Runtime Terror

Generated by Doxygen 1.9.1

Chapter 1

CS_450_RunTime_Terror

R1 Implementation

Bonus assignments included are -Variable Text Color -itoa funciton

Chapter 2

Class Index

2.1 Class List

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

date_time . . . ??
footer . . . ??
gdt_descriptor_struct . . . ??
gdt_entry_struct . . . ??
header . . . ??
heap . . . ??
idt_entry_struct . . . ??
idt_struct . . . ??
index_entry . . . ??
index_table . . . ??
page_dir . . . ??
page_entry . . . ??
page_table . . . ??
param . . . ??
PCB . . . ??
Queue . . . ??
struct . . . ??

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

mpx_core/include/string.h . . . ??
mpx_core/include/system.h . . . ??
mpx_core/include/core/asm.h . . . ??
mpx_core/include/core/interrupts.h . . . ??
mpx_core/include/core/io.h . . . ??
mpx_core/include/core/serial.h . . . ??
mpx_core/include/core/tables.h . . . ??
mpx_core/include/mem/heap.h . . . ??
mpx_core/include/mem/paging.h . . . ??
mpx_core/kernel/core/interrupts.c . . . ??
mpx_core/kernel/core/kmain.c . . . ??
mpx_core/kernel/core/serial.c . . . ??
mpx_core/kernel/core/system.c . . . ??
mpx_core/kernel/core/tables.c . . . ??
mpx_core/kernel/mem/heap.c . . . ??
mpx_core/kernel/mem/paging.c . . . ??
mpx_core/lib/string.c . . . ??
mpx_core/modules/mpx_supt.c . . . ??
mpx_core/modules/mpx_supt.h . . . ??
mpx_core/modules/R1/comHand.c . . . ??
mpx_core/modules/R1/comHand.h . . . ??
mpx_core/modules/R1/userFunctions.c . . . ??
mpx_core/modules/R1/userFunctions.h . . . ??
mpx_core/modules/R2/PCB.c . . . ??
mpx_core/modules/R2/PCB.h . . . ??

Chapter 4

Class Documentation

4.1 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

4.1.1 Detailed Description

Definition at line 32 of file system.h.

4.1.2 Member Data Documentation

4.1.2.1 day_m

`int date_time::day_m`

Definition at line 37 of file system.h.

4.1.2.2 day_w

```
int date_time::day_w
```

Definition at line 36 of file system.h.

4.1.2.3 day_y

```
int date_time::day_y
```

Definition at line 38 of file system.h.

4.1.2.4 hour

```
int date_time::hour
```

Definition at line 35 of file system.h.

4.1.2.5 min

```
int date_time::min
```

Definition at line 34 of file system.h.

4.1.2.6 mon

```
int date_time::mon
```

Definition at line 39 of file system.h.

4.1.2.7 sec

```
int date_time::sec
```

Definition at line 33 of file system.h.

4.1.2.8 year

```
int date_time::year
```

Definition at line 40 of file system.h.

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

- mpx_core/include/system.h

4.2 footer Struct Reference

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

Public Attributes

- header head

4.2.1 Detailed Description

Definition at line 18 of file heap.h.

4.2.2 Member Data Documentation

4.2.2.1 head

```
header footer::head
```

Definition at line 19 of file heap.h.

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

- mpx_core/include/mem/heap.h

4.3 gdt_descriptor_struct Struct Reference

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

Public Attributes

- u16int limit
- u32int base

4.3.1 Detailed Description

Definition at line 25 of file tables.h.

4.3.2 Member Data Documentation

4.3.2.1 base

```
u32int gdt_descriptor_struct::base
```

Definition at line 28 of file tables.h.

4.3.2.2 limit

```
u16int gdt_descriptor_struct::limit
```

Definition at line 27 of file tables.h.

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

- mpx_core/include/core/tables.h

4.4 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

4.4.1 Detailed Description

Definition at line 32 of file tables.h.

4.4.2 Member Data Documentation

4.4.2.1 access

```
u8int gdt_entry_struct::access
```

Definition at line 37 of file tables.h.

4.4.2.2 base_high

```
u8int gdt_entry_struct::base_high
```

Definition at line 39 of file tables.h.

4.4.2.3 base_low

```
u16int gdt_entry_struct::base_low
```

Definition at line 35 of file tables.h.

4.4.2.4 base_mid

```
u8int gdt_entry_struct::base_mid
```

Definition at line 36 of file tables.h.

4.4.2.5 flags

```
u8int gdt_entry_struct::flags
```

Definition at line 38 of file tables.h.

4.4.2.6 limit_low

```
u16int gdt_entry_struct::limit_low
```

Definition at line 34 of file tables.h.

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

- mpx_core/include/core/tables.h

4.5 header Struct Reference

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

Public Attributes

- int size
- int index_id

4.5.1 Detailed Description

Definition at line 13 of file heap.h.

4.5.2 Member Data Documentation

4.5.2.1 index_id

```
int header::index_id
```

Definition at line 15 of file heap.h.

4.5.2.2 size

```
int header::size
```

Definition at line 14 of file heap.h.

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

- mpx_core/include/mem/heap.h

4.6 heap Struct Reference

`#include <heap.h>`

Public Attributes

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

4.6.1 Detailed Description

Definition at line 35 of file heap.h.

4.6.2 Member Data Documentation

4.6.2.1 base

`u32int heap::base`

Definition at line 37 of file heap.h.

4.6.2.2 index

`index_table heap::index`

Definition at line 36 of file heap.h.

4.6.2.3 max_size

`u32int heap::max_size`

Definition at line 38 of file heap.h.

4.6.2.4 min_size

```
u32int heap::min_size
```

Definition at line 39 of file heap.h.

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

- mpx_core/include/mem/heap.h

4.7 idt_entry_struct Struct Reference

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

Public Attributes

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

4.7.1 Detailed Description

Definition at line 8 of file tables.h.

4.7.2 Member Data Documentation

4.7.2.1 base_high

```
u16int idt_entry_struct::base_high
```

Definition at line 14 of file tables.h.

4.7.2.2 base_low

```
u16int idt_entry_struct::base_low
```

Definition at line 10 of file tables.h.

4.7.2.3 flags

```
u8int idt_entry_struct::flags
```

Definition at line 13 of file tables.h.

4.7.2.4 sselect

```
u16int idt_entry_struct::sselect
```

Definition at line 11 of file tables.h.

4.7.2.5 zero

```
u8int idt_entry_struct::zero
```

Definition at line 12 of file tables.h.

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

- mpx_core/include/core/tables.h

4.8 idt_struct Struct Reference

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

Public Attributes

- u16int limit
- u32int base

4.8.1 Detailed Description

Definition at line 18 of file tables.h.

4.8.2 Member Data Documentation

4.8.2.1 base

```
u32int idt_struct::base
```

Definition at line 21 of file tables.h.

4.8.2.2 limit

```
u16int idt_struct::limit
```

Definition at line 20 of file tables.h.

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

- mpx_core/include/core/tables.h

4.9 index_entry Struct Reference

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

Public Attributes

- int size
- int empty
- u32int block

4.9.1 Detailed Description

Definition at line 22 of file heap.h.

4.9.2 Member Data Documentation

4.9.2.1 block

```
u32int index_entry::block
```

Definition at line 25 of file heap.h.

