MPX-Fall2020-Group9

5

Generated by Doxygen 1.9.0

Chapter 1

MPX-Fall2020-Group9

WVU CS 450 MPX Project files Making operating system// test message

Chapter 2

Class Index

2.1 Class List

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

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

Chapter 4

Class Documentation

4.1 alarm Struct Reference

`#include <R4commands.h>`

Public Attributes

- char alarmName [20]
- int alarmTime
- struct alarm * nextAlarm
- struct alarm * prevAlarm

4.1.1 Detailed Description

Definition at line 3 of file R4commands.h.

4.1.2 Member Data Documentation

4.1.2.1 alarmName

`char alarm::alarmName[20]`

Definition at line 5 of file R4commands.h.

4.1.2.2 alarmTime

`int alarm::alarmTime`

Definition at line 6 of file R4commands.h.

4.1.2.3 nextAlarm

```
struct alarm* alarm::nextAlarm
```

Definition at line 7 of file R4commands.h.

4.1.2.4 prevAlarm

```
struct alarm* alarm::prevAlarm
```

Definition at line 8 of file R4commands.h.

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

- modules/R4/R4commands.h

4.2 alarmList Struct Reference

```
#include <R4commands.h>
```

Public Attributes

- int count
- alarm * head
- alarm * tail

4.2.1 Detailed Description

Definition at line 11 of file R4commands.h.

4.2.2 Member Data Documentation

4.2.2.1 count

```
int alarmList::count
```

Definition at line 13 of file R4commands.h.

4.2.2.2 head

```
alarm* alarmList::head
```

Definition at line 14 of file R4commands.h.

4.2.2.3 tail

```
alarm* alarmList::tail
```

Definition at line 15 of file R4commands.h.

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

- modules/R4/R4commands.h

4.3 CMCB Struct Reference

```
#include <R5commands.h>
```

Public Attributes

- char type
- u32int beginningAddr
- u32int size
- struct CMCB * nextCMCB
- struct CMCB * prevCMCB

4.3.1 Detailed Description

Definition at line 1 of file R5commands.h.

4.3.2 Member Data Documentation

4.3.2.1 beginningAddr

```
u32int CMCB::beginningAddr
```

Definition at line 4 of file R5commands.h.

4.3.2.2 nextCMCB

```
struct CMCB* CMCB::nextCMCB
```

Definition at line 7 of file R5commands.h.

4.3.2.3 prevCMCB

```
struct CMCB* CMCB::prevCMCB
```

Definition at line 8 of file R5commands.h.

4.3.2.4 size

```
u32int CMCB::size
```

Definition at line 5 of file R5commands.h.

4.3.2.5 type

```
char CMCB::type
```

Definition at line 3 of file R5commands.h.

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

- modules/R5/R5commands.h

4.4 context Struct Reference

```
#include <R3commands.h>
```

Public Attributes

- u32int gs
- u32int fs
- u32int es
- u32int ds
- u32int edi
- u32int esi
- u32int ebp
- u32int esp
- u32int ebx
- u32int edx
- u32int ecx
- u32int eax
- u32int eip
- u32int cs
- u32int eflags

4.4.1 Detailed Description

Definition at line 3 of file R3commands.h.

4.4.2 Member Data Documentation

4.4.2.1 cs

```
u32int context::cs
```

Definition at line 7 of file R3commands.h.

4.4.2.2 ds

```
u32int context::ds
```

Definition at line 5 of file R3commands.h.

4.4.2.3 eax

```
u32int context::eax
```

Definition at line 6 of file R3commands.h.

4.4.2.4 ebp

```
u32int context::ebp
```

Definition at line 6 of file R3commands.h.

4.4.2.5 ebx

```
u32int context::ebx
```

Definition at line 6 of file R3commands.h.

4.4.2.6 ecx

```
u32int context::ecx
```

Definition at line 6 of file R3commands.h.

4.4.2.7 edi

```
u32int context::edi
```

Definition at line 6 of file R3commands.h.

4.4.2.8 edx

```
u32int context::edx
```

Definition at line 6 of file R3commands.h.

4.4.2.9 eflags

```
u32int context::eflags
```

Definition at line 7 of file R3commands.h.

4.4.2.10 eip

```
u32int context::eip
```

Definition at line 7 of file R3commands.h.

4.4.2.11 es

```
u32int context::es
```

Definition at line 5 of file R3commands.h.

4.4.2.12 esi

```
u32int context::esi
```

Definition at line 6 of file R3commands.h.

4.4.2.13 esp

```
u32int context::esp
```

Definition at line 6 of file R3commands.h.

4.4.2.14 fs

```
u32int context::fs
```

Definition at line 5 of file R3commands.h.

4.4.2.15 gs

```
u32int context::gs
```

Definition at line 5 of file R3commands.h.

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

- modules/R3/R3commands.h

4.5 date_time Struct Reference

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

Public Attributes

- int sec
- int min
- int hour
- int day_w
- int day_m
- int day_y
- int mon
- int year

4.5.1 Detailed Description

Definition at line 30 of file system.h.

4.5.2 Member Data Documentation

4.5.2.1 day_m

```
int date_time::day_m
```

Definition at line 35 of file system.h.

4.5.2.2 day_w

```
int date_time::day_w
```

Definition at line 34 of file system.h.

4.5.2.3 day_y

```
int date_time::day_y
```

Definition at line 36 of file system.h.

4.5.2.4 hour

```
int date_time::hour
```

Definition at line 33 of file system.h.

4.5.2.5 min

```
int date_time::min
```

Definition at line 32 of file system.h.

4.5.2.6 mon

```
int date_time::mon
```

Definition at line 37 of file system.h.

4.5.2.7 sec

```
int date_time::sec
```

Definition at line 31 of file system.h.

4.5.2.8 year

```
int date_time::year
```

Definition at line 38 of file system.h.

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

- include/system.h

4.6 footer Struct Reference

```
#include <heap.h>
```

Public Attributes

- header head

4.6.1 Detailed Description

Definition at line 16 of file heap.h.

4.6.2 Member Data Documentation

4.6.2.1 head

```
header footer::head
```

Definition at line 17 of file heap.h.

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

- include/mem/heap.h

4.7 gdt_descriptor_struct Struct Reference

```
#include <tables.h>
```

Public Attributes

- u16int limit
- u32int base

4.7.1 Detailed Description

Definition at line 23 of file tables.h.

4.7.2 Member Data Documentation

4.7.2.1 base

```
u32int gdt_descriptor_struct::base
```

Definition at line 26 of file tables.h.

4.7.2.2 limit

```
u16int gdt_descriptor_struct::limit
```

Definition at line 25 of file tables.h.

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

- include/core/tables.h

4.8 gdt_entry_struct Struct Reference

`#include <tables.h>`

Public Attributes

- u16int limit_low
- u16int base_low
- u8int base_mid
- u8int access
- u8int flags
- u8int base_high

4.8.1 Detailed Description

Definition at line 30 of file tables.h.

4.8.2 Member Data Documentation

4.8.2.1 access

`u8int gdt_entry_struct::access`

Definition at line 35 of file tables.h.

4.8.2.2 base_high

`u8int gdt_entry_struct::base_high`

Definition at line 37 of file tables.h.

4.8.2.3 base_low

`u16int gdt_entry_struct::base_low`

Definition at line 33 of file tables.h.

4.8.2.4 base_mid

```
u8int gdt_entry_struct::base_mid
```

Definition at line 34 of file tables.h.

4.8.2.5 flags

```
u8int gdt_entry_struct::flags
```

Definition at line 36 of file tables.h.

4.8.2.6 limit_low

```
u16int gdt_entry_struct::limit_low
```

Definition at line 32 of file tables.h.

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

- include/core/tables.h

4.9 header Struct Reference

```
#include <heap.h>
```

Public Attributes

- int size
- int index_id

4.9.1 Detailed Description

Definition at line 11 of file heap.h.

4.9.2 Member Data Documentation

4.9.2.1 index_id

```
int header::index_id
```

Definition at line 13 of file heap.h.

4.9.2.2 size

```
int header::size
```

Definition at line 12 of file heap.h.

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

- include/mem/heap.h

4.10 heap Struct Reference

```
#include <heap.h>
```

Public Attributes

- index_table index
- u32int base
- u32int max_size
- u32int min_size

4.10.1 Detailed Description

Definition at line 33 of file heap.h.

4.10.2 Member Data Documentation

4.10.2.1 base

```
u32int heap::base
```

Definition at line 35 of file heap.h.

4.10.2.2 index

```
index_table heap::index
```

Definition at line 34 of file heap.h.

4.10.2.3 max_size

```
u32int heap::max_size
```

Definition at line 36 of file heap.h.

4.10.2.4 min_size

```
u32int heap::min_size
```

Definition at line 37 of file heap.h.

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

- include/mem/heap.h

4.11 idt_entry_struct Struct Reference

```
#include <tables.h>
```

Public Attributes

- u16int base_low
- u16int sselect
- u8int zero
- u8int flags
- u16int base_high

4.11.1 Detailed Description

Definition at line 6 of file tables.h.

4.11.2 Member Data Documentation

4.11.2.1 base_high

`u16int idt_entry_struct::base_high`

Definition at line 12 of file tables.h.

4.11.2.2 base_low

`u16int idt_entry_struct::base_low`

Definition at line 8 of file tables.h.

4.11.2.3 flags

`u8int idt_entry_struct::flags`

Definition at line 11 of file tables.h.

4.11.2.4 sselect

`u16int idt_entry_struct::sselect`

Definition at line 9 of file tables.h.

4.11.2.5 zero

`u8int idt_entry_struct::zero`

Definition at line 10 of file tables.h.

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

- include/core/tables.h

4.12 idt_struct Struct Reference

`#include <tables.h>`

Public Attributes

- u16int limit
- u32int base

4.12.1 Detailed Description

Definition at line 16 of file tables.h.

4.12.2 Member Data Documentation

4.12.2.1 base

```
u32int idt_struct::base
```

Definition at line 19 of file tables.h.

4.12.2.2 limit

```
u16int idt_struct::limit
```

Definition at line 18 of file tables.h.

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

- include/core/tables.h

4.13 index_entry Struct Reference

```
#include <heap.h>
```

Public Attributes

- int size
- int empty
- u32int block

4.13.1 Detailed Description

Definition at line 20 of file heap.h.

4.13.2 Member Data Documentation

4.13.2.1 block

```
u32int index_entry::block
```

Definition at line 23 of file heap.h.

4.13.2.2 empty

```
int index_entry::empty
```

Definition at line 22 of file heap.h.

4.13.2.3 size

```
int index_entry::size
```

Definition at line 21 of file heap.h.

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

- include/mem/heap.h

4.14 index_table Struct Reference

```
#include <heap.h>
```

Public Attributes

- index_entry table [0x1000]
- int id

4.14.1 Detailed Description

Definition at line 27 of file heap.h.

4.14.2 Member Data Documentation

4.14.2.1 id

```
int index_table::id
```

Definition at line 29 of file heap.h.

4.14.2.2 table

```
index_entry index_table::table[0x1000]
```

Definition at line 28 of file heap.h.

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

- include/mem/heap.h

4.15 memList Struct Reference

```
#include <R5commands.h>
```

Public Attributes

- int count
- CMCB * head
- CMCB * tail

4.15.1 Detailed Description

Definition at line 17 of file R5commands.h.

4.15.2 Member Data Documentation

4.15.2.1 count

```
int memList::count
```

Definition at line 19 of file R5commands.h.

4.15.2.2 head

```
CMCB* memList::head
```

Definition at line 20 of file R5commands.h.

4.15.2.3 tail

```
CMCB* memList::tail
```

Definition at line 21 of file R5commands.h.

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

- modules/R5/R5commands.h

4.16 page_dir Struct Reference

```
#include <paging.h>
```

Public Attributes

- page_table * tables [1024]
- u32int tables_phys [1024]

4.16.1 Detailed Description

Definition at line 34 of file paging.h.

4.16.2 Member Data Documentation

4.16.2.1 tables

```
page_table* page_dir::tables[1024]
```

Definition at line 35 of file paging.h.

4.16.2.2 tables_phys

```
u32int page_dir::tables_phys[1024]
```

Definition at line 36 of file paging.h.

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

- include/mem/paging.h

4.17 page_entry Struct Reference

```
#include <paging.h>
```

Public Attributes

- u32int present: 1
- u32int writeable: 1
- u32int usermode: 1
- u32int accessed: 1
- u32int dirty: 1
- u32int reserved: 7
- u32int frameaddr: 20

4.17.1 Detailed Description

Definition at line 12 of file paging.h.

4.17.2 Member Data Documentation

4.17.2.1 accessed

```
u32int page_entry::accessed
```

Definition at line 16 of file paging.h.

4.17.2.2 dirty

```
u32int page_entry::dirty
```

Definition at line 17 of file paging.h.

4.17.2.3 frameaddr

```
u32int page_entry::frameaddr
```

Definition at line 19 of file paging.h.

4.17.2.4 present

```
u32int page_entry::present
```

Definition at line 13 of file paging.h.

4.17.2.5 reserved

```
u32int page_entry::reserved
```

Definition at line 18 of file paging.h.

4.17.2.6 usermode

```
u32int page_entry::usermode
```

Definition at line 15 of file paging.h.

4.17.2.7 writeable

```
u32int page_entry::writeable
```

Definition at line 14 of file paging.h.

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

- include/mem/paging.h

4.18 page_table Struct Reference

```
#include <paging.h>
```

Public Attributes

- page_entry pages [1024]

4.18.1 Detailed Description

Definition at line 26 of file paging.h.

4.18.2 Member Data Documentation

4.18.2.1 pages

```
page_entry page_table::pages[1024]
```

Definition at line 27 of file paging.h.

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

- include/mem/paging.h

4.19 param Struct Reference

```
#include <mpx_supt.h>
```

Public Attributes

- int op_code
- int device_id
- char * buffer_ptr
- int * count_ptr

4.19.1 Detailed Description

Definition at line 31 of file mpx_supt.h.

4.19.2 Member Data Documentation

4.19.2.1 buffer_ptr

```
char* param::buffer_ptr
```

Definition at line 34 of file mpx_supt.h.

4.19.2.2 count_ptr

```
int* param::count_ptr
```

Definition at line 35 of file mpx_supt.h.

4.19.2.3 device_id

```
int param::device_id
```

Definition at line 33 of file mpx_supt.h.

4.19.2.4 op_code

```
int param::op_code
```

Definition at line 32 of file mpx_supt.h.

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

- modules/mpx_supt.h

4.20 PCB Struct Reference

```
#include <R2_Internal_Functions_And_Structures.h>
```

Public Attributes

- char processName [20]
- char processClass
- int priority
- int runningStatus
- int suspendedStatus
- unsigned char stack [1024]
- unsigned char * stackTop
- unsigned char * stackBase
- struct PCB * nextPCB
- struct PCB * prevPCB

4.20.1 Detailed Description

Definition at line 1 of file R2_Internal_Functions_And_Structures.h.

4.20.2 Member Data Documentation

4.20.2.1 nextPCB

```
struct PCB* PCB::nextPCB
```

Definition at line 11 of file R2_Internal_Functions_And_Structures.h.

4.20.2.2 prevPCB

```
struct PCB* PCB::prevPCB
```

Definition at line 12 of file R2_Internal_Functions_And_Structures.h.

4.20.2.3 priority

```
int PCB::priority
```

Definition at line 5 of file R2_Internal_Functions_And_Structures.h.

4.20.2.4 processClass

```
char PCB::processClass
```

Definition at line 4 of file R2_Internal_Functions_And_Structures.h.

4.20.2.5 processName

```
char PCB::processName[20]
```

Definition at line 3 of file R2_Internal_Functions_And_Structures.h.

4.20.2.6 runningStatus

```
int PCB::runningStatus
```

Definition at line 6 of file R2_Internal_Functions_And_Structures.h.

4.20.2.7 stack

```
unsigned char PCB::stack[1024]
```

Definition at line 8 of file R2_Internal_Functions_And_Structures.h.

4.20.2.8 stackBase

```
unsigned char* PCB::stackBase
```

Definition at line 10 of file R2_Internal_Functions_And_Structures.h.

4.20.2.9 stackTop

```
unsigned char* PCB::stackTop
```

Definition at line 9 of file R2_Internal_Functions_And_Structures.h.

4.20.2.10 suspendedStatus

```
int PCB::suspendedStatus
```

Definition at line 7 of file R2_Internal_Functions_And_Structures.h.

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

- modules/R2/R2_Internal_Functions_And_Structures.h

4.21 queue Struct Reference

```
#include <R2_Internal_Functions_And_Structures.h>
```

Public Attributes

- int count
- PCB * head
- PCB * tail

4.21.1 Detailed Description

Definition at line 15 of file R2_Internal_Functions_And_Structures.h.

4.21.2 Member Data Documentation

4.21.2.1 count

```
int queue::count
```

Definition at line 17 of file R2_Internal_Functions_And_Structures.h.

4.21.2.2 head

```
PCB* queue::head
```

Definition at line 18 of file R2_Internal_Functions_And_Structures.h.

4.21.2.3 tail

```
PCB* queue::tail
```

Definition at line 19 of file R2_Internal_Functions_And_Structures.h.

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

- modules/R2/R2_Internal_Functions_And_Structures.h

Chapter 5

File Documentation

5.1 include/core/asm.h File Reference

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

5.2 include/core/interrupts.h File Reference

Functions

- void init_irq (void)
- void init_pic (void)

5.2.1 Function Documentation

5.2.1.1 init_irq()

```
void init_irq (
            void  )
```

Definition at line 67 of file interrupts.c.

```
68 {
69   int i;
70
71   // Necessary interrupt handlers for protected mode
72   u32int isrs[17] = {
73       (u32int)divide_error,
74       (u32int)debug,
75       (u32int)nmi,
76       (u32int)breakpoint,
77       (u32int)overflow,
78       (u32int)bounds,
79       (u32int)invalid_op,
80       (u32int)device_not_available,
81       (u32int)double_fault,
82       (u32int)coprocessor_segment,
```

```
83        (u32int)invalid_tss,
84        (u32int)segment_not_present,
85        (u32int)stack_segment,
86        (u32int)general_protection,
87        (u32int)page_fault,
88        (u32int)reserved,
89        (u32int)coprocessor};
90
91    // Install handlers; 0x08=sel, 0x8e=flags
92    for (i = 0; i < 32; i++)
93    {
94      if (i < 17)
95        idt_set_gate(i, isrs[i], 0x08, 0x8e);
96      else
97        idt_set_gate(i, (u32int)reserved, 0x08, 0x8e);
98    }
99    // Ignore interrupts from the real time clock
100    idt_set_gate(0x08, (u32int)rtc_isr, 0x08, 0x8e);
101    idt_set_gate(60, (u32int)sys_call_isr, 0x08, 0x8e);
102 }
```

5.2.1.2 init_pic()

```
void init_pic (
              void  )
```

Definition at line 110 of file interrupts.c.

```
111 {
112    outb(PIC1, ICW1); //send initialization code words 1 to PIC1
113    io_wait();
114    outb(PIC2, ICW1); //send icw1 to PIC2
115    io_wait();
116    outb(PIC1 + 1, 0x20); //icw2: remap irq0 to 32
117    io_wait();
118    outb(PIC2 + 1, 0x28); //icw2: remap irq8 to 40
119    io_wait();
120    outb(PIC1 + 1, 4); //icw3
121    io_wait();
122    outb(PIC2 + 1, 2); //icw3
123    io_wait();
124    outb(PIC1 + 1, ICW4); //icw4: 80x86, automatic handling
125    io_wait();
126    outb(PIC2 + 1, ICW4); //icw4: 80x86, automatic handling
127    io_wait();
128    outb(PIC1 + 1, 0xFF); //disable irqs for PIC1
129    io_wait();
130    outb(PIC2 + 1, 0xFF); //disable irqs for PIC2
131 }
```

5.3 include/core/io.h File Reference

Macros

- #define outb(port, data) asm volatile ("outb %%al,%%dx" : : "a" (data), "d" (port))
- #define inb(port)

5.3.1 Macro Definition Documentation

5.3.1.1 inb

```
#define inb(
              port )
```

Value:

```
      ({                          \
      unsigned char r;                    \
      asm volatile ("inb %%dx,%%al": "=a" (r): "d" (port)); \
      r;                                  \
    })
```

Definition at line 15 of file io.h.

5.3.1.2 outb

```
#define outb(
              port,
              data )  asm volatile ("outb %%al,%%dx" :  :  "a" (data), "d" (port))
```

Definition at line 8 of file io.h.

5.4 include/core/serial.h File Reference

Macros

- #define COM1 0x3f8
- #define COM2 0x2f8
- #define COM3 0x3e8
- #define COM4 0x2e8

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)
- int ∗ polling (char ∗buffer, int ∗count)

5.4.1 Macro Definition Documentation

5.4.1.1 COM1

```
#define COM1 0x3f8
```

Definition at line 4 of file serial.h.

5.4.1.2 COM2

```
#define COM2 0x2f8
```

Definition at line 5 of file serial.h.

5.4.1.3 COM3

```
#define COM3 0x3e8
```

Definition at line 6 of file serial.h.

5.4.1.4 COM4

```
#define COM4 0x2e8
```

Definition at line 7 of file serial.h.

5.4.2 Function Documentation

5.4.2.1 init_serial()

```
int init_serial (
            int device )
```

Definition at line 22 of file serial.c.

```
23 {
24   outb(device + 1, 0x00);          //disable interrupts
25   outb(device + 3, 0x80);          //set line control register
26   outb(device + 0, 115200 / 9600); //set bsd least sig bit
27   outb(device + 1, 0x00);          //brd most significant bit
28   outb(device + 3, 0x03);          //lock divisor; 8bits, no parity, one stop
29   outb(device + 2, 0xC7);          //enable fifo, clear, 14byte threshold
30   outb(device + 4, 0x0B);          //enable interrupts, rts/dsr set
31   (void)inb(device);               //read bit to reset port
32   return NO_ERROR;
33 }
```

5.4.2.2 polling()

```
int* polling (
            char * buffer,
            int * count )
```

Definition at line 92 of file serial.c.

```
93 {
94   // insert your code to gather keyboard input via the technique of polling.
95
96   char keyboard_character;
97
98   int cursor = 0;
99
100   char log[] = {'\0', '\0', '\0', '\0'};
101
102   int characters_in_buffer = 0;
103
104   while (1)
105   {
106
107     if (inb(COM1 + 5) & 1)
108     {                                   // is there input char?
109       keyboard_character = inb(COM1); //read the char from COM1
110
111       if (keyboard_character == '\n' || keyboard_character == '\r')
112       { // HANDLEING THE CARRIAGE RETURN AND NEW LINE CHARACTERS
113
114         buffer[characters_in_buffer] = '\0';
115         break;
116       }
117       else if ((keyboard_character == 127 || keyboard_character == 8) && cursor > 0)
118       { // HANDELING THE BACKSPACE CHARACTER
119
120         //serial_println("Handleing backspace character.");
121         serial_print("\033[K");
122
123         buffer[cursor - 1] = '\0';
124         serial_print("\b \b");
125         serial_print(buffer + cursor);
126         cursor--;
127
128         int temp_cursor = cursor;
129
130         while (buffer[temp_cursor + 1] != '\0')
131         {
132           buffer[temp_cursor] = buffer[temp_cursor + 1];
133           buffer[temp_cursor + 1] = '\0';
134           temp_cursor++;
135         }
136
137         characters_in_buffer--;
138         cursor = characters_in_buffer;
139       }
140       else if (keyboard_character == '~' && cursor < 99)
141       { //HANDLING THE DELETE KEY
142         // \033[3~
143
144         serial_print("\033[K");
145
146         buffer[cursor + 1] = '\0';
147         serial_print("\b \b");
148         serial_print(buffer + cursor);
149
150         int temp_cursor = cursor + 1;
151
152         while (buffer[temp_cursor + 1] != '\0')
153         {
154           buffer[temp_cursor] = buffer[temp_cursor + 1];
155           buffer[temp_cursor + 1] = '\0';
156           temp_cursor++;
157         }
158
159         characters_in_buffer--;
160         cursor = characters_in_buffer;
161       }
162       else if (keyboard_character == '\033')
163       { // HANDLEING FIRST CHARACTER FOR ARROW KEYS
164
165         log[0] = keyboard_character;
166       }
167       else if (keyboard_character == '[' && log[0] == '\033')
168       { // HANDLEING SECOND CHARACTER FOR ARROW KEYS
169
```

```
170         log[1] = keyboard_character;
171       }
172       else if (log[0] == '\033' && log[1] == '[')
173       { // HANDLEING LAST CHARACTER FOR ARROW KEYS
174         log[2] = keyboard_character;
175 
176         if (keyboard_character == 'A')
177         { //Up arrow
178           //Call a history function from the commhand or do nothing
179         }
180         else if (keyboard_character == 'B')
181         { //Down arrow
182           //Call a history command from the commhand or do nothing
183         }
184         else if (keyboard_character == 'C' && cursor != 99)
185         { //Right arrow
186 
187           serial_print("\033[C");
188           cursor++;
189         }
190         else if (keyboard_character == 'D' && cursor != 0)
191         { //Left arrow
192 
193           serial_print("\033[D");
194           cursor--;
195         }
196 
197         memset(log, '\0', 4);
198       }
199       else
200       {
201 
202         if (cursor == 0 && buffer[cursor] == '\0') //Adding character at beginning of buffer
203         {
204           buffer[cursor] = keyboard_character;
205           serial_print(buffer + cursor);
206           cursor++;
207         }
208         else if (buffer[cursor] == '\0') //Adding character at the end of the buffer
209         {
210           buffer[cursor] = keyboard_character;
211           serial_print(buffer + cursor);
212           cursor++;
213         }
214         else //Inserting character to the middle of the buffer
215         {
216           char temp_buffer[strlen(buffer)];
217           memset(temp_buffer, '\0', strlen(buffer));
218 
219           int temp_cursor = 0;
220           while (temp_cursor <= characters_in_buffer) //Filling the temp_buffer with all of the
    characters from buffer, and inserting the new character.
221           {
222             if (temp_cursor < cursor)
223             {
224               temp_buffer[temp_cursor] = buffer[temp_cursor];
225             }
226             else if (temp_cursor > cursor)
227             {
228               temp_buffer[temp_cursor] = buffer[temp_cursor - 1];
229             }
230             else
231             { //temp_cursor == cursor
232               temp_buffer[temp_cursor] = keyboard_character;
233             }
234             temp_cursor++;
235           }
236 
237           temp_cursor = 0;
238           int temp_buffer_size = strlen(temp_buffer);
239           while (temp_cursor <= temp_buffer_size) //Setting the contents of the buffer equal to the
    temp_buffer.
240           {
241             buffer[temp_cursor] = temp_buffer[temp_cursor];
242             temp_cursor++;
243           }
244 
245           serial_print("\033[K");
246           serial_print(&keyboard_character);
247           serial_print(buffer + cursor + 1);
248           cursor++;
249         }
250         characters_in_buffer++;
251       }
252     }
253   }
254 
```

```
255   *count = characters_in_buffer; // buffer count
256
257   return count;
258 }
```

5.4.2.3 serial_print()

```
int serial_print (
              const char * msg )
```

Definition at line 56 of file serial.c.

```
57 {
58   int i;
59   for (i = 0; *(i + msg) != '\0'; i++)
60   {
61     outb(serial_port_out, *(i + msg));
62   }
63   if (*msg == '\r')
64     outb(serial_port_out, '\n');
65   return NO_ERROR;
66 }
```

5.4.2.4 serial_println()

```
int serial_println (
              const char * msg )
```

Definition at line 40 of file serial.c.

```
41 {
42   int i;
43   for (i = 0; *(i + msg) != '\0'; i++)
44   {
45     outb(serial_port_out, *(i + msg));
46   }
47   outb(serial_port_out, '\r');
48   outb(serial_port_out, '\n');
49   return NO_ERROR;
50 }
```

5.4.2.5 set_serial_in()

```
int set_serial_in (
              int device )
```

Definition at line 86 of file serial.c.

```
87 {
88   serial_port_in = device;
89   return NO_ERROR;
90 }
```

5.4.2.6 set_serial_out()

```
int set_serial_out (
            int device )
```

Definition at line 74 of file serial.c.

```
75 {
76   serial_port_out = device;
77   return NO_ERROR;
78 }
```

5.5 include/core/tables.h File Reference

```
#include "system.h"
```

Classes

- struct idt_entry_struct
- struct idt_struct
- struct gdt_descriptor_struct
- struct gdt_entry_struct

Functions

- struct idt_entry_struct __attribute__ ((packed)) idt_entry
- void idt_set_gate (u8int idx, u32int base, u16int sel, u8int flags)
- void gdt_init_entry (int idx, u32int base, u32int limit, u8int access, u8int flags)
- void init_idt ()
- void init_gdt ()

Variables

- u16int base_low
- u16int sselect
- u8int zero
- u8int flags
- u16int base_high
- u16int limit
- u32int base
- u16int limit_low
- u8int base_mid
- u8int access

5.5.1 Function Documentation

5.5.1.1 __attribute__()

```
struct gdt_entry_struct __attribute__ (
             (packed)  )
```

5.5.1.2 gdt_init_entry()

```
void gdt_init_entry (
             int idx,
             u32int base,
             u32int limit,
             u8int access,
             u8int flags )
```

Definition at line 57 of file tables.c.

```
59 {
60   gdt_entry *new_entry = &gdt_entries[idx];
61   new_entry->base_low  = (base & 0xFFFF);
62   new_entry->base_mid  = (base » 16) & 0xFF;
63   new_entry->base_high = (base » 24) & 0xFF;
64   new_entry->limit_low = (limit & 0xFFFF);
65   new_entry->flags  = (limit » 16) & 0xFF;
66   new_entry->flags |= flags & 0xF0;
67   new_entry->access = access;
68 }
```

5.5.1.3 idt_set_gate()

```
void idt_set_gate (
             u8int idx,
             u32int base,
             u16int sel,
             u8int flags )
```

Definition at line 27 of file tables.c.

```
29 {
30   idt_entry *new_entry = &idt_entries[idx];
31   new_entry->base_low  = (base &  0xFFFF);
32   new_entry->base_high = (base » 16) & 0xFFFF;
33   new_entry->sselect   = sel;
34   new_entry->zero = 0;
35   new_entry->flags = flags;
36 }
```

5.5.1.4 init_gdt()

```
void init_gdt ( )
```

Definition at line 75 of file tables.c.

```
76 {
77   gdt_ptr.limit = 5 * sizeof(gdt_entry) - 1;
78   gdt_ptr.base  = (u32int) gdt_entries;
79
80   u32int limit = 0xFFFFFFFF;
81   gdt_init_entry(0, 0, 0, 0, 0);           //required null segment
82   gdt_init_entry(1, 0, limit, 0x9A, 0xCF); //code segment
83   gdt_init_entry(2, 0, limit, 0x92, 0xCF); //data segment
84   gdt_init_entry(3, 0, limit, 0xFA, 0xCF); //user mode code segment
85   gdt_init_entry(4, 0, limit, 0xF2, 0xCF); //user mode data segment
86
87   write_gdt_ptr((u32int) &gdt_ptr, sizeof(gdt_ptr));
88 }
```

5.5.1.5 init_idt()

```
void init_idt ( )
```

Definition at line 43 of file tables.c.

```
44 {
45   idt_ptr.limit = 256*sizeof(idt_descriptor) - 1;
46   idt_ptr.base  = (u32int)idt_entries;
47   memset(idt_entries, 0, 256*sizeof(idt_descriptor));
48
49   write_idt_ptr((u32int)&idt_ptr);
50 }
```

5.5.2 Variable Documentation

5.5.2.1 access

```
u8int access
```

Definition at line 3 of file tables.h.

5.5.2.2 base

```
u32int base
```

Definition at line 1 of file tables.h.

5.5.2.3 base_high

```
u8int base_high
```

Definition at line 4 of file tables.h.

5.5.2.4 base_low

```
u16int base_low
```

Definition at line 0 of file tables.h.

5.5.2.5 base_mid

```
u8int base_mid
```

Definition at line 2 of file tables.h.

5.5.2.6 flags

```
u8int flags
```

Definition at line 3 of file tables.h.

5.5.2.7 limit

```
u16int limit
```

Definition at line 0 of file tables.h.

5.5.2.8 limit_low

```
u16int limit_low
```

Definition at line 0 of file tables.h.

5.5.2.9 sselect

```
u16int sselect
```

Definition at line 1 of file tables.h.

5.5.2.10 zero

```
u8int zero
```

Definition at line 2 of file tables.h.

5.6 include/mem/heap.h File Reference

Classes

- struct header
- struct footer
- struct index_entry
- struct index_table
- struct heap

Macros

- #define TABLE_SIZE 0x1000
- #define KHEAP_BASE 0xD000000
- #define KHEAP_MIN 0x10000
- #define KHEAP_SIZE 0x1000000

Functions

- u32int _kmalloc (u32int size, int align, u32int *phys_addr)
- u32int kmalloc (u32int size)
- u32int kfree ()
- void init_kheap ()
- u32int alloc (u32int size, heap *hp, int align)
- heap * make_heap (u32int base, u32int max, u32int min)

5.6.1 Macro Definition Documentation

5.6.1.1 KHEAP_BASE

```
#define KHEAP_BASE 0xD000000
```

Definition at line 6 of file heap.h.

5.6.1.2 KHEAP_MIN

```
#define KHEAP_MIN 0x10000
```

Definition at line 7 of file heap.h.

5.6.1.3 KHEAP_SIZE

```
#define KHEAP_SIZE 0x1000000
```

Definition at line 8 of file heap.h.

5.6.1.4 TABLE_SIZE

```
#define TABLE_SIZE 0x1000
```

Definition at line 5 of file heap.h.

