtual address range KLIMIT to PLIMIT is reserved by kernel to store data structures specific to the current process.

4.6.1 Define Documentation

4.6.1.1 **#define KLIMIT** (1<<24)

Virtual address range 0 to KLIMIT is reserved by kernel to store global data structures.

4.6 /data/home/phil/svn/jinue/include/jinue/vm.h File Reference

Kernel image must be completely inside this region. This region has the same mapping in the address space of all processes. Size must be a multiple of the size described by a single page directory entry (`PTE_SIZE * PAGE_SIZE`).

Definition at line 22 of file `vm.h`.

4.6.1.2 `#define PAGE_BITS 12`

number of bits in virtual address for offset inside page

Definition at line 5 of file `vm.h`.

Referenced by `alloc()`.

4.6.1.3 `#define PAGE_MASK (PAGE_SIZE - 1)`

bit mask for offset in page

Definition at line 11 of file `vm.h`.

Referenced by `alloc()`.

4.6.1.4 `#define PAGE_OFFSET_OF(x) ((unsigned long)(x) & PAGE_MASK)`

offset in page of virtual address

Definition at line 14 of file `vm.h`.

Referenced by `slab_prepare_page()`.

4.6.1.5 `#define PAGE_SIZE (1<<PAGE_BITS)`

size of page

Definition at line 8 of file `vm.h`.

Referenced by `alloc()`, `alloc_init()`, `kinit()`, `vm_alloc()`, `vm_map()`, `vm_valloc()`, `vm_vfree()`, and `vm_vfree_block()`.

4.6.1.6 `#define PLIMIT (KLIMIT + (1<<24))`

Virtual address range `KLIMIT` to `PLIMIT` is reserved by kernel to store data structures specific to the current process.

The mapping of this region changes from one address space to the next. Size must be a multiple of the size described by a single page directory entry (`PTE_SIZE * PAGE_SIZE`).

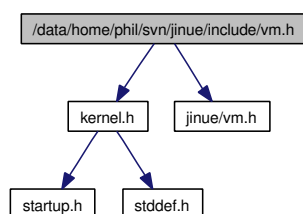
Definition at line 30 of file vm.h.

4.7 /data/home/phil/svn/jinue/include/vm.h File Reference

```
#include <kernel.h>
```

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

Include dependency graph for vm.h:



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



Defines

- `#define PAGE_TABLE_BITS 10`
number of bits in virtual address for page table entry
- `#define PAGE_TABLE_ENTRIES (1<<PAGE_TABLE_BITS)`
number of entries in page table
- `#define PAGE_TABLE_MASK (PAGE_TABLE_ENTRIES - 1)`
bit mask for page table entry
- `#define PAGE_TABLE_SIZE PAGE_SIZE`
size of a page table
- `#define PTE_SIZE 4`
size of a page table entry, in bytes
- `#define PAGE_TABLE_OFFSET_OF(x) (((unsigned long)(x) >> PAGE_BITS) & PAGE_TABLE_MASK)`

page table entry offset of virtual address

- **#define PAGE_DIRECTORY_OFFSET_OF(x)** ((unsigned long)(x) >> (PAGE_BITS + PAGE_TABLE_BITS))

page directory entry offset of virtual address

- **#define PAGE_TABLES_MAPPING_KLIMIT**

This is where the page tables are mapped in every address space.

- **#define PAGE_DIRECTORY_MAPPING (KLIMIT + PAGE_TABLE_ENTRIES * PAGE_TABLE_SIZE)**

This is where the page directory is mapped in every address space.

- **#define PMAPPING_START (PAGE_DIRECTORY + PAGE_TABLE_SIZE)**

limits of region spanning from KLIMIT to PLIMIT actually available for mappings

- **#define PMAPPING_END PLIMIT**

- **#define PAGE_DIRECTORY (pte_t *)PAGE_DIRECTORY_MAPPING)**

page directory in virtual memory

- **#define PAGE_TABLES (page_table_t *)PAGE_TABLES_MAPPING)**

page tables in virtual memory

- **#define PAGE_TABLE_OF(x) (PAGE_TABLES[PAGE_DIRECTORY_OFFSET_OF(x)])**

page table in virtual memory

- **#define PDE_OF(x) (&PAGE_DIRECTORY[PAGE_DIRECTORY_OFFSET_OF(x)])**

address of page directory entry in virtual memory

- **#define PTE_OF(x) (&PAGE_TABLE_OF(x)[PAGE_TABLE_OFFSET_OF(x)])**

address of page table entry in virtual memory

- **#define PAGE_TABLES_TABLE (PAGE_TABLE_OF(PAGE_TABLES_MAPPING))**

page table which maps all page tables in memory

- `#define PAGE_TABLE_PTE_OF(x) (&PAGE_TABLES_[PAGE_DIRECTORY_OFFSET_OF(x)])`
address of page entry in PAGE_OF_PAGE_TABLES
- `#define VM_FLAG_PRESENT (1<< 0)`
page is present in memory
- `#define VM_FLAG_READ_ONLY (1<< 1)`
page is read only
- `#define VM_FLAG_KERNEL 0`
kernel mode page (default)
- `#define VM_FLAG_USER (1<< 2)`
user mode page
- `#define VM_FLAG_WRITE_THROUGH (1<< 3)`
write-through cache policy for page
- `#define VM_FLAG_CACHE_DISABLE (1<< 4)`
uncached page
- `#define VM_FLAG_ACCESSED (1<< 5)`
page was accessed (read)
- `#define VM_FLAG_DIRTY (1<< 6)`
page was written to
- `#define VM_FLAG_BIG_PAGE (1<< 7)`
page directory entry describes a 4M page
- `#define VM_FLAG_GLOBAL (1<< 8)`
page is global (mapped in every address space)

Typedefs

- `typedef unsigned long pte_t`
type of a page table (or page directory) entry
- `typedef pte_t page_table_t [PAGE_TABLE_ENTRIES]`
type of a page table

Functions

- void **vm_map** (**addr_t** vaddr, **addr_t** paddr, unsigned long flags)
- void **vm_unmap** (**addr_t** addr)

4.7.1 Define Documentation

4.7.1.1 `#define PAGE_DIRECTORY (pte_t
*)PAGE_DIRECTORY_MAPPING)`

page directory in virtual memory

Definition at line 58 of file vm.h.

4.7.1.2 `#define PAGE_DIRECTORY_MAPPING (KLIMIT +
PAGE_TABLE_ENTRIES * PAGE_TABLE_SIZE)`

This is where the page directory is mapped in every address space.

It must reside in region spanning from KLIMIT to PLIMIT.

Definition at line 47 of file vm.h.

4.7.1.3 `#define PAGE_DIRECTORY_OFFSET -
OF(x) ((unsigned long)(x) >> (PAGE_BITS +
PAGE_TABLE_BITS))`

page directory entry offset of virtual address

Definition at line 31 of file vm.h.

4.7.1.4 `#define PAGE_TABLE_BITS 10`

number of bits in virtual address for page table entry

Definition at line 13 of file vm.h.

4.7.1.5 `#define PAGE_TABLE_ENTRIES (1<<PAGE_
TABLE_BITS)`

number of entries in page table

Definition at line 16 of file vm.h.

Referenced by vm_map().

4.7.1.6 `#define PAGE_TABLE_MASK (PAGE_TABLE_ENTRIES - 1)`

bit mask for page table entry

Definition at line 19 of file vm.h.

4.7.1.7 `#define PAGE_TABLE_OF(x) (PAGE_TABLES[PAGE_DIRECTORY_OFFSET_OF(x)])`

page table in virtual memory

Definition at line 64 of file vm.h.

Referenced by vm_map().

4.7.1.8 `#define PAGE_TABLE_OFFSET_OF(x) (((unsigned long)(x) >> PAGE_BITS) & PAGE_TABLE_MASK)`

page table entry offset of virtual address

Definition at line 28 of file vm.h.

4.7.1.9 `#define PAGE_TABLE_PTE_OF(x) (&PAGE_TABLES_TABLE[PAGE_DIRECTORY_OFFSET_OF(x)])`

address of page entry in PAGE_OF_PAGE_TABLES

Definition at line 76 of file vm.h.

Referenced by vm_map().

4.7.1.10 `#define PAGE_TABLE_SIZE PAGE_SIZE`

size of a page table

Definition at line 22 of file vm.h.

4.7.1.11 `#define PAGE_TABLES (page_table_t *)PAGE_TABLES_MAPPING)`

page tables in virtual memory

Definition at line 61 of file vm.h.

4.7.1.12 #define PAGE_TABLES_MAPPING KLIMIT

This is where the page tables are mapped in every address space.

This requires a virtual memory region of size 4M, which must reside completely inside region spanning from KLIMIT to PLIMIT. Must be aligned on a 4M boundary

Definition at line 43 of file vm.h.

4.7.1.13 #define PAGE_TABLES_TABLE (PAGE_TABLE_OF(PAGE_TABLES_MAPPING))

page table which maps all page tables in memory

Definition at line 73 of file vm.h.

4.7.1.14 #define PDE_OF(x) (&PAGE_DIRECTORY[PAGE_DIRECTORY_OFFSET_OF(x)])

address of page directory entry in virtual memory

Definition at line 67 of file vm.h.

Referenced by vm_map().

4.7.1.15 #define PMAPPING_END PLIMIT

Definition at line 52 of file vm.h.

4.7.1.16 #define PMAPPING_START (PAGE_DIRECTORY + PAGE_TABLE_SIZE)

limits of region spanning from KLIMIT to PLIMIT actually available for mappings

Definition at line 51 of file vm.h.

4.7.1.17 #define PTE_OF(x) (&PAGE_TABLE_OF(x)[PAGE_TABLE_OFFSET_OF(x)])

address of page table entry in virtual memory

Definition at line 70 of file vm.h.

Referenced by vm_map(), and vm_unmap().

4.7.1.18 #define PTE_SIZE 4

size of a page table entry, in bytes

Definition at line 25 of file vm.h.

4.7.1.19 #define VM_FLAG_ACCESSED (1<< 5)

page was accessed (read)

Definition at line 100 of file vm.h.

4.7.1.20 #define VM_FLAG_BIG_PAGE (1<< 7)

page directory entry describes a 4M page

Definition at line 106 of file vm.h.

4.7.1.21 #define VM_FLAG_CACHE_DISABLE (1<< 4)

uncached page

Definition at line 97 of file vm.h.

4.7.1.22 #define VM_FLAG_DIRTY (1<< 6)

page was written to

Definition at line 103 of file vm.h.

4.7.1.23 #define VM_FLAG_GLOBAL (1<< 8)

page is global (mapped in every address space)

Definition at line 109 of file vm.h.

4.7.1.24 #define VM_FLAG_KERNEL 0

kernel mode page (default)

Definition at line 88 of file vm.h.

4.7.1.25 #define VM_FLAG_PRESENT (1<< 0)

page is present in memory

Definition at line 82 of file vm.h.

Referenced by `vm_map()`.

4.7.1.26 `#define VM_FLAG_READ_ONLY (1<< 1)`

page is read only

Definition at line 85 of file vm.h.

4.7.1.27 `#define VM_FLAG_USER (1<< 2)`

user mode page

Definition at line 91 of file vm.h.

4.7.1.28 `#define VM_FLAG_WRITE_THROUGH (1<< 3)`

write-through cache policy for page

Definition at line 94 of file vm.h.

4.7.2 Typedef Documentation

4.7.2.1 `typedef pte_t page_table_t[PAGE_TABLE_ENTRIES]`

type of a page table

Definition at line 34 of file vm.h.

4.7.2.2 `typedef unsigned long pte_t`

type of a page table (or page directory) entry

Definition at line 10 of file vm.h.

4.7.3 Function Documentation

4.7.3.1 `void vm_map(addr_t vaddr, addr_t paddr, unsigned long flags)`

Definition at line 5 of file vm.c.

References `alloc()`, `PAGE_SIZE`, `PAGE_TABLE_ENTRIES`, `PAGE_TABLE_OF`, `PAGE_TABLE_PTE_OF`, `PDE_OF`, `PTE_OF`, and `VM_FLAG_PRESENT`.

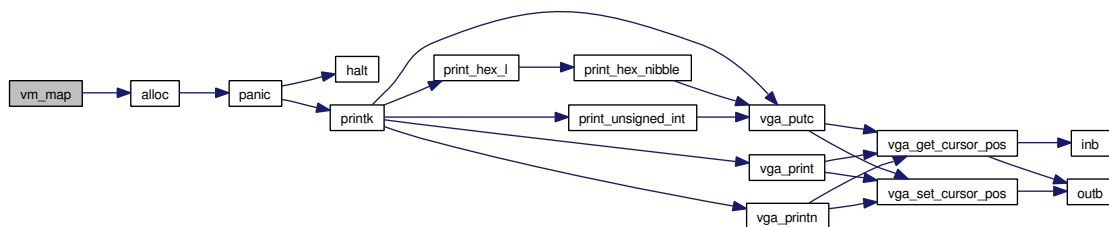
Referenced by `vm_alloc()`.

```

5                                     {
6     pte_t *pte;
7     addr_t page_table;
8     int idx;
9
10    pte = PDE_OF(vaddr);
11
12    /* check if page table must be created */
13    if( !(*pte & VM_FLAG_PRESENT) ) {
14        /* allocate a page for page table */
15        page_table = alloc(PAGE_SIZE);
16
17        /* link to page table from page directory */
18        *pte = (pte_t)page_table | VM_FLAG_PRESENT;
19
20        /* map page table in the region of memory reserved for that purpose */
21        pte = PAGE_TABLE_PTE_OF(vaddr);
22        *pte = (pte_t)page_table | VM_FLAG_PRESENT;
23
24        /* obtain virtual address of new page table */
25        pte = PAGE_TABLE_OF(vaddr);
26
27        /* zero content of page table */
28        for(idx = 0; idx < PAGE_TABLE_ENTRIES; ++idx) {
29            pte[idx] = 0;
30        }
31    }
32
33    /* perform the actual mapping */
34    pte = PTE_OF(vaddr);
35    *pte = (pte_t)paddr | VM_FLAG_PRESENT;
36 }

```

Here is the call graph for this function:



4.7.3.2 void vm_unmap (addr_t *addr*)

Definition at line 38 of file vm.c.

References PTE_OF.

Referenced by vm_free().

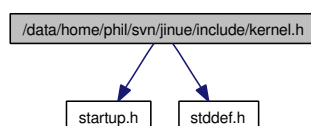
```
38                                     {
39     pte_t *pte;
40
41     pte = PTE_OF(addr);
42     *pte = 0;
43 }
```

4.8 /data/home/phil/svn/jinue/include/kernel.h File Reference

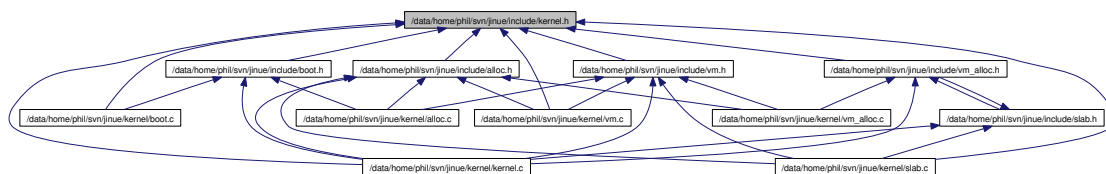
```
#include <startup.h>
```

```
#include <stddef.h>
```

Include dependency graph for kernel.h:



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



Defines

- `#define kernel_start ((addr_t)start)`

Typedefs

- `typedef void * addr_t`
- `typedef unsigned long count_t`

Functions

- `void kernel (void)`
- `void kinit (void)`
- `void idle (void)`

Variables

- `addr_t kernel_top`
- `size_t kernel_size`

4.8.1 Define Documentation

4.8.1.1 `#define kernel_start ((addr_t)start)`

Definition at line 10 of file kernel.h.