4.9.2.2 empty

```
int index_entry::empty
```

Definition at line 24 of file heap.h.

4.9.2.3 size

```
int index_entry::size
```

Definition at line 23 of file heap.h.

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

- mpx_core/include/mem/heap.h

4.10 index_table Struct Reference

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

Public Attributes

- index_entry table [TABLE_SIZE]
- int id

4.10.1 Detailed Description

Definition at line 29 of file heap.h.

4.10.2 Member Data Documentation

4.10.2.1 id

```
int index_table::id
```

Definition at line 31 of file heap.h.

4.10.2.2 table

```
index_entry index_table::table[TABLE_SIZE]
```

Definition at line 30 of file heap.h.

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

- mpx_core/include/mem/heap.h

4.11 page_dir Struct Reference

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

Public Attributes

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

4.11.1 Detailed Description

Definition at line 36 of file paging.h.

4.11.2 Member Data Documentation

4.11.2.1 tables

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

Definition at line 37 of file paging.h.

4.11.2.2 tables_phys

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

Definition at line 38 of file paging.h.

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

- mpx_core/include/mem/paging.h

4.12 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

4.12.1 Detailed Description

Definition at line 14 of file paging.h.

4.12.2 Member Data Documentation

4.12.2.1 accessed

`u32int page_entry::accessed`

Definition at line 18 of file paging.h.

4.12.2.2 dirty

`u32int page_entry::dirty`

Definition at line 19 of file paging.h.

4.12.2.3 frameaddr

`u32int page_entry::frameaddr`

Definition at line 21 of file paging.h.

4.12.2.4 present

```
u32int page_entry::present
```

Definition at line 15 of file paging.h.

4.12.2.5 reserved

```
u32int page_entry::reserved
```

Definition at line 20 of file paging.h.

4.12.2.6 usermode

```
u32int page_entry::usermode
```

Definition at line 17 of file paging.h.

4.12.2.7 writeable

```
u32int page_entry::writeable
```

Definition at line 16 of file paging.h.

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

- mpx_core/include/mem/paging.h

4.13 page_table Struct Reference

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

Public Attributes

- page_entry pages [1024]

4.13.1 Detailed Description

Definition at line 28 of file paging.h.

4.13.2 Member Data Documentation

4.13.2.1 pages

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

Definition at line 29 of file paging.h.

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

- mpx_core/include/mem/paging.h

4.14 param Struct Reference

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

Public Attributes

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

4.14.1 Detailed Description

Definition at line 33 of file mpx_supt.h.

4.14.2 Member Data Documentation

4.14.2.1 buffer_ptr

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

Definition at line 36 of file mpx_supt.h.

4.14.2.2 count_ptr

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

Definition at line 37 of file mpx_supt.h.

4.14.2.3 device_id

```
int param::device_id
```

Definition at line 35 of file mpx_supt.h.

4.14.2.4 op_code

```
int param::op_code
```

Definition at line 34 of file mpx_supt.h.

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

- mpx_core/modules/mpx_supt.h

4.15 PCB Struct Reference

Public Attributes

- unsigned char stack [1KMEM]
- unsigned char * stackTop
- struct PCB * prev
- struct PCB * next
- char Process_Name [10]
- int Process_Class
- int Priority
- int ReadyState
- int SuspendedState

4.15.1 Detailed Description

Definition at line 27 of file PCB.c.

4.15.2 Member Data Documentation

4.15.2.1 next

```
struct PCB* PCB::next
```

Definition at line 31 of file PCB.c.

4.15.2.2 prev

```
struct PCB* PCB::prev
```

Definition at line 30 of file PCB.c.

4.15.2.3 Priority

```
int PCB::Priority
```

Definition at line 34 of file PCB.c.

4.15.2.4 Process_Class

```
int PCB::Process_Class
```

Definition at line 33 of file PCB.c.

4.15.2.5 Process_Name

```
char PCB::Process_Name[10]
```

Definition at line 32 of file PCB.c.

4.15.2.6 ReadyState

```
int PCB::ReadyState
```

Definition at line 35 of file PCB.c.

4.15.2.7 stack

```
unsigned char PCB::stack[1KMEM]
```

Definition at line 28 of file PCB.c.

4.15.2.8 stackTop

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

Definition at line 29 of file PCB.c.

4.15.2.9 SuspendedState

```
int PCB::SuspendedState
```

Definition at line 36 of file PCB.c.

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

- mpx_core/modules/R2/PCB.c

4.16 Queue Struct Reference

Public Attributes

- int count
- PCB * head
- PCB * tail

4.16.1 Detailed Description

Definition at line 7 of file PCB.c.

4.16.2 Member Data Documentation

4.16.2.1 count

```
int Queue::count
```

Definition at line 8 of file PCB.c.

4.16.2.2 head

```
PCB* Queue::head
```

Definition at line 9 of file PCB.c.

4.16.2.3 tail

```
PCB* Queue::tail
```

Definition at line 10 of file PCB.c.

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

- mpx_core/modules/R2/PCB.c

4.17 struct Struct Reference

```
#include <PCB.h>
```

Public Attributes

- int count

4.17.1 Detailed Description

Definition at line 9 of file PCB.h.

4.17.2 Member Data Documentation

4.17.2.1 count

```
int struct::count
```

Definition at line 11 of file PCB.h.

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

- mpx_core/modules/R2/PCB.h

Chapter 5

File Documentation

5.1 mpx_core/include/core/asm.h File Reference

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

5.2 mpx_core/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
92    // Install handlers; 0x08=sel, 0x8e=flags
93    for(i=0; i<32; i++){
94      if (i<17) idt_set_gate(i, isrs[i], 0x08, 0x8e);
95      else idt_set_gate(i, (u32int)reserved, 0x08, 0x8e);
96    }
97    // Ignore interrupts from the real time clock
98    idt_set_gate(0x08, (u32int)rtc_isr, 0x08, 0x8e);
99  }
```

5.2.1.2 init_pic()

```
void init_pic (
              void  )
```

Definition at line 107 of file interrupts.c.

```
108 {
109   outb(PIC1,ICW1);   //send initialization code words 1 to PIC1
110   io_wait();
111   outb(PIC2,ICW1);   //send icw1 to PIC2
112   io_wait();
113   outb(PIC1+1,0x20); //icw2: remap irq0 to 32
114   io_wait();
115   outb(PIC2+1,0x28); //icw2: remap irq8 to 40
116   io_wait();
117   outb(PIC1+1,4);    //icw3
118   io_wait();
119   outb(PIC2+1,2);    //icw3
120   io_wait();
121   outb(PIC1+1,ICW4); //icw4: 80x86, automatic handling
122   io_wait();
123   outb(PIC2+1,ICW4); //icw4: 80x86, automatic handling
124   io_wait();
125   outb(PIC1+1,0xFF); //disable irqs for PIC1
126   io_wait();
127   outb(PIC2+1,0xFF); //disable irqs for PIC2
128 }
```

5.3 mpx_core/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 17 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 10 of file io.h.

5.4 mpx_core/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 6 of file serial.h.

5.4.1.2 COM2

```
#define COM2 0x2f8
```

Definition at line 7 of file serial.h.

5.4.1.3 COM3

```
#define COM3 0x3e8
```

Definition at line 8 of file serial.h.

5.4.1.4 COM4

```
#define COM4 0x2e8
```

Definition at line 9 of file serial.h.

5.4.2 Function Documentation

5.4.2.1 init_serial()

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

Definition at line 28 of file serial.c.