5.6.2 Function Documentation

5.6.2.1 _kmalloc()

```
u32int _kmalloc (
            u32int size,
            int align,
            u32int * phys_addr )
```

Definition at line 24 of file heap.c.

```
25 {
26   u32int *addr;
27
28   // Allocate on the kernel heap if one has been created
29   if (kheap != 0){
30     addr = (u32int*)alloc(size, kheap, page_align);
31     if (phys_addr){
32       page_entry *page = get_page((u32int)addr, kdir, 0);
33       *phys_addr = (page->frameaddr*0x1000) + ((u32int)addr & 0xFFF);
34     }
35     return (u32int)addr;
36   }
37   // Else, allocate directly from physical memory
38   else {
39     if (page_align && (phys_alloc_addr & 0xFFFFF000)){
40       phys_alloc_addr &= 0xFFFFF000;
41       phys_alloc_addr += 0x1000;
42     }
43     addr = (u32int*)phys_alloc_addr;
44     if (phys_addr){
45       *phys_addr = phys_alloc_addr;
46     }
47     phys_alloc_addr += size;
48     return (u32int)addr;
49   }
50 }
```

5.6.2.2 alloc()

```
u32int alloc (
              u32int size,
              heap * hp,
              int align )
```

Definition at line 57 of file heap.c.

```
58 {
59   no_warn(size||align||h);
60   static u32int heap_addr = KHEAP_BASE;
61
62   u32int base = heap_addr;
63   heap_addr += size;
64
65   if (heap_addr > KHEAP_BASE + KHEAP_MIN)
66     serial_println("Heap is full!");
67
68   return base;
69 }
```

5.6.2.3 init_kheap()

```
void init_kheap ( )
```

5.6.2.4 kfree()

```
u32int kfree ( )
```

5.6.2.5 kmalloc()

```
u32int kmalloc (
              u32int size )
```

Definition at line 52 of file heap.c.

```
53 {
54   return _kmalloc(size,0,0);
55 }
```

5.6.2.6 make_heap()

```
heap* make_heap (
              u32int base,
              u32int max,
              u32int min )
```

Definition at line 71 of file heap.c.

```
72 {
73   no_warn(base||max||min);
74   return (heap*)kmalloc(sizeof(heap));
75 }
```

5.7 include/mem/paging.h File Reference

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

Classes

- struct page_entry
- struct page_table
- struct page_dir

Macros

- #define PAGE_SIZE 0x1000

Functions

- void set_bit (u32int addr)
- void clear_bit (u32int addr)
- u32int get_bit (u32int addr)
- u32int first_free ()
- void init_paging ()
- void load_page_dir (page_dir *new_page_dir)
- page_entry * get_page (u32int addr, page_dir *dir, int make_table)
- void new_frame (page_entry *page)

5.7.1 Macro Definition Documentation

5.7.1.1 PAGE_SIZE

```
#define PAGE_SIZE 0x1000
```

Definition at line 6 of file paging.h.

5.7.2 Function Documentation

5.7.2.1 clear_bit()

```
void clear_bit (
            u32int addr )
```

Definition at line 44 of file paging.c.

```
45 {
46   u32int frame  = addr/page_size;
47   u32int index  = frame/32;
48   u32int offset = frame%32;
49   frames[index] &= ~(1 « offset);
50 }
```

5.7.2.2 first_free()

```
u32int first_free ( )
```

5.7.2.3 get_bit()

```
u32int get_bit (
            u32int addr )
```

Definition at line 56 of file paging.c.

```
57 {
58   u32int frame  = addr/page_size;
59   u32int index  = frame/32;
60   u32int offset = frame%32;
61   return (frames[index] & (1 « offset));
62 }
```

5.7.2.4 get_page()

```
page_entry* get_page (
            u32int addr,
            page_dir * dir,
            int make_table )
```

Definition at line 85 of file paging.c.

```
86 {
87   u32int phys_addr;
88   u32int index = addr / page_size / 1024;
89   u32int offset = addr / page_size % 1024;
90
91   //return it if it exists
92   if (dir->tables[index])
93     return &dir->tables[index]->pages[offset];
94
95   //create it
96   else if (make_table){
97     dir->tables[index] = (page_table*)_kmalloc(sizeof(page_table), 1, &phys_addr);
98     dir->tables_phys[index] = phys_addr | 0x7; //enable present, writable
99     return &dir->tables[index]->pages[offset];
100   }
101   else return 0;
102 }
```

5.7.2.5 init_paging()

```
void init_paging ( )
```

Definition at line 111 of file paging.c.

```
112 {
113   //create frame bitmap
114   nframes = (u32int)(mem_size/page_size);
115   frames = (u32int*)kmalloc(nframes/32);
116   memset(frames, 0, nframes/32);
117 
118   //create kernel directory
119   kdir = (page_dir*)_kmalloc(sizeof(page_dir), 1, 0); //page aligned
120   memset(kdir, 0, sizeof(page_dir));
121 
122   //get pages for kernel heap
123   u32int i = 0x0;
124   for(i=KHEAP_BASE; i<(KHEAP_BASE+KHEAP_MIN); i+=1){
125     get_page(i,kdir,1);
126   }
127 
128   //perform identity mapping of used memory
129   //note: placement_addr gets incremented in get_page,
130   //so we're mapping the first frames as well
131   i = 0x0;
132   while (i < (phys_alloc_addr+0x10000)){
133     new_frame(get_page(i,kdir,1));
134     i += page_size;
135   }
136 
137   //allocate heap frames now that the placement addr has increased.
138   //placement addr increases here for heap
139   for(i=KHEAP_BASE; i<(KHEAP_BASE+KHEAP_MIN);i+=PAGE_SIZE){
140     new_frame(get_page(i,kdir,1));
141   }
142 
143   //load the kernel page directory; enable paging
144   load_page_dir(kdir);
145 
146   //setup the kernel heap
147   kheap = make_heap(KHEAP_BASE, KHEAP_SIZE, KHEAP_BASE+KHEAP_MIN);
148 }
```

5.7.2.6 load_page_dir()

```
void load_page_dir (
              page_dir * new_page_dir )
```

Definition at line 158 of file paging.c.

```
159 {
160   cdir = new_dir;
161   asm volatile ("mov %0,%%cr3":: "b"(&cdir->tables_phys[0]));
162   u32int cr0;
163   asm volatile ("mov %%cr0,%0": "=b"(cr0));
164   cr0 |= 0x80000000;
165   asm volatile ("mov %0,%%cr0":: "b"(cr0));
166 }
```

5.7.2.7 new_frame()

```
void new_frame (
              page_entry * page )
```

Definition at line 173 of file paging.c.

```
174 {
175   u32int index;
```

```
176   if (page->frameaddr != 0) return;
177   if ( (u32int)(-1) == (index=find_free()) ) kpanic("Out of memory");
178
179   //mark a frame as in-use
180   set_bit(index*page_size);
181   page->present   = 1;
182   page->frameaddr = index;
183   page->writeable = 1;
184   page->usermode  = 0;
185 }
```

5.7.2.8 set_bit()

```
void set_bit (
              u32int addr )
```

Definition at line 32 of file paging.c.

```
33 {
34   u32int frame  = addr/page_size;
35   u32int index  = frame/32;
36   u32int offset = frame%32;
37   frames[index] |= (1 « offset);
38 }
```

5.8 include/string.h File Reference

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

Functions

- int isspace (const char *c)
- void * memset (void *s, int c, size_t n)
- char * strcpy (char *s1, const char *s2)
- char * strcat (char *s1, const char *s2)
- int strlen (const char *s)
- int strcmp (const char *s1, const char *s2)
- char * strtok (char *s1, const char *s2)
- int atoi (const char *s)

5.8.1 Function Documentation

5.8.1.1 atoi()

```
int atoi (
            const char * s )
```

Definition at line 48 of file string.c.

```
49 {
50   int res=0;
51    int charVal=0;
52    char sign = ' ';
53    char c = *s;
54
55
56    while(isspace(&c)){ ++s; c = *s;} // advance past whitespace
57
58
59    if (*s == '-' || *s == '+') sign = *(s++); // save the sign
60
61
62    while(*s != '\0'){
63         charVal = *s - 48;
64     res = res * 10 + charVal;
65     s++;
66
67    }
68
69
70    if ( sign == '-') res=res * -1;
71
72   return res; // return integer
73 }
```

5.8.1.2 isspace()

```
int isspace (
            const char * c )
```

Definition at line 119 of file string.c.

```
120 {
121   if (*c == ' '  ||
122       *c == '\n' ||
123       *c == '\r' ||
124       *c == '\f' ||
125       *c == '\t' ||
126       *c == '\v'){
127     return 1;
128   }
129   return 0;
130 }
```

5.8.1.3 memset()

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

Definition at line 137 of file string.c.

```
138 {
139   unsigned char *p = (unsigned char *) s;
140   while(n--){
141     *p++ = (unsigned char) c;
142   }
143   return s;
144 }
```

5.8.1.4 strcat()

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

Definition at line 106 of file string.c.

```
107 {
108   char *rc = s1;
109   if (*s1) while(*++s1);
110   while( (*s1++ = *s2++) );
111   return rc;
112 }
```

5.8.1.5 strcmp()

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

Definition at line 79 of file string.c.

```
80 {
81
82   // Remarks:
83   // 1) If we made it to the end of both strings (i. e. our pointer points to a
84   //    '\0' character), the function will return 0
85   // 2) If we didn't make it to the end of both strings, the function will
86   //    return the difference of the characters at the first index of
87   //    indifference.
88   while ( (*s1) && (*s1==*s2) ){
89     ++s1;
90     ++s2;
91   }
92   return ( *(unsigned char *)s1 - *(unsigned char *)s2 );
93 }
```

5.8.1.6 strcpy()

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

Definition at line 36 of file string.c.

```
37 {
38   char *rc = s1;
39   while( (*s1++ = *s2++) );
40   return rc; // return pointer to destination string
41 }
```

5.8.1.7 strlen()

```
int strlen (
              const char * s )
```

Definition at line 24 of file string.c.

```
25 {
26   int r1 = 0;
27   if (*s) while(*s++) r1++;
28   return r1;//return length of string
29 }
```

5.8.1.8 strtok()

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

Definition at line 151 of file string.c.

```
152 {
153   static char *tok_tmp = NULL;
154   const char *p = s2;
155
156   //new string
157   if (s1!=NULL){
158     tok_tmp = s1;
159   }
160   //old string cont'd
161   else {
162     if (tok_tmp==NULL){
163       return NULL;
164     }
165     s1 = tok_tmp;
166   }
167
168   //skip leading s2 characters
169   while ( *p && *s1 ){
170     if (*s1==*p){
171       ++s1;
172       p = s2;
173       continue;
174     }
175     ++p;
176   }
177
178   //no more to parse
179   if (!*s1){
180     return (tok_tmp = NULL);
181   }
182
183   //skip non-s2 characters
184   tok_tmp = s1;
185   while (*tok_tmp){
186     p = s2;
187     while (*p){
188       if (*tok_tmp==*p++){
189     *tok_tmp++ = '\0';
190     return s1;
191       }
192     }
193     ++tok_tmp;
194   }
195
196   //end of string
197   tok_tmp = NULL;
198   return s1;
199 }
```

5.9 include/system.h File Reference

Classes

- struct date_time

Macros

- #define NULL 0
- #define no_warn(p) if (p) while (1) break
- #define asm __asm__
- #define volatile __volatile__
- #define sti() asm volatile ("sti"::)

- #define cli() asm volatile ("cli"::)
- #define nop() asm volatile ("nop"::)
- #define hlt() asm volatile ("hlt"::)
- #define iret() asm volatile ("iret"::)
- #define GDT_CS_ID 0x01
- #define GDT_DS_ID 0x02

Typedefs

- typedef unsigned int size_t
- typedef unsigned char u8int
- typedef unsigned short u16int
- typedef unsigned long u32int

Functions

- void klogv (const char *msg)
- void kpanic (const char *msg)

5.9.1 Macro Definition Documentation

5.9.1.1 asm

```
#define asm __asm__
```

Definition at line 11 of file system.h.

5.9.1.2 cli

```
#define cli( ) asm volatile ("cli"::)
```

Definition at line 15 of file system.h.

5.9.1.3 GDT_CS_ID

```
#define GDT_CS_ID 0x01
```

Definition at line 20 of file system.h.

5.9.1.4 GDT_DS_ID

```
#define GDT_DS_ID 0x02
```

Definition at line 21 of file system.h.

5.9.1.5 hlt

```
#define hlt( ) asm volatile ("hlt"::)
```

Definition at line 17 of file system.h.

5.9.1.6 iret

```
#define iret( ) asm volatile ("iret"::)
```

Definition at line 18 of file system.h.

5.9.1.7 no_warn

```
#define no_warn(
             p ) if (p) while (1) break
```

Definition at line 7 of file system.h.

5.9.1.8 nop

```
#define nop( ) asm volatile ("nop"::)
```

Definition at line 16 of file system.h.

5.9.1.9 NULL

```
#define NULL 0
```

Definition at line 4 of file system.h.

5.9.1.10 sti

```
#define sti( ) asm volatile ("sti"::)
```

Definition at line 14 of file system.h.

5.9.1.11 volatile

```
#define volatile __volatile__
```

Definition at line 12 of file system.h.

5.9.2 Typedef Documentation

5.9.2.1 size_t

```
typedef unsigned int size_t
```

Definition at line 24 of file system.h.

5.9.2.2 u16int

```
typedef unsigned short u16int
```

Definition at line 26 of file system.h.

5.9.2.3 u32int

```
typedef unsigned long u32int
```

Definition at line 27 of file system.h.

5.9.2.4 u8int

```
typedef unsigned char u8int
```

Definition at line 25 of file system.h.

5.9.3 Function Documentation

5.9.3.1 klogv()

```
void klogv (
            const char * msg )
```

Definition at line 11 of file system.c.

```
12 {
13   char logmsg[64] = {'\0'}, prefix[] = "klogv: ";
14   strcat(logmsg, prefix);
15   strcat(logmsg, msg);
16   serial_println(logmsg);
17 }
```

5.9.3.2 kpanic()

```
void kpanic (
            const char * msg )
```

Definition at line 24 of file system.c.

```
25 {
26   cli(); //disable interrupts
27   char logmsg[64] = {'\0'}, prefix[] = "Panic: ";
28   strcat(logmsg, prefix);
29   strcat(logmsg, msg);
30   klogv(logmsg);
31   hlt(); //halt
32 }
```

5.10 kernel/core/interrupts.c File Reference

```
#include <system.h>
#include <core/io.h>
#include <core/serial.h>
#include <core/tables.h>
#include <core/interrupts.h>
```

Macros

- #define PIC1 0x20
- #define PIC2 0xA0
- #define ICW1 0x11
- #define ICW4 0x01
- #define io_wait() asm volatile("outb $0x80")

Functions

- void divide_error ()
- void debug ()
- void nmi ()
- void breakpoint ()
- void overflow ()
- void bounds ()
- void invalid_op ()
- void device_not_available ()
- void double_fault ()
- void coprocessor_segment ()
- void invalid_tss ()
- void segment_not_present ()
- void stack_segment ()
- void general_protection ()
- void page_fault ()
- void reserved ()
- void coprocessor ()
- void rtc_isr ()
- void sys_call_isr ()
- void isr0 ()
- void do_isr ()
- void init_irq (void)
- void init_pic (void)
- void do_divide_error ()
- void do_debug ()
- void do_nmi ()
- void do_breakpoint ()
- void do_overflow ()
- void do_bounds ()
- void do_invalid_op ()
- void do_device_not_available ()
- void do_double_fault ()
- void do_coprocessor_segment ()
- void do_invalid_tss ()
- void do_segment_not_present ()
- void do_stack_segment ()
- void do_general_protection ()
- void do_page_fault ()
- void do_reserved ()
- void do_coprocessor ()

Variables

- idt_entry idt_entries [256]

5.10.1 Macro Definition Documentation

5.10.1.1 ICW1

```
#define ICW1 0x11
```

Definition at line 20 of file interrupts.c.

5.10.1.2 ICW4

```
#define ICW4 0x01
```

Definition at line 21 of file interrupts.c.

5.10.1.3 io_wait

```
#define io_wait( ) asm volatile("outb $0x80")
```

Definition at line 28 of file interrupts.c.

5.10.1.4 PIC1

```
#define PIC1 0x20
```

Definition at line 16 of file interrupts.c.

5.10.1.5 PIC2

```
#define PIC2 0xA0
```

Definition at line 17 of file interrupts.c.

5.10.2 Function Documentation

5.10.2.1 bounds()

```
void bounds ( )
```

5.10.2.2 breakpoint()

```
void breakpoint ( )
```

5.10.2.3 coprocessor()

```
void coprocessor ( )
```

5.10.2.4 coprocessor_segment()

```
void coprocessor_segment ( )
```

5.10.2.5 debug()

```
void debug ( )
```

5.10.2.6 device_not_available()

```
void device_not_available ( )
```

5.10.2.7 divide_error()

```
void divide_error ( )
```

5.10.2.8 do_bounds()

```
void do_bounds ( )
```

Definition at line 153 of file interrupts.c.

```
154 {
155   kpanic("Bounds error");
156 }
```

5.10.2.9 do_breakpoint()

```
void do_breakpoint ( )
```

Definition at line 145 of file interrupts.c.

```
146 {
147   kpanic("Breakpoint");
148 }
```

5.10.2.10 do_coprocessor()

```
void do_coprocessor ( )
```

Definition at line 197 of file interrupts.c.

```
198 {
199   kpanic("Coprocessor error");
200 }
```

5.10.2.11 do_coprocessor_segment()

```
void do_coprocessor_segment ( )
```

Definition at line 169 of file interrupts.c.

```
170 {
171   kpanic("Coprocessor segment error");
172 }
```

5.10.2.12 do_debug()

```
void do_debug ( )
```

Definition at line 137 of file interrupts.c.

```
138 {
139   kpanic("Debug");
140 }
```

5.10.2.13 do_device_not_available()

```
void do_device_not_available ( )
```

Definition at line 161 of file interrupts.c.

```
162 {
163   kpanic("Device not available");
164 }
```

5.10.2.14 do_divide_error()

```
void do_divide_error ( )
```

Definition at line 133 of file interrupts.c.

```
134 {
135   kpanic("Division-by-zero");
136 }
```

5.10.2.15 do_double_fault()

```
void do_double_fault ( )
```

Definition at line 165 of file interrupts.c.

```
166 {
167   kpanic("Double fault");
168 }
```

5.10.2.16 do_general_protection()

```
void do_general_protection ( )
```

Definition at line 185 of file interrupts.c.

```
186 {
187   kpanic("General protection fault");
188 }
```

5.10.2.17 do_invalid_op()

```
void do_invalid_op ( )
```

Definition at line 157 of file interrupts.c.

```
158 {
159   kpanic("Invalid operation");
160 }
```

5.10.2.18 do_invalid_tss()

```
void do_invalid_tss ( )
```

Definition at line 173 of file interrupts.c.

```
174 {
175   kpanic("Invalid TSS");
176 }
```

5.10.2.19 do_isr()

```
void do_isr ( )
```

Definition at line 54 of file interrupts.c.

```
55 {
56   char in = inb(COM2);
57   serial_print(&in);
58   serial_println("here");
59   outb(0x20, 0x20); //EOI
60 }
```

5.10.2.20 do_nmi()

```
void do_nmi ( )
```

Definition at line 141 of file interrupts.c.

```
142 {
143   kpanic("NMI");
144 }
```

5.10.2.21 do_overflow()

```
void do_overflow ( )
```

Definition at line 149 of file interrupts.c.

```
150 {
151   kpanic("Overflow error");
152 }
```

5.10.2.22 do_page_fault()

```
void do_page_fault ( )
```

Definition at line 189 of file interrupts.c.

```
190 {
191   kpanic("Page Fault");
192 }
```

5.10.2.23 do_reserved()

```
void do_reserved ( )
```

Definition at line 193 of file interrupts.c.

```
194 {
195   serial_println("die: reserved");
196 }
```

5.10.2.24 do_segment_not_present()

```
void do_segment_not_present ( )
```

Definition at line 177 of file interrupts.c.

```
178 {
179   kpanic("Segment not present");
180 }
```

5.10.2.25 do_stack_segment()

```
void do_stack_segment ( )
```

Definition at line 181 of file interrupts.c.

```
182 {
183   kpanic("Stack segment error");
184 }
```

5.10.2.26 double_fault()

```
void double_fault ( )
```

5.10.2.27 general_protection()

```
void general_protection ( )
```

5.10.2.28 init_irq()

```
void init_irq (
              void  )
```

Definition at line 67 of file interrupts.c.

```
68 {
69   int i;
70
71   // Necessary interrupt handlers for protected mode
72   u32int isrs[17] = {
73       (u32int)divide_error,
74       (u32int)debug,
75       (u32int)nmi,
76       (u32int)breakpoint,
77       (u32int)overflow,
78       (u32int)bounds,
79       (u32int)invalid_op,
80       (u32int)device_not_available,
81       (u32int)double_fault,
82       (u32int)coprocessor_segment,
83       (u32int)invalid_tss,
84       (u32int)segment_not_present,
85       (u32int)stack_segment,
86       (u32int)general_protection,
87       (u32int)page_fault,
```

```
88        (u32int)reserved,
89        (u32int)coprocessor};
90
91    // Install handlers; 0x08=sel, 0x8e=flags
92    for (i = 0; i < 32; i++)
93    {
94      if (i < 17)
95        idt_set_gate(i, isrs[i], 0x08, 0x8e);
96      else
97        idt_set_gate(i, (u32int)reserved, 0x08, 0x8e);
98    }
99    // Ignore interrupts from the real time clock
100    idt_set_gate(0x08, (u32int)rtc_isr, 0x08, 0x8e);
101    idt_set_gate(60, (u32int)sys_call_isr, 0x08, 0x8e);
102 }
```

5.10.2.29 init_pic()

```
void init_pic (
              void  )
```

Definition at line 110 of file interrupts.c.

```
111 {
112    outb(PIC1, ICW1); //send initialization code words 1 to PIC1
113    io_wait();
114    outb(PIC2, ICW1); //send icw1 to PIC2
115    io_wait();
116    outb(PIC1 + 1, 0x20); //icw2: remap irq0 to 32
117    io_wait();
118    outb(PIC2 + 1, 0x28); //icw2: remap irq8 to 40
119    io_wait();
120    outb(PIC1 + 1, 4); //icw3
121    io_wait();
122    outb(PIC2 + 1, 2); //icw3
123    io_wait();
124    outb(PIC1 + 1, ICW4); //icw4: 80x86, automatic handling
125    io_wait();
126    outb(PIC2 + 1, ICW4); //icw4: 80x86, automatic handling
127    io_wait();
128    outb(PIC1 + 1, 0xFF); //disable irqs for PIC1
129    io_wait();
130    outb(PIC2 + 1, 0xFF); //disable irqs for PIC2
131 }
```

5.10.2.30 invalid_op()

```
void invalid_op ( )
```

5.10.2.31 invalid_tss()

```
void invalid_tss ( )
```

5.10.2.32 isr0()

```
void isr0 ( )
```

5.10.2.33 nmi()

```
void nmi ( )
```

5.10.2.34 overflow()

```
void overflow ( )
```

5.10.2.35 page_fault()

```
void page_fault ( )
```

5.10.2.36 reserved()

```
void reserved ( )
```

5.10.2.37 rtc_isr()

```
void rtc_isr ( )
```

5.10.2.38 segment_not_present()

```
void segment_not_present ( )
```

5.10.2.39 stack_segment()

```
void stack_segment ( )
```

5.10.2.40 sys_call_isr()

```
void sys_call_isr ( )
```

5.10.3 Variable Documentation

5.10.3.1 idt_entries

```
idt_entry idt_entries[256]  [extern]
```

Definition at line 17 of file tables.c.

5.11 kernel/core/kmain.c File Reference

```
#include <stdint.h>
#include <string.h>
#include <system.h>
#include <core/io.h>
#include <core/serial.h>
#include <core/tables.h>
#include <core/interrupts.h>
#include <mem/heap.h>
#include <mem/paging.h>
#include "modules/mpx_supt.h"
#include "modules/R1/commhand.h"
#include "modules/R2/R2commands.h"
#include "modules/R2/R2_Internal_Functions_And_Structures.h"
#include "modules/R3/R3commands.h"
#include "modules/R4/R4commands.h"
#include "modules/R5/R5commands.h"
```

Functions

- void kmain (void)

5.11.1 Function Documentation

5.11.1.1 kmain()

```
void kmain (
             void  )
```

Definition at line 32 of file kmain.c.

```
33 {
34    // extern uint32_t magic;
35    // Uncomment if you want to access the multiboot header
36    // extern void *mbd;
37    // char *boot_loader_name = (char*)((long*)mbd)[16];
38
39    // 0) Initialize Serial I/O
40    // functions to initialize serial I/O can be found in serial.c
41    // there are 3 functions to call
42
43    init_serial(COM1);
44    set_serial_in(COM1);
45    set_serial_out(COM1);
46
47    klogv("Starting MPX boot sequence...");
48    klogv("Initialized serial I/O on COM1 device...");
49
50    // 1) Initialize the support software by identifying the current
51    //     MPX Module.  This will change with each module.
52    // you will need to call mpx_init from the mpx_supt.c
53
54    mpx_init(MODULE_R5);
55
56    // 2) Check that the boot was successful and correct when using grub
57    // Comment this when booting the kernel directly using QEMU, etc.
58    //if ( magic != 0x2BADB002 ){
59    //  kpanic("Boot was not error free. Halting.");
60    //}
61
62    // 3) Descriptor Tables -- tables.c
63    //  you will need to initialize the global
64    // this keeps track of allocated segments and pages
65    klogv("Initializing descriptor tables...");
66
67    init_gdt();
68    init_idt();
69
70    init_pic();
71    sti();
72
73    // 4)  Interrupt vector table --  tables.c
74    // this creates and initializes a default interrupt vector table
75    // this function is in tables.c
76
77    init_irq();
78
79    klogv("Interrupt vector table initialized!");
80
81    // 5) Virtual Memory -- paging.c  -- init_paging
82    //  this function creates the kernel's heap
83    //  from which memory will be allocated when the program calls
84    // sys_alloc_mem UNTIL the memory management module  is completed
85    // this allocates memory using discrete "pages" of physical memory
86    // NOTE:  You will only have about 70000 bytes of dynamic memory
87    //
88    klogv("Initializing virtual memory...");
89
90    init_paging();
91
92    // 6) Call YOUR command handler -  interface method
93    klogv("Transferring control to commhand...");
94    //commhand(); //Removed for R4
95
96    // allocateMemLists();
97    //allocateAlarms();
98
99    initializeHeap((u32int)50000);
100    mpx_init(MEM_MODULE);
101    sys_set_malloc((allocateMemory));
102    sys_set_free((freeMemory));
103    allocateQueues();
104
105    createPCB("Commhand", 's', 9);
106    PCB *new_pcb = findPCB("Commhand");
107    context *cp = (context *)(new_pcb->stackTop);
108    memset(cp, 0, sizeof(context));
109    cp->fs = 0x10;
110    cp->gs = 0x10;
```

```
111     cp->ds = 0x10;
112     cp->es = 0x10;
113     cp->cs = 0x8;
114     cp->ebp = (u32int)(new_pcb->stack);
115     cp->esp = (u32int)(new_pcb->stackTop);
116     cp->eip = (u32int)commhand; // The function correlating to the process, ie. Proc1
117     cp->eflags = 0x202;
118 
119     // createPCB("Alarm", 'a', 1);
120     // PCB *AlarmPCB = findPCB("Alarm");
121     // context *cpAlarm = (context *)(AlarmPCB->stackTop);
122     // memset(cpAlarm, 0, sizeof(context));
123     // cpAlarm->fs = 0x10;
124     // cpAlarm->gs = 0x10;
125     // cpAlarm->ds = 0x10;
126     // cpAlarm->es = 0x10;
127     // cpAlarm->cs = 0x8;
128     // cpAlarm->ebp = (u32int)(AlarmPCB->stack);
129     // cpAlarm->esp = (u32int)(AlarmPCB->stackTop);
130     // cpAlarm->eip = (u32int)alarmPCB; // The function correlating to the process, ie. Proc1
131     // cpAlarm->eflags = 0x202;
132 
133     createPCB("Idle", 's', 0);
134     PCB *idlePCB = findPCB("Idle");
135     context *cpIDLE = (context *)(idlePCB->stackTop);
136     memset(cpIDLE, 0, sizeof(context));
137     cpIDLE->fs = 0x10;
138     cpIDLE->gs = 0x10;
139     cpIDLE->ds = 0x10;
140     cpIDLE->es = 0x10;
141     cpIDLE->cs = 0x8;
142     cpIDLE->ebp = (u32int)(idlePCB->stack);
143     cpIDLE->esp = (u32int)(idlePCB->stackTop);
144     cpIDLE->eip = (u32int)idle; // The function correlating to the process, ie. Proc1
145     cpIDLE->eflags = 0x202;
146 
147     asm volatile("int $60");
148 
149     // 7) System Shutdown on return from your command handler
150 
151     klogv("Starting system shutdown procedure...");
152 
153     /* Shutdown Procedure */
154     klogv("Shutdown complete. You may now turn off the machine. (QEMU: C-a x)");
155     hlt();
156 }
```

5.12 kernel/core/serial.c File Reference

```
#include <stdint.h>
#include <string.h>
#include <core/io.h>
#include <core/serial.h>
```

Macros

- #define NO_ERROR 0

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)
- int * polling (char *buffer, int *count)

Variables

- int serial_port_out = 0
- int serial_port_in = 0

5.12.1 Macro Definition Documentation

5.12.1.1 NO_ERROR

```
#define NO_ERROR 0
```

Definition at line 12 of file serial.c.

5.12.2 Function Documentation

5.12.2.1 init_serial()

```
int init_serial (
              int device )
```

Definition at line 22 of file serial.c.

```
23 {
24   outb(device + 1, 0x00);          //disable interrupts
25   outb(device + 3, 0x80);          //set line control register
26   outb(device + 0, 115200 / 9600); //set bsd least sig bit
27   outb(device + 1, 0x00);          //brd most significant bit
28   outb(device + 3, 0x03);          //lock divisor; 8bits, no parity, one stop
29   outb(device + 2, 0xC7);          //enable fifo, clear, 14byte threshold
30   outb(device + 4, 0x0B);          //enable interrupts, rts/dsr set
31   (void)inb(device);               //read bit to reset port
32   return NO_ERROR;
33 }
```

5.12.2.2 polling()

```
int* polling (
              char * buffer,
              int * count )
```

Definition at line 92 of file serial.c.

```
93 {
94   // insert your code to gather keyboard input via the technique of polling.
95
96   char keyboard_character;
97
98   int cursor = 0;
99
100   char log[] = {'\0', '\0', '\0', '\0'};
101
102   int characters_in_buffer = 0;
103
```

```
104   while (1)
105   {
106
107     if (inb(COM1 + 5) & 1)
108     {                                   // is there input char?
109       keyboard_character = inb(COM1); //read the char from COM1
110
111       if (keyboard_character == '\n' || keyboard_character == '\r')
112       { // HANDLEING THE CARRIAGE RETURN AND NEW LINE CHARACTERS
113
114         buffer[characters_in_buffer] = '\0';
115         break;
116       }
117       else if ((keyboard_character == 127 || keyboard_character == 8) && cursor > 0)
118       { // HANDELING THE BACKSPACE CHARACTER
119
120         //serial_println("Handleing backspace character.");
121         serial_print("\033[K");
122
123         buffer[cursor - 1] = '\0';
124         serial_print("\b \b");
125         serial_print(buffer + cursor);
126         cursor--;
127
128         int temp_cursor = cursor;
129
130         while (buffer[temp_cursor + 1] != '\0')
131         {
132           buffer[temp_cursor] = buffer[temp_cursor + 1];
133           buffer[temp_cursor + 1] = '\0';
134           temp_cursor++;
135         }
136
137         characters_in_buffer--;
138         cursor = characters_in_buffer;
139       }
140       else if (keyboard_character == '~' && cursor < 99)
141       { //HANDLING THE DELETE KEY
142         // \033[3~
143
144         serial_print("\033[K");
145
146         buffer[cursor + 1] = '\0';
147         serial_print("\b \b");
148         serial_print(buffer + cursor);
149
150         int temp_cursor = cursor + 1;
151
152         while (buffer[temp_cursor + 1] != '\0')
153         {
154           buffer[temp_cursor] = buffer[temp_cursor + 1];
155           buffer[temp_cursor + 1] = '\0';
156           temp_cursor++;
157         }
158
159         characters_in_buffer--;
160         cursor = characters_in_buffer;
161       }
162       else if (keyboard_character == '\033')
163       { // HANDLEING FIRST CHARACTER FOR ARROW KEYS
164
165         log[0] = keyboard_character;
166       }
167       else if (keyboard_character == '[' && log[0] == '\033')
168       { // HANDLEING SECOND CHARACTER FOR ARROW KEYS
169
170         log[1] = keyboard_character;
171       }
172       else if (log[0] == '\033' && log[1] == '[')
173       { // HANDLEING LAST CHARACTER FOR ARROW KEYS
174         log[2] = keyboard_character;
175
176         if (keyboard_character == 'A')
177         { //Up arrow
178           //Call a history function from the commhand or do nothing
179         }
180         else if (keyboard_character == 'B')
181         { //Down arrow
182           //Call a history command from the commhand or do nothing
183         }
184         else if (keyboard_character == 'C' && cursor != 99)
185         { //Right arrow
186
187           serial_print("\033[C");
188           cursor++;
189         }
190         else if (keyboard_character == 'D' && cursor != 0)
```

```
191         { //Left arrow
192
193           serial_print("\033[D");
194           cursor--;
195         }
196
197         memset(log, '\0', 4);
198       }
199       else
200       {
201
202         if (cursor == 0 && buffer[cursor] == '\0') //Adding character at beginning of buffer
203         {
204           buffer[cursor] = keyboard_character;
205           serial_print(buffer + cursor);
206           cursor++;
207         }
208         else if (buffer[cursor] == '\0') //Adding character at the end of the buffer
209         {
210           buffer[cursor] = keyboard_character;
211           serial_print(buffer + cursor);
212           cursor++;
213         }
214         else //Inserting character to the middle of the buffer
215         {
216           char temp_buffer[strlen(buffer)];
217           memset(temp_buffer, '\0', strlen(buffer));
218
219           int temp_cursor = 0;
220           while (temp_cursor <= characters_in_buffer) //Filling the temp_buffer with all of the
      characters from buffer, and inserting the new character.
221           {
222             if (temp_cursor < cursor)
223             {
224               temp_buffer[temp_cursor] = buffer[temp_cursor];
225             }
226             else if (temp_cursor > cursor)
227             {
228               temp_buffer[temp_cursor] = buffer[temp_cursor - 1];
229             }
230             else
231             { //temp_cursor == cursor
232               temp_buffer[temp_cursor] = keyboard_character;
233             }
234             temp_cursor++;
235           }
236
237           temp_cursor = 0;
238           int temp_buffer_size = strlen(temp_buffer);
239           while (temp_cursor <= temp_buffer_size) //Setting the contents of the buffer equal to the
      temp_buffer.
240           {
241             buffer[temp_cursor] = temp_buffer[temp_cursor];
242             temp_cursor++;
243           }
244
245           serial_print("\033[K");
246           serial_print(&keyboard_character);
247           serial_print(buffer + cursor + 1);
248           cursor++;
249         }
250         characters_in_buffer++;
251       }
252     }
253   }
254
255   *count = characters_in_buffer; // buffer count
256
257   return count;
258 }
```

5.12.2.3 serial_print()

```
int serial_print (
            const char * msg )
```

Definition at line 56 of file serial.c.

```
57 {
```

```
58   int i;
59   for (i = 0; *(i + msg) != '\0'; i++)
60   {
61     outb(serial_port_out, *(i + msg));
62   }
63   if (*msg == '\r')
64     outb(serial_port_out, '\n');
65   return NO_ERROR;
66 }
```

5.12.2.4 serial_println()

```
int serial_println (
              const char * msg )
```

Definition at line 40 of file serial.c.

```
41 {
42   int i;
43   for (i = 0; *(i + msg) != '\0'; i++)
44   {
45     outb(serial_port_out, *(i + msg));
46   }
47   outb(serial_port_out, '\r');
48   outb(serial_port_out, '\n');
49   return NO_ERROR;
50 }
```

5.12.2.5 set_serial_in()

```
int set_serial_in (
              int device )
```

Definition at line 86 of file serial.c.

```
87 {
88   serial_port_in = device;
89   return NO_ERROR;
90 }
```

5.12.2.6 set_serial_out()

```
int set_serial_out (
              int device )
```

Definition at line 74 of file serial.c.

```
75 {
76   serial_port_out = device;
77   return NO_ERROR;
78 }
```

5.12.3 Variable Documentation

5.12.3.1 serial_port_in

```
int serial_port_in = 0
```

Definition at line 16 of file serial.c.

