

MPX Thunder Krakens

R4

Generated by Doxygen 1.8.6

Fri Apr 8 2016 01:17:21

Contents

1	Main Page	1
2	Data Structure Index	3
2.1	Data Structures	3
3	File Index	5
3.1	File List	5
4	Data Structure Documentation	7
4.1	cmcb Struct Reference	7
4.1.1	Detailed Description	7
4.1.2	Field Documentation	7
4.1.2.1	begin_address	7
4.1.2.2	size	7
4.1.2.3	type	7
4.2	context Struct Reference	8
4.2.1	Detailed Description	8
4.2.2	Field Documentation	9
4.2.2.1	cs	9
4.2.2.2	ds	9
4.2.2.3	eax	9
4.2.2.4	ebp	9
4.2.2.5	ebx	9
4.2.2.6	ecx	9
4.2.2.7	edi	9
4.2.2.8	edx	9
4.2.2.9	eflags	9
4.2.2.10	eip	9
4.2.2.11	es	9

4.2.2.12	esi	10
4.2.2.13	esp	10
4.2.2.14	fs	10
4.2.2.15	gs	10
4.3	function_name Struct Reference	10
4.3.1	Detailed Description	10
4.3.2	Field Documentation	10
4.3.2.1	function	10
4.3.2.2	help	11
4.3.2.3	nameStr	11
4.3.2.4	usage	11
4.4	Imcb Struct Reference	11
4.4.1	Detailed Description	11
4.4.2	Field Documentation	11
4.4.2.1	size	11
4.4.2.2	type	11
4.5	mcb Struct Reference	12
4.5.1	Detailed Description	12
4.5.2	Field Documentation	12
4.5.2.1	complete_mcb	12
4.5.2.2	limited_mcb	12
4.5.2.3	next	12
4.5.2.4	prev	12
4.6	param Struct Reference	13
4.6.1	Detailed Description	13
4.6.2	Field Documentation	13
4.6.2.1	device_id	13
4.6.2.2	op_code	13
4.7	pcb_queue Struct Reference	13
4.7.1	Detailed Description	14
4.7.2	Field Documentation	14
4.7.2.1	count	14
4.7.2.2	head	14
4.7.2.3	tail	14
4.8	pcb_struct Struct Reference	15
4.8.1	Detailed Description	15
4.8.2	Field Documentation	15

4.8.2.1	class	15
4.8.2.2	is_suspended	16
4.8.2.3	name	16
4.8.2.4	next	16
4.8.2.5	prev	16
4.8.2.6	priority	16
4.8.2.7	running_state	16
4.8.2.8	stack_base	16
4.8.2.9	stack_top	16
5	File Documentation	17
5.1	documentation/mainpage.dox File Reference	17
5.2	include/core/serial.h File Reference	17
5.2.1	Detailed Description	18
5.2.2	Macro Definition Documentation	18
5.2.2.1	COM1	18
5.2.2.2	COM2	18
5.2.2.3	COM3	18
5.2.2.4	COM4	18
5.2.2.5	USER_INPUT_BUFFER_SIZE	18
5.2.2.6	WithEcho	18
5.2.2.7	WithoutEcho	19
5.2.3	Function Documentation	19
5.2.3.1	get_input_line	19
5.2.3.2	init_serial	19
5.2.3.3	serial_print	19
5.2.3.4	serial_println	19
5.2.3.5	set_serial_in	20
5.2.3.6	set_serial_out	20
5.3	include/string.h File Reference	20
5.3.1	Detailed Description	23
5.3.2	Function Documentation	23
5.3.2.1	atoi	24
5.3.2.2	isspace	25
5.3.2.3	memset	26
5.3.2.4	printf	27
5.3.2.5	sprintf	28

5.3.2.6	strcat	28
5.3.2.7	strcmp	29
5.3.2.8	strcpy	29
5.3.2.9	strlen	30
5.3.2.10	strtok	30
5.4	lib/string.c File Reference	30
5.4.1	Detailed Description	34
5.4.2	Function Documentation	34
5.4.2.1	atoi	34
5.4.2.2	isspace	35
5.4.2.3	memset	36
5.4.2.4	printf	37
5.4.2.5	sprintf	38
5.4.2.6	strcat	38
5.4.2.7	strcmp	39
5.4.2.8	strcpy	39
5.4.2.9	strlen	40
5.4.2.10	strtok	40
5.5	modules/errno.h File Reference	40
5.5.1	Detailed Description	41
5.5.2	Macro Definition Documentation	42
5.5.2.1	E_EMPTYPCB	42
5.5.2.2	E_FREEMEM	42
5.5.2.3	E_INVPARA	42
5.5.2.4	E_INVSTRF	42
5.5.2.5	E_INVUSRI	42
5.5.2.6	E_NOERROR	42
5.5.2.7	E_NULL_PTR	42
5.5.2.8	E_PCB_SYS	42
5.5.2.9	E_PROGERR	42
5.5.3	Typedef Documentation	42
5.5.3.1	error_t	42
5.6	modules/mpx_supt.c File Reference	42
5.6.1	Function Documentation	43
5.6.1.1	get_op_code	44
5.6.1.2	idle	44
5.6.1.3	mpx_init	44

5.6.1.4	sys_alloc_mem	44
5.6.1.5	sys_free_mem	45
5.6.1.6	sys_req	45
5.6.1.7	sys_set_free	45
5.6.1.8	sys_set_malloc	45
5.6.2	Variable Documentation	45
5.6.2.1	current_module	45
5.6.2.2	params	45
5.6.2.3	student_free	45
5.6.2.4	student_malloc	45
5.7	modules/mpx_supt.h File Reference	45
5.7.1	Detailed Description	48
5.7.2	Macro Definition Documentation	48
5.7.2.1	EXIT	48
5.7.2.2	IDLE	48
5.7.2.3	MODULE_R1	48
5.7.2.4	MODULE_R2	48
5.7.2.5	MODULE_R3	48
5.7.2.6	MODULE_R4	48
5.7.2.7	MODULE_R5	48
5.7.2.8	READ	48
5.7.2.9	WRITE	49
5.7.3	Function Documentation	49
5.7.3.1	get_op_code	49
5.7.3.2	idle	49
5.7.3.3	mpx_init	49
5.7.3.4	sys_alloc_mem	49
5.7.3.5	sys_free_mem	50
5.7.3.6	sys_req	50
5.7.3.7	sys_set_free	50
5.7.3.8	sys_set_malloc	50
5.7.4	Variable Documentation	50
5.7.4.1	__attribute__	50
5.8	modules/r1/r1.c File Reference	50
5.8.1	Detailed Description	53
5.8.2	Macro Definition Documentation	54
5.8.2.1	COMPLETION	54

5.8.2.2	MAX_ARGC	54
5.8.2.3	MAX_HISTORY	54
5.8.2.4	MOD_VERSION	54
5.8.3	Enumeration Type Documentation	54
5.8.3.1	CommandPaserStat	54
5.8.4	Function Documentation	54
5.8.4.1	__attribute__	54
5.8.4.2	command_line_parser	54
5.8.4.3	commhand	54
5.8.4.4	help_usages	55
5.8.4.5	print_help	55
5.8.5	Variable Documentation	56
5.8.5.1	DoubleQuoteWriting	56
5.8.5.2	NormalWriting	57
5.8.5.3	NotWriting	57
5.8.5.4	SingleQuoteWriting	57
5.9	modules/r1/r1.h File Reference	57
5.9.1	Detailed Description	58
5.9.2	Enumeration Type Documentation	59
5.9.2.1	comm_type	59
5.9.3	Function Documentation	59
5.9.3.1	__attribute__	59
5.9.3.2	command_line_parser	59
5.9.3.3	commhand	59
5.9.3.4	help_usages	59
5.9.3.5	print_help	60
5.9.4	Variable Documentation	61
5.9.4.1	help	61
5.9.4.2	mcb	62
5.9.4.3	mpx	62
5.9.4.4	pcb	62
5.10	modules/r1/sys_clock.c File Reference	62
5.10.1	Detailed Description	67
5.10.2	Macro Definition Documentation	68
5.10.2.1	RTC_INDEX_DAY_MONTH	68
5.10.2.2	RTC_INDEX_DAY_WEEK	68
5.10.2.3	RTC_INDEX_HOUR	68

5.10.2.4	RTC_INDEX_HOUR_ALARM	68
5.10.2.5	RTC_INDEX_MINUTE	68
5.10.2.6	RTC_INDEX_MINUTE_ALARM	68
5.10.2.7	RTC_INDEX_MONTH	68
5.10.2.8	RTC_INDEX_SECOND	68
5.10.2.9	RTC_INDEX_SECOND_ALARM	68
5.10.2.10	RTC_INDEX_YEAR	68
5.10.3	Function Documentation	68
5.10.3.1	get_date	68
5.10.3.2	get_date_main	69
5.10.3.3	get_time	69
5.10.3.4	get_time_main	70
5.10.3.5	set_date	70
5.10.3.6	set_date_main	71
5.10.3.7	set_date_str	71
5.10.3.8	set_time	72
5.10.3.9	set_time_main	73
5.10.3.10	set_time_str	73
5.11	modules/r1/sys_clock.h File Reference	74
5.11.1	Detailed Description	77
5.11.2	Function Documentation	77
5.11.2.1	get_date	77
5.11.2.2	get_date_main	78
5.11.2.3	get_time	78
5.11.2.4	get_time_main	79
5.11.2.5	set_date	79
5.11.2.6	set_date_main	80
5.11.2.7	set_date_str	80
5.11.2.8	set_time	81
5.11.2.9	set_time_main	82
5.11.2.10	set_time_str	82
5.12	modules/r2/pcb.c File Reference	83
5.12.1	Detailed Description	89
5.12.2	Enumeration Type Documentation	89
5.12.2.1	process_state	89
5.12.2.2	process_suspended	89
5.12.3	Function Documentation	89

5.12.3.1	__attribute__	89
5.12.3.2	allocate_pcb	89
5.12.3.3	block_pcb	90
5.12.3.4	find_pcb	90
5.12.3.5	free_pcb	91
5.12.3.6	get_running_process	92
5.12.3.7	get_stack_base	92
5.12.3.8	get_stack_top	93
5.12.3.9	insert_pcb	93
5.12.3.10	pcb_init	93
5.12.3.11	remove_pcb	94
5.12.3.12	resume_pcb	94
5.12.3.13	save_running_process	94
5.12.3.14	set_pcb_priority	95
5.12.3.15	setup_pcb	96
5.12.3.16	show_all_processes	96
5.12.3.17	show_blocked_processes	97
5.12.3.18	show_pcb	98
5.12.3.19	show_ready_processes	98
5.12.3.20	shutdown_pcb	99
5.12.3.21	suspend_pcb	99
5.12.3.22	unblock_pcb	100
5.12.4	Variable Documentation	100
5.12.4.1	__attribute__	100
5.12.4.2	blocked	100
5.12.4.3	false	100
5.12.4.4	ready	100
5.12.4.5	running	100
5.12.4.6	true	100
5.13	modules/r2/pcb.h File Reference	100
5.13.1	Detailed Description	106
5.13.2	Macro Definition Documentation	107
5.13.2.1	COMMHAND_PCB_NAME	107
5.13.2.2	IDLE_PCB_NAME	107
5.13.2.3	SIZE_OF_PCB_NAME	107
5.13.2.4	SIZE_OF_STACK	107
5.13.3	Enumeration Type Documentation	107

5.13.3.1	process_class	107
5.13.4	Function Documentation	107
5.13.4.1	__attribute__	107
5.13.4.2	allocate_pcb	107
5.13.4.3	block_pcb	108
5.13.4.4	find_pcb	108
5.13.4.5	free_pcb	109
5.13.4.6	get_running_process	109
5.13.4.7	get_stack_base	110
5.13.4.8	get_stack_top	110
5.13.4.9	insert_pcb	111
5.13.4.10	pcb_init	111
5.13.4.11	remove_pcb	111
5.13.4.12	resume_pcb	112
5.13.4.13	save_running_process	112
5.13.4.14	set_pcb_priority	113
5.13.4.15	setup_pcb	113
5.13.4.16	show_all_processes	114
5.13.4.17	show_blocked_processes	114
5.13.4.18	show_pcb	115
5.13.4.19	show_ready_processes	116
5.13.4.20	shutdown_pcb	116
5.13.4.21	suspend_pcb	117
5.13.4.22	unblock_pcb	117
5.13.5	Variable Documentation	117
5.13.5.1	pcb_class_app	117
5.13.5.2	pcb_class_sys	117
5.14	modules/r2/pcb_comm.c File Reference	117
5.14.1	Detailed Description	119
5.14.2	Function Documentation	120
5.14.2.1	resume_pcb_main	120
5.14.2.2	set_pcb_priority_main	120
5.14.2.3	show_pcb_main	121
5.14.2.4	suspend_pcb_main	121
5.15	modules/r2/pcb_comm.h File Reference	121
5.15.1	Detailed Description	124
5.15.2	Function Documentation	125

5.15.2.1	resume_pcb_main	125
5.15.2.2	set_pcb_priority_main	125
5.15.2.3	show_pcb_main	126
5.15.2.4	suspend_pcb_main	126
5.16	modules/r3/context.c File Reference	126
5.16.1	Detailed Description	128
5.16.2	Function Documentation	128
5.16.2.1	load_process	129
5.16.2.2	load_r3_main	130
5.16.2.3	sys_call	130
5.16.3	Variable Documentation	131
5.16.3.1	cop	131
5.16.3.2	old_context	131
5.17	modules/r3/context.h File Reference	131
5.17.1	Detailed Description	133
5.17.2	Function Documentation	133
5.17.2.1	load_process	133
5.17.2.2	load_r3_main	134
5.17.2.3	sys_call	135
5.17.3	Variable Documentation	135
5.17.3.1	cop	135
5.17.3.2	old_context	135
5.18	modules/r5/mcb.c File Reference	135
5.18.1	Detailed Description	142
5.18.2	Enumeration Type Documentation	142
5.18.2.1	mcb_type	142
5.18.3	Function Documentation	142
5.18.3.1	__attribute__	142
5.18.3.2	init_heap	142
5.18.3.3	init_heap_main	143
5.18.3.4	is_mcb_empty	143
5.18.3.5	is_mcb_empty_main	144
5.18.3.6	mcb_allocate	144
5.18.3.7	mcb_allocate_main	145
5.18.3.8	mcb_allocate_mpx	145
5.18.3.9	mcb_free_main	146
5.18.3.10	mcb_free_mpx	146

5.18.3.11	show_all_mcb	146
5.18.3.12	show_allocated_mcb	147
5.18.3.13	show_free_mcb	147
5.18.3.14	show_mcb	148
5.18.3.15	show_mcb_main	149
5.18.3.16	shutdown_mcb	149
5.18.4	Variable Documentation	149
5.18.4.1	__attribute__	149
5.18.4.2	allocated	149
5.18.4.3	allocated_mem_list	149
5.18.4.4	end_of_memory	149
5.18.4.5	free	149
5.18.4.6	free_mem_list	149
5.18.4.7	start_of_memory	149
5.19	modules/r5/mcb.h File Reference	149
5.19.1	Detailed Description	153
5.19.2	Macro Definition Documentation	153
5.19.2.1	MAX_HEAP_SIZE	153
5.19.3	Function Documentation	153
5.19.3.1	init_heap	153
5.19.3.2	is_mcb_empty	153
5.19.3.3	mcb_allocate	154
5.19.3.4	mcb_allocate_mpx	154
5.19.3.5	mcb_allocate_mpx2	154
5.19.3.6	mcb_free_mpx	154
5.19.3.7	show_all_mcb	154
5.19.3.8	show_allocated_mcb	155
5.19.3.9	show_free_mcb	156
5.19.3.10	show_mcb	156
5.19.3.11	show_mcb_main	157
5.19.3.12	shutdown_mcb	157
5.19.4	Variable Documentation	157
5.19.4.1	start_of_memory	157

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 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:

cmcb	Complete Memory Control Block Struct	7
context	Context structure that holds the 15 CPU register values to begin and resume process execution . . .	8
function_name	A structure to represent each function	10
lmcb	Limited Memory Control Block Struct	11
mcb	Memory Control Block Struct	12
param	A structure to represent interrupt	13
pcb_queue	Queue structure that will store PCBs	13
pcb_struct	Struct that will describe PCB Processes	15

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

include/ string.h	Many usefull functions that used for handling string	20
include/core/ serial.h	Serial - Header	17
lib/ string.c	Many usefull functions that used for handling string	30
modules/ errno.h	This file contains the type of errors. The error can be from invalid paramter passed to a function, or invalid input format	40
modules/ mpx_supt.c		42
modules/ mpx_supt.h	MPX System Supplementaries	45
modules/r1/ r1.c	The commandhander and functions associations for Module R1	50
modules/r1/ r1.h	The command handler and functions associations for Module R1	57
modules/r1/ sys_clock.c	The main file that manipulates and controls the system's clock	62
modules/r1/ sys_clock.h	The main file that manipulates and controls the system's clock	74
modules/r2/ pcb.c	The Process Control Block	83
modules/r2/ pcb.h	The Process Control Block	100
modules/r2/ pcb_comm.c	The main functions that manipulate the PCB	117
modules/r2/ pcb_comm.h	The main functions that manipulate the PCB	121
modules/r3/ context.c	Context Switching	126
modules/r3/ context.h	Context Switching	131
modules/r5/ mcb.c	Memory Control Block	135

modules/r5/ mcb.h	
Memory Control Block	149

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`
function'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 Imcb 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](#) Imcb::size

Indicates size of block in bytes.

4.4.2.2 [enum mcb_type](#) Imcb::type

Type indicating free or allocated.

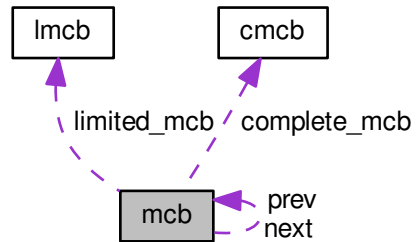
The documentation for this struct was generated from the following file:

- [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

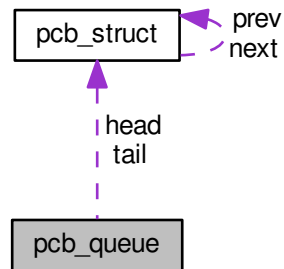
The documentation for this struct was generated from the following file:

- [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.