```
28                              {
29          outb(device + 1, 0x00); //disable interrupts
30          outb(device + 3, 0x80); //set line control register
31          outb(device + 0, 115200 / 9600); //set bsd least sig bit
32          outb(device + 1, 0x00); //brd most significant bit
33          outb(device + 3, 0x03); //lock divisor; 8bits, no parity, one stop
34          outb(device + 2, 0xC7); //enable fifo, clear, 14byte threshold
35          outb(device + 4, 0x0B); //enable interrupts, rts/dsr set
36          (void) inb(device); //read bit to reset port
37          return NO_ERROR;
38  }
```

5.4.2.2 polling()

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

Definition at line 95 of file serial.c.

```
95                                              {
96          int pointerLoc = 0;
97          int numCharacters = 0;
98          int flag = 1;
99          char letter = NULL;
100          while (flag) { // Run continuously
101
102                 if (inb(COM1 + 5) & 1) { // Is a character available?
103                         letter = inb(COM1); //Get the character
104
105                         //Special Cases
106
107                         //ENTER
108                         if (letter == '\n' || letter == '\r') {
109                                 cmdBuffer[pointerLoc] = '\0';
110                                 flag = 0;
111                                 serial_print("\n");
112                         }
113
114                         else if (letter == '\033') {
115                                 letter = inb(COM1);
116                                 if (letter == '[') {
117                                         letter = inb(COM1);
118
119                                         //Right Arrow Case
120                                         if (letter == 'C') {
121                                                 if (pointerLoc < numCharacters) {
122                                                         pointerLoc++;
123                                                         serial_print("\033[C");
124                                                 }
125                                         }
126
127                                         //Left Arrow Case
128                                         else if (letter == 'D') {
129                                                 if (pointerLoc > 0) {
130                                                         pointerLoc--;
131                                                         serial_print("\033[D");
132                                                 }
133
134                                         }
135                                         else if (letter == 'A') {
136                                         //up
137                                         }
138                                         else if (letter == 'B') {
139                                         //down
140                                         }
141
142                                         //DELETE
143                                         else if (letter == '3') {
144                                                 letter = inb(COM1);
145                                                 if (letter == '~') {
146                                                         if (pointerLoc < numCharacters) {
147                                                                 int bufIndex;
148                                                                 for (bufIndex = pointerLoc; bufIndex <
      *count; bufIndex++) {
149                                                                         cmdBuffer[bufIndex] =
      cmdBuffer[bufIndex + 1];
150                                                                 }
151                                                                 serial_print("\033[1P");
152                                                                 numCharacters--;
153                                                                 inb(COM1);
154                                                         }
155                                                 }
156                                         }
157                                 }
158                         }
159
160                         //BACKSPACE
161                         else if (letter == 127) {
162                                 if(pointerLoc > 0){
163                                         if(pointerLoc > numCharacters){
164                                                 cmdBuffer[pointerLoc - 1] = NULL;
165                                         }
166                                         else{
167                                                 int bufIndex;
168                                                 for (bufIndex = pointerLoc; bufIndex <= numCharacters;
      bufIndex++) {
```

```
169                                                     cmdBuffer[bufIndex-1] = cmdBuffer[bufIndex];
    //replaces the last typed character with null.
170                                                 }
171                                         }
172                                         numCharacters--;
173                                         pointerLoc--;
174                                         serial_print("\033[D\033[P");
175                                         inb(COM1);
176                                 }
177                         }
178
179
180                         //passes any other characters 0-9,a-z, upper and lower case to the command
    handler to be dealt with.
181                         else {
182                                 if (numCharacters < * count) {
183                                         if(pointerLoc < numCharacters) {
184                                                 int bufIndex;
185                                                 for(bufIndex = numCharacters + 1; bufIndex > pointerLoc;
    bufIndex--)
186                                                 {
187                                                         cmdBuffer[bufIndex] = cmdBuffer[bufIndex - 1];
188                                                 }
189                                                 cmdBuffer[pointerLoc] = letter;
190                                                 numCharacters++; //increments the total number of
    characters passed in so far.
191                                                 pointerLoc++; //increments the pointer location per
    input.
192
193                                                 //int i = 0;
194                                                 // for(i = 0; i <= numCharacters + 1)
195                                                 serial_print("\033[s\033[K");
196                                                 serial_print(&cmdBuffer[pointerLoc-1]);
197                                                 serial_print("\033[u\033[C");
198                                         }
199                                         else {
200                                               cmdBuffer[pointerLoc] = letter;
201                                               serial_print(&cmdBuffer[pointerLoc]);
202                                               pointerLoc++; //increments the pointer location per input.
203                                               numCharacters++; //increments the total number of
    characters passed in so far.
204                                         }
205                                 }
206                         }
207                 }
208         }
209         return 0;
210 }
```

5.4.2.3 serial_print()

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

Definition at line 59 of file serial.c.

```
59                                      {
60         int i;
61         for (i = 0;*(i + msg) != '\0'; i++) {
62                 outb(serial_port_out, *(i + msg));
63         }
64         if ( * msg == '\r') 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 45 of file serial.c.

```
45                                      {
46         int i;
47         for (i = 0;*(i + msg) != '\0'; i++) {
48                 outb(serial_port_out, *(i + msg));
49         }
50         outb(serial_port_out, '\r');
51         outb(serial_port_out, '\n');
52         return NO_ERROR;
53 }
```

5.4.2.5 set_serial_in()

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

Definition at line 85 of file serial.c.

```
85                                      {
86         serial_port_in = device;
87         return NO_ERROR;
88 }
```

5.4.2.6 set_serial_out()

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

Definition at line 74 of file serial.c.

```
74                                      {
75         serial_port_out = device;
76         return NO_ERROR;
77 }
```

5.5 mpx_core/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 59 of file tables.c.

```
61 {
62   gdt_entry *new_entry = &gdt_entries[idx];
63   new_entry->base_low  = (base & 0xFFFF);
64   new_entry->base_mid  = (base » 16) & 0xFF;
65   new_entry->base_high = (base » 24) & 0xFF;
66   new_entry->limit_low = (limit & 0xFFFF);
67   new_entry->flags  = (limit » 16) & 0xFF;
68   new_entry->flags |= flags & 0xF0;
69   new_entry->access = access;
70 }
```

5.5.1.3 idt_set_gate()

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

Definition at line 29 of file tables.c.

```
31 {
32   idt_entry *new_entry = &idt_entries[idx];
33   new_entry->base_low  = (base &  0xFFFF);
34   new_entry->base_high = (base » 16) & 0xFFFF;
35   new_entry->sselect   = sel;
36   new_entry->zero = 0;
37   new_entry->flags = flags;
38 }
```

5.5.1.4 init_gdt()

```
void init_gdt ( )
```

Definition at line 77 of file tables.c.

```
78 {
79   gdt_ptr.limit = 5 * sizeof(gdt_entry) - 1;
80   gdt_ptr.base  = (u32int) gdt_entries;
81
82   u32int limit = 0xFFFFFFFF;
83   gdt_init_entry(0, 0, 0, 0, 0);           //required null segment
84   gdt_init_entry(1, 0, limit, 0x9A, 0xCF); //code segment
85   gdt_init_entry(2, 0, limit, 0x92, 0xCF); //data segment
86   gdt_init_entry(3, 0, limit, 0xFA, 0xCF); //user mode code segment
87   gdt_init_entry(4, 0, limit, 0xF2, 0xCF); //user mode data segment
88
89   write_gdt_ptr((u32int) &gdt_ptr, sizeof(gdt_ptr));
90 }
```

5.5.1.5 init_idt()

```
void init_idt ( )
```

Definition at line 45 of file tables.c.

```
46 {
47   idt_ptr.limit = 256*sizeof(idt_descriptor) - 1;
48   idt_ptr.base  = (u32int)idt_entries;
49   memset(idt_entries, 0, 256*sizeof(idt_descriptor));
50
51   write_idt_ptr((u32int)&idt_ptr);
52 }
```

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 mpx_core/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 8 of file heap.h.