Referenced by `alloc_init()`, and `kinit()`.

4.8.2 Typedef Documentation

4.8.2.1 `typedef void* addr_t`

Definition at line 7 of file kernel.h.

4.8.2.2 `typedef unsigned long count_t`

Definition at line 8 of file kernel.h.

4.8.3 Function Documentation

4.8.3.1 `void idle (void)`

Definition at line 52 of file kernel.c.

Referenced by `kernel()`.

```
52         {  
53     while(1) {}  
54 }
```

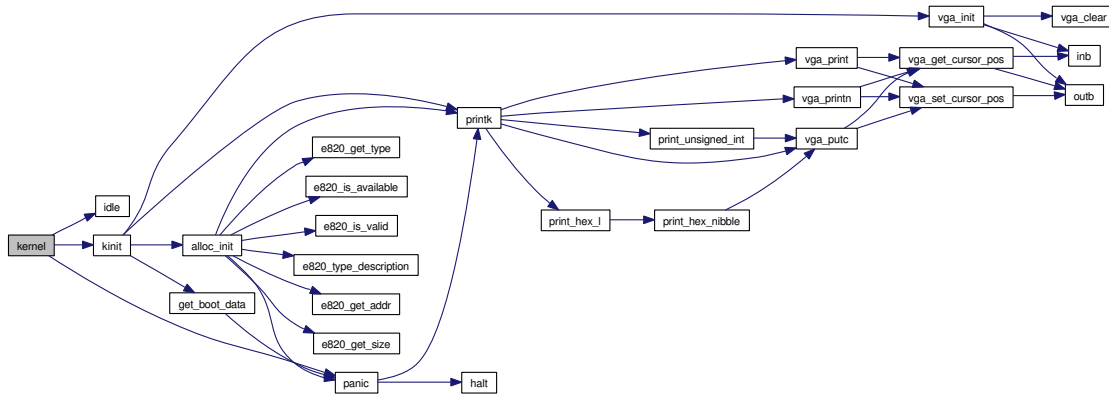
4.8.3.2 `void kernel (void)`

Definition at line 16 of file kernel.c.

References `idle()`, `kinit()`, and `panic()`.

```
16     {  
17     kinit();  
18     idle();  
19  
20     panic("idle() returned.");  
21 }
```

Here is the call graph for this function:



4.8.3.3 void kinit (void)

Definition at line 23 of file kernel.c.

References alloc_init(), assert(), get_boot_data(), kernel_size, kernel_start, kernel_top, PAGE_SIZE, printk(), boot_t::sysize, and vga_init().

Referenced by kernel().

```

23     {
24         boot_t *boot;
25         unsigned int remainder;
26
27         /* say hello */
28         vga_init();
29         printk("Kernel started.\n");
30
31         /* we assume the kernel starts on a page boundary */
32         assert((unsigned int)kernel_start % PAGE_SIZE == 0);
33
34         /* find out kernel size and set kernel_top
35          * (top of kernel, aligned to page boundary) */
36         boot = get_boot_data();
37
38         kernel_size = boot->sysize * 16;
39         remainder   = kernel_size % PAGE_SIZE;
40
41         printk("Kernel size is %u (+%u) bytes.\n", kernel_size, PAGE_SIZE - remainder);
42
43         if(remainder != 0) {
44             kernel_size += PAGE_SIZE - remainder;
45         }
46         kernel_top  = kernel_start + kernel_size;
47     }

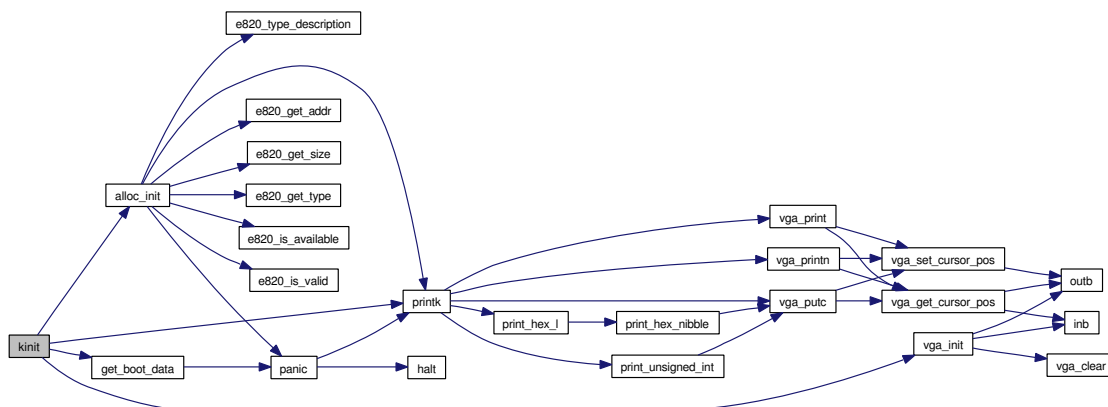
```

```

48      /* initialize allocator */
49      alloc_init();
50 }

```

Here is the call graph for this function:



4.8.4 Variable Documentation

4.8.4.1 size_t kernel_size

Definition at line 14 of file kernel.c.

Referenced by kinit().

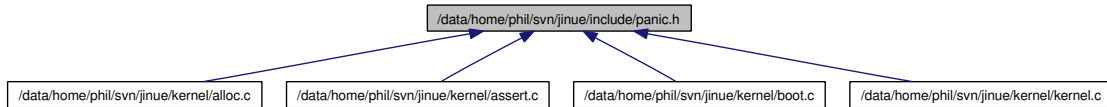
4.8.4.2 addr_t kernel_top

Definition at line 13 of file kernel.c.

Referenced by alloc_init(), and kinit().

4.9 /data/home/phil/svn/jinue/include/panic.h File Reference

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



Functions

- void **panic** (const char *message)

4.9.1 Function Documentation

4.9.1.1 void panic (const char * *message*)

Definition at line 4 of file panic.c.

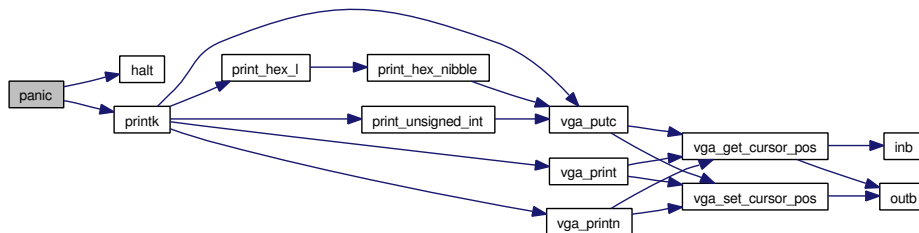
References halt(), and printk().

Referenced by __assert_failed(), alloc(), alloc_init(), get_boot_data(), and kernel().

```

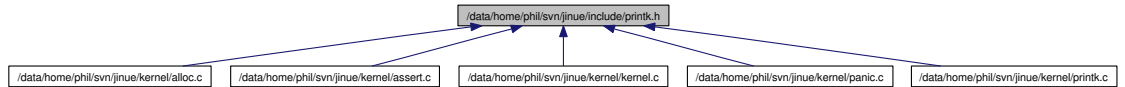
4          {
5      printk("KERNEL PANIC: %s\n", message);
6      halt();
7  }
```

Here is the call graph for this function:



4.10 /data/home/phil/svn/jinue/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)
- void **print_hex_nibble** (unsigned char byte)
- void **print_hex_b** (unsigned char byte)
- void **print_hex_w** (unsigned short word)
- void **print_hex_l** (unsigned long dword)
- void **print_hex_q** (unsigned long long qword)

4.10.1 Function Documentation

4.10.1.1 void print_hex_b (unsigned char *byte*)

Definition at line 105 of file printk.c.

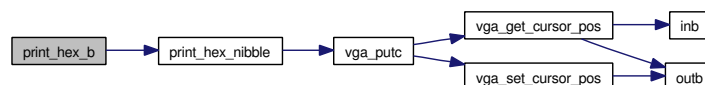
References `print_hex_nibble()`.

```

105     {
106     print_hex_nibble( (char)byte );
107     print_hex_nibble( (char)(byte>>4) );
108 }

```

Here is the call graph for this function:



4.10.1.2 void print_hex_l (unsigned long *dword*)

Definition at line 118 of file printk.c.

References `print_hex_nibble()`.

Referenced by `printk()`.

```

118                                     {
119     int off;
120
121     for(off=32-4; off>=0; off-=4) {
122         print_hex_nibble( (char)(dword>>off) );
123     }
124 }
```

Here is the call graph for this function:

4.10.1.3 void print_hex_nibble (unsigned char *byte*)

Definition at line 91 of file printk.c.

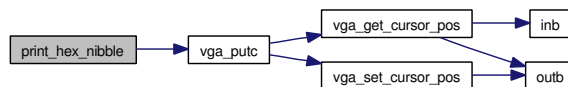
References `vga_putc()`.

Referenced by `print_hex_b()`, `print_hex_l()`, `print_hex_q()`, and `print_hex_w()`.

```

91                                     {
92     char c;
93
94     c = byte & 0xf;
95     if(c < 10) {
96         c += '0';
97     }
98     else {
99         c+= ('a' - 10);
100    }
101
102    vga_putc(c);
103 }
```

Here is the call graph for this function:



4.10.1.4 void print_hex_q (unsigned long long *qword*)

Definition at line 126 of file printk.c.

References print_hex_nibble().

```

126                                     {
127     int off;
128
129     for(off=64-4; off>=0; off-=4) {
130         print_hex_nibble( (char)(qword>>off) );
131     }
132 }
```

Here is the call graph for this function:



4.10.1.5 void print_hex_w (unsigned short *word*)

Definition at line 110 of file printk.c.

References print_hex_nibble().

```

110                                     {
111     int off;
112
113     for(off=16-4; off>=0; off-=4) {
114         print_hex_nibble( (char)(word>>off) );
115     }
116 }
```

Here is the call graph for this function:



4.10.1.6 void print_unsigned_int (unsigned int *n*)

Definition at line 67 of file printk.c.

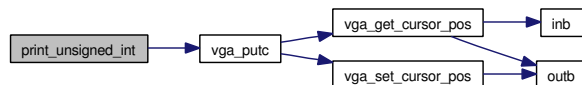
References `vga_putc()`.

Referenced by `printk()`.

```

67                                     {
68     unsigned int flag = 0;
69     unsigned int pwr;
70     unsigned int digit;
71     char c;
72
73     if(n == 0) {
74         vga_putc('0');
75         return;
76     }
77
78     for(pwr = 1000 * 1000 * 1000; pwr > 0; pwr /= 10) {
79         digit = n / pwr;
80
81         if(digit != 0 || flag) {
82             c = (char)digit + '0';
83             vga_putc(c);
84
85             flag = 1;
86             n -= digit * pwr;
87         }
88     }
89 }
```

Here is the call graph for this function:

4.10.1.7 void printk (const char * *format*, ...)

Definition at line 6 of file printk.c.

References `print_hex_l()`, `print_unsigned_int()`, `vm_alloc_t::size`, `va_arg`, `va_end`, `va_start`, `vga_print()`, `vga_printn()`, and `vga_putc()`.

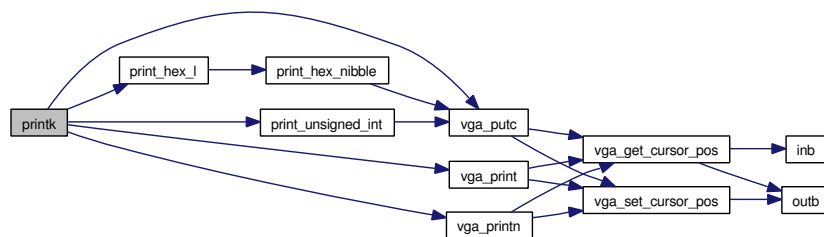
Referenced by `__assert_failed()`, `alloc_init()`, `kinit()`, and `panic()`.

```

6                                     {
7     va_list ap;
8     const char *idx, *anchor;
```

```
9      ptrdiff_t size;
10
11      va_start(ap, format);
12
13      idx = format;
14
15      while(1) {
16          anchor = idx;
17
18          while( *idx != 0 && *idx != '%' ) {
19              ++idx;
20          }
21
22          size = idx - anchor;
23
24          if(size > 0) {
25              vga_printn(anchor, size);
26          }
27
28          if(*idx == 0 || *(idx+1) == 0) {
29              break;
30          }
31
32          ++idx;
33
34          switch( *idx ) {
35              case '%':
36                  vga_putc('%');
37                  break;
38
39              case 'c':
40                  /* promotion, promotion */
41                  vga_putc( (char)va_arg(ap, int) );
42                  break;
43
44              case 's':
45                  vga_print( va_arg(ap, const char *) );
46                  break;
47
48              case 'u':
49                  print_unsigned_int( va_arg(ap, unsigned int) );
50                  break;
51
52              case 'x':
53                  print_hex_l( va_arg(ap, unsigned long) );
54                  break;
55
56              default:
57                  va_end(ap);
58                  return;
59          }
60
61          ++idx;
62      }
63
64      va_end(ap);
65 }
```

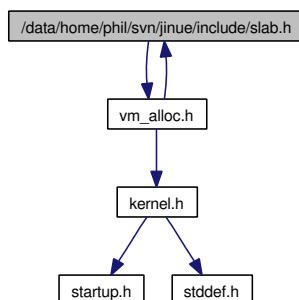
Here is the call graph for this function:



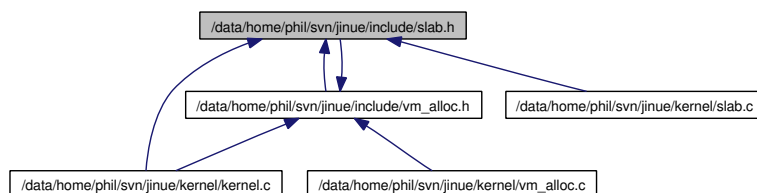
4.11 /data/home/phil/svn/jinue/include/slab.h File Reference

```
#include <vm_alloc.h>
```

Include dependency graph for slab.h:



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



Data Structures

- struct `slab_header_t`
- struct `slab_cache_t`

Typedefs

- typedef struct `slab_header_t` `slab_header_t`
- typedef struct `slab_cache_t` `slab_cache_t`

Functions

- `addr_t` `slab_alloc` (`slab_cache_t` *cache)
- void `slab_free` (`slab_cache_t` *cache, `addr_t` obj)

- void slab_prepare_page (slab_cache_t *cache, addr_t page)

4.11.1 Typedef Documentation

4.11.1.1 typedef struct slab_cache_t slab_cache_t

Definition at line 24 of file slab.h.

4.11.1.2 typedef struct slab_header_t slab_header_t

Definition at line 13 of file slab.h.

4.11.2 Function Documentation

4.11.2.1 addr_t slab_alloc (slab_cache_t * *cache*)

Definition at line 7 of file slab.c.

References NULL.

Referenced by vm_vfree_block().

```
7                                     {  
8         return NULL;  
9 }
```

4.11.2.2 void slab_free (slab_cache_t * *cache*, addr_t *obj*)

Definition at line 11 of file slab.c.

