LameOS

1.0

Generated by Doxygen 1.9.1

Chapter 1

Class Index

1.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

Chapter 3

Class Documentation

3.1 idt_desc Struct Reference

Interrupt Descriptor Table (IDT) Descriptor.

`#include <idt.h>`

Public Attributes

- uint16_t offset_1
- uint16_t selector
- uint8_t zero
- uint8_t type_attr
- uint16_t offset_2

3.1.1 Detailed Description

Interrupt Descriptor Table (IDT) Descriptor.

This structure represents a single entry in the Interrupt Descriptor Table (IDT). The IDT is used by the processor to handle interrupts and exceptions. Each IDT descriptor corresponds to a specific interrupt or exception and provides the necessary information for the processor to handle them correctly.

3.1.2 Member Data Documentation

3.1.2.1 offset_1

`uint16_t idt_desc::offset_1`

3.1.2.2 offset_2

```
uint16_t idt_desc::offset_2
```

3.1.2.3 selector

```
uint16_t idt_desc::selector
```

3.1.2.4 type_attr

```
uint8_t idt_desc::type_attr
```

3.1.2.5 zero

```
uint8_t idt_desc::zero
```

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

- /home/fransys/prog/C/bootable_code/src/idt/idt.h

3.2 idtr_desc Struct Reference

IDT Register (IDTR) Descriptor.

```
#include <idt.h>
```

Public Attributes

- uint16_t limit
- uint32_t base

3.2.1 Detailed Description

IDT Register (IDTR) Descriptor.

This structure represents the IDT Register (IDTR) descriptor, which provides the base address and limit of the Interrupt Descriptor Table (IDT). The IDTR is a control register used by the processor to locate and access the IDT.

3.2.2 Member Data Documentation

3.2.2.1 base

```
uint32_t idtr_desc::base
```

3.2.2.2 limit

```
uint16_t idtr_desc::limit
```

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

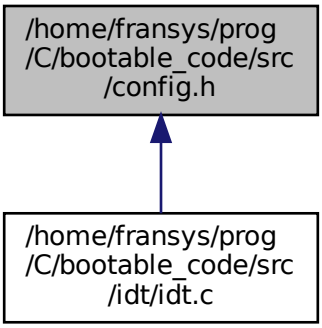
- /home/fransys/prog/C/bootable_code/src/idt/idt.h

Chapter 4

File Documentation

4.1 /home/fransys/prog/C/bootable_code/src/config.h File Reference

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



Macros

- #define KERNEL_CODE_SELECTOR 0X08

  Code Segment Selector.
- #define KERNEL_DATA_SELECTOR 0X10

  Data Segment Selector.
- #define LAMEOS_TOTAL_INTERRUPTS 512

  Macro Constant Defining Total Interrupts.

4.1.1 Macro Definition Documentation

4.1.1.1 KERNEL_CODE_SELECTOR

```
#define KERNEL_CODE_SELECTOR 0X08
```

Code Segment Selector.

The offset of the code_seg entry in the GDT is 0x08.

4.1.1.2 KERNEL_DATA_SELECTOR

```
#define KERNEL_DATA_SELECTOR 0X10
```

Data Segment Selector.

The offset of the data_seg entry in the GDT is 0x10.

4.1.1.3 LAMEOS_TOTAL_INTERRUPTS

```
#define LAMEOS_TOTAL_INTERRUPTS 512
```

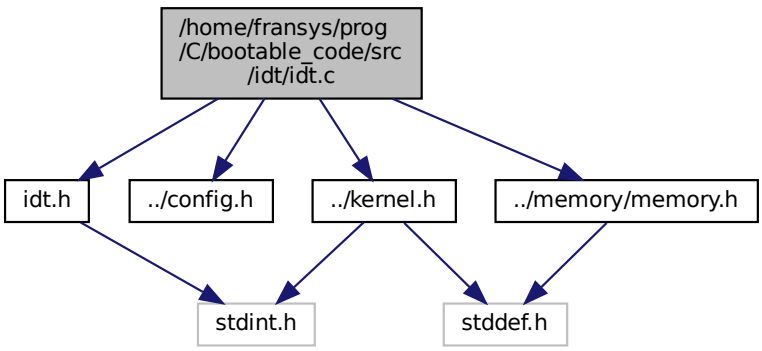
Macro Constant Defining Total Interrupts.