5.6.1.2 KHEAP_MIN

```
#define KHEAP_MIN 0x10000
```

Definition at line 9 of file heap.h.

5.6.1.3 KHEAP_SIZE

```
#define KHEAP_SIZE 0x1000000
```

Definition at line 10 of file heap.h.

5.6.1.4 TABLE_SIZE

```
#define TABLE_SIZE 0x1000
```

Definition at line 7 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 26 of file heap.c.

```
27 {
28   u32int *addr;
29
30   // Allocate on the kernel heap if one has been created
31   if (kheap != 0){
32     addr = (u32int*)alloc(size, kheap, page_align);
33     if (phys_addr){
34       page_entry *page = get_page((u32int)addr, kdir, 0);
35       *phys_addr = (page->frameaddr*0x1000) + ((u32int)addr & 0xFFF);
36     }
37     return (u32int)addr;
38   }
39   // Else, allocate directly from physical memory
40   else {
41     if (page_align && (phys_alloc_addr & 0xFFFFF000)){
42       phys_alloc_addr &= 0xFFFFF000;
43       phys_alloc_addr += 0x1000;
44     }
45     addr = (u32int*)phys_alloc_addr;
46     if (phys_addr){
47       *phys_addr = phys_alloc_addr;
48     }
49     phys_alloc_addr += size;
50     return (u32int)addr;
51   }
52 }
```

5.6.2.2 alloc()

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

Definition at line 59 of file heap.c.

```
60 {
61   no_warn(size||align||h);
62   static u32int heap_addr = KHEAP_BASE;
63
64   u32int base = heap_addr;
65   heap_addr += size;
66
67   if (heap_addr > KHEAP_BASE + KHEAP_MIN)
68     serial_println("Heap is full!");
69
70   return base;
71 }
```

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 54 of file heap.c.

```
55 {
56   return _kmalloc(size,0,0);
57 }
```

5.6.2.6 make_heap()

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

Definition at line 73 of file heap.c.

```
74 {
75   no_warn(base||max||min);
76   return (heap*)kmalloc(sizeof(heap));
77 }
```

5.7 mpx_core/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 8 of file paging.h.

5.7.2 Function Documentation

5.7.2.1 clear_bit()

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

Definition at line 46 of file paging.c.

```
47 {
48   u32int frame  = addr/page_size;
49   u32int index  = frame/32;
50   u32int offset = frame%32;
51   frames[index] &= ~(1 << offset);
52 }
```

5.7.2.2 first_free()

```
u32int first_free ( )
```

5.7.2.3 get_bit()

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

Definition at line 58 of file paging.c.

```
59 {
60   u32int frame  = addr/page_size;
61   u32int index  = frame/32;
62   u32int offset = frame%32;
63   return (frames[index] & (1 « offset));
64 }
```

5.7.2.4 get_page()

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

Definition at line 87 of file paging.c.

```
88 {
89   u32int phys_addr;
90   u32int index = addr / page_size / 1024;
91   u32int offset = addr / page_size % 1024;
92
93   //return it if it exists
94   if (dir->tables[index])
95     return &dir->tables[index]->pages[offset];
96
97   //create it
98   else if (make_table){
99     dir->tables[index] = (page_table*)_kmalloc(sizeof(page_table), 1, &phys_addr);
100     dir->tables_phys[index] = phys_addr | 0x7; //enable present, writable
101     return &dir->tables[index]->pages[offset];
102   }
103   else return 0;
104 }
```

5.7.2.5 init_paging()

```
void init_paging ( )
```

Definition at line 113 of file paging.c.

```
114 {
115   //create frame bitmap
116   nframes = (u32int)(mem_size/page_size);
117   frames = (u32int*)kmalloc(nframes/32);
118   memset(frames, 0, nframes/32);
119
120   //create kernel directory
121   kdir = (page_dir*)_kmalloc(sizeof(page_dir), 1, 0); //page aligned
122   memset(kdir, 0, sizeof(page_dir));
123
124   //get pages for kernel heap
125   u32int i = 0x0;
126   for(i=KHEAP_BASE; i<(KHEAP_BASE+KHEAP_MIN); i+=1){
127     get_page(i,kdir,1);
128   }
129
130   //perform identity mapping of used memory
131   //note: placement_addr gets incremented in get_page,
132   //so we're mapping the first frames as well
133   i = 0x0;
134   while (i < (phys_alloc_addr+0x10000)){
```

```
135     new_frame(get_page(i,kdir,1));
136     i += page_size;
137   }
138
139   //allocate heap frames now that the placement addr has increased.
140   //placement addr increases here for heap
141   for(i=KHEAP_BASE; i<(KHEAP_BASE+KHEAP_MIN);i+=PAGE_SIZE){
142     new_frame(get_page(i,kdir,1));
143   }
144
145   //load the kernel page directory; enable paging
146   load_page_dir(kdir);
147
148   //setup the kernel heap
149   kheap = make_heap(KHEAP_BASE, KHEAP_SIZE, KHEAP_BASE+KHEAP_MIN);
150 }
```

5.7.2.6 load_page_dir()

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

Definition at line 160 of file paging.c.

```
161 {
162   cdir = new_dir;
163   asm volatile ("mov %0,%%cr3":: "b"(&cdir->tables_phys[0]));
164   u32int cr0;
165   asm volatile ("mov %%cr0,%0": "=b"(cr0));
166   cr0 |= 0x80000000;
167   asm volatile ("mov %0,%%cr0":: "b"(cr0));
168 }
```

5.7.2.7 new_frame()

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

Definition at line 175 of file paging.c.

```
176 {
177   u32int index;
178   if (page->frameaddr != 0) return;
179   if ( (u32int)(-1) == (index=find_free()) ) kpanic("Out of memory");
180
181   //mark a frame as in-use
182   set_bit(index*page_size);
183   page->present   = 1;
184   page->frameaddr = index;
185   page->writeable = 1;
186   page->usermode  = 0;
187 }
```

5.7.2.8 set_bit()

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

Definition at line 34 of file paging.c.

```
35 {
36   u32int frame  = addr/page_size;
37   u32int index  = frame/32;
38   u32int offset = frame%32;
39   frames[index] |= (1 « offset);
40 }
```

5.8 mpx_core/include/string.h File Reference

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

Functions

- int isspace (const char *c)

  Description: Determine if a character is whitespace.
- void * memset (void *s, int c, size_t n)

  Description: Set a region of memory.
- char * strcpy (char *s1, const char *s2)

  Description: Copy one string to another.
- char * strcat (char *s1, const char *s2)

  Description: Concatenate the contents of one string onto another.
- int strlen (const char *s)

  Description: Returns the length of a string.
- int strcmp (const char *s1, const char *s2)

  Description: String comparison.
- char * strtok (char *s1, const char *s2)

  Description: Split string into tokens.
- int atoi (const char *s)

  Description: Convert an ASCII string to an integer.