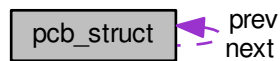
The documentation for this struct was generated from the following file:

- `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

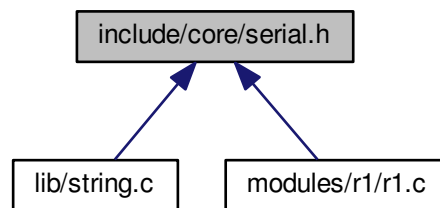
File Documentation

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

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

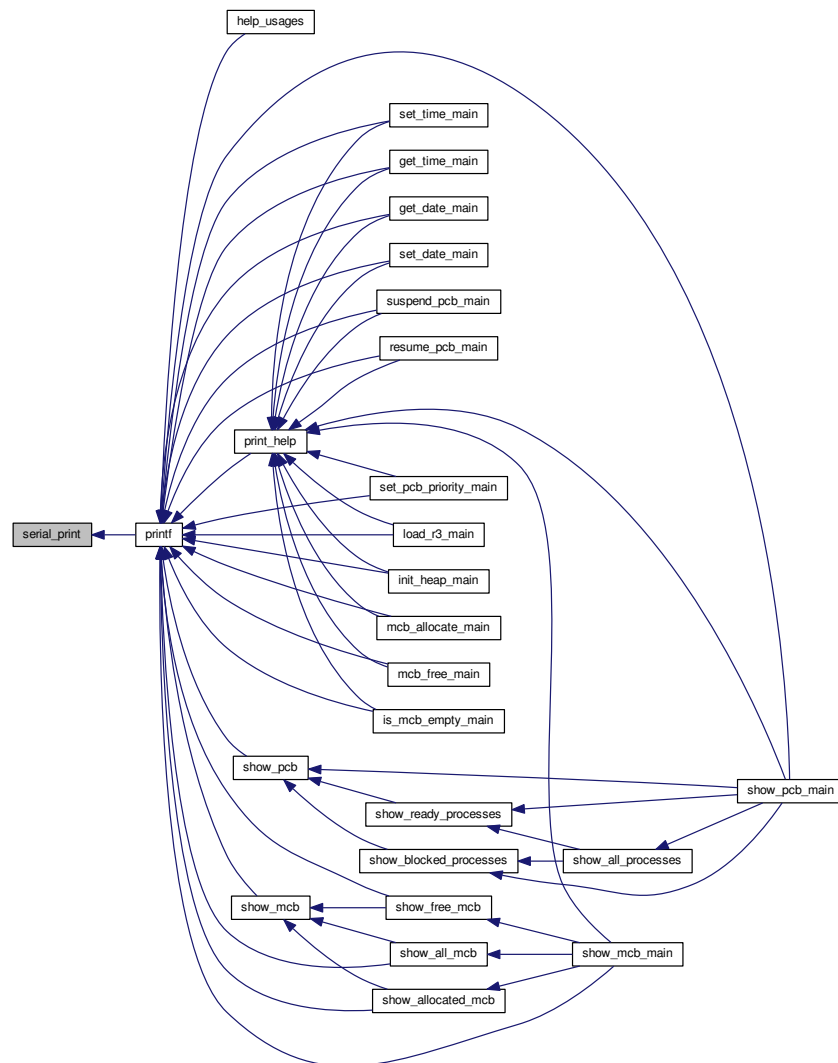
5.2.3 Function Documentation

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)`

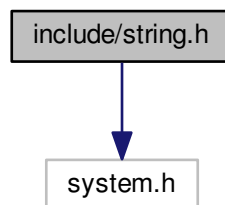
5.2.3.6 `int set_serial_out (int device)`

5.3 include/string.h File Reference

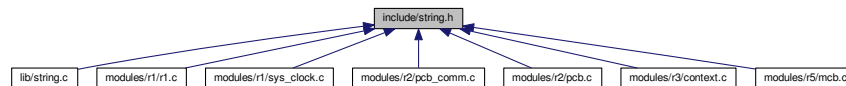
Many usefull functions that used for handling string.

```
#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

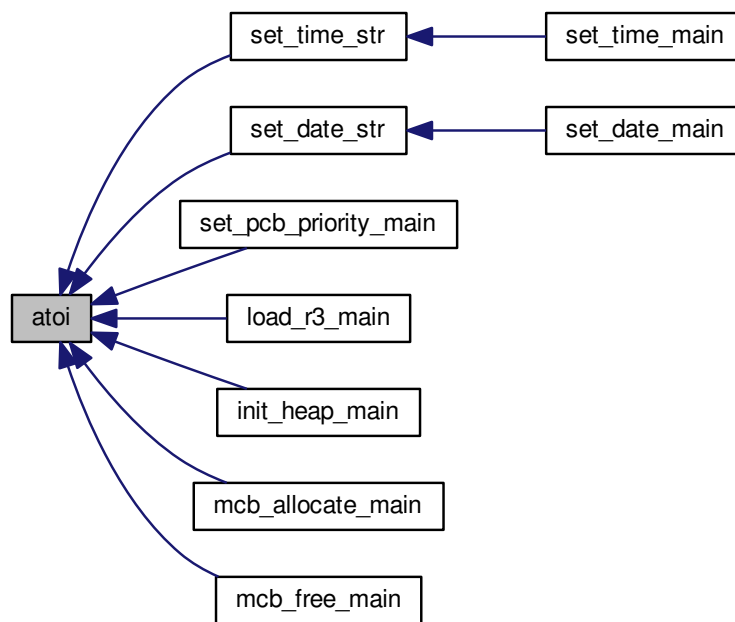
5.3.2 Function Documentation

5.3.2.1 int atoi (const char * s)

Here is the call graph for this function:

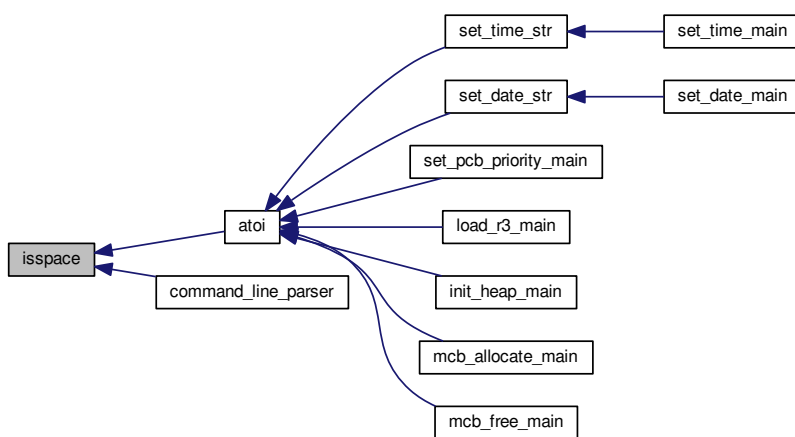


Here is the caller graph for this function:



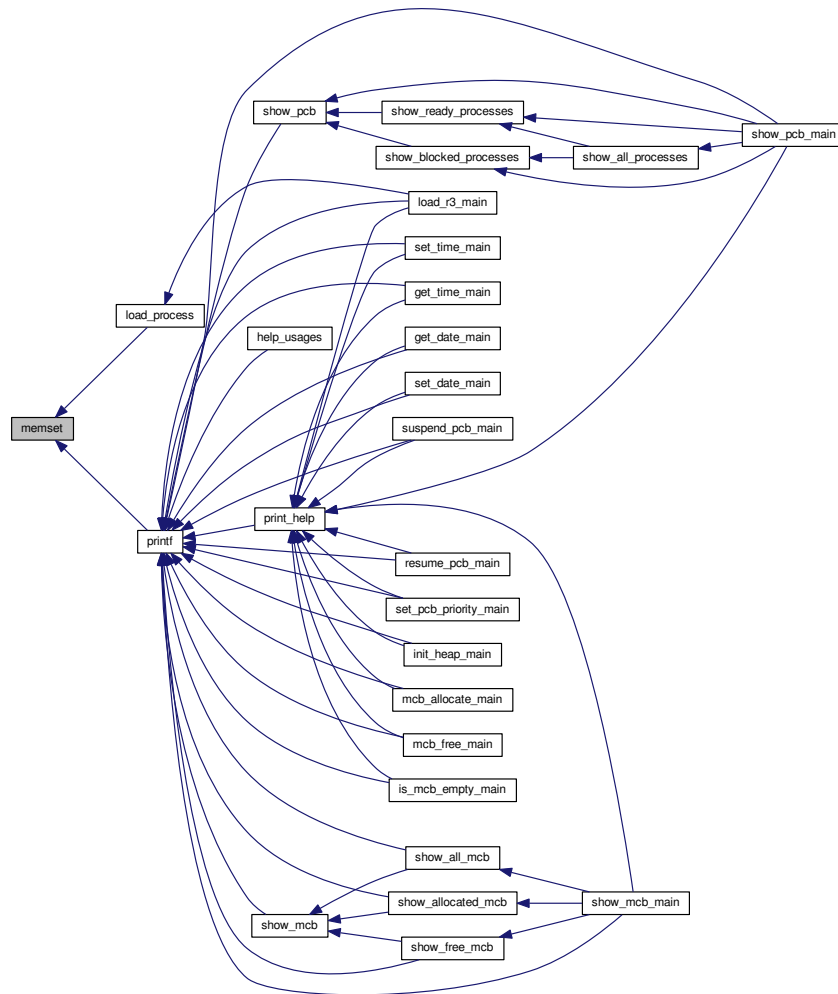
5.3.2.2 int isspace (const char * c)

Here is the caller graph for this function:



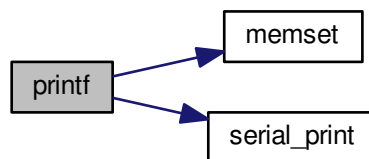
5.3.2.3 void* memset (void * s, int c, size_t n)

Here is the caller graph for this function:

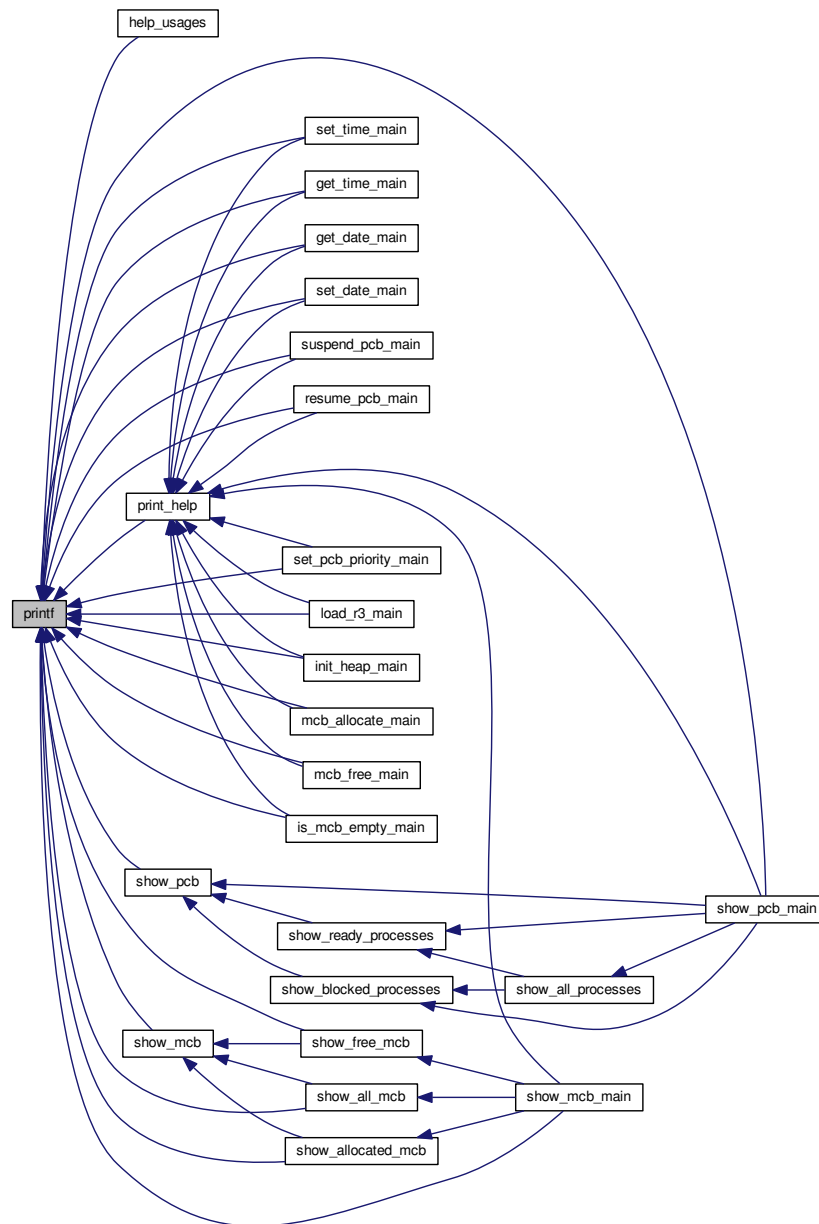


5.3.2.4 int printf (const char * *format*, ...)

Here is the call graph for this function:



Here is the caller graph for this function:

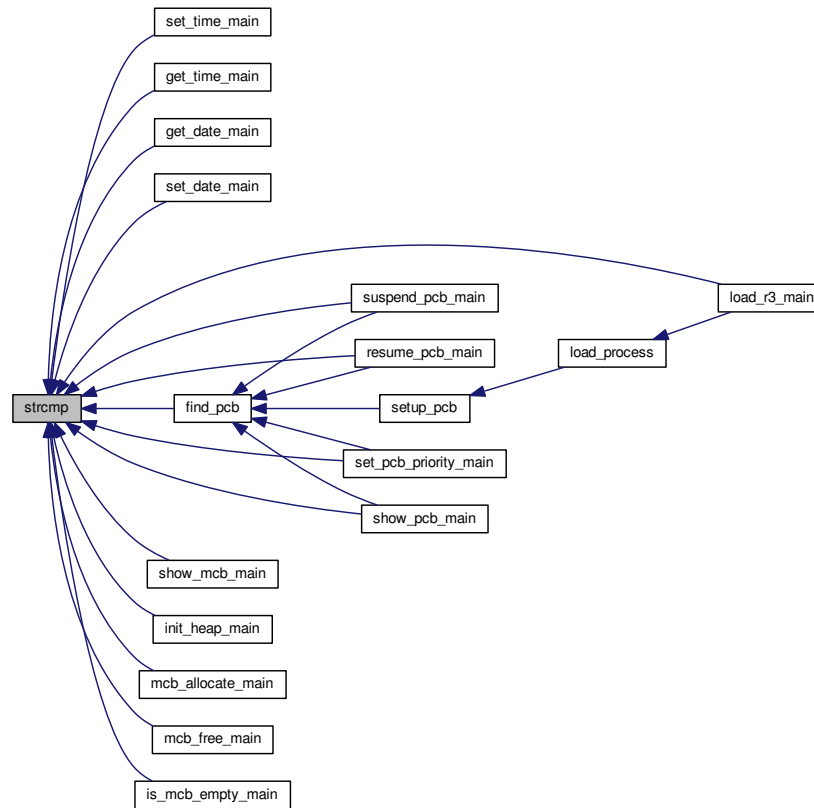


5.3.2.5 `int sprintf (char * str, const char * format, ...)`

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

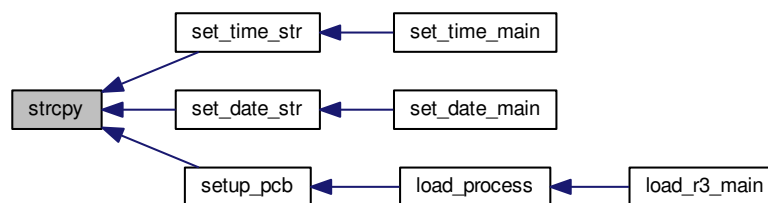
5.3.2.7 int strcmp (const char * s1, const char * s2)

Here is the caller graph for this function:



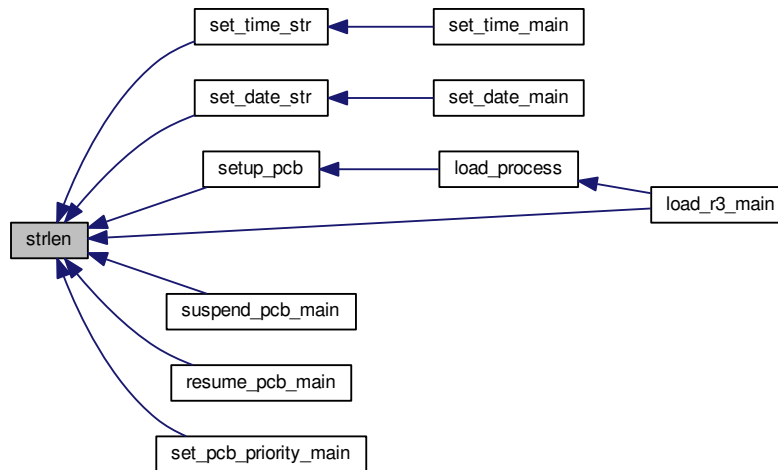
5.3.2.8 char* strcpy (char * s1, const char * s2)

Here is the caller graph for this function:



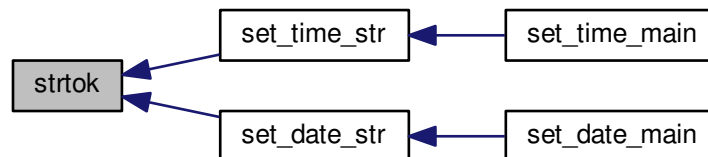
5.3.2.9 int strlen (const char * s)

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

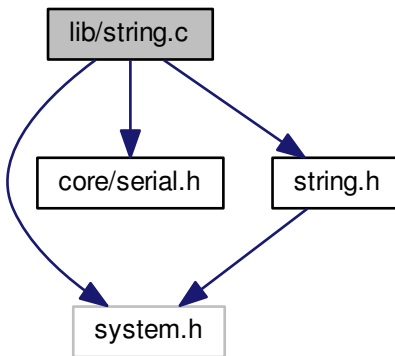
Many usefull functions that used for handling string.

```

#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.
blsRight	Is align right.
bHasSign	Has + / -.

Returns

blsValid 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

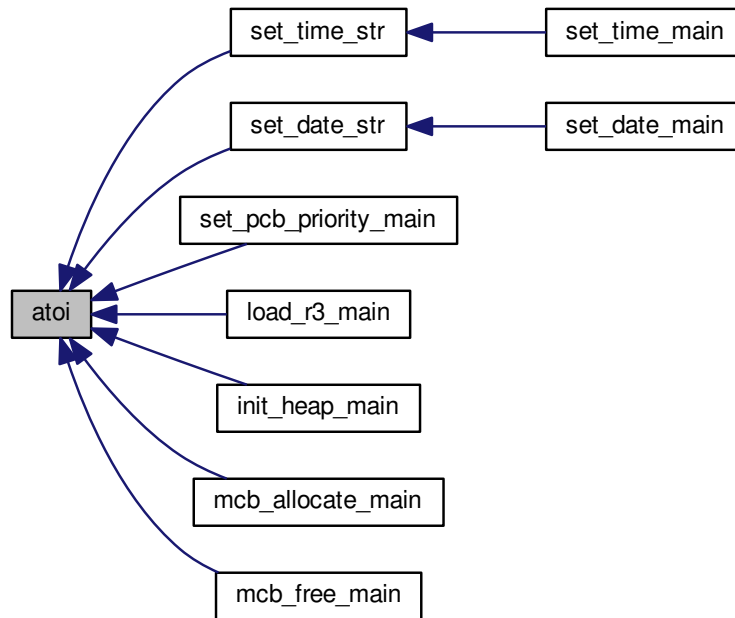
5.4.2 Function Documentation

5.4.2.1 int atoi (const char * s)

Here is the call graph for this function:

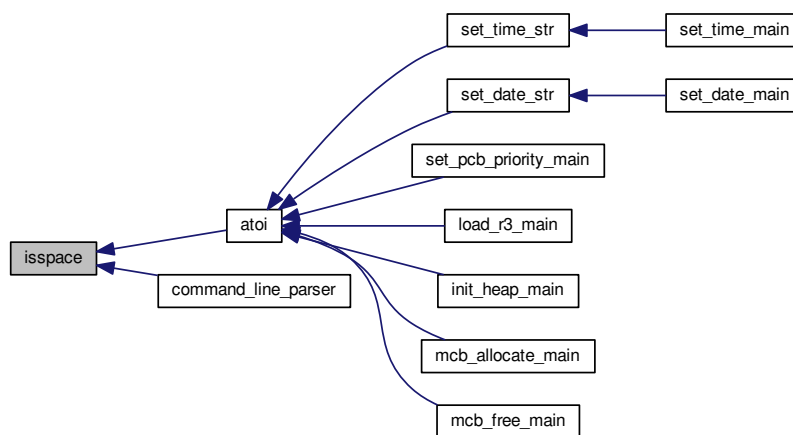


Here is the caller graph for this function:



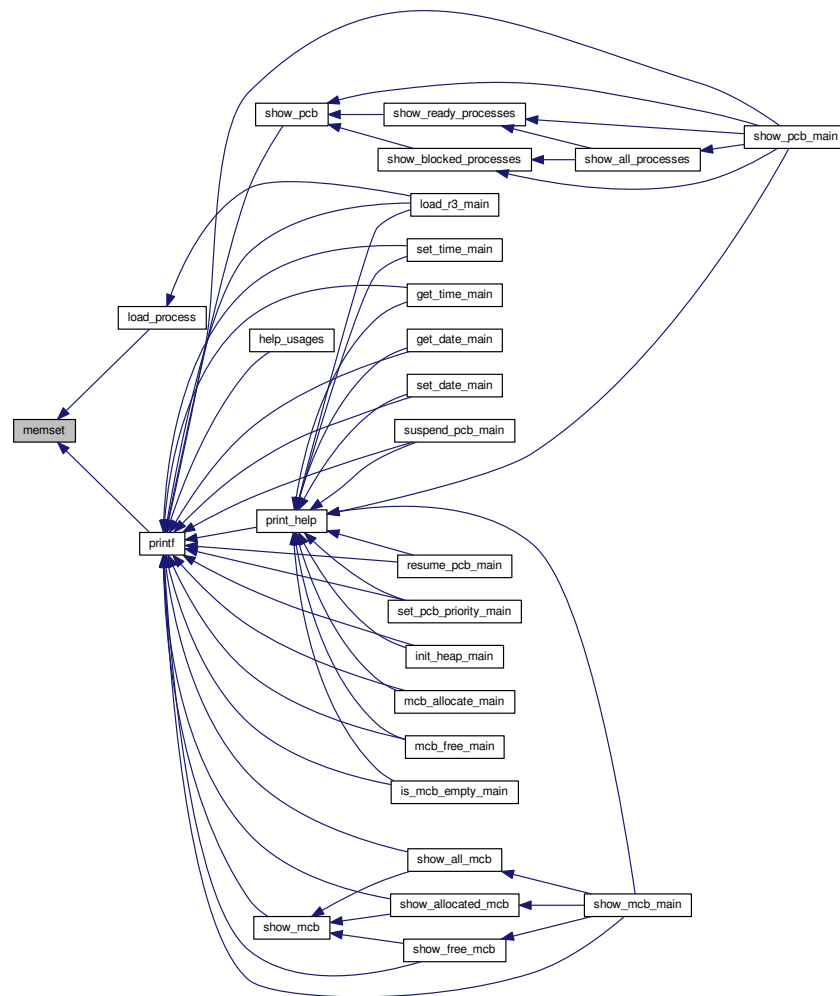
5.4.2.2 `int isspace (const char * c)`

Here is the caller graph for this function:



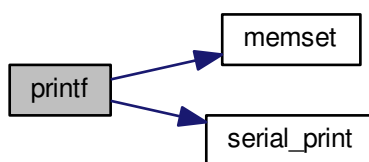
5.4.2.3 void* memset (void * s, int c, size_t n)

Here is the caller graph for this function:

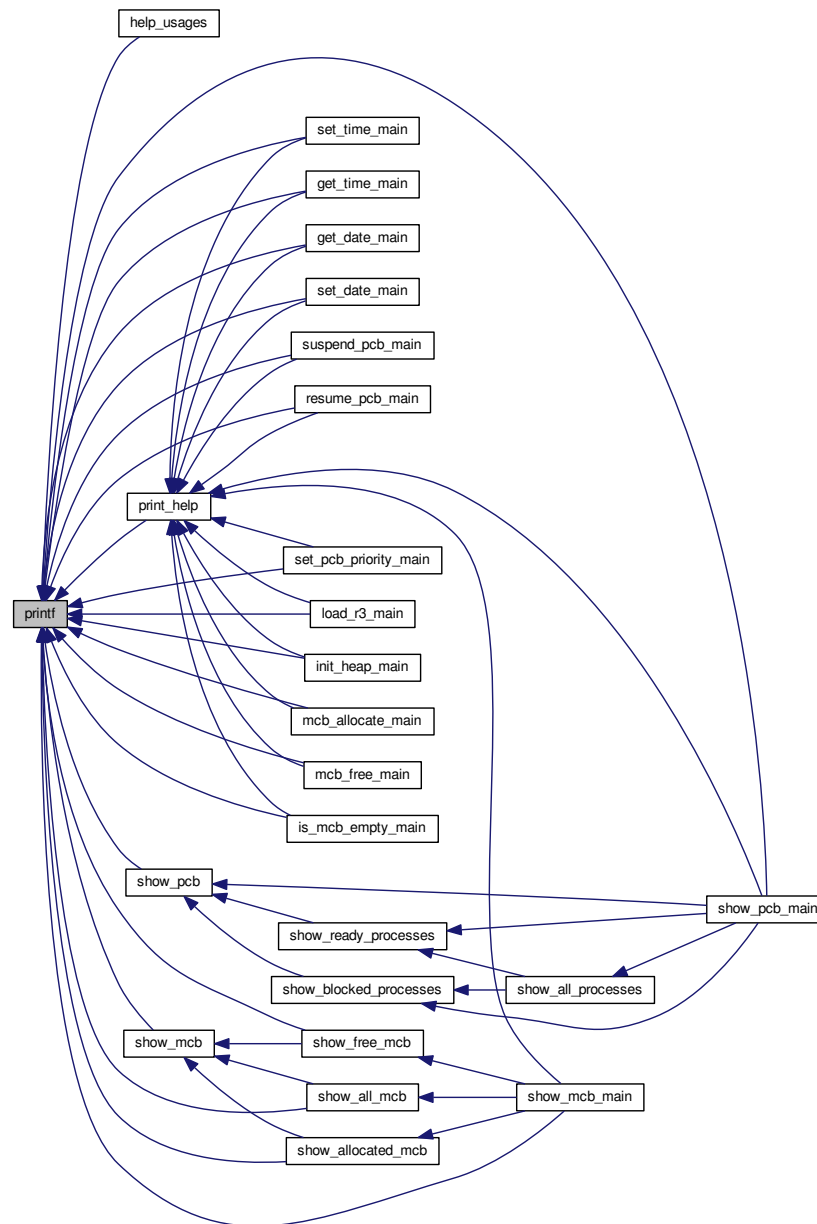


5.4.2.4 int printf (const char * *format*, ...)

Here is the call graph for this function:



Here is the caller graph for this function:

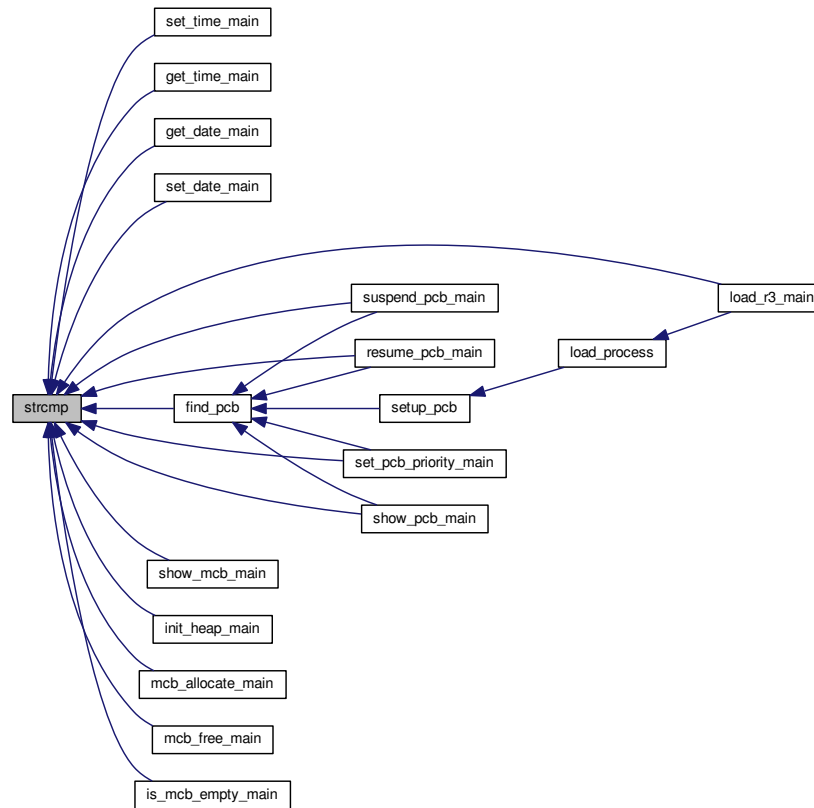


5.4.2.5 `int sprintf (char * str, const char * format, ...)`

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

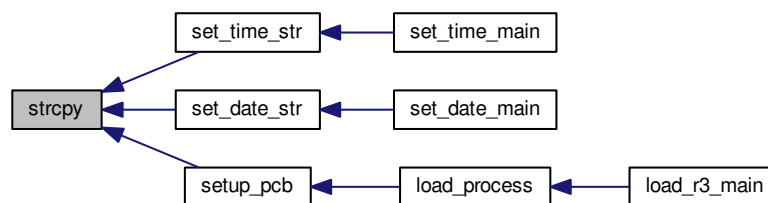
5.4.2.7 int strcmp (const char * s1, const char * s2)

Here is the caller graph for this function:



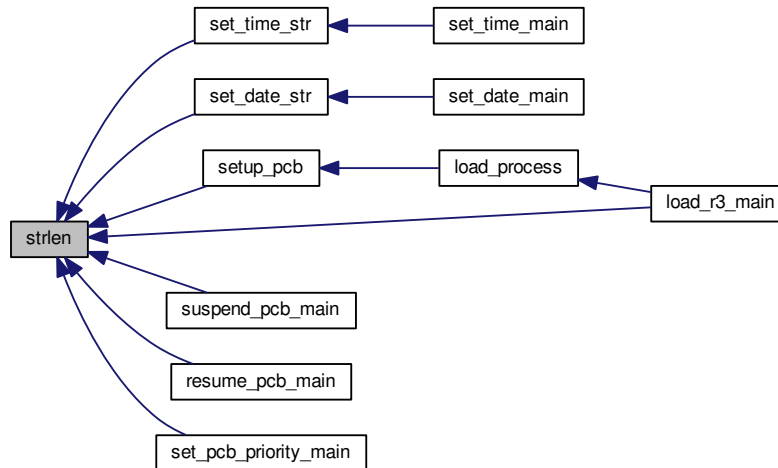
5.4.2.8 char* strcpy (char * s1, const char * s2)

Here is the caller graph for this function:



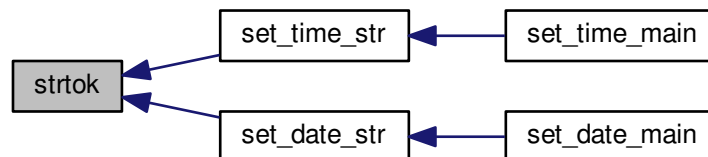
5.4.2.9 `int strlen (const char * s)`

Here is the caller graph for this function:



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

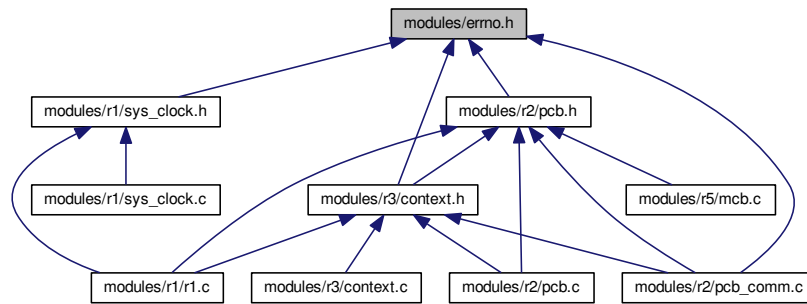
Here is the caller graph for this function:



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_EMPTYPCB 6`
The pcb queue is empty.
- `#define E_PCB_SYS 7`
- `#define E_PROGERR 99`

Typedefs

`error_t`.

The datatype 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

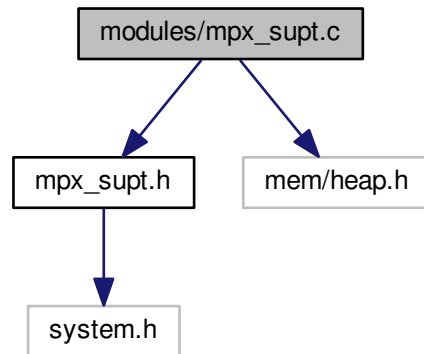
5.5.3 Typedef Documentation

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 ()`

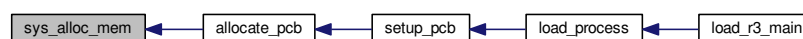
Here is the call graph for this function:



5.6.1.3 `void mpx_init (int cur_mod)`

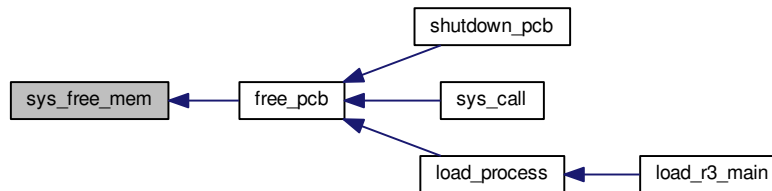
5.6.1.4 `void* sys_alloc_mem (u32int size)`

Here is the caller graph for this function:

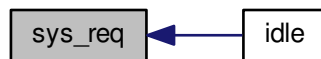


5.6.1.5 `int sys_free_mem (void * ptr)`

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

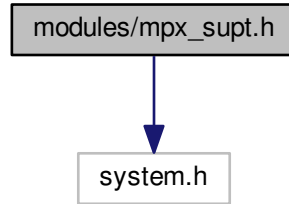
5.6.2.1 `int current_module = -1`5.6.2.2 `param params`5.6.2.3 `int(* student_free)(void *)`5.6.2.4 `u32int(* student_malloc)(u32int)`

5.7 modules/mpx_supt.h File Reference

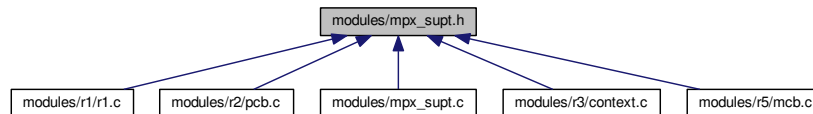
MPX System Supplementaries.

```
#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	<i>op_code (IDLE)</i>
-----	-----------------------

- int [sys_req](#) (int op_code)

mpx_init*Initialize MPX support software**Parameters*

int	<i>cur_mod (symbolic constants MODULE_R1, MODULE_R2, etc</i>
-----	--

- void [mpx_init](#) (int cur_mod)

set_malloc*Sets the memory allocation function for sys_alloc_mem**Parameters*

Function	<i>pointer</i>
----------	----------------

- 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	<i>of bytes to allocate</i>
--------	-----------------------------

- void * [sys_alloc_mem](#) (u32int size)

sys_free_mem*Frees memory**Parameters*

Pointer	<i>to block of memory to free</i>
---------	-----------------------------------

- 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 ()`

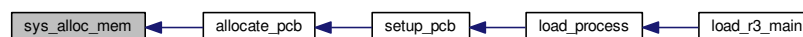
Here is the caller graph for this function:

5.7.3.2 `void idle ()`

Here is the call graph for this function:

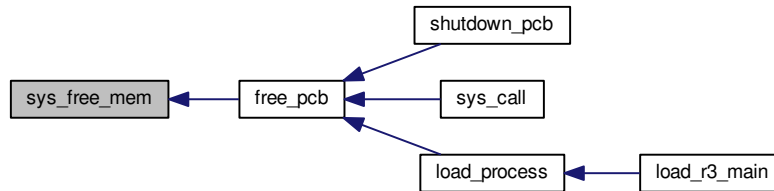
5.7.3.3 `void mpx_init (int cur_mod)`5.7.3.4 `void* sys_alloc_mem (u32int size)`

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

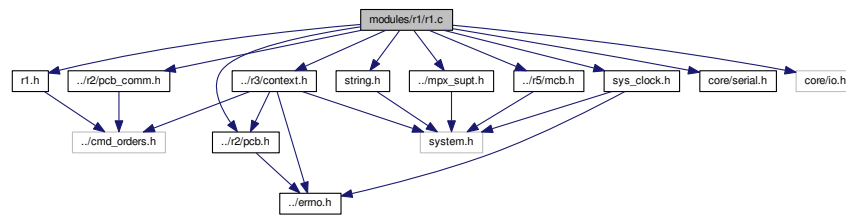
The commandhandler and functions associations for Module R1.


```

#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 function.

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	<i>The complete input command.</i>
argc	<i>The number of tokens found.</i>
argv	<i>The array of tokens.</i>
MaxArgNum	<i>The maximum number of tokens that array can hold.</i>
MaxStrLen	<i>The maximum length of each token that string can hold.</i>

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	<i>The index number of that function.</i>
----------------	---

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 commandhandler 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

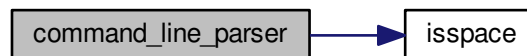
5.8.3.1 `enum CommandPaserStat`

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)`

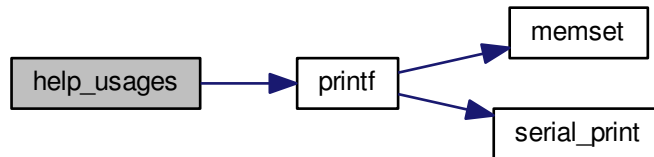
Here is the call graph for this function:



5.8.4.3 `void commhand ()`

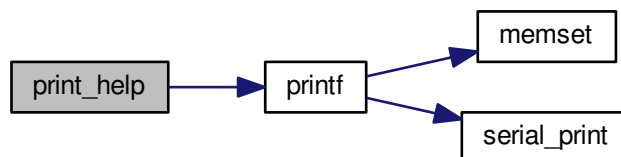
5.8.4.4 `int help_usages (enum comm_type type)`

Here is the call graph for this function:

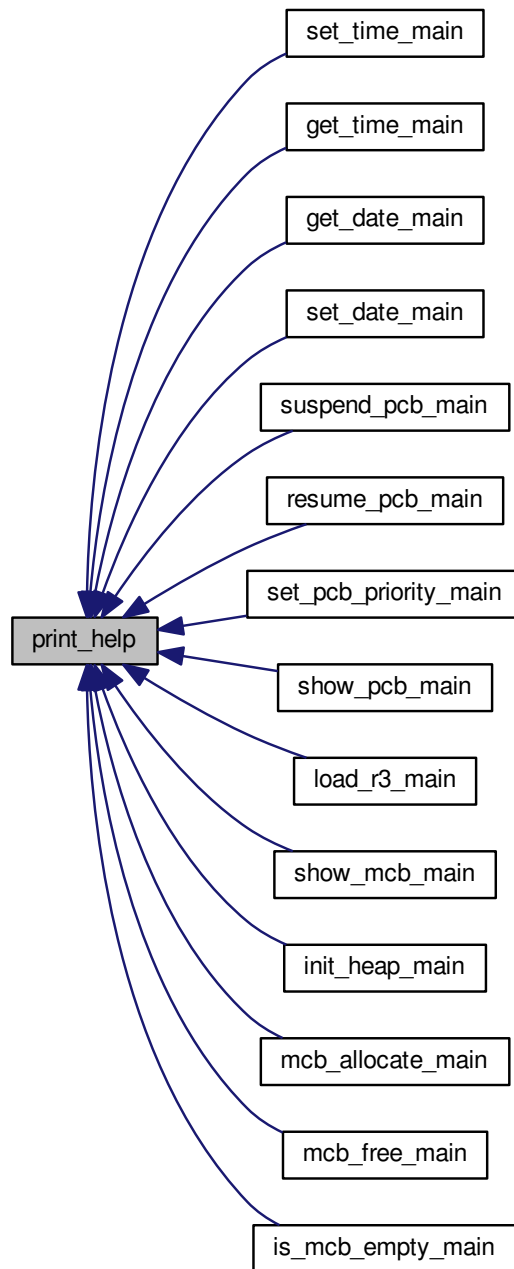


5.8.4.5 `void print_help (const int function_index)`

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

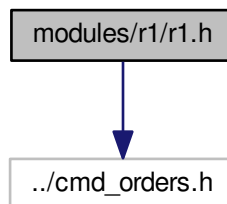
5.8.5.4 SingleQuoteWriting

5.9 modules/r1/r1.h File Reference

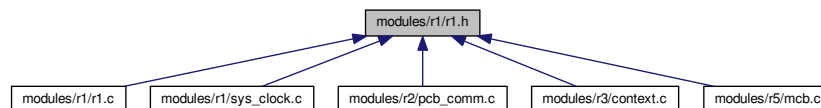
The command handler and functions associations for Module R1.

```
#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	<i>The complete input command.</i>
argc	<i>The number of tokens found.</i>
argv	<i>The array of tokens.</i>
MaxArgNum	<i>The maximum number of tokens that array can hold.</i>
MaxStrLen	<i>The maximum length of each token that string can hold.</i>

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	<i>The index number of that function.</i>
----------------	---

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

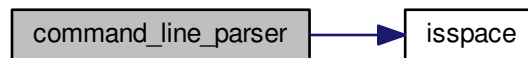
5.9.2.1 enum comm_type

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*)

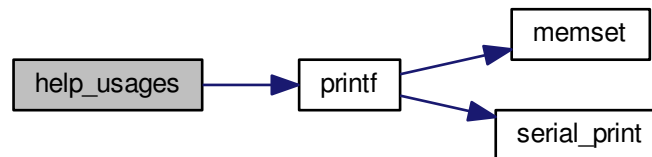
Here is the call graph for this function:



5.9.3.3 void commhand ()

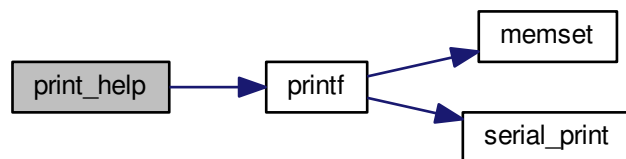
5.9.3.4 int help_usages (enum comm_type *type*)

Here is the call graph for this function:

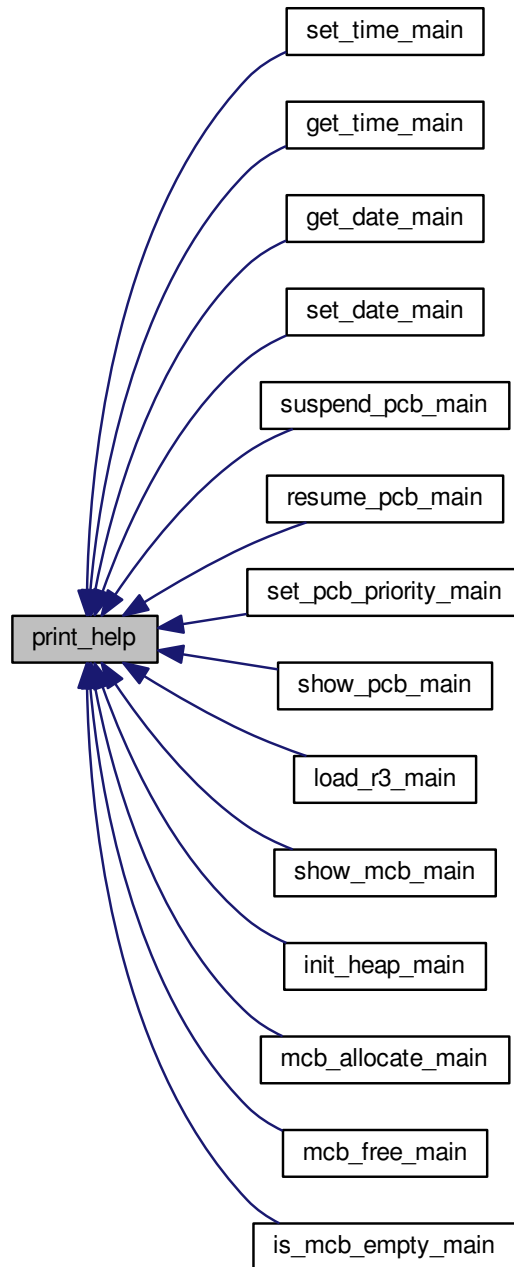


5.9.3.5 void print_help (const int *function_index*)

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

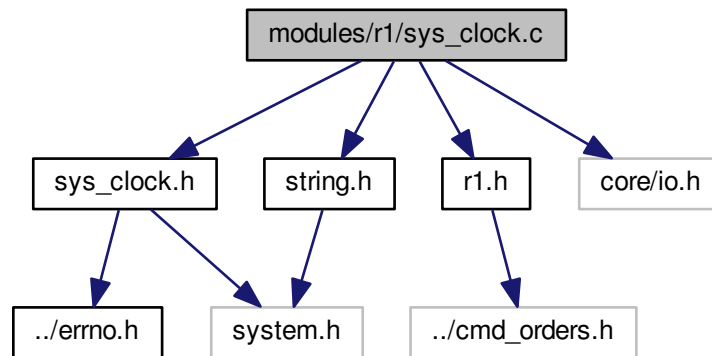
5.9.4.3 mpx

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	<i>The number of tokens found.</i>
argv	<i>The array of tokens.</i>

Returns

0

- int [set_time_main](#) (int argc, char **argv)

get_time_main.

Retrieves system's current time.

Parameters

argc	<i>The number of tokens found.</i>
argv	<i>The array of tokens.</i>

Returns

0

- int [get_time_main](#) (int argc, char **argv)

is_digit

determines if a character represents a digit.

Parameters

ch	<i>The character</i>
----	----------------------

Returns

1 if it is digit, otherwise returns 0.

set_time_str.

Sets the time for the system by string.

Parameters

timeStr	<i>The string type of current Time.</i>
---------	---

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	<i>The value of current time and date</i>
----------------	---

Returns

VOID

- void [get_time](#) (date_time *dateTimeValues)

set_time.

Sets the time for the system by date_time struct.

Parameters

dateTimeValues	<i>The struct that holds the time values.</i>
----------------	---

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	<i>The struct that holds the value of current date</i>
----------------	--

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	<i>The value of the year</i>
mon	<i>The value of the month</i>
day	<i>The value of the day of month</i>

Returns

VOID

set_date.

Sets the date of the system.

Parameters

dateTimeValues	<i>The struct that holds the value of date</i>
----------------	--

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	<i>The number of tokens.</i>
argv	<i>The array of tokens.</i>

Returns

0

- int [get_date_main](#) (int argc, char **argv)

set_date_str.

Sets the date for the system by string.

Parameters

str	<i>The string type of current date.</i>
-----	---

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	<i>The number of tokens.</i>
argv	<i>The array of tokens.</i>

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`

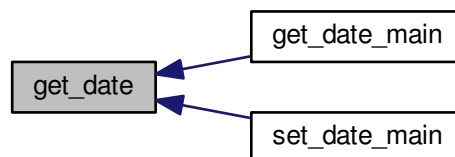
5.10.2.9 `#define RTC_INDEX_SECOND_ALARM 0x01`

5.10.2.10 `#define RTC_INDEX_YEAR 0x09`

5.10.3 Function Documentation

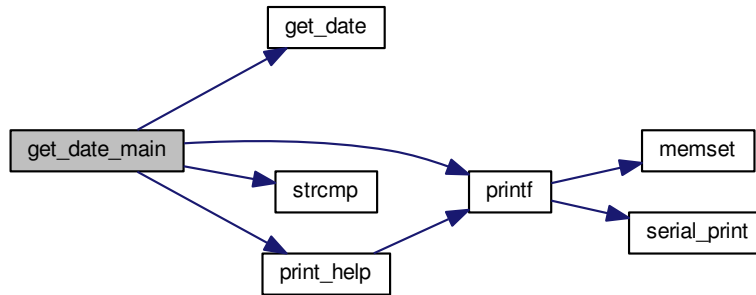
5.10.3.1 `void get_date (date_time * dateTimeValues)`

Here is the caller graph for this function:



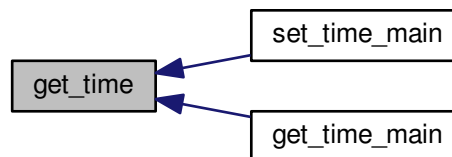
5.10.3.2 int get_date_main (int argc, char ** argv)

Here is the call graph for this function:



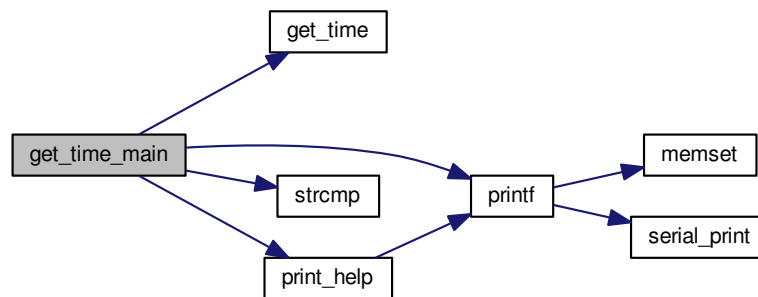
5.10.3.3 void get_time (date_time * dateTimeValues)

Here is the caller graph for this function:



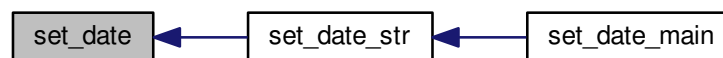
5.10.3.4 `int get_time_main (int argc, char ** argv)`

Here is the call graph for this function:



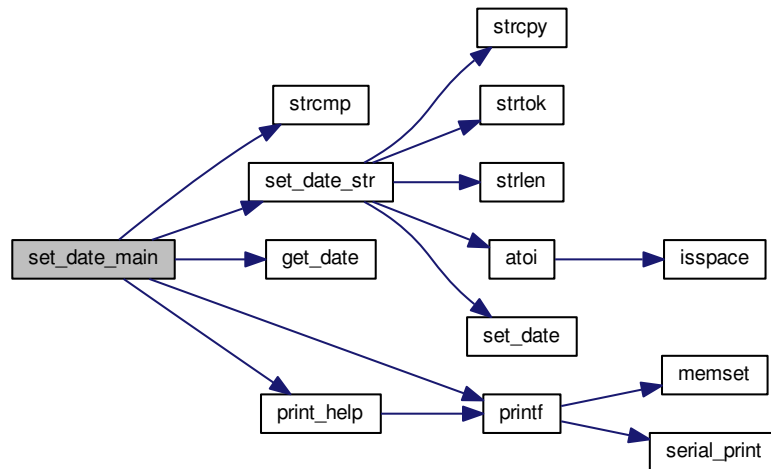
5.10.3.5 `error_t set_date (const date_time * dateTimeValues)`

Here is the caller graph for this function:



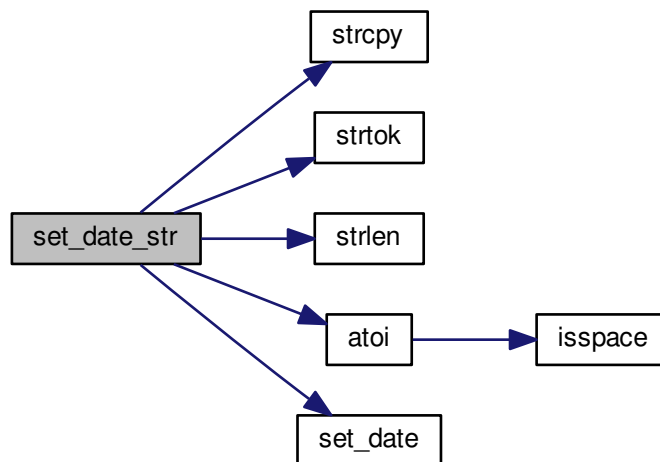
5.10.3.6 int set_date_main (int argc, char ** argv)

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:

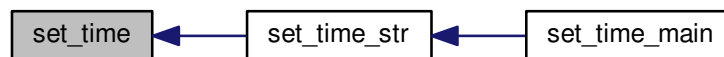


Here is the caller graph for this function:



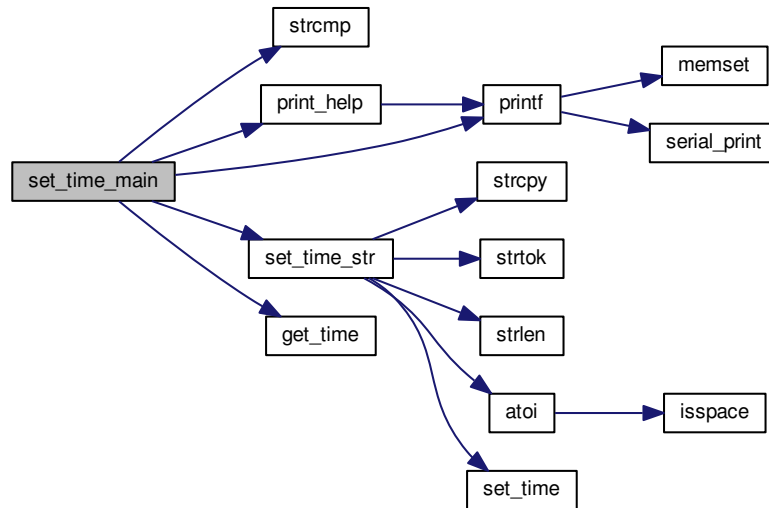
5.10.3.8 `error_t set_time (const date_time * dateTimeValues)`

Here is the caller graph for this function:



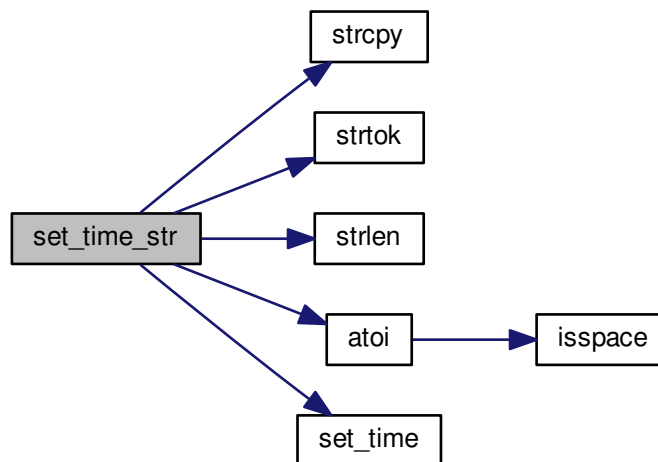
5.10.3.9 int set_time_main (int argc, char ** argv)

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:

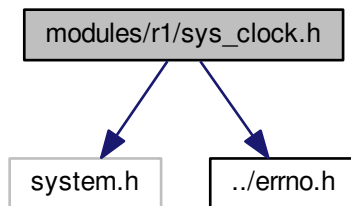


5.11 modules/r1/sys_clock.h File Reference

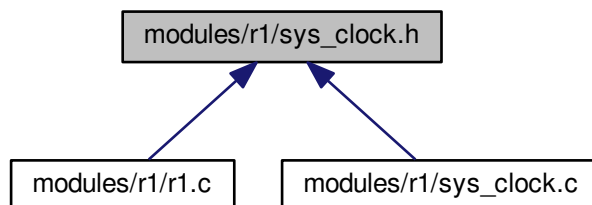
The main file that manipulates and controls the system's clock.

```
#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	<i>The number of tokens found.</i>
argv	<i>The array of tokens.</i>

Returns

0

- int [set_time_main](#) (int argc, char **argv)

get_time_main.

Retrieves system's current time.

Parameters

argc	<i>The number of tokens found.</i>
argv	<i>The array of tokens.</i>

Returns

0

- int [get_time_main](#) (int argc, char **argv)

set_time_str.

Sets the time for the system by string.

Parameters

timeStr	<i>The string type of current Time.</i>
---------	---

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	<i>The value of current time and date</i>
----------------	---

Returns

VOID

- void [get_time](#) (date_time *dateTimeValues)

set_time.

Sets the time for the system by date_time struct.

Parameters

dateTimeValues	<i>The struct that holds the time values.</i>
----------------	---

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	<i>The number of tokens.</i>
argv	<i>The array of tokens.</i>

Returns

0

- int [set_date_main](#) (int argc, char **argv)

get_date_main.

Retrieves system's current date.

Parameters

argc	<i>The number of tokens.</i>
argv	<i>The array of tokens.</i>

Returns

0

- int [get_date_main](#) (int argc, char **argv)

get_date.

Retrieves system's current date.

Parameters

dateTimeValues	<i>The struct that holds the value of current date</i>
----------------	--

Returns

VOID

- void [get_date](#) (date_time *dateTimeValues)

set_date_str.

Sets the date for the system by string.

Parameters

str	<i>The string type of current date.</i>
-----	---

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	<i>The struct that holds the value of date</i>
----------------	--

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

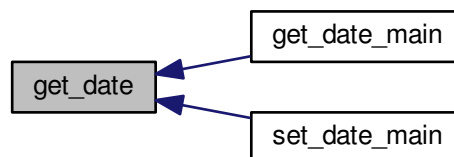
February 2nd, 2016

Version

R1

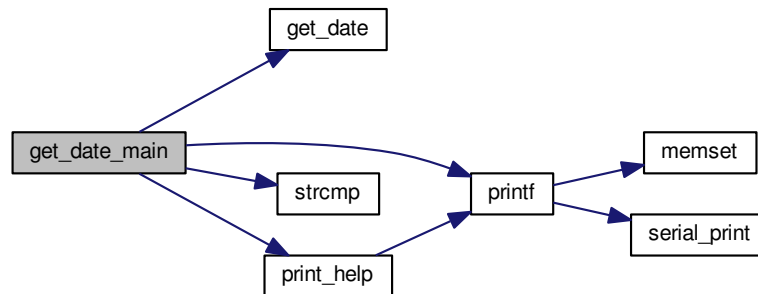
5.11.2 Function Documentation**5.11.2.1 void get_date (date_time * dateTimeValues)**

Here is the caller graph for this function:



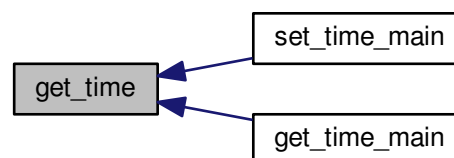
5.11.2.2 `int get_date_main (int argc, char ** argv)`

Here is the call graph for this function:



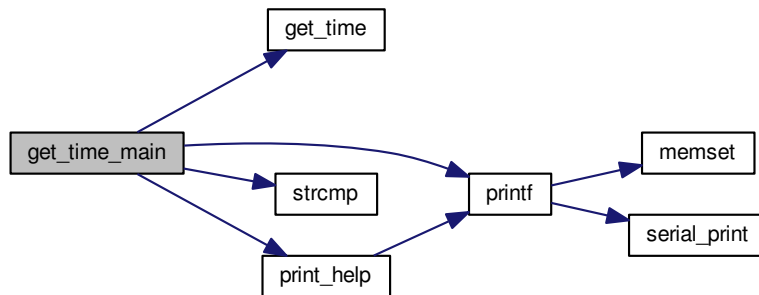
5.11.2.3 `void get_time (date_time * dateTimeValues)`

Here is the caller graph for this function:



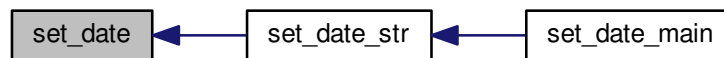
5.11.2.4 `int get_time_main (int argc, char ** argv)`

Here is the call graph for this function:



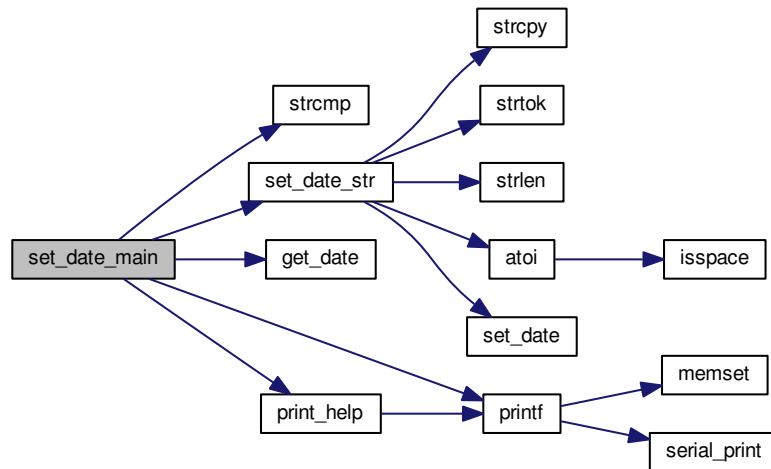
5.11.2.5 `error_t set_date (const date_time * dateTimeValues)`

Here is the caller graph for this function:



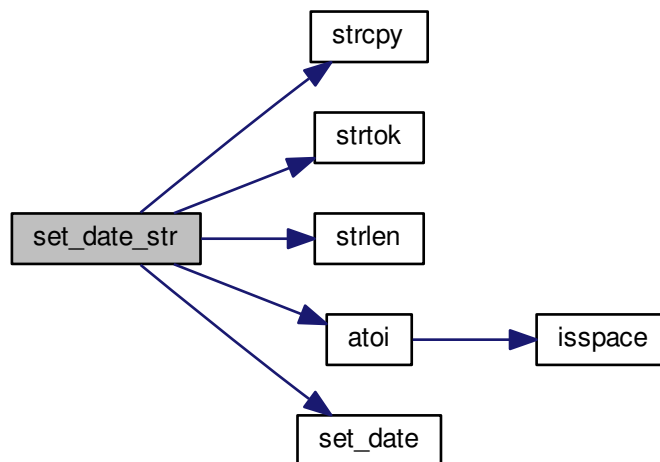
5.11.2.6 `int set_date_main (int argc, char ** argv)`

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:

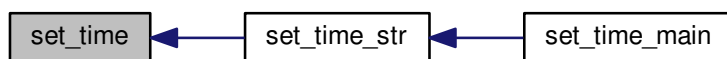


Here is the caller graph for this function:



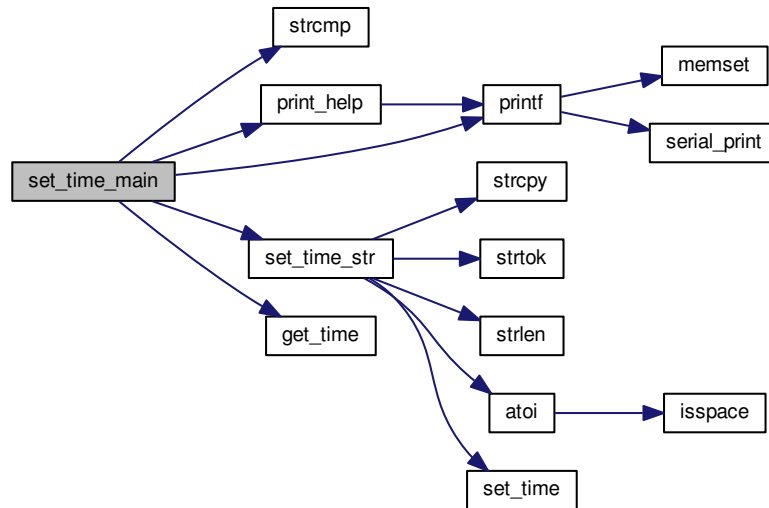
5.11.2.8 `error_t set_time (const date_time * dateTimeValues)`

Here is the caller graph for this function:



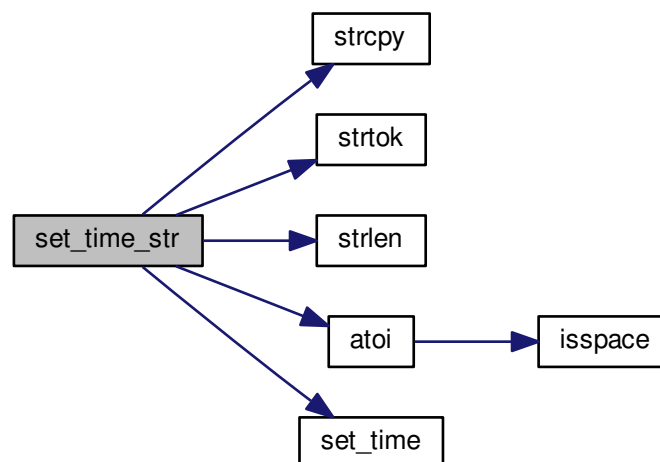
5.11.2.9 `int set_time_main (int argc, char ** argv)`

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:



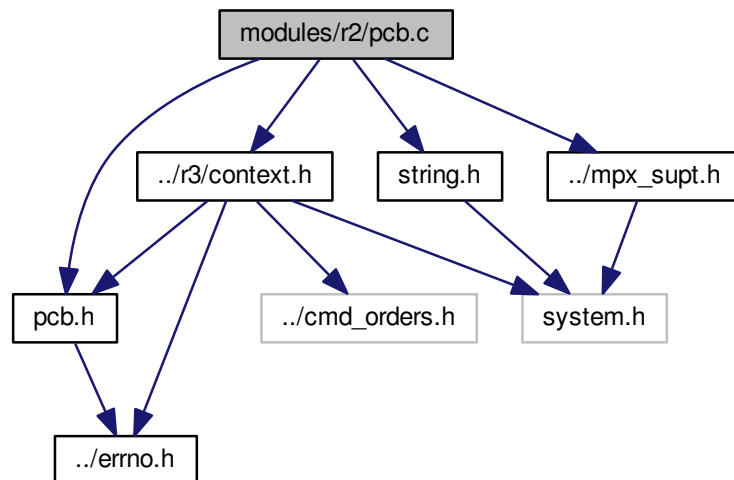
Here is the caller graph for this function:



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	<i>The pointer to the PCB</i>
-------------------------	-------------------------------

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	<i>The pointer to the PCB</i>
-------------------------	-------------------------------

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 occurred, 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.
pPriority	The assigned priority

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:

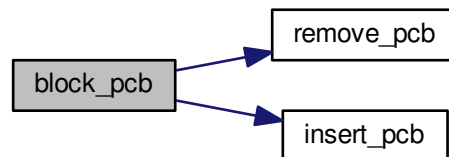


Here is the caller graph for this function:



5.12.3.3 `error_t block_pcb (struct pcb_struct * pcb_ptr)`

Here is the call graph for this function:

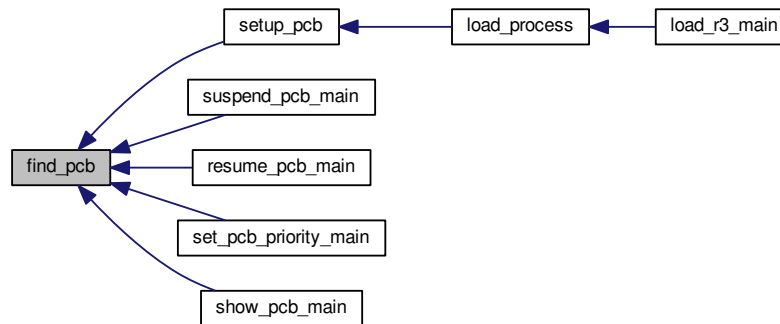


5.12.3.4 `struct pcb_struct* find_pcb (const char * pName)`

Here is the call graph for this function:



Here is the caller graph for this function:

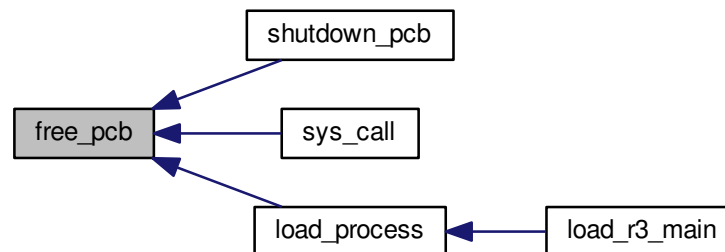


5.12.3.5 `error_t free_pcb (struct pcb_struct * pcb_ptr)`

Here is the call graph for this function:

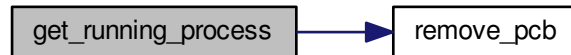


Here is the caller graph for this function:

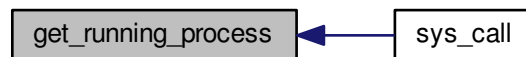


5.12.3.6 `struct pcb_struct* get_running_process ()`

Here is the call graph for this function:

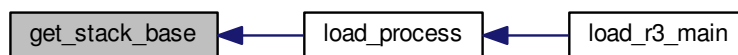


Here is the caller graph for this function:



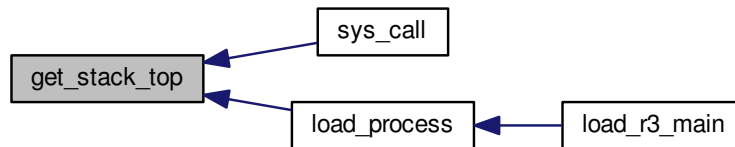
5.12.3.7 `unsigned char* get_stack_base (struct pcb_struct * pcb_ptr)`

Here is the caller graph for this function:



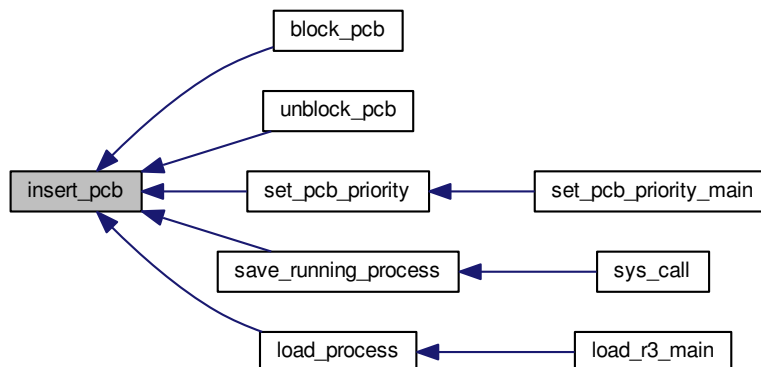
5.12.3.8 unsigned char* get_stack_top (struct pcb_struct * pcb_ptr)

Here is the caller graph for this function:



5.12.3.9 error_t insert_pcb (struct pcb_struct * pcb_ptr)

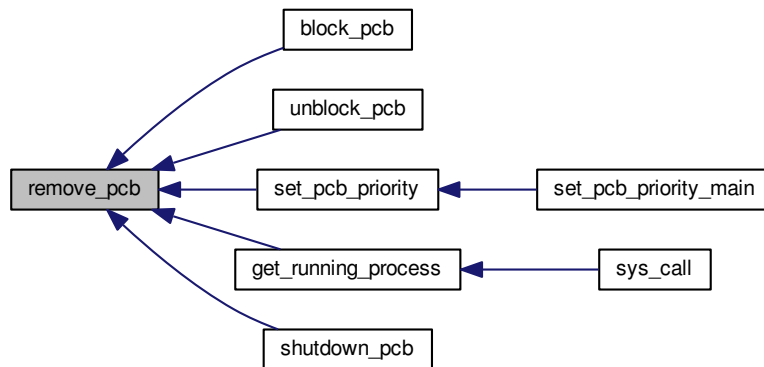
Here is the caller graph for this function:



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:



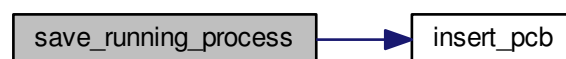
5.12.3.12 `error_t resume_pcb (struct pcb_struct * pcb_ptr)`

Here is the caller graph for this function:

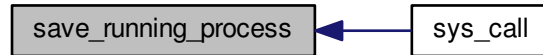


5.12.3.13 `error_t save_running_process (struct pcb_struct * pcb_ptr, struct context * new_stack_top)`

Here is the call graph for this function:

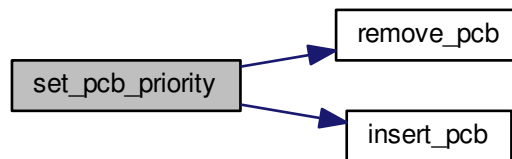


Here is the caller graph for this function:

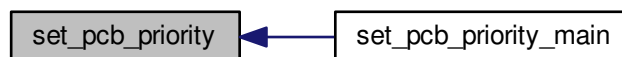


5.12.3.14 `error_t set_pcb_priority (struct pcb_struct * pcb_ptr, const unsigned char pPriority)`

Here is the call graph for this function:

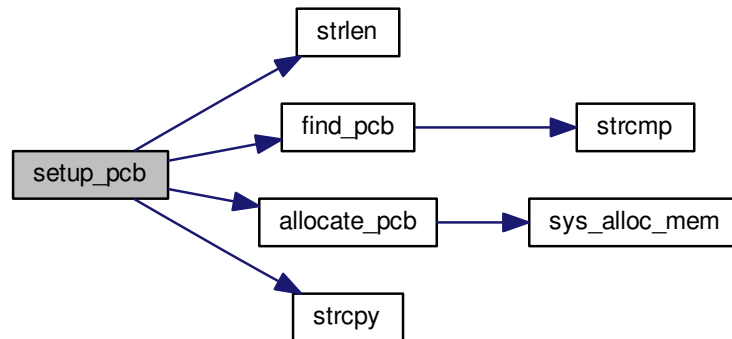


Here is the caller graph for this function:

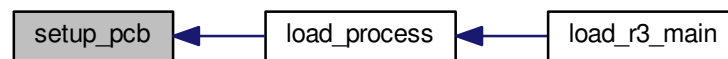


5.12.3.15 `struct pcb_struct* setup_pcb (const char * pName, const enum process_class pClass, const unsigned char pPriority)`

Here is the call graph for this function:

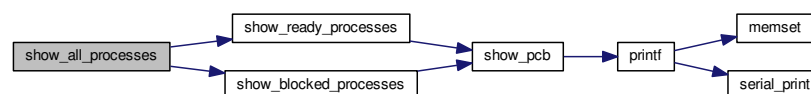


Here is the caller graph for this function:



5.12.3.16 `void show_all_processes ()`

Here is the call graph for this function:

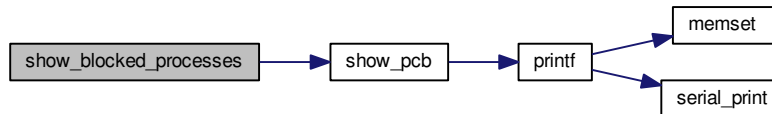


Here is the caller graph for this function:

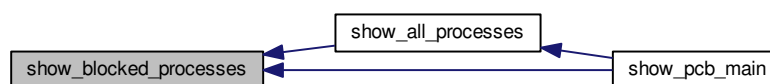


5.12.3.17 void show_blocked_processes ()

Here is the call graph for this function:

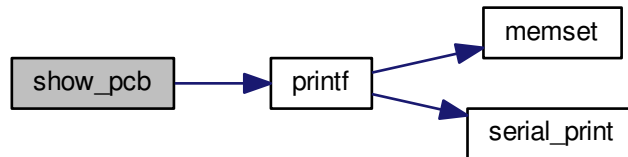


Here is the caller graph for this function:

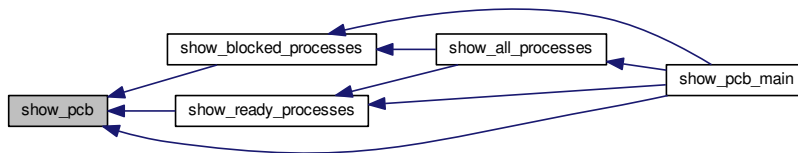


5.12.3.18 `error_t show_pcb (struct pcb_struct * pcb_ptr)`

Here is the call graph for this function:

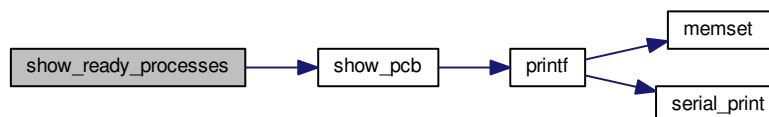


Here is the caller graph for this function:

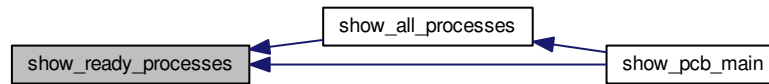


5.12.3.19 `void show_ready_processes ()`

Here is the call graph for this function:

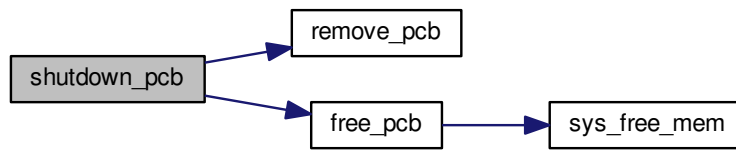


Here is the caller graph for this function:



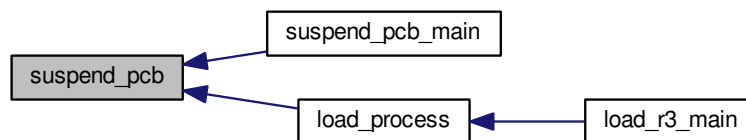
5.12.3.20 void shutdown_pcb ()

Here is the call graph for this function:



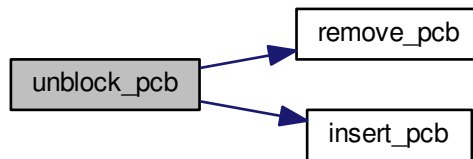
5.12.3.21 error_t suspend_pcb (struct pcb_struct * pcb_ptr)

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`

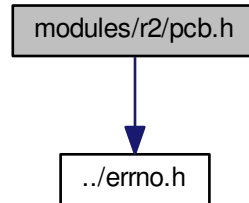
PCB process is suspended.

