MPX Thunder Krakens

R4

Generated by Doxygen 1.8.6

Fri Apr 8 2016 01:17:21

Contents

Chapter 1

Main Page

Welcome to the Programmer's manual for the Thunder Kracken's MPX Operating system. This document catalogues all of the information one may need to know regarding the use and modification of this Operating system and its contents. Included is a complete API of every method created for the operating system which includes all inputs and outputs as well as a brief summary of the purpose of each method. This will give you a more in depth look at all of the ordinary user commands as well as the internal commands used to perform functions that normal users cannot access. Most likely these commands will be the most important for making new programs on the operating system. This document also lists the documentation for the files files in the operating system. This includes all of the variables and methods used in each file. These will help direct you as to where certain functions are defined. For general usage tips, please refer to the user manual. We hope you find working with the Thunder Kracken's MPX Operating System as enjoyable as we do and we thank you for using our product.

Chapter 2

Data Structure Index

2.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

Chapter 4

Data Structure Documentation

4.1 cmcb Struct Reference

Complete Memory Control Block Struct.

Data Fields

- enum mcb_type type

  Type indicating free or allocated.
- void * begin_address

  Beginning address.
- u32int size

  Indicates size of block in bytes.

4.1.1 Detailed Description

Complete Memory Control Block Struct.

4.1.2 Field Documentation

4.1.2.1 void* cmcb::begin_address

Beginning address.

4.1.2.2 u32int cmcb::size

Indicates size of block in bytes.

4.1.2.3 enum mcb_type cmcb::type

Type indicating free or allocated.

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

- modules/r5/mcb.c

4.2 context Struct Reference

Context structure that holds the 15 CPU register values to begin and resume process execution.

`#include <context.h>`

Data Fields

- u32int gs

  Segment register.
- u32int fs

  Segment register.
- u32int es

  Segment register.
- u32int ds

  Segment register.
- u32int edi

  General-purpose register.
- u32int esi

  General-purpose register.
- u32int ebp

  General-purpose register.
- u32int esp

  General-purpose register.
- u32int ebx

  General-purpose register.
- u32int edx

  General-purpose register.
- u32int ecx

  General-purpose register.
- u32int eax

  General-purpose register.
- u32int eip

  Status and control register.
- u32int cs

  Status and control register.
- u32int eflags

  Status and control register.

4.2.1 Detailed Description

Context structure that holds the 15 CPU register values to begin and resume process execution.

4.2.2 Field Documentation

4.2.2.1 u32int context::cs

Status and control register.

4.2.2.2 u32int context::ds

Segment register.

4.2.2.3 u32int context::eax

General-purpose register.

4.2.2.4 u32int context::ebp

General-purpose register.

4.2.2.5 u32int context::ebx

General-purpose register.

4.2.2.6 u32int context::ecx

General-purpose register.

4.2.2.7 u32int context::edi

General-purpose register.

4.2.2.8 u32int context::edx

General-purpose register.

4.2.2.9 u32int context::eflags

Status and control register.

4.2.2.10 u32int context::eip

Status and control register.

4.2.2.11 u32int context::es

Segment register.

4.2.2.12 u32int context::esi

General-purpose register.

4.2.2.13 u32int context::esp

General-purpose register.

4.2.2.14 u32int context::fs

Segment register.

4.2.2.15 u32int context::gs

Segment register.

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

- modules/r3/context.h

4.3 function_name Struct Reference

A structure to represent each function.

Data Fields

- char * nameStr

  fuction's name
- int(* function)(int argc, char **argv)

  the function
- char * usage

  function's usage or use cases
- char * help

  function's help information

4.3.1 Detailed Description

A structure to represent each function.

4.3.2 Field Documentation

4.3.2.1 int(* function_name::function)(int argc, char **argv)

the function

4.3.2.2 char* function_name::help

function's help information

4.3.2.3 char* function_name::nameStr

fuction's name

4.3.2.4 char* function_name::usage

function's usage or use cases

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

- modules/r1/r1.c

4.4 lmcb Struct Reference

Limited Memory Control Block Struct.

Data Fields

- enum mcb_type type

  Type indicating free or allocated.
- u32int size

  Indicates size of block in bytes.

4.4.1 Detailed Description

Limited Memory Control Block Struct.

4.4.2 Field Documentation

4.4.2.1 u32int lmcb::size

Indicates size of block in bytes.

4.4.2.2 enum mcb_type lmcb::type

Type indicating free or allocated.

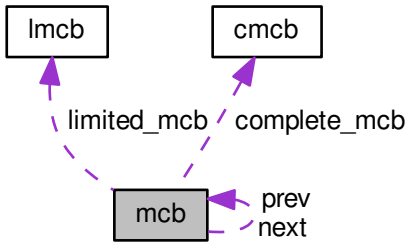
The documentation for this struct was generated from the following file:

- modules/r5/mcb.c

4.5 mcb Struct Reference

Memory Control Block Struct.

Collaboration diagram for mcb:



Data Fields

- struct cmcb * complete_mcb
  Complete Memory Control Block.
- struct lmcb * limited_mcb
  Limited Memory Control Block.
- struct mcb * prev
- struct mcb * next

4.5.1 Detailed Description

Memory Control Block Struct.

4.5.2 Field Documentation

4.5.2.1 struct cmcb* mcb::complete_mcb

Complete Memory Control Block.

4.5.2.2 struct lmcb* mcb::limited_mcb

Limited Memory Control Block.

4.5.2.3 struct mcb * mcb::next

4.5.2.4 struct mcb* mcb::prev

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

- modules/r5/mcb.c

4.6 param Struct Reference

A structure to represent interrupt.

`#include <mpx_supt.h>`

Data Fields

- int op_code

  interrupt's operation

- int device_id

  interrupt's device

4.6.1 Detailed Description

A structure to represent interrupt.

4.6.2 Field Documentation

4.6.2.1 int param::device_id

interrupt's device

4.6.2.2 int param::op_code

interrupt's operation

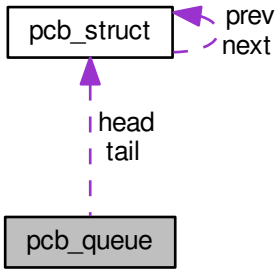
The documentation for this struct was generated from the following file:

- modules/mpx_supt.h

4.7 pcb_queue Struct Reference

Queue structure that will store PCBs.

Collaboration diagram for pcb_queue:



Data Fields

- int count

  The length of the queue.
- struct pcb_struct * head

  Pointer to the start/head of the queue.
- struct pcb_struct * tail

  Pointer to the end/tail of the queue.

4.7.1 Detailed Description

Queue structure that will store PCBs.

4.7.2 Field Documentation

4.7.2.1 int pcb_queue::count

The length of the queue.

4.7.2.2 struct pcb_struct* pcb_queue::head

Pointer to the start/head of the queue.

4.7.2.3 struct pcb_struct* pcb_queue::tail

Pointer to the end/tail of the queue.

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

- modules/r2/pcb.c

4.8 pcb_struct Struct Reference

Struct that will describe PCB Processes.

Collaboration diagram for pcb_struct:

Data Fields

- char name [SIZE_OF_PCB_NAME]

  PCB's name.
- enum process_class class

  PCB's class is an application or system process.
- unsigned char priority

  PCB's priority an integer between 0 and 9.
- enum process_state running_state

  PCB's states are ready, running, or blocked.
- enum process_suspended is_suspended

  PCB process is either suspended or not suspended.
- unsigned char * stack_top

  Pointer to top of the stack.
- unsigned char * stack_base

  Pointer to base of the stack.
- struct pcb_struct * prev

  Pointer to the previous PCB in the queue.
- struct pcb_struct * next

  Pointer to the next PCB in the queue.

4.8.1 Detailed Description

Struct that will describe PCB Processes.

4.8.2 Field Documentation

4.8.2.1 enum process_class pcb_struct::class

PCB's class is an application or system process.

4.8.2.2 enum process_suspended pcb_struct::is_suspended

PCB process is either suspended or not suspended.

4.8.2.3 char pcb_struct::name[SIZE_OF_PCB_NAME]

PCB's name.

4.8.2.4 struct pcb_struct* pcb_struct::next

Pointer to the next PCB in the queue.

4.8.2.5 struct pcb_struct* pcb_struct::prev

Pointer to the previous PCB in the queue.

4.8.2.6 unsigned char pcb_struct::priority

PCB's priority an integer between 0 and 9.

Processes with higher priority values execute before lower priority processes.

4.8.2.7 enum process_state pcb_struct::running_state

PCB's states are ready, running, or blocked.

4.8.2.8 unsigned char* pcb_struct::stack_base

Pointer to base of the stack.

4.8.2.9 unsigned char* pcb_struct::stack_top

Pointer to top of the stack.

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

- modules/r2/pcb.c

Chapter 5

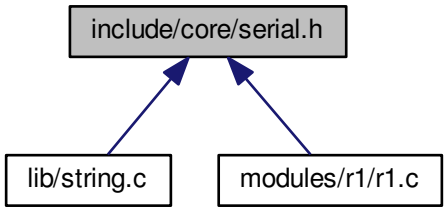
File Documentation

5.1 documentation/mainpage.dox File Reference

5.2 include/core/serial.h File Reference

Serial - Header.

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



Macros

- #define COM1 0x3f8
- #define COM2 0x2f8
- #define COM3 0x3e8
- #define COM4 0x2e8
- #define WithoutEcho 0
- #define WithEcho 1
- #define USER_INPUT_BUFFER_SIZE 100

Functions

- int init_serial (int device)
- int serial_println (const char *msg)
- int serial_print (const char *msg)
- int set_serial_out (int device)
- int set_serial_in (int device)

get_input_line

Get user's input from keyborad.

Parameters

buffer	*The pointer to the buffer where store the user's input.*
buffer_size	*The size of that buffer.*
bWithEcho	*With echo or not*

Returns

VOID

- void get_input_line (char *buffer, const int bWithEcho)

5.2.1 Detailed Description

Serial - Header.

Author

Thunder Krakens

Date

February 2nd, 2016

Version

R1

5.2.2 Macro Definition Documentation

5.2.2.1 #define COM1 0x3f8

5.2.2.2 #define COM2 0x2f8

5.2.2.3 #define COM3 0x3e8

5.2.2.4 #define COM4 0x2e8

5.2.2.5 #define USER_INPUT_BUFFER_SIZE 100

5.2.2.6 #define WithEcho 1

5.2.2.7 #define WithoutEcho 0

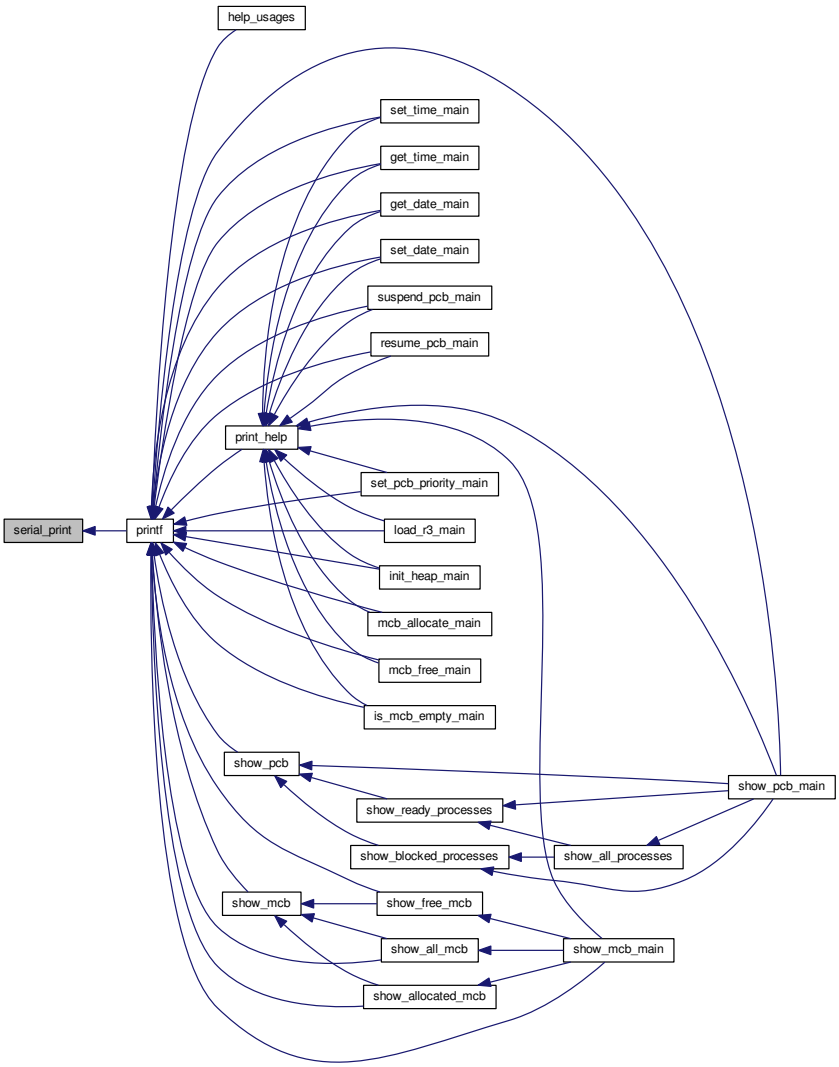
5.2.3 Function Documentation

5.2.3.1 void get_input_line (char * *buffer,* const int *bWithEcho*)

5.2.3.2 int init_serial (int *device*)

5.2.3.3 int serial_print (const char * *msg*)

Here is the caller graph for this function:



5.2.3.4 int serial_println (const char * *msg*)

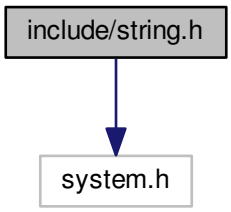
5.2.3.5 int set_serial_in (int *device*)

5.2.3.6 int set_serial_out (int *device*)

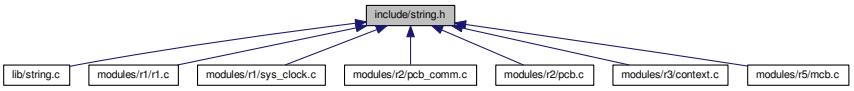
5.3 include/string.h File Reference

Many usefull functions that used for handling string.

`#include <system.h>`
Include dependency graph for string.h:



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



Functions

isspace.

Identifies if its space

Parameters

A	*constant character*

Returns

1 if it is space, otherwise return 0.

- int isspace (const char *c)

memset.

Sets region of memory

Parameters

s	*destination*
c	*byte to write*
n	*count*

Returns

the pointer to the memory space.

- void ∗ memset (void ∗s, int c, size_t n)

strcpy.

Copies one string to another.

Parameters

s1	*Destination string*
s2	*Source string*

Returns

pointer to the destination String

- char ∗ strcpy (char ∗s1, const char ∗s2)

strcat.

Concatenate the contents of one string onto another.

Parameters

s1	*Destination string*
s2	*Source string*

Returns

pointer to destination String

- char ∗ strcat (char ∗s1, const char ∗s2)

strlen.

Returns the length of a string.

Parameters

s	*String input.*

Returns

count Length of the String

- int strlen (const char ∗s)

strcmp.

String comparison.

Parameters

s1	*First string to use for the compare.*
s2	*Second string to use for the compare.*

Returns

whether they are the same or not.

- int strcmp (const char ∗s1, const char ∗s2)

strtok.

Split string into tokens.

Parameters

s1	*String*
s2	*Delimiter*

Returns

the pointer to the token.

- char ∗ strtok (char ∗s1, const char ∗s2)

atoi.

Convert an ASCII string to an integer.

Parameters

s	*String.*

Returns

The converted integer.

- int atoi (const char ∗s)

sprintf.

Generate a formatted string.

%[-x]c output a character, '-' - align right, x - the output width

%[-x]s output a string, '-' - align right, x - the output width

%[{-,+}x]d output a character, '-' - align right, '+' - align right and display '+' sign, x - the output width

%[-x]X (capital 'X') output a hexadecimal number, '-' - align right, x - the output width

note: Output width will be ignored if width is smaller than actual length.

Parameters

str	*- Output string.*
format	*- The format of the string.*
...	*- All of the additional parameters.*

Returns

vsprintf(str, format, ap) - Return the string with its format and pointer.

- int sprintf (char *str, const char *format,...)

printf.

Print out a formatted string.

%[-x]c output a character, '-' - align right, x - the output width

%[-x]s output a string, '-' - align right, x - the output width

%[{-,+}x]d output a character, '-' - align right, '+' - align right and display '+' sign, x - the output width

%[-x]X (capital 'X') output a hexadecimal number, '-' - align right, x - the output width

note: Output width will be ignored if width is smaller than actual length.

Parameters

str	*- Output string.*
format	*- The format of the string.*
...	*- All of the additional parameters.*

Returns

vsprintf(str, format, ap) - Return the string with its format and pointer.

- int printf (const char *format,...)

5.3.1 Detailed Description

Many usefull functions that used for handling string.

Author

Thunder Krakens

Date

February 2nd, 2016

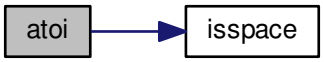
Version

R1

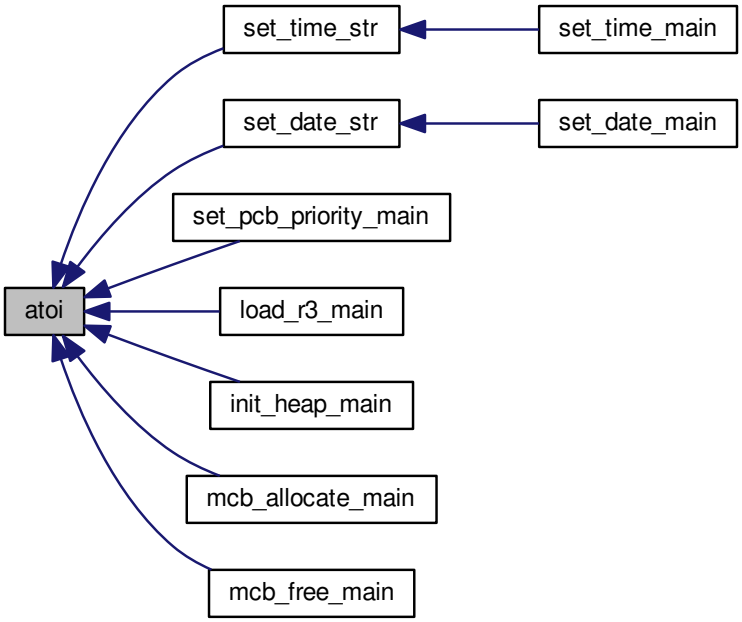
5.3.2 Function Documentation

5.3.2.1 int atoi (const char * s)

Here is the call graph for this function:

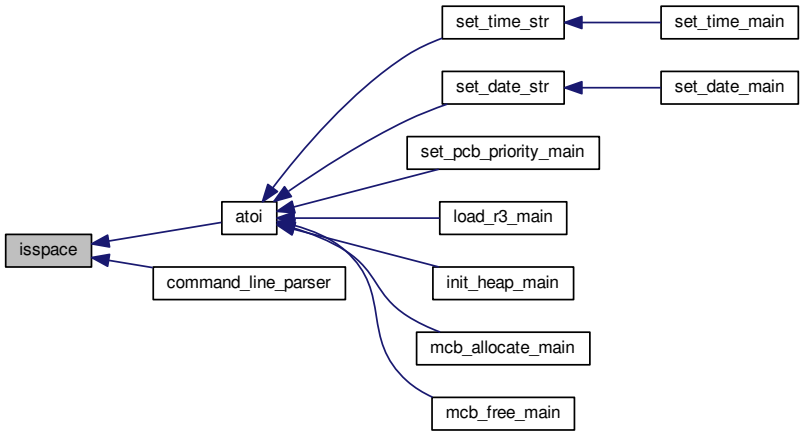


Here is the caller graph for this function:



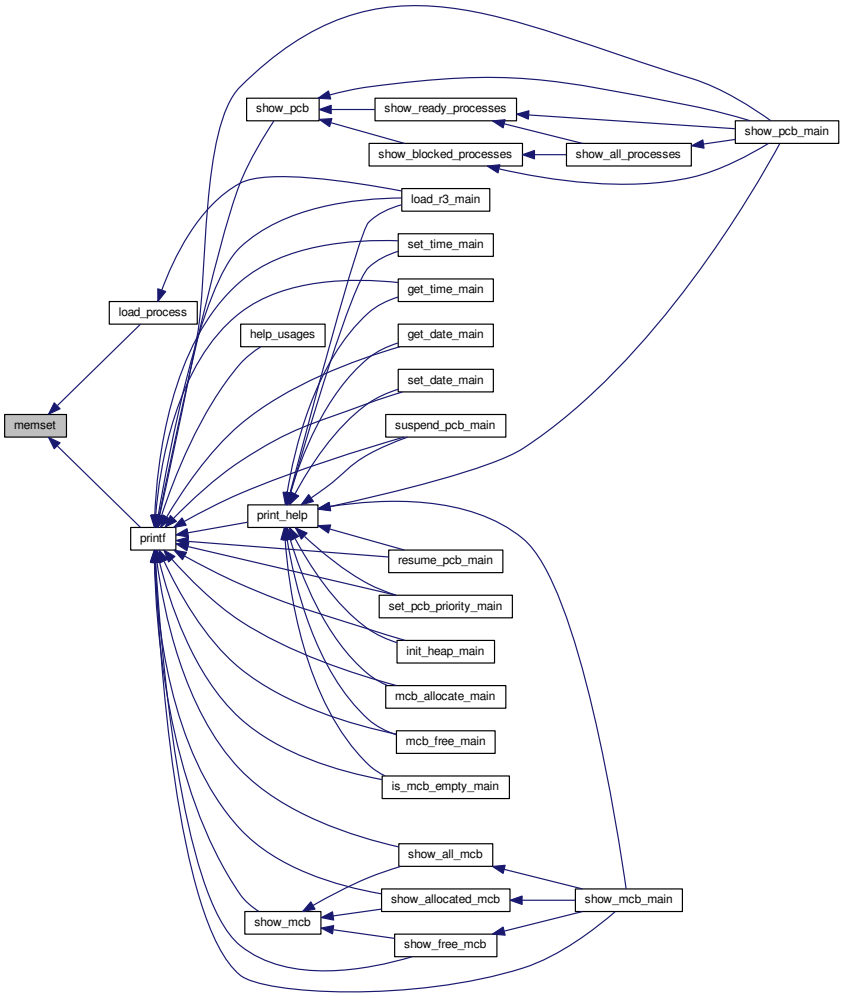
5.3.2.2 int isspace (const char * c)

Here is the caller graph for this function:



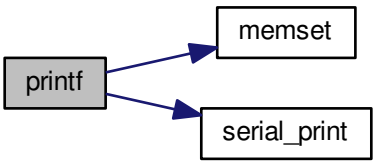
5.3.2.3 void* memset (void * *s*, int *c*, size_t *n*)

Here is the caller graph for this function:

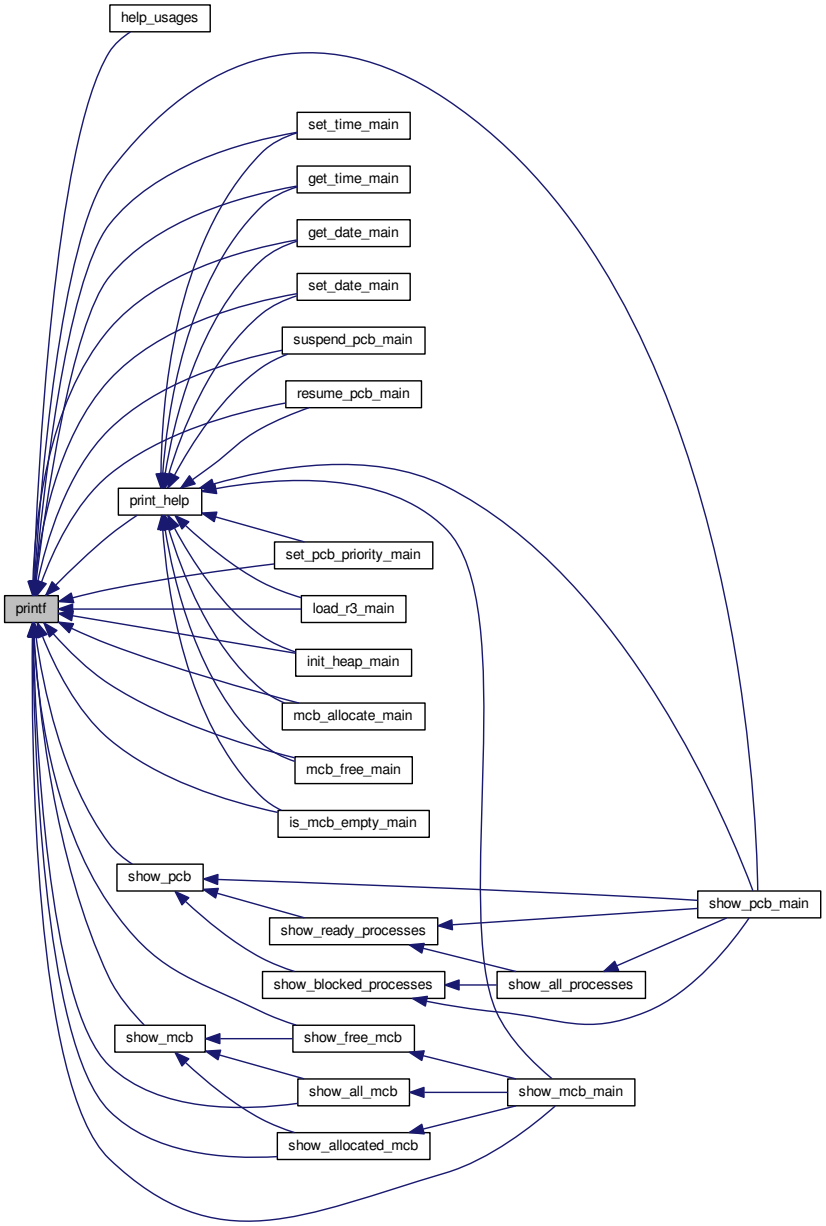


5.3.2.4 int printf (const char * *format*, ...)