5.12.3.2 serial_port_out

```
int serial_port_out = 0
```

Definition at line 15 of file serial.c.

5.13 kernel/core/system.c File Reference

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

Functions

- void klogv (const char *msg)
- void kpanic (const char *msg)

5.13.1 Function Documentation

5.13.1.1 klogv()

```
void klogv (
            const char * msg )
```

Definition at line 11 of file system.c.

```
12 {
13   char logmsg[64] = {'\0'}, prefix[] = "klogv: ";
14   strcat(logmsg, prefix);
15   strcat(logmsg, msg);
16   serial_println(logmsg);
17 }
```

5.13.1.2 kpanic()

```
void kpanic (
            const char * msg )
```

Definition at line 24 of file system.c.

```
25 {
26   cli(); //disable interrupts
27   char logmsg[64] = {'\0'}, prefix[] = "Panic: ";
28   strcat(logmsg, prefix);
29   strcat(logmsg, msg);
30   klogv(logmsg);
31   hlt(); //halt
32 }
```

5.14 kernel/core/tables.c File Reference

```
#include <string.h>
#include <core/tables.h>
```

Functions

- void write_gdt_ptr (u32int, size_t)
- void write_idt_ptr (u32int)
- void idt_set_gate (u8int idx, u32int base, u16int sel, u8int flags)
- void init_idt ()
- void gdt_init_entry (int idx, u32int base, u32int limit, u8int access, u8int flags)
- void init_gdt ()

Variables

- gdt_descriptor gdt_ptr
- gdt_entry gdt_entries [5]
- idt_descriptor idt_ptr
- idt_entry idt_entries [256]

5.14.1 Function Documentation

5.14.1.1 gdt_init_entry()

```
void gdt_init_entry (
            int idx,
            u32int base,
            u32int limit,
            u8int access,
            u8int flags )
```

Definition at line 57 of file tables.c.

```
59 {
60   gdt_entry *new_entry = &gdt_entries[idx];
61   new_entry->base_low  = (base & 0xFFFF);
62   new_entry->base_mid  = (base » 16) & 0xFF;
63   new_entry->base_high = (base » 24) & 0xFF;
64   new_entry->limit_low = (limit & 0xFFFF);
65   new_entry->flags  = (limit » 16) & 0xFF;
66   new_entry->flags |= flags & 0xF0;
67   new_entry->access = access;
68 }
```

5.14.1.2 idt_set_gate()

```
void idt_set_gate (
              u8int idx,
              u32int base,
              u16int sel,
              u8int flags )
```

Definition at line 27 of file tables.c.

```
29 {
30   idt_entry *new_entry = &idt_entries[idx];
31   new_entry->base_low  = (base &  0xFFFF);
32   new_entry->base_high = (base » 16) & 0xFFFF;
33   new_entry->sselect   = sel;
34   new_entry->zero = 0;
35   new_entry->flags = flags;
36 }
```

5.14.1.3 init_gdt()

```
void init_gdt ( )
```

Definition at line 75 of file tables.c.

```
76 {
77   gdt_ptr.limit = 5 * sizeof(gdt_entry) - 1;
78   gdt_ptr.base  = (u32int) gdt_entries;
79
80   u32int limit = 0xFFFFFFFF;
81   gdt_init_entry(0, 0, 0, 0, 0);           //required null segment
82   gdt_init_entry(1, 0, limit, 0x9A, 0xCF); //code segment
83   gdt_init_entry(2, 0, limit, 0x92, 0xCF); //data segment
84   gdt_init_entry(3, 0, limit, 0xFA, 0xCF); //user mode code segment
85   gdt_init_entry(4, 0, limit, 0xF2, 0xCF); //user mode data segment
86
87   write_gdt_ptr((u32int) &gdt_ptr, sizeof(gdt_ptr));
88 }
```

5.14.1.4 init_idt()

```
void init_idt ( )
```

Definition at line 43 of file tables.c.

```
44 {
45   idt_ptr.limit = 256*sizeof(idt_descriptor) - 1;
46   idt_ptr.base  = (u32int)idt_entries;
47   memset(idt_entries, 0, 256*sizeof(idt_descriptor));
48
49   write_idt_ptr((u32int)&idt_ptr);
50 }
```

5.14.1.5 write_gdt_ptr()

```
void write_gdt_ptr (
              u32int ,
              size_t  )
```

5.14.1.6 write_idt_ptr()

```
void write_idt_ptr (
             u32int  )
```

5.14.2 Variable Documentation

5.14.2.1 gdt_entries

```
gdt_entry gdt_entries[5]
```

Definition at line 13 of file tables.c.

5.14.2.2 gdt_ptr

```
gdt_descriptor gdt_ptr
```

Definition at line 12 of file tables.c.

5.14.2.3 idt_entries

```
idt_entry idt_entries[256]
```

Definition at line 17 of file tables.c.

5.14.2.4 idt_ptr

```
idt_descriptor idt_ptr
```

Definition at line 16 of file tables.c.

5.15 kernel/mem/heap.c File Reference

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

Functions

- u32int _kmalloc (u32int size, int page_align, u32int *phys_addr)
- u32int kmalloc (u32int size)
- u32int alloc (u32int size, heap *h, int align)
- heap * make_heap (u32int base, u32int max, u32int min)

Variables

- heap * kheap = 0
- heap * curr_heap = 0
- page_dir * kdir
- void * end
- void _end
- void __end
- u32int phys_alloc_addr = (u32int)&end

5.15.1 Function Documentation

5.15.1.1 _kmalloc()

```
u32int _kmalloc (
              u32int size,
              int page_align,
              u32int * phys_addr )
```

Definition at line 24 of file heap.c.

```
25 {
26   u32int *addr;
27
28   // Allocate on the kernel heap if one has been created
29   if (kheap != 0){
30     addr = (u32int*)alloc(size, kheap, page_align);
31     if (phys_addr){
32       page_entry *page = get_page((u32int)addr, kdir, 0);
33       *phys_addr = (page->frameaddr*0x1000) + ((u32int)addr & 0xFFF);
34     }
35     return (u32int)addr;
36   }
37   // Else, allocate directly from physical memory
38   else {
39     if (page_align && (phys_alloc_addr & 0xFFFFF000)){
40       phys_alloc_addr &= 0xFFFFF000;
41       phys_alloc_addr += 0x1000;
42     }
43     addr = (u32int*)phys_alloc_addr;
44     if (phys_addr){
45       *phys_addr = phys_alloc_addr;
46     }
47     phys_alloc_addr += size;
48     return (u32int)addr;
49   }
50 }
```

5.15.1.2 alloc()

```
u32int alloc (
              u32int size,
              heap * h,
              int align )
```

Definition at line 57 of file heap.c.

```
58 {
59   no_warn(size||align||h);
60   static u32int heap_addr = KHEAP_BASE;
61
62   u32int base = heap_addr;
63   heap_addr += size;
64
65   if (heap_addr > KHEAP_BASE + KHEAP_MIN)
66     serial_println("Heap is full!");
67
68   return base;
69 }
```

5.15.1.3 kmalloc()

```
u32int kmalloc (
              u32int size )
```

Definition at line 52 of file heap.c.

```
53 {
54   return _kmalloc(size,0,0);
55 }
```

5.15.1.4 make_heap()

```
heap* make_heap (
              u32int base,
              u32int max,
              u32int min )
```

Definition at line 71 of file heap.c.

```
72 {
73   no_warn(base||max||min);
74   return (heap*)kmalloc(sizeof(heap));
75 }
```

5.15.2 Variable Documentation

5.15.2.1 __end

```
void __end
```

Definition at line 18 of file heap.c.

5.15.2.2 _end

```
void _end
```

Definition at line 18 of file heap.c.

5.15.2.3 curr_heap

```
heap* curr_heap = 0
```

Definition at line 15 of file heap.c.

5.15.2.4 end

```
void* end  [extern]
```

5.15.2.5 kdir

```
page_dir* kdir  [extern]
```

Definition at line 21 of file paging.c.

5.15.2.6 kheap

```
heap* kheap = 0
```

Definition at line 14 of file heap.c.

5.15.2.7 phys_alloc_addr

```
u32int phys_alloc_addr = (u32int)&end
```

Definition at line 22 of file heap.c.

5.16 kernel/mem/paging.c File Reference

```
#include <system.h>
#include <string.h>
#include "mem/heap.h"
#include "mem/paging.h"
```

Functions

- void set_bit (u32int addr)
- void clear_bit (u32int addr)
- u32int get_bit (u32int addr)
- u32int find_free ()
- page_entry * get_page (u32int addr, page_dir *dir, int make_table)
- void init_paging ()
- void load_page_dir (page_dir *new_dir)
- void new_frame (page_entry *page)

Variables

- u32int mem_size = 0x4000000
- u32int page_size = 0x1000
- u32int nframes
- u32int * frames
- page_dir * kdir = 0
- page_dir * cdir = 0
- u32int phys_alloc_addr
- heap * kheap

5.16.1 Function Documentation

5.16.1.1 clear_bit()

```
void clear_bit (
            u32int addr )
```

Definition at line 44 of file paging.c.

```
45 {
46   u32int frame  = addr/page_size;
47   u32int index  = frame/32;
48   u32int offset = frame%32;
49   frames[index] &= ~(1 << offset);
50 }
```

5.16.1.2 find_free()

```
u32int find_free ( )
```

Definition at line 68 of file paging.c.

```
69 {
70   u32int i,j;
71   for (i=0; i<nframes/32; i++)
72     if (frames[i] != 0xFFFFFFFF) //if frame not full
73       for (j=0; j<32; j++) //find first free bit
74     if (!(frames[i] & (1 « j)))
75       return i*32+j;
76
77   return -1; //no free frames
78 }
```

5.16.1.3 get_bit()

```
u32int get_bit (
              u32int addr )
```

Definition at line 56 of file paging.c.

```
57 {
58   u32int frame  = addr/page_size;
59   u32int index  = frame/32;
60   u32int offset = frame%32;
61   return (frames[index] & (1 « offset));
62 }
```

5.16.1.4 get_page()

```
page_entry* get_page (
              u32int addr,
              page_dir * dir,
              int make_table )
```

Definition at line 85 of file paging.c.

```
86 {
87   u32int phys_addr;
88   u32int index = addr / page_size / 1024;
89   u32int offset = addr / page_size % 1024;
90
91   //return it if it exists
92   if (dir->tables[index])
93     return &dir->tables[index]->pages[offset];
94
95   //create it
96   else if (make_table){
97     dir->tables[index] = (page_table*)_kmalloc(sizeof(page_table), 1, &phys_addr);
98     dir->tables_phys[index] = phys_addr | 0x7; //enable present, writable
99     return &dir->tables[index]->pages[offset];
100   }
101   else return 0;
102 }
```

5.16.1.5 init_paging()

```
void init_paging ( )
```

Definition at line 111 of file paging.c.

```
112 {
113   //create frame bitmap
114   nframes = (u32int)(mem_size/page_size);
115   frames = (u32int*)kmalloc(nframes/32);
116   memset(frames, 0, nframes/32);
117
118   //create kernel directory
119   kdir = (page_dir*)_kmalloc(sizeof(page_dir), 1, 0); //page aligned
120   memset(kdir, 0, sizeof(page_dir));
121
122   //get pages for kernel heap
123   u32int i = 0x0;
124   for(i=KHEAP_BASE; i<(KHEAP_BASE+KHEAP_MIN); i+=1){
125     get_page(i,kdir,1);
126   }
127
128   //perform identity mapping of used memory
129   //note: placement_addr gets incremented in get_page,
130   //so we're mapping the first frames as well
131   i = 0x0;
132   while (i < (phys_alloc_addr+0x10000)){
133     new_frame(get_page(i,kdir,1));
134     i += page_size;
135   }
136
137   //allocate heap frames now that the placement addr has increased.
138   //placement addr increases here for heap
139   for(i=KHEAP_BASE; i<(KHEAP_BASE+KHEAP_MIN);i+=PAGE_SIZE){
140     new_frame(get_page(i,kdir,1));
141   }
142
143   //load the kernel page directory; enable paging
144   load_page_dir(kdir);
145
146   //setup the kernel heap
147   kheap = make_heap(KHEAP_BASE, KHEAP_SIZE, KHEAP_BASE+KHEAP_MIN);
148 }
```

5.16.1.6 load_page_dir()

```
void load_page_dir (
              page_dir * new_dir )
```

Definition at line 158 of file paging.c.

```
159 {
160   cdir = new_dir;
161   asm volatile ("mov %0,%%cr3":: "b"(&cdir->tables_phys[0]));
162   u32int cr0;
163   asm volatile ("mov %%cr0,%0": "=b"(cr0));
164   cr0 |= 0x80000000;
165   asm volatile ("mov %0,%%cr0":: "b"(cr0));
166 }
```

5.16.1.7 new_frame()

```
void new_frame (
              page_entry * page )
```

Definition at line 173 of file paging.c.

```
174 {
175   u32int index;
```

```
176   if (page->frameaddr != 0) return;
177   if ( (u32int)(-1) == (index=find_free()) ) kpanic("Out of memory");
178
179   //mark a frame as in-use
180   set_bit(index*page_size);
181   page->present   = 1;
182   page->frameaddr = index;
183   page->writeable = 1;
184   page->usermode  = 0;
185 }
```

5.16.1.8 set_bit()

```
void set_bit (
              u32int addr )
```

Definition at line 32 of file paging.c.

```
33 {
34   u32int frame  = addr/page_size;
35   u32int index  = frame/32;
36   u32int offset = frame%32;
37   frames[index] |= (1 « offset);
38 }
```

5.16.2 Variable Documentation

5.16.2.1 cdir

```
page_dir* cdir = 0
```

Definition at line 22 of file paging.c.

5.16.2.2 frames

```
u32int* frames
```

Definition at line 19 of file paging.c.

5.16.2.3 kdir

```
page_dir* kdir = 0
```

Definition at line 21 of file paging.c.

5.16.2.4 kheap

```
heap* kheap  [extern]
```

Definition at line 14 of file heap.c.

5.16.2.5 mem_size

```
u32int mem_size = 0x4000000
```

Definition at line 15 of file paging.c.

5.16.2.6 nframes

```
u32int nframes
```

Definition at line 18 of file paging.c.

5.16.2.7 page_size

```
u32int page_size = 0x1000
```

Definition at line 16 of file paging.c.

5.16.2.8 phys_alloc_addr

```
u32int phys_alloc_addr  [extern]
```

Definition at line 22 of file heap.c.

5.17 lib/string.c File Reference

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

Functions

- int strlen (const char *s)
- char * strcpy (char *s1, const char *s2)
- int atoi (const char *s)
- int strcmp (const char *s1, const char *s2)
- 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)

5.17.1 Function Documentation

5.17.1.1 atoi()

```
int atoi (
            const char * s )
```

Definition at line 48 of file string.c.

```
49 {
50   int res=0;
51    int charVal=0;
52    char sign = ' ';
53    char c = *s;
54
55
56    while(isspace(&c)){ ++s; c = *s;} // advance past whitespace
57
58
59    if (*s == '-' || *s == '+') sign = *(s++); // save the sign
60
61
62    while(*s != '\0'){
63         charVal = *s - 48;
64     res = res * 10 + charVal;
65     s++;
66
67    }
68
69
70    if ( sign == '-') res=res * -1;
71
72   return res; // return integer
73 }
```

5.17.1.2 isspace()

```
int isspace (
            const char * c )
```

Definition at line 119 of file string.c.

```
120 {
121   if (*c == ' '  ||
122       *c == '\n' ||
123       *c == '\r' ||
124       *c == '\f' ||
125       *c == '\t' ||
126       *c == '\v'){
127     return 1;
128   }
129   return 0;
130 }
```

5.17.1.3 memset()

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

Definition at line 137 of file string.c.

```
138 {
139   unsigned char *p = (unsigned char *) s;
140   while(n--){
141     *p++ = (unsigned char) c;
142   }
143   return s;
144 }
```

5.17.1.4 strcat()

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

Definition at line 106 of file string.c.

```
107 {
108   char *rc = s1;
109   if (*s1) while(*++s1);
110   while( (*s1++ = *s2++) );
111   return rc;
112 }
```

5.17.1.5 strcmp()

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

Definition at line 79 of file string.c.

```
80 {
81
82   // Remarks:
83   // 1) If we made it to the end of both strings (i. e. our pointer points to a
84   //    '\0' character), the function will return 0
85   // 2) If we didn't make it to the end of both strings, the function will
86   //    return the difference of the characters at the first index of
87   //    indifference.
88   while ( (*s1) && (*s1==*s2) ){
89     ++s1;
90     ++s2;
91   }
92   return ( *(unsigned char *)s1 - *(unsigned char *)s2 );
93 }
```

5.17.1.6 strcpy()

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

Definition at line 36 of file string.c.

```
37 {
38   char *rc = s1;
39   while( (*s1++ = *s2++) );
40   return rc; // return pointer to destination string
41 }
```

5.17.1.7 strlen()

```
int strlen (
              const char * s )
```

Definition at line 24 of file string.c.

```
25 {
26   int r1 = 0;
27   if (*s) while(*s++) r1++;
28   return r1;//return length of string
29 }
```

5.17.1.8 strtok()

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

Definition at line 151 of file string.c.

```
152 {
153   static char *tok_tmp = NULL;
154   const char *p = s2;
155
156   //new string
157   if (s1!=NULL){
158     tok_tmp = s1;
159   }
160   //old string cont'd
161   else {
162     if (tok_tmp==NULL){
163       return NULL;
164     }
165     s1 = tok_tmp;
166   }
167
168   //skip leading s2 characters
169   while ( *p && *s1 ){
170     if (*s1==*p){
171       ++s1;
172       p = s2;
173       continue;
174     }
175     ++p;
176   }
177
178   //no more to parse
179   if (!*s1){
180     return (tok_tmp = NULL);
181   }
182
183   //skip non-s2 characters
184   tok_tmp = s1;
```

```
185   while (*tok_tmp){
186     p = s2;
187     while (*p){
188       if (*tok_tmp==*p++){
189     *tok_tmp++ = '\0';
190     return s1;
191       }
192     }
193     ++tok_tmp;
194   }
195
196   //end of string
197   tok_tmp = NULL;
198   return s1;
199 }
```

5.18 modules/mpx_supt.c File Reference

```
#include "mpx_supt.h"
#include <mem/heap.h>
#include <string.h>
#include <core/serial.h>
#include "R2/R2commands.h"
#include "R2/R2_Internal_Functions_And_Structures.h"
#include "R3/R3commands.h"
```

Functions

- int sys_req (int op_code, int device_id, char *buffer_ptr, int *count_ptr)
- 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 ()
- u32int * sys_call (context *registers)

Variables

- param params
- int current_module = -1
- u32int(* student_malloc)(u32int)
- int(* student_free)(void *)
- PCB * COP
- context * callerContext

5.18.1 Function Documentation

5.18.1.1 idle()

```
void idle ( )
```

Definition at line 178 of file mpx_supt.c.

```
179 {
180   char msg[30];
181   int count = 0;
182
183   memset(msg, '\0', sizeof(msg));
184   strcpy(msg, "IDLE PROCESS EXECUTING.\n");
185   count = strlen(msg);
186
187   while (1)
188   {
189     sys_req(WRITE, DEFAULT_DEVICE, msg, &count);
190     sys_req(IDLE, DEFAULT_DEVICE, NULL, NULL);
191   }
192 }
```

5.18.1.2 mpx_init()

```
void mpx_init (
              int cur_mod )
```

Definition at line 114 of file mpx_supt.c.

```
115 {
116
117   current_module = cur_mod;
118   if (cur_mod == MEM_MODULE)
119     mem_module_active = TRUE;
120
121   if (cur_mod == IO_MODULE)
122     io_module_active = TRUE;
123 }
```

5.18.1.3 sys_alloc_mem()

```
void* sys_alloc_mem (
              u32int size )
```

Definition at line 150 of file mpx_supt.c.

```
151 {
152   if (!mem_module_active)
153     return (void *)kmalloc(size);
154   else
155     return (void *)(*student_malloc)(size);
156 }
```

5.18.1.4 sys_call()

```
u32int* sys_call (
              context * registers )
```

Definition at line 196 of file mpx_supt.c.

```
197 { // Benjamin and Anastase programmed this function
198
199   PCB *tempOOP = NULL;
200   if (COP == NULL)
201   { // sys_call has not been called yet.
202
203     callerContext = registers;
204   }
205   else
206   {
207     if (params.op_code == IDLE)
208     { // Save the context (reassign COP's stack top).
209       COP->runningStatus = 0;
210       COP->stackTop = (unsigned char *)registers;
211       tempOOP = COP;
212     }
213     else if (params.op_code == EXIT)
214     { // free COP.
215       sys_free_mem(COP);
216     }
217   }
218
219   queue *ready = getReady();
220
221   if (ready->head != NULL)
222   {
223     COP = ready->head;
224     removePCB(COP);
225     COP->runningStatus = 1;
226
227     if (tempOOP != NULL)
228     {
229       insertPCB(tempOOP);
230     }
231
232     return (u32int *)COP->stackTop;
233   }
234   return (u32int *)callerContext;
235 }
```

5.18.1.5 sys_free_mem()

```
int sys_free_mem (
              void * ptr )
```

Definition at line 163 of file mpx_supt.c.

```
164 {
165   if (mem_module_active)
166     return (*student_free)(ptr);
167   // otherwise we don't free anything
168   return -1;
169 }
```

5.18.1.6 sys_req()

```
int sys_req (
              int op_code,
              int device_id,
```

```
              char * buffer_ptr,
              int * count_ptr )
```

Definition at line 50 of file mpx_supt.c.

```
55 {
56   int return_code = 0;
57
58   if (op_code == IDLE || op_code == EXIT)
59   {
60     // store the process's operation request
61     // triger interrupt 60h to invoke
62     params.op_code = op_code;
63     asm volatile("int $60");
64   } // idle or exit
65
66   else if (op_code == READ || op_code == WRITE)
67   {
68     // validate buffer pointer and count pointer
69     if (buffer_ptr == NULL)
70       return_code = INVALID_BUFFER;
71     else if (count_ptr == NULL || *count_ptr <= 0)
72       return_code = INVALID_COUNT;
73
74     // if parameters are valid store in the params structure
75     if (return_code == 0)
76     {
77       params.op_code = op_code;
78       params.device_id = device_id;
79       params.buffer_ptr = buffer_ptr;
80       params.count_ptr = count_ptr;
81
82       if (!io_module_active)
83       {
84         // if default device
85         if (op_code == READ)
86           return_code = *(polling(buffer_ptr, count_ptr));
87
88         else //must be WRITE
89           return_code = serial_print(buffer_ptr);
90       }
91       else
92       { // I/O module is implemented
93         asm volatile("int $60");
94       } // NOT IO_MODULE
95     }
96   }
97   else
98     return_code = INVALID_OPERATION;
99
100   return return_code;
101 } // end of sys_req
```

5.18.1.7 sys_set_free()

```
void sys_set_free (
              int(*)(void *) func )
```

Definition at line 140 of file mpx_supt.c.

```
141 {
142   student_free = func;
143 }
```

5.18.1.8 sys_set_malloc()

```
void sys_set_malloc (
              u32int(*)(u32int) func )
```

Definition at line 130 of file mpx_supt.c.

```
131 {
132   student_malloc = func;
133 }
```

5.18.2 Variable Documentation

5.18.2.1 callerContext

```
context* callerContext
```

Definition at line 195 of file mpx_supt.c.

5.18.2.2 COP

```
PCB* COP
```

Definition at line 194 of file mpx_supt.c.

5.18.2.3 current_module

```
int current_module = -1
```

Definition at line 21 of file mpx_supt.c.

5.18.2.4 params

```
param params
```

Definition at line 18 of file mpx_supt.c.

5.18.2.5 student_free

```
int(* student_free) (void *)
```

Definition at line 31 of file mpx_supt.c.

5.18.2.6 student_malloc

```
u32int(* student_malloc) (u32int)
```

Definition at line 27 of file mpx_supt.c.

5.19 modules/mpx_supt.h File Reference

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

Classes

- struct param

Macros

- #define EXIT 0
- #define IDLE 1
- #define READ 2
- #define WRITE 3
- #define INVALID_OPERATION 4
- #define TRUE 1
- #define FALSE 0
- #define MODULE_R1 0
- #define MODULE_R2 1
- #define MODULE_R3 2
- #define MODULE_R4 4
- #define MODULE_R5 8
- #define MODULE_F 9
- #define IO_MODULE 10
- #define MEM_MODULE 11
- #define INVALID_BUFFER 1000
- #define INVALID_COUNT 2000
- #define DEFAULT_DEVICE 111
- #define COM_PORT 222

Functions

- int sys_req (int op_code, int device_id, char *buffer_ptr, int *count_ptr)
- 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 ()

5.19.1 Macro Definition Documentation

5.19.1.1 COM_PORT

```
#define COM_PORT 222
```

Definition at line 29 of file mpx_supt.h.

5.19.1.2 DEFAULT_DEVICE

```
#define DEFAULT_DEVICE 111
```

Definition at line 28 of file mpx_supt.h.

5.19.1.3 EXIT

```
#define EXIT 0
```

Definition at line 6 of file mpx_supt.h.

5.19.1.4 FALSE

```
#define FALSE 0
```

Definition at line 13 of file mpx_supt.h.

5.19.1.5 IDLE

```
#define IDLE 1
```

Definition at line 7 of file mpx_supt.h.

5.19.1.6 INVALID_BUFFER

```
#define INVALID_BUFFER 1000
```

Definition at line 25 of file mpx_supt.h.

5.19.1.7 INVALID_COUNT

```
#define INVALID_COUNT 2000
```

Definition at line 26 of file mpx_supt.h.

5.19.1.8 INVALID_OPERATION

```
#define INVALID_OPERATION 4
```

Definition at line 10 of file mpx_supt.h.

5.19.1.9 IO_MODULE

```
#define IO_MODULE 10
```

Definition at line 21 of file mpx_supt.h.

5.19.1.10 MEM_MODULE

```
#define MEM_MODULE 11
```

Definition at line 22 of file mpx_supt.h.

5.19.1.11 MODULE_F

```
#define MODULE_F 9
```

Definition at line 20 of file mpx_supt.h.

5.19.1.12 MODULE_R1

```
#define MODULE_R1 0
```

Definition at line 15 of file mpx_supt.h.

5.19.1.13 MODULE_R2

```
#define MODULE_R2 1
```

Definition at line 16 of file mpx_supt.h.

5.19.1.14 MODULE_R3

```
#define MODULE_R3 2
```

Definition at line 17 of file mpx_supt.h.

5.19.1.15 MODULE_R4

```
#define MODULE_R4 4
```

Definition at line 18 of file mpx_supt.h.

5.19.1.16 MODULE_R5

```
#define MODULE_R5 8
```

Definition at line 19 of file mpx_supt.h.

5.19.1.17 READ

```
#define READ 2
```

Definition at line 8 of file mpx_supt.h.

5.19.1.18 TRUE

```
#define TRUE 1
```

Definition at line 12 of file mpx_supt.h.

5.19.1.19 WRITE

```
#define WRITE 3
```

Definition at line 9 of file mpx_supt.h.

5.19.2 Function Documentation

5.19.2.1 idle()

```
void idle ( )
```

Definition at line 178 of file mpx_supt.c.

```
179 {
180   char msg[30];
181   int count = 0;
182 
183   memset(msg, '\0', sizeof(msg));
184   strcpy(msg, "IDLE PROCESS EXECUTING.\n");
185   count = strlen(msg);
186 
187   while (1)
188   {
189     sys_req(WRITE, DEFAULT_DEVICE, msg, &count);
190     sys_req(IDLE, DEFAULT_DEVICE, NULL, NULL);
191   }
192 }
```

5.19.2.2 mpx_init()

```
void mpx_init (
              int cur_mod )
```

Definition at line 114 of file mpx_supt.c.

```
115 {
116 
117   current_module = cur_mod;
118   if (cur_mod == MEM_MODULE)
119     mem_module_active = TRUE;
120 
121   if (cur_mod == IO_MODULE)
122     io_module_active = TRUE;
123 }
```

5.19.2.3 sys_alloc_mem()

```
void* sys_alloc_mem (
              u32int size )
```

Definition at line 150 of file mpx_supt.c.

```
151 {
152   if (!mem_module_active)
153     return (void *)kmalloc(size);
154   else
155     return (void *)(*student_malloc)(size);
156 }
```

5.19.2.4 sys_free_mem()

```
int sys_free_mem (
              void * ptr )
```

Definition at line 163 of file mpx_supt.c.

```
164 {
165   if (mem_module_active)
166     return (*student_free)(ptr);
167   // otherwise we don't free anything
168   return -1;
169 }
```

5.19.2.5 sys_req()

```
int sys_req (
              int op_code,
              int device_id,
              char * buffer_ptr,
              int * count_ptr )
```

Definition at line 50 of file mpx_supt.c.

```
55 {
56   int return_code = 0;
57
58   if (op_code == IDLE || op_code == EXIT)
59   {
60     // store the process's operation request
61     // triger interrupt 60h to invoke
62     params.op_code = op_code;
63     asm volatile("int $60");
64   } // idle or exit
65
66   else if (op_code == READ || op_code == WRITE)
67   {
68     // validate buffer pointer and count pointer
69     if (buffer_ptr == NULL)
70       return_code = INVALID_BUFFER;
71     else if (count_ptr == NULL || *count_ptr <= 0)
72       return_code = INVALID_COUNT;
73
74     // if parameters are valid store in the params structure
75     if (return_code == 0)
76     {
77       params.op_code = op_code;
78       params.device_id = device_id;
79       params.buffer_ptr = buffer_ptr;
80       params.count_ptr = count_ptr;
81
82       if (!io_module_active)
83       {
84         // if default device
85         if (op_code == READ)
86           return_code = *(polling(buffer_ptr, count_ptr));
87
88         else //must be WRITE
89           return_code = serial_print(buffer_ptr);
90       }
91       else
92       { // I/O module is implemented
93         asm volatile("int $60");
94       } // NOT IO_MODULE
95     }
96   }
97   else
98     return_code = INVALID_OPERATION;
99
100   return return_code;
101 } // end of sys_req
```

5.19.2.6 sys_set_free()

```
void sys_set_free (
            int(*)(void *) func )
```

Definition at line 140 of file mpx_supt.c.

```
141 {
142   student_free = func;
143 }
```

5.19.2.7 sys_set_malloc()

```
void sys_set_malloc (
            u32int(*)(u32int) func )
```

Definition at line 130 of file mpx_supt.c.

```
131 {
132   student_malloc = func;
133 }
```

5.20 modules/R1/commhand.c File Reference

```
#include <core/serial.h>
#include <string.h>
#include "../mpx_supt.h"
#include "../utilities.h"
#include "R1commands.h"
#include "../R2/R2commands.h"
#include "../R2/R2_Internal_Functions_And_Structures.h"
#include "../R3/R3commands.h"
#include "../R4/R4commands.h"
#include "../R5/R5commands.h"
```

Functions

- void commhand ()