The IDT is an array of 512 descriptors, each 8 bytes long. Although in reality only 256 are actually available for use by programmers. The rest are reserved by the CPU for one reason or another.

4.2 /home/fransys/prog/C/bootable_code/src/idt/idt.c File Reference

```
#include "idt.h"
#include "../config.h"
#include "../kernel.h"
#include "../memory/memory.h"
```

Include dependency graph for idt.c:



Functions

- void idt_load (struct idtr_desc *ptr)

  Wrapper function for assembly routine idt_load.
- void idt_zero ()

  Interrupt Zero Definition.
- void idt_set (int interrupt_no, void *address)

  Defines an IDT descriptor.
- void idt_init ()

  Initialize Kernel Interrupt Descriptor Table (IDT).

Variables

- struct idt_desc idt_descriptors [LAMEOS_TOTAL_INTERRUPTS]

  Array of 512 IDT Descriptors.
- struct idtr_desc idtr_descriptor

  A struct representing the IDT register (IDTR).

4.2.1 Function Documentation

4.2.1.1 idt_init()

```
void idt_init ( )
```

Initialize Kernel Interrupt Descriptor Table (IDT).

Initializes kernel IDT array by zeroing every descpritor in the array, Sets the IDTR descriptor limit and base, Intended to set each IDT descriptor, but currently only sets the interrupt descriptor 0, Concludes Loads the IDTR by calling wrapper function idt_load, for the asm function of the same name. The asm routine idt_load loads the IDTR with the kernel IDTR struct.

Note

There is a 1:1 mapping between the IDT and the CPU's interrupt numbers.

See also

memset in src/memory/memory.c

idt_set in src/idt/idt.c

idt_load in src/idt/idt.asm

4.2.1.2 idt_load()

```
void idt_load (
            struct idtr_desc * ptr )
```

Wrapper function for assembly routine idt_load.

The wrapper fct is called from within idt_init. It loads the IDTR by calling the assembly function idt_load. By loading the kernel IDTR struct, the processor knows where the kernel IDT struct-array is located in memory.

Note

The assembly routine is exposed to the linker by `global idt_load` in the idt.asm file.

See also

idt_init in src/idt/idt.c

idt_load in src/idt/idt.asm

Parameters

ptr	a void pointer to the IDTR descriptor

4.2.1.3 idt_set()

```
void idt_set (
            int interrupt_no,
            void * address )
```

Defines an IDT descriptor.

Defines a descriptor by setting the offset, selector, zero, type_attr, and offset_2 fields of the descriptor. The offset is the address of the programmable interrupt routine. The selector is the kernel code selector. The zero field is unused and set to zero. The type_attr field is set to 0xEE, which is the type and attributes for a 32-bit interrupt gate. The offset_2 field is the upper 16 bits of the offset.

See also

idt_init in src/idt/idt.c

Parameters

interrupt_no	The CPU interrupt number to map fct address to.
address	The address of the programmable interrupt routine.

4.2.1.4 idt_zero()

```
void idt_zero ( )
```

Interrupt Zero Definition.

This interrupt routine is called by the CPU when a divide by zero exception occurs. It is mapped to interrupt 0 in the CPU's IDT when idt_init is called. The routine clears the screen and prints an error message.

See also

idt_load in src/idt/idt.asm

4.2.2 Variable Documentation

4.2.2.1 idt_descriptors

```
struct idt_desc idt_descriptors[LAMEOS_TOTAL_INTERRUPTS]
```

Array of 512 IDT Descriptors.

The kernel maintains an array of 512 IDT descriptors. Each descriptor corresponds to a specific interrupt or exception. The array is initialized by idt_init.

4.2.2.2 idtr_descriptor

```
struct idtr_desc idtr_descriptor
```

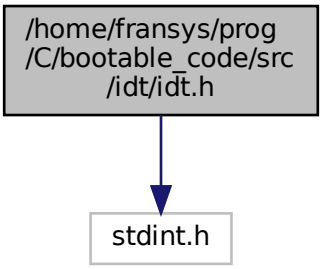
A struct representing the IDT register (IDTR).

The IDTR is a control register used by the processor to locate and access the IDT. The IDTR is initialized by idt_init.