5.13 `modules/r2/pcb.h` File Reference

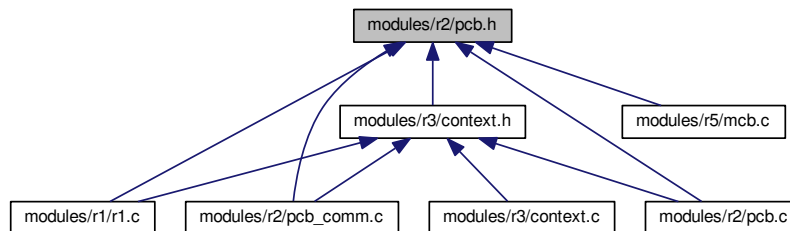
The Process Control Block.

```
#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	<i>The pointer to the PCB</i>
-------------------------	-------------------------------

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	<i>Process Name (length < 10).</i>
pClass	<i>Process class (system or application).</i>
pPriority	<i>Process priority (0 ~ 9).</i>

Returns

NULL if error ocured, 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	<i>The char pointer to the desired searched name</i>
-----------------------	--

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.
pPriority	The assigned priority

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:

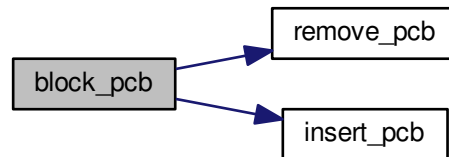


Here is the caller graph for this function:



5.13.4.3 `error_t block_pcb (struct pcb_struct * pcb_ptr)`

Here is the call graph for this function:

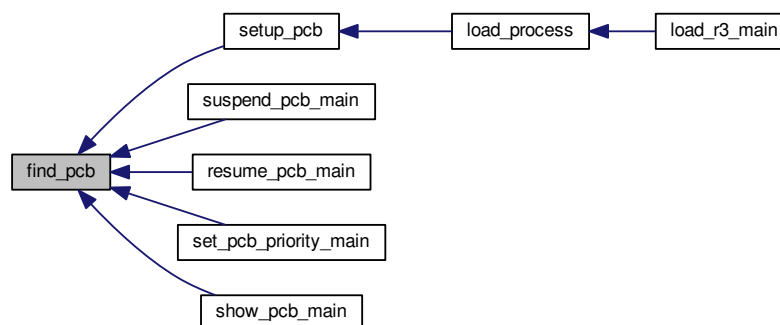


5.13.4.4 `struct pcb_struct* find_pcb (const char * pName)`

Here is the call graph for this function:



Here is the caller graph for this function:

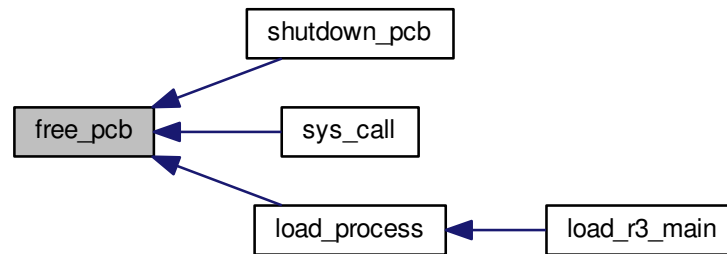


5.13.4.5 `error_t free_pcb (struct pcb_struct * pcb_ptr)`

Here is the call graph for this function:

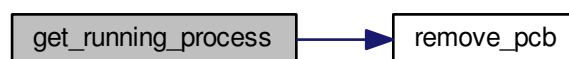


Here is the caller graph for this function:

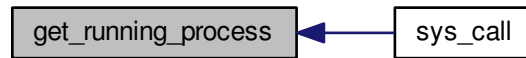


5.13.4.6 `struct pcb_struct* get_running_process ()`

Here is the call graph for this function:

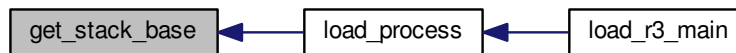


Here is the caller graph for this function:



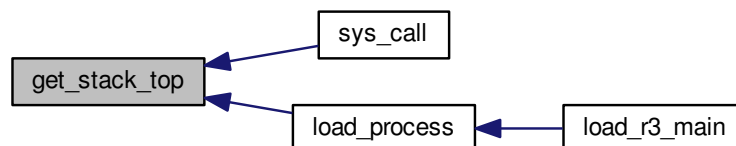
5.13.4.7 `unsigned char* get_stack_base (struct pcb_struct * pcb_ptr)`

Here is the caller graph for this function:



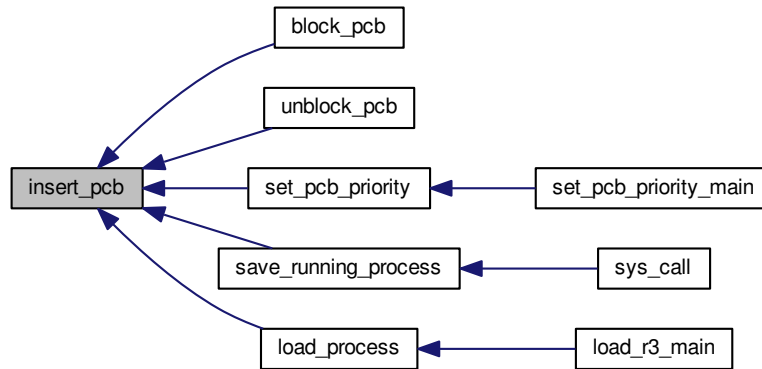
5.13.4.8 `unsigned char* get_stack_top (struct pcb_struct * pcb_ptr)`

Here is the caller graph for this function:



5.13.4.9 `error_t insert_pcb (struct pcb_struct * pcb_ptr)`

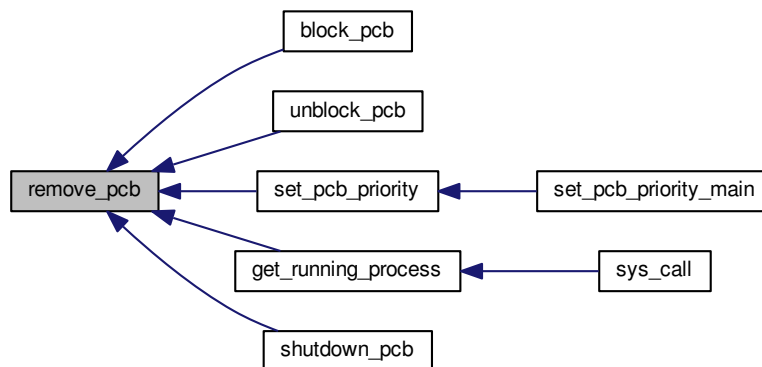
Here is the caller graph for this function:



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:



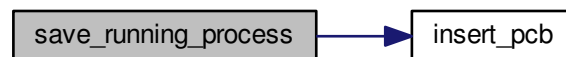
5.13.4.12 `error_t resume_pcb (struct pcb_struct * pcb_ptr)`

Here is the caller graph for this function:

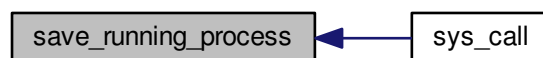


5.13.4.13 `error_t save_running_process (struct pcb_struct * pcb_ptr, struct context * new_stack_top)`

Here is the call graph for this function:

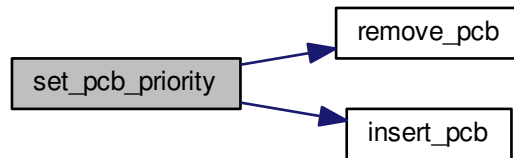


Here is the caller graph for this function:



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:

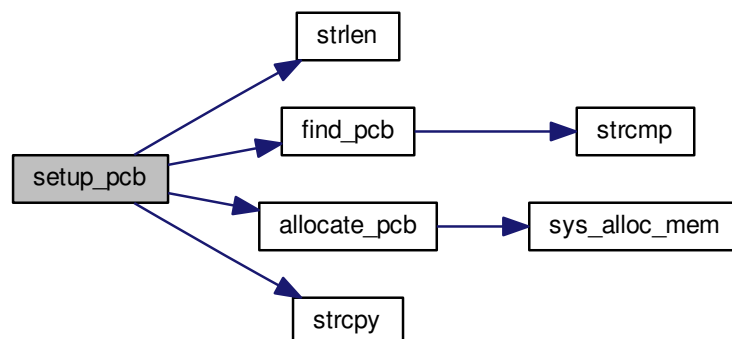


Here is the caller graph for this function:

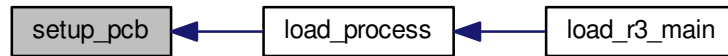


5.13.4.15 `struct pcb_struct* setup_pcb (const char * pName, const enum process_class pClass, const unsigned char pPriority)`

Here is the call graph for this function:

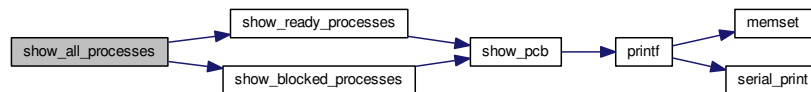


Here is the caller graph for this function:



5.13.4.16 void show_all_processes ()

Here is the call graph for this function:

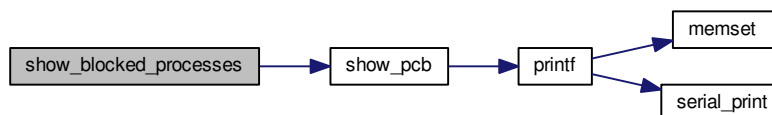


Here is the caller graph for this function:

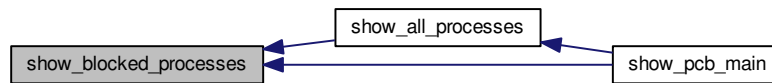


5.13.4.17 void show_blocked_processes ()

Here is the call graph for this function:

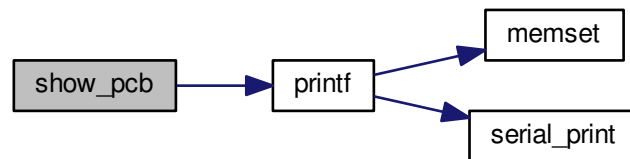


Here is the caller graph for this function:

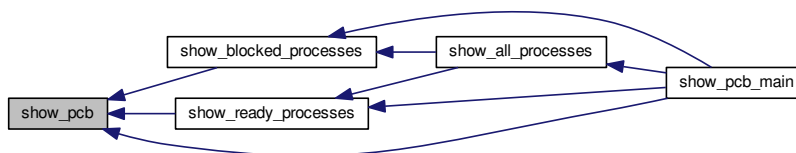


5.13.4.18 `error_t show_pcb (struct pcb_struct * pcb_ptr)`

Here is the call graph for this function:

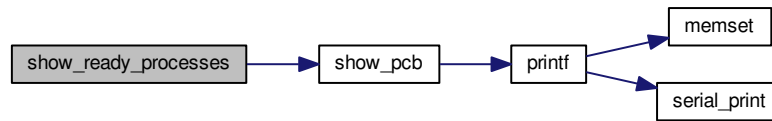


Here is the caller graph for this function:

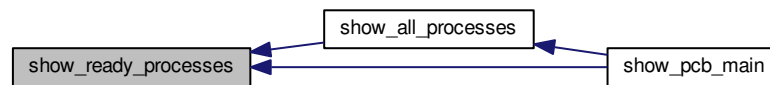


5.13.4.19 void show_ready_processes ()

Here is the call graph for this function:

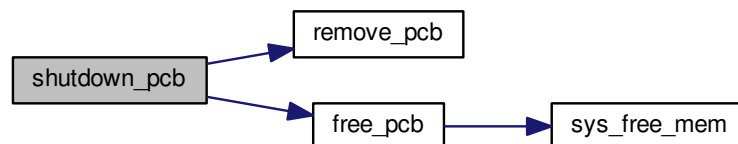


Here is the caller graph for this function:



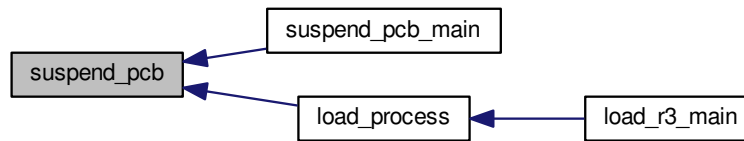
5.13.4.20 void shutdown_pcb ()

Here is the call graph for this function:



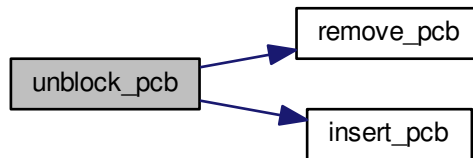
5.13.4.21 `error_t suspend_pcb (struct pcb_struct * pcb_ptr)`

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.

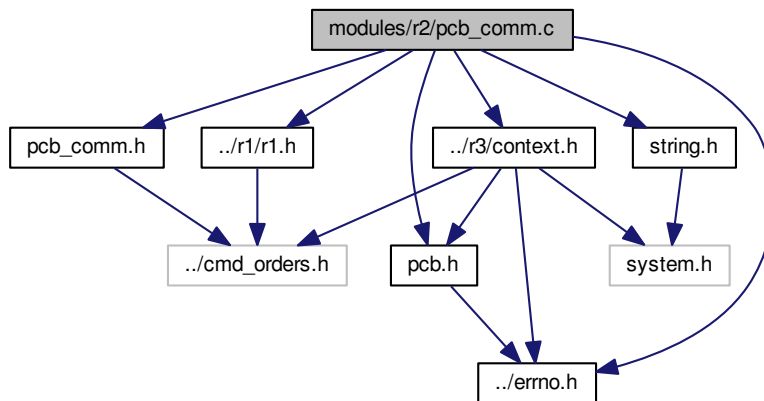
Process is a system process.

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

<code>argc</code>	The number of tokens found.
<code>argv</code>	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	<i>The number of tokens found.</i>
argv	<i>The array of tokens.</i>

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	<i>The number of tokens found.</i>
argv	<i>The array of tokens.</i>

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	<i>The number of tokens found.</i>
argv	<i>The array of tokens.</i>

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

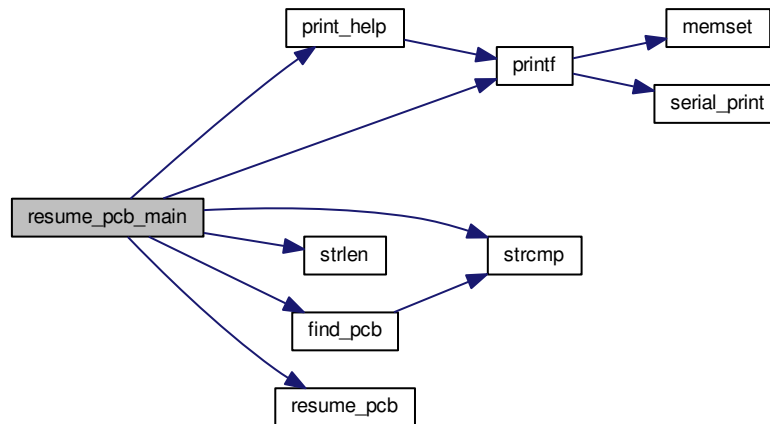
Version

R2

5.14.2 Function Documentation

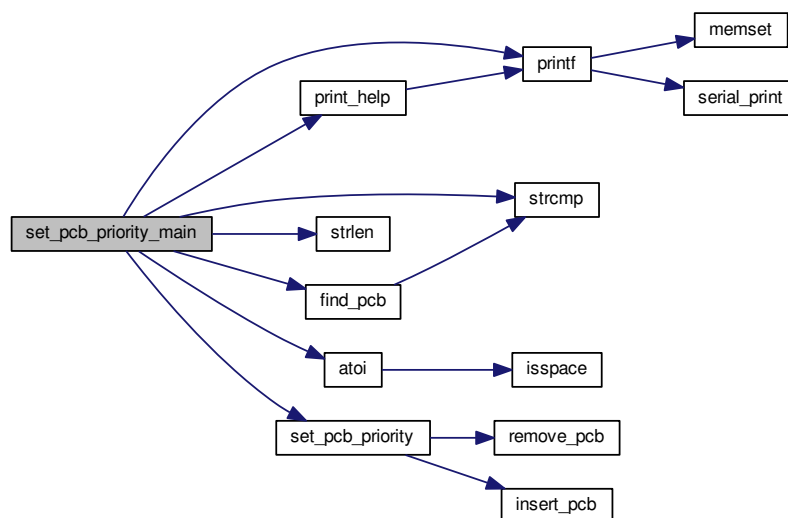
5.14.2.1 `int resume_pcb_main (int argc, char ** argv)`

Here is the call graph for this function:



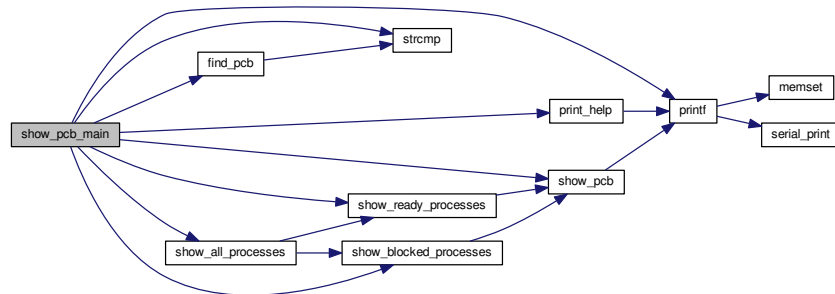
5.14.2.2 `int set_pcb_priority_main (int argc, char ** argv)`

Here is the call graph for this function:



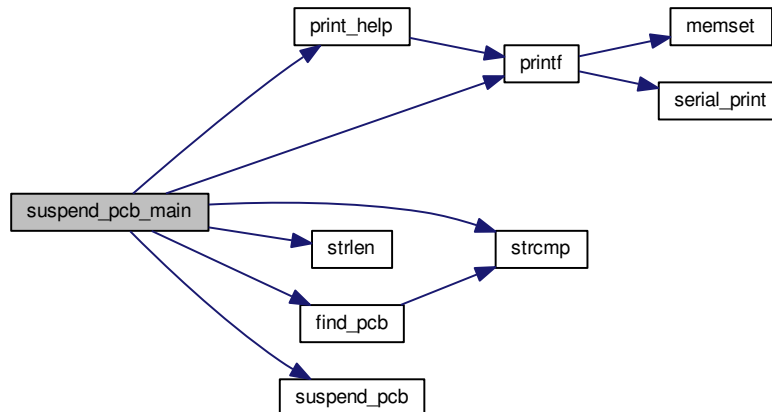
5.14.2.3 int show_pcb_main (int argc, char ** argv)

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:

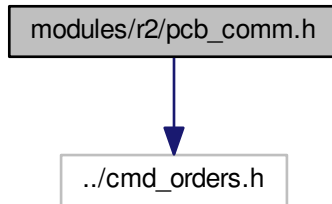


5.15 modules/r2/pcb_comm.h File Reference

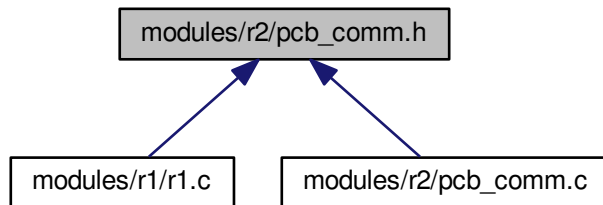
The main functions that manipulate the PCB.

```
#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	<i>The number of tokens found.</i>
argv	<i>The array of tokens.</i>

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	<i>The number of tokens found.</i>
argv	<i>The array of tokens.</i>

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	<i>The number of tokens found.</i>
argv	<i>The array of tokens.</i>

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	<i>The number of tokens found.</i>
argv	<i>The array of tokens.</i>

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

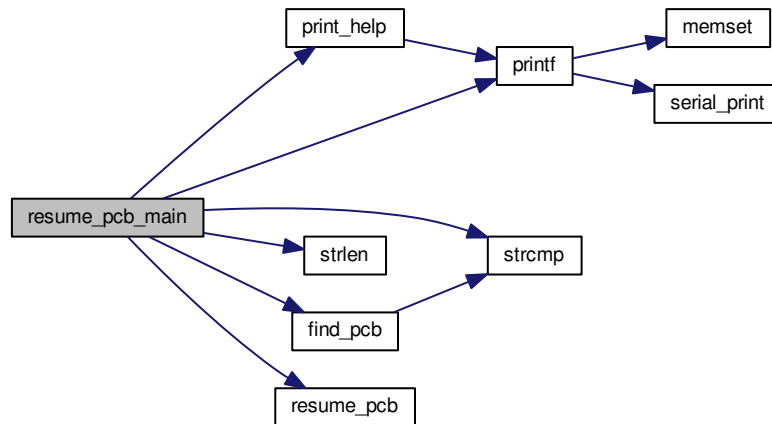
Version

R2

5.15.2 Function Documentation

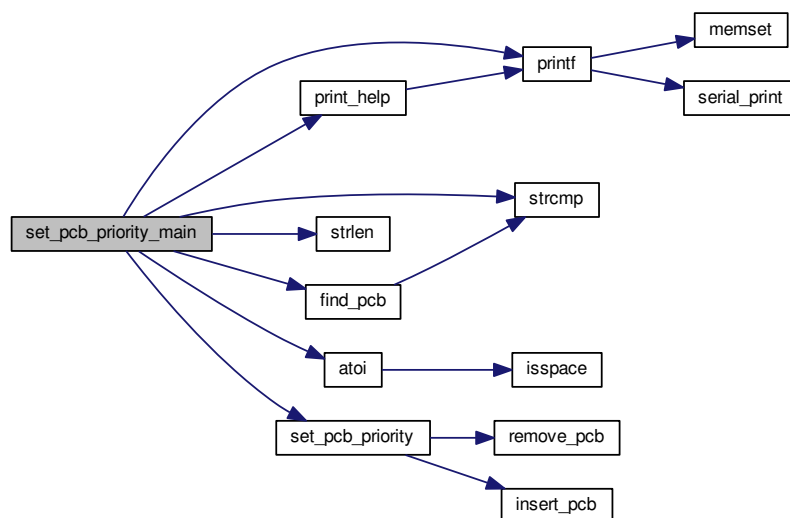
5.15.2.1 `int resume_pcb_main (int argc, char ** argv)`

Here is the call graph for this function:



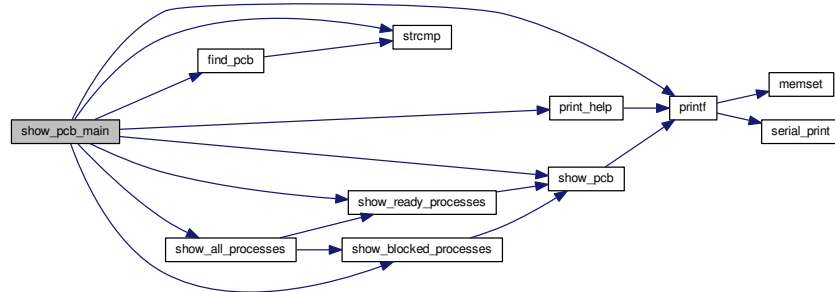
5.15.2.2 `int set_pcb_priority_main (int argc, char ** argv)`

Here is the call graph for this function:



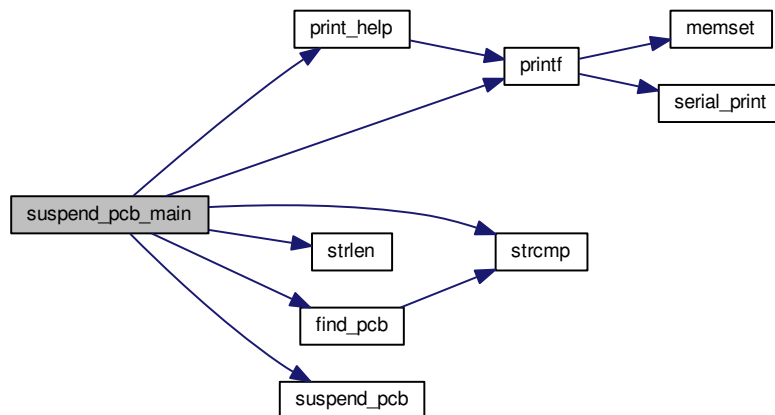
5.15.2.3 int show_pcb_main (int argc, char ** argv)