5.20.1 Function Documentation

5.20.1.1 commhand()

```
void commhand ( )
```

Definition at line 14 of file commhand.c.

```
15 {
16     printMessage(" \n");
17     printMessage(" \n");
18     printMessage(" \n");
19     printMessage(" \n");
20     printMessage(" \n");
21     printMessage(" \n");
22     printMessage(" \n");
23     printMessage(" \n");
24     printMessage(" \n");
25     printMessage(" \n");
26     printMessage(" \n");
27     printMessage(" \n");
28     printMessage(" \n");
29     printMessage(" \n");
30     printMessage(" \n");
31     printMessage(" \n");
32     printMessage(" \n");
33     printMessage(" \n");
34     printMessage(" \n");
35     printMessage(" \n");
36     printMessage(" \n");
37     printMessage(" \n");
38     printMessage("\n");
39     printMessage("\n");
40
41     printMessage("
       _____\n");
42     printMessage("           /
              \\\n");
43     printMessage("           |
       _____      |\n");
44     printMessage("           |   |
         |    |\n");
45     printMessage("           |   |  C:\\> Welcome to our CS 450 Project! Type help to see what you can
       do!   |    |\n");
46     printMessage("           |   |
         |    |\n");
47     printMessage("           |   |
         |    |\n");
48     printMessage("           |   |
         |    |\n");
49     printMessage("           |   |
         |    |\n");
50     printMessage("           |   |
         |    |\n");
51     printMessage("           |   |
         |    |\n");
52     printMessage("           |   |
         |    |\n");
53     printMessage("           |   |
         |    |\n");
54     printMessage("           |   |
         |    |\n");
55     printMessage("           |   |
         |    |\n");
56     printMessage("           |   |
         |    |\n");
57     printMessage("           |   |
         |    |\n");
58     printMessage("           |   |
         |    |\n");
59     printMessage("           |   |
         |    |\n");
60     printMessage("           |
       |_____|    |\n");
61     printMessage("           |
              |\n");
62     printMessage("
       \\\_____/\n");
63     printMessage("                                        \\\_____/\n");
64     printMessage("                                                 _____\n");
65     printMessage("                                    _-’    .-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.   ---  ‘-_\n");
66     printMessage("                                 _-’.-.-.  .---.-.-.-.-.-.-.-.-.-.-.-.-.-.-.--.   .-.-.‘-_\n");
67     printMessage("                              _-’.-.-.-.  .---.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-‘__‘.
       .-.-.-.-.‘-_\n");
68     printMessage("                           _-’.-.-.-.-.  .-----.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-----.
       .-.-.-.-.‘-_\n");
69     printMessage("                        _-’.-.-.-.-.-.  .---.-.  .-------------------------.  .-.---.
       .---.-.-.-.‘-_\n");
```

```
70      printMessage("
        :--------------------------------------------------------------------------:\n");
71      printMessage("
        `---._.-----------------------------------------------------------------._.-\n");
72
73      printMessage("\n\n\n");
74
75      char cmdBuffer[100];
76      int bufferSize;
77      char processName[20];
78      int processPriority;
79
80      int quitFlag = 0;
81
82      while (!quitFlag)
83      {
84          //get a command: cal polling fx
85
86          memset(cmdBuffer, '\0', 100);
87
88          bufferSize = 99; // reset size before each call to read
89
90          sys_req(READ, DEFAULT_DEVICE, cmdBuffer, &bufferSize);
91
92          printMessage("\n");
93
94          if (strcmp(cmdBuffer, "help") == 0)
95          {
96              help();
97          }
98          else if (strcmp(cmdBuffer, "version") == 0)
99          {
100             version();
101         }
102         else if (strcmp(cmdBuffer, "getDate") == 0)
103         {
104             getDate();
105         }
106         else if (strcmp(cmdBuffer, "setDate") == 0)
107         {
108             setDate();
109         }
110         else if (strcmp(cmdBuffer, "getTime") == 0)
111         {
112             getTime();
113         }
114         else if (strcmp(cmdBuffer, "setTime") == 0)
115         {
116             setTime();
117         }
118         // else if (strcmp(cmdBuffer, "createPCB") == 0)
119         // {
120         //  printMessage("Please enter a name for the PCB you wish to create. (The name can be no more
        than 20 characters)\n");
121         //  sys_req(READ, DEFAULT_DEVICE, cmdBuffer, &bufferSize);
122         //  printMessage("\n");
123         //  strcpy(processName, cmdBuffer);
124         //  memset(cmdBuffer, '\0', 100);
125
126         //  printMessage("Please enter a class for the PCB you wish to create. ('a' for application or
        's' for system)\n");
127         //  sys_req(READ, DEFAULT_DEVICE, cmdBuffer, &bufferSize);
128         //  printMessage("\n");
129         //  if (strcmp(cmdBuffer, "a") == 0)
130         //  {
131         //      processClass = 'a';
132         //  }
133         //  else if (strcmp(cmdBuffer, "s") == 0)
134         //  {
135         //      processClass = 's';
136         //  }
137         //  else
138         //  {
139         //      processClass = '\0';
140         //  }
141         //  memset(cmdBuffer, '\0', 100);
142
143         //  printMessage("Please enter a priority for the PCB you wish to create. (The priorities range
        from 0 to 9)\n");
144         //  sys_req(READ, DEFAULT_DEVICE, cmdBuffer, &bufferSize);
145         //  printMessage("\n");
146         //  processPriority = atoi(cmdBuffer);
147
148         //  createPCB(processName, processClass, processPriority);
149         // }
150         // else if (strcmp(cmdBuffer, "deletePCB") == 0)
151         // {
```

```
152         //  printMessage("Please enter the name for the PCB you wish to delete. (The name can be no more
        than 20 characters)\n");
153         //  sys_req(READ, DEFAULT_DEVICE, cmdBuffer, &bufferSize);
154         //  printMessage("\n");
155         //  strcpy(processName, cmdBuffer);
156 
157         //  deletePCB(processName);
158         // }
159         // else if (strcmp(cmdBuffer, "blockPCB") == 0)
160         // {
161         //  printMessage("Please enter the name for the PCB you wish to block. (The name can be no more
        than 20 characters)\n");
162         //  sys_req(READ, DEFAULT_DEVICE, cmdBuffer, &bufferSize);
163         //  printMessage("\n");
164         //  strcpy(processName, cmdBuffer);
165 
166         //  blockPCB(processName);
167         // }
168         // else if (strcmp(cmdBuffer, "unblockPCB") == 0)
169         // {
170         //  printMessage("Please enter the name for the PCB you wish to unblock. (The name can be no
        more than 20 characters)\n");
171         //  sys_req(READ, DEFAULT_DEVICE, cmdBuffer, &bufferSize);
172         //  printMessage("\n");
173         //  strcpy(processName, cmdBuffer);
174 
175         //  unblockPCB(processName);
176         // }
177         else if (strcmp(cmdBuffer, "suspendPCB") == 0)
178         {
179             printMessage("Please enter the name for the PCB you wish to suspend. (The name can be no
        more than 20 characters)\n");
180             sys_req(READ, DEFAULT_DEVICE, cmdBuffer, &bufferSize);
181             printMessage("\n");
182             strcpy(processName, cmdBuffer);
183 
184             suspendPCB(processName);
185         }
186         else if (strcmp(cmdBuffer, "resumePCB") == 0)
187         {
188             printMessage("Please enter the name for the PCB you wish to resume. (The name can be no more
        than 20 characters)\n");
189             sys_req(READ, DEFAULT_DEVICE, cmdBuffer, &bufferSize);
190             printMessage("\n");
191             strcpy(processName, cmdBuffer);
192 
193             resumePCB(processName);
194         }
195         else if (strcmp(cmdBuffer, "setPCBPriority") == 0)
196         {
197             printMessage("Please enter the name for the PCB you wish to change priorities for. (The name
        can be no more than 20 characters)\n");
198             sys_req(READ, DEFAULT_DEVICE, cmdBuffer, &bufferSize);
199             printMessage("\n");
200             strcpy(processName, cmdBuffer);
201 
202             printMessage("Please enter a priority for the PCB you wish to change priorities for. (The
        priorities range from 0 to 9)\n");
203             sys_req(READ, DEFAULT_DEVICE, cmdBuffer, &bufferSize);
204             printMessage("\n");
205             processPriority = atoi(cmdBuffer);
206 
207             setPCBPriority(processName, processPriority);
208         }
209         else if (strcmp(cmdBuffer, "showPCB") == 0)
210         {
211             printMessage("Please enter the name for the PCB you wish to see. (The name can be no more
        than 20 characters)\n");
212             sys_req(READ, DEFAULT_DEVICE, cmdBuffer, &bufferSize);
213             printMessage("\n");
214             strcpy(processName, cmdBuffer);
215 
216             showPCB(processName);
217         }
218         else if (strcmp(cmdBuffer, "showReady") == 0)
219         {
220             showReady();
221         }
222         else if (strcmp(cmdBuffer, "showSuspendedReady") == 0)
223         {
224             showSuspendedReady();
225         }
226         else if (strcmp(cmdBuffer, "showSuspendedBlocked") == 0)
227         {
228             showSuspendedBlocked();
229         }
230         else if (strcmp(cmdBuffer, "showBlocked") == 0)
```

```
231         {
232             showBlocked();
233         }
234         else if (strcmp(cmdBuffer, "showAll") == 0)
235         {
236             showAll();
237         }
238         // else if (strcmp(cmdBuffer, "yield") == 0)
239         // {
240         //  yield();
241         // }
242         else if (strcmp(cmdBuffer, "loadr3") == 0)
243         {
244             loadr3();
245         }
246         else if (strcmp(cmdBuffer, "infinitePCB") == 0)
247         {
248             infinitePCB();
249         }
250         // else if (strcmp(cmdBuffer, "addAlarm") == 0)
251         // {
252         //  addAlarm();
253         // }
254         // else if (strcmp(cmdBuffer, "initializeHeap") == 0) //// Need to set this up to take an input
    for the function it calls
255         // {
256 
257         //  printMessage("Please enter the desired heap size in Bytes. \n");
258         //  sys_req(READ, DEFAULT_DEVICE, cmdBuffer, &bufferSize);
259         //  printMessage("\n");
260         //  u32int size = atoi(cmdBuffer);
261 
262         //  initializeHeap(size);
263         // }
264         // else if (strcmp(cmdBuffer, "allocateMemory") == 0) //// Need to set this up to take an input
    for the function it calls
265         // {
266 
267         //  printMessage("Please enter the desired size of memory to allocate in Bytes. \n");
268         //  sys_req(READ, DEFAULT_DEVICE, cmdBuffer, &bufferSize);
269         //  printMessage("\n");
270         //  u32int size = atoi(cmdBuffer);
271 
272         //  allocateMemory(size);
273         // }
274         // else if (strcmp(cmdBuffer, "freeMemory") == 0) //// Need to set this up to take an input for
    the function it calls
275         // {
276 
277         //  printMessage("Please enter the address of the block you would like to free.\n");
278         //  sys_req(READ, DEFAULT_DEVICE, cmdBuffer, &bufferSize);
279         //  printMessage("\n");
280         //  int address = atoi(cmdBuffer);
281         //  freeMemory((u32int *)address);
282         // }
283         // else if (strcmp(cmdBuffer, "isEmpty") == 0) ////
    ------------------------------------------------------------------------------- TEMPORARY FOR
    TESTING
284         // {
285         //  isEmpty();
286         // }
287         else if (strcmp(cmdBuffer, "showFreeMemory") == 0)
288         {
289             showFreeMemory();
290         }
291         else if (strcmp(cmdBuffer, "showAllocatedMemory") == 0)
292         {
293             showAllocatedMemory();
294         }
295         else if (strcmp(cmdBuffer, "quit") == 0)
296         {
297             quitFlag = quit();
298 
299             if (quitFlag == 1)
300             {
301                 sys_req(EXIT, DEFAULT_DEVICE, NULL, NULL);
302             }
303 
304             printMessage("\n");
305         }
306 
307         else
308         {
309             printMessage("Unrecognized Command\n");
310         }
311 
312         sys_req(IDLE, DEFAULT_DEVICE, NULL, NULL);
```

```
313
314         // process the command: take array buffer chars and make a string. Decide what the cmd wants to
    do
315         // see if quit was entered: if string == quit = 1
316     }
317 }
```

5.21 modules/R1/commhand.h File Reference

Functions

- int commhand ()

5.21.1 Function Documentation

5.21.1.1 commhand()

```
int commhand ( )
```

Definition at line 14 of file commhand.c.

```
15 {
16     printMessage(" \n");
17     printMessage(" \n");
18     printMessage(" \n");
19     printMessage(" \n");
20     printMessage(" \n");
21     printMessage(" \n");
22     printMessage(" \n");
23     printMessage(" \n");
24     printMessage(" \n");
25     printMessage(" \n");
26     printMessage(" \n");
27     printMessage(" \n");
28     printMessage(" \n");
29     printMessage(" \n");
30     printMessage(" \n");
31     printMessage(" \n");
32     printMessage(" \n");
33     printMessage(" \n");
34     printMessage(" \n");
35     printMessage(" \n");
36     printMessage(" \n");
37     printMessage(" \n");
38     printMessage("\n");
39     printMessage("\n");
40
41     printMessage("
    _____\n");
42     printMessage("           /
           \\\n");
43     printMessage("           |
    _____     |\n");
44     printMessage("           |   |
       |     |\n");
45     printMessage("           |   |  C:\\> Welcome to our CS 450 Project! Type help to see what you can
    do!   |     |\n");
46     printMessage("           |   |
       |     |\n");
47     printMessage("           |   |
       |     |\n");
48     printMessage("           |   |
       |     |\n");
49     printMessage("           |   |
       |     |\n");
50     printMessage("           |   |
       |     |\n");
```

```
51      printMessage("           |   |
          |    |\n");
52      printMessage("           |   |
          |    |\n");
53      printMessage("           |   |
          |    |\n");
54      printMessage("           |   |
          |    |\n");
55      printMessage("           |   |
          |    |\n");
56      printMessage("           |   |
          |    |\n");
57      printMessage("           |   |
          |    |\n");
58      printMessage("           |   |
          |    |\n");
59      printMessage("           |   |
          |    |\n");
60      printMessage("           |
        |_____________________________________________________________________|    |\n");
61      printMessage("           |
              |\n");
62      printMessage("
        \\_____________________________________________________________________________/\n");
63      printMessage("                                    \\_____________________________________/\n");
64      printMessage("                                   _____________________________________\n");
65      printMessage("                                _-’     .-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.  --- `-_\n");
66      printMessage("                             _-’.-.-. .---.-.-.-.-.-.-.-.-.-.-.-.-.-.--.  .-.-.`-_\n");
67      printMessage("                          _-’.-.-.-. .---.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-`__`.
        .-.-.-.`-_\n");
68      printMessage("                       _-’.-.-.-.-. .-----.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-----.
        .-.-.-.-.`-_\n");
69      printMessage("                    _-’.-.-.-.-.-. .---.-. .--------------------------. .-.---.
        .---.-.-.-.`-_\n");
70      printMessage("
        :----------------------------------------------------------------------------:\n");
71      printMessage("
        `---._.-------------------------------------------------------------._.-\n");
72
73      printMessage("\n\n\n");
74
75      char cmdBuffer[100];
76      int bufferSize;
77      char processName[20];
78      int processPriority;
79
80      int quitFlag = 0;
81
82      while (!quitFlag)
83      {
84          //get a command: cal polling fx
85
86          memset(cmdBuffer, '\0', 100);
87
88          bufferSize = 99; // reset size before each call to read
89
90          sys_req(READ, DEFAULT_DEVICE, cmdBuffer, &bufferSize);
91
92          printMessage("\n");
93
94          if (strcmp(cmdBuffer, "help") == 0)
95          {
96              help();
97          }
98          else if (strcmp(cmdBuffer, "version") == 0)
99          {
100             version();
101         }
102         else if (strcmp(cmdBuffer, "getDate") == 0)
103         {
104             getDate();
105         }
106         else if (strcmp(cmdBuffer, "setDate") == 0)
107         {
108             setDate();
109         }
110         else if (strcmp(cmdBuffer, "getTime") == 0)
111         {
112             getTime();
113         }
114         else if (strcmp(cmdBuffer, "setTime") == 0)
115         {
116             setTime();
117         }
118         // else if (strcmp(cmdBuffer, "createPCB") == 0)
119         // {
120         //  printMessage("Please enter a name for the PCB you wish to create. (The name can be no more
```

```
    than 20 characters)\n");
121         //  sys_req(READ, DEFAULT_DEVICE, cmdBuffer, &bufferSize);
122         //  printMessage("\n");
123         //  strcpy(processName, cmdBuffer);
124         //  memset(cmdBuffer, '\0', 100);
125 
126         //  printMessage("Please enter a class for the PCB you wish to create. ('a' for application or
    's' for system)\n");
127         //  sys_req(READ, DEFAULT_DEVICE, cmdBuffer, &bufferSize);
128         //  printMessage("\n");
129         //  if (strcmp(cmdBuffer, "a") == 0)
130         //  {
131         //      processClass = 'a';
132         //  }
133         //  else if (strcmp(cmdBuffer, "s") == 0)
134         //  {
135         //      processClass = 's';
136         //  }
137         //  else
138         //  {
139         //      processClass = '\0';
140         //  }
141         //  memset(cmdBuffer, '\0', 100);
142 
143         //  printMessage("Please enter a priority for the PCB you wish to create. (The priorities range
    from 0 to 9)\n");
144         //  sys_req(READ, DEFAULT_DEVICE, cmdBuffer, &bufferSize);
145         //  printMessage("\n");
146         //  processPriority = atoi(cmdBuffer);
147 
148         //  createPCB(processName, processClass, processPriority);
149         // }
150         // else if (strcmp(cmdBuffer, "deletePCB") == 0)
151         // {
152         //  printMessage("Please enter the name for the PCB you wish to delete. (The name can be no more
    than 20 characters)\n");
153         //  sys_req(READ, DEFAULT_DEVICE, cmdBuffer, &bufferSize);
154         //  printMessage("\n");
155         //  strcpy(processName, cmdBuffer);
156 
157         //  deletePCB(processName);
158         // }
159         // else if (strcmp(cmdBuffer, "blockPCB") == 0)
160         // {
161         //  printMessage("Please enter the name for the PCB you wish to block. (The name can be no more
    than 20 characters)\n");
162         //  sys_req(READ, DEFAULT_DEVICE, cmdBuffer, &bufferSize);
163         //  printMessage("\n");
164         //  strcpy(processName, cmdBuffer);
165 
166         //  blockPCB(processName);
167         // }
168         // else if (strcmp(cmdBuffer, "unblockPCB") == 0)
169         // {
170         //  printMessage("Please enter the name for the PCB you wish to unblock. (The name can be no
    more than 20 characters)\n");
171         //  sys_req(READ, DEFAULT_DEVICE, cmdBuffer, &bufferSize);
172         //  printMessage("\n");
173         //  strcpy(processName, cmdBuffer);
174 
175         //  unblockPCB(processName);
176         // }
177         else if (strcmp(cmdBuffer, "suspendPCB") == 0)
178         {
179             printMessage("Please enter the name for the PCB you wish to suspend. (The name can be no
    more than 20 characters)\n");
180             sys_req(READ, DEFAULT_DEVICE, cmdBuffer, &bufferSize);
181             printMessage("\n");
182             strcpy(processName, cmdBuffer);
183 
184             suspendPCB(processName);
185         }
186         else if (strcmp(cmdBuffer, "resumePCB") == 0)
187         {
188             printMessage("Please enter the name for the PCB you wish to resume. (The name can be no more
    than 20 characters)\n");
189             sys_req(READ, DEFAULT_DEVICE, cmdBuffer, &bufferSize);
190             printMessage("\n");
191             strcpy(processName, cmdBuffer);
192 
193             resumePCB(processName);
194         }
195         else if (strcmp(cmdBuffer, "setPCBPriority") == 0)
196         {
197             printMessage("Please enter the name for the PCB you wish to change priorities for. (The name
    can be no more than 20 characters)\n");
198             sys_req(READ, DEFAULT_DEVICE, cmdBuffer, &bufferSize);
```

```
199             printMessage("\n");
200             strcpy(processName, cmdBuffer);
201 
202             printMessage("Please enter a priority for the PCB you wish to change priorities for. (The
    priorities range from 0 to 9)\n");
203             sys_req(READ, DEFAULT_DEVICE, cmdBuffer, &bufferSize);
204             printMessage("\n");
205             processPriority = atoi(cmdBuffer);
206 
207             setPCBPriority(processName, processPriority);
208         }
209         else if (strcmp(cmdBuffer, "showPCB") == 0)
210         {
211             printMessage("Please enter the name for the PCB you wish to see. (The name can be no more
    than 20 characters)\n");
212             sys_req(READ, DEFAULT_DEVICE, cmdBuffer, &bufferSize);
213             printMessage("\n");
214             strcpy(processName, cmdBuffer);
215 
216             showPCB(processName);
217         }
218         else if (strcmp(cmdBuffer, "showReady") == 0)
219         {
220             showReady();
221         }
222         else if (strcmp(cmdBuffer, "showSuspendedReady") == 0)
223         {
224             showSuspendedReady();
225         }
226         else if (strcmp(cmdBuffer, "showSuspendedBlocked") == 0)
227         {
228             showSuspendedBlocked();
229         }
230         else if (strcmp(cmdBuffer, "showBlocked") == 0)
231         {
232             showBlocked();
233         }
234         else if (strcmp(cmdBuffer, "showAll") == 0)
235         {
236             showAll();
237         }
238         // else if (strcmp(cmdBuffer, "yield") == 0)
239         // {
240         //  yield();
241         // }
242         else if (strcmp(cmdBuffer, "loadr3") == 0)
243         {
244             loadr3();
245         }
246         else if (strcmp(cmdBuffer, "infinitePCB") == 0)
247         {
248             infinitePCB();
249         }
250         // else if (strcmp(cmdBuffer, "addAlarm") == 0)
251         // {
252         //  addAlarm();
253         // }
254         // else if (strcmp(cmdBuffer, "initializeHeap") == 0) //// Need to set this up to take an input
    for the function it calls
255         // {
256 
257         //  printMessage("Please enter the desired heap size in Bytes. \n");
258         //  sys_req(READ, DEFAULT_DEVICE, cmdBuffer, &bufferSize);
259         //  printMessage("\n");
260         //  u32int size = atoi(cmdBuffer);
261 
262         //  initializeHeap(size);
263         // }
264         // else if (strcmp(cmdBuffer, "allocateMemory") == 0) //// Need to set this up to take an input
    for the function it calls
265         // {
266 
267         //  printMessage("Please enter the desired size of memory to allocate in Bytes. \n");
268         //  sys_req(READ, DEFAULT_DEVICE, cmdBuffer, &bufferSize);
269         //  printMessage("\n");
270         //  u32int size = atoi(cmdBuffer);
271 
272         //  allocateMemory(size);
273         // }
274         // else if (strcmp(cmdBuffer, "freeMemory") == 0) //// Need to set this up to take an input for
    the function it calls
275         // {
276 
277         //  printMessage("Please enter the address of the block you would like to free.\n");
278         //  sys_req(READ, DEFAULT_DEVICE, cmdBuffer, &bufferSize);
279         //  printMessage("\n");
280         //  int address = atoi(cmdBuffer);
```

```
281         //  freeMemory((u32int *)address);
282         // }
283         // else if (strcmp(cmdBuffer, "isEmpty") == 0) ////
        ------------------------------------------------------------------------------ TEMPORARY FOR
        TESTING
284         // {
285         //  isEmpty();
286         // }
287         else if (strcmp(cmdBuffer, "showFreeMemory") == 0)
288         {
289             showFreeMemory();
290         }
291         else if (strcmp(cmdBuffer, "showAllocatedMemory") == 0)
292         {
293             showAllocatedMemory();
294         }
295         else if (strcmp(cmdBuffer, "quit") == 0)
296         {
297             quitFlag = quit();
298 
299             if (quitFlag == 1)
300             {
301                 sys_req(EXIT, DEFAULT_DEVICE, NULL, NULL);
302             }
303 
304             printMessage("\n");
305         }
306 
307         else
308         {
309             printMessage("Unrecognized Command\n");
310         }
311 
312         sys_req(IDLE, DEFAULT_DEVICE, NULL, NULL);
313 
314         // process the command: take array buffer chars and make a string. Decide what the cmd wants to
        do
315         // see if quit was entered: if string == quit = 1
316     }
317 }
```

5.22 modules/R1/R1commands.c File Reference

```
#include <core/serial.h>
#include <string.h>
#include "../mpx_supt.h"
#include "../R2/R2_Internal_Functions_And_Structures.h"
#include "../R2/R2commands.h"
#include <core/io.h>
#include "../utilities.h"
```

Functions

- int BCDtoChar (unsigned char test, char *buffer)
- unsigned char intToBCD (int test)
- void help ()
- int version ()
- void getTime ()
- int setTime ()
- void getDate ()
- int setDate ()
- void deleteQueue (queue *queue)
- void removeAll ()
- int quit ()

5.22.1 Function Documentation

5.22.1.1 BCDtoChar()

```
int BCDtoChar (
            unsigned char test,
            char * buffer )
```

Definition at line 366 of file R1commands.c.

```
367 {
368
369     int val1 = (test / 16);
370     int val2 = (test % 16);
371
372     buffer[0] = val1 + '0';
373     buffer[1] = val2 + '0';
374
375     return 0;
376 }
```

5.22.1.2 deleteQueue()

```
void deleteQueue (
            queue * queue )
```

Definition at line 378 of file R1commands.c.

```
379 {
380     PCB *tempPtr;
381     int loop;
382     for (loop = 0; loop < queue->count; loop++)
383     {
384         tempPtr = queue->head;
385         removePCB(tempPtr);
386     }
387 }
```

5.22.1.3 getDate()

```
void getDate ( )
```

Definition at line 169 of file R1commands.c.

```
170 {
171
172     char buffer[4] = "\0\0\0\0";
173     int count = 4;
174     char divider = '/';
175     char newLine[1] = "\n";
176     int newLineCount = 1;
177
178     outb(0x70, 0x07); // getting Day of month value
179     BCDtoChar(inb(0x71), buffer);
180     buffer[2] = divider;
181     sys_req(WRITE, DEFAULT_DEVICE, buffer, &count);
182     memset(buffer, '\0', count);
183
184     outb(0x70, 0x08); // getting Month value
185     BCDtoChar(inb(0x71), buffer);
186     buffer[2] = divider;
187     sys_req(WRITE, DEFAULT_DEVICE, buffer, &count);
```

```
188     memset(buffer, '\0', count);
189
190     outb(0x70, 0x32); // getting Year value second byte
191     BCDtoChar(inb(0x71), buffer);
192     buffer[2] = '\0';
193     sys_req(WRITE, DEFAULT_DEVICE, buffer, &count);
194     memset(buffer, '\0', count);
195
196     outb(0x70, 0x09); // getting Year value first byte
197     BCDtoChar(inb(0x71), buffer);
198     sys_req(WRITE, DEFAULT_DEVICE, buffer, &count);
199     memset(buffer, '\0', count);
200
201     sys_req(WRITE, DEFAULT_DEVICE, newLine, &newLineCount);
202     memset(newLine, '\0', newLineCount);
203 }
```

5.22.1.4 getTime()

```
void getTime ( )
```

Definition at line 51 of file R1commands.c.

```
52 {
53
54     char buffer[4] = "\0\0\0";
55     int count = 4;
56     char divider = ':';
57     char newLine[1] = "\n";
58     int newLineCount = 1;
59
60     outb(0x70, 0x04); // getting Hour value
61     BCDtoChar(inb(0x71), buffer);
62     buffer[2] = divider;
63     sys_req(WRITE, DEFAULT_DEVICE, buffer, &count);
64     memset(buffer, '\0', count);
65
66     outb(0x70, 0x02); // getting Minute value
67     BCDtoChar(inb(0x71), buffer);
68     buffer[2] = divider;
69     sys_req(WRITE, DEFAULT_DEVICE, buffer, &count);
70     memset(buffer, '\0', count);
71
72     outb(0x70, 0x00); // getting Second value
73     BCDtoChar(inb(0x71), buffer);
74     buffer[2] = '\0';
75     sys_req(WRITE, DEFAULT_DEVICE, buffer, &count);
76     memset(buffer, '\0', count);
77
78     sys_req(WRITE, DEFAULT_DEVICE, newLine, &newLineCount);
79     memset(newLine, '\0', newLineCount);
80 }
```

5.22.1.5 help()

```
void help ( )
```

Definition at line 14 of file R1commands.c.

```
15 {
16     printMessage("help: Returns basic command information.\n");
17     printMessage("version: Returns the current version of the software.\n");
18     printMessage("getTime: Returns the current set time.\n");
19     printMessage("setTime: Allows the user to change the set time.\n");
20     printMessage("getDate: Returns the current set date.\n");
21     printMessage("setDate: Allows the user to change the set date.\n");
22     // printMessage("createPCB: Will create a PCB and put it into the ready queue by default.\n");
23     printMessage("deletePCB: Will delete a specific PCB from what ever queue it is in.\n");
24     // printMessage("blockPCB: Will change a specific PCB's state to blocked.\n");
25     // printMessage("unblockPCB: Will change a specific PCB's state to ready.\n");
26     printMessage("suspendPCB: Will suspend a specific PCB.\n");
27     printMessage("resumePCB: Will unsuspend a specific PCB.\n");
```

```
28      printMessage("setPCBPriority: Will change the priority of a specific PCB.\n");
29      printMessage("showPCB: Will display the name, class, state, suspended status, and priority of a
        specific PCB.\n");
30      printMessage("showReady: Will display the name, class, state, suspended status, and priority of every
        PCB in the ready queue.\n");
31      printMessage("showSuspendedReady: Will display the name, class, state, suspended status, and priority
        of every PCB in the suspended ready queue.\n");
32      printMessage("showSuspendedBlocked: Will display the name, class, state, suspended status, and
        priority of every PCB in the suspended blocked queue.\n");
33      printMessage("showBlocked: Will display the name, class, state, suspended status, and priority of
        every PCB in the blocked queue.\n");
34      printMessage("showReady: Will display the name, class, state, suspended status, and priority of every
        PCB in all 4 queues.\n");
35      // printMessage("yield: Will cause commhand to voluntarily allow other processes to use the
        CPU.(removed for R4)\n");
36      printMessage("loadr3: Will load all processes for R3. \n");
37      printMessage("infinitePCB: Will load a process that executes infinitely until suspended.\n");
38      //printMessage("addAlarm: Allows the user to make an alarm. The system is also able to keep track of
        multiple alarms.\n");
39      printMessage("showFreeMemory: Shows all of the free memory in the system.\n");
40      printMessage("showAllocatedMemory: Shows all of the allocated memory in the system.\n");
41      printMessage("quit: Allows the user to shut the system down.\n");
42  }
```

5.22.1.6 intToBCD()

```
unsigned char intToBCD (
              int test )
```

Definition at line 360 of file R1commands.c.

```
361 {
362
363     return (((test / 10) « 4) | (test % 10));
364 }
```

5.22.1.7 quit()

```
int quit ( )
```

Definition at line 412 of file R1commands.c.

```
413 {
414     int flag = 0;
415
416     printMessage("Are you sure you want to shutdown? y/n\n");
417
418     char quitAns[] = "\0\0";
419     int quitAnsLength = 1;
420     sys_req(READ, DEFAULT_DEVICE, quitAns, &quitAnsLength);
421     char answer = quitAns[0];
422
423     if (answer == 'y' || answer == 'Y')
424     {
425         flag = 1;
426         //removeAll processes.
427         removeAll();
428         printMessage("\n");
429     }
430     else if (answer == 'n' || answer == 'N')
431     {
432         flag = 0;
433         printMessage("\n");
434     }
435     else
436     {
437         printMessage("Invalid input!\n");
438     }
439
440     return flag;
441 }
```

5.22.1.8 removeAll()

```
void removeAll ( )
```

Definition at line 389 of file R1commands.c.

```
390 {
391     if (getReady()->head != NULL)
392     {
393         deleteQueue(getReady());
394     }
395
396     if (getBlocked()->head != NULL)
397     {
398         deleteQueue(getBlocked());
399     }
400
401     if (getSuspendedBlocked()->head != NULL)
402     {
403         deleteQueue(getSuspendedBlocked());
404     }
405
406     if (getSuspendedReady()->head != NULL)
407     {
408         deleteQueue(getSuspendedReady());
409     }
410 }
```

5.22.1.9 setDate()

```
int setDate ( )
```

Definition at line 205 of file R1commands.c.

```
206 {
207
208     int count = 4; // used to print year
209
211     printMessage("Please type the desired year. I.E.: yyyy.\n");
212
213     char year[5] = "\0\0\0\0\0"; // year buffer
214
215     int flag = 0; // thrown if input is invalid
216
217     do
218     {
219         sys_req(READ, DEFAULT_DEVICE, year, &count);
220         if (atoi(year) > 0)
221         {
222
223             printMessage("\n");
224             flag = 0;
225
226             char yearUpper[3] = "\0\0\0";
227             char yearLower[3] = "\0\0\0";
228
229             yearUpper[0] = year[0];
230             yearUpper[1] = year[1];
231             yearLower[0] = year[2];
232             yearLower[1] = year[3];
233
234             cli();
235
236             outb(0x70, 0x32); // Setting first byte year value
237             outb(0x71, intToBCD(atoi(yearUpper)));
238
239             outb(0x70, 0x09); // Setting second byte year value
240             outb(0x71, intToBCD(atoi(yearLower)));
241
242             sti();
243         }
244         else
245         {
246             printMessage("\nInvalid year.\n");
247             flag = 1;
248         }
249     } while (flag == 1);
250
```

```
252     printMessage("Please type the desired month. I.E.: mm.\n");
253
254     char month[4] = "\0\0\n\0";
255     count = 4; // used to print month
256
257     do
258     {
259         sys_req(READ, DEFAULT_DEVICE, month, &count);
260         if (atoi(month) < 13 && atoi(month) > 0)
261         {
262
263             printMessage("\n");
264             flag = 0;
265
266             cli();
267
268             outb(0x70, 0x08); // Setting month value
269             outb(0x71, intToBCD(atoi(month)));
270
271             sti();
272         }
273         else
274         {
275             printMessage("\nInvalid month.\n");
276             flag = 1;
277         }
278     } while (flag == 1);
279
280     printMessage("Please type the desired day of month. I.E.: dd.\n");
281
282     char day[4] = "\0\0\n\0";
283     count = 4; // used to print day
284
285     do
286     {
287         sys_req(READ, DEFAULT_DEVICE, day, &count);
288         printMessage("\n");
289         if ((atoi(year) % 4 == 0 && atoi(year) % 100 != 0) || atoi(year) % 400 == 0)
290         { // checking for leap year
291
292             printMessage("This is a leap year. February has 29 days.\n");
293
294             if ((atoi(month) == 1 || atoi(month) == 3 || atoi(month) == 5 || atoi(month) == 7 ||
    atoi(month) == 8 || atoi(month) == 10 || atoi(month) == 12) && atoi(day) > 31)
295             {
296                 flag = 1;
297                 printMessage("Invalid day.\n");
298             }
299             else if ((atoi(month) == 4 || atoi(month) == 6 || atoi(month) == 9 || atoi(month) == 11) &&
    atoi(day) > 30)
300             {
301                 flag = 1;
302                 printMessage("Invalid day.\n");
303             }
304             else if ((atoi(month) == 2) && atoi(day) > 29)
305             {
306                 flag = 1;
307                 printMessage("Invalid day.\n");
308             }
309             else
310             {
311
312                 flag = 0;
313                 cli();
314
315                 outb(0x70, 0x07); // Setting day of month value
316                 outb(0x71, intToBCD(atoi(day)));
317
318                 sti();
319             }
320         }
321         else if (atoi(year) % 4 != 0 || atoi(year) % 400 != 0)
322         { // checking for leap year
323
324             printMessage("This is not a leap year.\n");
325
326             if ((atoi(month) == 1 || atoi(month) == 3 || atoi(month) == 5 || atoi(month) == 7 ||
    atoi(month) == 8 || atoi(month) == 10 || atoi(month) == 12) && atoi(day) > 31)
327             {
328                 flag = 1;
329                 printMessage("Invalid day.\n");
330             }
331             else if ((atoi(month) == 4 || atoi(month) == 6 || atoi(month) == 9 || atoi(month) == 11) &&
    atoi(day) > 30)
332             {
333                 flag = 1;
334                 printMessage("Invalid day.\n");
335
```

```
336             }
337             else if ((atoi(month) == 2) && atoi(day) > 28)
338             {
339                 flag = 1;
340                 printMessage("Invalid day.\n");
341             }
342             else
343             {
344 
345                 cli();
346 
347                 outb(0x70, 0x07); // Setting day of month value
348                 outb(0x71, intToBCD(atoi(day)));
349 
350                 sti();
351             }
352         }
353 
354     } while (flag == 1);
355 
356     printMessage("The date has been set.\n");
357     return 0;
358 }
```