Referenced by vm_valloc().

```
11                                     {  
12 }
```

4.11.2.3 void slab_prepare_page (slab_cache_t * *cache*, addr_t *page*)

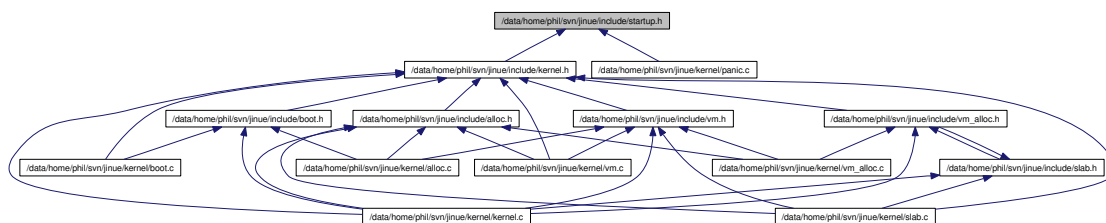
Definition at line 14 of file slab.c.

References assert, slab_header_t::available, slab_cache_t::empty, slab_header_t::free_list, slab_header_t::next, NULL, slab_cache_t::obj_size, PAGE_OFFSET_OF, slab_cache_t::per_slab, and slab_header_t::prev.

```
14                                     {
15     unsigned int cx;
16     size_t obj_size;
17     count_t per_slab;
18     slab_header_t *slab;
19     addr_t *ptr;
20     addr_t next;
21
22     /* we assume "page" is the starting address of a page */
23     assert( PAGE_OFFSET_OF(page) );
24
25     obj_size = cache->obj_size;
26     per_slab = cache->per_slab;
27
28     slab = (slab_header_t *)page;
29     slab->available = per_slab;
30     slab->free_list = page + sizeof(slab_header_t);
31
32     /* create free list */
33     ptr = (addr_t *)slab->free_list;
34
35     for(cx = 0; cx < per_slab - 1; ++cx) {
36         next = ptr + obj_size;
37         *ptr = next;
38         ptr = (addr_t *)next;
39     }
40
41     *ptr = NULL;
42
43     /* insert in list of empty slabs of cache */
44     slab->prev = NULL;
45     slab->next = cache->empty;
46     cache->empty = slab;
47 }
```

4.12 /data/home/phil/svn/jinue/include/startup.h File Reference

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



Functions

- void **start** (void)
- void **halt** (void)

4.12.1 Function Documentation

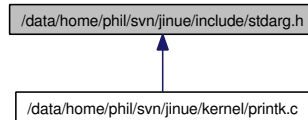
4.12.1.1 void halt (void)

Referenced by panic().

4.12.1.2 void start (void)

4.13 /data/home/phil/svn/jinue/include/stdarg.h File Reference

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



Defines

- `#define va_start(ap, parmN) __builtin_stdarg_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`

4.13.1 Define Documentation

4.13.1.1 `#define va_arg __builtin_va_arg`

Definition at line 7 of file stdarg.h.

Referenced by `printk()`.

4.13.1.2 `#define va_copy(dest, src) __builtin_va_copy((dest), (src))`

Definition at line 9 of file stdarg.h.

4.13.1.3 `#define va_end __builtin_va_end`

Definition at line 8 of file stdarg.h.

Referenced by `printk()`.

4.13.1.4 `#define va_start(ap, parmN) __builtin_stdarg_start((ap), (parmN))`

Definition at line 6 of file stdarg.h.

Referenced by printf().

4.13.2 Typedef Documentation

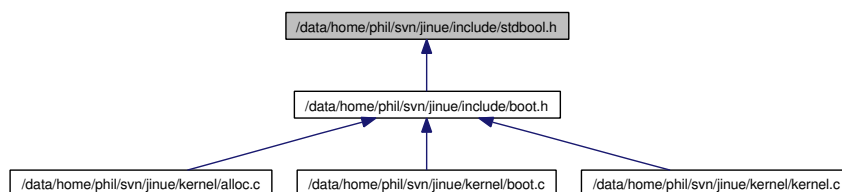
4.13.2.1 `typedef __builtin_va_list va_list`

Definition at line 4 of file stdarg.h.

4.14 /data/home/phil/svn/jinue/include/stdbool.h

File Reference

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



Defines

- `#define bool _Bool`
- `#define true 1`
- `#define false 0`
- `#define __bool_true_false_are_defined 1`

4.14.1 Define Documentation

4.14.1.1 `#define __bool_true_false_are_defined 1`

Definition at line 8 of file stdbool.h.

4.14.1.2 `#define bool _Bool`

Definition at line 4 of file stdbool.h.

4.14.1.3 `#define false 0`

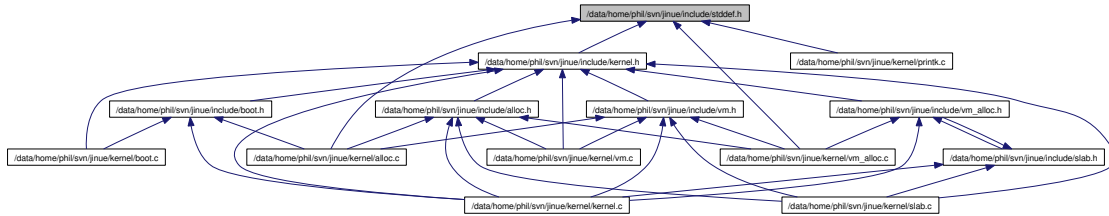
Definition at line 6 of file stdbool.h.

4.14.1.4 `#define true 1`

Definition at line 5 of file stdbool.h.

4.15 /data/home/phil/svn/jinue/include/stddef.h File Reference

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



Defines

- `#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`

4.15.1 Define Documentation

4.15.1.1 `#define NULL 0`

Definition at line 9 of file `stddef.h`.

Referenced by `slab_alloc()`, `slab_prepare_page()`, `vm_valloc()`, and `vm_vfree_block()`.

4.15.1.2 `#define offsetof(type, member) ((size_t) &((type *)0) -> member)`

Definition at line 12 of file `stddef.h`.

4.15.2 Typedef Documentation

4.15.2.1 typedef signed long ptrdiff_t

Definition at line 4 of file stddef.h.

4.15.2.2 typedef unsigned long size_t

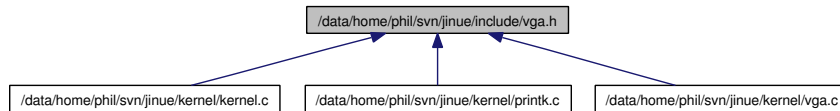
Definition at line 5 of file stddef.h.

4.15.2.3 typedef int wchar_t

Definition at line 6 of file stddef.h.

4.16 /data/home/phil/svn/jinue/include/vga.h File Reference

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



Defines

- #define **VGA_TEXT_VID_BASE** 0xb8000
- #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_DEFAULT** VGA_COLOR_GREEN
- #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_clear** (void)
- void **vga_print** (const char *message)
- void **vga_printn** (const char *message, unsigned int n)
- void **vga_putc** (char c)
- void **vga_scroll** (void)
- **vga_pos_t** **vga_get_cursor_pos** (void)
- void **vga_set_cursor_pos** (**vga_pos_t** pos)

4.16.1 Define Documentation

4.16.1.1 **#define VGA_COL(x) ((x) % (VGA_WIDTH))**

Definition at line 36 of file vga.h.

4.16.1.2 **#define VGA_COLOR_BLACK 0x00**

Definition at line 12 of file vga.h.

4.16.1.3 **#define VGA_COLOR_BLUE 0x01**

Definition at line 13 of file vga.h.

4.16.1.4 **#define VGA_COLOR_BRIGHTBLUE 0x09**

Definition at line 21 of file vga.h.

4.16.1.5 **#define VGA_COLOR_BRIGHTCYAN 0x0b**

Definition at line 23 of file vga.h.

4.16.1.6 **#define VGA_COLOR_BRIGHTGREEN 0x0a**

Definition at line 22 of file vga.h.

4.16.1.7 `#define VGA_COLOR_BRIGHTMAGENTA 0x0d`

Definition at line 25 of file vga.h.

4.16.1.8 `#define VGA_COLOR_BRIGHTRED 0x0c`

Definition at line 24 of file vga.h.

4.16.1.9 `#define VGA_COLOR_BRIGHTWHITE 0x0f`

Definition at line 27 of file vga.h.

4.16.1.10 `#define VGA_COLOR_BROWN 0x06`

Definition at line 18 of file vga.h.

4.16.1.11 `#define VGA_COLOR_CYAN 0x03`

Definition at line 15 of file vga.h.

4.16.1.12 `#define VGA_COLOR_DEFAULT VGA_COLOR_GREEN`

Definition at line 28 of file vga.h.

4.16.1.13 `#define VGA_COLOR_ERASE VGA_COLOR_RED`

Definition at line 29 of file vga.h.

Referenced by vga_clear(), and vga_scroll().

4.16.1.14 `#define VGA_COLOR_GRAY 0x08`

Definition at line 20 of file vga.h.

4.16.1.15 `#define VGA_COLOR_GREEN 0x02`

Definition at line 14 of file vga.h.

4.16.1.16 `#define VGA_COLOR_MAGENTA 0x05`

Definition at line 17 of file vga.h.

4.16.1.17 `#define VGA_COLOR_RED 0x04`

Definition at line 16 of file vga.h.

4.16.1.18 `#define VGA_COLOR_WHITE 0x07`

Definition at line 19 of file vga.h.

4.16.1.19 `#define VGA_COLOR_YELLOW 0x0e`

Definition at line 26 of file vga.h.

4.16.1.20 `#define VGA_CRTC_ADDR 0x3d4`

Definition at line 7 of file vga.h.

Referenced by `vga_get_cursor_pos()`, `vga_init()`, and `vga_set_cursor_pos()`.

4.16.1.21 `#define VGA_CRTC_DATA 0x3d5`

Definition at line 8 of file vga.h.

Referenced by `vga_get_cursor_pos()`, `vga_init()`, and `vga_set_cursor_pos()`.

4.16.1.22 `#define VGA_FB_FLAG_ACTIVE 1`

Definition at line 10 of file vga.h.

4.16.1.23 `#define VGA_LINE(x) ((x) / (VGA_WIDTH))`

Definition at line 35 of file vga.h.

4.16.1.24 `#define VGA_LINES 25`

Definition at line 31 of file vga.h.

Referenced by `vga_clear()`, and `vga_scroll()`.

4.16.1.25 `#define VGA_MISC_OUT_RD 0x3cc`

Definition at line 6 of file `vga.h`.

Referenced by `vga_init()`.

4.16.1.26 `#define VGA_MISC_OUT_WR 0x3c2`

Definition at line 5 of file `vga.h`.

Referenced by `vga_init()`.

4.16.1.27 `#define VGA_TAB_WIDTH 8`

Definition at line 33 of file `vga.h`.

4.16.1.28 `#define VGA_TEXT_VID_BASE 0xb8000`

Definition at line 4 of file `vga.h`.

Referenced by `vga_clear()`, and `vga_scroll()`.

4.16.1.29 `#define VGA_WIDTH 80`

Definition at line 32 of file `vga.h`.

Referenced by `vga_clear()`, and `vga_scroll()`.

4.16.2 Typedef Documentation

4.16.2.1 `typedef unsigned int vga_pos_t`

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

4.16.3 Function Documentation

4.16.3.1 `void vga_clear (void)`

Definition at line 25 of file `vga.c`.

References `VGA_COLOR_ERASE`, `VGA_LINES`, `VGA_TEXT_VID_BASE`, and `VGA_WIDTH`.

Referenced by `vga_init()`.

```

25         {
26         unsigned char *buffer = (unsigned char *)VGA_TEXT_VID_BASE;
27         unsigned int idx = 0;
28
29         while( idx < (VGA_LINES * VGA_WIDTH * 2) )      {
30             buffer[idx++] = 0x20;
31             buffer[idx++] = VGA_COLOR_ERASE;
32         }
33 }

```

4.16.3.2 `vga_pos_t vga_get_cursor_pos (void)`

Definition at line 50 of file `vga.c`.

References `inb()`, `outb()`, `VGA_CRTC_ADDR`, and `VGA_CRTC_DATA`.

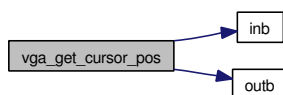
Referenced by `vga_print()`, `vga_printn()`, and `vga_putc()`.

```

50         {
51         unsigned char h, l;
52
53         outb(VGA_CRTC_ADDR, 0x0e);
54         h = inb(VGA_CRTC_DATA);
55         outb(VGA_CRTC_ADDR, 0x0f);
56         l = inb(VGA_CRTC_DATA);
57
58         return (h << 8) | l;
59 }

```

Here is the call graph for this function:



4.16.3.3 `void vga_init (void)`

Definition at line 7 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 `kinit()`.

```

7         {

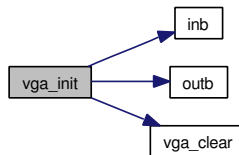
```

```

8      unsigned char data;
9
10     /* Set address select bit in a known state: CRTC regs at 0x3dx */
11     data = inb(VGA_MISC_OUT_RD);
12     data |= 1;
13     outb(VGA_MISC_OUT_WR, data);
14
15     /* Move cursor to line 0 col 0 */
16     outb(VGA_CRTC_ADDR, 0x0e);
17     outb(VGA_CRTC_DATA, 0x0);
18     outb(VGA_CRTC_ADDR, 0x0f);
19     outb(VGA_CRTC_DATA, 0x0);
20
21     /* Clear the screen */
22     vga_clear();
23 }

```

Here is the call graph for this function:



4.16.3.4 void vga_print (const char * message)

Definition at line 72 of file `vga.c`.

References `vga_get_cursor_pos()`, and `vga_set_cursor_pos()`.

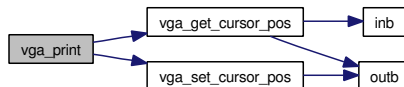
Referenced by `printk()`.

```

72     {
73     unsigned short int pos = vga_get_cursor_pos();
74     char c;
75
76     while( (c = *(message++)) ) {
77         pos = vga_raw_putc(c, pos);
78     }
79
80     vga_set_cursor_pos(pos);
81 }

```

Here is the call graph for this function:



4.16.3.5 void vga_printn (const char * *message*, unsigned int *n*)

Definition at line 83 of file vga.c.

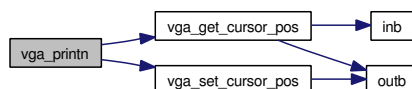
References vga_get_cursor_pos(), and vga_set_cursor_pos().

Referenced by printk().

```

83                                     {
84     vga_pos_t pos = vga_get_cursor_pos();
85     char c;
86
87     while(n) {
88         c = *(message++);
89         pos = vga_raw_putc(c, pos);
90         --n;
91     }
92
93     vga_set_cursor_pos(pos);
94 }
```

Here is the call graph for this function:



4.16.3.6 void vga_putc (char *c*)

Definition at line 96 of file vga.c.

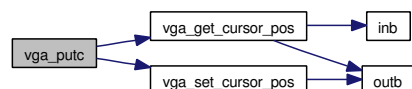
References vga_get_cursor_pos(), and vga_set_cursor_pos().

Referenced by print_hex_nibble(), print_unsigned_int(), and printk().

```

96     {
97     vga_pos_t pos = vga_get_cursor_pos();
98
99     pos = vga_raw_putc(c, pos);
100
101     vga_set_cursor_pos(pos);
102 }
```

Here is the call graph for this function:



4.16.3.7 void vga_scroll (void)

Definition at line 35 of file vga.c.