5.8.1 Function Documentation

5.8.1.1 atoi()

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

Description: Convert an ASCII string to an integer.

Parameters

s	String

Definition at line 50 of file string.c.

```
51 {
52   int res=0;
53    int charVal=0;
54    char sign = ' ';
55    char c = *s;
56 
57 
58    while(isspace(&c)){ ++s; c = *s;} // advance past whitespace
59 
60 
61    if (*s == '-' || *s == '+') sign = *(s++); // save the sign
62 
```

```
63
64    while(*s != '\0'){
65         charVal = *s - 48;
66     res = res * 10 + charVal;
67     s++;
68
69    }
70
71
72    if ( sign == '-') res=res * -1;
73
74   return res; // return integer
75 }
```

5.8.1.2 isspace()

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

Description: Determine if a character is whitespace.

Parameters

c	character to check

Definition at line 121 of file string.c.

```
122 {
123   if (*c == ' '  ||
124       *c == '\n' ||
125       *c == '\r' ||
126       *c == '\f' ||
127       *c == '\t' ||
128       *c == '\v'){
129     return 1;
130   }
131   return 0;
132 }
```

5.8.1.3 memset()

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

Description: Set a region of memory.

Parameters

s	destination
c	byte to write
n	count

Definition at line 139 of file string.c.

```
140 {
141   unsigned char *p = (unsigned char *) s;
```

```
142   while(n--){
143     *p++ = (unsigned char) c;
144   }
145   return s;
146 }
```

5.8.1.4 strcat()

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

Description: Concatenate the contents of one string onto another.

Parameters

s1	destination
s2	source

Definition at line 108 of file string.c.

```
109 {
110   char *rc = s1;
111   if (*s1) while(*++s1);
112   while( (*s1++ = *s2++) );
113   return rc;
114 }
```

5.8.1.5 strcmp()

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

Description: String comparison.

Parameters

s1	string 1
s2	string 2

Definition at line 81 of file string.c.

```
82 {
83
84   // Remarks:
85   // 1) If we made it to the end of both strings (i. e. our pointer points to a
86   //    '\0' character), the function will return 0
87   // 2) If we didn't make it to the end of both strings, the function will
88   //    return the difference of the characters at the first index of
89   //    indifference.
90   while ( (*s1) && (*s1==*s2) ){
91     ++s1;
92     ++s2;
93   }
94   return ( *(unsigned char *)s1 - *(unsigned char *)s2 );
95 }
```

5.8.1.6 strcpy()

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

Description: Copy one string to another.

Parameters

s1	destination
s2	source

Definition at line 38 of file string.c.

```
39 {
40   char *rc = s1;
41   while( (*s1++ = *s2++) );
42   return rc; // return pointer to destination string
43 }
```

5.8.1.7 strlen()

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

Description: Returns the length of a string.

Parameters

s	input string

Definition at line 26 of file string.c.

```
27 {
28   int r1 = 0;
29   if (*s) while(*s++) r1++;
30   return r1;//return length of string
31 }
```

5.8.1.8 strtok()

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

Description: Split string into tokens.

Parameters

s1	String
s2	delimiter

Definition at line 153 of file string.c.

```
154 {
155   static char *tok_tmp = NULL;
156   const char *p = s2;
157
158   //new string
159   if (s1!=NULL){
160     tok_tmp = s1;
161   }
162   //old string cont'd
163   else {
164     if (tok_tmp==NULL){
165       return NULL;
166     }
167     s1 = tok_tmp;
168   }
169
170   //skip leading s2 characters
171   while ( *p && *s1 ){
172     if (*s1==*p){
173       ++s1;
174       p = s2;
175       continue;
176     }
177     ++p;
178   }
179
180   //no more to parse
181   if (!*s1){
182     return (tok_tmp = NULL);
183   }
184
185   //skip non-s2 characters
186   tok_tmp = s1;
187   while (*tok_tmp){
188     p = s2;
189     while (*p){
190       if (*tok_tmp==*p++){
191     *tok_tmp++ = '\0';
192     return s1;
193       }
194     }
195     ++tok_tmp;
196   }
197
198   //end of string
199   tok_tmp = NULL;
200   return s1;
201 }
```

5.9 mpx_core/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 13 of file system.h.

5.9.1.2 cli

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

Definition at line 17 of file system.h.

5.9.1.3 GDT_CS_ID

```
#define GDT_CS_ID 0x01
```

Definition at line 22 of file system.h.

5.9.1.4 GDT_DS_ID

```
#define GDT_DS_ID 0x02
```

Definition at line 23 of file system.h.

5.9.1.5 hlt

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

Definition at line 19 of file system.h.

5.9.1.6 iret

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

Definition at line 20 of file system.h.

5.9.1.7 no_warn

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

Definition at line 9 of file system.h.

5.9.1.8 nop

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

Definition at line 18 of file system.h.

5.9.1.9 NULL

```
#define NULL 0
```

Definition at line 6 of file system.h.

5.9.1.10 sti

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

Definition at line 16 of file system.h.

5.9.1.11 volatile

```
#define volatile __volatile__
```

Definition at line 14 of file system.h.

5.9.2 Typedef Documentation

5.9.2.1 size_t

```
typedef unsigned int size_t
```

Definition at line 26 of file system.h.

5.9.2.2 u16int

```
typedef unsigned short u16int
```

Definition at line 28 of file system.h.

5.9.2.3 u32int

```
typedef unsigned long u32int
```

Definition at line 29 of file system.h.

5.9.2.4 u8int

```
typedef unsigned char u8int
```

Definition at line 27 of file system.h.

5.9.3 Function Documentation

5.9.3.1 klogv()

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

Definition at line 13 of file system.c.

```
14 {
15   char logmsg[64] = {'\0'}, prefix[] = "klogv: ";
16   strcat(logmsg, prefix);
17   strcat(logmsg, msg);
18   serial_println(logmsg);
19 }
```

5.9.3.2 kpanic()

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

Definition at line 26 of file system.c.

```
27 {
28   cli(); //disable interrupts
29   char logmsg[64] = {'\0'}, prefix[] = "Panic: ";
30   strcat(logmsg, prefix);
31   strcat(logmsg, msg);
32   klogv(logmsg);
33   hlt(); //halt
34 }
```

5.10 mpx_core/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 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 22 of file interrupts.c.

5.10.1.2 ICW4

```
#define ICW4 0x01
```

Definition at line 23 of file interrupts.c.

5.10.1.3 io_wait

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

Definition at line 30 of file interrupts.c.

5.10.1.4 PIC1

```
#define PIC1 0x20
```

Definition at line 18 of file interrupts.c.