Here is the call graph for this function:



Here is the caller graph for this function:

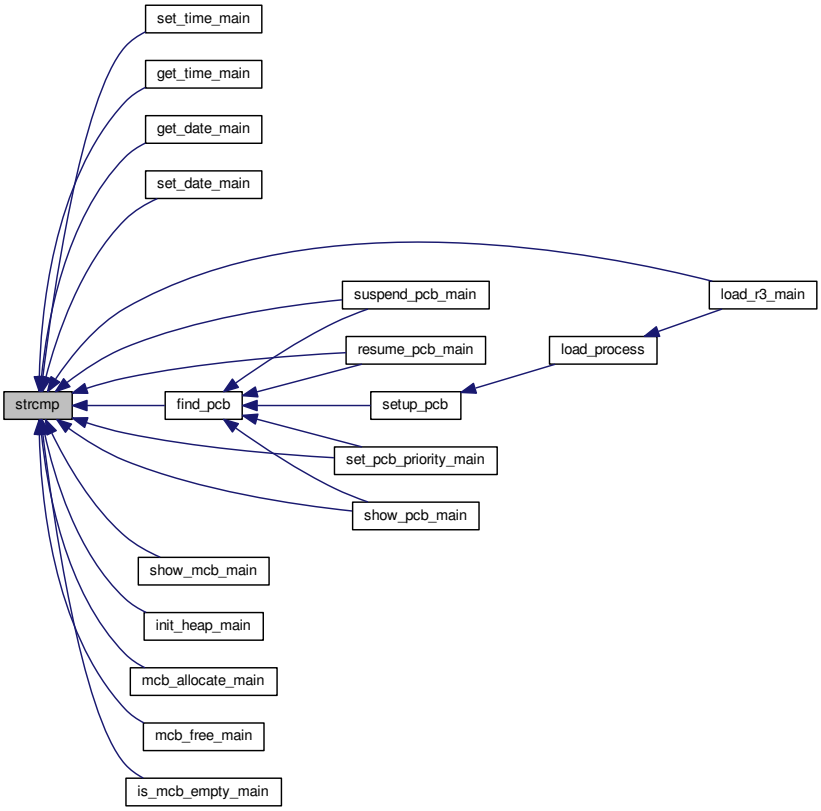


5.3.2.5 int sprintf (char * *str,* const char * *format,* ...)

5.3.2.6 char* strcat (char * *s1,* const char * *s2*)

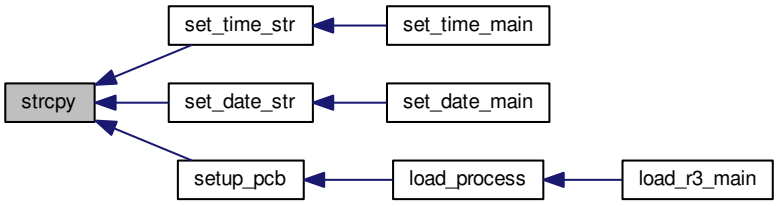
5.3.2.7 int strcmp (const char * *s1*, const char * *s2*)

Here is the caller graph for this function:



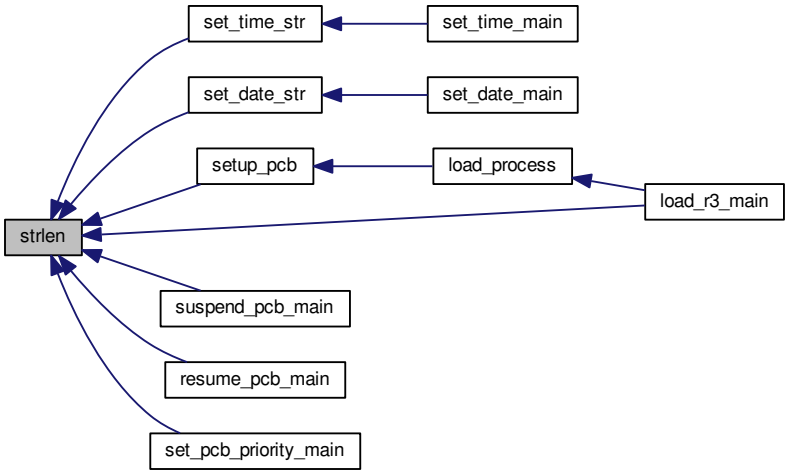
5.3.2.8 char* strcpy (char * *s1*, const char * *s2*)

Here is the caller graph for this function:



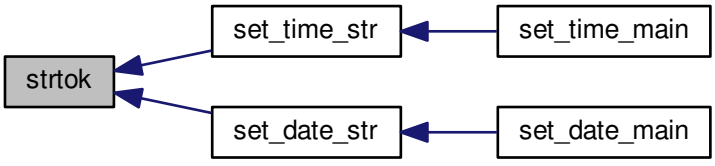
5.3.2.9 int strlen (const char * s)

Here is the caller graph for this function:



5.3.2.10 char* strtok (char * s1, const char * s2)

Here is the caller graph for this function:

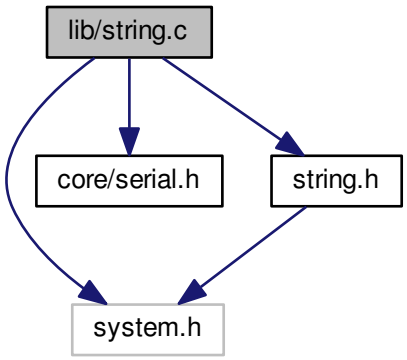


5.4 lib/string.c File Reference

Many usefull functions that used for handling string.

```
#include <system.h>
#include <core/serial.h>
#include <string.h>
```

Include dependency graph for string.c:



Functions

strlen.

Returns the length of a string.

Parameters

s	*String input.*

Returns

count Length of the String

- int strlen (const char *s)

strcpy.

Copies one string to another.

Parameters

s1	*Destination string*
s2	*Source string*

Returns

pointer to the destination String

- char * strcpy (char *s1, const char *s2)

atoi.

Convert an ASCII string to an integer.

Parameters

s	*String.*

Returns

The converted integer.

- int atoi (const char *s)

strcmp.

String comparison.

Parameters

s1	*First string to use for the compare.*
s2	*Second string to use for the compare.*

Returns

whether they are the same or not.

- int strcmp (const char *s1, const char *s2)

ParsePadding.

Parse the number for padding.

(static - Only can be access within this file).

Parameters

str	*Paddling String*
width	*Paddling Width*
DecWidth	*Width of decimal part.*
bIsRight	*Is align right.*
bHasSign	*Has + / -.*

Returns

bIsValid Returns the validity.

AddPad.

Add a certain number of paddings (static - Only can be access within this file).

Parameters

str	*In string.*
count	*Number of whitespace.*

Returns

VOID

NibbleToChar

convert a nibble into a single hexadecimal (static - Only can be access within this file)

Parameters

value	*The value of the nibble*

Returns

the character of the Hexadecimal number if valid, otherwise, return '∗'.

bytesToHexString.

Convert bytes into a hexadecimal string (static - Only can be access within this file).

Parameters

OutStr	*Output string.*
Value	*The value of bytes.*

Returns

VOID

vsprintf.

The actual function that perform the "printf" and "sprintf" function (static - Only can be access within this file).

Parameters

str	*Output string.*
format	*The format of the string.*
ap	*the pointer of the first additional parameter.*

Returns

0

sprintf.

Generate a formatted string.

%[-x]c output a character, '-' - align right, x - the output width

%[-x]s output a string, '-' - align right, x - the output width

%[{-,+}x]d output a character, '-' - align right, '+' - align right and display '+' sign, x - the output width

%[-x]X (capital 'X') output a hexadecimal number, '-' - align right, x - the output width

note: Output width will be ignored if width is smaller than actual length.

Parameters

str	*- Output string.*
format	*- The format of the string.*
...	*- All of the additional parameters.*

Returns

vsprintf(str, format, ap) - Return the string with its format and pointer.

- int sprintf (char ∗str, const char ∗format,...)

printf.

Print out a formatted string.

%[-x]c output a character, '-' - align right, x - the output width

%[-x]s output a string, '-' - align right, x - the output width

%[{-,+}x]d output a character, '-' - align right, '+' - align right and display '+' sign, x - the output width

%[-x]X (capital 'X') output a hexadecimal number, '-' - align right, x - the output width

note: Output width will be ignored if width is smaller than actual length.

Parameters

str	*- Output string.*
format	*- The format of the string.*
...	*- All of the additional parameters.*

Returns

vsprintf(str, format, ap) - Return the string with its format and pointer.

- int printf (const char *format,...)
- char * strcat (char *s1, const char *s2)
- int isspace (const char *c)
- void * memset (void *s, int c, size_t n)
- char * strtok (char *s1, const char *s2)

5.4.1 Detailed Description

Many usefull functions that used for handling string.

Author

Thunder Krakens

Date

February 2nd, 2016

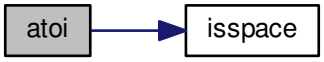
Version

R1

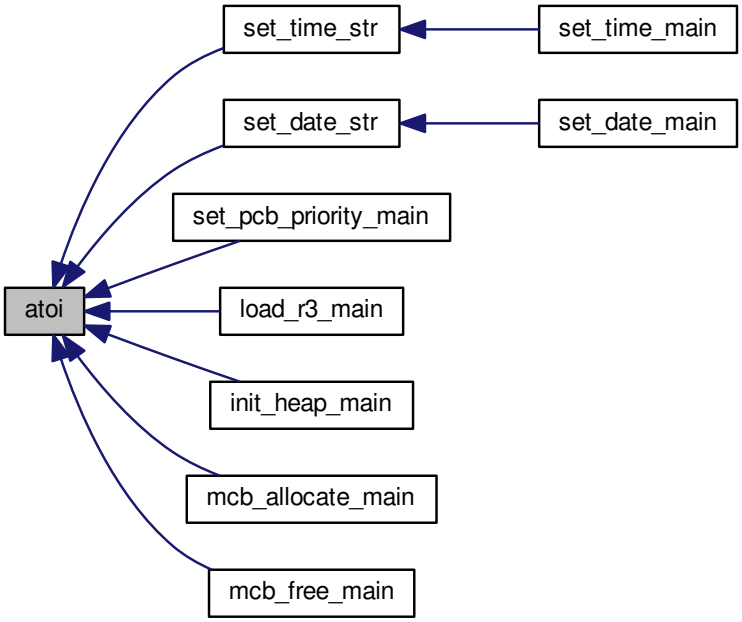
5.4.2 Function Documentation

5.4.2.1 int atoi (const char * *s*)

Here is the call graph for this function:

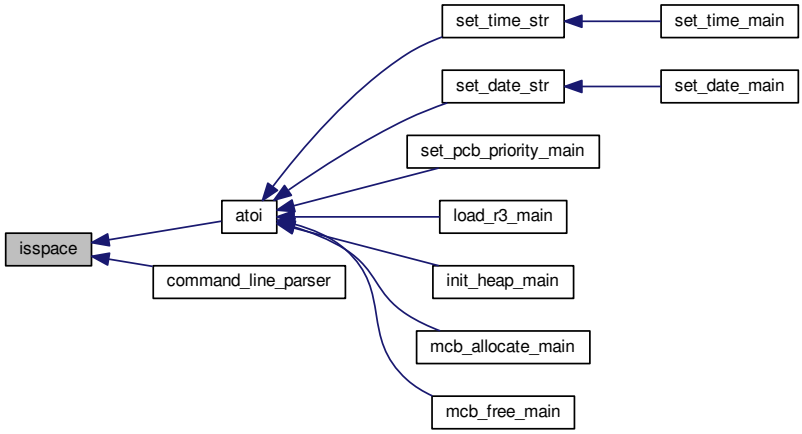


Here is the caller graph for this function:



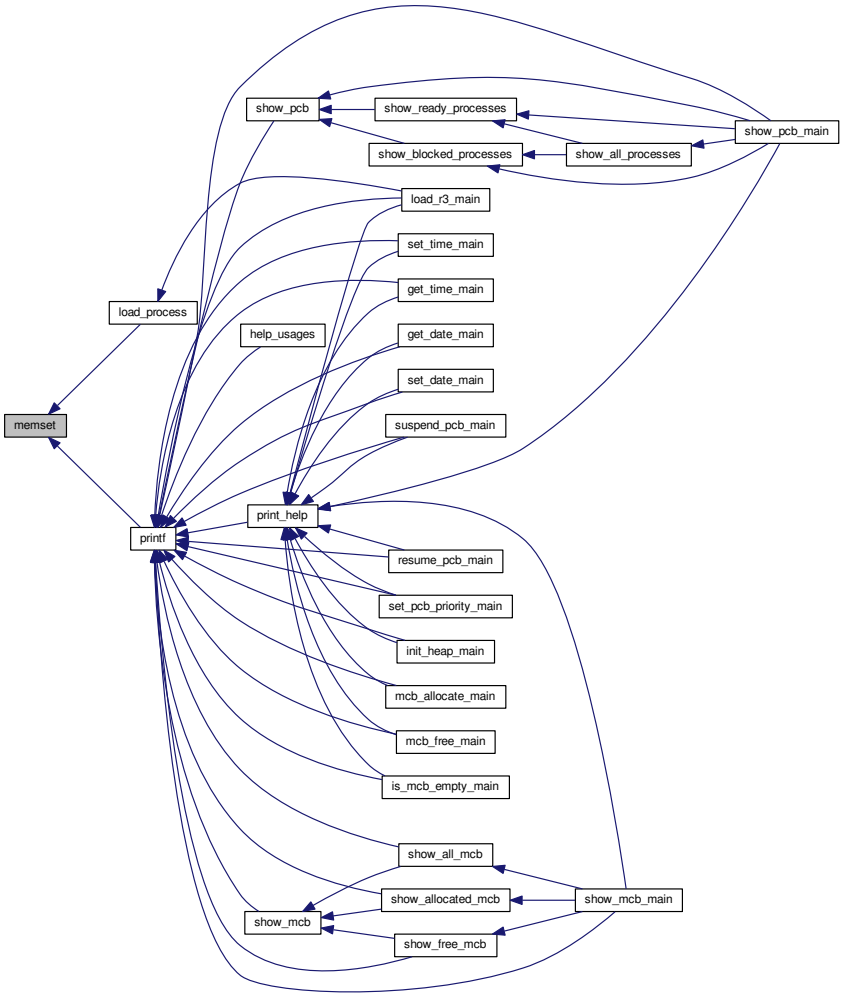
5.4.2.2 int isspace (const char * c)

Here is the caller graph for this function:



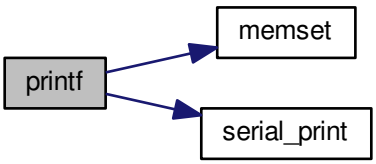
5.4.2.3 void* memset (void * *s*, int *c*, size_t *n*)

Here is the caller graph for this function:

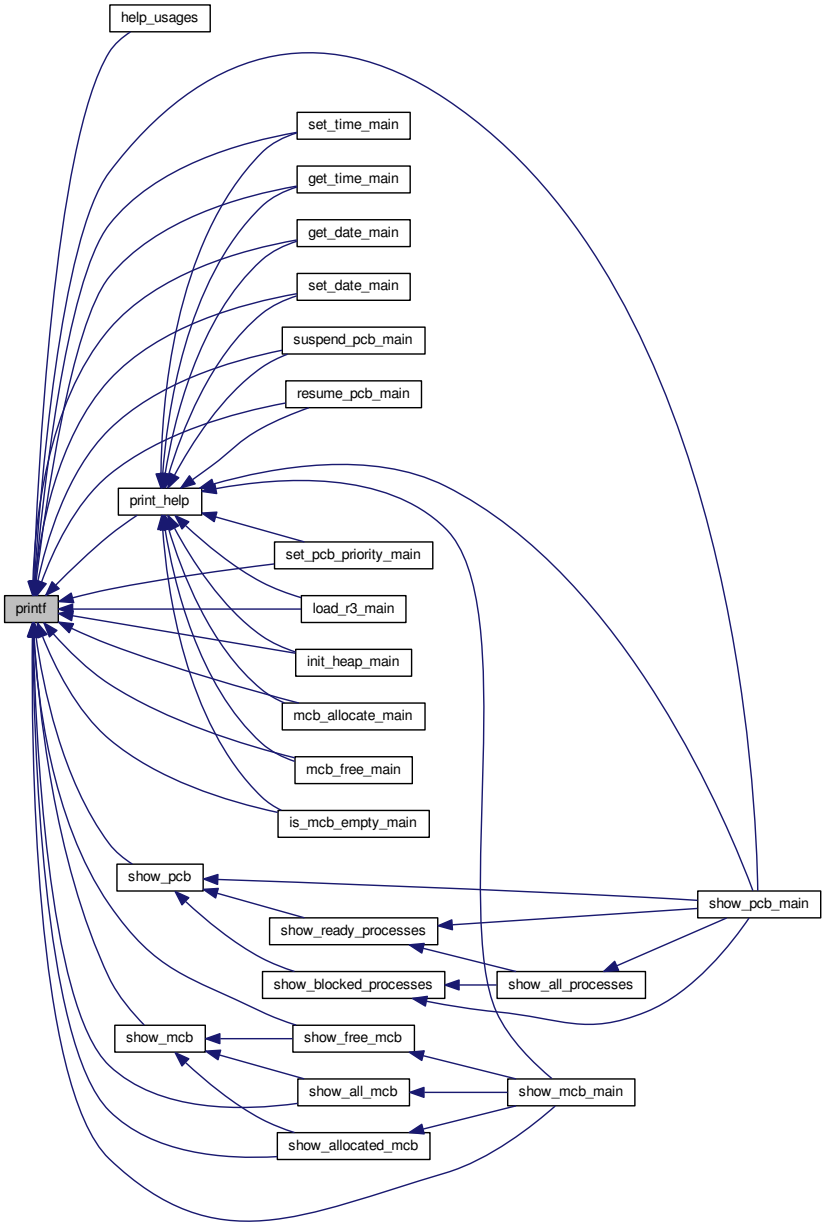


5.4.2.4 int printf (const char * *format*, ...)

Here is the call graph for this function:



Here is the caller graph for this function:

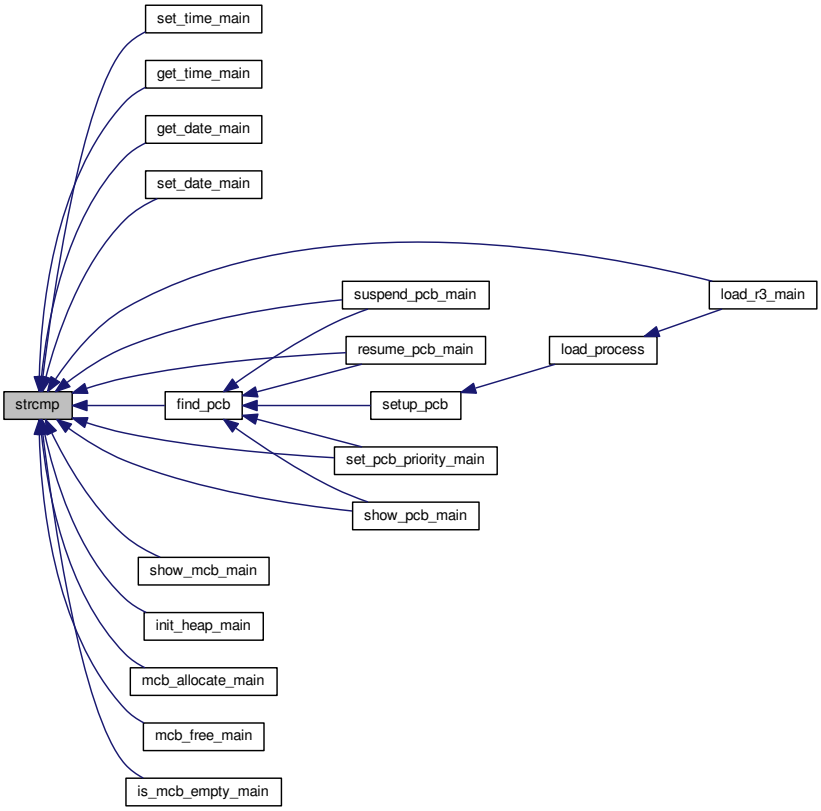


5.4.2.5 int sprintf (char * *str,* const char * *format,* ...)