References VGA_COLOR_ERASE, VGA_LINES, VGA_TEXT_VID_BASE, and VGA_WIDTH.

```

35      {
36      unsigned char *di = (unsigned char *)VGA_TEXT_VID_BASE;
37      unsigned char *si = (unsigned char *) (VGA_TEXT_VID_BASE + 2 * VGA_WIDTH);
38      unsigned int idx;
39
40      for(idx = 0; idx < 2 * VGA_WIDTH * (VGA_LINES - 1); ++idx) {
41          *(di++) = *(si++);
42      }
43
44      for(idx = 0; idx < VGA_WIDTH; ++idx) {
45          *(di++) = 0x20;
46          *(di++) = VGA_COLOR_ERASE;
47      }
48 }

```

4.16.3.8 void vga_set_cursor_pos (vga_pos_t pos)

Definition at line 61 of file vga.c.

References outb(), VGA_CRTC_ADDR, and VGA_CRTC_DATA.

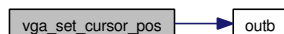
Referenced by vga_print(), vga_printn(), and vga_putc().

```

61      {
62      unsigned char h = pos >> 8;
63      unsigned char l = pos;
64
65      outb(VGA_CRTC_ADDR, 0x0e);
66      outb(VGA_CRTC_DATA, h);
67      outb(VGA_CRTC_ADDR, 0x0f);
68      outb(VGA_CRTC_DATA, l);
69 }

```

Here is the call graph for this function:

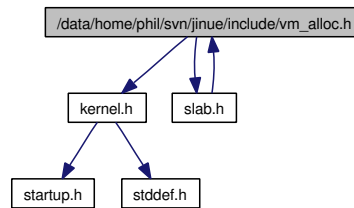


4.17 /data/home/phil/svn/jinue/include/vm_alloc.h File Reference

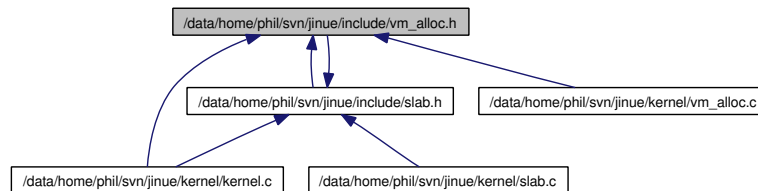
```
#include <kernel.h>
```

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

Include dependency graph for vm_alloc.h:



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



Data Structures

- struct **vm_link_t**
- struct **vm_alloc_t**

Typedefs

- typedef struct **vm_link_t** **vm_link_t**
- typedef struct **vm_alloc_t** **vm_alloc_t**

Functions

- **addr_t** **vm_valloc** (**vm_alloc_t** *pool)
- void **vm_vfree** (**vm_alloc_t** *pool, **addr_t** addr)

4.17 /data/home/phil/svn/jinue/include/vm_alloc.h File Reference

- void **vm_vfree_block** (vm_alloc_t *pool, addr_t addr, size_t size)
- addr_t **vm_alloc** (vm_alloc_t *pool, unsigned long flags)
- void **vm_free** (vm_alloc_t *pool, addr_t addr)

4.17.1 Typedef Documentation

4.17.1.1 typedef struct vm_alloc_t vm_alloc_t

Definition at line 21 of file vm_alloc.h.

4.17.1.2 typedef struct vm_link_t vm_link_t

Definition at line 13 of file vm_alloc.h.

4.17.2 Function Documentation

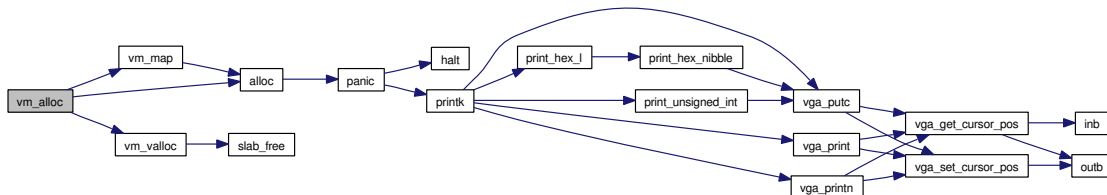
4.17.2.1 addr_t vm_alloc (vm_alloc_t * pool, unsigned long flags)

Definition at line 77 of file vm_alloc.c.

References `alloc()`, `PAGE_SIZE`, `vm_map()`, and `vm_valloc()`.

```
77                                     {
78     addr_t paddr, vaddr;
79
80     vaddr = vm_valloc(pool);
81     paddr = alloc(PAGE_SIZE);
82     vm_map(vaddr, paddr, flags);
83
84     return vaddr;
85 }
```

Here is the call graph for this function:



4.17.2.2 void vm_free (vm_alloc_t * pool, addr_t addr)

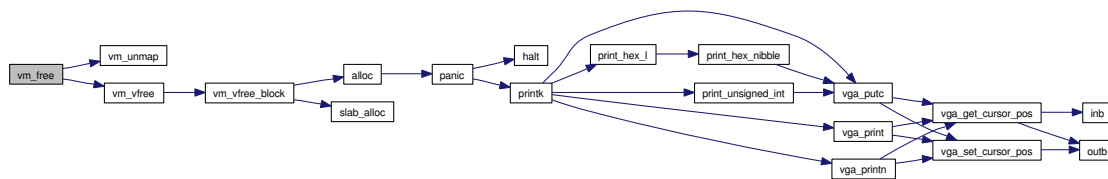
Definition at line 87 of file vm_alloc.c.

References vm_unmap(), and vm_vfree().

```

87                                     {
88     vm_unmap(addr);
89     vm_vfree(pool, addr);
90 }
```

Here is the call graph for this function:



4.17.2.3 addr_t vm_valloc (vm_alloc_t * pool)

Definition at line 6 of file vm_alloc.c.

References vm_link_t::addr, vm_alloc_t::cache, vm_alloc_t::head, vm_link_t::next, NULL, PAGE_SIZE, vm_link_t::size, and slab_free().

Referenced by vm_alloc().

```

6                                     {
7     addr_t addr;
8     vm_link_t *head;
9
10    head = pool->head;
11
12    if(head == (addr_t)NULL) {
13        return (addr_t)NULL;
14    }
15
16    addr = head->addr;
17    (head->size) -= PAGE_SIZE;;
18
19    if(head->size == 0) {
20        pool->head = head->next;
21        slab_free(pool->cache, head);
22        return addr;
23    }
24
25    (head->addr) += PAGE_SIZE;
```

4.17 /data/home/phil/svn/jinue/include/vm_alloc.h File Reference

```
26         return addr;
27 }
```

Here is the call graph for this function:



4.17.2.4 void vm_vfree (vm_alloc_t * pool, addr_t addr)

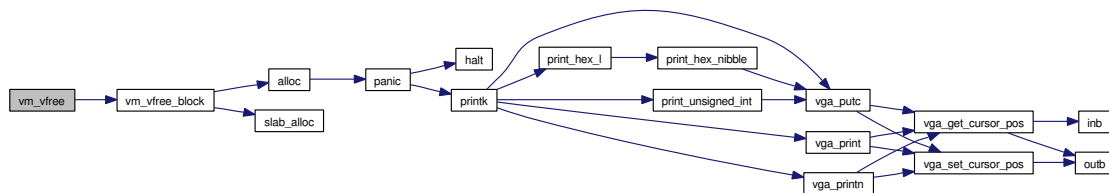
Definition at line 29 of file vm_alloc.c.

References PAGE_SIZE, and vm_vfree_block().

Referenced by vm_free().

```
29                                     {
30         vm_vfree_block(pool, addr, PAGE_SIZE);
31 }
```

Here is the call graph for this function:



4.17.2.5 void vm_vfree_block (vm_alloc_t * pool, addr_t addr, size_t size)

Definition at line 33 of file vm_alloc.c.

References vm_link_t::addr, alloc(), vm_alloc_t::cache, slab_cache_t::empty, vm_alloc_t::head, vm_link_t::next, NULL, PAGE_SIZE, slab_cache_t::partial, vm_link_t::size, slab_alloc(), and slab_cache_t::vm_allocator.

Referenced by vm_vfree().

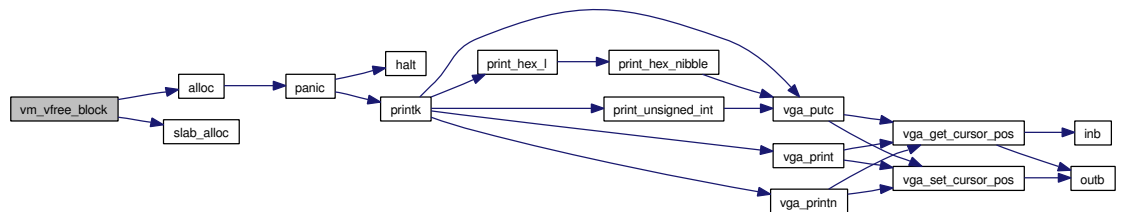
```
33                                     {
34         addr_t phys_page;
35         vm_link_t *link;
```

```

36
37      /* The virtual space allocator needs a slab cache from which to allocate
38         data structures for its free list. Also, each slab cache needs a
39         virtual space allocator to allocate slabs when needed.
40
41         There can be a mutual dependency between the virtual space allocator
42         and the slab cache. This is not a problem in general, but a special
43         bootstrapping procedure is needed for initialization of the virtual
44         space allocator in that case. The virtual space allocator will actually
45         "donate" a virtual page (backed by physical ram) to the cache for use as
46         a slab.
47
48         This case is handled here
49      */
50      if(pool->head == NULL) {
51          if(pool->cache->vm_allocator == pool) {
52              if(pool->cache->empty == NULL && pool->cache->partial == NULL) {
53                  phys_page = alloc(PAGE_SIZE);
54
55                  /*TODO: map page */
56
57                  size -= PAGE_SIZE;
58
59                  if(size == 0) {
60                      return;
61                  }
62
63                  addr += PAGE_SIZE;
64              }
65          }
66      }
67
68      link = (vm_link_t *)slab_alloc(pool->cache);
69      link->size = size;
70      link->addr = addr;
71
72      /* TODO: make this an atomic operation */
73      link->next = pool->head;
74      pool->head = link;
75 }

```

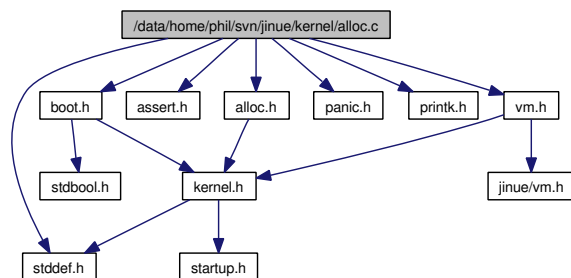
Here is the call graph for this function:



4.18 /data/home/phil/svn/jinue/kernel/alloc.c File Reference

```
#include <alloc.h>
#include <assert.h>
#include <boot.h>
#include <panic.h>
#include <printk.h>
#include <stddef.h>
#include <vm.h>
```

Include dependency graph for alloc.c:



Functions

- void **alloc_init** (void)
- **addr_t** **alloc** (size_t size)

4.18.1 Function Documentation

4.18.1.1 **addr_t** **alloc** (size_t *size*)

Definition at line 96 of file alloc.c.

References `assert`, `PAGE_BITS`, `PAGE_MASK`, `PAGE_SIZE`, and `panic()`.

Referenced by `vm_alloc()`, `vm_map()`, and `vm_vfree_block()`.

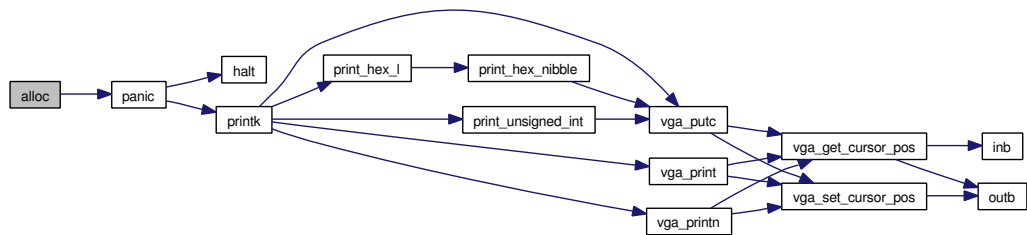
```
96                                     {
97     addr_t addr;
98     size_t pages;
99
```

```

100     pages = size >> PAGE_BITS;
101
102     if( (size & PAGE_MASK) != 0 ) {
103         ++pages;
104     }
105
106     if(_alloc_size < pages) {
107         panic("out of memory.");
108     }
109
110     addr = _alloc_addr;
111     _alloc_addr += pages * PAGE_SIZE;
112     _alloc_size -= pages;
113
114     /* returned address should be aligned on a page boundary */
115     assert( ((unsigned long)addr & PAGE_MASK) == 0 );
116
117     return addr;
118 }

```

Here is the call graph for this function:



4.18.1.2 void alloc_init (void)

Definition at line 12 of file alloc.c.

References e820_get_addr(), e820_get_size(), e820_get_type(), e820_is_available(), e820_is_valid(), e820_type_description(), kernel_start, kernel_top, PAGE_SIZE, panic(), printk(), and vm_alloc_t::size.

Referenced by kinit().

```

12     {
13         unsigned int idx;
14         unsigned int remainder;
15         bool avail;
16         size_t size;
17         e820_type_t type;
18         addr_t addr, fixed_addr, best_addr;
19         size_t fixed_size, best_size;

```



```

20
21     idx = 0;
22     best_size = 0;
23
24     printk("Dump of the BIOS memory map:\n");
25     printk(" address size      type\n");
26     while( e820_is_valid(idx) ) {
27         addr = e820_get_addr(idx);
28         size = e820_get_size(idx);
29         type = e820_get_type(idx);
30         avail = e820_is_available(idx);
31
32         ++idx;
33
34         printk("%c %x %x %s\n",
35             avail?'*':' ',
36             addr,
37             size,
38             e820_type_description(type) );
39
40         if( !avail ) {
41             continue;
42         }
43
44         fixed_addr = addr;
45         fixed_size = size;
46
47         /* is the region completely under the kernel ? */
48         if(addr + size > kernel_start) {
49             /* is the region completely above the kernel ? */
50             if(addr < kernel_top) {
51                 /* if the region touches the kernel, we take only
52                  * the part above the kernel, if there is one... */
53                 if(addr + size <= kernel_top) {
54                     /* ... and apparently, there is none */
55                     continue;
56                 }
57
58                 fixed_addr = kernel_top;
59                 fixed_size -= fixed_addr - addr;
60             }
61         }
62
63         /* we must make sure the starting address is aligned on a
64          * page boundary. The size will eventually be divided
65          * by the page size, and thus need not be aligned. */
66         remainder = (unsigned int)fixed_addr % PAGE_SIZE;
67         if(remainder != 0) {
68             remainder = PAGE_SIZE - remainder;
69             if(fixed_size < remainder) {
70                 continue;
71             }
72
73             fixed_addr += remainder;
74             fixed_size -= remainder;
75         }
76

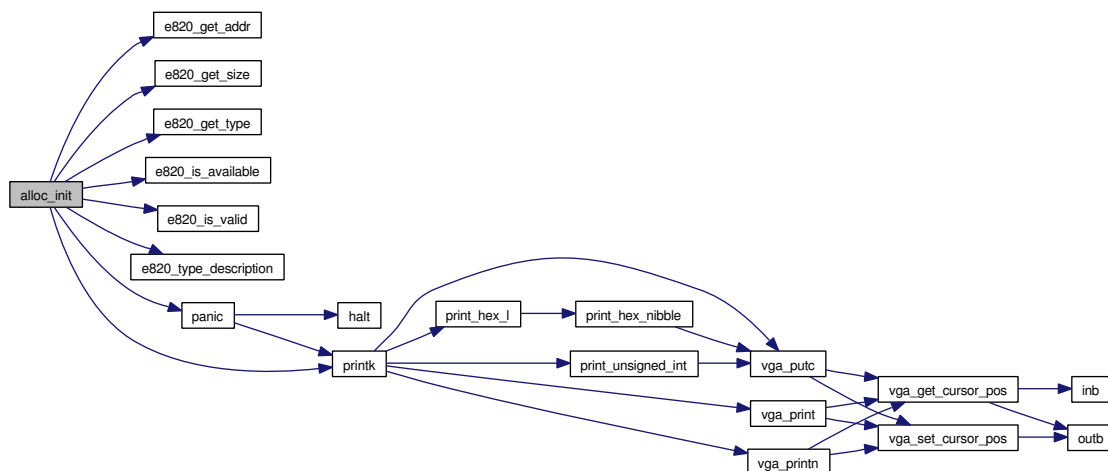
```

```

77         if(fixed_size > best_size) {
78             best_addr = fixed_addr;
79             best_size = fixed_size;
80         }
81     }
82
83     _alloc_addr = (addr_t)best_addr;
84     _alloc_size = best_size / PAGE_SIZE;
85
86     if(_alloc_size == 0) {
87         panic("no memory to allocate.");
88     }
89
90     printk("%u kilobytes (%u pages) available starting at %xh.\n",
91           _alloc_size * PAGE_SIZE / 1024,
92           _alloc_size,
93           _alloc_addr );
94 }