5.10.1.5 PIC2

```
#define PIC2 0xA0
```

Definition at line 19 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 150 of file interrupts.c.

```
151 {
152   kpanic("Bounds error");
153 }
```

5.10.2.9 do_breakpoint()

```
void do_breakpoint ( )
```

Definition at line 142 of file interrupts.c.

```
143 {
144   kpanic("Breakpoint");
145 }
```

5.10.2.10 do_coprocessor()

```
void do_coprocessor ( )
```

Definition at line 194 of file interrupts.c.

```
195 {
196   kpanic("Coprocessor error");
197 }
```

5.10.2.11 do_coprocessor_segment()

```
void do_coprocessor_segment ( )
```

Definition at line 166 of file interrupts.c.

```
167 {
168   kpanic("Coprocessor segment error");
169 }
```

5.10.2.12 do_debug()

```
void do_debug ( )
```

Definition at line 134 of file interrupts.c.

```
135 {
136   kpanic("Debug");
137 }
```

5.10.2.13 do_device_not_available()

```
void do_device_not_available ( )
```

Definition at line 158 of file interrupts.c.

```
159 {
160   kpanic("Device not available");
161 }
```

5.10.2.14 do_divide_error()

```
void do_divide_error ( )
```

Definition at line 130 of file interrupts.c.

```
131 {
132   kpanic("Division-by-zero");
133 }
```

5.10.2.15 do_double_fault()

```
void do_double_fault ( )
```

Definition at line 162 of file interrupts.c.

```
163 {
164   kpanic("Double fault");
165 }
```

5.10.2.16 do_general_protection()

```
void do_general_protection ( )
```

Definition at line 182 of file interrupts.c.

```
183 {
184   kpanic("General protection fault");
185 }
```

5.10.2.17 do_invalid_op()

```
void do_invalid_op ( )
```

Definition at line 154 of file interrupts.c.

```
155 {
156   kpanic("Invalid operation");
157 }
```

5.10.2.18 do_invalid_tss()

```
void do_invalid_tss ( )
```

Definition at line 170 of file interrupts.c.

```
171 {
172   kpanic("Invalid TSS");
173 }
```

5.10.2.19 do_isr()

```
void do_isr ( )
```

Definition at line 55 of file interrupts.c.

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

5.10.2.20 do_nmi()

```
void do_nmi ( )
```

Definition at line 138 of file interrupts.c.

```
139 {
140   kpanic("NMI");
141 }
```

5.10.2.21 do_overflow()

```
void do_overflow ( )
```

Definition at line 146 of file interrupts.c.

```
147 {
148   kpanic("Overflow error");
149 }
```

5.10.2.22 do_page_fault()

```
void do_page_fault ( )
```

Definition at line 186 of file interrupts.c.

```
187 {
188   kpanic("Page Fault");
189 }
```

5.10.2.23 do_reserved()

```
void do_reserved ( )
```

Definition at line 190 of file interrupts.c.

```
191 {
192   serial_println("die: reserved");
193 }
```

5.10.2.24 do_segment_not_present()

```
void do_segment_not_present ( )
```

Definition at line 174 of file interrupts.c.

```
175 {
176   kpanic("Segment not present");
177 }
```

5.10.2.25 do_stack_segment()

```
void do_stack_segment ( )
```

Definition at line 178 of file interrupts.c.

```
179 {
180   kpanic("Stack segment error");
181 }
```

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

5.10.2.29 init_pic()

```
void init_pic (
            void  )
```

Definition at line 107 of file interrupts.c.

```
108 {
109   outb(PIC1,ICW1);   //send initialization code words 1 to PIC1
110   io_wait();
111   outb(PIC2,ICW1);   //send icw1 to PIC2
112   io_wait();
113   outb(PIC1+1,0x20); //icw2: remap irq0 to 32
114   io_wait();
115   outb(PIC2+1,0x28); //icw2: remap irq8 to 40
116   io_wait();
117   outb(PIC1+1,4);    //icw3
118   io_wait();
119   outb(PIC2+1,2);    //icw3
120   io_wait();
121   outb(PIC1+1,ICW4); //icw4: 80x86, automatic handling
122   io_wait();
123   outb(PIC2+1,ICW4); //icw4: 80x86, automatic handling
124   io_wait();
125   outb(PIC1+1,0xFF); //disable irqs for PIC1
126   io_wait();
127   outb(PIC2+1,0xFF); //disable irqs for PIC2
128 }
```

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 ( )
```

5.10.3 Variable Documentation

5.10.3.1 idt_entries

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

Definition at line 19 of file tables.c.

5.11 mpx_core/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/comHand.h"
```

Functions

- void kmain (void)

5.11.1 Function Documentation

5.11.1.1 kmain()

```
void kmain (
            void  )
```

Definition at line 28 of file kmain.c.

```
29 {
30    extern uint32_t magic;
31    // Uncomment if you want to access the multiboot header
32    // extern void *mbd;
33    // char *boot_loader_name = (char*)((long*)mbd)[16];
34
35
36    // 0) Initialize Serial I/O
37    // functions to initialize serial I/O can be found in serial.c
38    // there are 3 functions to call
39    init_serial(COM1);
40    set_serial_in(COM1);
41    set_serial_out(COM1);
42
43
44    klogv("Starting MPX boot sequence...");
45    klogv("Initialized serial I/O on COM1 device...");
46
47    // 1) Initialize the support software by identifying the current
48    //     MPX Module.  This will change with each module.
49    // you will need to call mpx_init from the mpx_supt.c
50      mpx_init(MODULE_R1);
51
52    // 2) Check that the boot was successful and correct when using grub
53    // Comment this when booting the kernel directly using QEMU, etc.
54    if ( magic != 0x2BADB002 ){
55      //kpanic("Boot was not error free. Halting.");
56    }
57
58    // 3) Descriptor Tables -- tables.c
59    //  you will need to initialize the global
60    // this keeps track of allocated segments and pages
61    klogv("Initializing descriptor tables...");
62      init_gdt();
63
```

```
64     // 4)  Interrupt vector table --  tables.c
65     // this creates and initializes a default interrupt vector table
66     // this function is in tables.c
67     klogv("Interrupt vector table initialized!");
68     init_idt();
69     init_irq();
70     init_pic();
71
72    // 5) Virtual Memory -- paging.c  -- init_paging
73    //  this function creates the kernel's heap
74    //  from which memory will be allocated when the program calls
75    // sys_alloc_mem UNTIL the memory management module  is completed
76    // this allocates memory using discrete "pages" of physical memory
77    // NOTE:  You will only have about 70000 bytes of dynamic memory
78    klogv("Initializing virtual memory...");
79      init_paging();
80
81    // 6) Call YOUR command handler -  interface method
82    klogv("Transferring control to commhand...");
83    // INSERT OS LAUNCH TEXT HERE
84      comHand();
85
86    // 7) System Shutdown on return from your command handler
87    klogv("Starting system shutdown procedure...");
88
89    /* Shutdown Procedure */
90    klogv("Shutdown complete. You may now turn off the machine. (QEMU: C-a x)");
91    hlt();
92 }
```

5.12 mpx_core/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 *cmdBuffer, 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 18 of file serial.c.