5.4.2.6 char* strcat (char * *s1,* const char * *s2*)

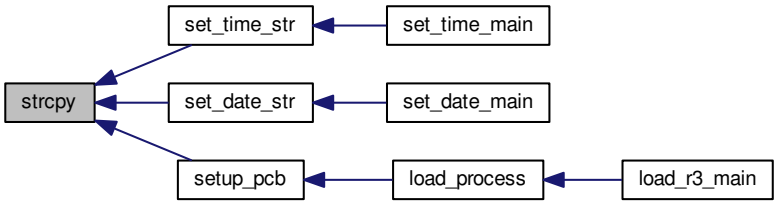
5.4.2.7 int strcmp (const char * *s1*, const char * *s2*)

Here is the caller graph for this function:



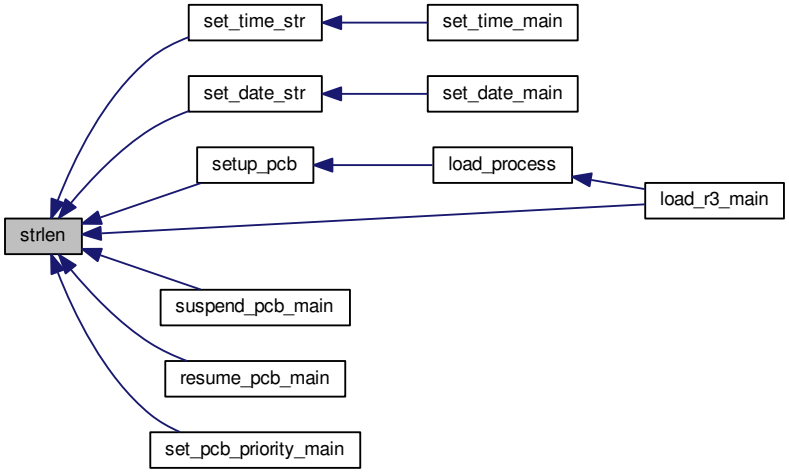
5.4.2.8 char* strcpy (char * *s1*, const char * *s2*)

Here is the caller graph for this function:



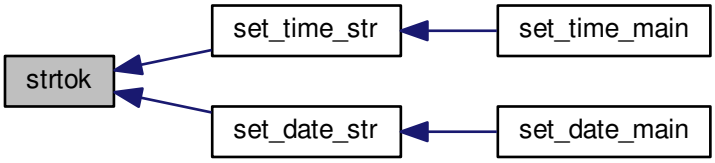
5.4.2.9 int strlen (const char * s)

Here is the caller graph for this function:



5.4.2.10 char* strtok (char * s1, const char * s2)

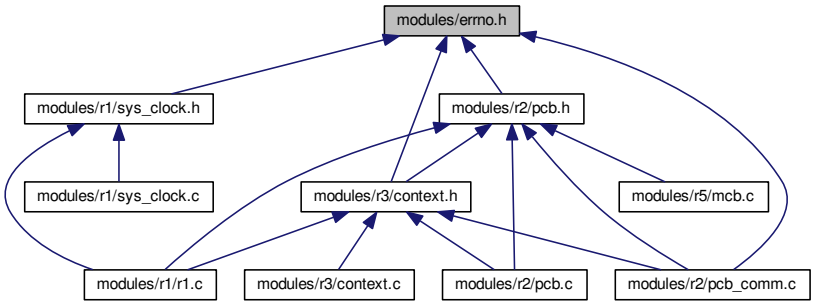
Here is the caller graph for this function:



5.5 modules/errno.h File Reference

This file contains the type of errors. The error can be from invalid paramter passed to a function, or invalid input format.

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



Macros

- #define E_NOERROR 0
- #define E_INVPARA 1
- #define E_INVSTRF 2
- #define E_INVUSRI 3
- #define E_FREEMEM 4

  Error we cannot actually free the memory space since the student_free had not been implemented before R5.
- #define E_NULL_PTR 5

  A NULL Pointer Error.
- #define E_EMPTPCB 6

  The pcb queue is empty.
- #define E_PCB_SYS 7
- #define E_PROGERR 99

Typedefs

error_t.

The datetype that holds the error code.

- typedef unsigned int error_t

5.5.1 Detailed Description

This file contains the type of errors. The error can be from invalid paramter passed to a function, or invalid input format.

Author

Thunder Krakens

Date

February 7nd, 2016

Version

R2

5.5.2 Macro Definition Documentation

5.5.2.1 #define E_EMPTPCB 6

The pcb queue is empty.

5.5.2.2 #define E_FREEMEM 4

Error we cannot actually free the memory space since the student_free had not been implemented before R5.

5.5.2.3 #define E_INVPARA 1

5.5.2.4 #define E_INVSTRF 2

5.5.2.5 #define E_INVUSRI 3

5.5.2.6 #define E_NOERROR 0

5.5.2.7 #define E_NULL_PTR 5

A NULL Pointer Error.

5.5.2.8 #define E_PCB_SYS 7

5.5.2.9 #define E_PROGERR 99

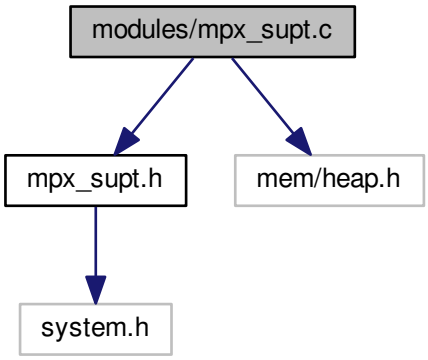
5.5.3 Typedef Documentation

5.5.3.1 typedef unsigned int error_t

5.6 modules/mpx_supt.c File Reference

```
#include "mpx_supt.h"
#include <mem/heap.h>
```

Include dependency graph for mpx_supt.c:



Functions

- int sys_req (int op_code)
- void mpx_init (int cur_mod)
- void sys_set_malloc (u32int(∗func)(u32int))
- void sys_set_free (int(∗func)(void ∗))
- void ∗ sys_alloc_mem (u32int size)
- int sys_free_mem (void ∗ptr)
- void idle ()
- int get_op_code ()

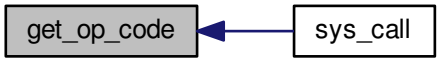
Variables

- param params
- int current_module = -1
- u32int(∗ student_malloc)(u32int)
- int(∗ student_free)(void ∗)

5.6.1 Function Documentation

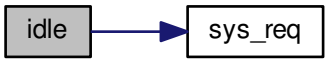
5.6.1.1 int get_op_code ( )

Here is the caller graph for this function:



5.6.1.2 void idle ( )

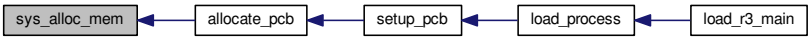
Here is the call graph for this function:



5.6.1.3 void mpx_init (int *cur_mod*)

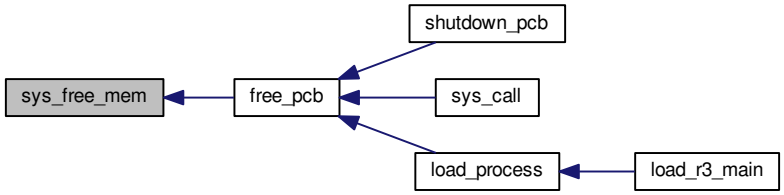
5.6.1.4 void* sys_alloc_mem (u32int *size*)

Here is the caller graph for this function:



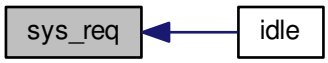
5.6.1.5 int sys_free_mem (void * *ptr*)

Here is the caller graph for this function:



5.6.1.6 int sys_req (int *op_code*)

Here is the caller graph for this function:



5.6.1.7 void sys_set_free (int(*)(void *) *func*)

5.6.1.8 void sys_set_malloc (u32int(*)(u32int) *func*)

5.6.2 Variable Documentation

5.6.2.1 int current_module = -1

5.6.2.2 param params

5.6.2.3 int(* student_free)(void *)

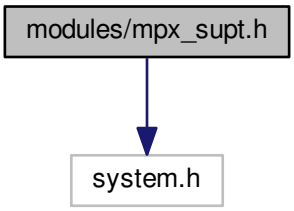
5.6.2.4 u32int(* student_malloc)(u32int)

5.7 modules/mpx_supt.h File Reference

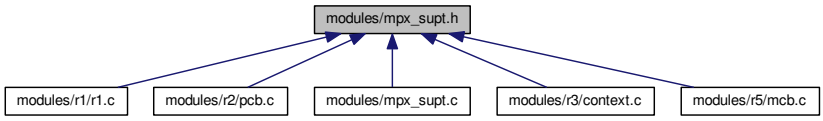
MPX System Supplementaries.

```
#include <system.h>
```

Include dependency graph for mpx_supt.h:



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



Data Structures

- struct param

  A structure to represent interrupt.

Macros

- #define EXIT 0
- #define IDLE 1
- #define READ 2
- #define WRITE 3
- #define MODULE_R1 0
- #define MODULE_R2 1
- #define MODULE_R3 2
- #define MODULE_R4 4
- #define MODULE_R5 8

Functions

sys_req

Generate interrupt 60H

Parameters

int	*op_code (IDLE)*

- int sys_req (int op_code)

mpx_init

Initialize MPX support software

Parameters

int	*cur_mod (symbolic constants MODULE_R1, MODULE_R2, etc*

- void mpx_init (int cur_mod)

set_malloc

Sets the memory allocation function for sys_alloc_mem

Parameters

Function	*pointer*

- void sys_set_malloc (u32int(∗func)(u32int))

set_free

Sets the memory free function for sys_free_mem

Parameters

s1-destination,s2-source	

- void sys_set_free (int(∗func)(void ∗))

sys_alloc_mem

Allocates a block of memory (similar to malloc)

Parameters

Number	*of bytes to allocate*

- void ∗ sys_alloc_mem (u32int size)

sys_free_mem

Frees memory

Parameters

Pointer	*to block of memory to free*

- int sys_free_mem (void ∗ptr)

idle

The idle process

Parameters

None	

- void idle ()

get_op_code

Returns the interrupt's operation code

Parameters

None	

- int get_op_code ()

Variables

- typedef __attribute__

5.7.1 Detailed Description

MPX System Supplementaries.

Author

Thunder Krakens

Date

March 18, 2016

Version

R3

5.7.2 Macro Definition Documentation

5.7.2.1 #define EXIT 0

5.7.2.2 #define IDLE 1

5.7.2.3 #define MODULE_R1 0

5.7.2.4 #define MODULE_R2 1

5.7.2.5 #define MODULE_R3 2

5.7.2.6 #define MODULE_R4 4

5.7.2.7 #define MODULE_R5 8

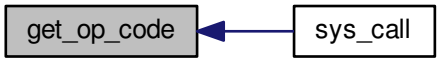
5.7.2.8 #define READ 2

5.7.2.9 #define WRITE 3

5.7.3 Function Documentation

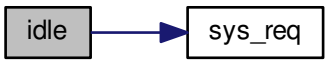
5.7.3.1 int get_op_code ( )

Here is the caller graph for this function:



5.7.3.2 void idle ( )

Here is the call graph for this function:



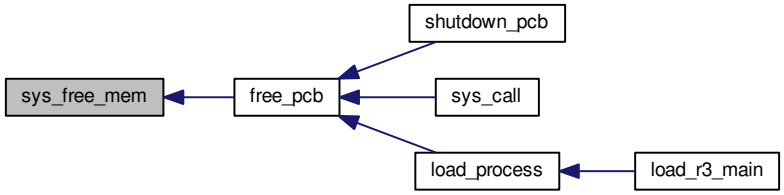
5.7.3.3 void mpx_init (int *cur_mod*)

5.7.3.4 void* sys_alloc_mem (u32int *size*)

Here is the caller graph for this function:

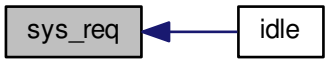
5.7.3.5 int sys_free_mem (void * *ptr*)

Here is the caller graph for this function:



5.7.3.6 int sys_req (int *op_code*)

Here is the caller graph for this function:



5.7.3.7 void sys_set_free (int(*)(void *) *func*)

5.7.3.8 void sys_set_malloc (u32int(*)(u32int) *func*)

5.7.4 Variable Documentation

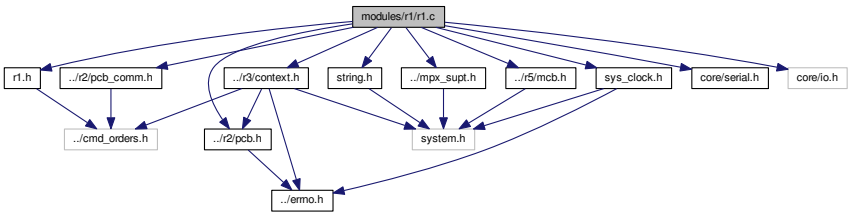
5.7.4.1 enum process_suspended __attribute__

5.8 modules/r1/r1.c File Reference

The commandhander and functions associations for Module R1.

```
#include "r1.h"
#include "sys_clock.h"
#include <string.h>
#include <core/serial.h>
#include <core/io.h>
#include "../r2/pcb_comm.h"
#include "../r2/pcb.h"
#include "../mpx_supt.h"
#include "../r3/context.h"
#include "../r5/mcb.h"
```

Include dependency graph for r1.c:



Data Structures

- struct function_name

  A structure to represent each function.

Macros

- #define MAX_ARGC 50
- #define MOD_VERSION "R5"
- #define COMPLETION "04/08/2016"
- #define MAX_HISTORY 10

Functions

exe_function.

Executes the specific fucntion.

Parameters

argc	*The number of tokens.*
argv	*The array of tokens.*

Returns

0

version

displays the version of the system currently running.

Parameters

argc	*The number of tokens.*
argv	*The array of tokens.*

Returns

0

shutdown

Closes all functions, and shuts down the system.

Parameters

argc	*The number of tokens found.*
argv	*The array of tokens.*

Returns

0 for shutdown, 1 for keep running.

help_usages

shows usage message for each function.

Parameters

start_from	*the index of the beginning function.*

Returns

0

- int help_usages (enum comm_type type)

help_function

displays help text for all functions.

Parameters

argc	*The number of tokens.*
argv	*The array of tokens.*

Returns

0

commhand

Accepts and handles commands from the user.

Returns

0

- void commhand ()

command_line_parser

Splits the complete command line into tokens by space, single quote, or double quote.

Parameters

CmdStr	*The complete input command.*
argc	*The number of tokens found.*
argv	*The array of tokens.*
MaxArgNum	*The maximum number of tokens that array can hold.*
MaxStrLen	*The maximum length of each token that string can hold.*

Returns

void

- void command_line_parser (const char ∗CmdStr, int ∗argc, char ∗∗argv, const int MaxArgNum, const int MaxStrLen)

print_help

prints the help message of a certain function that specified by the index number

Parameters

function_index	*The index number of that function.*

Returns

void

- void print_help (const int function_index)

Variables

- NotWriting
- NormalWriting
- DoubleQuoteWriting
- SingleQuoteWriting

CommandParserStat

The status of the command parser

- enum CommandPaserStat
- enum CommandPaserStat __attribute__ ((packed))

5.8.1 Detailed Description

The commandhander and functions associations for Module R1.

Author

Thunder Krakens

Date

April 8th, 2016

Version

R5

5.8.2 Macro Definition Documentation

5.8.2.1 #define COMPLETION "04/08/2016"

5.8.2.2 #define MAX_ARGC 50

5.8.2.3 #define MAX_HISTORY 10

5.8.2.4 #define MOD_VERSION "R5"

5.8.3 Enumeration Type Documentation

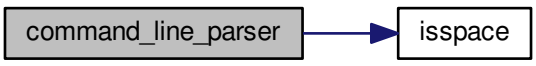
5.8.3.1 enum CommandPaserStat

5.8.4 Function Documentation

5.8.4.1 enum CommandPaserStat __attribute__ ((packed))

5.8.4.2 void command_line_parser (const char * *CmdStr,* int * *argc,* char ** *argv,* const int *MaxArgNum,* const int *MaxStrLen*)

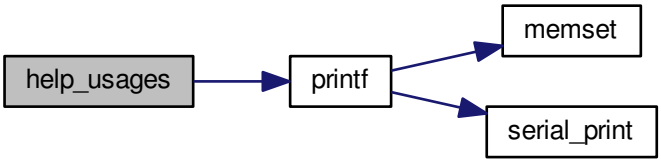
Here is the call graph for this function:



5.8.4.3 void commhand ( )

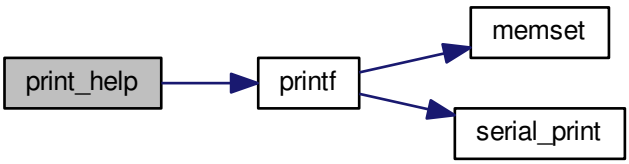
5.8.4.4 int help_usages (enum comm_type *type*)

Here is the call graph for this function:

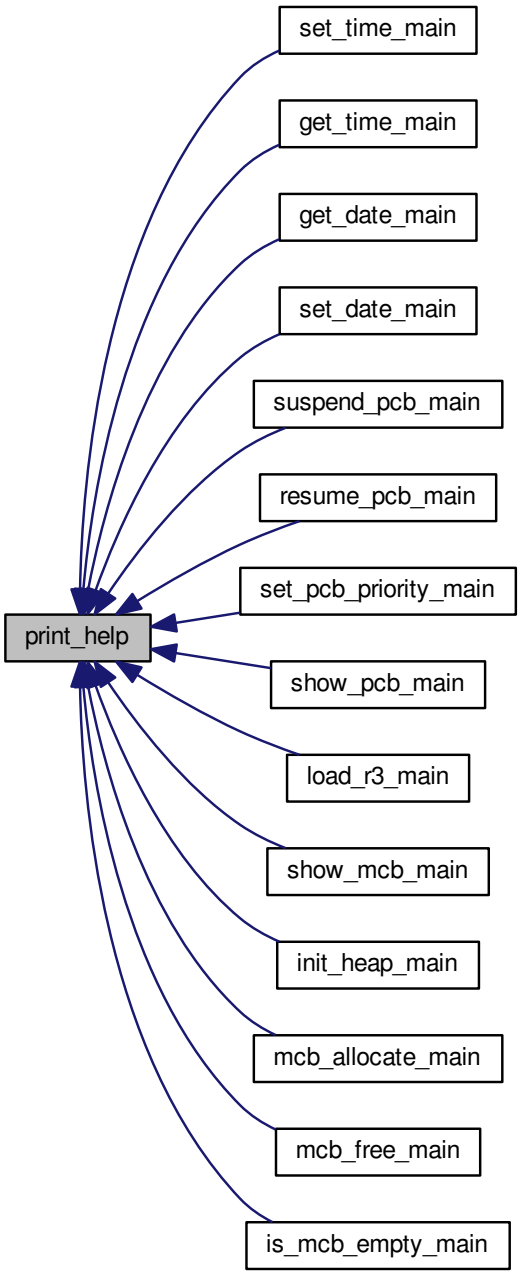


5.8.4.5 void print_help (const int *function_index*)

Here is the call graph for this function:



Here is the caller graph for this function:



5.8.5 Variable Documentation

5.8.5.1 DoubleQuoteWriting

5.8.5.2 NormalWriting

5.8.5.3 NotWriting

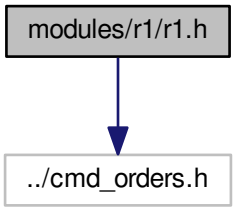
5.8.5.4 SingleQuoteWriting

5.9 modules/r1/r1.h File Reference

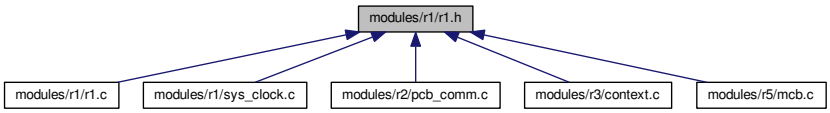
The command handler and functions associations for Module R1.

```
#include "../cmd_orders.h"
```

Include dependency graph for r1.h:



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



Enumerations

- enum comm_type

Functions

- enum comm_type __attribute__ ((packed))

commhand

Accepts and handles commands from the user.

Returns

VOID

- void commhand ()

command_line_parser

Splits the complete command line into tokens by space, single quote, or double quote.

Parameters

CmdStr	*The complete input command.*
argc	*The number of tokens found.*
argv	*The array of tokens.*
MaxArgNum	*The maximum number of tokens that array can hold.*
MaxStrLen	*The maximum length of each token that string can hold.*

Returns

void

- void command_line_parser (const char ∗CmdStr, int ∗argc, char ∗∗argv, const int MaxArgNum, const int MaxStrLen)

print_help

prints the help message of a certain function that specified by the index number

Parameters

function_index	*The index number of that function.*

Returns

void

- void print_help (const int function_index)
- int help_usages (enum comm_type type)

Variables

- mpx
- pcb
- mcb
- help

5.9.1 Detailed Description

The command handler and functions associations for Module R1.

Author

Thunder Krakens

Date

March 17, 2016

Version

R3 & R4

5.9.2 Enumeration Type Documentation

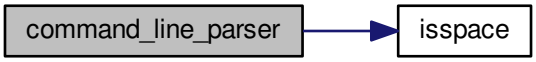
5.9.2.1 enum comm_type

5.9.3 Function Documentation

5.9.3.1 enum comm_type __attribute__ ((packed))

5.9.3.2 void command_line_parser (const char * *CmdStr,* int * *argc,* char ** *argv,* const int *MaxArgNum,* const int *MaxStrLen*)

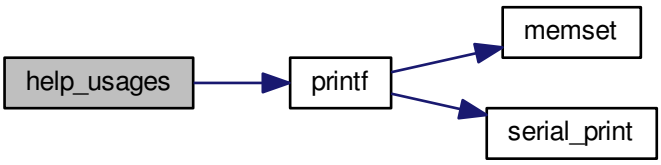
Here is the call graph for this function:



5.9.3.3 void commhand ( )

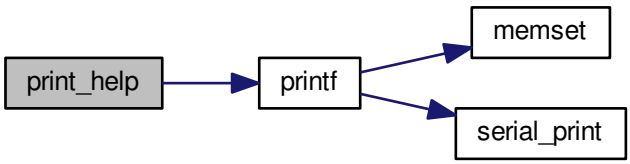
5.9.3.4 int help_usages (enum comm_type *type*)

Here is the call graph for this function:

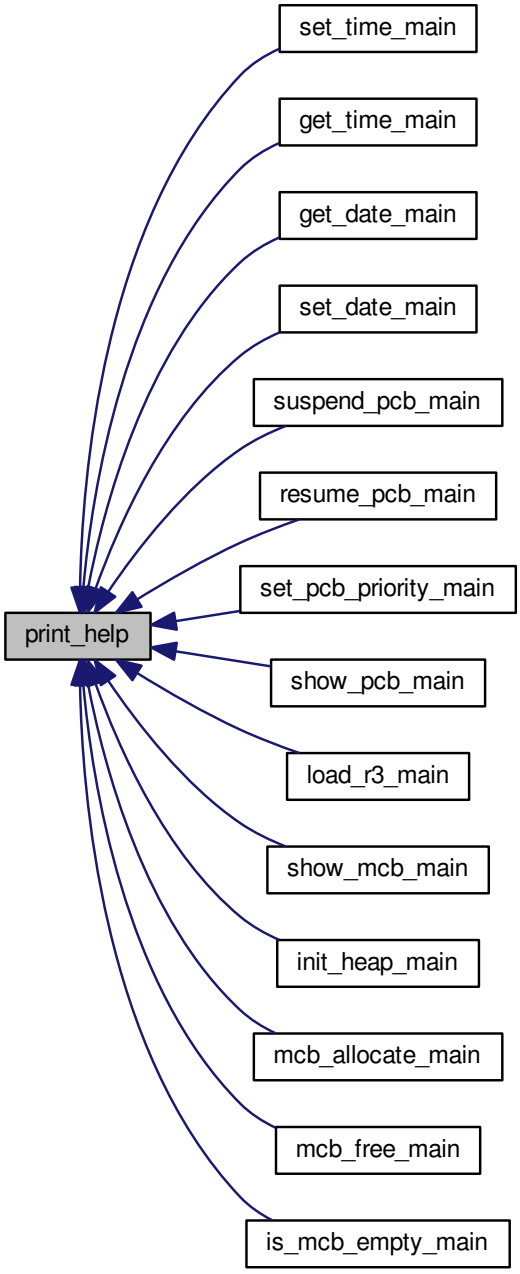


5.9.3.5 void print_help (const int *function_index*)

Here is the call graph for this function:



Here is the caller graph for this function:



5.9.4 Variable Documentation

5.9.4.1 help

5.9.4.2 mcb

5.9.4.3 mpx

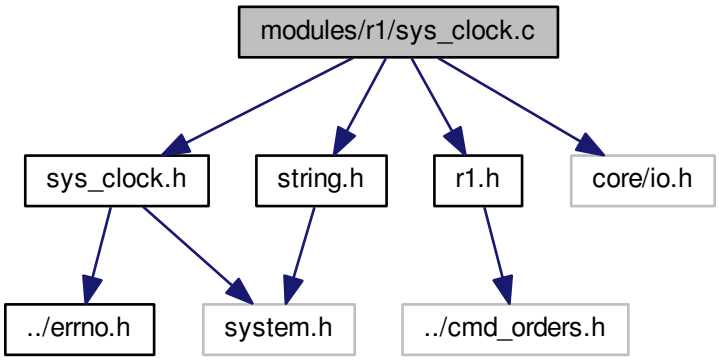
5.9.4.4 pcb

5.10 modules/r1/sys_clock.c File Reference

The main file that manipulates and controls the system's clock.

```
#include "sys_clock.h"
#include "r1.h"
#include <string.h>
#include <core/io.h>
```

Include dependency graph for sys_clock.c:



Macros

- #define RTC_INDEX_SECOND 0x00
- #define RTC_INDEX_SECOND_ALARM 0x01
- #define RTC_INDEX_MINUTE 0x02
- #define RTC_INDEX_MINUTE_ALARM 0x03
- #define RTC_INDEX_HOUR 0x04
- #define RTC_INDEX_HOUR_ALARM 0x05
- #define RTC_INDEX_DAY_WEEK 0x06
- #define RTC_INDEX_DAY_MONTH 0x07
- #define RTC_INDEX_MONTH 0x08
- #define RTC_INDEX_YEAR 0x09

Functions

set_time_main.