```

Here is the call graph for this function:

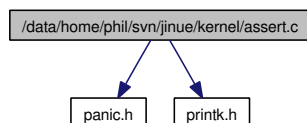


4.19 /data/home/phil/svn/jinue/kernel/assert.c File Reference

```
#include <panic.h>
```

```
#include <printk.h>
```

Include dependency graph for assert.c:



Functions

- void **__assert_failed** (const char **expr*, const char **file*, unsigned int *line*, const char **func*)

4.19.1 Function Documentation

4.19.1.1 void __assert_failed (const char * *expr*, const char * *file*, unsigned int *line*, const char * *func*)

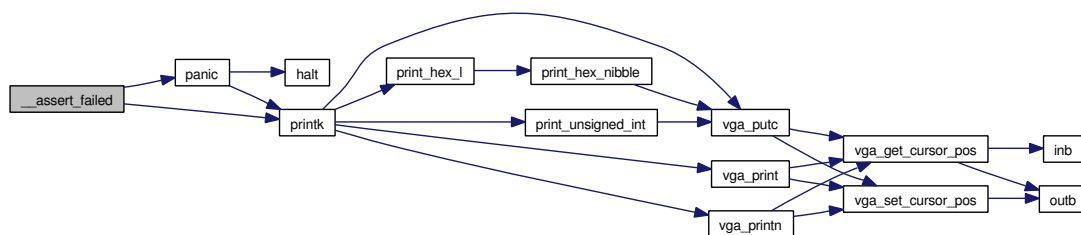
Definition at line 5 of file assert.c.

References `panic()`, and `printk()`.

```

9             {
10
11         printk(
12             "ASSERTION FAILED [%s]: %s at line %u in function %s.\n",
13             expr, file, line, func );
14
15         panic("Assertion failed.");
16 }
```

Here is the call graph for this function:



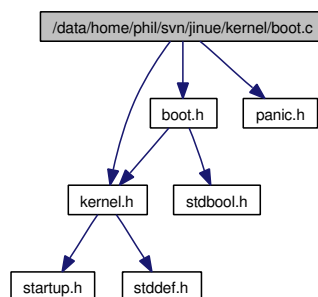
4.20 /data/home/phil/svn/jinue/kernel/boot.c File Reference

```
#include <boot.h>
```

```
#include <kernel.h>
```

```
#include <panic.h>
```

Include dependency graph for boot.c:



Functions

- `addr_t e820_get_addr` (unsigned int idx)
- `size_t e820_get_size` (unsigned int idx)
- `e820_type_t e820_get_type` (unsigned int idx)
- `bool e820_is_valid` (unsigned int idx)
- `bool e820_is_available` (unsigned int idx)
- `const char * e820_type_description` (e820_type_t type)
- `boot_t * get_boot_data` (void)

Variables

- `e820_t * e820_map`
- `addr_t boot_setup_addr`

4.20.1 Function Documentation

4.20.1.1 `addr_t e820_get_addr` (unsigned int *idx*)

Definition at line 8 of file `boot.c`.

Referenced by `alloc_init()`.

```
8           {
9   return (addr_t)(unsigned long)e820_map[idx].addr;
10 }
```

4.20.1.2 size_t e820_get_size (unsigned int *idx*)

Definition at line 12 of file boot.c.

References e820_t::size.

Referenced by alloc_init().

```
12           {
13   return (size_t)e820_map[idx].size;
14 }
```

4.20.1.3 e820_type_t e820_get_type (unsigned int *idx*)

Definition at line 16 of file boot.c.

References e820_t::type.

Referenced by alloc_init().

```
16           {
17   return e820_map[idx].type;
18 }
```

4.20.1.4 bool e820_is_available (unsigned int *idx*)

Definition at line 24 of file boot.c.

References E820_RAM.

Referenced by alloc_init().

```
24           {
25   return (e820_map[idx].type == E820_RAM);
26 }
```

4.20.1.5 bool e820_is_valid (unsigned int *idx*)

Definition at line 20 of file boot.c.

References vm_alloc_t::size.

Referenced by alloc_init().

```
20          {
21      return (e820_map[idx].size != 0);
22 }
```

4.20.1.6 const char* e820_type_description (e820_type_t *type*)

Definition at line 28 of file boot.c.

References E820_ACPI, E820_RAM, and E820_RESERVED.

Referenced by alloc_init().

```
28          {
29      switch(type) {
30
31      case E820_RAM:
32          return "available";
33
34      case E820_RESERVED:
35          return "unavailable/reserved";
36
37      case E820_ACPI:
38          return "unavailable/acpi";
39
40      default:
41          return "unavailable/other";
42      }
43 }
```

4.20.1.7 boot_t* get_boot_data (void)

Definition at line 45 of file boot.c.

References BOOT_MAGIC, boot_setup_addr, BOOT_SIGNATURE, boot_t::magic, panic(), and boot_t::signature.

Referenced by kinit().

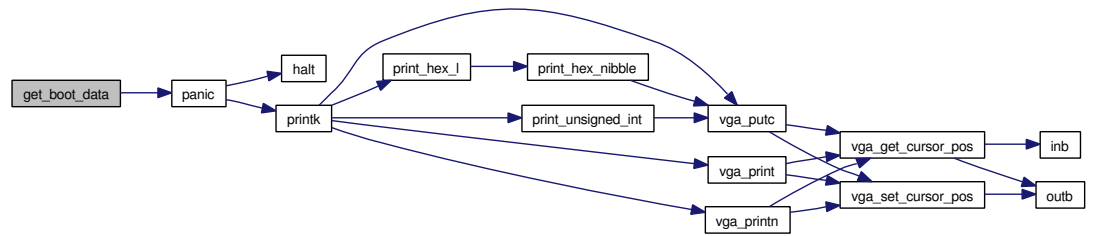
```
45          {
46      boot_t *boot;
47
48      boot = (boot_t *) ( boot_setup_addr - sizeof(boot_t) );
49
50      if(boot->signature != BOOT_SIGNATURE) {
51          panic("bad boot sector signature.");
52      }
53
54      if(boot->magic != BOOT_MAGIC) {
55          panic("bad boot sector magic.");
56      }
57 }
```

```

58         return boot;
59     }

```

Here is the call graph for this function:



4.20.2 Variable Documentation

4.20.2.1 addr_t boot_setup_addr

Definition at line 6 of file boot.c.

Referenced by get_boot_data().

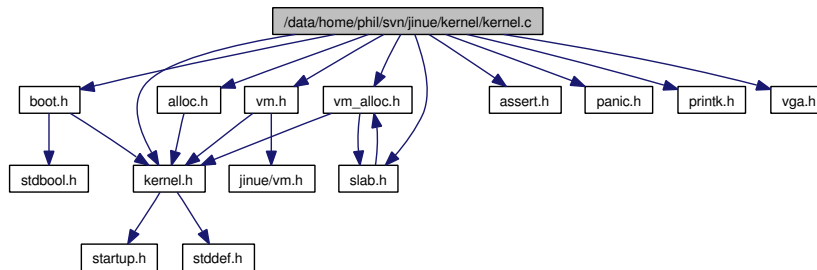
4.20.2.2 e820_t* e820_map

Definition at line 5 of file boot.c.

4.21 /data/home/phil/svn/jinue/kernel/kernel.c File Reference

```
#include <alloc.h>
#include <assert.h>
#include <boot.h>
#include <kernel.h>
#include <panic.h>
#include <printk.h>
#include <vga.h>
#include <vm.h>
#include <vm_alloc.h>
#include <slab.h>
```

Include dependency graph for kernel.c:



Functions

- void **kernel** (void)
- void **kinit** (void)
- void **idle** (void)

Variables

- **addr_t** **kernel_top**
- **size_t** **kernel_size**

4.21.1 Function Documentation

4.21.1.1 void idle (void)

Definition at line 52 of file kernel.c.

Referenced by kernel().

```

52         {
53     while(1) {}
54 }
```

4.21.1.2 void kernel (void)

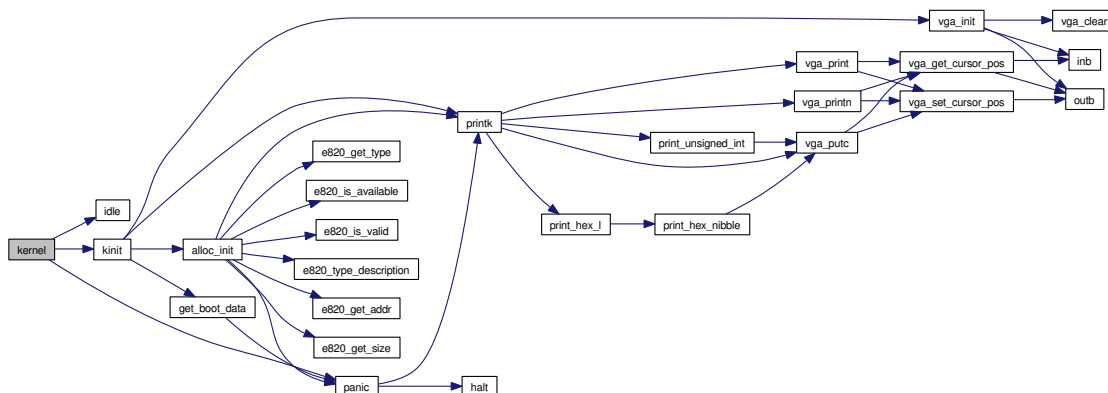
Definition at line 16 of file kernel.c.

References idle(), kinit(), and panic().

```

16         {
17     kinit();
18     idle();
19
20     panic("idle() returned.");
21 }
```

Here is the call graph for this function:



4.21.1.3 void kinit (void)

Definition at line 23 of file kernel.c.

References `alloc_init()`, `assert`, `get_boot_data()`, `kernel_size`, `kernel_start`, `kernel_top`, `PAGE_SIZE`, `printk()`, `boot_t::sysize`, and `vga_init()`.

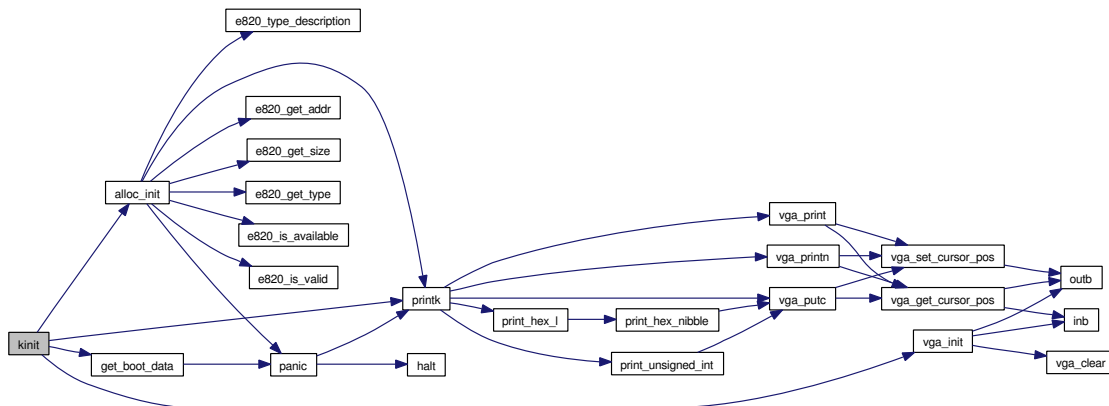
Referenced by `kernel()`.

```

23     {
24     boot_t *boot;
25     unsigned int remainder;
26
27     /* say hello */
28     vga_init();
29     printk("Kernel started.\n");
30
31     /* we assume the kernel starts on a page boundary */
32     assert((unsigned int)kernel_start % PAGE_SIZE == 0);
33
34     /* find out kernel size and set kernel_top
35      * (top of kernel, aligned to page boundary) */
36     boot = get_boot_data();
37
38     kernel_size = boot->sysize * 16;
39     remainder   = kernel_size % PAGE_SIZE;
40
41     printk("Kernel size is %u (+%u) bytes.\n", kernel_size, PAGE_SIZE - remainder);
42
43     if(remainder != 0) {
44         kernel_size += PAGE_SIZE - remainder;
45     }
46     kernel_top = kernel_start + kernel_size;
47
48     /* initialize allocator */
49     alloc_init();
50 }

```

Here is the call graph for this function:



4.21.2 Variable Documentation

4.21.2.1 `size_t kernel_size`

Definition at line 14 of file `kernel.c`.

Referenced by `kinit()`.

4.21.2.2 `addr_t kernel_top`

Definition at line 13 of file `kernel.c`.

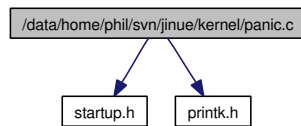
Referenced by `alloc_init()`, and `kinit()`.

4.22 /data/home/phil/svn/jinue/kernel/panic.c File Reference

```
#include <startup.h>
```

```
#include <printk.h>
```

Include dependency graph for panic.c:



Functions

- void **panic** (const char *message)

4.22.1 Function Documentation

4.22.1.1 void panic (const char * *message*)

Definition at line 4 of file panic.c.

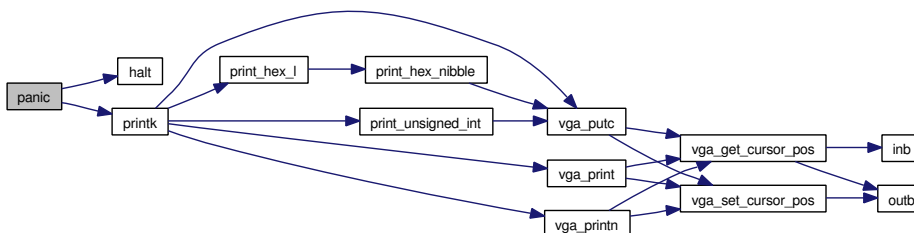
References `halt()`, and `printk()`.