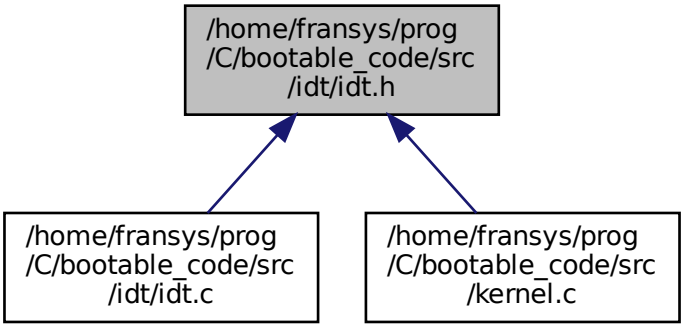
4.3 /home/fransys/prog/C/bootable_code/src/idt/idt.h File Reference

```
#include <stdint.h>
```

Include dependency graph for idt.h:



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



Classes

- struct idt_desc

  Interrupt Descriptor Table (IDT) Descriptor.
- struct idtr_desc

  IDT Register (IDTR) Descriptor.

Functions

- struct idt_desc __attribute__ ((packed))
- void idt_init ()

  Initialize Kernel Interrupt Descriptor Table (IDT).

Variables

- uint16_t offset_1
- uint16_t selector
- uint8_t zero
- uint8_t type_attr
- uint16_t offset_2
- uint16_t limit
- uint32_t base

4.3.1 Function Documentation

4.3.1.1 __attribute__()

```
struct idtr_desc __attribute__ (
             (packed)  )
```

4.3.1.2 idt_init()

```
void idt_init ( )
```

Initialize Kernel Interrupt Descriptor Table (IDT).

Initializes the Interrupt Descriptor Table (IDT) by: Zeroing out the user-IDT array, Setting the IDT Register (IDTR) descriptor limit and base, Setting the IDT descriptors for each programmed interrupt, and Loading the IDTR by calling the assembly function idt_load.

See also

idt_init in src/idt/idt.c

Initializes kernel IDT array by zeroing every descpritor in the array, Sets the IDTR descriptor limit and base, Intended to set each IDT descriptor, but currently only sets the interrupt descriptor 0, Concludes Loads the IDTR by calling wrapper function idt_load, for the asm function of the same name. The asm routine idt_load loads the IDTR with the kernel IDTR struct.

Note

There is a 1:1 mapping between the IDT and the CPU's interrupt numbers.

See also

memset in src/memory/memory.c

idt_set in src/idt/idt.c

idt_load in src/idt/idt.asm

4.3.2 Variable Documentation

4.3.2.1 base

```
uint32_t base
```

4.3.2.2 limit

```
uint16_t limit
```

4.3.2.3 offset_1

```
uint16_t offset_1
```

4.3.2.4 offset_2

```
uint16_t offset_2
```

4.3.2.5 selector

```
uint16_t selector
```

4.3.2.6 type_attr

```
uint8_t type_attr
```

4.3.2.7 zero

```
uint8_t zero
```

4.4 /home/fransys/prog/C/bootable_code/src/io/io.h File Reference

Functions

- unsigned char insb (unsigned short port)

  C wrapper of BIOS `insb` instruction Reads a byte in from a PIO port.
- unsigned short insw (unsigned short port)

  C wrapper of BIOS `insw` instruction Reads a word in from a PIO port.
- void outb (unsigned short port, unsigned char val)

  C wrapper of BIOS `outb` instruction Writes a byte out to a PIO port.
- void outw (unsigned short port, unsigned short val)

  C wrapper of BIOS `outw` instruction Writes a word out to a PIO port.

4.4.1 Function Documentation

4.4.1.1 insb()

```
unsigned char insb (
            unsigned short port )
```

C wrapper of BIOS `insb` instruction Reads a byte in from a PIO port.

See also

/src/io/io.asm

Parameters

port	The PIO port to read from, range 0x0000 - 0xFFFF (0-65535).

Returns

unsigned char, the byte read in from the port.

Note

This function is implemented in assembly. A char is 1 byte.

4.4.1.2 insw()

```
unsigned short insw (
            unsigned short port )
```

C wrapper of BIOS `insw` instruction Reads a word in from a PIO port.

Parameters

port	The PIO port to read from, range 0x0000 - 0xFFFF (0-65535).

Returns

unsigned short, the word read in from the port.

Note

This function is implemented in assembly. A short is 2 bytes.