Sets the time for the system.

Parameters

argc	*The number of tokens found.*
argv	*The array of tokens.*

Returns

0

- int set_time_main (int argc, char **argv)

get_time_main.

Retrieves system's current time.

Parameters

argc	*The number of tokens found.*
argv	*The array of tokens.*

Returns

0

- int get_time_main (int argc, char **argv)

is_digit

determines if a character represents a digit.

Parameters

ch	*The character*

Returns

1 if it is digit, otherwise returns 0.

set_time_str.

Sets the time for the system by string.

Parameters

timeStr	*The string type of current Time.*

Returns

0 if there is no error, otherwise return a error code.

- error_t set_time_str (const char *timeStr)

get_time.

Retrieves system's current time and date.

Parameters

dateTimeValues	*The value of current time and date*

Returns

VOID

- void get_time (date_time *dateTimeValues)

set_time.

Sets the time for the system by date_time struct.

Parameters

dateTimeValues	*The struct that holds the time values.*

Returns

0 if there is no error, otherwise return a error code.

- error_t set_time (const date_time *dateTimeValues)

get_date.

Retrieves system's current date.

Parameters

dateTimeValues	*The struct that holds the value of current date*

Returns

VOID

- void get_date (date_time *dateTimeValues)

is_date_value_valid.

Check if the date specified is valid, which means year should between 1970 ∼ 1969, month should between 1 ∼ 12, while the range of the day is based on the month and year.

Parameters

year	*The value of the year*
mon	*The value of the month*
day	*The value of the day of month*

Returns

VOID

set_date.

Sets the date of the system.

Parameters

dateTimeValues	*The struct that holds the value of date*

Returns

0 if there is no error, otherwise return a error code.

- error_t set_date (const date_time ∗dateTimeValues)

get_date_main.

Retrieves system's current date.

Parameters

argc	*The number of tokens.*
argv	*The array of tokens.*

Returns

0

- int get_date_main (int argc, char **argv)

set_date_str.

Sets the date for the system by string.

Parameters

str	*The string type of current date.*

Returns

0 if there is no error, otherwise return a error code.

- int set_date_str (const char *str)

set_date_main.

Sets system's date.

Parameters

argc	*The number of tokens.*
argv	*The array of tokens.*

Returns

0

- int set_date_main (int argc, char **argv)

5.10.1 Detailed Description

The main file that manipulates and controls the system's clock.

Author

Thunder Krakens

Date

February 2nd, 2016

Version

R1

5.10.2 Macro Definition Documentation

5.10.2.1 #define RTC_INDEX_DAY_MONTH 0x07

5.10.2.2 #define RTC_INDEX_DAY_WEEK 0x06

5.10.2.3 #define RTC_INDEX_HOUR 0x04

5.10.2.4 #define RTC_INDEX_HOUR_ALARM 0x05

5.10.2.5 #define RTC_INDEX_MINUTE 0x02

5.10.2.6 #define RTC_INDEX_MINUTE_ALARM 0x03

5.10.2.7 #define RTC_INDEX_MONTH 0x08

5.10.2.8 #define RTC_INDEX_SECOND 0x00

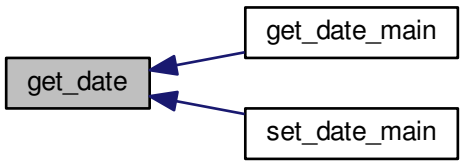
5.10.2.9 #define RTC_INDEX_SECOND_ALARM 0x01

5.10.2.10 #define RTC_INDEX_YEAR 0x09

5.10.3 Function Documentation

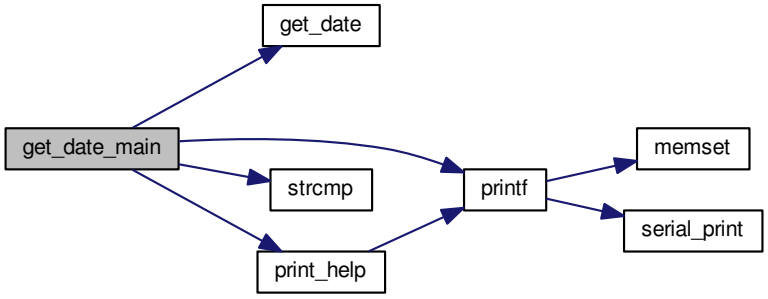
5.10.3.1 void get_date (date_time * *dateTimeValues*)

Here is the caller graph for this function:



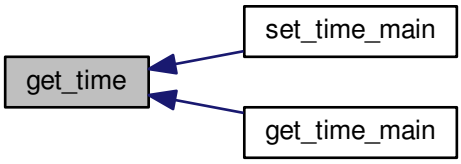
5.10.3.2 int get_date_main (int *argc*, char ** *argv*)

Here is the call graph for this function:



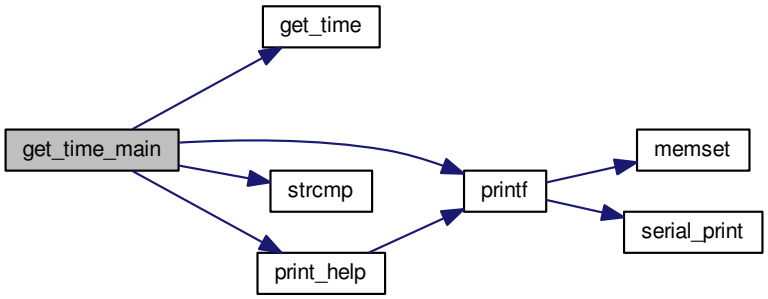
5.10.3.3 void get_time (date_time * *dateTimeValues*)

Here is the caller graph for this function:



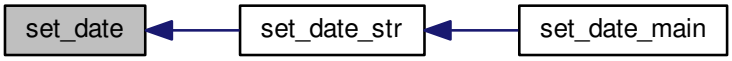
5.10.3.4 int get_time_main (int *argc*, char ** *argv*)

Here is the call graph for this function:



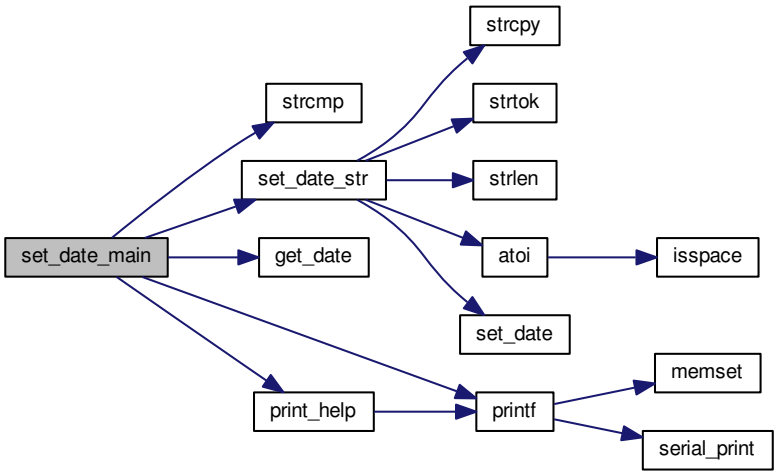
5.10.3.5 error_t set_date (const date_time * *dateTimeValues*)

Here is the caller graph for this function:



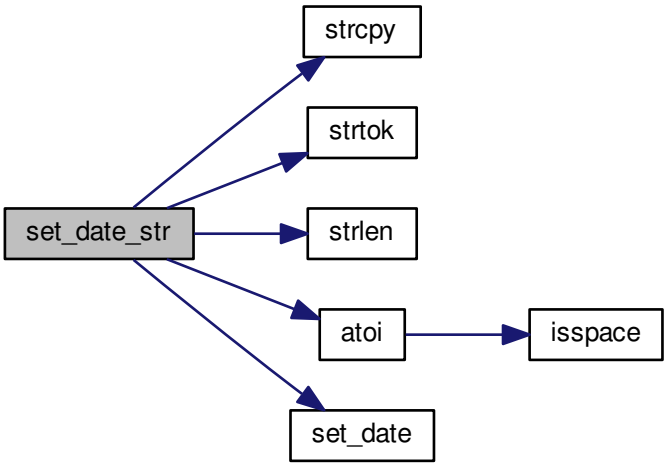
5.10.3.6 int set_date_main (int *argc*, char ** *argv*)

Here is the call graph for this function:



5.10.3.7 int set_date_str (const char * *str*)

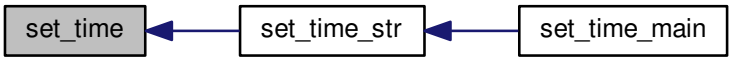
Here is the call graph for this function:



Here is the caller graph for this function:

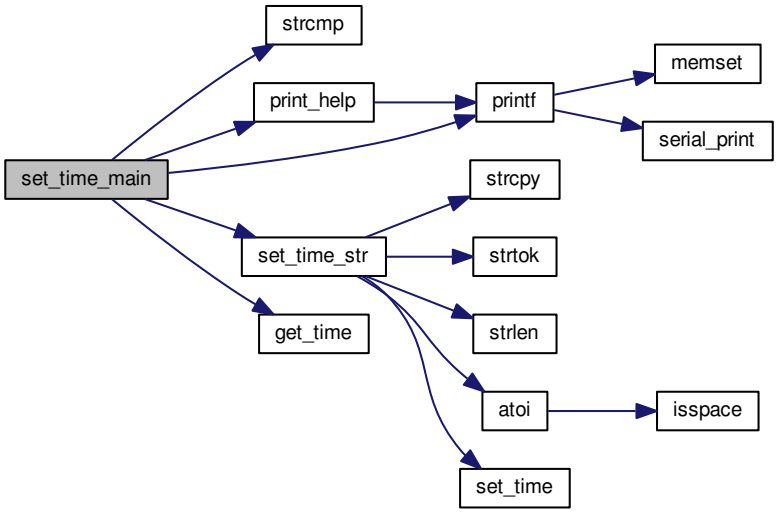
5.10.3.8 error_t set_time (const date_time * *dateTimeValues*)

Here is the caller graph for this function:



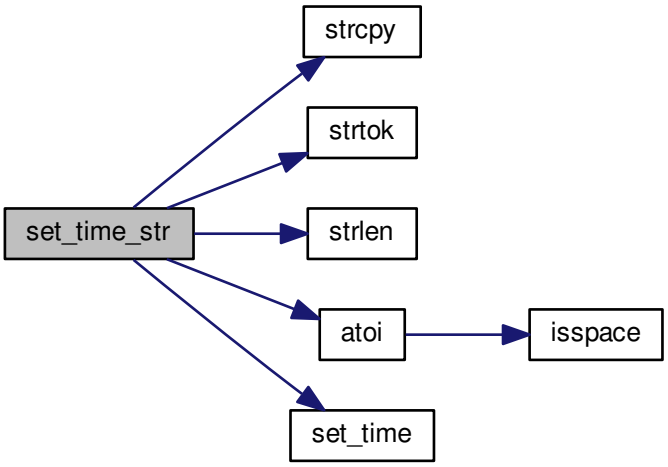
5.10.3.9 int set_time_main (int *argc*, char ** *argv*)

Here is the call graph for this function:



5.10.3.10 error_t set_time_str (const char * *timeStr*)

Here is the call graph for this function:



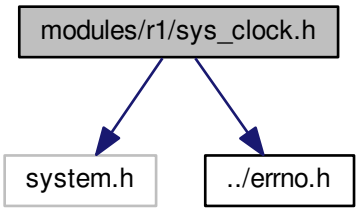
Here is the caller graph for this function:

5.11 modules/r1/sys_clock.h File Reference

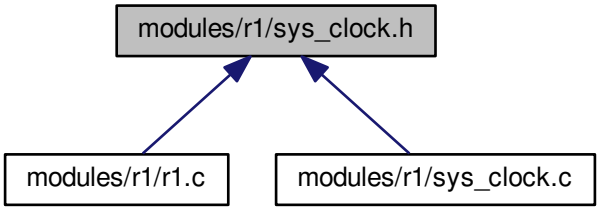
The main file that manipulates and controls the system's clock.

```
#include <system.h>
#include "../errno.h"
```

Include dependency graph for sys_clock.h:



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



Functions

set_time_main.

Sets the time for the system.

Parameters

argc	*The number of tokens found.*
argv	*The array of tokens.*

Returns

0

- int set_time_main (int argc, char **argv)

get_time_main.

Retrieves system's current time.

Parameters

argc	*The number of tokens found.*
argv	*The array of tokens.*

Returns

0

- int get_time_main (int argc, char **argv)

set_time_str.

Sets the time for the system by string.

Parameters

timeStr	*The string type of current Time.*

Returns

0 if there is no error, otherwise return a error code.

- error_t set_time_str (const char *timeStr)

get_time.

Retrieves system's current time and date.

Parameters

dateTimeValues	*The value of current time and date*

Returns

VOID

- void get_time (date_time *dateTimeValues)

set_time.

Sets the time for the system by date_time struct.

Parameters

dateTimeValues	*The struct that holds the time values.*

Returns

0 if there is no error, otherwise return a error code.

- error_t set_time (const date_time *dateTimeValues)

set_date_main.

Sets system's date.

Parameters

argc	*The number of tokens.*
argv	*The array of tokens.*

Returns

0

- int set_date_main (int argc, char **argv)

get_date_main.

Retrieves system's current date.

Parameters

argc	*The number of tokens.*
argv	*The array of tokens.*

Returns

0

- int get_date_main (int argc, char **argv)

get_date.

Retrieves system's current date.

Parameters

dateTimeValues	*The struct that holds the value of current date*

Returns

VOID

- void get_date (date_time *dateTimeValues)

set_date_str.

Sets the date for the system by string.

Parameters

str	*The string type of current date.*

Returns

0 if there is no error, otherwise return a error code.

- int set_date_str (const char *str)

set_date.

Sets the date of the system.

Parameters

dateTimeValues	*The struct that holds the value of date*

Returns

0 if there is no error, otherwise return a error code.

- error_t set_date (const date_time *dateTimeValues)

5.11.1 Detailed Description

The main file that manipulates and controls the system's clock.

Author

Thunder Krakens

Date

February 2nd, 2016

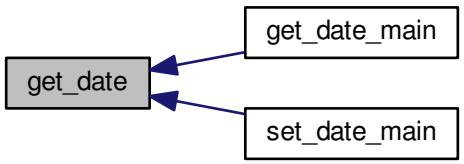
Version

R1

5.11.2 Function Documentation

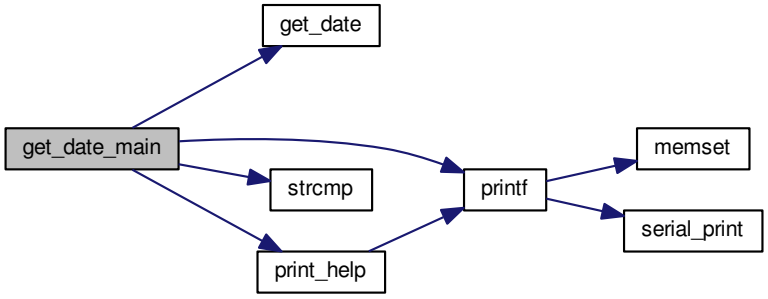
5.11.2.1 void get_date (date_time * *dateTimeValues*)

Here is the caller graph for this function:



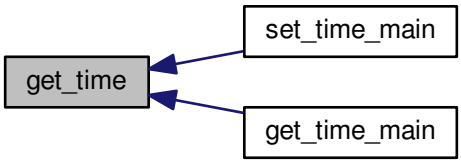
5.11.2.2 int get_date_main (int *argc*, char ** *argv*)

Here is the call graph for this function:



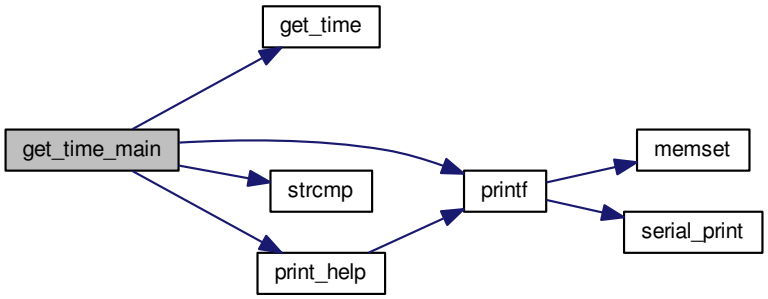
5.11.2.3 void get_time (date_time * *dateTimeValues*)

Here is the caller graph for this function:



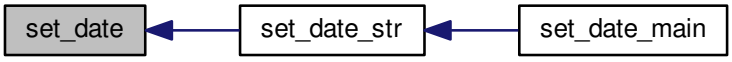
5.11.2.4 int get_time_main (int *argc*, char ** *argv*)

Here is the call graph for this function:



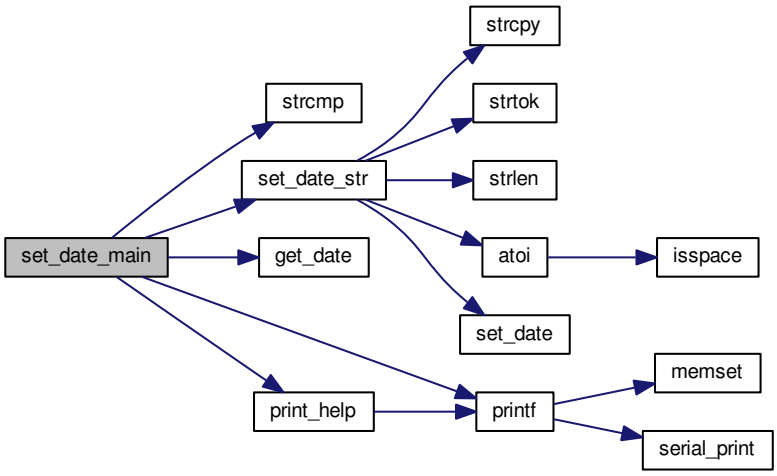
5.11.2.5 error_t set_date (const date_time * *dateTimeValues*)

Here is the caller graph for this function:



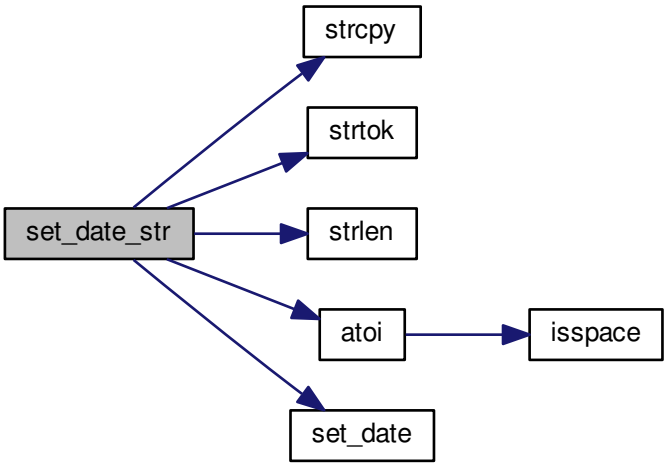
5.11.2.6 int set_date_main (int *argc*, char ** *argv*)

Here is the call graph for this function:



5.11.2.7 int set_date_str (const char * *str*)

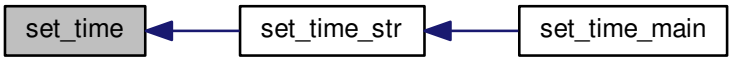
Here is the call graph for this function:



Here is the caller graph for this function:

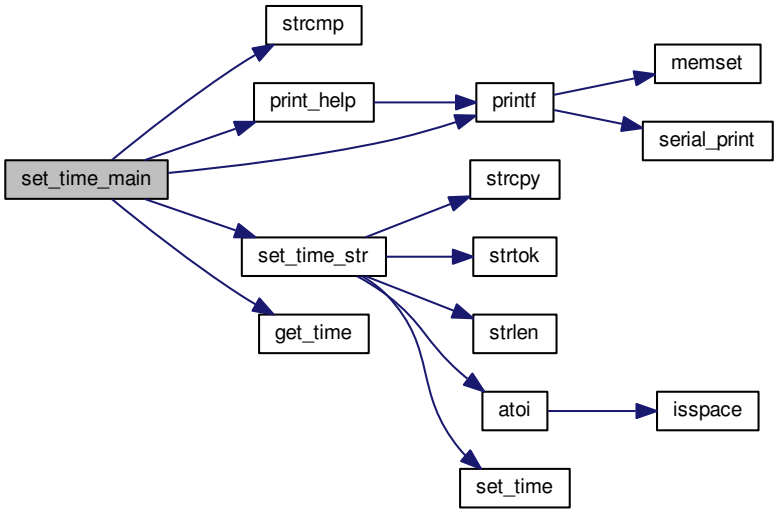
5.11.2.8 error_t set_time (const date_time * *dateTimeValues*)

Here is the caller graph for this function:



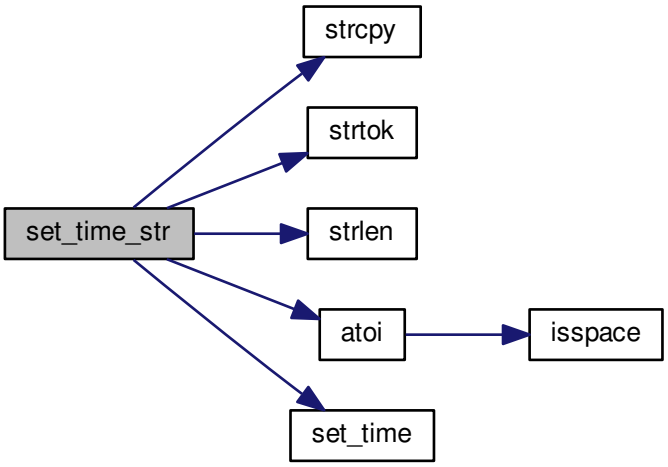
5.11.2.9 int set_time_main (int *argc*, char ** *argv*)

Here is the call graph for this function:



5.11.2.10 error_t set_time_str (const char * *timeStr*)

Here is the call graph for this function:



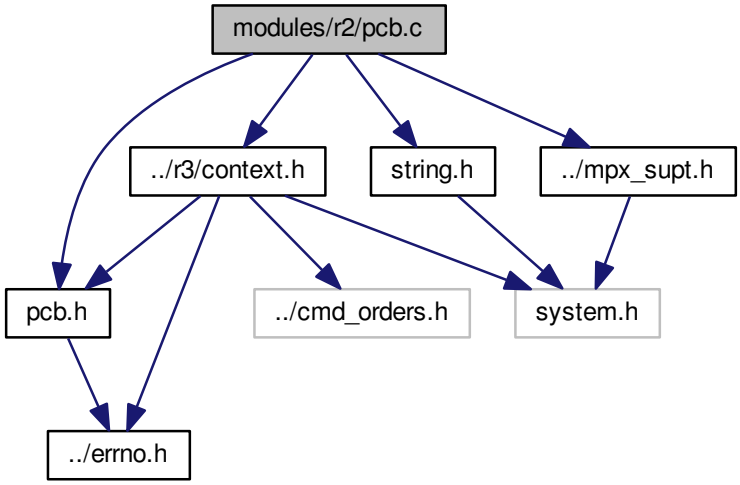
Here is the caller graph for this function:

5.12 modules/r2/pcb.c File Reference

The Process Control Block.

```
#include "pcb.h"
#include <string.h>
#include "../mpx_supt.h"
#include "../r3/context.h"
```

Include dependency graph for pcb.c:



Data Structures

- struct pcb_struct

  Struct that will describe PCB Processes.
- struct pcb_queue

  Queue structure that will store PCBs.