5.22.1.10 setTime()

```
int setTime ( )
```

Definition at line 82 of file R1commands.c.

```
83 {
84 
85     int count = 4; // counter for printing
86 
88     printMessage("Please type the desired hours. I.E.: hh.\n");
89 
90     char hour[4] = "\0\0\n\0";
91 
92     int flag = 0;
93 
94     do
95     {
96         sys_req(READ, DEFAULT_DEVICE, hour, &count);
97         if (atoi(hour) < 24 && atoi(hour) >= 0)
98         {
99 
100             printMessage("\n");
101             flag = 0;
102         }
103         else
104         {
105             printMessage("\nInvalid hours.\n");
106             flag = 1;
107         }
108     } while (flag == 1);
109 
111     printMessage("Please type the desired minutes. I.E.: mm.\n");
112 
113     char minute[4] = "\0\0\n\0";
114 
115     do
116     {
117         sys_req(READ, DEFAULT_DEVICE, minute, &count);
118         if (atoi(minute) < 60 && atoi(minute) >= 0)
119         {
120 
121             printMessage("\n");
122             flag = 0;
123         }
124         else
125         {
126             printMessage("\nInvalid minutes.\n");
127             flag = 1;
128         }
129     } while (flag == 1);
130 
132     printMessage("Please type the desired seconds. I.E.: ss.\n");
133     char second[4] = "\0\0\n\0";
134 
135     do
```

```
136     {
137         sys_req(READ, DEFAULT_DEVICE, second, &count);
138         if (atoi(second) < 60 && atoi(second) >= 0)
139         {
140 
141             printMessage("\n");
142             flag = 0;
143         }
144         else
145         {
146             printMessage("Invalid seconds.\n");
147             flag = 1;
148         }
149     } while (flag == 1);
150 
151     cli();
152 
153     outb(0x70, 0x04); // Hour
154     outb(0x71, intToBCD(atoi(hour)));
155 
156     outb(0x70, 0x02); // Minute
157     outb(0x71, intToBCD(atoi(minute)));
158 
159     outb(0x70, 0x00); // Second
160     outb(0x71, intToBCD(atoi(second)));
161 
162     sti();
163 
164     printMessage("The time has been set.\n");
165 
166     return 0;
167 }
```

5.22.1.11 version()

```
int version ( )
```

Definition at line 44 of file R1commands.c.

```
45 {
46     printMessage("Version 5\n");
47 
48     return 0;
49 }
```

5.23 modules/R1/R1commands.h File Reference

Functions

- void help ()
- void version ()
- void getTime ()
- void setTime ()
- void getDate ()
- void setDate ()
- unsigned int change_int_to_binary (int test)
- int BCDtoChar (unsigned char test, char *buffer)
- int quit ()

5.23.1 Function Documentation

5.23.1.1 BCDtoChar()

```
int BCDtoChar (
            unsigned char test,
            char * buffer )
```

Definition at line 366 of file R1commands.c.

```
367 {
368
369     int val1 = (test / 16);
370     int val2 = (test % 16);
371
372     buffer[0] = val1 + '0';
373     buffer[1] = val2 + '0';
374
375     return 0;
376 }
```

5.23.1.2 change_int_to_binary()

```
unsigned int change_int_to_binary (
            int test )
```

5.23.1.3 getDate()

```
void getDate ( )
```

Definition at line 169 of file R1commands.c.

```
170 {
171
172     char buffer[4] = "\0\0\0\0";
173     int count = 4;
174     char divider = '/';
175     char newLine[1] = "\n";
176     int newLineCount = 1;
177
178     outb(0x70, 0x07); // getting Day of month value
179     BCDtoChar(inb(0x71), buffer);
180     buffer[2] = divider;
181     sys_req(WRITE, DEFAULT_DEVICE, buffer, &count);
182     memset(buffer, '\0', count);
183
184     outb(0x70, 0x08); // getting Month value
185     BCDtoChar(inb(0x71), buffer);
186     buffer[2] = divider;
187     sys_req(WRITE, DEFAULT_DEVICE, buffer, &count);
188     memset(buffer, '\0', count);
189
190     outb(0x70, 0x32); // getting Year value second byte
191     BCDtoChar(inb(0x71), buffer);
192     buffer[2] = '\0';
193     sys_req(WRITE, DEFAULT_DEVICE, buffer, &count);
194     memset(buffer, '\0', count);
195
196     outb(0x70, 0x09); // getting Year value first byte
197     BCDtoChar(inb(0x71), buffer);
198     sys_req(WRITE, DEFAULT_DEVICE, buffer, &count);
199     memset(buffer, '\0', count);
200
201     sys_req(WRITE, DEFAULT_DEVICE, newLine, &newLineCount);
202     memset(newLine, '\0', newLineCount);
203 }
```

5.23.1.4 getTime()

```
void getTime ( )
```

Definition at line 51 of file R1commands.c.

```
52 {
53
54     char buffer[4] = "\0\0\0";
55     int count = 4;
56     char divider = ':';
57     char newLine[1] = "\n";
58     int newLineCount = 1;
59
60     outb(0x70, 0x04); // getting Hour value
61     BCDtoChar(inb(0x71), buffer);
62     buffer[2] = divider;
63     sys_req(WRITE, DEFAULT_DEVICE, buffer, &count);
64     memset(buffer, '\0', count);
65
66     outb(0x70, 0x02); // getting Minute value
67     BCDtoChar(inb(0x71), buffer);
68     buffer[2] = divider;
69     sys_req(WRITE, DEFAULT_DEVICE, buffer, &count);
70     memset(buffer, '\0', count);
71
72     outb(0x70, 0x00); // getting Second value
73     BCDtoChar(inb(0x71), buffer);
74     buffer[2] = '\0';
75     sys_req(WRITE, DEFAULT_DEVICE, buffer, &count);
76     memset(buffer, '\0', count);
77
78     sys_req(WRITE, DEFAULT_DEVICE, newLine, &newLineCount);
79     memset(newLine, '\0', newLineCount);
80 }
```

5.23.1.5 help()

```
void help ( )
```

Definition at line 14 of file R1commands.c.

```
15 {
16     printMessage("help: Returns basic command information.\n");
17     printMessage("version: Returns the current version of the software.\n");
18     printMessage("getTime: Returns the current set time.\n");
19     printMessage("setTime: Allows the user to change the set time.\n");
20     printMessage("getDate: Returns the current set date.\n");
21     printMessage("setDate: Allows the user to change the set date.\n");
22     // printMessage("createPCB: Will create a PCB and put it into the ready queue by default.\n");
23     printMessage("deletePCB: Will delete a specific PCB from what ever queue it is in.\n");
24     // printMessage("blockPCB: Will change a specific PCB's state to blocked.\n");
25     // printMessage("unblockPCB: Will change a specific PCB's state to ready.\n");
26     printMessage("suspendPCB: Will suspend a specific PCB.\n");
27     printMessage("resumePCB: Will unsuspend a specific PCB.\n");
28     printMessage("setPCBPriority: Will change the priority of a specific PCB.\n");
29     printMessage("showPCB: Will display the name, class, state, suspended status, and priority of a
       specific PCB.\n");
30     printMessage("showReady: Will display the name, class, state, suspended status, and priority of every
       PCB in the ready queue.\n");
31     printMessage("showSuspendedReady: Will display the name, class, state, suspended status, and priority
       of every PCB in the suspended ready queue.\n");
32     printMessage("showSuspendedBlocked: Will display the name, class, state, suspended status, and
       priority of every PCB in the suspended blocked queue.\n");
33     printMessage("showBlocked: Will display the name, class, state, suspended status, and priority of
       every PCB in the blocked queue.\n");
34     printMessage("showReady: Will display the name, class, state, suspended status, and priority of every
       PCB in all 4 queues.\n");
35     // printMessage("yield: Will cause commhand to voluntarily allow other processes to use the
       CPU.(removed for R4)\n");
36     printMessage("loadr3: Will load all processes for R3. \n");
37     printMessage("infinitePCB: Will load a process that executes infinitely until suspended.\n");
38     //printMessage("addAlarm: Allows the user to make an alarm. The system is also able to keep track of
       multiple alarms.\n");
39     printMessage("showFreeMemory: Shows all of the free memory in the system.\n");
40     printMessage("showAllocatedMemory: Shows all of the allocated memory in the system.\n");
41     printMessage("quit: Allows the user to shut the system down.\n");
42 }
```

5.23.1.6 quit()

```
int quit ( )
```

Definition at line 412 of file R1commands.c.

```
413 {
414     int flag = 0;
415 
416     printMessage("Are you sure you want to shutdown? y/n\n");
417 
418     char quitAns[] = "\0\0";
419     int quitAnsLength = 1;
420     sys_req(READ, DEFAULT_DEVICE, quitAns, &quitAnsLength);
421     char answer = quitAns[0];
422 
423     if (answer == 'y' || answer == 'Y')
424     {
425         flag = 1;
426         //removeAll processes.
427         removeAll();
428         printMessage("\n");
429     }
430     else if (answer == 'n' || answer == 'N')
431     {
432         flag = 0;
433         printMessage("\n");
434     }
435     else
436     {
437         printMessage("Invalid input!\n");
438     }
439 
440     return flag;
441 }
```

5.23.1.7 setDate()

```
void setDate ( )
```

Definition at line 205 of file R1commands.c.

```
206 {
207 
208     int count = 4; // used to print year
209 
211     printMessage("Please type the desired year. I.E.: yyyy.\n");
212 
213     char year[5] = "\0\0\0\0\0"; // year buffer
214 
215     int flag = 0; // thrown if input is invalid
216 
217     do
218     {
219         sys_req(READ, DEFAULT_DEVICE, year, &count);
220         if (atoi(year) > 0)
221         {
222 
223             printMessage("\n");
224             flag = 0;
225 
226             char yearUpper[3] = "\0\0\0";
227             char yearLower[3] = "\0\0\0";
228 
229             yearUpper[0] = year[0];
230             yearUpper[1] = year[1];
231             yearLower[0] = year[2];
232             yearLower[1] = year[3];
233 
234             cli();
235 
236             outb(0x70, 0x32); // Setting first byte year value
237             outb(0x71, intToBCD(atoi(yearUpper)));
238 
239             outb(0x70, 0x09); // Setting second byte year value
240             outb(0x71, intToBCD(atoi(yearLower)));
241 
242             sti();
```

```
243         }
244         else
245         {
246             printMessage("\nInvalid year.\n");
247             flag = 1;
248         }
249     } while (flag == 1);
250 
252     printMessage("Please type the desired month. I.E.: mm.\n");
253 
254     char month[4] = "\0\0\n\0";
255     count = 4; // used to print month
256 
257     do
258     {
259         sys_req(READ, DEFAULT_DEVICE, month, &count);
260         if (atoi(month) < 13 && atoi(month) > 0)
261         {
262 
263             printMessage("\n");
264             flag = 0;
265 
266             cli();
267 
268             outb(0x70, 0x08); // Setting month value
269             outb(0x71, intToBCD(atoi(month)));
270 
271             sti();
272         }
273         else
274         {
275             printMessage("\nInvalid month.\n");
276             flag = 1;
277         }
278     } while (flag == 1);
279 
281     printMessage("Please type the desired day of month. I.E.: dd.\n");
282 
283     char day[4] = "\0\0\n\0";
284     count = 4; // used to print day
285 
286     do
287     {
288         sys_req(READ, DEFAULT_DEVICE, day, &count);
289         printMessage("\n");
290         if ((atoi(year) % 4 == 0 && atoi(year) % 100 != 0) || atoi(year) % 400 == 0)
291         { // checking for leap year
292 
293             printMessage("This is a leap year. February has 29 days.\n");
294 
295             if ((atoi(month) == 1 || atoi(month) == 3 || atoi(month) == 5 || atoi(month) == 7 ||
    atoi(month) == 8 || atoi(month) == 10 || atoi(month) == 12) && atoi(day) > 31)
296             {
297                 flag = 1;
298                 printMessage("Invalid day.\n");
299             }
300             else if ((atoi(month) == 4 || atoi(month) == 6 || atoi(month) == 9 || atoi(month) == 11) &&
    atoi(day) > 30)
301             {
302                 flag = 1;
303                 printMessage("Invalid day.\n");
304             }
305             else if ((atoi(month) == 2) && atoi(day) > 29)
306             {
307                 flag = 1;
308                 printMessage("Invalid day.\n");
309             }
310             else
311             {
312 
313                 flag = 0;
314                 cli();
315 
316                 outb(0x70, 0x07); // Setting day of month value
317                 outb(0x71, intToBCD(atoi(day)));
318 
319                 sti();
320             }
321         }
322         else if (atoi(year) % 4 != 0 || atoi(year) % 400 != 0)
323         { // checking for leap year
324 
325             printMessage("This is not a leap year.\n");
326 
327             if ((atoi(month) == 1 || atoi(month) == 3 || atoi(month) == 5 || atoi(month) == 7 ||
    atoi(month) == 8 || atoi(month) == 10 || atoi(month) == 12) && atoi(day) > 31)
328             {
```

```
329                 flag = 1;
330                 printMessage("Invalid day.\n");
331             }
332             else if ((atoi(month) == 4 || atoi(month) == 6 || atoi(month) == 9 || atoi(month) == 11) &&
    atoi(day) > 30)
333             {
334                 flag = 1;
335                 printMessage("Invalid day.\n");
336             }
337             else if ((atoi(month) == 2) && atoi(day) > 28)
338             {
339                 flag = 1;
340                 printMessage("Invalid day.\n");
341             }
342             else
343             {
344 
345                 cli();
346 
347                 outb(0x70, 0x07); // Setting day of month value
348                 outb(0x71, intToBCD(atoi(day)));
349 
350                 sti();
351             }
352         }
353 
354     } while (flag == 1);
355 
356     printMessage("The date has been set.\n");
357     return 0;
358 }
```

5.23.1.8 setTime()

```
void setTime ( )
```

Definition at line 82 of file R1commands.c.

```
83 {
84 
85     int count = 4; // counter for printing
86 
88     printMessage("Please type the desired hours. I.E.: hh.\n");
89 
90     char hour[4] = "\0\0\n\0";
91 
92     int flag = 0;
93 
94     do
95     {
96         sys_req(READ, DEFAULT_DEVICE, hour, &count);
97         if (atoi(hour) < 24 && atoi(hour) >= 0)
98         {
99 
100             printMessage("\n");
101             flag = 0;
102         }
103         else
104         {
105             printMessage("\nInvalid hours.\n");
106             flag = 1;
107         }
108     } while (flag == 1);
109 
111     printMessage("Please type the desired minutes. I.E.: mm.\n");
112 
113     char minute[4] = "\0\0\n\0";
114 
115     do
116     {
117         sys_req(READ, DEFAULT_DEVICE, minute, &count);
118         if (atoi(minute) < 60 && atoi(minute) >= 0)
119         {
120 
121             printMessage("\n");
122             flag = 0;
123         }
124         else
125         {
126             printMessage("\nInvalid minutes.\n");
```

```
127             flag = 1;
128         }
129     } while (flag == 1);
130 
132     printMessage("Please type the desired seconds. I.E.: ss.\n");
133     char second[4] = "\0\0\n\0";
134 
135     do
136     {
137         sys_req(READ, DEFAULT_DEVICE, second, &count);
138         if (atoi(second) < 60 && atoi(second) >= 0)
139         {
140 
141             printMessage("\n");
142             flag = 0;
143         }
144         else
145         {
146             printMessage("Invalid seconds.\n");
147             flag = 1;
148         }
149     } while (flag == 1);
150 
151     cli();
152 
153     outb(0x70, 0x04); // Hour
154     outb(0x71, intToBCD(atoi(hour)));
155 
156     outb(0x70, 0x02); // Minute
157     outb(0x71, intToBCD(atoi(minute)));
158 
159     outb(0x70, 0x00); // Second
160     outb(0x71, intToBCD(atoi(second)));
161 
162     sti();
163 
164     printMessage("The time has been set.\n");
165 
166     return 0;
167 }
```

5.23.1.9 version()

```
void version ( )
```

Definition at line 44 of file R1commands.c.

```
45 {
46     printMessage("Version 5\n");
47 
48     return 0;
49 }
```

5.24 modules/R2/R2_Internal_Functions_And_Structures.c File Reference

```
#include <string.h>
#include <core/serial.h>
#include "../mpx_supt.h"
#include "../utilities.h"
#include "R2_Internal_Functions_And_Structures.h"
#include "../R3/R3commands.h"
```

Functions

- PCB * allocatePCB ()
- int freePCB (PCB *PCB_to_free)
- PCB * setupPCB (char *processName, unsigned char processClass, int processPriority)
- PCB * findPCB (char *processName)
- void insertPCB (PCB *PCB_to_insert)
- int removePCB (PCB *PCB_to_remove)
- void allocateQueues ()
- queue * getReady ()
- queue * getBlocked ()
- queue * getSuspendedReady ()
- queue * getSuspendedBlocked ()

Variables

- queue * ready
- queue * blocked
- queue * suspendedReady
- queue * suspendedBlocked

5.24.1 Function Documentation

5.24.1.1 allocatePCB()

```
PCB* allocatePCB ( )
```

Definition at line 17 of file R2_Internal_Functions_And_Structures.c.

```
18 {
19     //COLTON WILL PROGRAM THIS FUNCTION
20
21     //allocatePCB() will use sys_alloc_mem() to allocate memory for a new PCB, possible including the
       stack, and perform any reasonable initialization.
22     PCB *newPCB = (PCB *)sys_alloc_mem(sizeof(PCB));
23
24     char name[20] = "newPCB";
25     strcpy(newPCB->processName, name);
26
27     newPCB->suspendedStatus = 1;
28     newPCB->runningStatus = -1;
29     newPCB->stackTop = (newPCB->stack + 1024) - sizeof(context);
30     newPCB->stackBase = newPCB->stack;
31     newPCB->priority = 0;
32
33     // Setting the PCBs prev and next PCB
34     newPCB->nextPCB = NULL;
35     newPCB->prevPCB = NULL;
36
37     newPCB->processClass = NULL;
38
39     return newPCB;
40 }
```

5.24.1.2 allocateQueues()

```
void allocateQueues ( )
```

Definition at line 430 of file R2_Internal_Functions_And_Structures.c.

```
431 {
432     ready = sys_alloc_mem(sizeof(queue));
433     ready->count = 0;
434     ready->head = NULL;
435     ready->tail = NULL;
436 
437     blocked = sys_alloc_mem(sizeof(queue));
438     blocked->count = 0;
439     blocked->head = NULL;
440     blocked->tail = NULL;
441 
442     suspendedReady = sys_alloc_mem(sizeof(queue));
443     suspendedReady->count = 0;
444     suspendedReady->head = NULL;
445     suspendedReady->tail = NULL;
446 
447     suspendedBlocked = sys_alloc_mem(sizeof(queue));
448     suspendedBlocked->count = 0;
449     suspendedBlocked->head = NULL;
450     suspendedBlocked->tail = NULL;
451 }
```

5.24.1.3 findPCB()

```
PCB* findPCB (
            char * processName )
```

Definition at line 82 of file R2_Internal_Functions_And_Structures.c.

```
83 {
84     // ANASTASE WILL PROGRAM THIS FUNCTION
85 
86     //findPCB() will search all queues for a process with a given name.
87 
88     if (strlen(processName) > 20)
89     {
90 
91         printMessage("Invalid process name.\n");
92         return NULL;
93         //return cz we have to stop if the process name is too long
94     }
95     else
96     {
97         PCB *tempPCB = ready->head;
98         int value = 0;
99         while (value < ready->count)
100         {
101             if (strcmp(tempPCB->processName, processName) == 0)
102             {
103                 return tempPCB;
104             }
105             else
106             {
107                 tempPCB = tempPCB->nextPCB;
108                 value++;
109             }
110         }
111 
112         tempPCB = blocked->head;
113         value = 0;
114         while (value < blocked->count)
115         {
116             if (strcmp(tempPCB->processName, processName) == 0)
117             {
118                 return tempPCB;
119             }
120             else
121             {
122                 tempPCB = tempPCB->nextPCB;
123                 value++;
124             }
```

```
125         }
126
127         tempPCB = suspendedBlocked->head;
128         value = 0;
129         while (value < suspendedBlocked->count)
130         {
131             if (strcmp(tempPCB->processName, processName) == 0)
132             {
133                 return tempPCB;
134             }
135             else
136             {
137                 tempPCB = tempPCB->nextPCB;
138                 value++;
139             }
140         }
141
142         tempPCB = suspendedReady->head;
143         value = 0;
144         while (value < suspendedReady->count)
145         {
146             if (strcmp(tempPCB->processName, processName) == 0)
147             {
148                 return tempPCB;
149             }
150             else
151             {
152                 tempPCB = tempPCB->nextPCB;
153                 value++;
154             }
155         }
156
157         return NULL;
158     }
159 }
```

5.24.1.4 freePCB()

```
int freePCB (
              PCB * PCB_to_free )
```

Definition at line 42 of file R2_Internal_Functions_And_Structures.c.

```
43 {
44     // ANASTASE WILL PROGRAM THIS FUNCTION
45
46     //freePCB() will use sys_free_mem() to free all memory associated with a given PCB (the stack, the
       PCB itself, etc.)
47
48     return sys_free_mem(PCB_to_free);
49 }
```

5.24.1.5 getBlocked()

```
queue* getBlocked ( )
```

Definition at line 458 of file R2_Internal_Functions_And_Structures.c.

```
459 {
460     return blocked;
461 }
```

5.24.1.6 getReady()

```
queue* getReady ( )
```

Definition at line 453 of file R2_Internal_Functions_And_Structures.c.

```
454 {
455     return ready;
456 }
```

5.24.1.7 getSuspendedBlocked()

```
queue* getSuspendedBlocked ( )
```

Definition at line 468 of file R2_Internal_Functions_And_Structures.c.

```
469 {
470     return suspendedBlocked;
471 }
```

5.24.1.8 getSuspendedReady()

```
queue* getSuspendedReady ( )
```

Definition at line 463 of file R2_Internal_Functions_And_Structures.c.

```
464 {
465     return suspendedReady;
466 }
```

5.24.1.9 insertPCB()

```
void insertPCB (
            PCB * PCB_to_insert )
```

Definition at line 161 of file R2_Internal_Functions_And_Structures.c.

```
162 {
163     //BENJAMIN WILL PROGRAM THIS FUNCTION
164
165     //insertPCB() will insert a PCB into the appropriate queue.
166     //Note: The ready queue is a priority queue and the blocked queue is a FIFO queue.
167
168     if (PCB_to_insert->runningStatus == 0 && PCB_to_insert->suspendedStatus == 1)
169     { // Insert into ready queue
170         PCB *tempPtr = ready->head;
171
172         if (tempPtr != NULL)
173         {
174             int temp = 0;
175             while (temp < ready->count)
176             {
177                 if (PCB_to_insert->priority > ready->head->priority)
178                 { // insert at head
179                     PCB_to_insert->nextPCB = tempPtr;
180                     tempPtr->prevPCB = PCB_to_insert;
181                     ready->head = PCB_to_insert;
182                     ready->count++;
183                     break;
184                 }
185                 else if (PCB_to_insert->priority <= ready->tail->priority)
186                 { // insert at tail
```

```
187                     ready->tail->nextPCB = PCB_to_insert;
188                     PCB_to_insert->prevPCB = ready->tail;
189                     ready->tail = PCB_to_insert;
190                     ready->count++;
191                     break;
192                 }
193                 else if (PCB_to_insert->priority > tempPtr->priority)
194                 { // insert at middle
195                     PCB *prevPtr = tempPtr->prevPCB;
196 
197                     prevPtr->nextPCB = PCB_to_insert;
198 
199                     PCB_to_insert->prevPCB = prevPtr;
200                     PCB_to_insert->nextPCB = tempPtr;
201 
202                     tempPtr->prevPCB = PCB_to_insert;
203 
204                     ready->count++;
205                     break;
206                 }
207                 else
208                 { // move tempPtr through the queue
209                     tempPtr = tempPtr->nextPCB;
210                 }
211                 temp++;
212             }
213         }
214         else
215         {
216             ready->head = PCB_to_insert;
217             ready->tail = PCB_to_insert;
218             ready->count++;
219         }
220     }
221     else if (PCB_to_insert->runningStatus == 0 && PCB_to_insert->suspendedStatus == 0)
222     { // Insert into suspended ready queue
223         PCB *tempPtr = suspendedReady->head;
224 
225         if (tempPtr != NULL)
226         {
227             int temp = 0;
228             while (temp < suspendedReady->count)
229             {
230                 if (PCB_to_insert->priority > suspendedReady->head->priority)
231                 { // insert at head
232                     PCB_to_insert->nextPCB = tempPtr;
233                     tempPtr->prevPCB = PCB_to_insert;
234                     suspendedReady->head = PCB_to_insert;
235                     suspendedReady->count++;
236                     break;
237                 }
238                 else if (PCB_to_insert->priority <= suspendedReady->tail->priority)
239                 { // insert at tail
240 
241                     suspendedReady->tail->nextPCB = PCB_to_insert;
242                     PCB_to_insert->prevPCB = suspendedReady->tail;
243                     suspendedReady->tail = PCB_to_insert;
244                     suspendedReady->count++;
245                     break;
246                 }
247                 else if (PCB_to_insert->priority > tempPtr->priority)
248                 { // insert at middle
249                     PCB *prevPtr = tempPtr->prevPCB;
250 
251                     prevPtr->nextPCB = PCB_to_insert;
252 
253                     PCB_to_insert->prevPCB = prevPtr;
254                     PCB_to_insert->nextPCB = tempPtr;
255 
256                     tempPtr->prevPCB = PCB_to_insert;
257 
258                     ready->count++;
259                     break;
260                 }
261                 else
262                 { // move tempPtr through the queue
263                     tempPtr = tempPtr->nextPCB;
264                 }
265                 temp++;
266             }
267         }
268         else
269         {
270             suspendedReady->count++;
271             suspendedReady->head = PCB_to_insert;
272             suspendedReady->tail = PCB_to_insert;
273         }
```

```
274     }
275     else if (PCB_to_insert->runningStatus == -1 && PCB_to_insert->suspendedStatus == 1)
276     { // Insert into blocked queue
277         if (blocked->head != NULL)
278         {
279             blocked->tail->nextPCB = PCB_to_insert;
280             PCB_to_insert->prevPCB = blocked->tail;
281             blocked->tail = PCB_to_insert;
282             blocked->count++;
283         }
284         else
285         {
286             blocked->head = PCB_to_insert;
287             blocked->tail = PCB_to_insert;
288             blocked->count++;
289         }
290     }
291     else if (PCB_to_insert->runningStatus == -1 && PCB_to_insert->suspendedStatus == 0)
292     { // Insert into suspended blocked queue
293         if (suspendedBlocked->head != NULL)
294         {
295             suspendedBlocked->tail->nextPCB = PCB_to_insert;
296             PCB_to_insert->prevPCB = suspendedBlocked->tail;
297             suspendedBlocked->tail = PCB_to_insert;
298             suspendedBlocked->count++;
299         }
300         else
301         {
302             suspendedBlocked->head = PCB_to_insert;
303             suspendedBlocked->tail = PCB_to_insert;
304             suspendedBlocked->count++;
305         }
306     }
307 }
```

5.24.1.10 removePCB()

```
int removePCB (
            PCB * PCB_to_remove )
```

Definition at line 309 of file R2_Internal_Functions_And_Structures.c.

```
310 {
311     //BENJAMIN WILL PROGRAM THIS FUNCTION
312
313     //removePCB() will remove a PCB from the queue in which it is currently stored.
314
315     if (PCB_to_remove == NULL)
316     {
317         return 1;
318     }
319     else if (PCB_to_remove == ready->head)
320     {
321         //PCB *removedNext = PCB_to_remove->nextPCB;
322
323         ready->head = PCB_to_remove->nextPCB;
324         ready->head->prevPCB = NULL;
325         PCB_to_remove->nextPCB = NULL;
326         ready->count--;
327         return 0;
328     }
329     else if (PCB_to_remove == blocked->head)
330     {
331         PCB *removedNext = PCB_to_remove->nextPCB;
332         blocked->head = removedNext;
333         removedNext->prevPCB = NULL;
334         PCB_to_remove->nextPCB = NULL;
335         blocked->count--;
336         return 0;
337     }
338     else if (PCB_to_remove == suspendedReady->head)
339     {
340         PCB *removedNext = PCB_to_remove->nextPCB;
341
342         suspendedReady->head = removedNext;
343         removedNext->prevPCB = NULL;
344         PCB_to_remove->nextPCB = NULL;
345         suspendedReady->count--;
346         return 0;
```

```
347     }
348     else if (PCB_to_remove == suspendedBlocked->head)
349     {
350         PCB *removedNext = PCB_to_remove->nextPCB;
351 
352         suspendedBlocked->head = removedNext;
353         removedNext->prevPCB = NULL;
354         PCB_to_remove->nextPCB = NULL;
355         suspendedBlocked->count--;
356         return 0;
357     }
358     else if (PCB_to_remove == ready->tail)
359     {
360         PCB *removedPrev = PCB_to_remove->prevPCB;
361 
362         ready->tail = removedPrev;
363         removedPrev->nextPCB = NULL;
364         PCB_to_remove->prevPCB = NULL;
365         ready->count--;
366         return 0;
367     }
368     else if (PCB_to_remove == blocked->tail)
369     {
370         PCB *removedPrev = PCB_to_remove->prevPCB;
371 
372         blocked->tail = removedPrev;
373         removedPrev->nextPCB = NULL;
374         PCB_to_remove->prevPCB = NULL;
375         blocked->count--;
376         return 0;
377     }
378     else if (PCB_to_remove == suspendedReady->tail)
379     {
380         PCB *removedPrev = PCB_to_remove->prevPCB;
381 
382         suspendedReady->tail = removedPrev;
383         removedPrev->nextPCB = NULL;
384         PCB_to_remove->prevPCB = NULL;
385         suspendedReady->count--;
386         return 0;
387     }
388     else if (PCB_to_remove == suspendedBlocked->tail)
389     {
390         PCB *removedPrev = PCB_to_remove->prevPCB;
391 
392         suspendedBlocked->tail = removedPrev;
393         removedPrev->nextPCB = NULL;
394         PCB_to_remove->prevPCB = NULL;
395         suspendedBlocked->count--;
396         return 0;
397     }
398     else
399     {
400         // PCB *tempPrev = PCB_to_remove->prevPCB;
401         // PCB *tempNext = PCB_to_remove->nextPCB;
402 
403         PCB_to_remove->prevPCB->nextPCB = PCB_to_remove->nextPCB;
404         PCB_to_remove->nextPCB->prevPCB = PCB_to_remove->prevPCB;
405 
406         PCB_to_remove->nextPCB = NULL;
407         PCB_to_remove->prevPCB = NULL;
408 
409         if (PCB_to_remove->runningStatus == 0 && PCB_to_remove->suspendedStatus == 1)
410         {
411             ready->count--;
412         }
413         else if (PCB_to_remove->runningStatus == -1 && PCB_to_remove->suspendedStatus == 1)
414         {
415             blocked->count--;
416         }
417         else if (PCB_to_remove->runningStatus == 0 && PCB_to_remove->suspendedStatus == 0)
418         {
419             suspendedReady->count--;
420         }
421         else if (PCB_to_remove->runningStatus == -1 && PCB_to_remove->suspendedStatus == 0)
422         {
423             suspendedBlocked->count--;
424         }
425 
426         return 0;
427     }
428 }
```

5.24.1.11 setupPCB()

```
PCB* setupPCB (
            char * processName,
            unsigned char processClass,
            int processPriority )
```

Definition at line 51 of file R2_Internal_Functions_And_Structures.c.

```
52 {
53     //COLTON WILL PROGRAM THIS FUNCTION
54
55     //setupPcb() will call allocatePCB() to create an empty PCB, initializes the PCB information, sets
       the PCB state to ready, not suspended.
56
57     PCB *returnedPCB = allocatePCB();
58
59     if (findPCB(processName)->processName == processName)
60     {
61         printMessage("There is already a PCB with this name.\n");
62
63         returnedPCB = NULL;
64     }
65     else
66     {
67
68         strcpy(returnedPCB->processName, processName);
69         returnedPCB->processClass = processClass;
70         returnedPCB->priority = processPriority;
71         returnedPCB->runningStatus = 0;
72         returnedPCB->suspendedStatus = 1;
73         returnedPCB->stackBase = returnedPCB->stack;
74         returnedPCB->stackTop = returnedPCB->stack + 1024 - sizeof(context);
75         returnedPCB->nextPCB = NULL;
76         returnedPCB->prevPCB = NULL;
77     }
78
79     return returnedPCB;
80 }
```

5.24.2 Variable Documentation

5.24.2.1 blocked

```
queue* blocked
```

Definition at line 11 of file R2_Internal_Functions_And_Structures.c.

5.24.2.2 ready

```
queue* ready
```

Definition at line 10 of file R2_Internal_Functions_And_Structures.c.

5.24.2.3 suspendedBlocked

```
queue* suspendedBlocked
```

Definition at line 13 of file R2_Internal_Functions_And_Structures.c.

5.24.2.4 suspendedReady

```
queue* suspendedReady
```

Definition at line 12 of file R2_Internal_Functions_And_Structures.c.