4.4.1.3 outb()

```
void outb (
            unsigned short port,
            unsigned char val )
```

C wrapper of BIOS `outb` instruction Writes a byte out to a PIO port.

Parameters

port	The PIO port to write to, range 0x0000 - 0xFFFF (0-65535).
val	The byte to write out to the port.

Note

This function is implemented in assembly. A char is 1 byte.

4.4.1.4 outw()

```
void outw (
            unsigned short port,
            unsigned short val )
```

C wrapper of BIOS `outw` instruction Writes a word out to a PIO port.

Parameters

port	The PIO port to write to, range 0x0000 - 0xFFFF (0-65535).
val	The word to write out to the port.

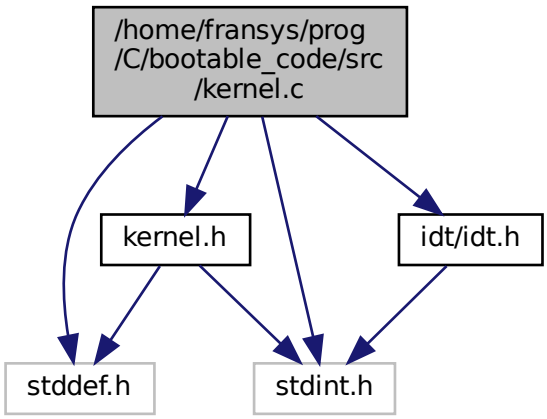
Note

This function is implemented in assembly. A short is 2 bytes.

4.5 /home/fransys/prog/C/bootable_code/src/kernel.c File Reference

```
#include "kernel.h"
#include "idt/idt.h"
#include <stddef.h>
#include <stdint.h>
```

Include dependency graph for kernel.c:



Functions

- uint16_t term_make_char (char c, char color)

  Decodes a character and color into a uint16_t.
- void term_putchar (int x, int y, char c, char color)

  Writes a character to the VGA framebuffer.
- void term_initialize ()

  Initializes the VGA framebuffer.
- size_t strlen (const char ∗str)

  Returns the length of a string.
- void term_writechar (char c, char color)

  Writes a character, advancing cursor, newline if necessary.
- void print (const char ∗str)

  Writes a string using term_writechar.
- void lame_color_show ()

  This is what LameOS is all about.
- void kernel_main ()

Variables

- uint16_t * video_mem = 0

  Pointer to VGA Framebuffer.
- uint16_t term_row = 0

  VGA Framebuffer Width.
- uint16_t term_col = 0

  VGA Framebuffer Height.

4.5.1 Function Documentation

4.5.1.1 kernel_main()

```
void kernel_main ( )
```

4.5.1.2 lame_color_show()

```
void lame_color_show ( )
```

This is what LameOS is all about.

This function iterates kaleidoscopically through all characters and colors in the VGA framebuffer. It does this forever. EPILEPSY WARNING!

4.5.1.3 print()

```
void print (
            const char * str )
```

Writes a string using term_writechar.

This function writes a string by iterating through the string and writing each character using term_writechar to the VGA framebuffer.

Parameters

str	The string to write.

4.5.1.4 strlen()

```
size_t strlen (
            const char * str )
```

Returns the length of a string.

This function returns the length of a string by iterating through the string until it reaches a null terminator, maintaining a count as it goes.

Parameters

str	The string to get the length of.

Returns

size_t The length of the string.

4.5.1.5 term_initialize()

```
void term_initialize ( )
```

Initializes the VGA framebuffer.

This function initializes the VGA framebuffer by clearing the screen and setting the video_mem pointer to 0xB8000. The screen is cleared by calling term_putchar with space characters and a black background on position in the framebuffer.

Note

sets term_row and term_col to 0. Useful for related functions.

4.5.1.6 term_make_char()

```
uint16_t term_make_char (
             char c,
             char color )
```

Decodes a character and color into a uint16_t.

The VGA framebuffer is a 2D array of uint16_t. Each uint16_t represents a character and its color. The first 8 bits of the uint16_t are the character and the last 8 bits are the color.

Parameters

c	The character to display.
color	The color of the character.

Returns

uint16_t The character and color encoded into a uint16_t.

4.5.1.7 term_putchar()

```
void term_putchar (
            int x,
            int y,
            char c,
            char color )
```

Writes a character to the VGA framebuffer.