Enumerations

- enum process_state

  PCB process states/statuses.
- enum process_suspended

  PCB process suspended or not suspended status.

Functions

- enum process_state __attribute__ ((packed))

pcb_init

Initiates the PCB queues

- void pcb_init ()

suspend_pcb

Suspends the specific PCB.

Parameters

pcb_ptr	*The pointer to the PCB*

Returns

The error code. Possible error code to be returned: E_NOERROR No error. E_NULL_PTR Null pointer error.

- error_t suspend_pcb (struct pcb_struct *pcb_ptr)

resume_pcb

Resumes the specific PCB.

Parameters

pcb_ptr	*The pointer to the PCB*

Returns

The error code. Possible error code to be returned: E_NOERROR No error. E_NULL_PTR Null pointer error.

- error_t resume_pcb (struct pcb_struct *pcb_ptr)

allocate_pcb

allocate a space for the PCB structure.

Returns

The pointer that point to the PCB structure.

- struct pcb_struct * allocate_pcb ()

setup_pcb

allocate a space for the PCB structure, setup the properties of the PCB.

NOTE: pName must less than SIZE_OF_PCB_NAME character, pClass should be either "application" or "system" , and pPriority must within the range of [0, 9].

Parameters

pName	*Process Name (length < SIZE_OF_PCB_NAME).*
pClass	*Process class (system or application).*
pPriority	*Process priority (0 ∼ 9).*

Returns

NULL if error occured, otherwise, the pointer that point to the PCB structure.

- struct pcb_struct * setup_pcb (const char *pName, const enum process_class pClass, const unsigned char pPriority)

free_pcb

Frees all memory associated with given PCB, including the PCB itself, the stack, etc, with sys_free_mem()

Parameters

pcb_ptr	*The pointer to the PCB*

Returns

The error code. Possible error code to be returned: E_NOERROR No error. E_INVPARA The PCB probably had not been removed from queue before free it. E_FREEMEM The memory space cannot be actually free, since the student_free had not been implemented yet.

- error_t free_pcb (struct pcb_struct *pcb_ptr)

find_pcb

Will search all queues for a process named pName

Parameters

pName	*The char pointer to the desired searched name*

Returns

PCB pointer if found, NULL if PCB is not found

- struct pcb_struct * find_pcb (const char *pName)

insert_pcb

Inserts PCB into the appropriate queue.

Parameters

pcb_ptr	*The pointer to the PCB*

Returns

The error code. Possible error code to be returned: E_NOERROR No error. E_NULL_PTR Null pointer error. E_INVPARA The given PCB has running status or abnormal data members.

- error_t insert_pcb (struct pcb_struct *pcb_ptr)

remove_pcb

Removes PCB from the queue it is currently in.

Parameters

pcb_ptr	*The pointer to the PCB*

Returns

The error code. Possible error code to be returned: E_NOERROR No error. E_NULL_PTR Null pointer error. E_INVPARA The given PCB has abnormal data members.

- error_t remove_pcb (struct pcb_struct *pcb_ptr)

show_pcb

Displays the name, class, state, suspend status, and priority of a PCB.

Parameters

pName	*The PCB pointer.*

Returns

The error code. Possible error code to be returned: E_NOERROR No error. E_NULL_PTR Null pointer error.

- error_t show_pcb (struct pcb_struct *pcb_ptr)

show_blocked_processes

displays all blocked processes and their attributes

Returns

VOID.

- void show_blocked_processes ()

show_ready_processes

Displays all of the ready processes and their attributes.

Returns

VOID.

- void show_ready_processes ()

show_all_processes

Displays all of the processes and their attributes.

Returns

VOID.

- void show_all_processes ()

block_pcb

puts the given pcb into the blocked state and places it into the correct queue

Parameters

pcb_ptr	*The pointer to the PCB*

Returns

The error code. Possible error code to be returned: E_NOERROR No error. E_NULL_PTR Null pointer error. E_INVPARA The given PCB has abnormal data members (By "remove_pcb" or "insert_pcb").

- error_t block_pcb (struct pcb_struct *pcb_ptr)

unblock_pcb

puts the given pcb into the unblocked state and places it into the correct queue

Parameters

pcb_ptr	*The pointer to the PCB*

Returns

The error code. Possible error code to be returned: E_NOERROR No error. E_NULL_PTR Null pointer error. E_INVPARA The given PCB has abnormal data members (By "remove_pcb" or "insert_pcb").

- error_t unblock_pcb (struct pcb_struct *pcb_ptr)

set_pcb_priority

Sets the priority of the selected PCB

Parameters

pcb_ptr	*The PCB pointer.*
pPriorty	*The assigned priorirty*

Returns

The error code. Possible error code to be returned: E_NOERROR No error. E_NULL_PTR Null pointer error. E_INVPARA The pPriority is out of range. Or, the given PCB has abnormal data members (By "remove_pcb" or "insert_pcb").

- error_t set_pcb_priority (struct pcb_struct *pcb_ptr, const unsigned char pPriority)

get_running_process

gets a unsuspended and unblocked process from the front of the queue, and sets it to running state.

Parameters

None	

Returns

NULL if there is no process available, otherwise, the pointer that point to the PCB structure.

- struct pcb_struct * get_running_process ()

save_running_process

sets the running process to ready state, and inserts it to the ready queue.

Parameters

pcb_ptr	*The pointer to the PCB.*
new_stack_top	*The pointer to the new stack top.*

Returns

The error code. Possible error code to be returned: E_NOERROR No error. E_NULL_PTR Null pointer error. E_INVPARA The given PCB has abnormal data members (By "insert_pcb").

- error_t save_running_process (struct pcb_struct *pcb_ptr, struct context *new_stack_top)

get_stack_top

gets the pointer to the stack top of the specific PCB.

Parameters

pcb_ptr	*The pointer to the PCB.*

Returns

NULL if the pcb_ptr is NULL, otherwise, the pointer that point to the stack top of the specific PCB.

- unsigned char * get_stack_top (struct pcb_struct *pcb_ptr)

get_stack_base

gets the pointer to the stack base of the specific PCB.

Parameters

pcb_ptr	*The pointer to the PCB.*

Returns

NULL if the pcb_ptr is NULL, otherwise, the pointer that point to the stack base of the specific PCB.

- unsigned char * get_stack_base (struct pcb_struct *pcb_ptr)

shutdown_pcb

called when system is going to shutdown, removes all PCBs, free all PCBs.

Returns

VOID

- void shutdown_pcb ()

Variables

- running

  PCB in the running state.
- ready

  PCB in the ready state.
- blocked

  < PCB in the blocked state.
- true

PCB process is suspended.

- false

  < PCB process is not suspended.

- struct pcb_struct __attribute__

5.12.1 Detailed Description

The Process Control Block.

Author

Thunder Krakens

Date

March 18th, 2016

Version

R3

5.12.2 Enumeration Type Documentation

5.12.2.1 enum process_state

PCB process states/statuses.

5.12.2.2 enum process_suspended

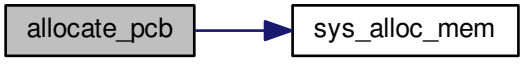
PCB process suspended or not suspended status.

5.12.3 Function Documentation

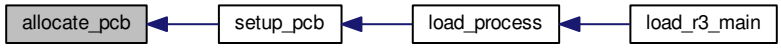
5.12.3.1 enum process_state __attribute__ ((packed))

5.12.3.2 struct pcb_struct* allocate_pcb ( )

Here is the call graph for this function:

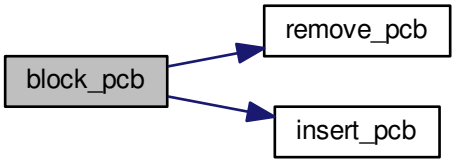


Here is the caller graph for this function:



5.12.3.3 error_t block_pcb (struct pcb_struct * *pcb_ptr*)

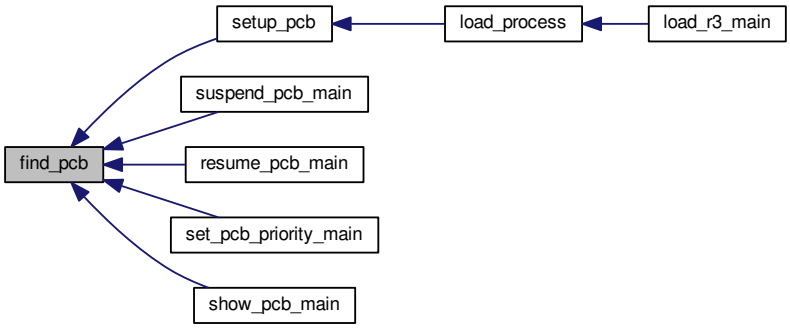
Here is the call graph for this function:



5.12.3.4 struct pcb_struct* find_pcb (const char * *pName*)

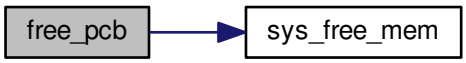
Here is the call graph for this function:

Here is the caller graph for this function:

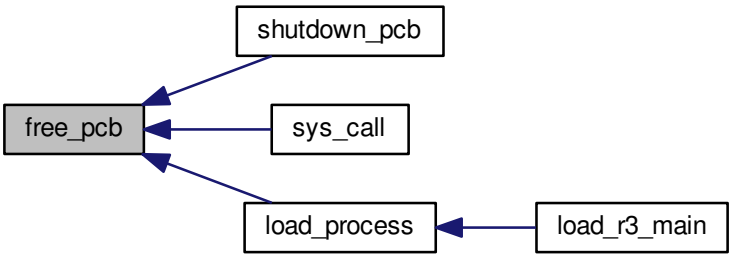


5.12.3.5 error_t free_pcb (struct pcb_struct * *pcb_ptr*)

Here is the call graph for this function:

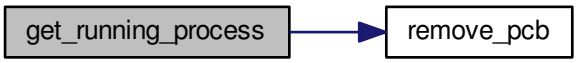


Here is the caller graph for this function:



5.12.3.6 struct pcb_struct* get_running_process ( )

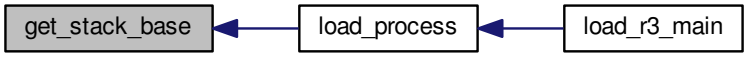
Here is the call graph for this function:



Here is the caller graph for this function:

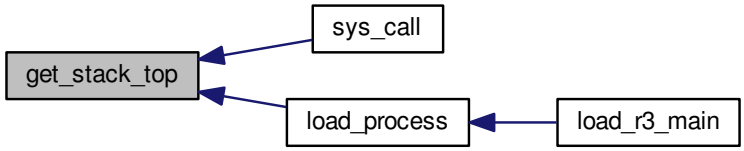
5.12.3.7 unsigned char* get_stack_base (struct pcb_struct * *pcb_ptr*)

Here is the caller graph for this function:



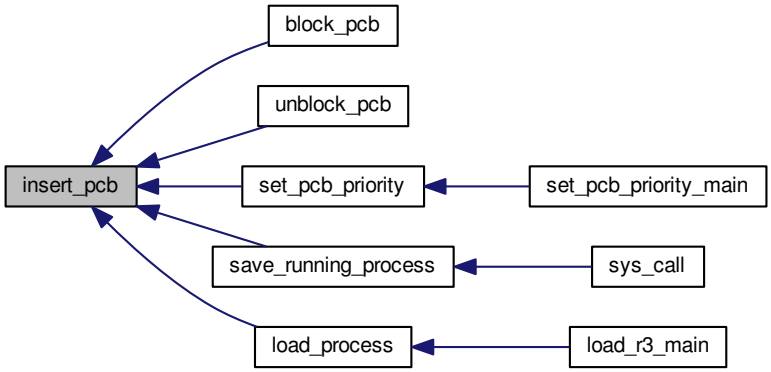
5.12.3.8 unsigned char* get_stack_top (struct pcb_struct * *pcb_ptr*)

Here is the caller graph for this function:



5.12.3.9 error_t insert_pcb (struct pcb_struct * *pcb_ptr*)

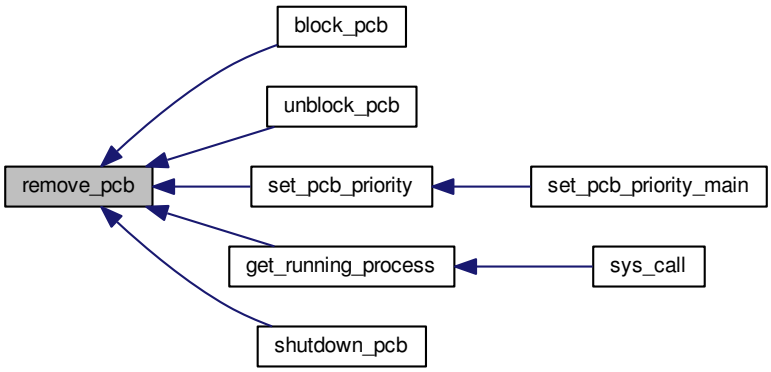
Here is the caller graph for this function:



5.12.3.10 void pcb_init ( )

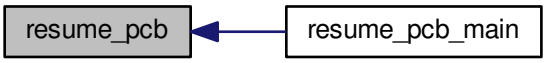
5.12.3.11 error_t remove_pcb (struct pcb_struct * *pcb_ptr*)

Here is the caller graph for this function:



5.12.3.12 error_t resume_pcb (struct pcb_struct * *pcb_ptr*)

Here is the caller graph for this function:



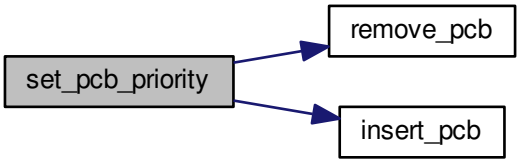
5.12.3.13 error_t save_running_process (struct pcb_struct * *pcb_ptr,* struct context * *new_stack_top*)

Here is the call graph for this function:

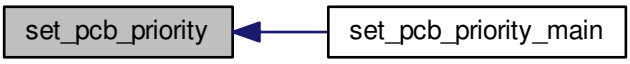
Here is the caller graph for this function:

5.12.3.14 error_t set_pcb_priority (struct pcb_struct * *pcb_ptr,* const unsigned char *pPriority*)

Here is the call graph for this function:

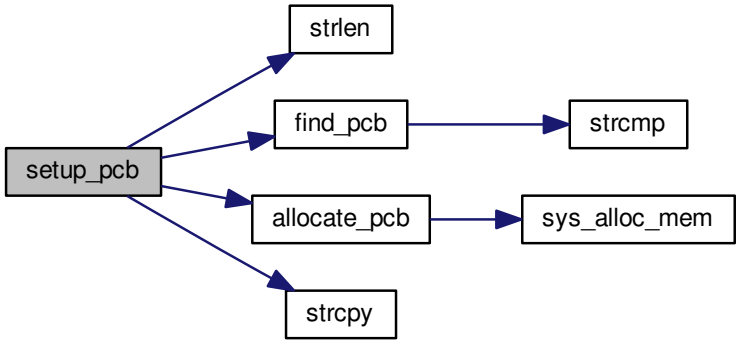


Here is the caller graph for this function:

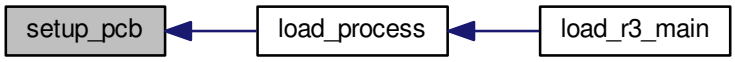


5.12.3.15 struct pcb_struct* setup_pcb (const char * *pName*, const enum process_class *pClass*, const unsigned char *pPriority*)

Here is the call graph for this function:

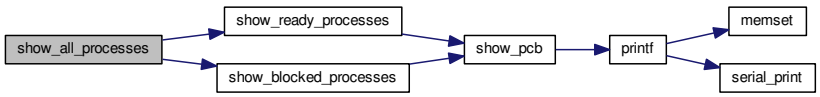


Here is the caller graph for this function:

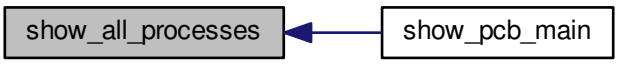


5.12.3.16 void show_all_processes ( )

Here is the call graph for this function:

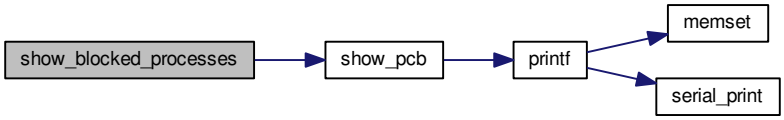


Here is the caller graph for this function:

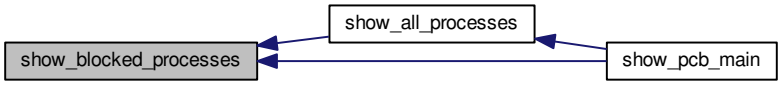


5.12.3.17 void show_blocked_processes ( )

Here is the call graph for this function:

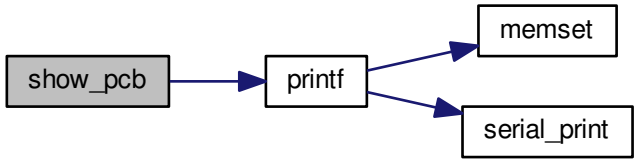


Here is the caller graph for this function:

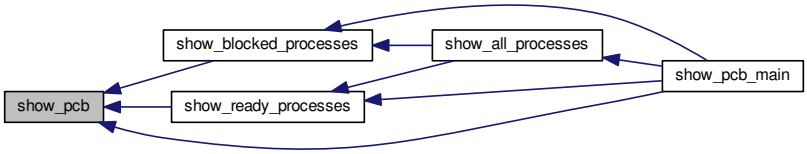


5.12.3.18 error_t show_pcb (struct pcb_struct * *pcb_ptr*)

Here is the call graph for this function:

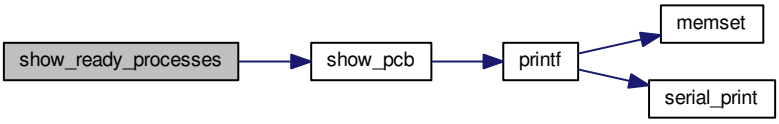


Here is the caller graph for this function:

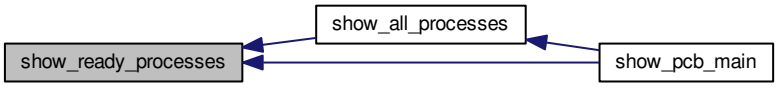


5.12.3.19 void show_ready_processes ( )

Here is the call graph for this function:

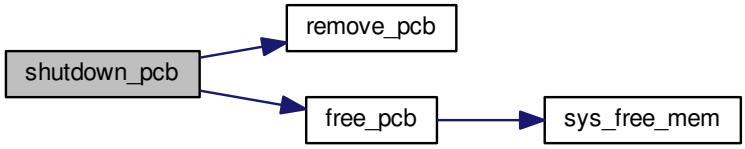


Here is the caller graph for this function:



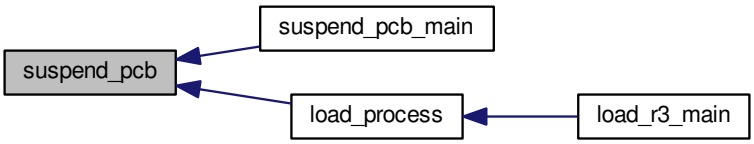
5.12.3.20 void shutdown_pcb ( )

Here is the call graph for this function:



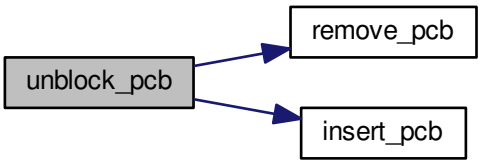
5.12.3.21 error_t suspend_pcb (struct pcb_struct * *pcb_ptr*)

Here is the caller graph for this function:



5.12.3.22 error_t unblock_pcb (struct pcb_struct * *pcb_ptr*)

Here is the call graph for this function:



5.12.4 Variable Documentation

5.12.4.1 struct pcb_struct __attribute__

5.12.4.2 blocked

< PCB in the blocked state.

PCB in the blocked state.

5.12.4.3 false

< PCB process is not suspended.

PCB process is not suspended.

5.12.4.4 ready

PCB in the ready state.

5.12.4.5 running

PCB in the running state.

5.12.4.6 true

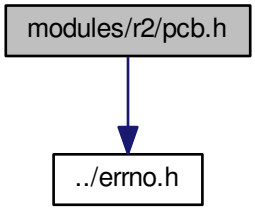
PCB process is suspended.

5.13 modules/r2/pcb.h File Reference

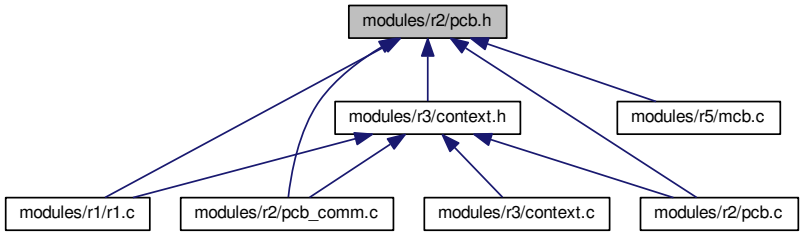
The Process Control Block.

```
#include "../errno.h"
```

Include dependency graph for pcb.h:



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



Macros

- #define SIZE_OF_STACK 1024
- #define SIZE_OF_PCB_NAME 10
- #define COMMHAND_PCB_NAME "commhand"
- #define IDLE_PCB_NAME "idle"

Enumerations

- enum process_class

  PCB process class types.

Functions

- enum process_class __attribute__ ((packed))

pcb_init

Initiates the PCB queues

- void pcb_init ()

allocate_pcb

allocate a space for the PCB structure.