5.25 modules/R2/R2_Internal_Functions_And_Structures.h File Reference

Classes

- struct PCB
- struct queue

Typedefs

- typedef struct PCB PCB
- typedef struct queue queue

Functions

- PCB * allocatePCB ()
- int freePCB (PCB *PCB_to_free)
- PCB * setupPCB (char *processName, unsigned char processClass, int processPriority)
- PCB * findPCB (char *processName)
- void insertPCB (PCB *PCB_to_insert)
- int removePCB (PCB *PCB_to_remove)
- void allocateQueues ()
- queue * getReady ()
- queue * getBlocked ()
- queue * getSuspendedReady ()
- queue * getSuspendedBlocked ()

5.25.1 Typedef Documentation

5.25.1.1 PCB

```
typedef struct PCB PCB
```

5.25.1.2 queue

```
typedef struct queue queue
```

5.25.2 Function Documentation

5.25.2.1 allocatePCB()

```
PCB* allocatePCB ( )
```

Definition at line 17 of file R2_Internal_Functions_And_Structures.c.

```
18 {
19     //COLTON WILL PROGRAM THIS FUNCTION
20 
21     //allocatePCB() will use sys_alloc_mem() to allocate memory for a new PCB, possible including the
       stack, and perform any reasonable initialization.
22     PCB *newPCB = (PCB *)sys_alloc_mem(sizeof(PCB));
23 
24     char name[20] = "newPCB";
25     strcpy(newPCB->processName, name);
26 
27     newPCB->suspendedStatus = 1;
28     newPCB->runningStatus = -1;
29     newPCB->stackTop = (newPCB->stack + 1024) - sizeof(context);
30     newPCB->stackBase = newPCB->stack;
31     newPCB->priority = 0;
32 
33     // Setting the PCBs prev and next PCB
34     newPCB->nextPCB = NULL;
35     newPCB->prevPCB = NULL;
36 
37     newPCB->processClass = NULL;
38 
39     return newPCB;
40 }
```

5.25.2.2 allocateQueues()

```
void allocateQueues ( )
```

Definition at line 430 of file R2_Internal_Functions_And_Structures.c.

```
431 {
432     ready = sys_alloc_mem(sizeof(queue));
433     ready->count = 0;
434     ready->head = NULL;
435     ready->tail = NULL;
436 
437     blocked = sys_alloc_mem(sizeof(queue));
438     blocked->count = 0;
439     blocked->head = NULL;
440     blocked->tail = NULL;
441 
442     suspendedReady = sys_alloc_mem(sizeof(queue));
443     suspendedReady->count = 0;
444     suspendedReady->head = NULL;
445     suspendedReady->tail = NULL;
446 
447     suspendedBlocked = sys_alloc_mem(sizeof(queue));
448     suspendedBlocked->count = 0;
449     suspendedBlocked->head = NULL;
450     suspendedBlocked->tail = NULL;
451 }
```

5.25.2.3 findPCB()

```
PCB* findPCB (
            char * processName )
```

Definition at line 82 of file R2_Internal_Functions_And_Structures.c.

```
83 {
84     // ANASTASE WILL PROGRAM THIS FUNCTION
85 
86     //findPCB() will search all queues for a process with a given name.
87 
88     if (strlen(processName) > 20)
89     {
90 
91         printMessage("Invalid process name.\n");
92         return NULL;
93         //return cz we have to stop if the process name is too long
94     }
95     else
96     {
97         PCB *tempPCB = ready->head;
98         int value = 0;
99         while (value < ready->count)
100         {
101             if (strcmp(tempPCB->processName, processName) == 0)
102             {
103                 return tempPCB;
104             }
105             else
106             {
107                 tempPCB = tempPCB->nextPCB;
108                 value++;
109             }
110         }
111 
112         tempPCB = blocked->head;
113         value = 0;
114         while (value < blocked->count)
115         {
116             if (strcmp(tempPCB->processName, processName) == 0)
117             {
118                 return tempPCB;
119             }
120             else
121             {
122                 tempPCB = tempPCB->nextPCB;
123                 value++;
124             }
125         }
126 
127         tempPCB = suspendedBlocked->head;
128         value = 0;
129         while (value < suspendedBlocked->count)
130         {
131             if (strcmp(tempPCB->processName, processName) == 0)
132             {
133                 return tempPCB;
134             }
135             else
136             {
137                 tempPCB = tempPCB->nextPCB;
138                 value++;
139             }
140         }
141 
142         tempPCB = suspendedReady->head;
143         value = 0;
144         while (value < suspendedReady->count)
145         {
146             if (strcmp(tempPCB->processName, processName) == 0)
147             {
148                 return tempPCB;
149             }
150             else
151             {
152                 tempPCB = tempPCB->nextPCB;
153                 value++;
154             }
155         }
156 
157         return NULL;
158     }
159 }
```

5.25.2.4 freePCB()

```
int freePCB (
            PCB * PCB_to_free )
```

Definition at line 42 of file R2_Internal_Functions_And_Structures.c.

```
43 {
44     // ANASTASE WILL PROGRAM THIS FUNCTION
45
46     //freePCB() will use sys_free_mem() to free all memory associated with a given PCB (the stack, the
       PCB itself, etc.)
47
48     return sys_free_mem(PCB_to_free);
49 }
```

5.25.2.5 getBlocked()

```
queue* getBlocked ( )
```

Definition at line 458 of file R2_Internal_Functions_And_Structures.c.

```
459 {
460     return blocked;
461 }
```

5.25.2.6 getReady()

```
queue* getReady ( )
```

Definition at line 453 of file R2_Internal_Functions_And_Structures.c.

```
454 {
455     return ready;
456 }
```

5.25.2.7 getSuspendedBlocked()

```
queue* getSuspendedBlocked ( )
```

Definition at line 468 of file R2_Internal_Functions_And_Structures.c.

```
469 {
470     return suspendedBlocked;
471 }
```

5.25.2.8 getSuspendedReady()

```
queue* getSuspendedReady ( )
```

Definition at line 463 of file R2_Internal_Functions_And_Structures.c.

```
464 {
465     return suspendedReady;
466 }
```

5.25.2.9 insertPCB()

```
void insertPCB (
            PCB * PCB_to_insert )
```

Definition at line 161 of file R2_Internal_Functions_And_Structures.c.

```
162 {
163     //BENJAMIN WILL PROGRAM THIS FUNCTION
164
165     //insertPCB() will insert a PCB into the appropriate queue.
166     //Note: The ready queue is a priority queue and the blocked queue is a FIFO queue.
167
168     if (PCB_to_insert->runningStatus == 0 && PCB_to_insert->suspendedStatus == 1)
169     { // Insert into ready queue
170         PCB *tempPtr = ready->head;
171
172         if (tempPtr != NULL)
173         {
174             int temp = 0;
175             while (temp < ready->count)
176             {
177                 if (PCB_to_insert->priority > ready->head->priority)
178                 { // insert at head
179                     PCB_to_insert->nextPCB = tempPtr;
180                     tempPtr->prevPCB = PCB_to_insert;
181                     ready->head = PCB_to_insert;
182                     ready->count++;
183                     break;
184                 }
185                 else if (PCB_to_insert->priority <= ready->tail->priority)
186                 { // insert at tail
187                     ready->tail->nextPCB = PCB_to_insert;
188                     PCB_to_insert->prevPCB = ready->tail;
189                     ready->tail = PCB_to_insert;
190                     ready->count++;
191                     break;
192                 }
193                 else if (PCB_to_insert->priority > tempPtr->priority)
194                 { // insert at middle
195                     PCB *prevPtr = tempPtr->prevPCB;
196
197                     prevPtr->nextPCB = PCB_to_insert;
198
199                     PCB_to_insert->prevPCB = prevPtr;
200                     PCB_to_insert->nextPCB = tempPtr;
201
202                     tempPtr->prevPCB = PCB_to_insert;
203
204                     ready->count++;
205                     break;
206                 }
207                 else
208                 { // move tempPtr through the queue
209                     tempPtr = tempPtr->nextPCB;
210                 }
211                 temp++;
212             }
213         }
214         else
215         {
216             ready->head = PCB_to_insert;
217             ready->tail = PCB_to_insert;
218             ready->count++;
219         }
220     }
221     else if (PCB_to_insert->runningStatus == 0 && PCB_to_insert->suspendedStatus == 0)
222     { // Insert into suspended ready queue
223         PCB *tempPtr = suspendedReady->head;
224
225         if (tempPtr != NULL)
226         {
227             int temp = 0;
228             while (temp < suspendedReady->count)
229             {
230                 if (PCB_to_insert->priority > suspendedReady->head->priority)
231                 { // insert at head
232                     PCB_to_insert->nextPCB = tempPtr;
233                     tempPtr->prevPCB = PCB_to_insert;
234                     suspendedReady->head = PCB_to_insert;
235                     suspendedReady->count++;
236                     break;
237                 }
238                 else if (PCB_to_insert->priority <= suspendedReady->tail->priority)
239                 { // insert at tail
```

```
240
241                     suspendedReady->tail->nextPCB = PCB_to_insert;
242                     PCB_to_insert->prevPCB = suspendedReady->tail;
243                     suspendedReady->tail = PCB_to_insert;
244                     suspendedReady->count++;
245                     break;
246                 }
247                 else if (PCB_to_insert->priority > tempPtr->priority)
248                 { // insert at middle
249                     PCB *prevPtr = tempPtr->prevPCB;
250
251                     prevPtr->nextPCB = PCB_to_insert;
252
253                     PCB_to_insert->prevPCB = prevPtr;
254                     PCB_to_insert->nextPCB = tempPtr;
255
256                     tempPtr->prevPCB = PCB_to_insert;
257
258                     ready->count++;
259                     break;
260                 }
261                 else
262                 { // move tempPtr through the queue
263                     tempPtr = tempPtr->nextPCB;
264                 }
265                 temp++;
266             }
267         }
268         else
269         {
270             suspendedReady->count++;
271             suspendedReady->head = PCB_to_insert;
272             suspendedReady->tail = PCB_to_insert;
273         }
274     }
275     else if (PCB_to_insert->runningStatus == -1 && PCB_to_insert->suspendedStatus == 1)
276     { // Insert into blocked queue
277         if (blocked->head != NULL)
278         {
279             blocked->tail->nextPCB = PCB_to_insert;
280             PCB_to_insert->prevPCB = blocked->tail;
281             blocked->tail = PCB_to_insert;
282             blocked->count++;
283         }
284         else
285         {
286             blocked->head = PCB_to_insert;
287             blocked->tail = PCB_to_insert;
288             blocked->count++;
289         }
290     }
291     else if (PCB_to_insert->runningStatus == -1 && PCB_to_insert->suspendedStatus == 0)
292     { // Insert into suspended blocked queue
293         if (suspendedBlocked->head != NULL)
294         {
295             suspendedBlocked->tail->nextPCB = PCB_to_insert;
296             PCB_to_insert->prevPCB = suspendedBlocked->tail;
297             suspendedBlocked->tail = PCB_to_insert;
298             suspendedBlocked->count++;
299         }
300         else
301         {
302             suspendedBlocked->head = PCB_to_insert;
303             suspendedBlocked->tail = PCB_to_insert;
304             suspendedBlocked->count++;
305         }
306     }
307 }
```

5.25.2.10 removePCB()

```
int removePCB (
            PCB * PCB_to_remove )
```

Definition at line 309 of file R2_Internal_Functions_And_Structures.c.

```
310 {
311     //BENJAMIN WILL PROGRAM THIS FUNCTION
312
```

```
313     //removePCB() will remove a PCB from the queue in which it is currently stored.
314
315     if (PCB_to_remove == NULL)
316     {
317         return 1;
318     }
319     else if (PCB_to_remove == ready->head)
320     {
321         //PCB *removedNext = PCB_to_remove->nextPCB;
322
323         ready->head = PCB_to_remove->nextPCB;
324         ready->head->prevPCB = NULL;
325         PCB_to_remove->nextPCB = NULL;
326         ready->count--;
327         return 0;
328     }
329     else if (PCB_to_remove == blocked->head)
330     {
331         PCB *removedNext = PCB_to_remove->nextPCB;
332         blocked->head = removedNext;
333         removedNext->prevPCB = NULL;
334         PCB_to_remove->nextPCB = NULL;
335         blocked->count--;
336         return 0;
337     }
338     else if (PCB_to_remove == suspendedReady->head)
339     {
340         PCB *removedNext = PCB_to_remove->nextPCB;
341
342         suspendedReady->head = removedNext;
343         removedNext->prevPCB = NULL;
344         PCB_to_remove->nextPCB = NULL;
345         suspendedReady->count--;
346         return 0;
347     }
348     else if (PCB_to_remove == suspendedBlocked->head)
349     {
350         PCB *removedNext = PCB_to_remove->nextPCB;
351
352         suspendedBlocked->head = removedNext;
353         removedNext->prevPCB = NULL;
354         PCB_to_remove->nextPCB = NULL;
355         suspendedBlocked->count--;
356         return 0;
357     }
358     else if (PCB_to_remove == ready->tail)
359     {
360         PCB *removedPrev = PCB_to_remove->prevPCB;
361
362         ready->tail = removedPrev;
363         removedPrev->nextPCB = NULL;
364         PCB_to_remove->prevPCB = NULL;
365         ready->count--;
366         return 0;
367     }
368     else if (PCB_to_remove == blocked->tail)
369     {
370         PCB *removedPrev = PCB_to_remove->prevPCB;
371
372         blocked->tail = removedPrev;
373         removedPrev->nextPCB = NULL;
374         PCB_to_remove->prevPCB = NULL;
375         blocked->count--;
376         return 0;
377     }
378     else if (PCB_to_remove == suspendedReady->tail)
379     {
380         PCB *removedPrev = PCB_to_remove->prevPCB;
381
382         suspendedReady->tail = removedPrev;
383         removedPrev->nextPCB = NULL;
384         PCB_to_remove->prevPCB = NULL;
385         suspendedReady->count--;
386         return 0;
387     }
388     else if (PCB_to_remove == suspendedBlocked->tail)
389     {
390         PCB *removedPrev = PCB_to_remove->prevPCB;
391
392         suspendedBlocked->tail = removedPrev;
393         removedPrev->nextPCB = NULL;
394         PCB_to_remove->prevPCB = NULL;
395         suspendedBlocked->count--;
396         return 0;
397     }
398     else
399     {
```

```
400         // PCB *tempPrev = PCB_to_remove->prevPCB;
401         // PCB *tempNext = PCB_to_remove->nextPCB;
402
403         PCB_to_remove->prevPCB->nextPCB = PCB_to_remove->nextPCB;
404         PCB_to_remove->nextPCB->prevPCB = PCB_to_remove->prevPCB;
405
406         PCB_to_remove->nextPCB = NULL;
407         PCB_to_remove->prevPCB = NULL;
408
409         if (PCB_to_remove->runningStatus == 0 && PCB_to_remove->suspendedStatus == 1)
410         {
411             ready->count--;
412         }
413         else if (PCB_to_remove->runningStatus == -1 && PCB_to_remove->suspendedStatus == 1)
414         {
415             blocked->count--;
416         }
417         else if (PCB_to_remove->runningStatus == 0 && PCB_to_remove->suspendedStatus == 0)
418         {
419             suspendedReady->count--;
420         }
421         else if (PCB_to_remove->runningStatus == -1 && PCB_to_remove->suspendedStatus == 0)
422         {
423             suspendedBlocked->count--;
424         }
425
426         return 0;
427     }
428 }
```

5.25.2.11 setupPCB()

```
PCB* setupPCB (
              char * processName,
              unsigned char processClass,
              int processPriority )
```

Definition at line 51 of file R2_Internal_Functions_And_Structures.c.

```
52 {
53     //COLTON WILL PROGRAM THIS FUNCTION
54
55     //setupPcb() will call allocatePCB() to create an empty PCB, initializes the PCB information, sets
       the PCB state to ready, not suspended.
56
57     PCB *returnedPCB = allocatePCB();
58
59     if (findPCB(processName)->processName == processName)
60     {
61         printMessage("There is already a PCB with this name.\n");
62
63         returnedPCB = NULL;
64     }
65     else
66     {
67
68         strcpy(returnedPCB->processName, processName);
69         returnedPCB->processClass = processClass;
70         returnedPCB->priority = processPriority;
71         returnedPCB->runningStatus = 0;
72         returnedPCB->suspendedStatus = 1;
73         returnedPCB->stackBase = returnedPCB->stack;
74         returnedPCB->stackTop = returnedPCB->stack + 1024 - sizeof(context);
75         returnedPCB->nextPCB = NULL;
76         returnedPCB->prevPCB = NULL;
77     }
78
79     return returnedPCB;
80 }
```

5.26 modules/R2/R2commands.c File Reference

```
#include <string.h>
#include "../mpx_supt.h"
```

```
#include "../utilities.h"
#include "R2_Internal_Functions_And_Structures.h"
#include "R2commands.h"
#include <core/serial.h>
```

Functions

- void createPCB (char *processName, char processClass, int processPriority)
- void deletePCB (char *processName)
- void blockPCB (char *processName)
- void unblockPCB (char *processName)
- void suspendPCB (char *processName)
- void resumePCB (char *processName)
- void setPCBPriority (char *processName, int newProcessPriority)
- void showPCB (char *processName)
- void showQueue (PCB *pcb, int count)
- void showReady ()
- void showSuspendedReady ()
- void showSuspendedBlocked ()
- void showBlocked ()
- void showAll ()

5.26.1 Function Documentation

5.26.1.1 blockPCB()

```
void blockPCB (
            char * processName )
```

Definition at line 98 of file R2commands.c.

```
99 { // ANASTASE WILL PROGRAM THIS FUNCTION
100
101     // find pcb and validate process name
102     PCB *pcb_to_block = findPCB(processName);
103
104     if (pcb_to_block != NULL)
105     {
106         pcb_to_block->runningStatus = -1; // blocked
107         removePCB(pcb_to_block);
108         insertPCB(pcb_to_block);
109
110         printMessage("The PCB was successfully blocked!\n");
111     }
112 }
```

5.26.1.2 createPCB()

```
void createPCB (
            char * processName,
            char processClass,
            int processPriority )
```

Definition at line 12 of file R2commands.c.

```
13 { // BENJAMIN WILL PROGRAM THIS FUNCTION
14     /*
15     The createPCB command will call setupPCB() and insert the PCB in the appropriate queue
16     */
17     /*
18     Error Checking:
19     Name must be unique and valid.
20     Class must be valid.
21     Priority must be valid.
22     */
23
24     if (findPCB(processName) != NULL || strlen(processName) > 20)
25     { // Check if the process has a unique name, and if it has a valid name.
26         printMessage("The PCB could not be created as it either does not have a unique name or the name
       is longer than 20 characters!\n");
27     }
28     else if (processClass != 'a' && processClass != 's')
29     { // Check if the process has a valid class.
30         printMessage("The PCB could not be created as it does not have a valid class!\n");
31     }
32     else if (processPriority < 0 || processPriority > 9)
33     { // Check if the process has a valid priority.
34         printMessage("The PCB could not be created as it does not have a valid priority!\n");
35     }
36     else
37     { // Make the PCB
38         PCB *createdPCB = setupPCB(processName, processClass, processPriority);
39
40         printMessage("The PCB was created!\n");
41
42         insertPCB(createdPCB);
43     }
44 }
```

5.26.1.3 deletePCB()

```
void deletePCB (
            char * processName )
```

Definition at line 46 of file R2commands.c.

```
47 { // BENJAMIN WILL PROGRAM THIS FUNCTION
48     /*
49     The deletePCB command will remove a PCB from the appropriate queue and then free all associated
       memory.
50     This method will need to find the pcb, unlink it from the appropriate queue, and then free it.
51     */
52     /*
53     Error Checking:
54     Name must be valid.
55     */
56
57     if (strlen(processName) > 20)
58     { // Check if the process has a valid name.
59         printMessage("The PCB could not be deleted as the name is longer than 20 characters!\n");
60     }
61
62     PCB *PCB_to_delete = findPCB(processName);
63
64     if (PCB_to_delete == NULL)
65     {
66         printMessage("The PCB you want to remove does not exist\n");
67     }
68     else if (strcmp(processName, "infinite") == 0 && PCB_to_delete->suspendedStatus != 0)
69     {
70         printMessage("In order to delete the infinite process it must be suspended first.\n");
71     }
```

```
72      else if (PCB_to_delete->processClass == 's')
73      {
74          printMessage("You do not have permission to delete system processes!\n");
75      }
76      else
77      {
78          int removed = removePCB(PCB_to_delete);
79          if (removed == 1)
80          {
81              printMessage("The PCB could not be unlinked.\n");
82          }
83          else
84          {
85              int result = sys_free_mem(PCB_to_delete);
86              if (result == -1)
87              {
88                  // printMessage("The PCB could not be successfully deleted\n");
89              }
90              else
91              {
92                  printMessage("The desired PCB was deleted\n");
93              }
94          }
95      }
96  }
```

5.26.1.4 resumePCB()

```
void resumePCB (
              char * processName )
```

Definition at line 168 of file R2commands.c.

```
169 { // COLTON WILL PROGRAM THIS FUNCTION
170     /*
171     Places a PCB in the not suspended state and reinserts it into the appropriate queue
172     */
177
178     PCB *PCBtoResume = findPCB(processName);
179
180     if (PCBtoResume == NULL || strlen(processName) > 20)
181     {
182         printMessage("This is not a valid name.\n");
183     }
184     else
185     {
186         removePCB(PCBtoResume);
187         PCBtoResume->suspendedStatus = 1;
188         insertPCB(PCBtoResume);
189
190         printMessage("The PCB was successfully resumed!\n");
191     }
192 }
```

5.26.1.5 setPCBPriority()

```
void setPCBPriority (
              char * processName,
              int newProcessPriority )
```

Definition at line 194 of file R2commands.c.

```
195 { // ANASTASE WILL PROGRAM THIS FUNCTION
196
197     // Sets a PCB's priority and reinserts the process into the correct place in the correct queue
198
199     /*
200     Error Checking:
201     Name must be valid.
202     newPriority
203     */
```

```
204
205     // find the process and validate the name
206     PCB *tempPCB = findPCB(processName);
207
208     if ((tempPCB != NULL) && (newProcessPriority >= 0) && (newProcessPriority < 10))
209     {
210         tempPCB->priority = newProcessPriority;
211         removePCB(tempPCB);
212         insertPCB(tempPCB);
213
214         printMessage("The PCB's priority was successfully changed!\n");
215     }
216 }
```

5.26.1.6 showAll()

```
void showAll ( )
```

Definition at line 438 of file R2commands.c.

```
439 { // COLTON WILL PROGRAM THIS FUNCTION
440     /*
441     Displays the following information for each PCB in the ready and blocked queues:
442         Process Name
443         Class
444         State
445         Suspended Status
446         Priority
447     */
448     /*
449     Error Checking:
450     None
451     */
452     showReady();
453     printMessage("\n");
454
455     showSuspendedReady();
456     printMessage("\n");
457
458     showBlocked();
459     printMessage("\n");
460
461     showSuspendedBlocked();
462     printMessage("\n");
463 }
```

5.26.1.7 showBlocked()

```
void showBlocked ( )
```

Definition at line 418 of file R2commands.c.

```
419 { // ANASTASE WILL PROGRAM THIS FUNCTION
420     /*
421     Displays the following information for each PCB in the blocked queue:
422         Process Name
423         Class
424         State
425         Suspended Status
426         Priority
427         HEAD
428     */
429     /*
430     Error Checking:
431     None
432     */
433
434     printMessage("The blocked queue:\n");
435     showQueue(getBlocked()->head, getBlocked()->count);
436 }
```

5.26.1.8 showPCB()

```
void showPCB (
            char * processName )
```

Definition at line 218 of file R2commands.c.

```
219 { // BENJAMIN WILL PROGRAM THIS FUNCTION
220     /*
221     Displays the following information for a PCB:
222         Process Name
223         Class
224         State
225         Suspended Status
226         Priority
227     */
228
229     /*
230     Error Checking:
231     Name must be valid.
232     */
233
234     if (strlen(processName) > 20)
235     { // Check if the process has a valid name.
236         printMessage("The PCB could not be shown as the name is longer than 20 characters!\n");
237     }
238     else
239     {
240         PCB *PCB_to_show = findPCB(processName);
241
242         if (PCB_to_show == NULL)
243         { // Check to see if the PCB exists.
244             printMessage("The PCB could not be shown, as it does not exist!\n");
245         }
246         else
247         {
248             // Print out the PCB name.
249             printMessage("The process name is: ");
250             int length = strlen(PCB_to_show->processName);
251             sys_req(WRITE, DEFAULT_DEVICE, PCB_to_show->processName, &length);
252             printMessage("\n");
253
254             // Print out PCB class
255             printMessage("The process class is: ");
256
257             if (PCB_to_show->processClass == 'a')
258             {
259                 printMessage("application.\n");
260             }
261             else
262             {
263                 printMessage("system.\n");
264             }
265
266             // Print out the PCB state
267
268             if (PCB_to_show->runningStatus == 0)
269             { // The process is ready.
270                 printMessage("The process is ready!\n");
271             }
272             else if (PCB_to_show->runningStatus == -1)
273             { // The process is blocked.
274                 printMessage("The process is blocked!\n");
275             }
276             else if (PCB_to_show->runningStatus == 1)
277             { // The process is running.
278                 printMessage("The process is running!\n");
279             }
280
281             // Print out the PCB suspended status
282
283             if (PCB_to_show->suspendedStatus == 0)
284             { // The process is suspended
285                 printMessage("The process is suspended!\n");
286             }
287             else if (PCB_to_show->suspendedStatus == 1)
288             { // The process is not suspended
289                 printMessage("The process is not suspended!\n");
290             }
291
292             // Print out the PCB priority
293             switch (PCB_to_show->priority)
294             {
295             case 0:
296                 printMessage("The process priority is 0!\n");
```

```
297                 break;
298
299             case 1:
300                 printMessage("The process priority is 1!\n");
301                 break;
302
303             case 2:
304                 printMessage("The process priority is 2!\n");
305                 break;
306
307             case 3:
308                 printMessage("The process priority is 3!\n");
309                 break;
310
311             case 4:
312                 printMessage("The process priority is 4!\n");
313                 break;
314
315             case 5:
316                 printMessage("The process priority is 5!\n");
317                 break;
318
319             case 6:
320                 printMessage("The process priority is 6!\n");
321                 break;
322
323             case 7:
324                 printMessage("The process priority is 7!\n");
325                 break;
326
327             case 8:
328                 printMessage("The process priority is 8!\n");
329                 break;
330
331             case 9:
332                 printMessage("The process priority is 9!\n");
333                 break;
334
335             default:
336                 break;
337             }
338         }
339     }
340 }
```

5.26.1.9 showQueue()

```
void showQueue (
              PCB * pcb,
              int count )
```

Definition at line 342 of file R2commands.c.

```
343 {
344     if (count == 0)
345     {
346         // the queue is empty
347         printMessage("The queue is empty.\n");
348         return;
349     }
350     // The queue is not empty
351
352     int value;
353     for (value = 0; value < count; value++)
354     {
355         // Print out the process
356         showPCB(pcb->processName);
357         pcb = pcb->nextPCB;
358     }
359 }
```

5.26.1.10 showReady()

```
void showReady ( )
```

Definition at line 361 of file R2commands.c.

```
362 { // COLTON WILL PROGRAM THIS FUNCTION
363     /*
364     Displays the following information for each PCB in the ready queue:
365         Process Name
366         Class
367         State
368         Suspended Status
369         Priority
370     */
371     /*
372     Error Checking:
373     None
374     */
375 
376     printMessage("The ready queue:\n");
377     showQueue(getReady()->head, getReady()->count);
378 }
```

5.26.1.11 showSuspendedBlocked()

```
void showSuspendedBlocked ( )
```

Definition at line 399 of file R2commands.c.

```
400 { // COLTON WILL PROGRAM THIS FUNCTION
401     /*
402     Displays the following information for each PCB in the suspended blocked queue:
403         Process Name
404         Class
405         State
406         Suspended Status
407         Priority
408     */
409     /*
410     Error Checking:
411     None
412     */
413 
414     printMessage("The suspended blocked queue:\n");
415     showQueue(getSuspendedBlocked()->head, getSuspendedBlocked()->count);
416 }
```

5.26.1.12 showSuspendedReady()

```
void showSuspendedReady ( )
```

Definition at line 380 of file R2commands.c.

```
381 { // COLTON WILL PROGRAM THIS FUNCTION
382     /*
383     Displays the following information for each PCB in the suspended ready queue:
384         Process Name
385         Class
386         State
387         Suspended Status
388         Priority
389     */
390     /*
391     Error Checking:
392     None
393     */
394 
395     printMessage("The suspended ready queue:\n");
396     showQueue(getSuspendedReady()->head, getSuspendedReady()->count);
397 }
```

5.26.1.13 suspendPCB()

```
void suspendPCB (
            char * processName )
```

Definition at line 138 of file R2commands.c.

```
139 { // COLTON WILL PROGRAM THIS FUNCTION
140     /*
141     Places a PCB in the suspended state and reinserts it into the appropriate queue
142     */
147
148     PCB *PCBtoSuspend = findPCB(processName);
149
150     if (PCBtoSuspend == NULL || strlen(processName) > 20)
151     {
152         printMessage("This is not a valid name.\n");
153     }
154     else if (PCBtoSuspend->processClass == 's')
155     {
156         printMessage("You do not have permission to suspend system processes!\n");
157     }
158     else
159     {
160         removePCB(PCBtoSuspend);
161         PCBtoSuspend->suspendedStatus = 0;
162         insertPCB(PCBtoSuspend);
163
164         printMessage("The PCB was successfully suspended!\n");
165     }
166 }
```

5.26.1.14 unblockPCB()

```
void unblockPCB (
            char * processName )
```

Definition at line 114 of file R2commands.c.

```
115 { // ANASTASE WILL PROGRAM THIS FUNCTION
116
117     /*
118     Places a PCB in the unblocked state and reinserts it into the appropriate queue.
119     */
120     /*
121     Error Checking:
122     Name must be valid.
123
124     */
125
126     PCB *pcb_to_unblock = findPCB(processName);
127     if (pcb_to_unblock != NULL)
128     {
129         pcb_to_unblock->runningStatus = 0; // ready
130         removePCB(pcb_to_unblock);         // is this the right place to put that function?
131         insertPCB(pcb_to_unblock);
132
133         printMessage("The PCB was successfully unblocked!\n");
134     }
135 }
```

5.27 modules/R2/R2commands.h File Reference

Functions

- void createPCB (char *processName, char processClass, int processPriority)
- void deletePCB (char *processName)
- void blockPCB (char *processName)

- void unblockPCB (char *processName)
- void suspendPCB (char *processName)
- void resumePCB (char *processName)
- void setPCBPriority (char *processName, int newProcessPriority)
- void showPCB (char *processName)
- void showReady ()
- void showSuspendedBlocked ()
- void showSuspendedReady ()
- void showBlocked ()
- void showAll ()

5.27.1 Function Documentation

5.27.1.1 blockPCB()

```
void blockPCB (
            char * processName )
```

Definition at line 98 of file R2commands.c.

```
99 { // ANASTASE WILL PROGRAM THIS FUNCTION
100
101     // find pcb and validate process name
102     PCB *pcb_to_block = findPCB(processName);
103
104     if (pcb_to_block != NULL)
105     {
106         pcb_to_block->runningStatus = -1; // blocked
107         removePCB(pcb_to_block);
108         insertPCB(pcb_to_block);
109
110         printMessage("The PCB was successfully blocked!\n");
111     }
112 }
```

5.27.1.2 createPCB()

```
void createPCB (
            char * processName,
            char processClass,
            int processPriority )
```

Definition at line 12 of file R2commands.c.

```
13 { // BENJAMIN WILL PROGRAM THIS FUNCTION
14     /*
15     The createPCB command will call setupPCB() and insert the PCB in the appropriate queue
16     */
17     /*
18     Error Checking:
19     Name must be unique and valid.
20     Class must be valid.
21     Priority must be valid.
22     */
23
24     if (findPCB(processName) != NULL || strlen(processName) > 20)
25     { // Check if the process has a unique name, and if it has a valid name.
26         printMessage("The PCB could not be created as it either does not have a unique name or the name
       is longer than 20 characters!\n");
27     }
28     else if (processClass != 'a' && processClass != 's')
```

```
29     { // Check if the process has a valid class.
30         printMessage("The PCB could not be created as it does not have a valid class!\n");
31     }
32     else if (processPriority < 0 || processPriority > 9)
33     { // Check if the process has a valid priority.
34         printMessage("The PCB could not be created as it does not have a valid priority!\n");
35     }
36     else
37     { // Make the PCB
38         PCB *createdPCB = setupPCB(processName, processClass, processPriority);
39 
40         printMessage("The PCB was created!\n");
41 
42         insertPCB(createdPCB);
43     }
44 }
```

5.27.1.3 deletePCB()

```
void deletePCB (
            char * processName )
```

Definition at line 46 of file R2commands.c.

```
47 { // BENJAMIN WILL PROGRAM THIS FUNCTION
48     /*
49     The deletePCB command will remove a PCB from the appropriate queue and then free all associated
       memory.
50     This method will need to find the pcb, unlink it from the appropriate queue, and then free it.
51     */
52     /*
53     Error Checking:
54     Name must be valid.
55     */
56 
57     if (strlen(processName) > 20)
58     { // Check if the process has a valid name.
59         printMessage("The PCB could not be deleted as the name is longer than 20 characters!\n");
60     }
61 
62     PCB *PCB_to_delete = findPCB(processName);
63 
64     if (PCB_to_delete == NULL)
65     {
66         printMessage("The PCB you want to remove does not exist\n");
67     }
68     else if (strcmp(processName, "infinite") == 0 && PCB_to_delete->suspendedStatus != 0)
69     {
70         printMessage("In order to delete the infinite process it must be suspended first.\n");
71     }
72     else if (PCB_to_delete->processClass == 's')
73     {
74         printMessage("You do not have permission to delete system processes!\n");
75     }
76     else
77     {
78         int removed = removePCB(PCB_to_delete);
79         if (removed == 1)
80         {
81             printMessage("The PCB could not be unlinked.\n");
82         }
83         else
84         {
85             int result = sys_free_mem(PCB_to_delete);
86             if (result == -1)
87             {
88                 // printMessage("The PCB could not be successfully deleted\n");
89             }
90             else
91             {
92                 printMessage("The desired PCB was deleted\n");
93             }
94         }
95     }
96 }
```

5.27.1.4 resumePCB()

```
void resumePCB (
            char * processName )
```

Definition at line 168 of file R2commands.c.

```
169 { // COLTON WILL PROGRAM THIS FUNCTION
170     /*
171     Places a PCB in the not suspended state and reinserts it into the appropriate queue
172     */
177
178     PCB *PCBtoResume = findPCB(processName);
179
180     if (PCBtoResume == NULL || strlen(processName) > 20)
181     {
182         printMessage("This is not a valid name.\n");
183     }
184     else
185     {
186         removePCB(PCBtoResume);
187         PCBtoResume->suspendedStatus = 1;
188         insertPCB(PCBtoResume);
189
190         printMessage("The PCB was successfully resumed!\n");
191     }
192 }
```