This function writes a character and color, given by c and color, to the VGA framebuffer at the specified location, given by x and y. The function first converts the x and y to a 1D index, then writes the character and color to the framebuffer at that index.

Parameters

x	The x coordinate, column, range 0-79.
y	The y coordinate, row, range 0-24.
c	The character to display, range 0-255.
color	The color of the character, range 0-15.

4.5.1.8 term_writechar()

```
void term_writechar (
            char c,
            char color )
```

Writes a character, advancing cursor, newline if necessary.

Writes a character, advancing the cursor. If the cursor is at the end of the line, the cursor is moved to the next line.

Parameters

c	The character to write.
color	The color of the character.

4.5.2 Variable Documentation

4.5.2.1 term_col

```
uint16_t term_col = 0
```

VGA Framebuffer Height.

The VGA framebuffer is 25 characters high.

4.5.2.2 term_row

```
uint16_t term_row = 0
```

VGA Framebuffer Width.

The VGA framebuffer is 80 characters wide.

4.5.2.3 video_mem

```
uint16_t* video_mem = 0
```

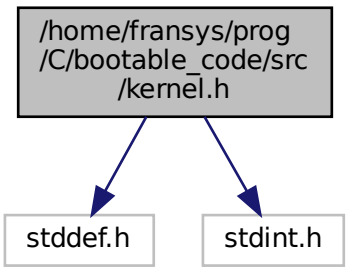
Pointer to VGA Framebuffer.

The kernel uses the VGA framebuffer to display text on the screen. The framebuffer is located at 0xB8000. The kernel writes to the framebuffer using the term_putchar function.

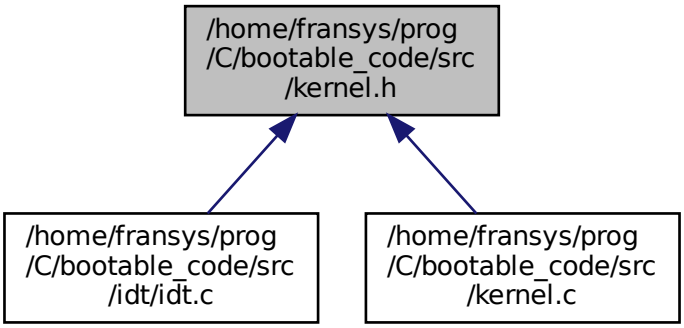
4.6 /home/fransys/prog/C/bootable_code/src/kernel.h File Reference

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

Include dependency graph for kernel.h:



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



Macros

- #define VGA_WIDTH 80

  Macro Constant for VGA framebuffer width.
- #define VGA_HEIGHT 25

  Macro Constant for VGA framebuffer height.

Functions

- void kernel_main ()
- void term_initialize ()

  Initializes the VGA framebuffer.
- size_t strlen (const char ∗str)

  Returns the length of a string.
- uint16_t term_make_char (char c, char color)

  Decodes a character and color into a uint16_t.
- void term_putchar (int x, int y, char c, char color)

  Writes a character to the VGA framebuffer.
- void term_writechar (char c, char color)

  Writes a character, advancing cursor, newline if necessary.
- void print (const char ∗str)

  Writes a string using term_writechar.
- void lame_color_show ()

  This is what LameOS is all about.

4.6.1 Macro Definition Documentation

4.6.1.1 VGA_HEIGHT

```
#define VGA_HEIGHT 25
```

Macro Constant for VGA framebuffer height.

4.6.1.2 VGA_WIDTH

```
#define VGA_WIDTH 80
```

Macro Constant for VGA framebuffer width.

4.6.2 Function Documentation

4.6.2.1 kernel_main()

```
void kernel_main ( )
```

4.6.2.2 lame_color_show()

```
void lame_color_show ( )
```

This is what LameOS is all about.

This function iterates kaleidoscopically through all characters and colors in the VGA framebuffer. It does this forever. EPILEPSY WARNING!

4.6.2.3 print()

```
void print (
            const char * str )
```

Writes a string using term_writechar.

This function writes a string by iterating through the string and writing each character using term_writechar to the VGA framebuffer.

Parameters

str	The string to write.

4.6.2.4 strlen()

```
size_t strlen (
            const char * str )
```

Returns the length of a string.

This function returns the length of a string by iterating through the string until it reaches a null terminator, maintaining a count as it goes.

Parameters

str	The string to get the length of.

Returns

size_t The length of the string.