Referenced by `__assert_failed()`, `alloc()`, `alloc_init()`, `get_boot_data()`, and `kernel()`.

```

4      {
5          printk("KERNEL PANIC: %s\n", message);
6          halt();
7      }
  
```

Here is the call graph for this function:



4.23 /data/home/phil/svn/jinue/kernel/printk.c

File Reference

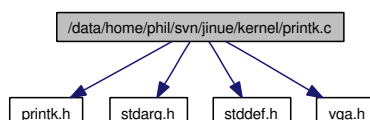
```
#include <printk.h>
```

```
#include <stdarg.h>
```

```
#include <stddef.h>
```

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

Include dependency graph for printk.c:



Functions

- void **printk** (const char *format,...)
- void **print__unsigned__int** (unsigned int n)
- void **print__hex__nibble** (unsigned char byte)
- void **print__hex__b** (unsigned char byte)
- void **print__hex__w** (unsigned short word)
- void **print__hex__l** (unsigned long dword)
- void **print__hex__q** (unsigned long long qword)

4.23.1 Function Documentation

4.23.1.1 void print__hex__b (unsigned char *byte*)

Definition at line 105 of file printk.c.

References `print__hex__nibble()`.

```
105                                     {
106     print_hex_nibble( (char)byte );
107     print_hex_nibble( (char)(byte>>4) );
108 }
```

Here is the call graph for this function:



4.23.1.2 void print_hex_l (unsigned long *dword*)

Definition at line 118 of file printk.c.

References `print_hex_nibble()`.

Referenced by `printk()`.

```

118                                     {
119     int off;
120
121     for(off=32-4; off>=0; off-=4) {
122         print_hex_nibble( (char)(dword>>off) );
123     }
124 }
```

Here is the call graph for this function:



4.23.1.3 void print_hex_nibble (unsigned char *byte*)

Definition at line 91 of file printk.c.

References `vga_putc()`.

Referenced by `print_hex_b()`, `print_hex_l()`, `print_hex_q()`, and `print_hex_w()`.

```

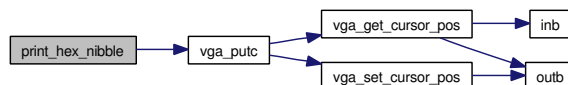
91                                     {
92     char c;
93
94     c = byte & 0xf;
95     if(c < 10) {
96         c += '0';
97     }
98     else {
```

```

99             c+= ('a' - 10);
100         }
101
102         vga_putc(c);
103     }

```

Here is the call graph for this function:



4.23.1.4 void print_hex_q (unsigned long long *qword*)

Definition at line 126 of file printk.c.

References `print_hex_nibble()`.

```

126                                     {
127         int off;
128
129         for(off=64-4; off>=0; off-=4) {
130             print_hex_nibble( (char)(qword>>off) );
131         }
132     }

```

Here is the call graph for this function:



4.23.1.5 void print_hex_w (unsigned short *word*)

Definition at line 110 of file printk.c.

References `print_hex_nibble()`.

```

110                                     {
111         int off;
112
113         for(off=16-4; off>=0; off-=4) {
114             print_hex_nibble( (char)(word>>off) );
115         }
116     }

```


Here is the call graph for this function:



4.23.1.6 void print_unsigned_int (unsigned int *n*)

Definition at line 67 of file printk.c.

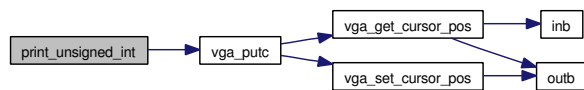
References `vga_putc()`.

Referenced by `printk()`.

```

67                                     {
68     unsigned int flag = 0;
69     unsigned int pwr;
70     unsigned int digit;
71     char c;
72
73     if(n == 0) {
74         vga_putc('0');
75         return;
76     }
77
78     for(pwr = 1000 * 1000 * 1000; pwr > 0; pwr /= 10) {
79         digit = n / pwr;
80
81         if(digit != 0 || flag) {
82             c = (char)digit + '0';
83             vga_putc(c);
84
85             flag = 1;
86             n -= digit * pwr;
87         }
88     }
89 }
  
```

Here is the call graph for this function:



4.23.1.7 void printk (const char * *format*, ...)

Definition at line 6 of file printk.c.

References `print_hex_l()`, `print_unsigned_int()`, `vm_alloc_t::size`, `va_arg`, `va_end`, `va_start`, `vga_print()`, `vga_printn()`, and `vga_putc()`.

Referenced by `__assert_failed()`, `alloc_init()`, `kinit()`, and `panic()`.

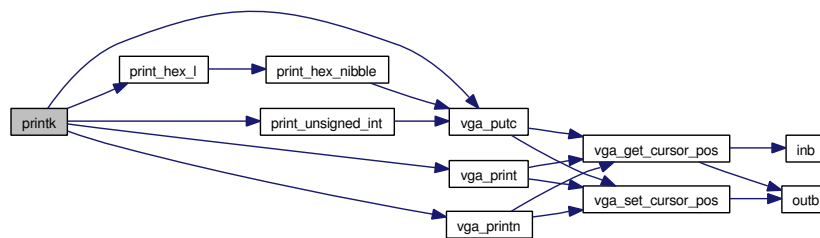
```

6                                     {
7     va_list ap;
8     const char *idx, *anchor;
9     ptrdiff_t size;
10
11     va_start(ap, format);
12
13     idx = format;
14
15     while(1) {
16         anchor = idx;
17
18         while( *idx != 0 && *idx != '%' ) {
19             ++idx;
20         }
21
22         size = idx - anchor;
23
24         if(size > 0) {
25             vga_printn(anchor, size);
26         }
27
28         if(*idx == 0 || *(idx+1) == 0) {
29             break;
30         }
31
32         ++idx;
33
34         switch( *idx ) {
35             case '%':
36                 vga_putc('%');
37                 break;
38
39             case 'c':
40                 /* promotion, promotion */
41                 vga_putc( (char)va_arg(ap, int) );
42                 break;
43
44             case 's':
45                 vga_print( va_arg(ap, const char *) );
46                 break;
47
48             case 'u':
49                 print_unsigned_int( va_arg(ap, unsigned int) );
50                 break;
51
52             case 'x':
53                 print_hex_l( va_arg(ap, unsigned long) );
54                 break;
55
56             default:
57                 va_end(ap);

```

```
58             return;
59         }
60
61         ++idx;
62     }
63
64     va_end(ap);
65 }
```

Here is the call graph for this function:

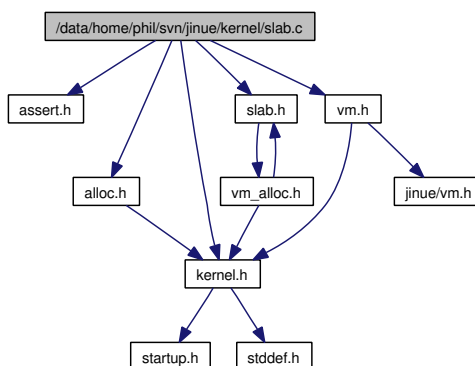


4.24 /data/home/phil/svn/jinue/kernel/slab.c

File Reference

```
#include <assert.h>
#include <alloc.h>
#include <kernel.h>
#include <slab.h>
#include <vm.h>
```

Include dependency graph for slab.c:



Functions

- `addr_t slab_alloc (slab_cache_t *cache)`
- `void slab_free (slab_cache_t *cache, addr_t obj)`
- `void slab_prepare_page (slab_cache_t *cache, addr_t page)`

4.24.1 Function Documentation

4.24.1.1 `addr_t slab_alloc (slab_cache_t * cache)`

Definition at line 7 of file slab.c.

References NULL.

Referenced by `vm_vfree_block()`.

```
7                                     {
8     return NULL;
9 }
```

4.24.1.2 void slab_free (slab_cache_t * cache, addr_t obj)

Definition at line 11 of file slab.c.

Referenced by vm_valloc().

```
11                                     {
12 }
```

4.24.1.3 void slab_prepare_page (slab_cache_t * cache, addr_t page)

Definition at line 14 of file slab.c.

References assert, slab_header_t::available, slab_cache_t::empty, slab_header_t::free_list, slab_header_t::next, NULL, slab_cache_t::obj_size, PAGE_OFFSET_OF, slab_cache_t::per_slab, and slab_header_t::prev.

```
14                                     {
15     unsigned int cx;
16     size_t obj_size;
17     count_t per_slab;
18     slab_header_t *slab;
19     addr_t *ptr;
20     addr_t next;
21
22     /* we assume "page" is the starting address of a page */
23     assert( PAGE_OFFSET_OF(page) );
24
25     obj_size = cache->obj_size;
26     per_slab = cache->per_slab;
27
28     slab = (slab_header_t *)page;
29     slab->available = per_slab;
30     slab->free_list = page + sizeof(slab_header_t);
31
32     /* create free list */
33     ptr = (addr_t *)slab->free_list;
34
35     for(cx = 0; cx < per_slab - 1; ++cx) {
36         next = ptr + obj_size;
37         *ptr = next;
38         ptr = (addr_t *)next;
39     }
40
41     *ptr = NULL;
42
43     /* insert in list of empty slabs of cache */
44     slab->prev = NULL;
45     slab->next = cache->empty;
46     cache->empty = slab;
47 }
```

4.25 /data/home/phil/svn/jinue/kernel/vga.c

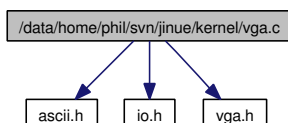
File Reference

```
#include <ascii.h>
```

```
#include <io.h>
```

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

Include dependency graph for vga.c:



Functions

- void **vga_init** (void)
- 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)
- void **vga_printn** (const char *message, unsigned int n)
- void **vga_putc** (char c)

4.25.1 Function Documentation

4.25.1.1 void vga_clear (void)

Definition at line 25 of file vga.c.

References `VGA_COLOR_ERASE`, `VGA_LINES`, `VGA_TEXT_VID_BASE`, and `VGA_WIDTH`.

Referenced by `vga_init()`.

```

25      {
26          unsigned char *buffer = (unsigned char *)VGA_TEXT_VID_BASE;
27          unsigned int idx = 0;
28
29          while( idx < (VGA_LINES * VGA_WIDTH * 2) )      {
30              buffer[idx++] = 0x20;
31              buffer[idx++] = VGA_COLOR_ERASE;
32          }
33  }
```

4.25.1.2 vga_pos_t vga_get_cursor_pos (void)

Definition at line 50 of file vga.c.

References `inb()`, `outb()`, `VGA_CRTC_ADDR`, and `VGA_CRTC_DATA`.

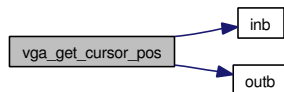
Referenced by `vga_print()`, `vga_printn()`, and `vga_putc()`.

```

50      {
51      unsigned char h, l;
52
53      outb(VGA_CRTC_ADDR, 0x0e);
54      h = inb(VGA_CRTC_DATA);
55      outb(VGA_CRTC_ADDR, 0x0f);
56      l = inb(VGA_CRTC_DATA);
57
58      return (h << 8) | l;
59 }

```

Here is the call graph for this function:



4.25.1.3 void vga_init (void)

Definition at line 7 of file vga.c.

References `inb()`, `outb()`, `vga_clear()`, `VGA_CRTC_ADDR`, `VGA_CRTC_DATA`, `VGA_MISC_OUT_RD`, and `VGA_MISC_OUT_WR`.

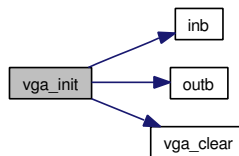
Referenced by `kinit()`.

```

7      {
8      unsigned char data;
9
10     /* Set address select bit in a known state: CRTC regs at 0x3dx */
11     data = inb(VGA_MISC_OUT_RD);
12     data |= 1;
13     outb(VGA_MISC_OUT_WR, data);
14
15     /* Move cursor to line 0 col 0 */
16     outb(VGA_CRTC_ADDR, 0x0e);
17     outb(VGA_CRTC_DATA, 0x0);
18     outb(VGA_CRTC_ADDR, 0x0f);
19     outb(VGA_CRTC_DATA, 0x0);
20
21     /* Clear the screen */
22     vga_clear();
23 }

```

Here is the call graph for this function:



4.25.1.4 void vga_print (const char * *message*)

Definition at line 72 of file `vga.c`.

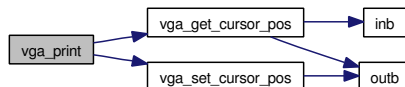
References `vga_get_cursor_pos()`, and `vga_set_cursor_pos()`.

Referenced by `printk()`.

```

72      {
73      unsigned short int pos = vga_get_cursor_pos();
74      char c;
75
76      while( (c = *(message++)) ) {
77          pos = vga_raw_putc(c, pos);
78      }
79
80      vga_set_cursor_pos(pos);
81  }
```

Here is the call graph for this function:



4.25.1.5 void vga_printn (const char * *message*, unsigned int *n*)

Definition at line 83 of file `vga.c`.

References `vga_get_cursor_pos()`, and `vga_set_cursor_pos()`.

Referenced by `printk()`.

```

83      {
84      vga_pos_t pos = vga_get_cursor_pos();
85      char c;
86  }
```

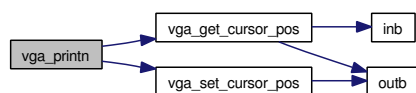


```

87     while(n) {
88         c = *(message++);
89         pos = vga_raw_putc(c, pos);
90         --n;
91     }
92
93     vga_set_cursor_pos(pos);
94 }

```

Here is the call graph for this function:



4.25.1.6 void vga_putc (char c)

Definition at line 96 of file vga.c.

References vga_get_cursor_pos(), and vga_set_cursor_pos().

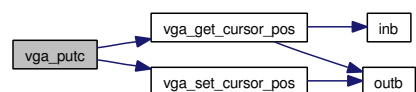
Referenced by print_hex_nibble(), print_unsigned_int(), and printk().

```

96     {
97         vga_pos_t pos = vga_get_cursor_pos();
98
99         pos = vga_raw_putc(c, pos);
100
101         vga_set_cursor_pos(pos);
102 }

```

Here is the call graph for this function:



4.25.1.7 void vga_scroll (void)

Definition at line 35 of file vga.c.

References VGA_COLOR_ERASE, VGA_LINES, VGA_TEXT_VID_BASE, and VGA_WIDTH.

```

35         {
36         unsigned char *di = (unsigned char *)VGA_TEXT_VID_BASE;
37         unsigned char *si = (unsigned char *) (VGA_TEXT_VID_BASE + 2 * VGA_WIDTH);
38         unsigned int idx;
39
40         for(idx = 0; idx < 2 * VGA_WIDTH * (VGA_LINES - 1); ++idx) {
41             *(di++) = *(si++);
42         }
43
44         for(idx = 0; idx < VGA_WIDTH; ++idx) {
45             *(di++) = 0x20;
46             *(di++) = VGA_COLOR_ERASE;
47         }
48 }

```

4.25.1.8 void vga_set_cursor_pos (vga_pos_t pos)

Definition at line 61 of file vga.c.

References `outb()`, `VGA_CRTC_ADDR`, and `VGA_CRTC_DATA`.