5.27.1.5 setPCBPriority()

```
void setPCBPriority (
            char * processName,
            int newProcessPriority )
```

Definition at line 194 of file R2commands.c.

```
195 { // ANASTASE WILL PROGRAM THIS FUNCTION
196
197     // Sets a PCB's priority and reinserts the process into the correct place in the correct queue
198
199     /*
200     Error Checking:
201     Name must be valid.
202     newPriority
203     */
204
205     // find the process and validate the name
206     PCB *tempPCB = findPCB(processName);
207
208     if ((tempPCB != NULL) && (newProcessPriority >= 0) && (newProcessPriority < 10))
209     {
210         tempPCB->priority = newProcessPriority;
211         removePCB(tempPCB);
212         insertPCB(tempPCB);
213
214         printMessage("The PCB's priority was successfully changed!\n");
215     }
216 }
```

5.27.1.6 showAll()

```
void showAll ( )
```

Definition at line 438 of file R2commands.c.

```
439 { // COLTON WILL PROGRAM THIS FUNCTION
440     /*
441     Displays the following information for each PCB in the ready and blocked queues:
442         Process Name
```

```
443         Class
444         State
445         Suspended Status
446         Priority
447     */
448     /*
449     Error Checking:
450     None
451     */
452     showReady();
453     printMessage("\n");
454 
455     showSuspendedReady();
456     printMessage("\n");
457 
458     showBlocked();
459     printMessage("\n");
460 
461     showSuspendedBlocked();
462     printMessage("\n");
463 }
```

5.27.1.7 showBlocked()

```
void showBlocked ( )
```

Definition at line 418 of file R2commands.c.

```
419 { // ANASTASE WILL PROGRAM THIS FUNCTION
420     /*
421     Displays the following information for each PCB in the blocked queue:
422         Process Name
423         Class
424         State
425         Suspended Status
426         Priority
427         HEAD
428     */
429     /*
430     Error Checking:
431     None
432     */
433 
434     printMessage("The blocked queue:\n");
435     showQueue(getBlocked()->head, getBlocked()->count);
436 }
```

5.27.1.8 showPCB()

```
void showPCB (
             char * processName )
```

Definition at line 218 of file R2commands.c.

```
219 { // BENJAMIN WILL PROGRAM THIS FUNCTION
220     /*
221     Displays the following information for a PCB:
222         Process Name
223         Class
224         State
225         Suspended Status
226         Priority
227     */
228 
229     /*
230     Error Checking:
231     Name must be valid.
232     */
233 
234     if (strlen(processName) > 20)
235     { // Check if the process has a valid name.
236         printMessage("The PCB could not be shown as the name is longer than 20 characters!\n");
```

```
237     }
238     else
239     {
240         PCB *PCB_to_show = findPCB(processName);
241 
242         if (PCB_to_show == NULL)
243         { // Check to see if the PCB exists.
244             printMessage("The PCB could not be shown, as it does not exist!\n");
245         }
246         else
247         {
248             // Print out the PCB name.
249             printMessage("The process name is: ");
250             int length = strlen(PCB_to_show->processName);
251             sys_req(WRITE, DEFAULT_DEVICE, PCB_to_show->processName, &length);
252             printMessage("\n");
253 
254             // Print out PCB class
255             printMessage("The process class is: ");
256 
257             if (PCB_to_show->processClass == 'a')
258             {
259                 printMessage("application.\n");
260             }
261             else
262             {
263                 printMessage("system.\n");
264             }
265 
266             // Print out the PCB state
267 
268             if (PCB_to_show->runningStatus == 0)
269             { // The process is ready.
270                 printMessage("The process is ready!\n");
271             }
272             else if (PCB_to_show->runningStatus == -1)
273             { // The process is blocked.
274                 printMessage("The process is blocked!\n");
275             }
276             else if (PCB_to_show->runningStatus == 1)
277             { // The process is running.
278                 printMessage("The process is running!\n");
279             }
280 
281             // Print out the PCB suspended status
282 
283             if (PCB_to_show->suspendedStatus == 0)
284             { // The process is suspended
285                 printMessage("The process is suspended!\n");
286             }
287             else if (PCB_to_show->suspendedStatus == 1)
288             { // The process is not suspended
289                 printMessage("The process is not suspended!\n");
290             }
291 
292             // Print out the PCB priority
293             switch (PCB_to_show->priority)
294             {
295             case 0:
296                 printMessage("The process priority is 0!\n");
297                 break;
298 
299             case 1:
300                 printMessage("The process priority is 1!\n");
301                 break;
302 
303             case 2:
304                 printMessage("The process priority is 2!\n");
305                 break;
306 
307             case 3:
308                 printMessage("The process priority is 3!\n");
309                 break;
310 
311             case 4:
312                 printMessage("The process priority is 4!\n");
313                 break;
314 
315             case 5:
316                 printMessage("The process priority is 5!\n");
317                 break;
318 
319             case 6:
320                 printMessage("The process priority is 6!\n");
321                 break;
322 
323             case 7:
```

```
324                 printMessage("The process priority is 7!\n");
325                 break;
326 
327             case 8:
328                 printMessage("The process priority is 8!\n");
329                 break;
330 
331             case 9:
332                 printMessage("The process priority is 9!\n");
333                 break;
334 
335             default:
336                 break;
337             }
338         }
339     }
340 }
```

5.27.1.9 showReady()

```
void showReady ( )
```

Definition at line 361 of file R2commands.c.

```
362 { // COLTON WILL PROGRAM THIS FUNCTION
363     /*
364     Displays the following information for each PCB in the ready queue:
365         Process Name
366         Class
367         State
368         Suspended Status
369         Priority
370     */
371     /*
372     Error Checking:
373     None
374     */
375 
376     printMessage("The ready queue:\n");
377     showQueue(getReady()->head, getReady()->count);
378 }
```

5.27.1.10 showSuspendedBlocked()

```
void showSuspendedBlocked ( )
```

Definition at line 399 of file R2commands.c.

```
400 { // COLTON WILL PROGRAM THIS FUNCTION
401     /*
402     Displays the following information for each PCB in the suspended blocked queue:
403         Process Name
404         Class
405         State
406         Suspended Status
407         Priority
408     */
409     /*
410     Error Checking:
411     None
412     */
413 
414     printMessage("The suspended blocked queue:\n");
415     showQueue(getSuspendedBlocked()->head, getSuspendedBlocked()->count);
416 }
```

5.27.1.11 showSuspendedReady()

```
void showSuspendedReady ( )
```

Definition at line 380 of file R2commands.c.

```
381 { // COLTON WILL PROGRAM THIS FUNCTION
382     /*
383     Displays the following information for each PCB in the suspended ready queue:
384         Process Name
385         Class
386         State
387         Suspended Status
388         Priority
389     */
390     /*
391     Error Checking:
392     None
393     */
394 
395     printMessage("The suspended ready queue:\n");
396     showQueue(getSuspendedReady()->head, getSuspendedReady()->count);
397 }
```

5.27.1.12 suspendPCB()

```
void suspendPCB (
            char * processName )
```

Definition at line 138 of file R2commands.c.

```
139 { // COLTON WILL PROGRAM THIS FUNCTION
140     /*
141     Places a PCB in the suspended state and reinserts it into the appropriate queue
142     */
147 
148     PCB *PCBtoSuspend = findPCB(processName);
149 
150     if (PCBtoSuspend == NULL || strlen(processName) > 20)
151     {
152         printMessage("This is not a valid name.\n");
153     }
154     else if (PCBtoSuspend->processClass == 's')
155     {
156         printMessage("You do not have permission to suspend system processes!\n");
157     }
158     else
159     {
160         removePCB(PCBtoSuspend);
161         PCBtoSuspend->suspendedStatus = 0;
162         insertPCB(PCBtoSuspend);
163 
164         printMessage("The PCB was successfully suspended!\n");
165     }
166 }
```

5.27.1.13 unblockPCB()

```
void unblockPCB (
            char * processName )
```

Definition at line 114 of file R2commands.c.

```
115 { // ANASTASE WILL PROGRAM THIS FUNCTION
116 
117     /*
118     Places a PCB in the unblocked state and reinserts it into the appropriate queue.
119     */
120     /*
```

```
121     Error Checking:
122     Name must be valid.
123
124     */
125
126     PCB *pcb_to_unblock = findPCB(processName);
127     if (pcb_to_unblock != NULL)
128     {
129         pcb_to_unblock->runningStatus = 0; // ready
130         removePCB(pcb_to_unblock);         // is this the right place to put that function?
131         insertPCB(pcb_to_unblock);
132
133         printMessage("The PCB was successfully unblocked!\n");
134     }
135 }
```

5.28 modules/R3/procsr3.c File Reference

```
#include "../include/system.h"
#include "../include/core/serial.h"
#include "../modules/mpx_supt.h"
#include "procsr3.h"
```

Macros

- #define RC_1 1
- #define RC_2 2
- #define RC_3 3
- #define RC_4 4
- #define RC_5 5

Functions

- void proc1 ()
- void proc2 ()
- void proc3 ()
- void proc4 ()
- void proc5 ()

Variables

- char * msg1 = "proc1 dispatched\n"
- char * msg2 = "proc2 dispatched\n"
- char * msg3 = "proc3 dispatched\n"
- char * msg4 = "proc4 dispatched\n"
- char * msg5 = "proc5 dispatched\n"
- int msgSize = 17
- char * er1 = "proc1 ran after it was terminated\n"
- char * er2 = "proc2 ran after it was terminated\n"
- char * er3 = "proc3 ran after it was terminated\n"
- char * er4 = "proc4 ran after it was terminated\n"
- char * er5 = "proc5 ran after it was terminated\n"
- int erSize = 34

5.28.1 Macro Definition Documentation

5.28.1.1 RC_1

```
#define RC_1 1
```

Definition at line 7 of file procsr3.c.

5.28.1.2 RC_2

```
#define RC_2 2
```

Definition at line 8 of file procsr3.c.

5.28.1.3 RC_3

```
#define RC_3 3
```

Definition at line 9 of file procsr3.c.

5.28.1.4 RC_4

```
#define RC_4 4
```

Definition at line 10 of file procsr3.c.

5.28.1.5 RC_5

```
#define RC_5 5
```

Definition at line 11 of file procsr3.c.

5.28.2 Function Documentation

5.28.2.1 proc1()

```
void proc1 ( )
```

Definition at line 27 of file procsr3.c.

```
28 {
29   int i;
30
31   // repeat forever if termination fails
32   while (1)
33   {
34     for (i = 0; i < RC_1; i++)
35     {
36       sys_req(WRITE, DEFAULT_DEVICE, msg1, &msgSize);
37       sys_req(IDLE, DEFAULT_DEVICE, NULL, NULL);
38     }
39     sys_req(EXIT, DEFAULT_DEVICE, NULL, NULL);
40     sys_req(WRITE, DEFAULT_DEVICE, er1, &erSize);
41   }
42 }
```

5.28.2.2 proc2()

```
void proc2 ( )
```

Definition at line 44 of file procsr3.c.

```
45 {
46   int i;
47
48   // repeat forever if termination fails
49   while (1)
50   {
51     for (i = 0; i < RC_2; i++)
52     {
53       sys_req(WRITE, DEFAULT_DEVICE, msg2, &msgSize);
54       sys_req(IDLE, DEFAULT_DEVICE, NULL, NULL);
55     }
56     sys_req(EXIT, DEFAULT_DEVICE, NULL, NULL);
57     sys_req(WRITE, DEFAULT_DEVICE, er2, &erSize);
58   }
59 }
```

5.28.2.3 proc3()

```
void proc3 ( )
```

Definition at line 61 of file procsr3.c.

```
62 {
63   int i;
64
65   // repeat forever if termination fails
66   while (1)
67   {
68     for (i = 0; i < RC_3; i++)
69     {
70       sys_req(WRITE, DEFAULT_DEVICE, msg3, &msgSize);
71       sys_req(IDLE, DEFAULT_DEVICE, NULL, NULL);
72     }
73     sys_req(EXIT, DEFAULT_DEVICE, NULL, NULL);
74     sys_req(WRITE, DEFAULT_DEVICE, er3, &erSize);
75   }
76 }
```

5.28.2.4 proc4()

```
void proc4 ( )
```

Definition at line 78 of file procsr3.c.

```
79 {
80   int i;
81
82   // repeat forever if termination fails
83   while (1)
84   {
85     for (i = 0; i < RC_4; i++)
86     {
87       sys_req(WRITE, DEFAULT_DEVICE, msg4, &msgSize);
88       sys_req(IDLE, DEFAULT_DEVICE, NULL, NULL);
89     }
90     sys_req(EXIT, DEFAULT_DEVICE, NULL, NULL);
91     sys_req(WRITE, DEFAULT_DEVICE, er4, &erSize);
92   }
93 }
```

5.28.2.5 proc5()

```
void proc5 ( )
```

Definition at line 95 of file procsr3.c.

```
96 {
97   int i;
98
99   // repeat forever if termination fails
100   while (1)
101   {
102     for (i = 0; i < RC_5; i++)
103     {
104       sys_req(WRITE, DEFAULT_DEVICE, msg5, &msgSize);
105       sys_req(IDLE, DEFAULT_DEVICE, NULL, NULL);
106     }
107     sys_req(EXIT, DEFAULT_DEVICE, NULL, NULL);
108     sys_req(WRITE, DEFAULT_DEVICE, er5, &erSize);
109   }
110 }
```

5.28.3 Variable Documentation

5.28.3.1 er1

```
char* er1 = "proc1 ran after it was terminated\n"
```

Definition at line 20 of file procsr3.c.

5.28.3.2 er2

```
char* er2 = "proc2 ran after it was terminated\n"
```

Definition at line 21 of file procsr3.c.

5.28.3.3 er3

```
char* er3 = "proc3 ran after it was terminated\n"
```

Definition at line 22 of file procsr3.c.

5.28.3.4 er4

```
char* er4 = "proc4 ran after it was terminated\n"
```

Definition at line 23 of file procsr3.c.

5.28.3.5 er5

```
char* er5 = "proc5 ran after it was terminated\n"
```

Definition at line 24 of file procsr3.c.

5.28.3.6 erSize

```
int erSize = 34
```

Definition at line 25 of file procsr3.c.

5.28.3.7 msg1

```
char* msg1 = "proc1 dispatched\n"
```

Definition at line 13 of file procsr3.c.

5.28.3.8 msg2

```
char* msg2 = "proc2 dispatched\n"
```

Definition at line 14 of file procsr3.c.

5.28.3.9 msg3

```
char* msg3 = "proc3 dispatched\n"
```

Definition at line 15 of file procsr3.c.

5.28.3.10 msg4

```
char* msg4 = "proc4 dispatched\n"
```

Definition at line 16 of file procsr3.c.

5.28.3.11 msg5

```
char* msg5 = "proc5 dispatched\n"
```

Definition at line 17 of file procsr3.c.

5.28.3.12 msgSize

```
int msgSize = 17
```

Definition at line 18 of file procsr3.c.

5.29 modules/R3/procsr3.h File Reference

Functions

- void proc1 ()
- void proc2 ()
- void proc3 ()
- void proc4 ()
- void proc5 ()

5.29.1 Function Documentation

5.29.1.1 proc1()

```
void proc1 ( )
```

Definition at line 27 of file procsr3.c.

```
28 {
29   int i;
30
31   // repeat forever if termination fails
32   while (1)
33   {
34     for (i = 0; i < RC_1; i++)
35     {
36       sys_req(WRITE, DEFAULT_DEVICE, msg1, &msgSize);
37       sys_req(IDLE, DEFAULT_DEVICE, NULL, NULL);
38     }
39     sys_req(EXIT, DEFAULT_DEVICE, NULL, NULL);
40     sys_req(WRITE, DEFAULT_DEVICE, er1, &erSize);
41   }
42 }
```

5.29.1.2 proc2()

```
void proc2 ( )
```

Definition at line 44 of file procsr3.c.

```
45 {
46   int i;
47
48   // repeat forever if termination fails
49   while (1)
50   {
51     for (i = 0; i < RC_2; i++)
52     {
53       sys_req(WRITE, DEFAULT_DEVICE, msg2, &msgSize);
54       sys_req(IDLE, DEFAULT_DEVICE, NULL, NULL);
55     }
56     sys_req(EXIT, DEFAULT_DEVICE, NULL, NULL);
57     sys_req(WRITE, DEFAULT_DEVICE, er2, &erSize);
58   }
59 }
```

5.29.1.3 proc3()

```
void proc3 ( )
```

Definition at line 61 of file procsr3.c.

```
62 {
63   int i;
64
65   // repeat forever if termination fails
66   while (1)
67   {
68     for (i = 0; i < RC_3; i++)
69     {
70       sys_req(WRITE, DEFAULT_DEVICE, msg3, &msgSize);
71       sys_req(IDLE, DEFAULT_DEVICE, NULL, NULL);
72     }
73     sys_req(EXIT, DEFAULT_DEVICE, NULL, NULL);
74     sys_req(WRITE, DEFAULT_DEVICE, er3, &erSize);
75   }
76 }
```

5.29.1.4 proc4()

```
void proc4 ( )
```

Definition at line 78 of file procsr3.c.

```
79 {
80   int i;
81
82   // repeat forever if termination fails
83   while (1)
84   {
85     for (i = 0; i < RC_4; i++)
86     {
87       sys_req(WRITE, DEFAULT_DEVICE, msg4, &msgSize);
88       sys_req(IDLE, DEFAULT_DEVICE, NULL, NULL);
89     }
90     sys_req(EXIT, DEFAULT_DEVICE, NULL, NULL);
91     sys_req(WRITE, DEFAULT_DEVICE, er4, &erSize);
92   }
93 }
```

5.29.1.5 proc5()

```
void proc5 ( )
```

Definition at line 95 of file procsr3.c.

```
96 {
97   int i;
98
99   // repeat forever if termination fails
100   while (1)
101   {
102     for (i = 0; i < RC_5; i++)
103     {
104       sys_req(WRITE, DEFAULT_DEVICE, msg5, &msgSize);
105       sys_req(IDLE, DEFAULT_DEVICE, NULL, NULL);
106     }
107     sys_req(EXIT, DEFAULT_DEVICE, NULL, NULL);
108     sys_req(WRITE, DEFAULT_DEVICE, er5, &erSize);
109   }
110 }
```

5.30 modules/R3/R3commands.c File Reference

```
#include <string.h>
#include "../mpx_supt.h"
#include <core/serial.h>
#include "../utilities.h"
#include "../R2/R2_Internal_Functions_And_Structures.h"
#include "../R2/R2commands.h"
#include "R3commands.h"
#include "procsr3.h"
```

Functions

- void yield ()
- void loadr3 ()

5.30.1 Function Documentation

5.30.1.1 loadr3()

```
void loadr3 ( )
```

Definition at line 18 of file R3commands.c.

```
19 {
20     //loadr3 will load all r3 "processes" (proc3.c file eCampus) into memory in a suspended ready state
       at any priority of your choosing.
21     // We may want to change these to use setupPCB instead of createPCB and suspendPCB
22     printMessage("Loading R3 Processes.\n\n");
23
24     createPCB("Process1", 'a', 1);
25     suspendPCB("Process1");
26     PCB *new_pcb1 = findPCB("Process1");
27     context *cp1 = (context *)(new_pcb1->stackTop);
28     memset(cp1, 0, sizeof(context));
29     cp1->fs = 0x10;
30     cp1->gs = 0x10;
31     cp1->ds = 0x10;
32     cp1->es = 0x10;
33     cp1->cs = 0x8;
34     cp1->ebp = (u32int)(new_pcb1->stack);
35     cp1->esp = (u32int)(new_pcb1->stackTop);
36     cp1->eip = (u32int)proc1; // The function correlating to the process, ie. Proc1
37     cp1->eflags = 0x202;
38
39     createPCB("Process2", 'a', 1);
40     suspendPCB("Process2");
41     PCB *new_pcb2 = findPCB("Process2");
42     context *cp2 = (context *)(new_pcb2->stackTop);
43     memset(cp2, 0, sizeof(context));
44     cp2->fs = 0x10;
45     cp2->gs = 0x10;
46     cp2->ds = 0x10;
47     cp2->es = 0x10;
48     cp2->cs = 0x8;
49     cp2->ebp = (u32int)(new_pcb2->stack);
50     cp2->esp = (u32int)(new_pcb2->stackTop);
51     cp2->eip = (u32int)proc2; // The function correlating to the process, ie. Proc1
52     cp2->eflags = 0x202;
53
54     createPCB("Process3", 'a', 1);
55     suspendPCB("Process3");
56     PCB *new_pcb3 = findPCB("Process3");
57     context *cp3 = (context *)(new_pcb3->stackTop);
58     memset(cp3, 0, sizeof(context));
59     cp3->fs = 0x10;
60     cp3->gs = 0x10;
61     cp3->ds = 0x10;
62     cp3->es = 0x10;
63     cp3->cs = 0x8;
64     cp3->ebp = (u32int)(new_pcb3->stack);
65     cp3->esp = (u32int)(new_pcb3->stackTop);
66     cp3->eip = (u32int)proc3; // The function correlating to the process, ie. Proc1
67     cp3->eflags = 0x202;
68
69     createPCB("Process4", 'a', 1);
70     suspendPCB("Process4");
71     PCB *new_pcb4 = findPCB("Process4");
72     context *cp4 = (context *)(new_pcb4->stackTop);
73     memset(cp4, 0, sizeof(context));
74     cp4->fs = 0x10;
75     cp4->gs = 0x10;
76     cp4->ds = 0x10;
77     cp4->es = 0x10;
78     cp4->cs = 0x8;
79     cp4->ebp = (u32int)(new_pcb4->stack);
80     cp4->esp = (u32int)(new_pcb4->stackTop);
81     cp4->eip = (u32int)proc4; // The function correlating to the process, ie. Proc1
82     cp4->eflags = 0x202;
83
84     createPCB("Process5", 'a', 1);
85     suspendPCB("Process5");
86     PCB *new_pcb5 = findPCB("Process5");
87     context *cp5 = (context *)(new_pcb5->stackTop);
88     memset(cp5, 0, sizeof(context));
```

```
89      cp5->fs = 0x10;
90      cp5->gs = 0x10;
91      cp5->ds = 0x10;
92      cp5->es = 0x10;
93      cp5->cs = 0x8;
94      cp5->ebp = (u32int)(new_pcb5->stack);
95      cp5->esp = (u32int)(new_pcb5->stackTop);
96      cp5->eip = (u32int)proc5; // The function correlating to the process, ie. Proc1
97      cp5->eflags = 0x202;
98  }
```

5.30.1.2 yield()

```
void yield ( )
```

Definition at line 13 of file R3commands.c.

```
14 { // temporary command - only in R3
15      asm volatile("int $60");
16 }
```

5.31 modules/R3/R3commands.h File Reference

Classes

- struct context

Typedefs

- typedef struct context context

Functions

- void yield ()
- void loadr3 ()

5.31.1 Typedef Documentation

5.31.1.1 context

```
typedef struct context context
```

5.31.2 Function Documentation

5.31.2.1 loadr3()

```
void loadr3 ( )
```

Definition at line 18 of file R3commands.c.

```
19 {
20     //loadr3 will load all r3 "processes" (proc3.c file eCampus) into memory in a suspended ready state
       at any priority of your choosing.
21     // We may want to change these to use setupPCB instead of createPCB and suspendPCB
22     printMessage("Loading R3 Processes.\n\n");
23
24     createPCB("Process1", 'a', 1);
25     suspendPCB("Process1");
26     PCB *new_pcb1 = findPCB("Process1");
27     context *cp1 = (context *)(new_pcb1->stackTop);
28     memset(cp1, 0, sizeof(context));
29     cp1->fs = 0x10;
30     cp1->gs = 0x10;
31     cp1->ds = 0x10;
32     cp1->es = 0x10;
33     cp1->cs = 0x8;
34     cp1->ebp = (u32int)(new_pcb1->stack);
35     cp1->esp = (u32int)(new_pcb1->stackTop);
36     cp1->eip = (u32int)proc1; // The function correlating to the process, ie. Proc1
37     cp1->eflags = 0x202;
38
39     createPCB("Process2", 'a', 1);
40     suspendPCB("Process2");
41     PCB *new_pcb2 = findPCB("Process2");
42     context *cp2 = (context *)(new_pcb2->stackTop);
43     memset(cp2, 0, sizeof(context));
44     cp2->fs = 0x10;
45     cp2->gs = 0x10;
46     cp2->ds = 0x10;
47     cp2->es = 0x10;
48     cp2->cs = 0x8;
49     cp2->ebp = (u32int)(new_pcb2->stack);
50     cp2->esp = (u32int)(new_pcb2->stackTop);
51     cp2->eip = (u32int)proc2; // The function correlating to the process, ie. Proc1
52     cp2->eflags = 0x202;
53
54     createPCB("Process3", 'a', 1);
55     suspendPCB("Process3");
56     PCB *new_pcb3 = findPCB("Process3");
57     context *cp3 = (context *)(new_pcb3->stackTop);
58     memset(cp3, 0, sizeof(context));
59     cp3->fs = 0x10;
60     cp3->gs = 0x10;
61     cp3->ds = 0x10;
62     cp3->es = 0x10;
63     cp3->cs = 0x8;
64     cp3->ebp = (u32int)(new_pcb3->stack);
65     cp3->esp = (u32int)(new_pcb3->stackTop);
66     cp3->eip = (u32int)proc3; // The function correlating to the process, ie. Proc1
67     cp3->eflags = 0x202;
68
69     createPCB("Process4", 'a', 1);
70     suspendPCB("Process4");
71     PCB *new_pcb4 = findPCB("Process4");
72     context *cp4 = (context *)(new_pcb4->stackTop);
73     memset(cp4, 0, sizeof(context));
74     cp4->fs = 0x10;
75     cp4->gs = 0x10;
76     cp4->ds = 0x10;
77     cp4->es = 0x10;
78     cp4->cs = 0x8;
79     cp4->ebp = (u32int)(new_pcb4->stack);
80     cp4->esp = (u32int)(new_pcb4->stackTop);
81     cp4->eip = (u32int)proc4; // The function correlating to the process, ie. Proc1
82     cp4->eflags = 0x202;
83
84     createPCB("Process5", 'a', 1);
85     suspendPCB("Process5");
86     PCB *new_pcb5 = findPCB("Process5");
87     context *cp5 = (context *)(new_pcb5->stackTop);
88     memset(cp5, 0, sizeof(context));
89     cp5->fs = 0x10;
90     cp5->gs = 0x10;
91     cp5->ds = 0x10;
92     cp5->es = 0x10;
93     cp5->cs = 0x8;
94     cp5->ebp = (u32int)(new_pcb5->stack);
95     cp5->esp = (u32int)(new_pcb5->stackTop);
96     cp5->eip = (u32int)proc5; // The function correlating to the process, ie. Proc1
97     cp5->eflags = 0x202;
```

```
98 }
```

5.31.2.2 yield()

```
void yield ( )
```

Definition at line 13 of file R3commands.c.

```
14 { // temporary command - only in R3
15     asm volatile("int $60");
16 }
```

5.32 modules/R4/R4commands.c File Reference

```
#include <string.h>
#include "../mpx_supt.h"
#include <core/serial.h>
#include <core/io.h>
#include "../R2/R2_Internal_Functions_And_Structures.h"
#include "../R2/R2commands.h"
#include "../R3/R3commands.h"
#include "R4commands.h"
#include "../utilities.h"
#include "../R1/R1commands.h"
```

Functions

- void alarmPCB ()
- void infinitePCB ()
- void infiniteFunc ()
- void allocateAlarmQueue ()
- alarm * allocateAlarms ()
- alarmList * getAlarms ()
- void addAlarm ()
- int convertTime (char *hours, char *minutes, char *seconds)
- void iterateAlarms ()

Variables

- alarmList * alarms

5.32.1 Function Documentation

5.32.1.1 addAlarm()

```
void addAlarm ( )
```

Definition at line 86 of file R4commands.c.

```
87 {
88
89     unblockPCB("Alarm");
90
91     printMessage("Please enter a name for the alarm you want to create.\n\n");
92
93     alarm *Alarm_to_insert = allocateAlarms();
94
95     int nameLength = strlen(Alarm_to_insert->alarmName);
96     sys_req(READ, DEFAULT_DEVICE, Alarm_to_insert->alarmName, &nameLength);
97
98     printMessage("Please type the desired hours. I.E.: hh.\n");
99
100     char hour[4] = "\0\0\n\0";
101
102     int flag = 0;
103
104     do
105     {
106         int hourLength = strlen(hour);
107         sys_req(READ, DEFAULT_DEVICE, hour, &hourLength);
108         if (atoi(hour) < 24 && atoi(hour) >= 0)
109         {
110
111             printMessage("\n");
112             flag = 0;
113         }
114         else
115         {
116             printMessage("\nInvalid hours.\n");
117
118             flag = 1;
119         }
120     } while (flag == 1);
121
123     printMessage("Please type the desired minutes. I.E.: mm.\n");
124
125     char minute[4] = "\0\0\n\0";
126
127     do
128     {
129         int minuteLength = strlen(minute);
130         sys_req(READ, DEFAULT_DEVICE, minute, &minuteLength);
131         if (atoi(minute) < 60 && atoi(minute) >= 0)
132         {
133             printMessage("\n");
134             flag = 0;
135         }
136         else
137         {
138             printMessage("\nInvalid minutes.\n");
139             flag = 1;
140         }
141     } while (flag == 1);
142
144     printMessage("Please type the desired seconds. I.E.: ss.\n");
145
146     char second[4] = "\0\0\n\0";
147
148     do
149     {
150         int secondLength = strlen(second);
151         sys_req(READ, DEFAULT_DEVICE, second, &secondLength);
152         if (atoi(second) < 60 && atoi(second) >= 0)
153         {
154
155             printMessage("\n");
156             flag = 0;
157         }
158         else
159         {
160             printMessage("\nInvalid seconds.\n");
161             flag = 1;
162         }
163     } while (flag == 1);
164
165     // Storing time in the alarm to insert
166     Alarm_to_insert->alarmTime = convertTime(hour, minute, second);
167
168     // Inserting the alarm
```

```
169     if (getAlarms()->head != NULL)
170     {
171         getAlarms()->tail->nextAlarm = Alarm_to_insert;
172         Alarm_to_insert->prevAlarm = getAlarms()->tail;
173         getAlarms()->tail = Alarm_to_insert;
174         getAlarms()->count++;
175     }
176     else
177     {
178         getAlarms()->head = Alarm_to_insert;
179         getAlarms()->tail = Alarm_to_insert;
180         getAlarms()->count++;
181     }
182 }
```

5.32.1.2 alarmPCB()

```
void alarmPCB ( )
```

Definition at line 17 of file R4commands.c.

```
18 {
19     if (alarms->head == NULL && findPCB("Alarm")->runningStatus != -1)
20     {
21         blockPCB("Alarm");
22     }
23     else
24     {
25         iterateAlarms();
26     }
27 }
```

5.32.1.3 allocateAlarmQueue()

```
void allocateAlarmQueue ( )
```

Definition at line 57 of file R4commands.c.

```
58 {
59     alarms = sys_alloc_mem(sizeof(alarmList));
60     alarms->count = NULL;
61     alarms->head = NULL;
62     alarms->tail = NULL;
63 }
```

5.32.1.4 allocateAlarms()

```
alarm* allocateAlarms ( )
```

Definition at line 65 of file R4commands.c.

```
66 {
67     alarm *newAlarm = (alarm *)sys_alloc_mem(sizeof(alarm));
68 
69     char name[20] = "newAlarm";
70     strcpy(newAlarm->alarmName, name);
71 
72     newAlarm->alarmTime = 0;
73 
74     // Setting the alarms prev and next PCB
75     newAlarm->nextAlarm = NULL;
76     newAlarm->prevAlarm = NULL;
77 
78     return newAlarm;
79 }
```

5.32.1.5 convertTime()

```
int convertTime (
            char * hours,
            char * minutes,
            char * seconds )
```

Definition at line 184 of file R4commands.c.

```
185 {
186     int result = (atoi(hours) * 3600);
187     result += (atoi(minutes) * 60);
188     result += (atoi(seconds));
189 
190     return result;
191 }
```

5.32.1.6 getAlarms()

```
alarmList* getAlarms ( )
```

Definition at line 81 of file R4commands.c.

```
82 {
83     return alarms;
84 }
```

5.32.1.7 infiniteFunc()

```
void infiniteFunc ( )
```

Definition at line 46 of file R4commands.c.

```
47 {
48     while (1)
49     {
50 
51         printMessage("Infinite Process Executing.\n");
52 
53         sys_req(IDLE, DEFAULT_DEVICE, NULL, NULL);
54     }
55 }
```

5.32.1.8 infinitePCB()

```
void infinitePCB ( )
```

Definition at line 29 of file R4commands.c.

```
30 {
31     createPCB("infinite", 'a', 1);
32     PCB *new_pcb = findPCB("infinite");
33     context *cp = (context *)(new_pcb->stackTop);
34     memset(cp, 0, sizeof(context));
35     cp->fs = 0x10;
36     cp->gs = 0x10;
37     cp->ds = 0x10;
38     cp->es = 0x10;
39     cp->cs = 0x8;
40     cp->ebp = (u32int)(new_pcb->stack);
41     cp->esp = (u32int)(new_pcb->stackTop);
42     cp->eip = (u32int)infiniteFunc; // The function correlating to the process, ie. Proc1
43     cp->eflags = 0x202;
44 }
```

5.32.1.9 iterateAlarms()

```
void iterateAlarms ( )
```

Definition at line 193 of file R4commands.c.

```
194 {
195     char hours[4] = "\0\0\0\0";
196     outb(0x70, 0x04); // getting current Hour value
197     BCDtoChar(inb(0x71), hours);
198 
199     char minutes[4] = "\0\0\0\0";
200     outb(0x70, 0x02); // getting current Minute value
201     BCDtoChar(inb(0x71), minutes);
202 
203     char seconds[4] = "\0\0\0\0";
204     outb(0x70, 0x00); // getting current Minute value
205     BCDtoChar(inb(0x71), seconds);
206 
207     int currentTime = convertTime(hours, minutes, seconds);
208 
209     alarm *tempAlarm = getAlarms()->head;
210 
211     while (tempAlarm != NULL)
212     {
213         if (currentTime >= getAlarms()->head->alarmTime)
214         {
215             // do something for alarm.
216             printMessage(getAlarms()->head->alarmName);
217             getAlarms()->head = getAlarms()->head->nextAlarm;
218         }
219         else if (currentTime >= getAlarms()->tail->alarmTime)
220         {
221             printMessage(getAlarms()->tail->alarmName);
222             getAlarms()->tail = getAlarms()->tail->prevAlarm;
223         }
224         else if (currentTime >= tempAlarm->alarmTime)
225         {
226             printMessage(tempAlarm->alarmName);
227             tempAlarm->prevAlarm->nextAlarm = tempAlarm->nextAlarm;
228             tempAlarm->nextAlarm->prevAlarm = tempAlarm->prevAlarm;
229             tempAlarm->nextAlarm = NULL;
230             tempAlarm->prevAlarm = NULL;
231         }
232         else
233         {
234             // iterates if not time
235             tempAlarm = tempAlarm->nextAlarm;
236         }
237     }
238 }
```

5.32.2 Variable Documentation

5.32.2.1 alarms

```
alarmList* alarms
```

Definition at line 15 of file R4commands.c.