5.12.2 Function Documentation

5.12.2.1 init_serial()

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

Definition at line 28 of file serial.c.

```
28                              {
29         outb(device + 1, 0x00); //disable interrupts
30         outb(device + 3, 0x80); //set line control register
31         outb(device + 0, 115200 / 9600); //set bsd least sig bit
32         outb(device + 1, 0x00); //brd most significant bit
33         outb(device + 3, 0x03); //lock divisor; 8bits, no parity, one stop
34         outb(device + 2, 0xC7); //enable fifo, clear, 14byte threshold
35         outb(device + 4, 0x0B); //enable interrupts, rts/dsr set
36         (void) inb(device); //read bit to reset port
37         return NO_ERROR;
38 }
```

5.12.2.2 polling()

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

Definition at line 95 of file serial.c.

```
95                                              {
96         int pointerLoc = 0;
97         int numCharacters = 0;
98         int flag = 1;
99         char letter = NULL;
100         while (flag) { // Run continuously
101
102                 if (inb(COM1 + 5) & 1) { // Is a character available?
103                         letter = inb(COM1); //Get the character
104
105                         //Special Cases
106
107                         //ENTER
108                         if (letter == '\n' || letter == '\r') {
109                                 cmdBuffer[pointerLoc] = '\0';
110                                 flag = 0;
111                                 serial_print("\n");
112                         }
113
114                         else if (letter == '\033') {
115                                 letter = inb(COM1);
116                                 if (letter == '[') {
117                                         letter = inb(COM1);
118
119                                         //Right Arrow Case
120                                         if (letter == 'C') {
121                                                 if (pointerLoc < numCharacters) {
122                                                         pointerLoc++;
123                                                         serial_print("\033[C");
```

```
124                                             }
125                                     }
126
127                                     //Left Arrow Case
128                                     else if (letter == 'D') {
129                                             if (pointerLoc > 0) {
130                                                     pointerLoc--;
131                                                     serial_print("\033[D");
132                                             }
133
134                                     }
135                                     else if (letter == 'A') {
136                                     //up
137                                     }
138                                     else if (letter == 'B') {
139                                     //down
140                                     }
141
142                                     //DELETE
143                                     else if (letter == '3') {
144                                             letter = inb(COM1);
145                                             if (letter == '~') {
146                                                     if (pointerLoc < numCharacters) {
147                                                             int bufIndex;
148                                                             for (bufIndex = pointerLoc; bufIndex <
    *count; bufIndex++) {
149                                                                     cmdBuffer[bufIndex] =
    cmdBuffer[bufIndex + 1];
150                                                             }
151                                                             serial_print("\033[1P");
152                                                             numCharacters--;
153                                                             inb(COM1);
154                                                     }
155                                             }
156                                     }
157                             }
158                     }
159
160                     //BACKSPACE
161                     else if (letter == 127) {
162                             if(pointerLoc > 0){
163                                     if(pointerLoc > numCharacters){
164                                             cmdBuffer[pointerLoc - 1] = NULL;
165                                     }
166                                     else{
167                                             int bufIndex;
168                                             for (bufIndex = pointerLoc; bufIndex <= numCharacters;
    bufIndex++) {
169                                                     cmdBuffer[bufIndex-1] = cmdBuffer[bufIndex];
    //replaces the last typed character with null.
170                                             }
171                                     }
172                                     numCharacters--;
173                                     pointerLoc--;
174                                     serial_print("\033[D\033[P");
175                                     inb(COM1);
176                             }
177                     }
178
179
180                     //passes any other characters 0-9,a-z, upper and lower case to the command
    handler to be dealt with.
181                     else {
182                             if (numCharacters < * count) {
183                                     if(pointerLoc < numCharacters) {
184                                             int bufIndex;
185                                             for(bufIndex = numCharacters + 1; bufIndex > pointerLoc;
    bufIndex--)
186                                             {
187                                                     cmdBuffer[bufIndex] = cmdBuffer[bufIndex - 1];
188                                             }
189                                             cmdBuffer[pointerLoc] = letter;
190                                             numCharacters++; //increments the total number of
    characters passed in so far.
191                                             pointerLoc++; //increments the pointer location per
    input.
192
193                                             //int i = 0;
194                                             // for(i = 0; i <= numCharacters + 1)
195                                             serial_print("\033[s\033[K");
196                                             serial_print(&cmdBuffer[pointerLoc-1]);
197                                             serial_print("\033[u\033[C");
198                                     }
199                                     else {
200                                           cmdBuffer[pointerLoc] = letter;
201                                           serial_print(&cmdBuffer[pointerLoc]);
202                                           pointerLoc++; //increments the pointer location per input.
```

```
203                                                 numCharacters++; //increments the total number of
      characters passed in so far.
204                                         }
205                                     }
206                                 }
207                             }
208         }
209         return 0;
210 }
```

5.12.2.3 serial_print()

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

Definition at line 59 of file serial.c.

```
59                                  {
60         int i;
61         for (i = 0;*(i + msg) != '\0'; i++) {
62                 outb(serial_port_out, *(i + msg));
63         }
64         if ( * msg == '\r') 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 45 of file serial.c.

```
45                                      {
46         int i;
47         for (i = 0;*(i + msg) != '\0'; i++) {
48                 outb(serial_port_out, *(i + msg));
49         }
50         outb(serial_port_out, '\r');
51         outb(serial_port_out, '\n');
52         return NO_ERROR;
53 }
```

5.12.2.5 set_serial_in()

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

Definition at line 85 of file serial.c.

```
85                                  {
86         serial_port_in = device;
87         return NO_ERROR;
88 }
```

5.12.2.6 set_serial_out()

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

Definition at line 74 of file serial.c.

```
74                                  {
75          serial_port_out = device;
76          return NO_ERROR;
77 }
```

5.12.3 Variable Documentation

5.12.3.1 serial_port_in

```
int serial_port_in = 0
```

Definition at line 22 of file serial.c.

5.12.3.2 serial_port_out

```
int serial_port_out = 0
```

Definition at line 21 of file serial.c.

5.13 mpx_core/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 13 of file system.c.

```
14 {
15   char logmsg[64] = {'\0'}, prefix[] = "klogv: ";
16   strcat(logmsg, prefix);
17   strcat(logmsg, msg);
18   serial_println(logmsg);
19 }
```

5.13.1.2 kpanic()

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

Definition at line 26 of file system.c.

```
27 {
28   cli(); //disable interrupts
29   char logmsg[64] = {'\0'}, prefix[] = "Panic: ";
30   strcat(logmsg, prefix);
31   strcat(logmsg, msg);
32   klogv(logmsg);
33   hlt(); //halt
34 }
```

5.14 mpx_core/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 59 of file tables.c.

```
61 {
62   gdt_entry *new_entry = &gdt_entries[idx];
63   new_entry->base_low  = (base & 0xFFFF);
64   new_entry->base_mid  = (base » 16) & 0xFF;
65   new_entry->base_high = (base » 24) & 0xFF;
66   new_entry->limit_low = (limit & 0xFFFF);
67   new_entry->flags  = (limit » 16) & 0xFF;
68   new_entry->flags |= flags & 0xF0;
69   new_entry->access = access;
70 }
```

5.14.1.2 idt_set_gate()

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

Definition at line 29 of file tables.c.