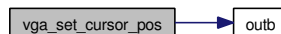
Referenced by `vga_print()`, `vga_printn()`, and `vga_putc()`.

```

61         {
62         unsigned char h = pos >> 8;
63         unsigned char l = pos;
64
65         outb(VGA_CRTC_ADDR, 0x0e);
66         outb(VGA_CRTC_DATA, h);
67         outb(VGA_CRTC_ADDR, 0x0f);
68         outb(VGA_CRTC_DATA, l);
69 }

```

Here is the call graph for this function:



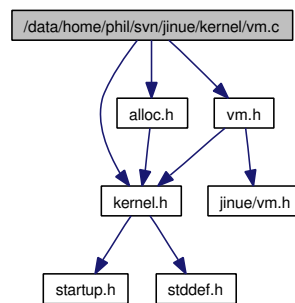
4.26 /data/home/phil/svn/jinue/kernel/vm.c File Reference

```
#include <kernel.h>
```

```
#include <alloc.h>
```

```
#include <vm.h>
```

Include dependency graph for vm.c:



Functions

- void **vm_map** (**addr_t** *vaddr*, **addr_t** *paddr*, unsigned long *flags*)
- void **vm_unmap** (**addr_t** *addr*)

4.26.1 Function Documentation

4.26.1.1 void vm_map (addr_t *vaddr*, addr_t *paddr*, unsigned long *flags*)

Definition at line 5 of file vm.c.

References `alloc()`, `PAGE_SIZE`, `PAGE_TABLE_ENTRIES`, `PAGE_TABLE_OF`, `PAGE_TABLE_PTE_OF`, `PDE_OF`, `PTE_OF`, and `VM_FLAG_PRESENT`.

Referenced by `vm_alloc()`.

```

5                                     {
6     pte_t *pte;
7     addr_t page_table;
8     int idx;
9
10    pte = PDE_OF(vaddr);

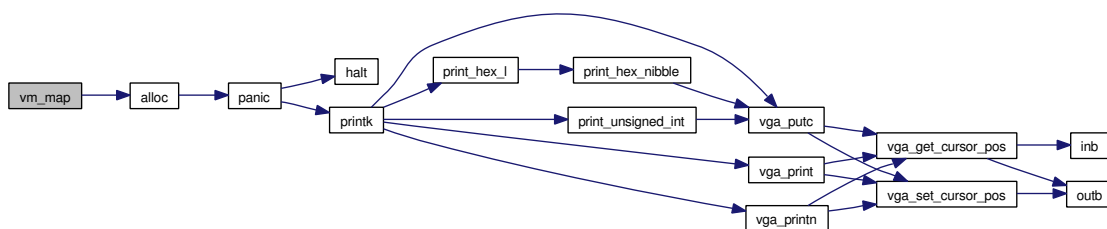
```

```

11
12      /* check if page table must be created */
13      if( !(*pte & VM_FLAG_PRESENT) ) {
14          /* allocate a page for page table */
15          page_table = alloc(PAGE_SIZE);
16
17          /* link to page table from page directory */
18          *pte = (pte_t)page_table | VM_FLAG_PRESENT;
19
20          /* map page table in the region of memory reserved for that purpose */
21          pte = PAGE_TABLE_PTE_OF(vaddr);
22          *pte = (pte_t)page_table | VM_FLAG_PRESENT;
23
24          /* obtain virtual address of new page table */
25          pte = PAGE_TABLE_OF(vaddr);
26
27          /* zero content of page table */
28          for(idx = 0; idx < PAGE_TABLE_ENTRIES; ++idx) {
29              pte[idx] = 0;
30          }
31      }
32
33      /* perform the actual mapping */
34      pte = PTE_OF(vaddr);
35      *pte = (pte_t)paddr | VM_FLAG_PRESENT;
36 }

```

Here is the call graph for this function:



4.26.1.2 void vm_unmap(addr_t addr)

Definition at line 38 of file vm.c.

References PTE_OF.

Referenced by vm_free().

```

38      {
39          pte_t *pte;
40
41          pte = PTE_OF(addr);

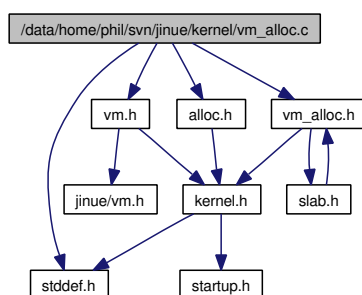
```

```
42         *pte = 0;  
43     }
```

4.27 /data/home/phil/svn/jinue/kernel/vm_alloc.c File Reference

```
#include <stddef.h>
#include <alloc.h>
#include <vm.h>
#include <vm_alloc.h>
```

Include dependency graph for vm_alloc.c:



Functions

- **addr_t vm_valloc** (vm_alloc_t *pool)
- **void vm_vfree** (vm_alloc_t *pool, addr_t addr)
- **void vm_vfree_block** (vm_alloc_t *pool, addr_t addr, size_t size)
- **addr_t vm_alloc** (vm_alloc_t *pool, unsigned long flags)
- **void vm_free** (vm_alloc_t *pool, addr_t addr)

4.27.1 Function Documentation

4.27.1.1 addr_t vm_alloc (vm_alloc_t * pool, unsigned long flags)

Definition at line 77 of file vm_alloc.c.

References alloc(), PAGE_SIZE, vm_map(), and vm_valloc().

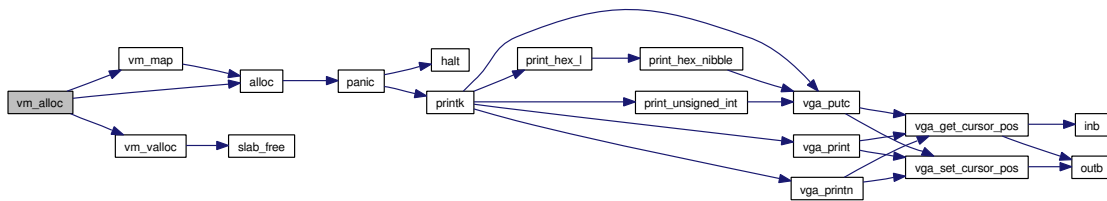
```

77                                     {
78     addr_t paddr, vaddr;
79
80     vaddr = vm_valloc(pool);
```

4.27 /data/home/phil/svn/jinue/kernel/vm_alloc.c File Reference 97

```
81     paddr = alloc(PAGE_SIZE);
82     vm_map(vaddr, paddr, flags);
83
84     return vaddr;
85 }
```

Here is the call graph for this function:



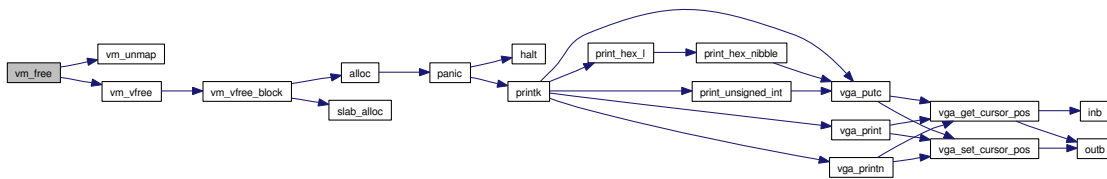
4.27.1.2 void vm_free (vm_alloc_t * pool, addr_t addr)

Definition at line 87 of file vm_alloc.c.

References vm_unmap(), and vm_vfree().

```
87     {
88     vm_unmap(addr);
89     vm_vfree(pool, addr);
90 }
```

Here is the call graph for this function:



4.27.1.3 addr_t vm_valloc (vm_alloc_t * pool)

Definition at line 6 of file vm_alloc.c.

References vm_link_t::addr, vm_alloc_t::cache, vm_alloc_t::head, vm_link_t::next, NULL, PAGE_SIZE, vm_link_t::size, and slab_free().

Referenced by vm_alloc().

```

6          {
7      addr_t addr;
8      vm_link_t *head;
9
10     head = pool->head;
11
12     if(head == (addr_t)NULL) {
13         return (addr_t)NULL;
14     }
15
16     addr = head->addr;
17     (head->size) -= PAGE_SIZE;;
18
19     if(head->size == 0) {
20         pool->head = head->next;
21         slab_free(pool->cache, head);
22         return addr;
23     }
24
25     (head->addr) += PAGE_SIZE;
26     return addr;
27 }

```

Here is the call graph for this function:



4.27.1.4 void vm_vfree (vm_alloc_t * pool, addr_t addr)

Definition at line 29 of file vm_alloc.c.

References PAGE_SIZE, and vm_vfree_block().

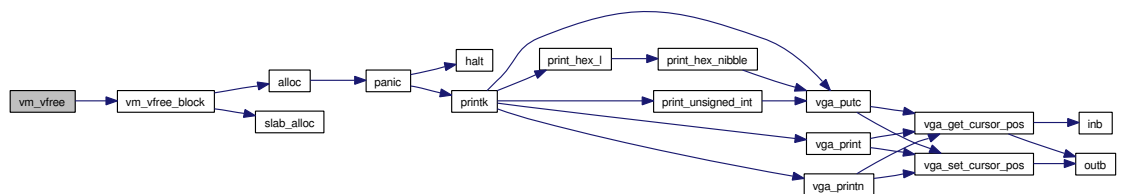
Referenced by vm_free().

```

29          {
30      vm_vfree_block(pool, addr, PAGE_SIZE);
31 }

```

Here is the call graph for this function:



4.27.1.5 void vm_vfree_block (vm_alloc_t * pool, addr_t addr, size_t size)

Definition at line 33 of file vm_alloc.c.

References vm_link_t::addr, alloc(), vm_alloc_t::cache, slab_cache_t::empty, vm_alloc_t::head, vm_link_t::next, NULL, PAGE_SIZE, slab_cache_t::partial, vm_link_t::size, slab_alloc(), and slab_cache_t::vm_allocator.

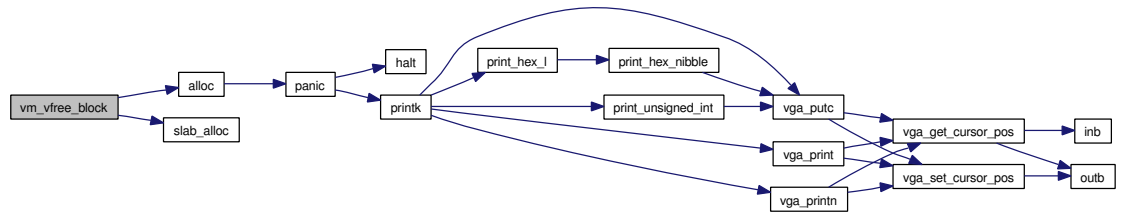
Referenced by vm_vfree().

```

33                                     {
34         addr_t phys_page;
35         vm_link_t *link;
36
37         /* The virtual space allocator needs a slab cache from which to allocate
38            data structures for its free list. Also, each slab cache needs a
39            virtual space allocator to allocate slabs when needed.
40
41            There can be a mutual dependency between the virtual space allocator
42            and the slab cache. This is not a problem in general, but a special
43            bootstrapping procedure is needed for initialization of the virtual
44            space allocator in that case. The virtual space allocator will actually
45            "donate" a virtual page (backed by physical ram) to the cache for use as
46            a slab.
47
48            This case is handled here
49        */
50         if(pool->head == NULL) {
51             if(pool->cache->vm_allocator == pool) {
52                 if(pool->cache->empty == NULL && pool->cache->partial == NULL) {
53                     phys_page = alloc(PAGE_SIZE);
54
55                     /*TODO: map page */
56
57                     size -= PAGE_SIZE;
58
59                     if(size == 0) {
60                         return;
61                     }
62
63                     addr += PAGE_SIZE;
64                 }
65             }
66         }
67
68         link = (vm_link_t *)slab_alloc(pool->cache);
69         link->size = size;
70         link->addr = addr;
71
72         /* TODO: make this an atomic operation */
73         link->next = pool->head;
74         pool->head = link;
75     }

```

Here is the call graph for this function:



Index

/data/home/phil/svn/jinue/include/alloc.h, 79
 15 /data/home/phil/svn/jinue/kernel/boot.c,
 /data/home/phil/svn/jinue/include/ascii.h, 81
 19 /data/home/phil/svn/jinue/kernel/kernel.c,
 /data/home/phil/svn/jinue/include/assert.h, 85
 20 /data/home/phil/svn/jinue/kernel/panic.c,
 /data/home/phil/svn/jinue/include/boot.h, 89
 22 /data/home/phil/svn/jinue/kernel/printk.c,
 /data/home/phil/svn/jinue/include/io.h, 90
 27 /data/home/phil/svn/jinue/kernel/slab.c,
 /data/home/phil/svn/jinue/include/jinue/vm.h, 96
 28 /data/home/phil/svn/jinue/kernel/vga.c,
 /data/home/phil/svn/jinue/include/kernel.h, 98
 41 /data/home/phil/svn/jinue/kernel/vm.c,
 /data/home/phil/svn/jinue/include/panic.h, 103
 45 /data/home/phil/svn/jinue/kernel/vm_
 /data/home/phil/svn/jinue/include/printk.h, alloc.c, 106
 46 __assert_failed
 /data/home/phil/svn/jinue/include/slab.h, assert.c, 79
 52 assert.h, 20
 /data/home/phil/svn/jinue/include/startup.h, bool_true_false_are_defined
 55 stdbool.h, 58
 /data/home/phil/svn/jinue/include/stdarg.h,
 56 addr
 /data/home/phil/svn/jinue/include/stdbool.h, 820_t, 7
 58 vm_link_t, 14
 /data/home/phil/svn/jinue/include/stddef.h, t
 59 kernel.h, 42
 /data/home/phil/svn/jinue/include/vga.h,
 61 alloc
 /data/home/phil/svn/jinue/include/vm.h, alloc.c, 75
 31 alloc.h, 16
 /data/home/phil/svn/jinue/include/vm_
 alloc.h, 70 alloc_init, 76
 /data/home/phil/svn/jinue/kernel/alloc.h
 75 alloc, 16
 /data/home/phil/svn/jinue/kernel/assert.c, alloc_init, 16

- alloc_init
 - alloc.c, 76
 - alloc.h, 16
- ascii.h
 - CHAR_BS, 19
 - CHAR_CR, 19
 - CHAR_HT, 19
 - CHAR_LF, 19
- assert
 - assert.h, 20
- assert.c
 - __assert_failed, 79
- assert.h
 - __assert_failed, 20
 - assert, 20
- available
 - slab_header_t, 10
- bool
 - stdbool.h, 58
- boot.c
 - boot_setup_addr, 84
 - e820_get_addr, 81
 - e820_get_size, 82
 - e820_get_type, 82
 - e820_is_available, 82
 - e820_is_valid, 82
 - e820_map, 84
 - e820_type_description, 83
 - get_boot_data, 83
- boot.h
 - BOOT_MAGIC, 23
 - BOOT_SIGNATURE, 23
 - E820_ACPI, 23
 - e820_addr_t, 24
 - e820_get_addr, 24
 - e820_get_size, 24
 - e820_get_type, 24
 - e820_is_available, 25
 - e820_is_valid, 25
 - E820_RAM, 23
 - E820_RESERVED, 23
 - e820_size_t, 24
 - e820_type_description, 25
 - e820_type_t, 24
 - get_boot_data, 26
 - SETUP_HEADER, 23
 - BOOT_MAGIC
 - boot.h, 23
 - boot_setup_addr
 - boot.c, 84
 - BOOT_SIGNATURE
 - boot.h, 23
 - boot_t, 5
 - magic, 5
 - ram_size, 6
 - root_dev, 6
 - root_flags, 6
 - setup_sects, 6
 - signature, 6
 - syzise, 6
 - vid_mode, 6
- cache
 - vm_alloc_t, 12
- CHAR_BS
 - ascii.h, 19
- CHAR_CR
 - ascii.h, 19
- CHAR_HT
 - ascii.h, 19
- CHAR_LF
 - ascii.h, 19
- count_t
 - kernel.h, 42
- E820_ACPI
 - boot.h, 23
- e820_addr_t
 - boot.h, 24
- e820_get_addr
 - boot.c, 81
 - boot.h, 24
- e820_get_size
 - boot.c, 82
 - boot.h, 24
- e820_get_type
 - boot.c, 82
 - boot.h, 24
- e820_is_available
 - boot.c, 82
 - boot.h, 25