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

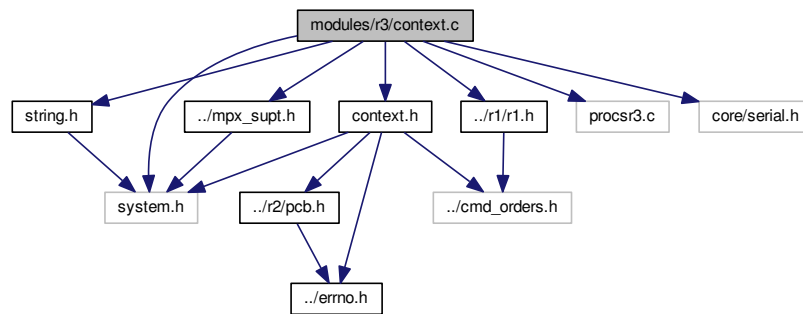
Context Switching.

```

#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	<i>character selected.</i>
----	----------------------------

Returns

a digit between 0 and 9.

sys_call

system call interrupt

Parameters

context*	<i>registers current registers</i>
----------	------------------------------------

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	<i>Process Name</i>
pClass	<i>Process Class</i>

pPriority	<i>Process Priority</i>
*function()	<i>A function pointer</i>

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	<i>The number of tokens found.</i>
argv	<i>The array of tokens.</i>

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

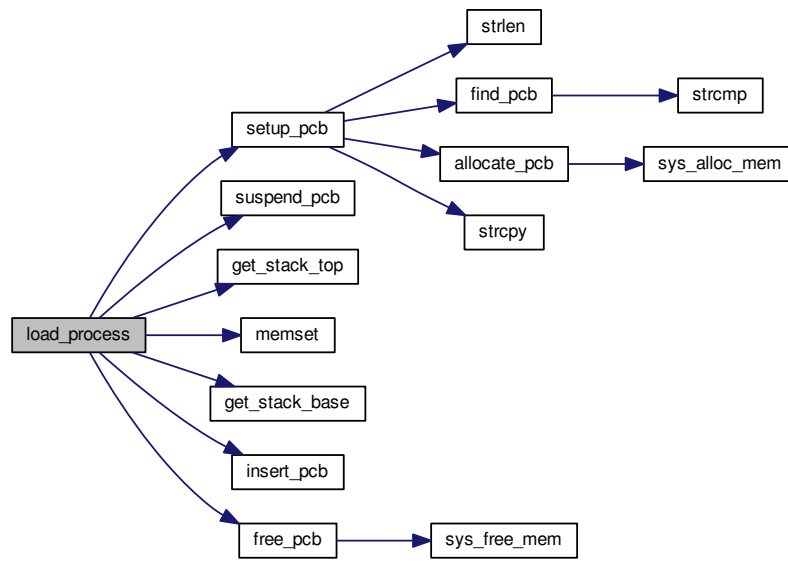
Version

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:

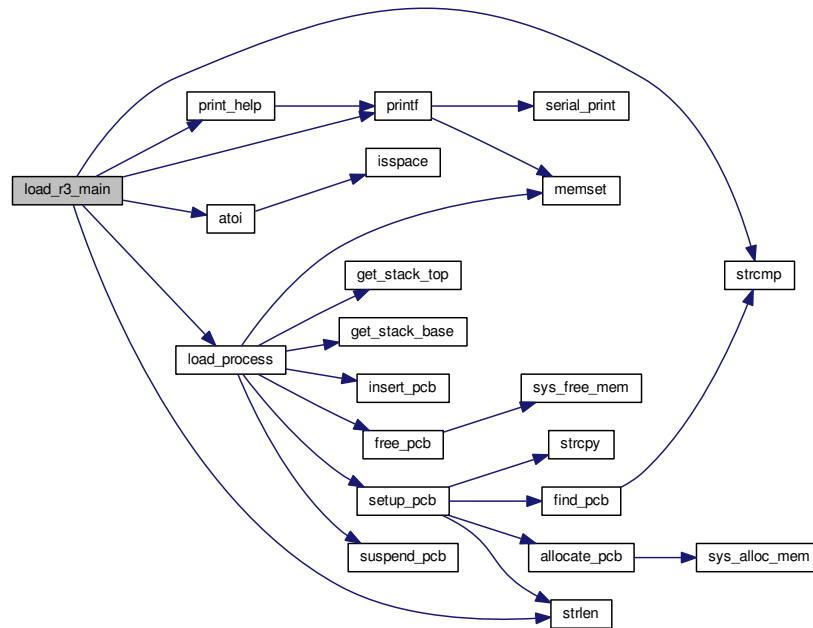


Here is the caller graph for this function:



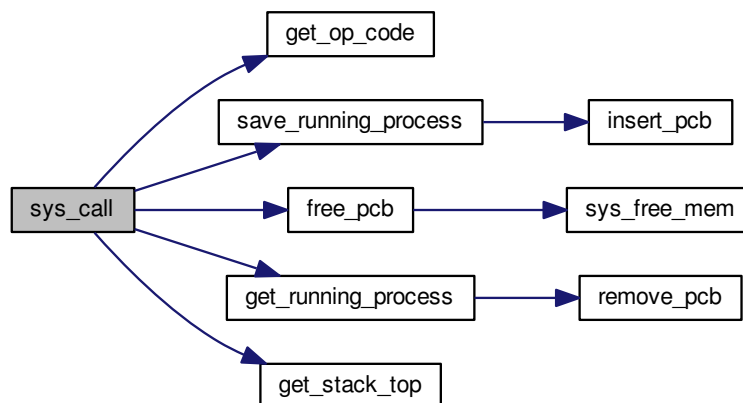
5.16.2.2 `int load_r3_main (int argc, char ** argv)`

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

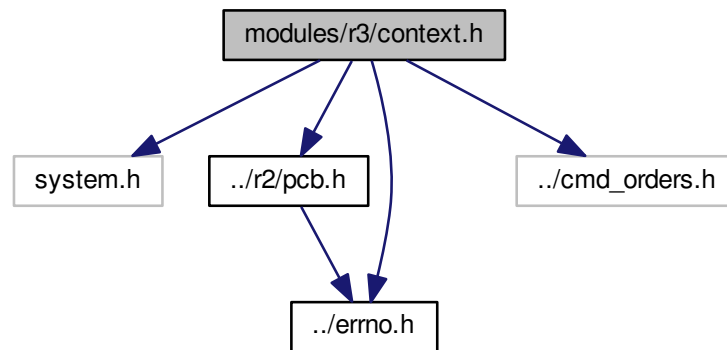
5.16.3.1 struct pcb_struct* cop = NULL

5.16.3.2 struct context* old_context = NULL

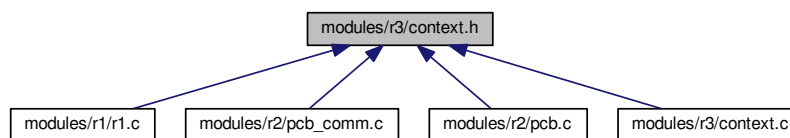
5.17 modules/r3/context.h File Reference

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*	<i>registers current registers</i>
----------	------------------------------------

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	<i>Process Name</i>
pClass	<i>Process Class</i>
pPriority	<i>Process Priority</i>
*function()	<i>A function pointer</i>

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	<i>The number of tokens found.</i>
argv	<i>The array of tokens.</i>

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

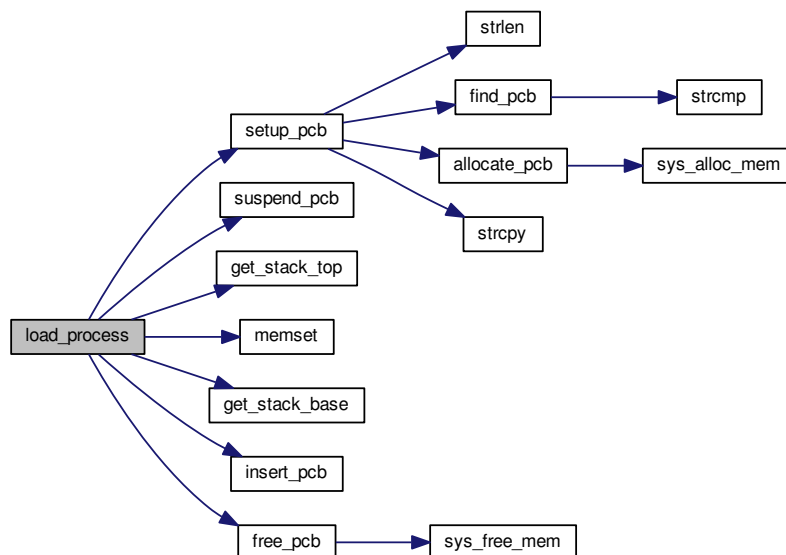
Version

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:

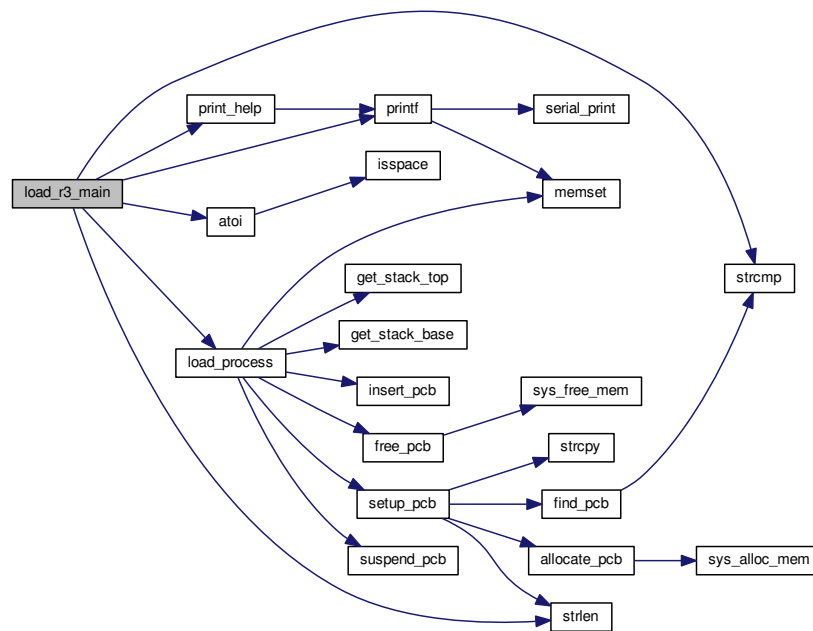


Here is the caller graph for this function:



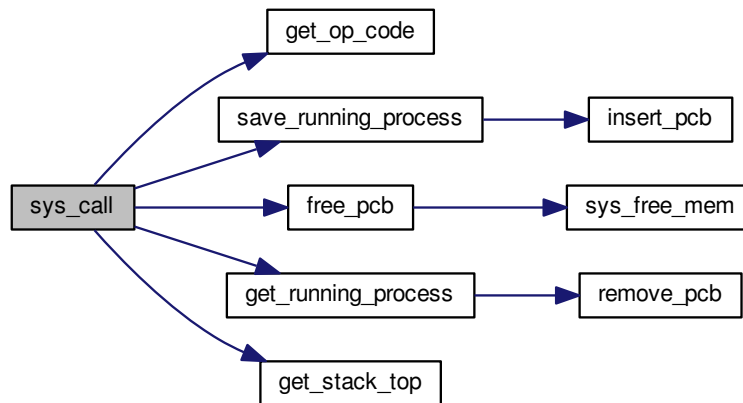
5.17.2.2 `int load_r3_main (int argc, char ** argv)`

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

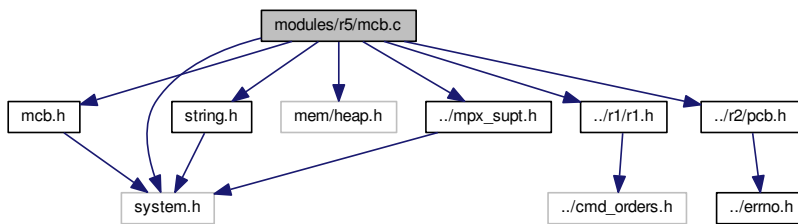
Memory Control Block.

```

#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

<code>mem_size</code>	<i>The size of memory</i>
-----------------------	---------------------------

Returns

the total size of memory

`init_mem_block`

Initiate memory block

Parameters

start_pos	<i>Starting position</i>
size	<i>Size of MCB</i>
type	<i>Type of MCB</i>

Returns

VOID

prev_adjacent_mcb

Previous adjacent MCB in Memory

Parameters

mcb_ptr	<i>MCB Pointer</i>
---------	--------------------

Returns

the pointer to the previous MCB

next_adjacent_mcb

Next adjacent MCB in Memory

Parameters

mcb_ptr	<i>MCB Pointer</i>
---------	--------------------

Returns

the pointer to the next MCB

find_mcb_by_address

Finds the MCB by address

Parameters

mem_ptr	<i>Memory address pointer</i>
---------	-------------------------------

Returns

Pointer to the MCB

insert_mcb_to_queue

Inserts MCB to the queue

Parameters

mcb_ptr	<i>MCB Pointer</i>
type	<i>The type of MCB</i>

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	<i>The number of tokens found.</i>
argv	<i>The array of tokens.</i>

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	<i>The number of tokens found.</i>
argv	<i>The array of tokens.</i>

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	<i>The number of tokens found.</i>
argv	<i>The array of tokens.</i>

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))

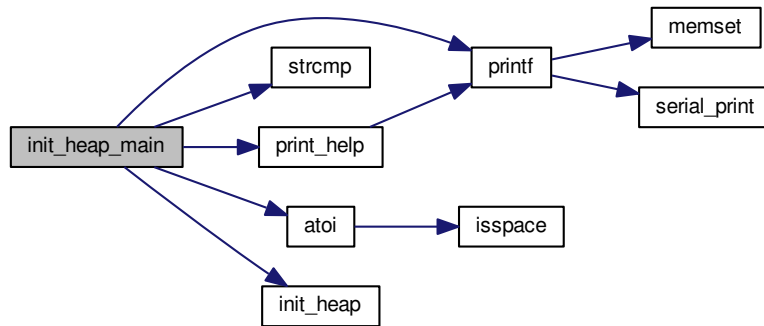
5.18.3.2 void init_heap (u32int size)

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:



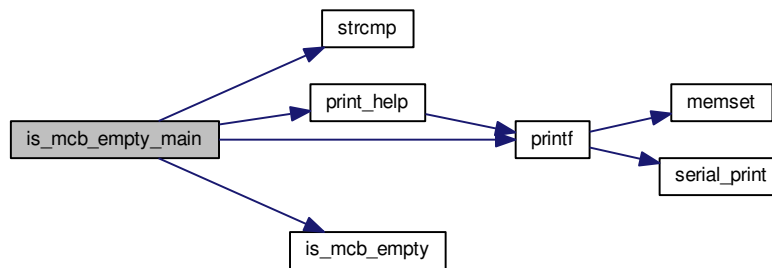
5.18.3.4 int is_mcb_empty ()

Here is the caller graph for this function:



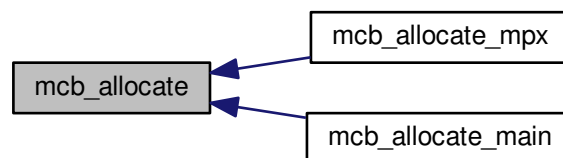
5.18.3.5 `int is_mcb_empty_main (int argc, char ** argv)`

Here is the call graph for this function:



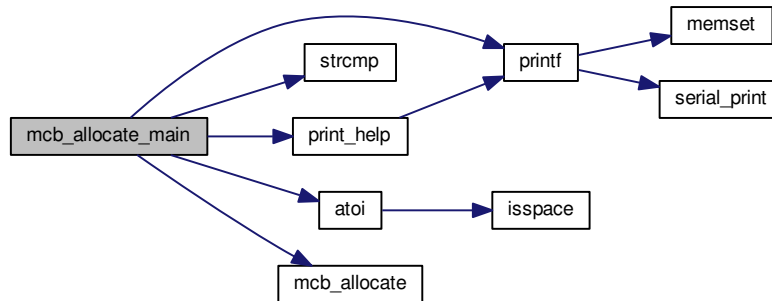
5.18.3.6 `void* mcb_allocate (u32int mem_size)`

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:



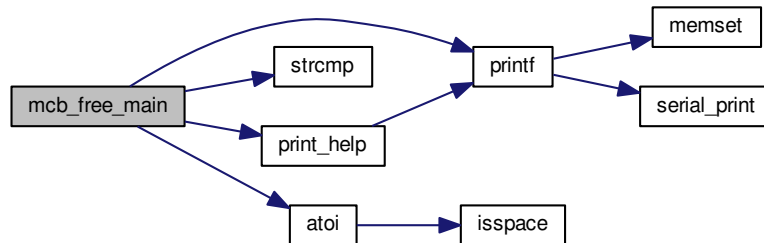
5.18.3.8 u32int mcb_allocate_mpx (u32int size)

Here is the call graph for this function:



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

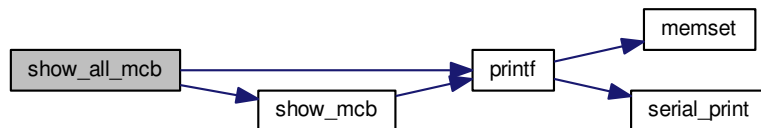
Here is the call graph for this function:



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:

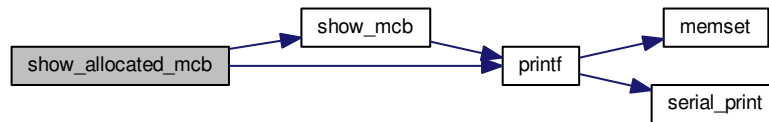


Here is the caller graph for this function:



5.18.3.12 void show_allocated_mcb ()

Here is the call graph for this function:

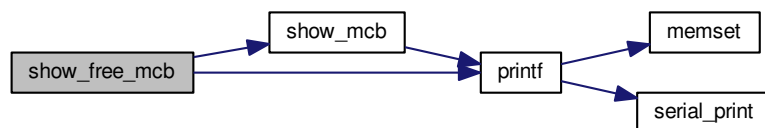


Here is the caller graph for this function:



5.18.3.13 void show_free_mcb ()

Here is the call graph for this function:

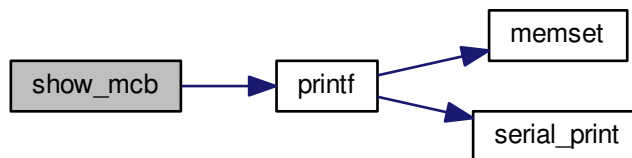


Here is the caller graph for this function:

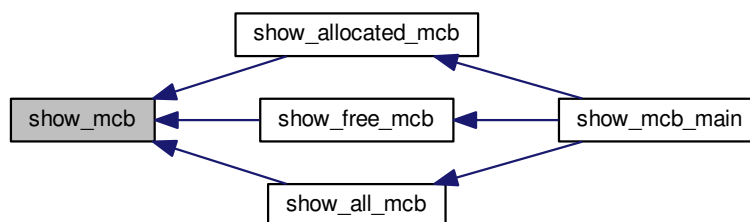


5.18.3.14 void show_mcb (struct mcb * mcb_ptr)

Here is the call graph for this function:

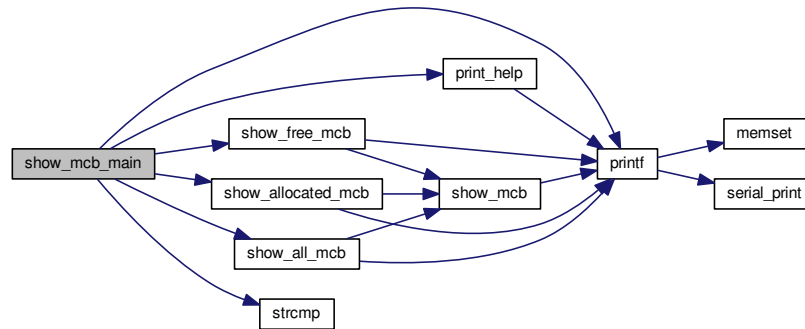


Here is the caller graph for this function:



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

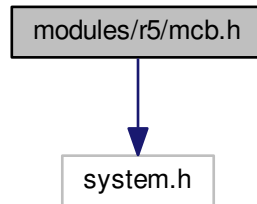
Global variable labeling start of memory.