Returns

The pointer that point to the PCB structure.

- struct pcb_struct * allocate_pcb ()

free_pcb

Frees all memory associated with given PCB, including the PCB itself, the stack, etc, with sys_free_mem()

Parameters

pcb_ptr	*The pointer to the PCB*

Returns

The error code. Possible error code to be returned: E_NOERROR No error. E_INVPARA The PCB probably had not been removed from queue before free it.

- error_t free_pcb (struct pcb_struct *pcb_ptr)

setup_pcb

allocate a space for the PCB structure, setup the properties of the PCB.

NOTE: pName must less than 10 character, pClass should be either "application" or "system" , and pPriority must within the range of [0, 9].

Parameters

pName	*Process Name (length < 10).*
pClass	*Process class (system or application).*
pPriority	*Process priority (0 ~ 9).*

Returns

NULL if error occured, otherwise, the pointer that point to the PCB structure.

- struct pcb_struct * setup_pcb (const char *pName, const enum process_class pClass, const unsigned char pPriority)

find_pcb

Will search all queues for a process named pName

Parameters

pName	*The char pointer to the desired searched name*

Returns

PCB pointer if found, NULL if PCB is not found

- struct pcb_struct * find_pcb (const char *pName)

insert_pcb

Inserts PCB into the appropriate queue.

Parameters

pcb_ptr	*The pointer to the PCB*

Returns

The error code. Possible error code to be returned: E_NOERROR No error. E_NULL_PTR Null pointer error. E_INVPARA The given PCB has running status or abnormal data members.

- error_t insert_pcb (struct pcb_struct *pcb_ptr)

remove_pcb

Removes PCB from the queue it is currently in.

Parameters

pcb_ptr	*The pointer to the PCB*

Returns

The error code. Possible error code to be returned: E_NOERROR No error. E_NULL_PTR Null pointer error. E_INVPARA The given PCB has abnormal data members.

- error_t remove_pcb (struct pcb_struct *pcb_ptr)

suspend_pcb

Suspends the specific PCB.

Parameters

pcb_ptr	*The pointer to the PCB*

Returns

The error code. Possible error code to be returned: E_NOERROR No error. E_NULL_PTR Null pointer error.

- error_t suspend_pcb (struct pcb_struct *pcb_ptr)

resume_pcb

Resumes the specific PCB.

Parameters

pcb_ptr	*The pointer to the PCB*

Returns

The error code. Possible error code to be returned: E_NOERROR No error. E_NULL_PTR Null pointer error.

- error_t resume_pcb (struct pcb_struct *pcb_ptr)

set_pcb_priority

Sets the priority of the selected PCB

Parameters

pcb_ptr	*The PCB pointer.*
pPriorty	*The assigned priorirty*

Returns

The error code. Possible error code to be returned: E_NOERROR No error. E_NULL_PTR Null pointer error. E_INVPARA The pPriority is out of range. Or, the given PCB has abnormal data members (By "remove_pcb" or "insert_pcb").

- error_t set_pcb_priority (struct pcb_struct *pcb_ptr, const unsigned char pPriority)

show_pcb

Displays the name, class, state, suspend status, and priority of a PCB.

Parameters

pName	*The PCB pointer.*

Returns

The error code. Possible error code to be returned: E_NOERROR No error. E_NULL_PTR Null pointer error.

- error_t show_pcb (struct pcb_struct *pcb_ptr)

show_all_processes

Displays all of the processes and their attributes.

Returns

VOID.

- void show_all_processes ()

show_ready_processes

Displays all of the ready processes and their attributes.

Returns

VOID.

- void show_ready_processes ()

show_blocked_processes

displays all blocked processes and their attributes

Returns

VOID.

- void show_blocked_processes ()

block_pcb

puts the given pcb into the blocked state and places it into the correct queue

Parameters

pcb_ptr	*The pointer to the PCB*

Returns

The error code. Possible error code to be returned: E_NOERROR No error. E_NULL_PTR Null pointer error. E_INVPARA The given PCB has abnormal data members (By "remove_pcb" or "insert_pcb").

- error_t block_pcb (struct pcb_struct *pcb_ptr)

unblock_pcb

puts the given pcb into the unblocked state and places it into the correct queue

Parameters

pcb_ptr	*The pointer to the PCB*

Returns

The error code. Possible error code to be returned: E_NOERROR No error. E_NULL_PTR Null pointer error. E_INVPARA The given PCB has abnormal data members (By "remove_pcb" or "insert_pcb").

- error_t unblock_pcb (struct pcb_struct *pcb_ptr)

get_running_process

gets a unsuspended and unblocked process from the front of the queue, and sets it to running state.

Parameters

None	

Returns

NULL if there is no process available, otherwise, the pointer that point to the PCB structure.

- struct pcb_struct * get_running_process ()

save_running_process

sets the running process to ready state, and inserts it to the ready queue.

Parameters

pcb_ptr	*The pointer to the PCB.*
new_stack_top	*The pointer to the new stack top.*

Returns

The error code. Possible error code to be returned: E_NOERROR No error. E_NULL_PTR Null pointer error. E_INVPARA The given PCB has abnormal data members (By "insert_pcb").

- error_t save_running_process (struct pcb_struct *pcb_ptr, struct context *new_stack_top)

get_stack_top

gets the pointer to the stack top of the specific PCB.

Parameters

pcb_ptr	*The pointer to the PCB.*

Returns

NULL if the pcb_ptr is NULL, otherwise, the pointer that point to the stack top of the specific PCB.

- unsigned char * get_stack_top (struct pcb_struct *pcb_ptr)

get_stack_base

gets the pointer to the stack base of the specific PCB.

Parameters

pcb_ptr	*The pointer to the PCB.*

Returns

NULL if the pcb_ptr is NULL, otherwise, the pointer that point to the stack base of the specific PCB.

- unsigned char * get_stack_base (struct pcb_struct *pcb_ptr)

shutdown_pcb

called when system is going to shutdown, removes all PCBs, free all PCBs.

Returns

VOID

- void shutdown_pcb ()

Variables

- pcb_class_app

  Process is an application process.
- pcb_class_sys

  < Process is a system process.

5.13.1 Detailed Description

The Process Control Block.

Author

Thunder Krakens

Date

February 7th, 2016

Version

R3

5.13.2 Macro Definition Documentation

5.13.2.1 #define COMMHAND_PCB_NAME "commhand"

5.13.2.2 #define IDLE_PCB_NAME "idle"

5.13.2.3 #define SIZE_OF_PCB_NAME 10

5.13.2.4 #define SIZE_OF_STACK 1024

5.13.3 Enumeration Type Documentation

5.13.3.1 enum process_class

PCB process class types.

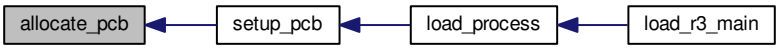
5.13.4 Function Documentation

5.13.4.1 enum process_class __attribute__ ((packed))

5.13.4.2 struct pcb_struct* allocate_pcb ( )

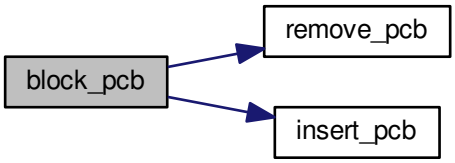
Here is the call graph for this function:

Here is the caller graph for this function:



5.13.4.3 error_t block_pcb (struct pcb_struct * *pcb_ptr*)

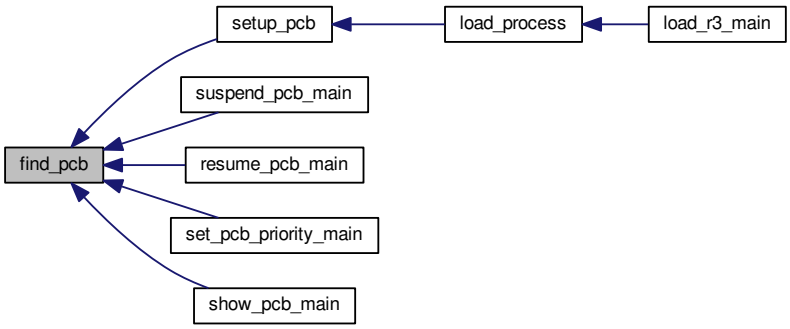
Here is the call graph for this function:



5.13.4.4 struct pcb_struct* find_pcb (const char * *pName*)

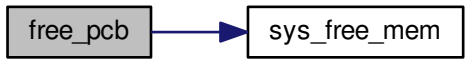
Here is the call graph for this function:

Here is the caller graph for this function:

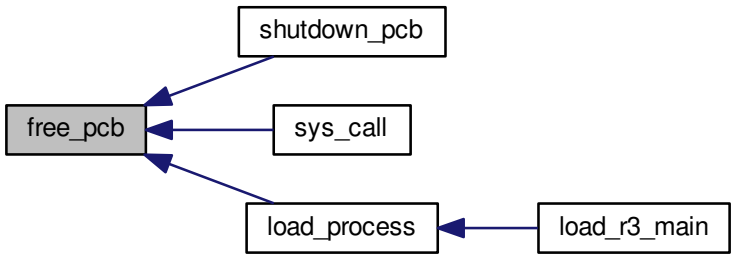


5.13.4.5 error_t free_pcb (struct pcb_struct * *pcb_ptr*)

Here is the call graph for this function:

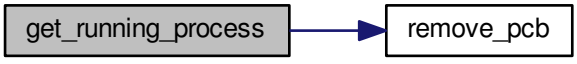


Here is the caller graph for this function:



5.13.4.6 struct pcb_struct* get_running_process ( )

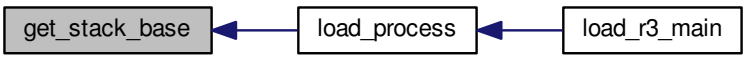
Here is the call graph for this function:



Here is the caller graph for this function:

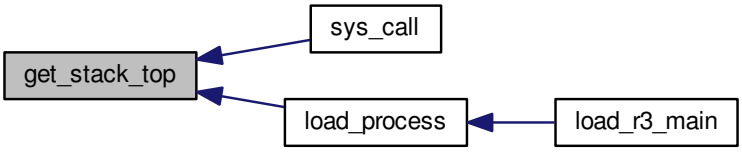
5.13.4.7 unsigned char* get_stack_base (struct pcb_struct * *pcb_ptr*)

Here is the caller graph for this function:



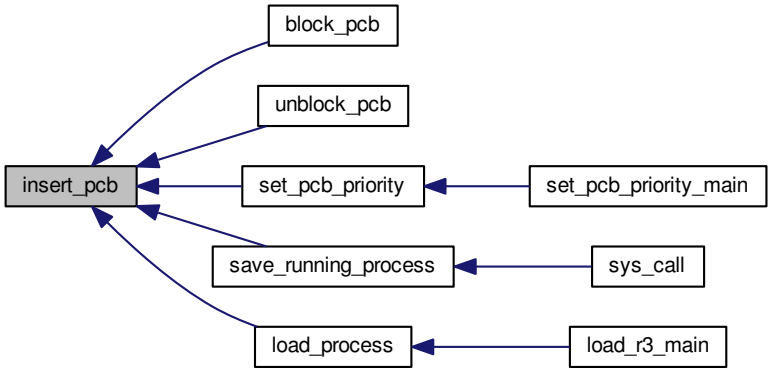
5.13.4.8 unsigned char* get_stack_top (struct pcb_struct * *pcb_ptr*)

Here is the caller graph for this function:



5.13.4.9 error_t insert_pcb (struct pcb_struct * *pcb_ptr*)

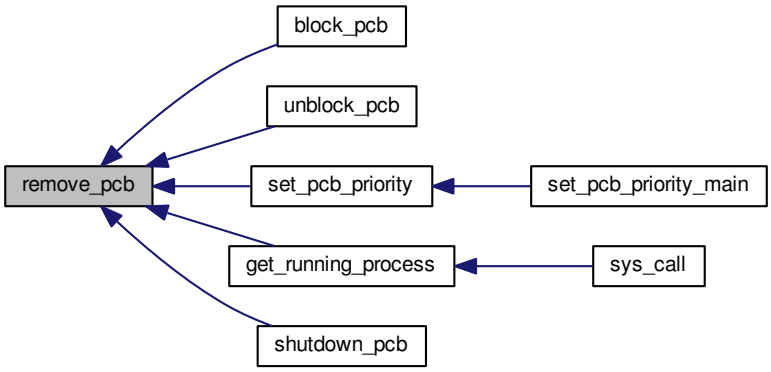
Here is the caller graph for this function:



5.13.4.10 void pcb_init ( )

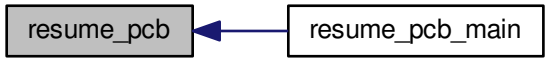
5.13.4.11 error_t remove_pcb (struct pcb_struct * *pcb_ptr*)

Here is the caller graph for this function:



5.13.4.12 error_t resume_pcb (struct pcb_struct * *pcb_ptr*)

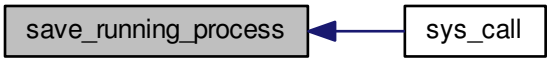
Here is the caller graph for this function:



5.13.4.13 error_t save_running_process (struct pcb_struct * *pcb_ptr,* struct context * *new_stack_top*)

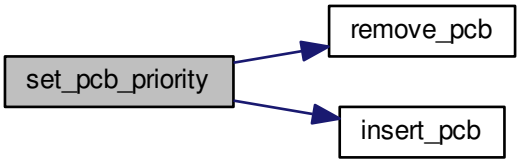
Here is the call graph for this function:

Here is the caller graph for this function:

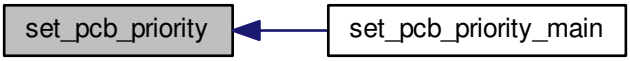


5.13.4.14 error_t set_pcb_priority (struct pcb_struct * *pcb_ptr,* const unsigned char *pPriority*)

Here is the call graph for this function:

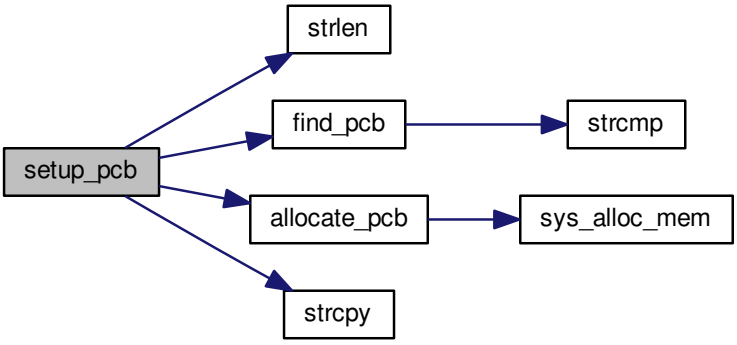


Here is the caller graph for this function:

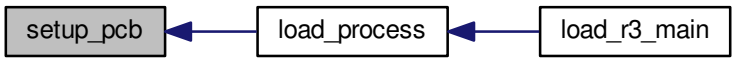


5.13.4.15 struct pcb_struct* setup_pcb (const char * *pName,* const enum process_class *pClass,* const unsigned char *pPriority*)

Here is the call graph for this function:

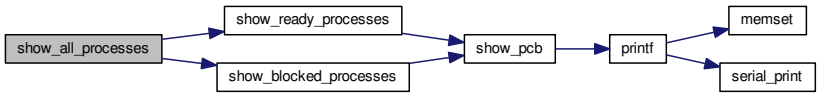


Here is the caller graph for this function:

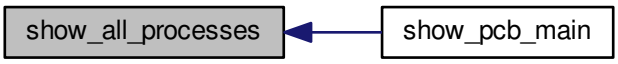


5.13.4.16 void show_all_processes ( )

Here is the call graph for this function:

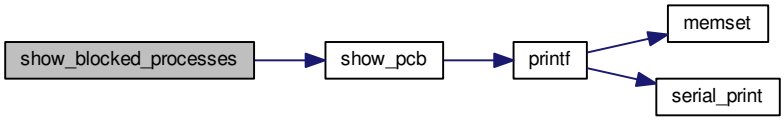


Here is the caller graph for this function:

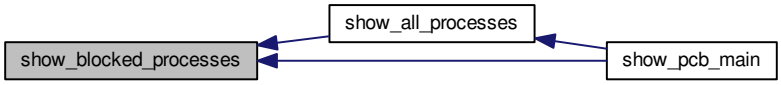


5.13.4.17 void show_blocked_processes ( )

Here is the call graph for this function:

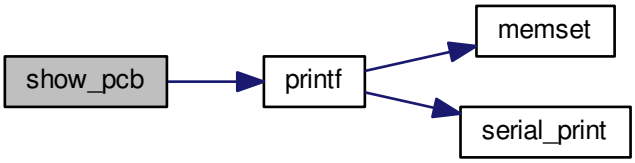


Here is the caller graph for this function:

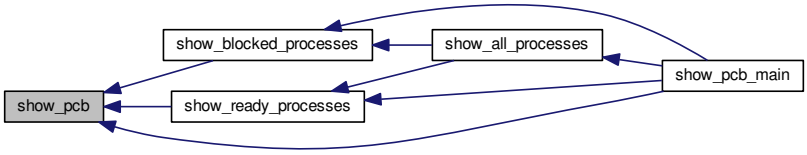


5.13.4.18 error_t show_pcb (struct pcb_struct * *pcb_ptr*)

Here is the call graph for this function:

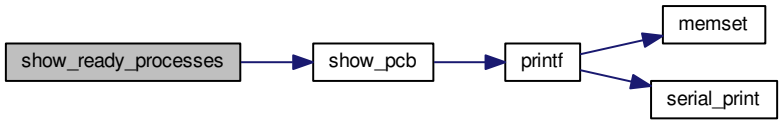


Here is the caller graph for this function:

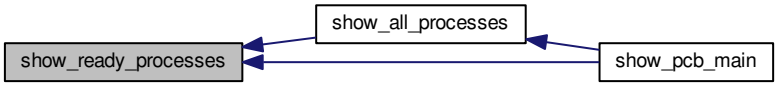


5.13.4.19 void show_ready_processes ( )

Here is the call graph for this function:

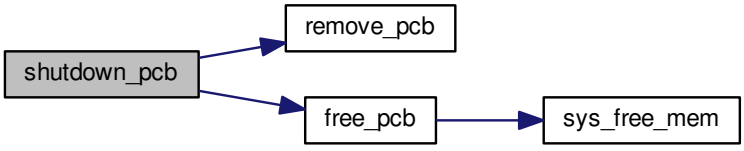


Here is the caller graph for this function:



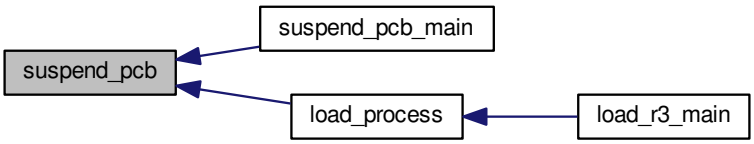
5.13.4.20 void shutdown_pcb ( )

Here is the call graph for this function:



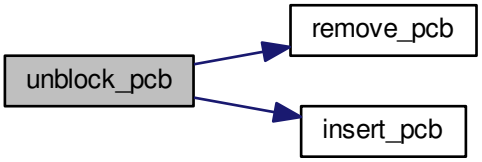
5.13.4.21 error_t suspend_pcb (struct pcb_struct * *pcb_ptr*)

Here is the caller graph for this function:



5.13.4.22 error_t unblock_pcb (struct pcb_struct * *pcb_ptr*)

Here is the call graph for this function:



5.13.5 Variable Documentation

5.13.5.1 pcb_class_app

Process is an application process.

5.13.5.2 pcb_class_sys

< Process is a system process.

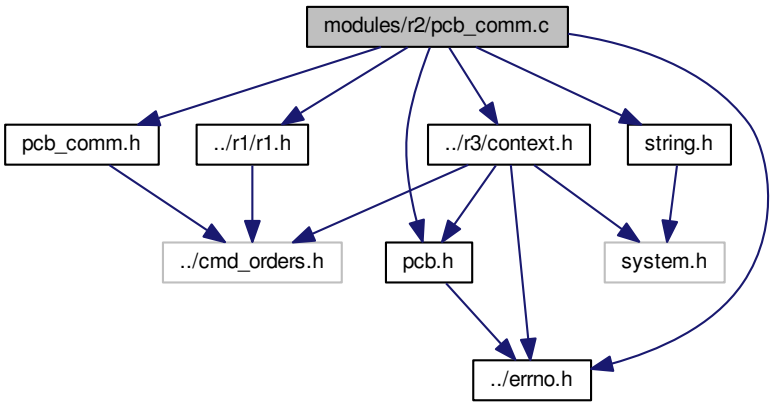
Process is a system process.

5.14 modules/r2/pcb_comm.c File Reference

The main functions that manipulate the PCB.

```
#include "pcb_comm.h"
#include "pcb.h"
#include <string.h>
#include "../errno.h"
#include "../r1/r1.h"
#include "../r3/context.h"
```

Include dependency graph for pcb_comm.c:



Functions

suspend_pcb_main.

The main function for the "suspend PCB".

Accepted formats: pcb suspend <name> pcb suspend –help

Parameters

argc	*The number of tokens found.*
argv	*The array of tokens.*

Returns

0

- int suspend_pcb_main (int argc, char **argv)

resume_pcb_main.

The main function for the "resume PCB".

Accepted formats: pcb resume <name> pcb resume –help

Parameters

argc	*The number of tokens found.*
argv	*The array of tokens.*

Returns

0

- int resume_pcb_main (int argc, char **argv)

set_pcb_priority_main.

The main function for the "set PCB priority".

Accepted formats: pcb setpriority <name> <priority> pcb setpriority –help

Parameters

argc	*The number of tokens found.*
argv	*The array of tokens.*

Returns

0

- int set_pcb_priority_main (int argc, char **argv)

show_pcb_main.

The main function for the "Show PCB", "Show all Processes", "Show Ready Processes", and "Show Blocked Processes".

Accepted formats: pcb show -name [name] pcb show -all pcb show -ready pcb show -blocked pcb show –help

Parameters

argc	*The number of tokens found.*
argv	*The array of tokens.*

Returns

0

- int show_pcb_main (int argc, char **argv)

5.14.1 Detailed Description

The main functions that manipulate the PCB.