-
- e820_is_valid
 - boot.c, 82
 - boot.h, 25
 - e820_map
 - boot.c, 84
 - E820_RAM
 - boot.h, 23
 - E820_RESERVED
 - boot.h, 23
 - e820_size_t
 - boot.h, 24
 - e820_t, 7
 - addr, 7
 - size, 7
 - type, 7
 - e820_type_description
 - boot.c, 83
 - boot.h, 25
 - e820_type_t
 - boot.h, 24
 - empty
 - slab_cache_t, 8
 - false
 - stdbool.h, 58
 - free_list
 - slab_header_t, 10
 - full
 - slab_cache_t, 9
 - get_boot_data
 - boot.c, 83
 - boot.h, 26
 - halt
 - startup.h, 55
 - head
 - vm_alloc_t, 12
 - idle
 - kernel.c, 86
 - kernel.h, 42
 - inb
 - io.h, 27
 - inl
 - io.h, 27
 - inw
 - io.h, 27
 - io.h
 - inb, 27
 - inl, 27
 - inw, 27
 - outb, 27
 - outl, 27
 - outw, 27
 - jinue/vm.h
 - KLIMIT, 28
 - PAGE_BITS, 29
 - PAGE_MASK, 29
 - PAGE_OFFSET_OF, 29
 - PAGE_SIZE, 29
 - PLIMIT, 29
 - kernel
 - kernel.c, 86
 - kernel.h, 42
 - kernel.c
 - idle, 86
 - kernel, 86
 - kernel_size, 88
 - kernel_top, 88
 - kinit, 86
 - kernel.h
 - addr_t, 42
 - count_t, 42
 - idle, 42
 - kernel, 42
 - kernel_size, 44
 - kernel_start, 42
 - kernel_top, 44
 - kinit, 43
 - kernel_size
 - kernel.c, 88
 - kernel.h, 44
 - kernel_start
 - kernel.h, 42
 - kernel_top
 - kernel.c, 88
 - kernel.h, 44
 - kinit
 - kernel.c, 86
-

- kernel.h, 43
- KLIMIT
 - jinue/vm.h, 28
- magic
 - boot_t, 5
- next
 - slab_header_t, 10
 - vm_link_t, 14
- NULL
 - stddef.h, 59
- obj_size
 - slab_cache_t, 8
- offsetof
 - stddef.h, 59
- outb
 - io.h, 27
- outl
 - io.h, 27
- outw
 - io.h, 27
- PAGE_BITS
 - jinue/vm.h, 29
- PAGE_DIRECTORY
 - vm.h, 34
- PAGE_DIRECTORY_MAPPING
 - vm.h, 34
- PAGE_DIRECTORY_OFFSET_OF
 - vm.h, 34
- PAGE_MASK
 - jinue/vm.h, 29
- PAGE_OFFSET_OF
 - jinue/vm.h, 29
- PAGE_SIZE
 - jinue/vm.h, 29
- PAGE_TABLE_BITS
 - vm.h, 34
- PAGE_TABLE_ENTRIES
 - vm.h, 34
- PAGE_TABLE_MASK
 - vm.h, 34
- PAGE_TABLE_OF
 - vm.h, 35
- PAGE_TABLE_OFFSET_OF
 - vm.h, 35
- PAGE_TABLE_PTE_OF
 - vm.h, 35
- PAGE_TABLE_SIZE
 - vm.h, 35
- page_table_t
 - vm.h, 38
- PAGE_TABLES
 - vm.h, 35
- PAGE_TABLES_MAPPING
 - vm.h, 35
- PAGE_TABLES_TABLE
 - vm.h, 36
- panic
 - panic.c, 89
 - panic.h, 45
- panic.c
 - panic, 89
- panic.h
 - panic, 45
- partial
 - slab_cache_t, 9
- PDE_OF
 - vm.h, 36
- per_slab
 - slab_cache_t, 8
- PLIMIT
 - jinue/vm.h, 29
- PMAPPING_END
 - vm.h, 36
- PMAPPING_START
 - vm.h, 36
- prev
 - slab_header_t, 10
- print_hex_b
 - printk.c, 90
 - printk.h, 46
- print_hex_l
 - printk.c, 91
 - printk.h, 46
- print_hex_nibble
 - printk.c, 91
 - printk.h, 47
- print_hex_q
 - vm.h, 36

-
- printk.c, 92
 - printk.h, 48
 - print_hex_w
 - printk.c, 92
 - printk.h, 48
 - print_unsigned_int
 - printk.c, 93
 - printk.h, 49
 - printk
 - printk.c, 93
 - printk.h, 49
 - printk.c
 - print_hex_b, 90
 - print_hex_l, 91
 - print_hex_nibble, 91
 - print_hex_q, 92
 - print_hex_w, 92
 - print_unsigned_int, 93
 - printk, 93
 - printk.h
 - print_hex_b, 46
 - print_hex_l, 46
 - print_hex_nibble, 47
 - print_hex_q, 48
 - print_hex_w, 48
 - print_unsigned_int, 49
 - printk, 49
 - PTE_OF
 - vm.h, 36
 - PTE_SIZE
 - vm.h, 36
 - pte_t
 - vm.h, 38
 - ptrdiff_t
 - stddef.h, 60
 - ram_size
 - boot_t, 6
 - root_dev
 - boot_t, 6
 - root_flags
 - boot_t, 6
 - SETUP_HEADER
 - boot.h, 23
 - setup_sects
 - boot_t, 6
 - signature
 - boot_t, 6
 - size
 - e820_t, 7
 - vm_alloc_t, 12
 - vm_link_t, 14
 - size_t
 - stddef.h, 60
 - slab.c
 - slab_alloc, 96
 - slab_free, 96
 - slab_prepare_page, 97
 - slab.h
 - slab_alloc, 53
 - slab_cache_t, 53
 - slab_free, 53
 - slab_header_t, 53
 - slab_prepare_page, 53
 - slab_alloc
 - slab.c, 96
 - slab.h, 53
 - slab_cache_t, 8
 - empty, 8
 - full, 9
 - obj_size, 8
 - partial, 9
 - per_slab, 8
 - slab.h, 53
 - vm_allocator, 9
 - slab_free
 - slab.c, 96
 - slab.h, 53
 - slab_header_t, 10
 - available, 10
 - free_list, 10
 - next, 10
 - prev, 10
 - slab.h, 53
 - slab_prepare_page
 - slab.c, 97
 - slab.h, 53
 - start
 - startup.h, 55
 - startup.h
 - halt, 55
-

-
- start, 55
 - stdarg.h
 - va_arg, 56
 - va_copy, 56
 - va_end, 56
 - va_list, 57
 - va_start, 56
 - stdbool.h
 - __bool_true_false_are_defined, 58
 - bool, 58
 - false, 58
 - true, 58
 - stddef.h
 - NULL, 59
 - offsetof, 59
 - ptrdiff_t, 60
 - size_t, 60
 - wchar_t, 60
 - sysize
 - boot_t, 6
 - true
 - stdbool.h, 58
 - type
 - e820_t, 7
 - va_arg
 - stdarg.h, 56
 - va_copy
 - stdarg.h, 56
 - va_end
 - stdarg.h, 56
 - va_list
 - stdarg.h, 57
 - va_start
 - stdarg.h, 56
 - vga.c
 - vga_clear, 98
 - vga_get_cursor_pos, 99
 - vga_init, 99
 - vga_print, 100
 - vga_printn, 100
 - vga_putc, 101
 - vga_scroll, 101
 - vga_set_cursor_pos, 102
 - vga.h
 - vga_clear, 65
 - VGA_COL, 62
 - VGA_COLOR_BLACK, 62
 - VGA_COLOR_BLUE, 62
 - VGA_COLOR_BRIGHTBLUE, 62
 - VGA_COLOR_BRIGHTCYAN, 62
 - VGA_COLOR_BRIGHTGREEN, 62
 - VGA_COLOR_BRIGHTMAGENTA, 62
 - VGA_COLOR_BRIGHTRED, 63
 - VGA_COLOR_BRIGHTWHITE, 63
 - VGA_COLOR_BROWN, 63
 - VGA_COLOR_CYAN, 63
 - VGA_COLOR_DEFAULT, 63
 - VGA_COLOR_ERASE, 63
 - VGA_COLOR_GRAY, 63
 - VGA_COLOR_GREEN, 63
 - VGA_COLOR_MAGENTA, 63
 - VGA_COLOR_RED, 64
 - VGA_COLOR_WHITE, 64
 - VGA_COLOR_YELLOW, 64
 - VGA_CRTC_ADDR, 64
 - VGA_CRTC_DATA, 64
 - VGA_FB_FLAG_ACTIVE, 64
 - vga_get_cursor_pos, 66
 - vga_init, 66
 - VGA_LINE, 64
 - VGA_LINES, 64
 - VGA_MISC_OUT_RD, 65
 - VGA_MISC_OUT_WR, 65
 - vga_pos_t, 65
 - vga_print, 67
 - vga_printn, 68
 - vga_putc, 68
 - vga_scroll, 68
 - vga_set_cursor_pos, 69
 - VGA_TAB_WIDTH, 65
 - VGA_TEXT_VID_BASE, 65
 - VGA_WIDTH, 65
 - vga_clear
-

-
- vga.c, 98
 - vga.h, 65
 - VGA_COL
 - vga.h, 62
 - VGA_COLOR_BLACK
 - vga.h, 62
 - VGA_COLOR_BLUE
 - vga.h, 62
 - VGA_COLOR_BRIGHTBLUE
 - vga.h, 62
 - VGA_COLOR_BRIGHTCYAN
 - vga.h, 62
 - VGA_COLOR_BRIGHTGREEN
 - vga.h, 62
 - VGA_COLOR_-
 - BRIGHTMAGENTA
 - vga.h, 62
 - VGA_COLOR_BRIGHTRED
 - vga.h, 63
 - VGA_COLOR_BRIGHTWHITE
 - vga.h, 63
 - VGA_COLOR_BROWN
 - vga.h, 63
 - VGA_COLOR_CYAN
 - vga.h, 63
 - VGA_COLOR_DEFAULT
 - vga.h, 63
 - VGA_COLOR_ERASE
 - vga.h, 63
 - VGA_COLOR_GRAY
 - vga.h, 63
 - VGA_COLOR_GREEN
 - vga.h, 63
 - VGA_COLOR_MAGENTA
 - vga.h, 63
 - VGA_COLOR_RED
 - vga.h, 64
 - VGA_COLOR_WHITE
 - vga.h, 64
 - VGA_COLOR_YELLOW
 - vga.h, 64
 - VGA_CRTC_ADDR
 - vga.h, 64
 - VGA_CRTC_DATA
 - vga.h, 64
 - VGA_FB_FLAG_ACTIVE
 - vga.h, 64
 - vga.h, 64
 - vga_get_cursor_pos
 - vga.c, 99
 - vga.h, 66
 - vga_init
 - vga.c, 99
 - vga.h, 66
 - VGA_LINE
 - vga.h, 64
 - VGA_LINES
 - vga.h, 64
 - VGA_MISC_OUT_RD
 - vga.h, 65
 - VGA_MISC_OUT_WR
 - vga.h, 65
 - vga_pos_t
 - vga.h, 65
 - vga_print
 - vga.c, 100
 - vga.h, 67
 - vga_printn
 - vga.c, 100
 - vga.h, 68
 - vga_putc
 - vga.c, 101
 - vga.h, 68
 - vga_scroll
 - vga.c, 101
 - vga.h, 68
 - vga_set_cursor_pos
 - vga.c, 102
 - vga.h, 69
 - VGA_TAB_WIDTH
 - vga.h, 65
 - VGA_TEXT_VID_BASE
 - vga.h, 65
 - VGA_WIDTH
 - vga.h, 65
 - vid_mode
 - boot_t, 6
 - vm.c
 - vm_map, 103
 - vm_unmap, 104
 - vm.h
 - PAGE_DIRECTORY, 34
-

-
- PAGE_DIRECTORY_-
 - MAPPING, 34
 - PAGE_DIRECTORY_-
 - OFFSET_OF, 34
 - PAGE_TABLE_BITS, 34
 - PAGE_TABLE_ENTRIES, 34
 - PAGE_TABLE_MASK, 34
 - PAGE_TABLE_OF, 35
 - PAGE_TABLE_OFFSET_OF,
 - 35
 - PAGE_TABLE_PTE_OF, 35
 - PAGE_TABLE_SIZE, 35
 - page_table_t, 38
 - PAGE_TABLES, 35
 - PAGE_TABLES_MAPPING, 35
 - PAGE_TABLES_TABLE, 36
 - PDE_OF, 36
 - PMAPPING_END, 36
 - PMAPPING_START, 36
 - PTE_OF, 36
 - PTE_SIZE, 36
 - pte_t, 38
 - VM_FLAG_ACCESSED, 37
 - VM_FLAG_BIG_PAGE, 37
 - VM_FLAG_CACHE_-
 - DISABLE, 37
 - VM_FLAG_DIRTY, 37
 - VM_FLAG_GLOBAL, 37
 - VM_FLAG_KERNEL, 37
 - VM_FLAG_PRESENT, 37
 - VM_FLAG_READ_ONLY, 38
 - VM_FLAG_USER, 38
 - VM_FLAG_WRITE_-
 - THROUGH, 38
 - vm_map, 38
 - vm_unmap, 39
 - vm_alloc
 - vm_alloc.c, 106
 - vm_alloc.h, 71
 - vm_alloc.c
 - vm_alloc, 106
 - vm_free, 107
 - vm_valloc, 107
 - vm_vfree, 108
 - vm_vfree_block, 108
 - vm_alloc.h
 - vm_alloc, 71
 - vm_alloc_t, 71
 - vm_free, 71
 - vm_link_t, 71
 - vm_valloc, 72
 - vm_vfree, 73
 - vm_vfree_block, 73
 - vm_alloc_t, 12
 - cache, 12
 - head, 12
 - size, 12
 - vm_alloc.h, 71
 - vm_allocator
 - slab_cache_t, 9
 - VM_FLAG_ACCESSED
 - vm.h, 37
 - VM_FLAG_BIG_PAGE
 - vm.h, 37
 - VM_FLAG_CACHE_DISABLE
 - vm.h, 37
 - VM_FLAG_DIRTY
 - vm.h, 37
 - VM_FLAG_GLOBAL
 - vm.h, 37
 - VM_FLAG_KERNEL
 - vm.h, 37
 - VM_FLAG_PRESENT
 - vm.h, 37
 - VM_FLAG_READ_ONLY
 - vm.h, 38
 - VM_FLAG_USER
 - vm.h, 38
 - VM_FLAG_WRITE_THROUGH
 - vm.h, 38
 - vm_free
 - vm_alloc.c, 107
 - vm_alloc.h, 71
 - vm_link_t, 14
 - addr, 14
 - next, 14
 - size, 14
 - vm_alloc.h, 71
 - vm_map
 - vm.c, 103
 - vm.h, 38
 - vm_unmap
-

- vm.c, 104
 - vm.h, 39
- vm_valloc
 - vm_alloc.c, 107
 - vm_alloc.h, 72
- vm_vfree
 - vm_alloc.c, 108
 - vm_alloc.h, 73
- vm_vfree_block
 - vm_alloc.c, 108
 - vm_alloc.h, 73
- wchar_t
 - stddef.h, 60