5.33 modules/R4/R4commands.h File Reference

Classes

- struct alarm
- struct alarmList

Typedefs

- typedef struct alarm alarm
- typedef struct alarmList alarmList

Functions

- void alarmPCB ()
- void infinitePCB ()
- void infiniteFunc ()
- void allocateAlarmQueue ()
- alarm ∗ allocateAlarms ()
- alarmList ∗ getAlarms ()
- void addAlarm ()
- int convertTime (char ∗hours, char ∗minutes, char ∗seconds)
- void iterateAlarms ()

5.33.1 Typedef Documentation

5.33.1.1 alarm

```
typedef struct alarm alarm
```

5.33.1.2 alarmList

```
typedef struct alarmList alarmList
```

5.33.2 Function Documentation

5.33.2.1 addAlarm()

```
void addAlarm ( )
```

Definition at line 86 of file R4commands.c.

```
87 {
88
89     unblockPCB("Alarm");
90
91     printMessage("Please enter a name for the alarm you want to create.\n\n");
92
93     alarm *Alarm_to_insert = allocateAlarms();
94
95     int nameLength = strlen(Alarm_to_insert->alarmName);
96     sys_req(READ, DEFAULT_DEVICE, Alarm_to_insert->alarmName, &nameLength);
97
98     printMessage("Please type the desired hours. I.E.: hh.\n");
99
100     char hour[4] = "\0\0\n\0";
101
102     int flag = 0;
103
104     do
105     {
106         int hourLength = strlen(hour);
107         sys_req(READ, DEFAULT_DEVICE, hour, &hourLength);
108         if (atoi(hour) < 24 && atoi(hour) >= 0)
109         {
110
111             printMessage("\n");
112             flag = 0;
113         }
114         else
115         {
116             printMessage("\nInvalid hours.\n");
117
118             flag = 1;
119         }
120     } while (flag == 1);
121
123     printMessage("Please type the desired minutes. I.E.: mm.\n");
124
125     char minute[4] = "\0\0\n\0";
126
127     do
128     {
129         int minuteLength = strlen(minute);
130         sys_req(READ, DEFAULT_DEVICE, minute, &minuteLength);
131         if (atoi(minute) < 60 && atoi(minute) >= 0)
132         {
133             printMessage("\n");
134             flag = 0;
135         }
136         else
137         {
138             printMessage("\nInvalid minutes.\n");
139             flag = 1;
140         }
141     } while (flag == 1);
142
144     printMessage("Please type the desired seconds. I.E.: ss.\n");
145
146     char second[4] = "\0\0\n\0";
147
148     do
149     {
150         int secondLength = strlen(second);
151         sys_req(READ, DEFAULT_DEVICE, second, &secondLength);
152         if (atoi(second) < 60 && atoi(second) >= 0)
153         {
154
155             printMessage("\n");
156             flag = 0;
157         }
158         else
159         {
160             printMessage("\nInvalid seconds.\n");
161             flag = 1;
162         }
163     } while (flag == 1);
164
165     // Storing time in the alarm to insert
166     Alarm_to_insert->alarmTime = convertTime(hour, minute, second);
167
168     // Inserting the alarm
```

```
169     if (getAlarms()->head != NULL)
170     {
171         getAlarms()->tail->nextAlarm = Alarm_to_insert;
172         Alarm_to_insert->prevAlarm = getAlarms()->tail;
173         getAlarms()->tail = Alarm_to_insert;
174         getAlarms()->count++;
175     }
176     else
177     {
178         getAlarms()->head = Alarm_to_insert;
179         getAlarms()->tail = Alarm_to_insert;
180         getAlarms()->count++;
181     }
182 }
```

5.33.2.2 alarmPCB()

```
void alarmPCB ( )
```

Definition at line 17 of file R4commands.c.

```
18 {
19     if (alarms->head == NULL && findPCB("Alarm")->runningStatus != -1)
20     {
21         blockPCB("Alarm");
22     }
23     else
24     {
25         iterateAlarms();
26     }
27 }
```

5.33.2.3 allocateAlarmQueue()

```
void allocateAlarmQueue ( )
```

Definition at line 57 of file R4commands.c.

```
58 {
59     alarms = sys_alloc_mem(sizeof(alarmList));
60     alarms->count = NULL;
61     alarms->head = NULL;
62     alarms->tail = NULL;
63 }
```

5.33.2.4 allocateAlarms()

```
alarm* allocateAlarms ( )
```

Definition at line 65 of file R4commands.c.

```
66 {
67     alarm *newAlarm = (alarm *)sys_alloc_mem(sizeof(alarm));
68 
69     char name[20] = "newAlarm";
70     strcpy(newAlarm->alarmName, name);
71 
72     newAlarm->alarmTime = 0;
73 
74     // Setting the alarms prev and next PCB
75     newAlarm->nextAlarm = NULL;
76     newAlarm->prevAlarm = NULL;
77 
78     return newAlarm;
79 }
```

5.33.2.5 convertTime()

```
int convertTime (
            char * hours,
            char * minutes,
            char * seconds )
```

Definition at line 184 of file R4commands.c.

```
185 {
186     int result = (atoi(hours) * 3600);
187     result += (atoi(minutes) * 60);
188     result += (atoi(seconds));
189 
190     return result;
191 }
```

5.33.2.6 getAlarms()

```
alarmList* getAlarms ( )
```

Definition at line 81 of file R4commands.c.

```
82 {
83     return alarms;
84 }
```

5.33.2.7 infiniteFunc()

```
void infiniteFunc ( )
```

Definition at line 46 of file R4commands.c.

```
47 {
48     while (1)
49     {
50 
51         printMessage("Infinite Process Executing.\n");
52 
53         sys_req(IDLE, DEFAULT_DEVICE, NULL, NULL);
54     }
55 }
```

5.33.2.8 infinitePCB()

```
void infinitePCB ( )
```

Definition at line 29 of file R4commands.c.

```
30 {
31     createPCB("infinite", 'a', 1);
32     PCB *new_pcb = findPCB("infinite");
33     context *cp = (context *)(new_pcb->stackTop);
34     memset(cp, 0, sizeof(context));
35     cp->fs = 0x10;
36     cp->gs = 0x10;
37     cp->ds = 0x10;
38     cp->es = 0x10;
39     cp->cs = 0x8;
40     cp->ebp = (u32int)(new_pcb->stack);
41     cp->esp = (u32int)(new_pcb->stackTop);
42     cp->eip = (u32int)infiniteFunc; // The function correlating to the process, ie. Proc1
43     cp->eflags = 0x202;
44 }
```

5.33.2.9 iterateAlarms()

```
void iterateAlarms ( )
```

Definition at line 193 of file R4commands.c.

```
194 {
195     char hours[4] = "\0\0\0\0";
196     outb(0x70, 0x04); // getting current Hour value
197     BCDtoChar(inb(0x71), hours);
198 
199     char minutes[4] = "\0\0\0\0";
200     outb(0x70, 0x02); // getting current Minute value
201     BCDtoChar(inb(0x71), minutes);
202 
203     char seconds[4] = "\0\0\0\0";
204     outb(0x70, 0x00); // getting current Minute value
205     BCDtoChar(inb(0x71), seconds);
206 
207     int currentTime = convertTime(hours, minutes, seconds);
208 
209     alarm *tempAlarm = getAlarms()->head;
210 
211     while (tempAlarm != NULL)
212     {
213         if (currentTime >= getAlarms()->head->alarmTime)
214         {
215             // do something for alarm.
216             printMessage(getAlarms()->head->alarmName);
217             getAlarms()->head = getAlarms()->head->nextAlarm;
218         }
219         else if (currentTime >= getAlarms()->tail->alarmTime)
220         {
221             printMessage(getAlarms()->tail->alarmName);
222             getAlarms()->tail = getAlarms()->tail->prevAlarm;
223         }
224         else if (currentTime >= tempAlarm->alarmTime)
225         {
226             printMessage(tempAlarm->alarmName);
227             tempAlarm->prevAlarm->nextAlarm = tempAlarm->nextAlarm;
228             tempAlarm->nextAlarm->prevAlarm = tempAlarm->prevAlarm;
229             tempAlarm->nextAlarm = NULL;
230             tempAlarm->prevAlarm = NULL;
231         }
232         else
233         {
234             // iterates if not time
235             tempAlarm = tempAlarm->nextAlarm;
236         }
237     }
238 }
```

5.34 modules/R5/R5commands.c File Reference

```
#include <core/serial.h>
#include <string.h>
#include "../mpx_supt.h"
#include <core/io.h>
#include <mem/heap.h>
#include "../utilities.h"
#include "../R2/R2commands.h"
#include "../R2/R2_Internal_Functions_And_Structures.h"
#include "R5commands.h"
#include "../R1/R1commands.h"
```

Functions

- void showMCB (CMCB *mem)
- u32int initializeHeap (u32int heapSize)

- void insertToList (CMCB *current, memList *list)
- u32int allocateMemory (u32int size)
- void removeFromAlloc (CMCB *temp)
- int freeMemory (void *memToFree)
- int isEmpty ()
- void showFreeMemory ()
- void showAllocatedMemory ()

Variables

- memList freeList
- memList allocatedList

5.34.1 Function Documentation

5.34.1.1 allocateMemory()

```
u32int allocateMemory (
            u32int size )
```

Definition at line 90 of file R5commands.c.

```
91 {
92     if (freeList.head != NULL)
93     {
94         CMCB *current = freeList.head;
95 
96         // get to block of appropriate size
97         while (current != NULL)
98         {
99             if (current->size == size + sizeof(CMCB) && freeList.count == 1)
100             {
101                 // remove from free list.
102                 current->nextCMCB->prevCMCB = current->prevCMCB;
103                 current->prevCMCB->nextCMCB = current->nextCMCB;
104                 current->nextCMCB = NULL;
105                 current->prevCMCB = NULL;
106                 // place current in alloc list.
107                 insertToList(current, &allocatedList);
108 
109                 // change current marker to 'a'.
110                 current->type = 'a';
111 
112                 // remove all freeList pointers to current.
113                 freeList.head = NULL;
114                 freeList.tail = NULL;
115 
116                 // return allocated block.
117                 return current->beginningAddr;
118             }
119             else if (current->size == size + sizeof(CMCB)) // current is excetly the size requested.
120             {
121                 // remove from free list.
122                 current->nextCMCB->prevCMCB = current->prevCMCB;
123                 current->prevCMCB->nextCMCB = current->nextCMCB;
124                 current->nextCMCB = NULL;
125                 current->prevCMCB = NULL;
126                 // place current in alloc list.
127                 insertToList(current, &allocatedList);
128 
129                 // change current marker to 'a'.
130                 current->type = 'a';
131 
132                 // return allocated block.
133                 return current->beginningAddr;
134             }
135             else if (current->size > size + sizeof(CMCB)) // current is greater than the size requested
```

```
136             {
137                 // remove from free list.
138                 CMCB *new = (CMCB *)(current->beginningAddr + size);                 // This CMCB
    pertains to the head of the free list at the new memory address
139                 new->beginningAddr = (current->beginningAddr + size + sizeof(CMCB)); // Could be
    tmp->beginningAddr + size + sizeof(CMCB)
140                 new->size = current->size - size - sizeof(CMCB);
141                 new->type = 'f';
142                 new->nextCMCB = current->nextCMCB;
143                 new->prevCMCB = current->prevCMCB;
144 
145                 if (current->prevCMCB != NULL)
146                 {
147                     new->prevCMCB->nextCMCB = new;
148                 }
149 
150                 if (current->nextCMCB != NULL)
151                 {
152                     new->nextCMCB->prevCMCB = new;
153                 }
154 
155                 if (freeList.head == current && freeList.tail == current)
156                 {
157                     freeList.head = new;
158                     freeList.tail = new;
159                 }
160                 else if (freeList.head == current)
161                 {
162                     freeList.head = new;
163                 }
164                 else if (freeList.tail == current)
165                 {
166                     freeList.tail = new;
167                 }
168 
169                 current->size = size;
170                 current->nextCMCB = NULL;
171                 current->prevCMCB = NULL;
172 
173                 // place current in alloc list.
174                 insertToList(current, &allocatedList);
175 
176                 // change current marker to 'a'.
177                 current->type = 'a';
178 
179                 // return allocated block.
180                 return current->beginningAddr;
181             }
182             current = current->nextCMCB;
183         }
184     }
185 
186     return NULL;
187 }
```

5.34.1.2 freeMemory()

```
int freeMemory (
            void * memToFree )
```

Definition at line 215 of file R5commands.c.

```
216 {
217     if (isEmpty())
218     {
219         printMessage("There is no memory to free!\n");
220         return 1;
221     }
222 
223     CMCB *temp = allocatedList.head;
224 
225     while (temp->beginningAddr != (u32int)memToFree)
226     {
227         temp = temp->nextCMCB;
228     }
229 
230     if (temp == NULL)
231     {
232         printMessage("There is no allocated memory at that address!\n");
```

```
233         return 1;
234     }
235     else
236     {
237 
238         // Remove memToFree from the allocatedList.
239         removeFromAlloc(temp);
240 
241         // Insert memToFree into the freeList in increasing order.
242         insertToList(temp, &freeList);
243         temp->type = 'f';
244 
245         // Merge memToFree to other free CMCBs if possible.
246         if (freeList.count >= 1)
247         {
248             CMCB *temp = freeList.head;
249             while (temp != NULL)
250             {
251                 if ((temp->beginningAddr + temp->size) == (temp->nextCMCB->beginningAddr -
    sizeof(CMCB))) // merge down
252                 {
253                     printMessage("Memory merge down\n");
254                     if (temp->nextCMCB->nextCMCB != NULL)
255                     {
256                         CMCB *next = temp->nextCMCB;
257                         temp->size += (next->size + sizeof(CMCB));
258                         temp->nextCMCB = next->nextCMCB;
259                         next->nextCMCB->prevCMCB = temp;
260                         next->prevCMCB = NULL;
261                         next->nextCMCB = NULL;
262                         freeList.count--;
263                     }
264                     else
265                     {
266                         printMessage("Merge down part 2\n");
267                         CMCB *next = temp->nextCMCB;
268                         temp->size += (next->size + sizeof(CMCB));
269                         next->prevCMCB = NULL;
270                         next->nextCMCB = NULL;
271                         temp->nextCMCB = NULL;
272                         freeList.count--;
273                     }
274                 }
275 
276                 if ((temp->prevCMCB->beginningAddr + temp->prevCMCB->size) == (temp->beginningAddr -
    sizeof(CMCB))) //merge up
277                 {
278                     printMessage("Memory merge up\n");
279                     CMCB *prev = temp->prevCMCB;
280                     prev->size += (temp->size + sizeof(CMCB));
281                     prev->nextCMCB = temp->nextCMCB;
282                     temp->nextCMCB = prev;
283                     temp->nextCMCB = NULL;
284                     temp->prevCMCB = NULL;
285                     freeList.count--;
286                 }
287                 temp = temp->nextCMCB;
288             }
289         }
290         else
291         {
292             freeList.head = temp;
293             freeList.tail = temp;
294             freeList.count = 1;
295         }
296 
297     } // end of else statement to free memory.
298     return 0;
299 } // end of Function.
```

5.34.1.3 initializeHeap()

```
u32int initializeHeap (
            u32int heapSize )
```

Definition at line 20 of file R5commands.c.

```
21 {
22     u32int memStart = kmalloc(heapSize + sizeof(CMCB));
```

```
23      CMCB *temp = (CMCB *)memStart;
24
25      // Create the first free block
26      temp->type = 'f';
27      temp->beginningAddr = memStart + sizeof(CMCB);
28      temp->size = heapSize;
29      //strcpy(temp->name, "first");
30      temp->nextCMCB = NULL;
31      temp->prevCMCB = NULL;
32
33      // Initialize alllocated list
34      allocatedList.count = 0;
35      allocatedList.head = NULL;
36      allocatedList.tail = NULL;
37
38      // Place first free block into the free list
39      freeList.count++;
40      freeList.head = temp;
41      freeList.tail = temp;
42
43      return memStart;
44  }
```

5.34.1.4 insertToList()

```
void insertToList (
              CMCB * current,
              memList * list )
```

Definition at line 46 of file R5commands.c.

```
47  {
48      if (list->head == NULL) // current is put into an empty list.
49      {
50          list->head = current;
51          list->tail = current;
52          list->count++;
53      }
54      else if (current->beginningAddr < list->head->beginningAddr) // current goes at the start of the
        list.
55      {
56          current->nextCMCB = list->head;
57          list->head->prevCMCB = current;
58          list->head = current;
59          list->count++;
60      }
61      else if (current->beginningAddr > list->tail->beginningAddr) // current goes at the end of the list.
62      {
63          current->prevCMCB = list->tail;
64          list->tail->nextCMCB = current;
65          list->tail = current;
66          list->count++;
67      }
68      else // current goes in the middle of list.
69      {
70          CMCB *temp = list->head;
71          while (temp != NULL)
72          {
73              if (current->beginningAddr < temp->beginningAddr)
74              {
75                  current->nextCMCB = temp;
76                  current->prevCMCB = temp->prevCMCB;
77                  temp->prevCMCB->nextCMCB = current;
78                  temp->prevCMCB = current;
79                  list->count++;
80                  break;
81              }
82              else
83              {
84                  temp = temp->nextCMCB;
85              }
86          }
87      }
88  }
```

5.34.1.5 isEmpty()

```
int isEmpty ( )
```

Definition at line 301 of file R5commands.c.

```
302 {
303     if (allocatedList.head == NULL && freeList.count == 1)
304     {
305         printMessage("The allocated list is empty.\n");
306         return TRUE;
307     }
308     else
309     {
310         printMessage("The allocated list is not empty.\n");
311         return FALSE;
312     }
313 }
```

5.34.1.6 removeFromAlloc()

```
void removeFromAlloc (
              CMCB * temp )
```

Definition at line 189 of file R5commands.c.

```
190 {
191     // Remove temp from allocatedList
192     if (temp == allocatedList.head)
193     {
194         allocatedList.head = temp->nextCMCB;
195         temp->nextCMCB = NULL;
196         allocatedList.count--;
197     }
198     else if (temp == allocatedList.tail)
199     {
200         allocatedList.tail = temp->prevCMCB;
201         allocatedList.tail->nextCMCB = NULL;
202         temp->prevCMCB = NULL;
203         allocatedList.count--;
204     }
205     else
206     {
207         temp->prevCMCB->nextCMCB = temp->nextCMCB;
208         temp->nextCMCB->prevCMCB = temp->prevCMCB;
209         temp->nextCMCB = NULL;
210         temp->prevCMCB = NULL;
211         allocatedList.count--;
212     }
213 }
```

5.34.1.7 showAllocatedMemory()

```
void showAllocatedMemory ( )
```

Definition at line 363 of file R5commands.c.

```
364 {
365     if (allocatedList.head == NULL)
366     {
367         printMessage("There is no allocated memory!\n");
368     }
369
370     CMCB *temp = allocatedList.head;
371     while (temp != NULL)
372     {
373         showMCB(temp);
374         temp = temp->nextCMCB;
375     }
376 }
```

5.34.1.8 showFreeMemory()

```
void showFreeMemory ( )
```

Definition at line 348 of file R5commands.c.

```
349 {
350     if (freeList.head == NULL)
351     {
352         printMessage("There is no free memory!\n");
353     }
354 
355     CMCB *temp = freeList.head;
356     while (temp != NULL)
357     {
358         showMCB(temp);
359         temp = temp->nextCMCB;
360     }
361 }
```

5.34.1.9 showMCB()

```
void showMCB (
              CMCB * mem )
```

Definition at line 315 of file R5commands.c.

```
316 {
317     int sizeLen;
318 
319     // Print the block type.
320     if (mem->type == 'a')
321     {
322         printMessage("The CMCBs type is: allocated.\n");
323     }
324     else if (mem->type == 'f')
325     {
326         printMessage("The CMCBs type is: free.\n");
327     }
328 
329     // Print the block size.
330     char size[20];
331     memset(size, '\0', 20);
332     strcpy(size, itoa(mem->size, size));
333     sizeLen = strlen(size);
334     printMessage("The size is: ");
335     sys_req(WRITE, DEFAULT_DEVICE, size, &sizeLen);
336     printMessage(" bytes.\n");
337 
338     // Print the block beginning address.
339     char temp[20];
340     memset(temp, '\0', 20);
341     strcpy(temp, itoa(mem->beginningAddr, temp));
342     sizeLen = strlen(temp);
343     printMessage("The beginning address of the block is: ");
344     sys_req(WRITE, DEFAULT_DEVICE, temp, &sizeLen);
345     printMessage(".\n\n");
346 }
```

5.34.2 Variable Documentation

5.34.2.1 allocatedList

```
memList allocatedList
```

Definition at line 18 of file R5commands.c.

5.34.2.2 freeList

```
memList freeList
```

Definition at line 17 of file R5commands.c.

5.35 modules/R5/R5commands.h File Reference

Classes

- struct CMCB
- struct memList

Typedefs

- typedef struct CMCB CMCB
- typedef struct memList memList

Functions

- u32int initializeHeap (u32int heapSize)
- u32int allocateMemory (u32int size)
- int freeMemory (void ∗memToFree)
- int isEmpty ()
- void showFreeMemory ()
- void showAllocatedMemory ()

5.35.1 Typedef Documentation

5.35.1.1 CMCB

```
typedef struct CMCB CMCB
```

5.35.1.2 memList

```
typedef struct memList memList
```

5.35.2 Function Documentation

5.35.2.1 allocateMemory()

```
u32int allocateMemory (
            u32int size )
```

Definition at line 90 of file R5commands.c.

```
91 {
92     if (freeList.head != NULL)
93     {
94         CMCB *current = freeList.head;
95 
96         // get to block of appropriate size
97         while (current != NULL)
98         {
99             if (current->size == size + sizeof(CMCB) && freeList.count == 1)
100             {
101                 // remove from free list.
102                 current->nextCMCB->prevCMCB = current->prevCMCB;
103                 current->prevCMCB->nextCMCB = current->nextCMCB;
104                 current->nextCMCB = NULL;
105                 current->prevCMCB = NULL;
106                 // place current in alloc list.
107                 insertToList(current, &allocatedList);
108 
109                 // change current marker to 'a'.
110                 current->type = 'a';
111 
112                 // remove all freeList pointers to current.
113                 freeList.head = NULL;
114                 freeList.tail = NULL;
115 
116                 // return allocated block.
117                 return current->beginningAddr;
118             }
119             else if (current->size == size + sizeof(CMCB)) // current is excetly the size requested.
120             {
121                 // remove from free list.
122                 current->nextCMCB->prevCMCB = current->prevCMCB;
123                 current->prevCMCB->nextCMCB = current->nextCMCB;
124                 current->nextCMCB = NULL;
125                 current->prevCMCB = NULL;
126                 // place current in alloc list.
127                 insertToList(current, &allocatedList);
128 
129                 // change current marker to 'a'.
130                 current->type = 'a';
131 
132                 // return allocated block.
133                 return current->beginningAddr;
134             }
135             else if (current->size > size + sizeof(CMCB)) // current is greater than the size requested
136             {
137                 // remove from free list.
138                 CMCB *new = (CMCB *)(current->beginningAddr + size);                // This CMCB
    pertains to the head of the free list at the new memory address
139                 new->beginningAddr = (current->beginningAddr + size + sizeof(CMCB)); // Could be
    tmp->beginningAddr + size + sizeof(CMCB)
140                 new->size = current->size - size - sizeof(CMCB);
141                 new->type = 'f';
142                 new->nextCMCB = current->nextCMCB;
143                 new->prevCMCB = current->prevCMCB;
144 
145                 if (current->prevCMCB != NULL)
146                 {
147                     new->prevCMCB->nextCMCB = new;
148                 }
149 
150                 if (current->nextCMCB != NULL)
151                 {
152                     new->nextCMCB->prevCMCB = new;
153                 }
154 
155                 if (freeList.head == current && freeList.tail == current)
156                 {
157                     freeList.head = new;
158                     freeList.tail = new;
159                 }
160                 else if (freeList.head == current)
161                 {
162                     freeList.head = new;
163                 }
164                 else if (freeList.tail == current)
165                 {
166                     freeList.tail = new;
```

```
167                 }
168
169                 current->size = size;
170                 current->nextCMCB = NULL;
171                 current->prevCMCB = NULL;
172
173                 // place current in alloc list.
174                 insertToList(current, &allocatedList);
175
176                 // change current marker to 'a'.
177                 current->type = 'a';
178
179                 // return allocated block.
180                 return current->beginningAddr;
181             }
182             current = current->nextCMCB;
183         }
184     }
185
186     return NULL;
187 }
```

5.35.2.2 freeMemory()

```
int freeMemory (
            void * memToFree )
```

Definition at line 215 of file R5commands.c.

```
216 {
217     if (isEmpty())
218     {
219         printMessage("There is no memory to free!\n");
220         return 1;
221     }
222
223     CMCB *temp = allocatedList.head;
224
225     while (temp->beginningAddr != (u32int)memToFree)
226     {
227         temp = temp->nextCMCB;
228     }
229
230     if (temp == NULL)
231     {
232         printMessage("There is no allocated memory at that address!\n");
233         return 1;
234     }
235     else
236     {
237
238         // Remove memToFree from the allocatedList.
239         removeFromAlloc(temp);
240
241         // Insert memToFree into the freeList in increasing order.
242         insertToList(temp, &freeList);
243         temp->type = 'f';
244
245         // Merge memToFree to other free CMCBs if possible.
246         if (freeList.count >= 1)
247         {
248             CMCB *temp = freeList.head;
249             while (temp != NULL)
250             {
251                 if ((temp->beginningAddr + temp->size) == (temp->nextCMCB->beginningAddr -
      sizeof(CMCB))) // merge down
252                 {
253                     printMessage("Memory merge down\n");
254                     if (temp->nextCMCB->nextCMCB != NULL)
255                     {
256                         CMCB *next = temp->nextCMCB;
257                         temp->size += (next->size + sizeof(CMCB));
258                         temp->nextCMCB = next->nextCMCB;
259                         next->nextCMCB->prevCMCB = temp;
260                         next->prevCMCB = NULL;
261                         next->nextCMCB = NULL;
262                         freeList.count--;
263                     }
264                     else
```

```
265                     {
266                         printMessage("Merge down part 2\n");
267                         CMCB *next = temp->nextCMCB;
268                         temp->size += (next->size + sizeof(CMCB));
269                         next->prevCMCB = NULL;
270                         next->nextCMCB = NULL;
271                         temp->nextCMCB = NULL;
272                         freeList.count--;
273                     }
274                 }
275 
276                 if ((temp->prevCMCB->beginningAddr + temp->prevCMCB->size) == (temp->beginningAddr -
    sizeof(CMCB))) //merge up
277                 {
278                     printMessage("Memory merge up\n");
279                     CMCB *prev = temp->prevCMCB;
280                     prev->size += (temp->size + sizeof(CMCB));
281                     prev->nextCMCB = temp->nextCMCB;
282                     temp->nextCMCB = prev;
283                     temp->nextCMCB = NULL;
284                     temp->prevCMCB = NULL;
285                     freeList.count--;
286                 }
287                 temp = temp->nextCMCB;
288             }
289         }
290         else
291         {
292             freeList.head = temp;
293             freeList.tail = temp;
294             freeList.count = 1;
295         }
296 
297     } // end of else statement to free memory.
298     return 0;
299 } // end of Function.
```

5.35.2.3 initializeHeap()

```
u32int initializeHeap (
              u32int heapSize )
```

Definition at line 20 of file R5commands.c.

```
21 {
22     u32int memStart = kmalloc(heapSize + sizeof(CMCB));
23     CMCB *temp = (CMCB *)memStart;
24 
25     // Create the first free block
26     temp->type = 'f';
27     temp->beginningAddr = memStart + sizeof(CMCB);
28     temp->size = heapSize;
29     //strcpy(temp->name, "first");
30     temp->nextCMCB = NULL;
31     temp->prevCMCB = NULL;
32 
33     // Initialize alllocated list
34     allocatedList.count = 0;
35     allocatedList.head = NULL;
36     allocatedList.tail = NULL;
37 
38     // Place first free block into the free list
39     freeList.count++;
40     freeList.head = temp;
41     freeList.tail = temp;
42 
43     return memStart;
44 }
```

5.35.2.4 isEmpty()

```
int isEmpty ( )
```

Definition at line 301 of file R5commands.c.

```
302 {
303     if (allocatedList.head == NULL && freeList.count == 1)
304     {
305         printMessage("The allocated list is empty.\n");
306         return TRUE;
307     }
308     else
309     {
310         printMessage("The allocated list is not empty.\n");
311         return FALSE;
312     }
313 }
```

5.35.2.5 showAllocatedMemory()

```
void showAllocatedMemory ( )
```

Definition at line 363 of file R5commands.c.

```
364 {
365     if (allocatedList.head == NULL)
366     {
367         printMessage("There is no allocated memory!\n");
368     }
369
370     CMCB *temp = allocatedList.head;
371     while (temp != NULL)
372     {
373         showMCB(temp);
374         temp = temp->nextCMCB;
375     }
376 }
```

5.35.2.6 showFreeMemory()

```
void showFreeMemory ( )
```

Definition at line 348 of file R5commands.c.

```
349 {
350     if (freeList.head == NULL)
351     {
352         printMessage("There is no free memory!\n");
353     }
354
355     CMCB *temp = freeList.head;
356     while (temp != NULL)
357     {
358         showMCB(temp);
359         temp = temp->nextCMCB;
360     }
361 }
```

5.36 modules/utilities.c File Reference

```
#include <core/serial.h>
#include <string.h>
#include "mpx_supt.h"
#include <core/io.h>
#include <mem/heap.h>
#include <system.h>
```

Functions

- char * reverseStr (char *str)
- char * itoa (int num, char *buffer)
- void printMessage (char *str)

5.36.1 Function Documentation

5.36.1.1 itoa()

```
char* itoa (
            int num,
            char * buffer )
```

Definition at line 26 of file utilities.c.

```
27 {
28     int i = 0;
29     int neg = FALSE;
30 
31     if (num == 0)
32     {
33         buffer[i] = '0';
34         buffer[++i] = '\0';
35     }
36 
37     if (num < 0)
38     {
39         neg = TRUE;
40         num = -num;
41     }
42 
43     do
44     {
45         buffer[i++] = (num % 10) + '0';
46     } while ((num /= 10) > 0);
47 
48     if (neg == TRUE)
49     {
50         buffer[i++] = '-';
51     }
52 
53     buffer = reverseStr(buffer);
54     buffer[i] = '\0';
55 
56     return buffer;
57 }
```

5.36.1.2 printMessage()

```
void printMessage (
            char * str )
```

Definition at line 59 of file utilities.c.

```
60 {
61     char Desc[137];
62 
63     size_t length = strlen(str);
64     if (length > (sizeof(Desc) - 2))
65     {
66         length = sizeof(Desc) - 2;
67         Desc[sizeof(Desc) - 1] = '\0';
68     }
69     strcpy(Desc, str);
70     int tempBuffer = strlen(Desc);
71     sys_req(WRITE, DEFAULT_DEVICE, (char *)Desc, &tempBuffer);
72 }
```

5.36.1.3 reverseStr()

```
char* reverseStr (
            char * str )
```

Definition at line 8 of file utilities.c.

```
9 {
10     int size = strlen(str);
11     char temp[size];
12
13
14     int i = 0;
15     while (size >= 0)
16     {
17         temp[i] = str[size - 1];
18         size--;
19         i++;
20     }
21     char* test= temp;
22
23     return test;
24 }
```

5.37 modules/utilities.h File Reference

Functions

- char * reverseStr (char *str)
- char * itoa (int num, char *buffer)
- void printMessage (char *str)

5.37.1 Function Documentation

5.37.1.1 itoa()

```
char* itoa (
            int num,
            char * buffer )
```

Definition at line 26 of file utilities.c.

```
27 {
28     int i = 0;
29     int neg = FALSE;
30
31     if (num == 0)
32     {
33         buffer[i] = '0';
34         buffer[++i] = '\0';
35     }
36
37     if (num < 0)
38     {
39         neg = TRUE;
40         num = -num;
41     }
42
43     do
44     {
45         buffer[i++] = (num % 10) + '0';
46     } while ((num /= 10) > 0);
47
48     if (neg == TRUE)
```

```
49     {
50         buffer[i++] = '-';
51     }
52 
53     buffer = reverseStr(buffer);
54     buffer[i] = '\0';
55 
56     return buffer;
57 }
```

5.37.1.2 printMessage()

```
void printMessage (
              char * str )
```

Definition at line 59 of file utilities.c.

```
60 {
61     char Desc[137];
62 
63     size_t length = strlen(str);
64     if (length > (sizeof(Desc) - 2))
65     {
66         length = sizeof(Desc) - 2;
67         Desc[sizeof(Desc) - 1] = '\0';
68     }
69     strcpy(Desc, str);
70     int tempBuffer = strlen(Desc);
71     sys_req(WRITE, DEFAULT_DEVICE, (char *)Desc, &tempBuffer);
72 }
```

5.37.1.3 reverseStr()

```
char* reverseStr (
              char * str )
```

Definition at line 8 of file utilities.c.

```
9 {
10     int size = strlen(str);
11     char temp[size];
12 
13 
14     int i = 0;
15     while (size >= 0)
16     {
17         temp[i] = str[size - 1];
18         size--;
19         i++;
20     }
21     char* test= temp;
22 
23     return test;
24 }
```

5.38 README.md File Reference