4.6.2.5 term_initialize()

```
void term_initialize ( )
```

Initializes the VGA framebuffer.

This function initializes the VGA framebuffer by clearing the screen and setting the video_mem pointer to 0xB8000. The screen is cleared by calling term_putchar with space characters and a black background on position in the framebuffer.

Note

sets term_row and term_col to 0. Useful for related functions.

4.6.2.6 term_make_char()

```
uint16_t term_make_char (
            char c,
            char color )
```

Decodes a character and color into a uint16_t.

The VGA framebuffer is a 2D array of uint16_t. Each uint16_t represents a character and its color. The first 8 bits of the uint16_t are the character and the last 8 bits are the color.

Parameters

c	The character to display.
color	The color of the character.

Returns

uint16_t The character and color encoded into a uint16_t.

4.6.2.7 term_putchar()

```
void term_putchar (
            int x,
            int y,
            char c,
            char color )
```

Writes a character to the VGA framebuffer.

This function writes a character and color, given by c and color, to the VGA framebuffer at the specified location, given by x and y. The function first converts the x and y to a 1D index, then writes the character and color to the framebuffer at that index.

Parameters

x	The x coordinate, column, range 0-79.
y	The y coordinate, row, range 0-24.
c	The character to display, range 0-255.
color	The color of the character, range 0-15.

4.6.2.8 term_writechar()

```
void term_writechar (
            char c,
            char color )
```

Writes a character, advancing cursor, newline if necessary.

Writes a character, advancing the cursor. If the cursor is at the end of the line, the cursor is moved to the next line.

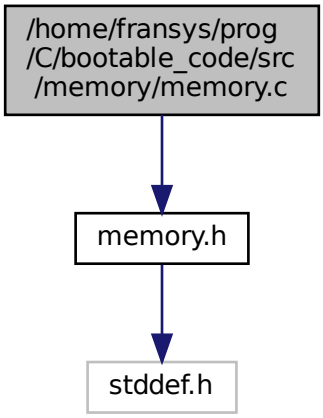
Parameters

c	The character to write.
color	The color of the character.

4.7 /home/fransys/prog/C/bootable_code/src/memory/memory.c File Reference

```
#include "memory.h"
```

Include dependency graph for memory.c:



Functions

- void * memset (void *ptr, int c, size_t size)

  Generic memset implementation.

4.7.1 Function Documentation

4.7.1.1 memset()

```
void* memset (
            void * ptr,
            int c,
            size_t size )
```

Generic memset implementation.

Takes a void pointer ptr to a memory location, an int c to fill each byte with, and a size_t size to fill to. (size_t is the loop parameter).

Parameters

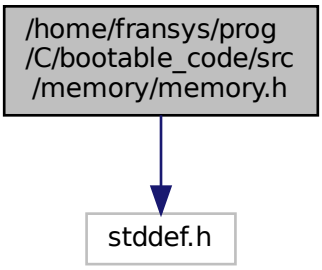
ptr	
c	
size	

Returns

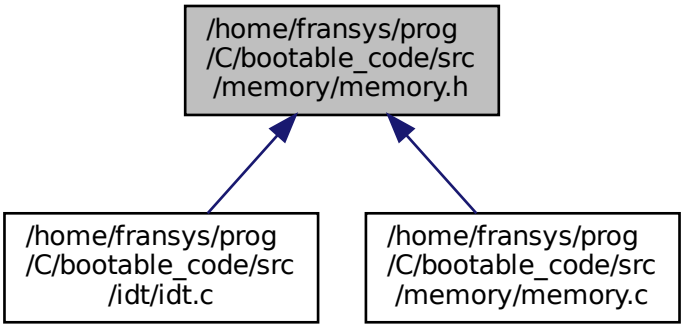
void*

4.8 /home/fransys/prog/C/bootable_code/src/memory/memory.h File Reference

`#include <stddef.h>`
Include dependency graph for memory.h:



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



Functions

- void * memset (void *ptr, int c, size_t size)

  Generic memset implementation.

4.8.1 Function Documentation

4.8.1.1 memset()

```
void* memset (
            void * ptr,
            int c,
            size_t size )
```

Generic memset implementation.

Takes a void pointer ptr to a memory location, an int c to fill each byte with, and a size_t size to fill to. (size_t is the loop parameter).

Parameters

ptr	
c	
size	

Returns

void*

Index