5.19 modules/r5/mcb.h File Reference

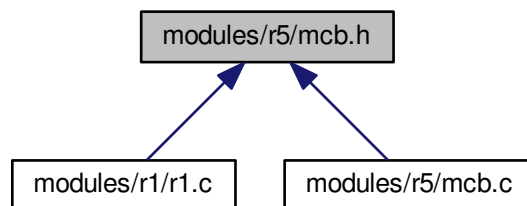
Memory Control Block.

```
#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:



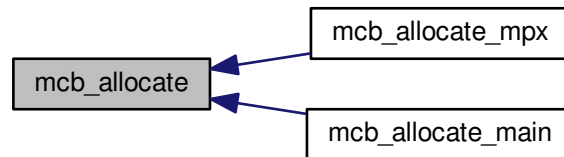
5.19.3.2 `int is_mcb_empty ()`

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:

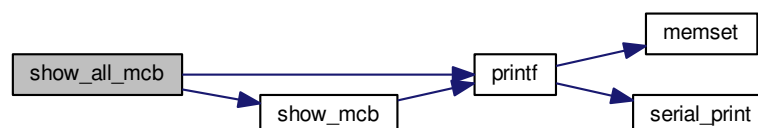


5.19.3.5 void* mcb_allocate_mpx2 (u32int size, const char * name)

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:

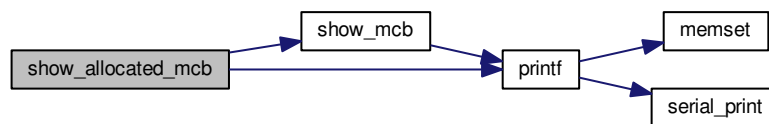


Here is the caller graph for this function:



5.19.3.8 void show_allocated_mcb ()

Here is the call graph for this function:

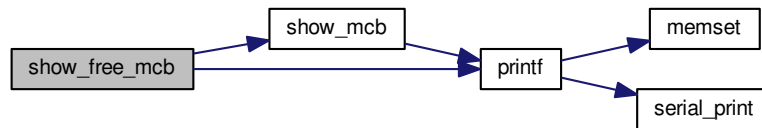


Here is the caller graph for this function:



5.19.3.9 void show_free_mcb ()

Here is the call graph for this function:

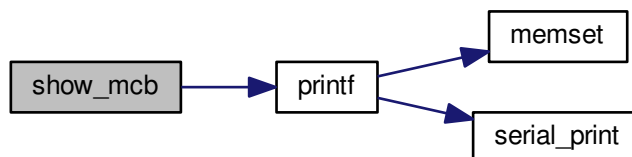


Here is the caller graph for this function:

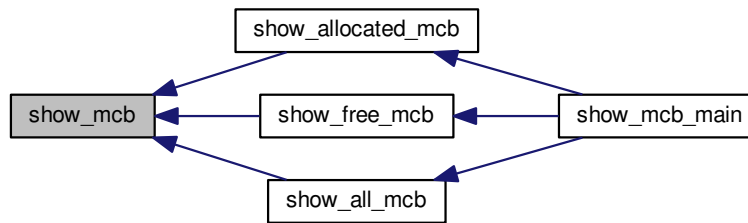


5.19.3.10 void show_mcb (struct mcb * mcb_ptr)

Here is the call graph for this function:

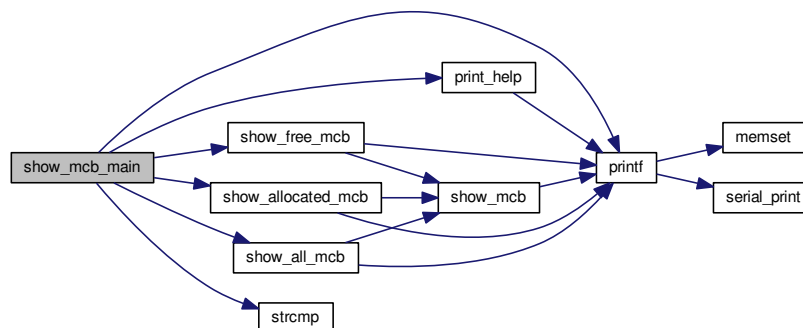


Here is the caller graph for this function:



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

- `__attribute__`
 - mcb.c, [142](#), [149](#)
 - mpx_supt.h, [50](#)
 - pcb.c, [89](#), [100](#)
 - pcb.h, [107](#)
 - r1.c, [54](#)
 - r1.h, [59](#)
- allocate_pcb
 - pcb.c, [89](#)
 - pcb.h, [107](#)
- allocated
 - mcb.c, [149](#)
- allocated_mem_list
 - mcb.c, [149](#)
- atoi
 - string.c, [34](#)
 - string.h, [23](#)
- begin_address
 - cmcb, [7](#)
- block_pcb
 - pcb.c, [90](#)
 - pcb.h, [107](#)
- blocked
 - pcb.c, [100](#)
- COM1
 - serial.h, [18](#)
- COM2
 - serial.h, [18](#)
- COM3
 - serial.h, [18](#)
- COM4
 - serial.h, [18](#)
- COMMHAND_PCB_NAME
 - pcb.h, [107](#)
- COMPLETION
 - r1.c, [54](#)
- class
 - pcb_struct, [15](#)
- cmcb, [7](#)
 - begin_address, [7](#)
 - size, [7](#)
 - type, [7](#)
- comm_type
 - r1.h, [59](#)
- command_line_parser
 - r1.c, [54](#)
 - r1.h, [59](#)
- CommandPaserStat
 - r1.c, [54](#)
- commhand
 - r1.c, [54](#)
 - r1.h, [59](#)
- complete_mcb
 - mcb, [12](#)
- context, [8](#)
 - cs, [9](#)
 - ds, [9](#)
 - eax, [9](#)
 - ebp, [9](#)
 - ebx, [9](#)
 - ecx, [9](#)
 - edi, [9](#)
 - edx, [9](#)
 - eflags, [9](#)
 - eip, [9](#)
 - es, [9](#)
 - esi, [9](#)
 - esp, [10](#)
 - fs, [10](#)
 - gs, [10](#)
- context.c
 - cop, [131](#)
 - load_process, [128](#)
 - load_r3_main, [129](#)
 - old_context, [131](#)
 - sys_call, [130](#)
- context.h
 - cop, [135](#)
 - load_process, [133](#)
 - load_r3_main, [134](#)
 - old_context, [135](#)
 - sys_call, [134](#)
- cop
 - context.c, [131](#)
 - context.h, [135](#)
- count
 - pcb_queue, [14](#)
- cs

- context, 9
- current_module
 - mpx_supt.c, 45
- device_id
 - param, 13
- documentation/mainpage.dox, 17
- DoubleQuoteWriting
 - r1.c, 56
- ds
 - context, 9
- E_EMPTPCB
 - errno.h, 42
- E_FREEMEM
 - errno.h, 42
- E_INVPARA
 - errno.h, 42
- E_INVSTRF
 - errno.h, 42
- E_INVUSRI
 - errno.h, 42
- E_NOERROR
 - errno.h, 42
- E_NULL_PTR
 - errno.h, 42
- E_PCB_SYS
 - errno.h, 42
- E_PROGERR
 - errno.h, 42
- EXIT
 - mpx_supt.h, 48
- eax
 - context, 9
- ebp
 - context, 9
- ebx
 - context, 9
- ecx
 - context, 9
- edi
 - context, 9
- edx
 - context, 9
- eflags
 - context, 9
- eip
 - context, 9
- end_of_memory
 - mcb.c, 149
- errno.h
 - E_EMPTPCB, 42
 - E_FREEMEM, 42
 - E_INVPARA, 42
 - E_INVSTRF, 42
 - E_INVUSRI, 42
 - E_NOERROR, 42
 - E_NULL_PTR, 42
 - E_PCB_SYS, 42
 - E_PROGERR, 42
- error_t
 - errno.h, 42
- error_t
 - context, 9
- esi
 - context, 9
- esp
 - context, 10
- false
 - pcb.c, 100
- find_pcb
 - pcb.c, 90
 - pcb.h, 108
- free
 - mcb.c, 149
- free_mem_list
 - mcb.c, 149
- free_pcb
 - pcb.c, 91
 - pcb.h, 108
- fs
 - context, 10
- function
 - function_name, 10
- function_name, 10
 - function, 10
 - help, 10
 - nameStr, 11
 - usage, 11
- get_date
 - sys_clock.c, 68
 - sys_clock.h, 77
- get_date_main
 - sys_clock.c, 68
 - sys_clock.h, 78
- get_input_line
 - serial.h, 19
- get_op_code
 - mpx_supt.c, 43
 - mpx_supt.h, 49
- get_running_process
 - pcb.c, 91
 - pcb.h, 109
- get_stack_base
 - pcb.c, 92
 - pcb.h, 110
- get_stack_top

- pcb.c, 92
- pcb.h, 110
- get_time
 - sys_clock.c, 69
 - sys_clock.h, 78
- get_time_main
 - sys_clock.c, 69
 - sys_clock.h, 78
- gs
 - context, 10
- head
 - pcb_queue, 14
- help
 - function_name, 10
 - r1.h, 61
- help_usages
 - r1.c, 54
 - r1.h, 59
- IDLE
 - mpx_supt.h, 48
- IDLE_PCB_NAME
 - pcb.h, 107
- idle
 - mpx_supt.c, 44
 - mpx_supt.h, 49
- include/core/serial.h, 17
- include/string.h, 20
- init_heap
 - mcb.c, 142
 - mcb.h, 153
- init_heap_main
 - mcb.c, 142
- init_serial
 - serial.h, 19
- insert_pcb
 - pcb.c, 93
 - pcb.h, 110
- is_mcb_empty
 - mcb.c, 143
 - mcb.h, 153
- is_mcb_empty_main
 - mcb.c, 143
- is_suspended
 - pcb_struct, 15
- isspace
 - string.c, 35
 - string.h, 24
- lib/string.c, 30
- limited_mcb
 - mcb, 12
- lmcb, 11
 - size, 11
- type, 11
- load_process
 - context.c, 128
 - context.h, 133
- load_r3_main
 - context.c, 129
 - context.h, 134
- MAX_ARGC
 - r1.c, 54
- MAX_HEAP_SIZE
 - mcb.h, 153
- MAX_HISTORY
 - r1.c, 54
- MOD_VERSION
 - r1.c, 54
- MODULE_R1
 - mpx_supt.h, 48
- MODULE_R2
 - mpx_supt.h, 48
- MODULE_R3
 - mpx_supt.h, 48
- MODULE_R4
 - mpx_supt.h, 48
- MODULE_R5
 - mpx_supt.h, 48
- mcb, 12
 - complete_mcb, 12
 - limited_mcb, 12
 - next, 12
 - prev, 12
 - r1.h, 61
- mcb.c
 - __attribute__, 142, 149
 - allocated, 149
 - allocated_mem_list, 149
 - end_of_memory, 149
 - free, 149
 - free_mem_list, 149
 - init_heap, 142
 - init_heap_main, 142
 - is_mcb_empty, 143
 - is_mcb_empty_main, 143
 - mcb_allocate, 144
 - mcb_allocate_main, 144
 - mcb_allocate_mpx, 145
 - mcb_free_main, 145
 - mcb_free_mpx, 146
 - mcb_type, 142
 - show_all_mcb, 146
 - show_allocated_mcb, 146
 - show_free_mcb, 147
 - show_mcb, 148
 - show_mcb_main, 148

- shutdown_mcb, [149](#)
- start_of_memory, [149](#)
- mcb.h
 - init_heap, [153](#)
 - is_mcb_empty, [153](#)
 - MAX_HEAP_SIZE, [153](#)
 - mcb_allocate, [153](#)
 - mcb_allocate_mpx, [154](#)
 - mcb_allocate_mpx2, [154](#)
 - mcb_free_mpx, [154](#)
 - show_all_mcb, [154](#)
 - show_allocated_mcb, [155](#)
 - show_free_mcb, [155](#)
 - show_mcb, [156](#)
 - show_mcb_main, [157](#)
 - shutdown_mcb, [157](#)
 - start_of_memory, [157](#)
- mcb_allocate
 - mcb.c, [144](#)
 - mcb.h, [153](#)
- mcb_allocate_main
 - mcb.c, [144](#)
- mcb_allocate_mpx
 - mcb.c, [145](#)
 - mcb.h, [154](#)
- mcb_allocate_mpx2
 - mcb.h, [154](#)
- mcb_free_main
 - mcb.c, [145](#)
- mcb_free_mpx
 - mcb.c, [146](#)
 - mcb.h, [154](#)
- mcb_type
 - mcb.c, [142](#)
- memset
 - string.c, [35](#)
 - string.h, [25](#)
- modules/errno.h, [40](#)
- modules/mpx_supt.c, [42](#)
- modules/mpx_supt.h, [45](#)
- modules/r1/r1.c, [50](#)
- modules/r1/r1.h, [57](#)
- modules/r1/sys_clock.c, [62](#)
- modules/r1/sys_clock.h, [74](#)
- modules/r2/pcb.c, [83](#)
- modules/r2/pcb.h, [100](#)
- modules/r2/pcb_comm.c, [117](#)
- modules/r2/pcb_comm.h, [121](#)
- modules/r3/context.c, [126](#)
- modules/r3/context.h, [131](#)
- modules/r5/mcb.c, [135](#)
- modules/r5/mcb.h, [149](#)
- mpx
 - r1.h, [62](#)
- mpx_init
 - mpx_supt.c, [44](#)
 - mpx_supt.h, [49](#)
- mpx_supt.c
 - current_module, [45](#)
 - get_op_code, [43](#)
 - idle, [44](#)
 - mpx_init, [44](#)
 - params, [45](#)
 - student_free, [45](#)
 - student_malloc, [45](#)
 - sys_alloc_mem, [44](#)
 - sys_free_mem, [44](#)
 - sys_req, [45](#)
 - sys_set_free, [45](#)
 - sys_set_malloc, [45](#)
- mpx_supt.h
 - __attribute__, [50](#)
 - EXIT, [48](#)
 - get_op_code, [49](#)
 - IDLE, [48](#)
 - idle, [49](#)
 - MODULE_R1, [48](#)
 - MODULE_R2, [48](#)
 - MODULE_R3, [48](#)
 - MODULE_R4, [48](#)
 - MODULE_R5, [48](#)
 - mpx_init, [49](#)
 - READ, [48](#)
 - sys_alloc_mem, [49](#)
 - sys_free_mem, [49](#)
 - sys_req, [50](#)
 - sys_set_free, [50](#)
 - sys_set_malloc, [50](#)
 - WRITE, [48](#)
- name
 - pcb_struct, [16](#)
- nameStr
 - function_name, [11](#)
- next
 - mcb, [12](#)
 - pcb_struct, [16](#)
- NormalWriting
 - r1.c, [56](#)
- NotWriting
 - r1.c, [57](#)
- old_context
 - context.c, [131](#)
 - context.h, [135](#)
- op_code
 - param, [13](#)
- param, [13](#)

- device_id, 13
- op_code, 13
- params
 - mpx_supt.c, 45
- pcb
 - r1.h, 62
- pcb.c
 - __attribute__, 89, 100
 - allocate_pcb, 89
 - block_pcb, 90
 - blocked, 100
 - false, 100
 - find_pcb, 90
 - free_pcb, 91
 - get_running_process, 91
 - get_stack_base, 92
 - get_stack_top, 92
 - insert_pcb, 93
 - pcb_init, 93
 - process_state, 89
 - process_suspended, 89
 - ready, 100
 - remove_pcb, 93
 - resume_pcb, 94
 - running, 100
 - save_running_process, 94
 - set_pcb_priority, 95
 - setup_pcb, 95
 - show_all_processes, 96
 - show_blocked_processes, 97
 - show_pcb, 97
 - show_ready_processes, 98
 - shutdown_pcb, 99
 - suspend_pcb, 99
 - true, 100
 - unblock_pcb, 99
- pcb.h
 - __attribute__, 107
 - allocate_pcb, 107
 - block_pcb, 107
 - COMMHAND_PCB_NAME, 107
 - find_pcb, 108
 - free_pcb, 108
 - get_running_process, 109
 - get_stack_base, 110
 - get_stack_top, 110
 - IDLE_PCB_NAME, 107
 - insert_pcb, 110
 - pcb_class_app, 117
 - pcb_class_sys, 117
 - pcb_init, 111
 - process_class, 107
 - remove_pcb, 111
 - resume_pcb, 111
 - SIZE_OF_PCB_NAME, 107
 - SIZE_OF_STACK, 107
 - save_running_process, 112
 - set_pcb_priority, 112
 - setup_pcb, 113
 - show_all_processes, 114
 - show_blocked_processes, 114
 - show_pcb, 115
 - show_ready_processes, 115
 - shutdown_pcb, 116
 - suspend_pcb, 116
 - unblock_pcb, 117
- pcb_class_app
 - pcb.h, 117
- pcb_class_sys
 - pcb.h, 117
- pcb_comm.c
 - resume_pcb_main, 120
 - set_pcb_priority_main, 120
 - show_pcb_main, 120
 - suspend_pcb_main, 121
- pcb_comm.h
 - resume_pcb_main, 125
 - set_pcb_priority_main, 125
 - show_pcb_main, 125
 - suspend_pcb_main, 126
- pcb_init
 - pcb.c, 93
 - pcb.h, 111
- pcb_queue, 13
 - count, 14
 - head, 14
 - tail, 14
- pcb_struct, 15
 - class, 15
 - is_suspended, 15
 - name, 16
 - next, 16
 - prev, 16
 - priority, 16
 - running_state, 16
 - stack_base, 16
 - stack_top, 16
- prev
 - mcb, 12
 - pcb_struct, 16
- print_help
 - r1.c, 55
 - r1.h, 59
- printf
 - string.c, 36
 - string.h, 26
- priority
 - pcb_struct, 16

process_class
 pcb.h, 107
process_state
 pcb.c, 89
process_suspended
 pcb.c, 89

r1.c
 __attribute__, 54
 COMPLETION, 54
 command_line_parser, 54
 CommandPaserStat, 54
 commhand, 54
 DoubleQuoteWriting, 56
 help_usages, 54
 MAX_ARGC, 54
 MAX_HISTORY, 54
 MOD_VERSION, 54
 NormalWriting, 56
 NotWriting, 57
 print_help, 55
 SingleQuoteWriting, 57
r1.h
 __attribute__, 59
 comm_type, 59
 command_line_parser, 59
 commhand, 59
 help, 61
 help_usages, 59
 mcb, 61
 mpx, 62
 pcb, 62
 print_help, 59
READ
 mpx_supt.h, 48
RTC_INDEX_HOUR
 sys_clock.c, 68
RTC_INDEX_MINUTE
 sys_clock.c, 68
RTC_INDEX_MONTH
 sys_clock.c, 68
RTC_INDEX_SECOND
 sys_clock.c, 68
RTC_INDEX_YEAR
 sys_clock.c, 68
ready
 pcb.c, 100
remove_pcb
 pcb.c, 93
 pcb.h, 111
resume_pcb
 pcb.c, 94
 pcb.h, 111
resume_pcb_main
 pcb_comm.c, 120
 pcb_comm.h, 125
running
 pcb.c, 100
running_state
 pcb_struct, 16

SIZE_OF_PCB_NAME
 pcb.h, 107
SIZE_OF_STACK
 pcb.h, 107
save_running_process
 pcb.c, 94
 pcb.h, 112
serial.h
 COM1, 18
 COM2, 18
 COM3, 18
 COM4, 18
 get_input_line, 19
 init_serial, 19
 serial_print, 19
 serial_println, 19
 set_serial_in, 19
 set_serial_out, 20
 WithEcho, 18
 WithoutEcho, 18
serial_print
 serial.h, 19
serial_println
 serial.h, 19
set_date
 sys_clock.c, 70
 sys_clock.h, 79
set_date_main
 sys_clock.c, 70
 sys_clock.h, 79
set_date_str
 sys_clock.c, 71
 sys_clock.h, 80
set_pcb_priority
 pcb.c, 95
 pcb.h, 112
set_pcb_priority_main
 pcb_comm.c, 120
 pcb_comm.h, 125
set_serial_in
 serial.h, 19
set_serial_out
 serial.h, 20
set_time
 sys_clock.c, 72
 sys_clock.h, 81
set_time_main

- sys_clock.c, 72
- sys_clock.h, 81
- set_time_str
 - sys_clock.c, 73
 - sys_clock.h, 82
- setup_pcb
 - pcb.c, 95
 - pcb.h, 113
- show_all_mcb
 - mcb.c, 146
 - mcb.h, 154
- show_all_processes
 - pcb.c, 96
 - pcb.h, 114
- show_allocated_mcb
 - mcb.c, 146
 - mcb.h, 155
- show_blocked_processes
 - pcb.c, 97
 - pcb.h, 114
- show_free_mcb
 - mcb.c, 147
 - mcb.h, 155
- show_mcb
 - mcb.c, 148
 - mcb.h, 156
- show_mcb_main
 - mcb.c, 148
 - mcb.h, 157
- show_pcb
 - pcb.c, 97
 - pcb.h, 115
- show_pcb_main
 - pcb_comm.c, 120
 - pcb_comm.h, 125
- show_ready_processes
 - pcb.c, 98
 - pcb.h, 115
- shutdown_mcb
 - mcb.c, 149
 - mcb.h, 157
- shutdown_pcb
 - pcb.c, 99
 - pcb.h, 116
- SingleQuoteWriting
 - r1.c, 57
- size
 - cmcb, 7
 - lmcb, 11
- sprintf
 - string.c, 38
 - string.h, 28
- stack_base
 - pcb_struct, 16
- stack_top
 - pcb_struct, 16
- start_of_memory
 - mcb.c, 149
 - mcb.h, 157
- strcat
 - string.c, 38
 - string.h, 28
- strcmp
 - string.c, 38
 - string.h, 28
- strcpy
 - string.c, 39
 - string.h, 29
- string.c
 - atoi, 34
 - isspace, 35
 - memset, 35
 - printf, 36
 - sprintf, 38
 - strcat, 38
 - strcmp, 38
 - strcpy, 39
 - strlen, 39
 - strtok, 40
- string.h
 - atoi, 23
 - isspace, 24
 - memset, 25
 - printf, 26
 - sprintf, 28
 - strcat, 28
 - strcmp, 28
 - strcpy, 29
 - strlen, 29
 - strtok, 30
- strlen
 - string.c, 39
 - string.h, 29
- strtok
 - string.c, 40
 - string.h, 30
- student_free
 - mpx_supt.c, 45
- student_malloc
 - mpx_supt.c, 45
- suspend_pcb
 - pcb.c, 99
 - pcb.h, 116
- suspend_pcb_main
 - pcb_comm.c, 121
 - pcb_comm.h, 126
- sys_alloc_mem
 - mpx_supt.c, 44

- mpx_supt.h, 49
- sys_call
 - context.c, 130
 - context.h, 134
- sys_clock.c
 - get_date, 68
 - get_date_main, 68
 - get_time, 69
 - get_time_main, 69
 - RTC_INDEX_HOUR, 68
 - RTC_INDEX_MINUTE, 68
 - RTC_INDEX_MONTH, 68
 - RTC_INDEX_SECOND, 68
 - RTC_INDEX_YEAR, 68
 - set_date, 70
 - set_date_main, 70
 - set_date_str, 71
 - set_time, 72
 - set_time_main, 72
 - set_time_str, 73
- sys_clock.h
 - get_date, 77
 - get_date_main, 78
 - get_time, 78
 - get_time_main, 78
 - set_date, 79
 - set_date_main, 79
 - set_date_str, 80
 - set_time, 81
 - set_time_main, 81
 - set_time_str, 82
- sys_free_mem
 - mpx_supt.c, 44
 - mpx_supt.h, 49
- sys_req
 - mpx_supt.c, 45
 - mpx_supt.h, 50
- sys_set_free
 - mpx_supt.c, 45
 - mpx_supt.h, 50
- sys_set_malloc
 - mpx_supt.c, 45
 - mpx_supt.h, 50
- tail
 - pcb_queue, 14
- true
 - pcb.c, 100
- type
 - cmcb, 7
 - lmcb, 11
- unblock_pcb
 - pcb.c, 99
 - pcb.h, 117
- usage
 - function_name, 11
- WRITE
 - mpx_supt.h, 48
- WithEcho
 - serial.h, 18
- WithoutEcho
 - serial.h, 18