Author

Thunder Krakens

Date

February 7th, 2016

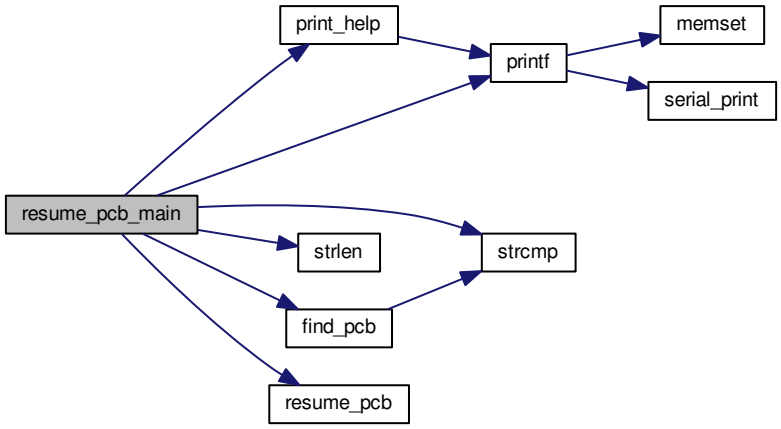
Version

R2

5.14.2 Function Documentation

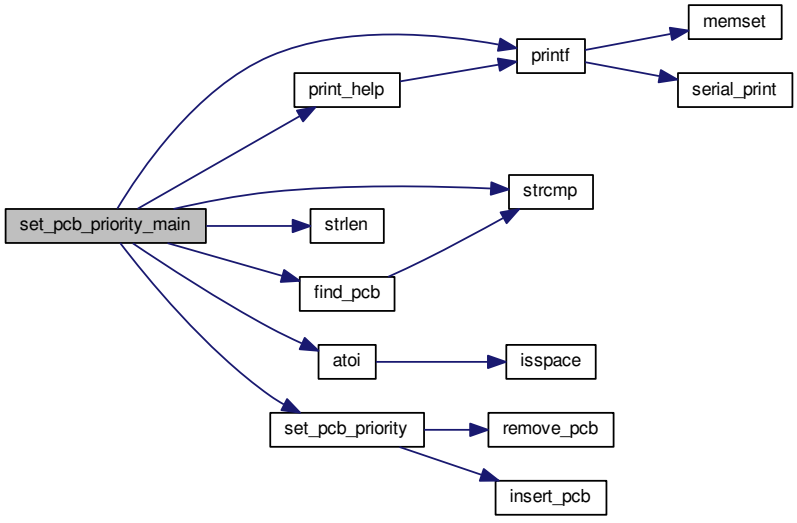
5.14.2.1 int resume_pcb_main (int *argc,* char ** *argv*)

Here is the call graph for this function:



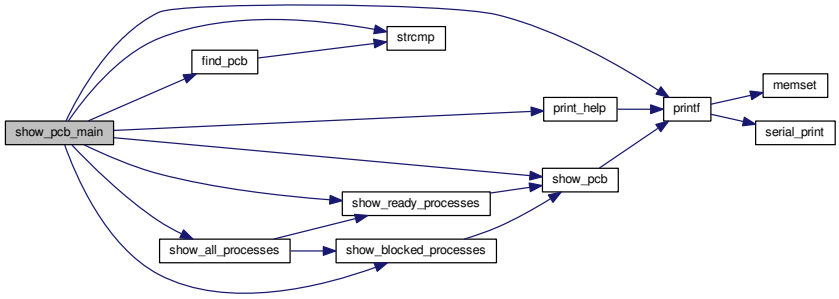
5.14.2.2 int set_pcb_priority_main (int *argc,* char ** *argv*)

Here is the call graph for this function:



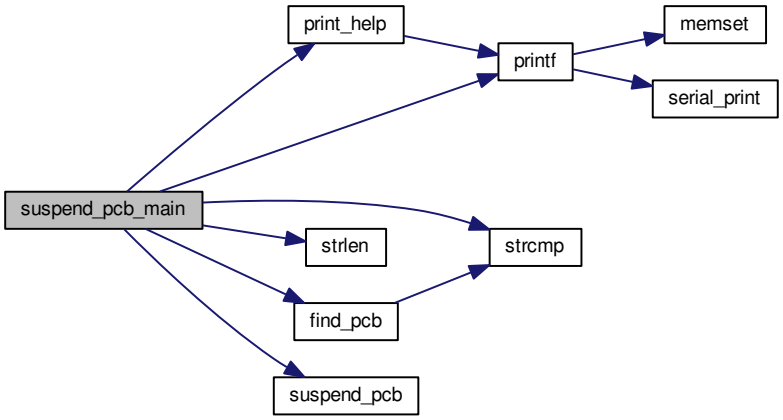
5.14.2.3 int show_pcb_main (int *argc*, char ** *argv*)

Here is the call graph for this function:



5.14.2.4 int suspend_pcb_main (int *argc*, char ** *argv*)

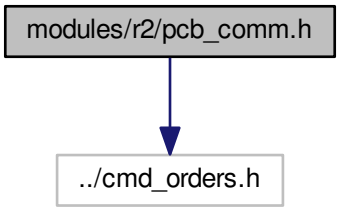
Here is the call graph for this function:



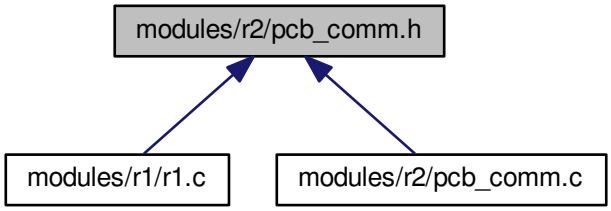
5.15 modules/r2/pcb_comm.h File Reference

The main functions that manipulate the PCB.

```
#include "../cmd_orders.h"
```
Include dependency graph for pcb_comm.h:



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



Functions

suspend_pcb_main.

The main function for the "suspend PCB".

Accepted formats: pcb suspend <name> pcb suspend –help

Parameters

argc	*The number of tokens found.*
argv	*The array of tokens.*

Returns

0

- int suspend_pcb_main (int argc, char **argv)

resume_pcb_main.

The main function for the "resume PCB".

Accepted formats: pcb resume <name> pcb resume –help

Parameters

argc	*The number of tokens found.*
argv	*The array of tokens.*

Returns

0

- int resume_pcb_main (int argc, char **argv)

set_pcb_priority_main.

The main function for the "set PCB priority".

Accepted formats: pcb setpriority <name> <priority> pcb setpriority –help

Parameters

argc	*The number of tokens found.*
argv	*The array of tokens.*

Returns

0

- int set_pcb_priority_main (int argc, char **argv)

show_pcb_main.

The main function for the "Show PCB", "Show all Processes", "Show Ready Processes", and "Show Blocked Processes".

Accepted formats: pcb show [name] pcb show -all pcb show -ready pcb show -blocked pcb show –help

Parameters

argc	*The number of tokens found.*
argv	*The array of tokens.*

Returns

0

- int show_pcb_main (int argc, char **argv)

5.15.1 Detailed Description

The main functions that manipulate the PCB.

Author

Thunder Krakens

Date

February 7th, 2016

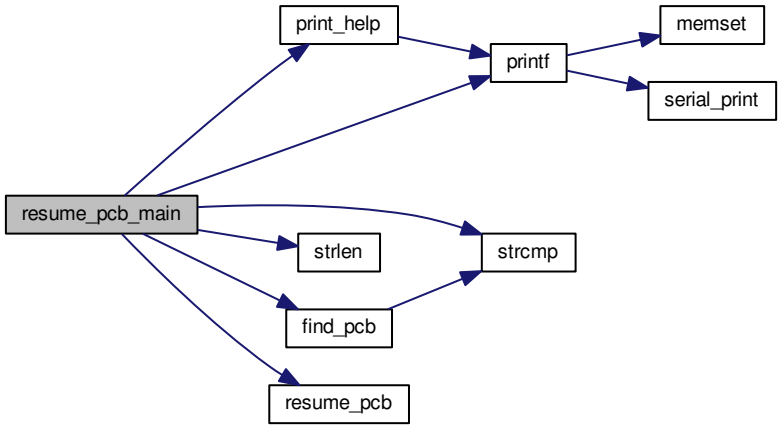
Version

R2

5.15.2 Function Documentation

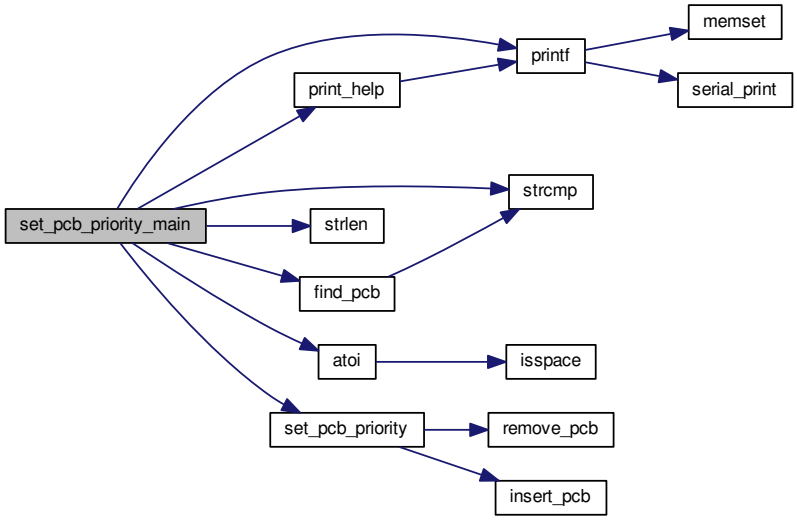
5.15.2.1 int resume_pcb_main (int *argc,* char ** *argv*)

Here is the call graph for this function:



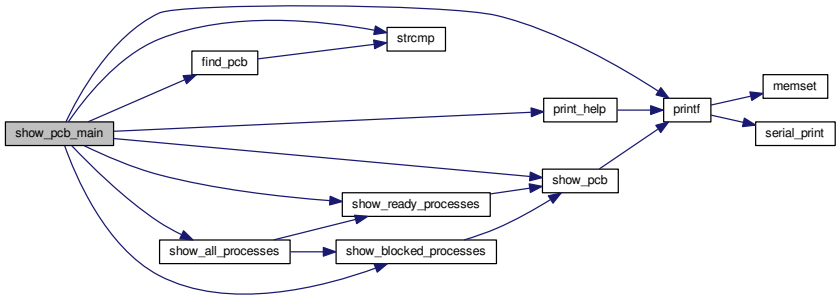
5.15.2.2 int set_pcb_priority_main (int *argc,* char ** *argv*)

Here is the call graph for this function:



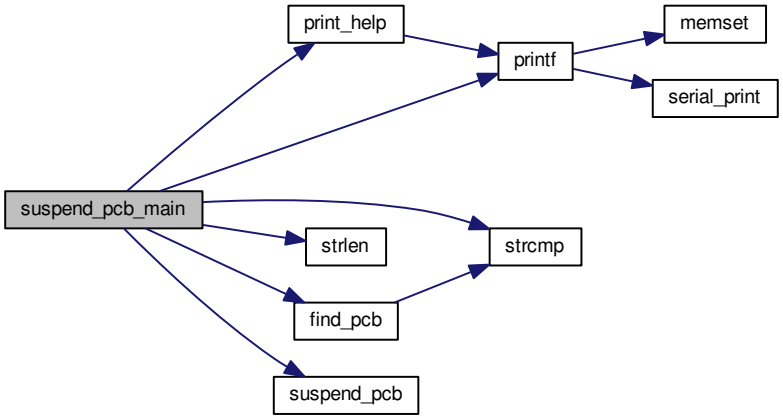
5.15.2.3 int show_pcb_main (int *argc*, char ** *argv*)

Here is the call graph for this function:



5.15.2.4 int suspend_pcb_main (int *argc*, char ** *argv*)

Here is the call graph for this function:

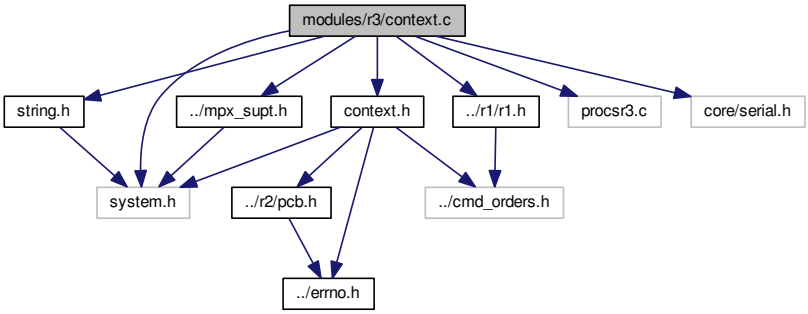


5.16 modules/r3/context.c File Reference

Context Switching.

```
#include <string.h>
#include "context.h"
#include "../mpx_supt.h"
#include "../r1/r1.h"
#include "procsr3.c"
```

Include dependency graph for context.c:



Functions

is_digit

Checks if the character is a digit.

Parameters

ch	*character selected.*

Returns

a digit between 0 and 9.

sys_call

system call interrupt

Parameters

context*	*registers current registers*

Returns

result if there is no current process running, it will load new context. If the process is still running, it will load its old context.

- u32int * sys_call (struct context *registers)

load_process

loads a process into the PCB.

Parameters

pName	*Process Name*
pClass	*Process Class*

pPriority	*Process Priority*
*function()	*A function pointer*

Returns

new_pcb Returns the values of the new PCB

- struct pcb_struct * load_process (const char *pName, const enum process_class pClass, const unsigned char pPriority, void(*function)())

load_r3_main

Loads the main function of R3.

Parameters

argc	*The number of tokens found.*
argv	*The array of tokens.*

Returns

0

- int load_r3_main (int argc, char **argv)

Variables

- struct pcb_struct * cop = NULL
- struct context * old_context = NULL

5.16.1 Detailed Description

Context Switching.

Author

Thunder Krakens

Date

March 18th, 2016

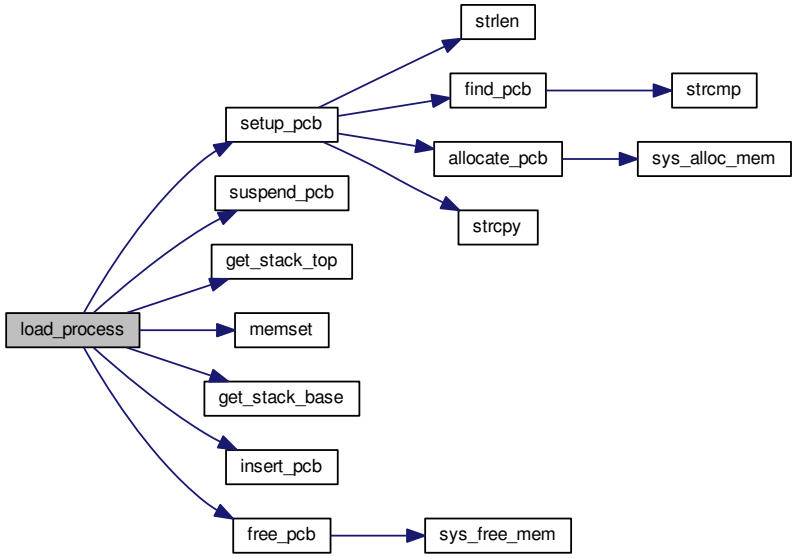
Version

R3

5.16.2 Function Documentation

5.16.2.1 struct pcb_struct* load_process (const char * *pName,* const enum process_class *pClass,* const unsigned char *pPriority,* void(*)() *function*)

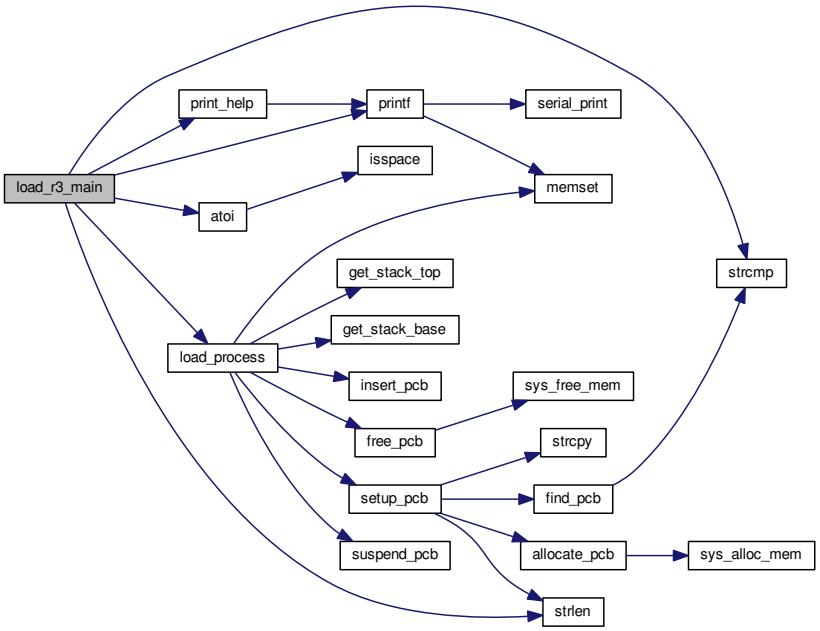
Here is the call graph for this function:



Here is the caller graph for this function:

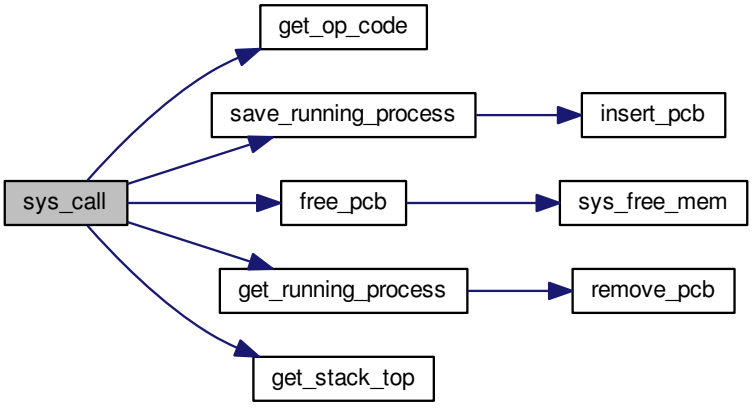
5.16.2.2 int load_r3_main (int *argc,* char ** *argv*)

Here is the call graph for this function:



5.16.2.3 u32int* sys_call (struct context * *registers*)

Here is the call graph for this function:



5.16.3 Variable Documentation

5.16.3.1 struct pcb_struct* cop = NULL

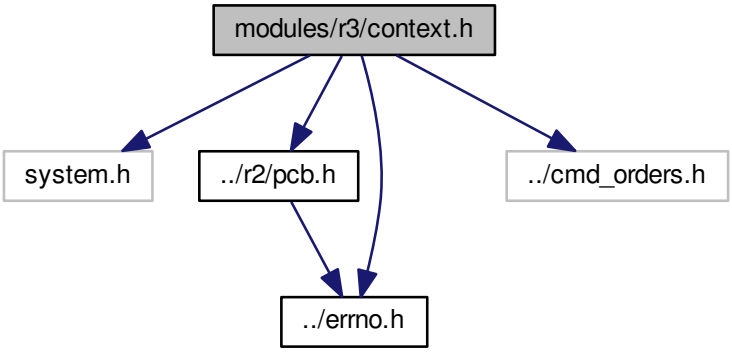
5.16.3.2 struct context* old_context = NULL

5.17 modules/r3/context.h File Reference

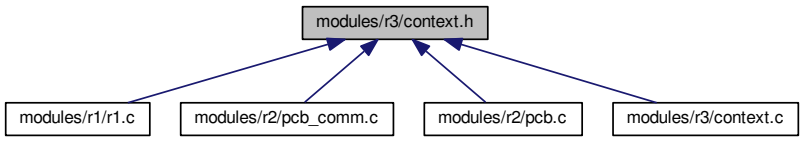
Context Switching.

```
#include <system.h>
#include "../r2/pcb.h"
#include "../errno.h"
#include "../cmd_orders.h"
```

Include dependency graph for context.h:



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



Data Structures

- struct context

  Context structure that holds the 15 CPU register values to begin and resume process execution.

Functions

sys_call

system call interrupt

Parameters

context∗	*registers current registers*

Returns

result if there is no current process running, it will load new context. If the process is still running, it will load its old context.

- u32int ∗ sys_call (struct context ∗registers)

load_process

loads a process into the PCB.

Parameters

pName	*Process Name*
pClass	*Process Class*
pPriority	*Process Priority*
∗function()	*A function pointer*

Returns

new_pcb Returns the values of the new PCB

- struct pcb_struct ∗ load_process (const char ∗pName, const enum process_class pClass, const unsigned char pPriority, void(∗function)())

load_r3_main

Loads the main function of R3.

Parameters

argc	*The number of tokens found.*
argv	*The array of tokens.*

Returns

0

- int load_r3_main (int argc, char ∗∗argv)

Variables

- struct context ∗ old_context
- struct pcb_struct ∗ cop

5.17.1 Detailed Description

Context Switching.

Author

Thunder Krakens

Date

March 18th, 2016

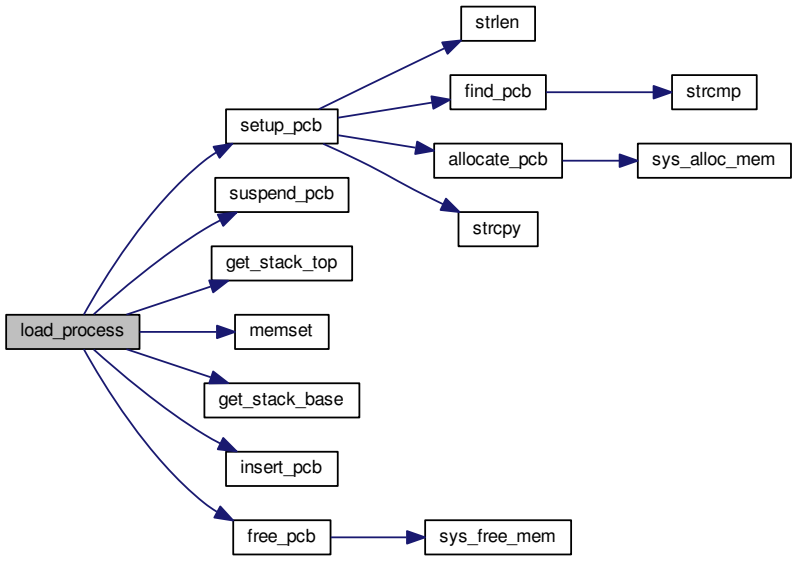
Version

R3

5.17.2 Function Documentation

5.17.2.1 struct pcb_struct* load_process (const char * *pName,* const enum process_class *pClass,* const unsigned char *pPriority,* void(*)() *function*)

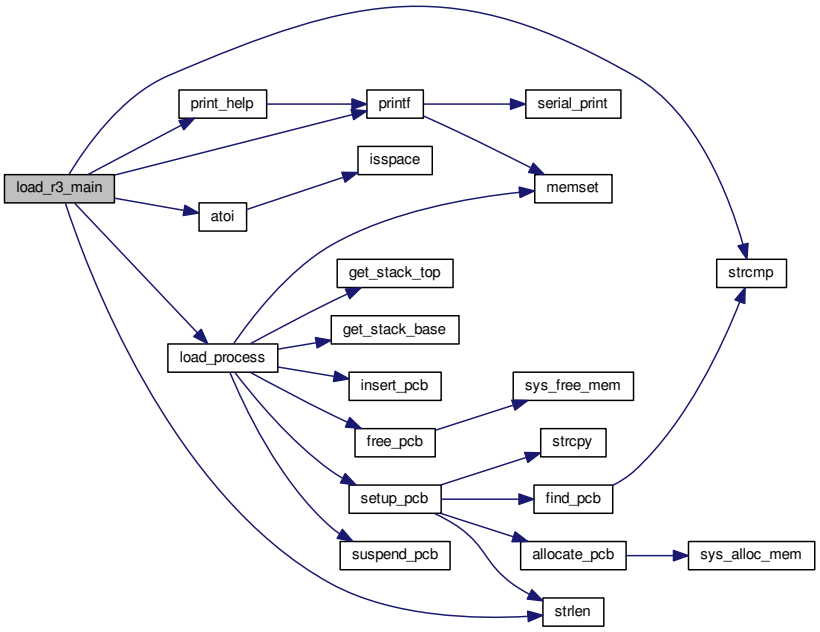
Here is the call graph for this function:



Here is the caller graph for this function:

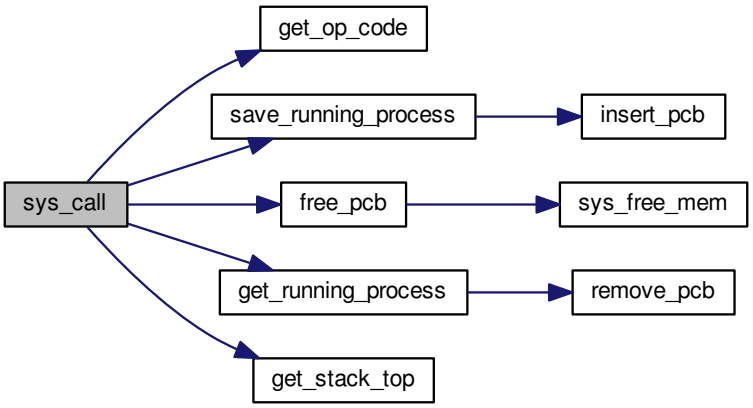
5.17.2.2 int load_r3_main (int *argc,* char ** *argv*)

Here is the call graph for this function:



5.17.2.3 u32int* sys_call (struct context * *registers*)

Here is the call graph for this function:



5.17.3 Variable Documentation

5.17.3.1 struct pcb_struct* cop

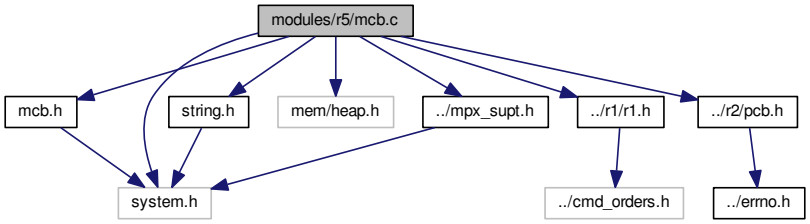
5.17.3.2 struct context* old_context

5.18 modules/r5/mcb.c File Reference

Memory Control Block.

```
#include "mcb.h"
#include <system.h>
#include <string.h>
#include <mem/heap.h>
#include "../mpx_supt.h"
#include "../r1/r1.h"
#include "../r2/pcb.h"
```

Include dependency graph for mcb.c:



Data Structures

- struct cmcb

  Complete Memory Control Block Struct.
- struct lmcb

  Limited Memory Control Block Struct.
- struct mcb

  Memory Control Block Struct.

Enumerations

- enum mcb_type

  PCB process class types.

Functions

- enum mcb_type __attribute__ ((packed))

get_mcb_total_size

Finds MCB's total size

Parameters

mem_size	*The size of memory*

Returns

the total size of memory

init_mem_block

Initiate memory block

Parameters

start_pos	*Starting position*
size	*Size of MCB*
type	*Type of MCB*

Returns

VOID

prev_adjacent_mcb

Previous adjacent MCB in Memory

Parameters

mcb_ptr	*MCB Pointer*

Returns

the pointer to the previous MCB

next_adjacent_mcb

Next adjacent MCB in Memory

Parameters

mcb_ptr	*MCB Pointer*

Returns

the pointer to the next MCB

find_mcb_by_address

Finds the MCB by address

Parameters

mem_ptr	*Memory address pointer*

Returns

Pointer to the MCB

insert_mcb_to_queue

Inserts MCB to the queue

Parameters

mcb_ptr	*MCB Pointer*
type	*The type of MCB*

remove_mcb_to_queue

Removes MCB from the queue

Parameters

mcb_ptr	*MCB Pointer*
type	*The type of MCB*

init_heap

Allocates all the memory for MPX

Parameters

size	*Size of heap in bytes*

- void init_heap (u32int size)

show_mcb

Displays the allocated or free memory block's address, previous and next pointers, and block's size.

Parameters

mcb_ptr	*MCB Pointer*

- void show_mcb (struct mcb ∗mcb_ptr)

find_first_fit_mcb

Finds the first block in the free_mem_list big enough to hold mem_size

Parameters

mem_size	*The size to be allocated from the heap*

Returns

The pointer to the MCB

mcb_allocate

Allocates a memory block

Parameters

mem_size	*The MCB size to be allocated*

Returns

Address to allocated MCB
NULL if not enough space in free memory found

- void ∗ mcb_allocate (u32int mem_size)

mcb_free

Frees a block of memory that was previously allocated

Parameters

mcb_ptr	*MCB Node pointer*

Returns

E_NOERROR No Error found
E_INVPARA Invalid Parameter

show_allocated_mcb

Displays all the allocated MCBs

- void show_allocated_mcb ()

show_free_mcb

Displays all the free memory

- void show_free_mcb ()

show_all_mcb

Displays all the free and allocated memory

- void show_all_mcb ()

mcb_allocate_mpx

Calls mcb_allocate to allocate memory block, used as parameter for sys_set_malloc in kmain.c

Parameters

size	*Size of block in bytes to allocate*

Returns

Address of allocated MCB

- u32int mcb_allocate_mpx (u32int size)

mcb_free_mpx

Calls mcb_free to free memory block, used as parameter for sys_set_free in kmain.c

Parameters

mem_ptr	*Memory Pointer*

Returns

0

- int mcb_free_mpx (void ∗mem_ptr)

is_mcb_empty

Checks if the heap is empty

Returns

0 or 1 (true or false)

- int is_mcb_empty ()

shutdown_mcb.

Shutdown the pcb during the shutdown procedure.

Returns

0

- void shutdown_mcb ()

show_mcb_main.

The function of show MCB for commhand.

Parameters

argc	*The number of tokens found.*
argv	*The array of tokens.*

Returns

0

- int show_mcb_main (int argc, char **argv)

is_digit

Checks if input is a digit used for testing for module R5

Parameters

ch	*single character*

Returns

0 or 1 (true or false)

is_all_digit

Checks if string is a digit used for testing for module R5

Parameters

str_ptr	*string array 0 or 1 (true or false)*

init_heap_main

The main function of initate heap only used for testing in commhand for module R5

Parameters

argc	*The number of tokens found.*
argv	*The array of tokens.*

Returns

0

- int init_heap_main (int argc, char **argv)

mcb_allocate_main.

The main function of MCB allocate only used for testing in commhand for module R5

Parameters

argc	*The number of tokens found.*
argv	*The array of tokens.*

Returns

0

- int mcb_allocate_main (int argc, char **argv)

mcb_free_main.

The main function of free MCB only used for testing in commhand for module R5

Parameters

argc	*The number of tokens found.*
argv	*The array of tokens.*

Returns

0

- int mcb_free_main (int argc, char **argv)

is_mcb_empty_main.

The main function of is MCB empty only used for testing in commhand for module R5

Parameters

argc	*The number of tokens found.*
argv	*The array of tokens.*

Returns

0

- int is_mcb_empty_main (int argc, char **argv)

Variables

- u32int start_of_memory

  Global variable labeling start of memory.
- u32int end_of_memory
- struct mcb * free_mem_list
- struct mcb * allocated_mem_list
- free

  Process is an application process.
- allocated

  < Process is a system process.
- struct cmcb __attribute__

5.18.1 Detailed Description

Memory Control Block.

Author

Thunder Krakens

Date

April 8th, 2016

Version

R5

5.18.2 Enumeration Type Documentation

5.18.2.1 enum mcb_type

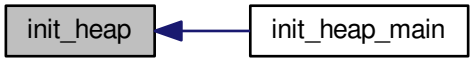
PCB process class types.

5.18.3 Function Documentation

5.18.3.1 enum mcb_type __attribute__ ((packed))

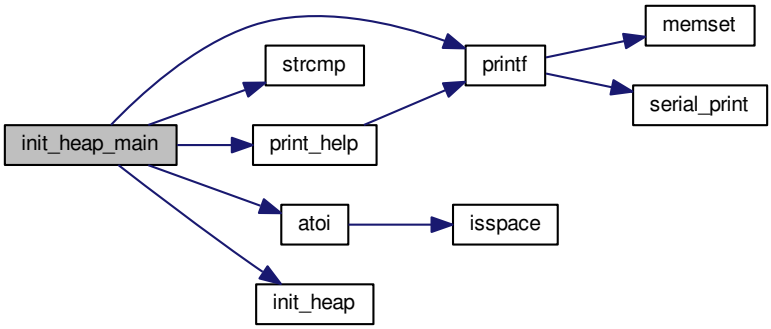
5.18.3.2 void init_heap (u32int *size*)

Here is the caller graph for this function:



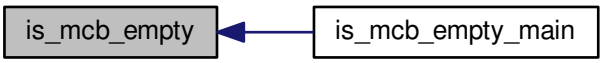
5.18.3.3 int init_heap_main (int *argc,* char ** *argv*)

Here is the call graph for this function:



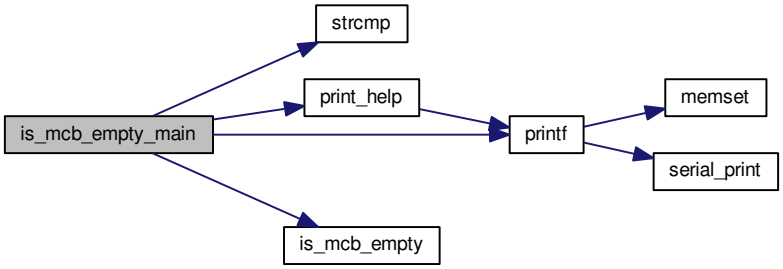
5.18.3.4 int is_mcb_empty ( )

Here is the caller graph for this function:



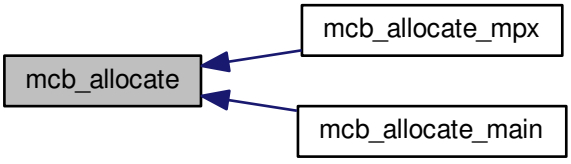
5.18.3.5 int is_mcb_empty_main (int *argc*, char ** *argv*)

Here is the call graph for this function:



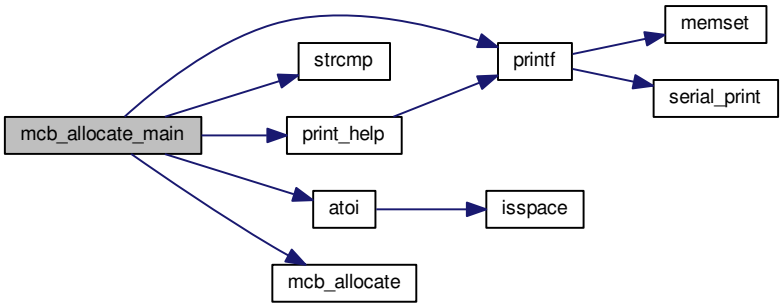
5.18.3.6 void* mcb_allocate (u32int *mem_size*)

Here is the caller graph for this function:



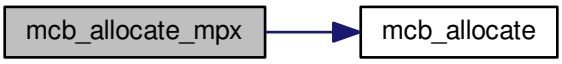
5.18.3.7 int mcb_allocate_main (int *argc,* char ** *argv*)

Here is the call graph for this function:



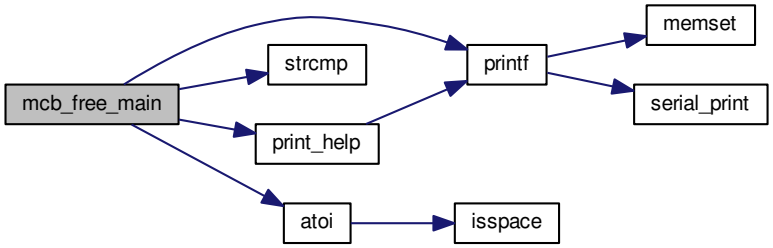
5.18.3.8 u32int mcb_allocate_mpx (u32int *size*)

Here is the call graph for this function:



5.18.3.9 int mcb_free_main (int *argc,* char ** *argv*)

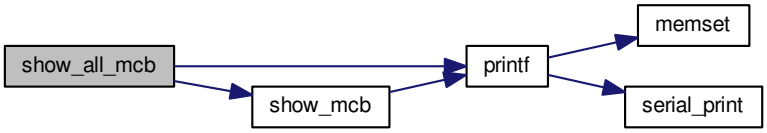
Here is the call graph for this function:



5.18.3.10 int mcb_free_mpx (void * *mem_ptr*)

5.18.3.11 void show_all_mcb ( )

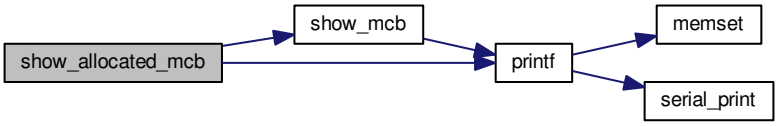
Here is the call graph for this function:



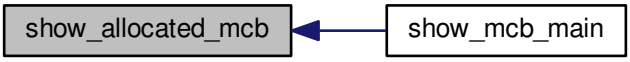
Here is the caller graph for this function:

5.18.3.12 void show_allocated_mcb ( )

Here is the call graph for this function:

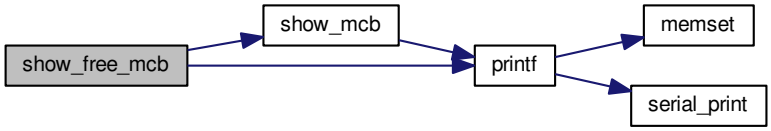


Here is the caller graph for this function:



5.18.3.13 void show_free_mcb ( )

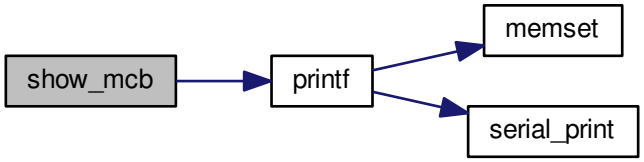
Here is the call graph for this function:



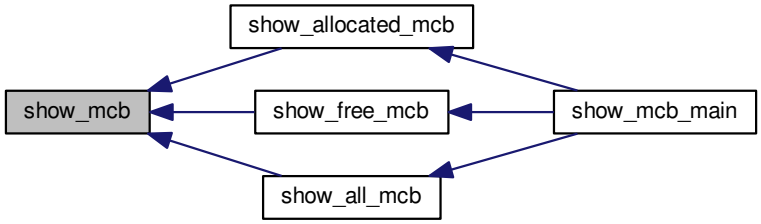
Here is the caller graph for this function:

5.18.3.14 void show_mcb (struct mcb * *mcb_ptr*)

Here is the call graph for this function:

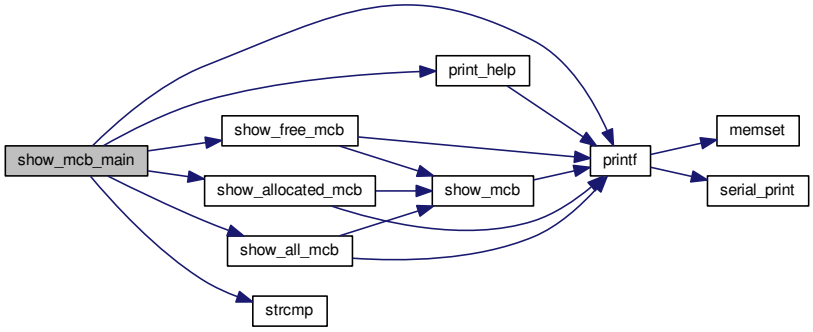


Here is the caller graph for this function:



5.18.3.15 int show_mcb_main (int *argc,* char ** *argv*)

Here is the call graph for this function:



5.18.3.16 void shutdown_mcb ( )

5.18.4 Variable Documentation

5.18.4.1 struct cmcb __attribute__

5.18.4.2 allocated

< Process is a system process.

Process is a system process.

5.18.4.3 struct mcb* allocated_mem_list

5.18.4.4 u32int end_of_memory

5.18.4.5 free

Process is an application process.

5.18.4.6 struct mcb* free_mem_list

5.18.4.7 u32int start_of_memory

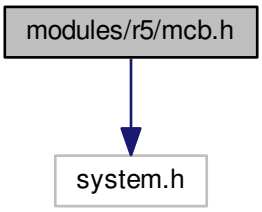
Global variable labeling start of memory.

5.19 modules/r5/mcb.h File Reference

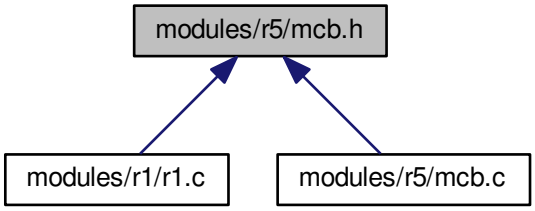
Memory Control Block.

```
#include <system.h>
```

Include dependency graph for mcb.h:



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



Macros

- #define MAX_HEAP_SIZE 5000

Functions

init_heap

Allocates all the memory for MPX

Parameters

size	*Size of heap in bytes*

- void init_heap (u32int size)

mcb_allocate

Allocates a memory block

Parameters

mem_size	*The MCB size to be allocated*

Returns

Address to allocated MCB
NULL if not enough space in free memory found

- void * mcb_allocate (u32int mem_size)

show_mcb

Displays the allocated or free memory block's address, previous and next pointers, and block's size.

Parameters

mcb_ptr	*MCB Pointer*

- void show_mcb (struct mcb *mcb_ptr)

show_free_mcb

Displays all the free memory

- void show_free_mcb ()

show_allocated_mcb

Displays all the allocated MCBs

- void show_allocated_mcb ()

show_all_mcb

Displays all the free and allocated memory

- void show_all_mcb ()

is_mcb_empty

Checks if the heap is empty

Returns

0 or 1 (true or false)

- int is_mcb_empty ()

mcb_free_mpx

Calls mcb_free to free memory block, used as parameter for sys_set_free in kmain.c

Parameters

mem_ptr	*Memory Pointer*

Returns

0

- int mcb_free_mpx (void *mem_ptr)

mcb_allocate_mpx

Calls mcb_allocate to allocate memory block, used as parameter for sys_set_malloc in kmain.c

Parameters

size	*Size of block in bytes to allocate*

Returns

Address of allocated MCB

- u32int mcb_allocate_mpx (u32int size)

mcb_allocate_mpx2

MCB allocate MPX

Parameters

mem_size	*Block size to allocate*
name	*name of the pcb process*

Returns

Address pointer to allocated memory only used for testing in commhand for module R5

- void * mcb_allocate_mpx2 (u32int size, const char *name)

show_mcb_main.

The function of show MCB for commhand.

Parameters

argc	*The number of tokens found.*
argv	*The array of tokens.*

Returns

0

- int show_mcb_main (int argc, char **argv)

shutdown_mcb.

Shutdown the pcb during the shutdown procedure.

Returns

0

- void shutdown_mcb ()

Variables

- u32int start_of_memory

  Global variable labeling start of memory.

5.19.1 Detailed Description

Memory Control Block.

Author

Thunder Krakens

Date

April 8th, 2016

Version

R5

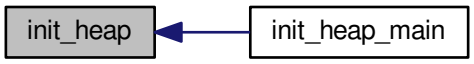
5.19.2 Macro Definition Documentation

5.19.2.1 #define MAX_HEAP_SIZE 5000

5.19.3 Function Documentation

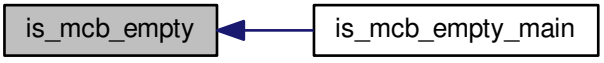
5.19.3.1 void init_heap (u32int *size*)

Here is the caller graph for this function:



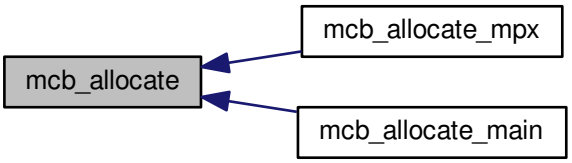
5.19.3.2 int is_mcb_empty ( )

Here is the caller graph for this function:



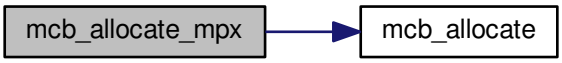
5.19.3.3 void* mcb_allocate (u32int *mem_size*)

Here is the caller graph for this function:



5.19.3.4 u32int mcb_allocate_mpx (u32int *size*)

Here is the call graph for this function:

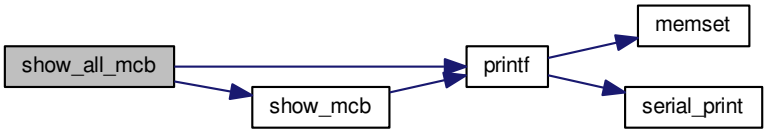


5.19.3.5 void* mcb_allocate_mpx2 (u32int *size,* const char * *name*)

5.19.3.6 int mcb_free_mpx (void * *mem_ptr*)

5.19.3.7 void show_all_mcb ( )

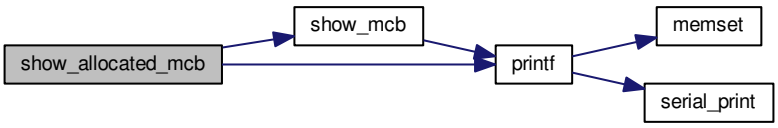
Here is the call graph for this function:



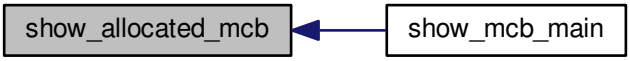
Here is the caller graph for this function:

5.19.3.8 void show_allocated_mcb ( )

Here is the call graph for this function:

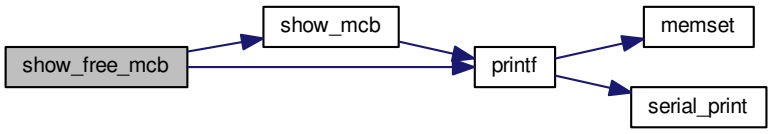


Here is the caller graph for this function:



5.19.3.9 void show_free_mcb ( )

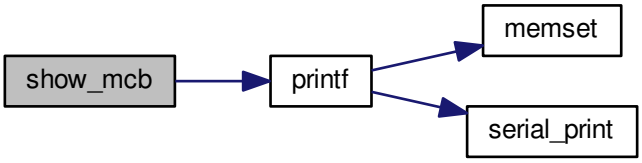
Here is the call graph for this function:



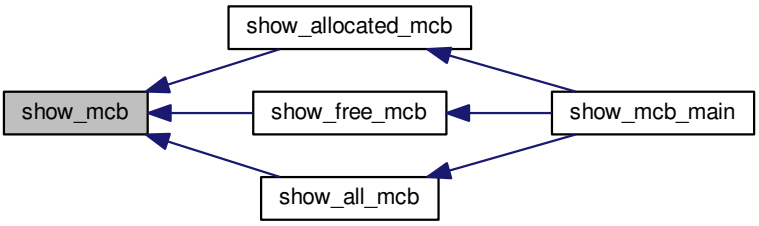
Here is the caller graph for this function:

5.19.3.10 void show_mcb (struct mcb * *mcb_ptr*)

Here is the call graph for this function:

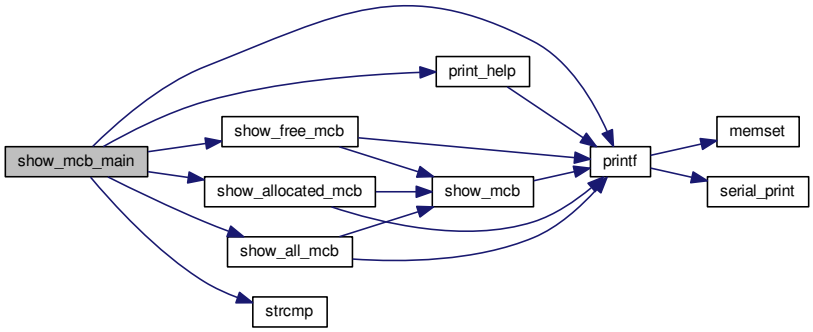


Here is the caller graph for this function:



5.19.3.11 int show_mcb_main (int *argc*, char ** *argv*)

Here is the call graph for this function:



5.19.3.12 void shutdown_mcb ( )

5.19.4 Variable Documentation

5.19.4.1 u32int start_of_memory

Global variable labeling start of memory.

Index