```
31 {
32   idt_entry *new_entry = &idt_entries[idx];
33   new_entry->base_low  = (base &  0xFFFF);
34   new_entry->base_high = (base » 16) & 0xFFFF;
35   new_entry->sselect   = sel;
36   new_entry->zero = 0;
37   new_entry->flags = flags;
38 }
```

5.14.1.3 init_gdt()

```
void init_gdt ( )
```

Definition at line 77 of file tables.c.

```
78 {
79   gdt_ptr.limit = 5 * sizeof(gdt_entry) - 1;
80   gdt_ptr.base  = (u32int) gdt_entries;
81
82   u32int limit = 0xFFFFFFFF;
83   gdt_init_entry(0, 0, 0, 0, 0);           //required null segment
84   gdt_init_entry(1, 0, limit, 0x9A, 0xCF); //code segment
85   gdt_init_entry(2, 0, limit, 0x92, 0xCF); //data segment
86   gdt_init_entry(3, 0, limit, 0xFA, 0xCF); //user mode code segment
87   gdt_init_entry(4, 0, limit, 0xF2, 0xCF); //user mode data segment
88
89   write_gdt_ptr((u32int) &gdt_ptr, sizeof(gdt_ptr));
90 }
```

5.14.1.4 init_idt()

```
void init_idt ( )
```

Definition at line 45 of file tables.c.

```
46 {
47   idt_ptr.limit = 256*sizeof(idt_descriptor) - 1;
48   idt_ptr.base  = (u32int)idt_entries;
49   memset(idt_entries, 0, 256*sizeof(idt_descriptor));
50
51   write_idt_ptr((u32int)&idt_ptr);
52 }
```

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 15 of file tables.c.

5.14.2.2 gdt_ptr

```
gdt_descriptor gdt_ptr
```

Definition at line 14 of file tables.c.

5.14.2.3 idt_entries

```
idt_entry idt_entries[256]
```

Definition at line 19 of file tables.c.

5.14.2.4 idt_ptr

```
idt_descriptor idt_ptr
```

Definition at line 18 of file tables.c.

5.15 mpx_core/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 26 of file heap.c.

```
27 {
28   u32int *addr;
29
30   // Allocate on the kernel heap if one has been created
31   if (kheap != 0){
32     addr = (u32int*)alloc(size, kheap, page_align);
33     if (phys_addr){
34       page_entry *page = get_page((u32int)addr, kdir, 0);
35       *phys_addr = (page->frameaddr*0x1000) + ((u32int)addr & 0xFFF);
36     }
37     return (u32int)addr;
38   }
39   // Else, allocate directly from physical memory
40   else {
41     if (page_align && (phys_alloc_addr & 0xFFFFF000)){
42       phys_alloc_addr &= 0xFFFFF000;
43       phys_alloc_addr += 0x1000;
44     }
45     addr = (u32int*)phys_alloc_addr;
46     if (phys_addr){
47       *phys_addr = phys_alloc_addr;
48     }
49     phys_alloc_addr += size;
50     return (u32int)addr;
51   }
52 }
```

5.15.1.2 alloc()

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

Definition at line 59 of file heap.c.

```
60 {
61   no_warn(size||align||h);
62   static u32int heap_addr = KHEAP_BASE;
63
64   u32int base = heap_addr;
65   heap_addr += size;
66
67   if (heap_addr > KHEAP_BASE + KHEAP_MIN)
68     serial_println("Heap is full!");
69
70   return base;
71 }
```

5.15.1.3 kmalloc()

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

Definition at line 54 of file heap.c.

```
55 {
56   return _kmalloc(size,0,0);
57 }
```

5.15.1.4 make_heap()

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

Definition at line 73 of file heap.c.

```
74 {
75   no_warn(base||max||min);
76   return (heap*)kmalloc(sizeof(heap));
77 }
```

5.15.2 Variable Documentation

5.15.2.1 __end

```
void __end
```

Definition at line 20 of file heap.c.

5.15.2.2 _end

```
void _end
```

Definition at line 20 of file heap.c.

5.15.2.3 curr_heap

```
heap* curr_heap = 0
```

Definition at line 17 of file heap.c.

5.15.2.4 end

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

5.15.2.5 kdir

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

Definition at line 23 of file paging.c.

5.15.2.6 kheap

```
heap* kheap = 0
```

Definition at line 16 of file heap.c.

5.15.2.7 phys_alloc_addr

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

Definition at line 24 of file heap.c.

5.16 mpx_core/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 46 of file paging.c.

```
47 {
48   u32int frame  = addr/page_size;
49   u32int index  = frame/32;
50   u32int offset = frame%32;
51   frames[index] &= ~(1 « offset);
52 }
```

5.16.1.2 find_free()

```
u32int find_free ( )
```

Definition at line 70 of file paging.c.

```
71 {
72   u32int i,j;
73   for (i=0; i<nframes/32; i++)
74     if (frames[i] != 0xFFFFFFFF) //if frame not full
75       for (j=0; j<32; j++) //find first free bit
76     if (!(frames[i] & (1 « j)))
77       return i*32+j;
78
79   return -1; //no free frames
80 }
```

5.16.1.3 get_bit()

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

Definition at line 58 of file paging.c.

```
59 {
60   u32int frame  = addr/page_size;
61   u32int index  = frame/32;
62   u32int offset = frame%32;
63   return (frames[index] & (1 « offset));
64 }
```

5.16.1.4 get_page()

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

Definition at line 87 of file paging.c.

```
88 {
89   u32int phys_addr;
90   u32int index = addr / page_size / 1024;
91   u32int offset = addr / page_size % 1024;
92
93   //return it if it exists
94   if (dir->tables[index])
95     return &dir->tables[index]->pages[offset];
96
97   //create it
98   else if (make_table){
99     dir->tables[index] = (page_table*)_kmalloc(sizeof(page_table), 1, &phys_addr);
100     dir->tables_phys[index] = phys_addr | 0x7; //enable present, writable
101     return &dir->tables[index]->pages[offset];
102   }
103   else return 0;
104 }
```

5.16.1.5 init_paging()

```
void init_paging ( )
```

Definition at line 113 of file paging.c.

```
114 {
115   //create frame bitmap
116   nframes = (u32int)(mem_size/page_size);
117   frames = (u32int*)kmalloc(nframes/32);
118   memset(frames, 0, nframes/32);
119
120   //create kernel directory
121   kdir = (page_dir*)_kmalloc(sizeof(page_dir), 1, 0); //page aligned
122   memset(kdir, 0, sizeof(page_dir));
123
124   //get pages for kernel heap
125   u32int i = 0x0;
126   for(i=KHEAP_BASE; i<(KHEAP_BASE+KHEAP_MIN); i+=1){
127     get_page(i,kdir,1);
128   }
129
130   //perform identity mapping of used memory
131   //note: placement_addr gets incremented in get_page,
132   //so we're mapping the first frames as well
133   i = 0x0;
134   while (i < (phys_alloc_addr+0x10000)){
135     new_frame(get_page(i,kdir,1));
136     i += page_size;
137   }
138
139   //allocate heap frames now that the placement addr has increased.
140   //placement addr increases here for heap
141   for(i=KHEAP_BASE; i<(KHEAP_BASE+KHEAP_MIN);i+=PAGE_SIZE){
142     new_frame(get_page(i,kdir,1));
143   }
144
145   //load the kernel page directory; enable paging
146   load_page_dir(kdir);
147
148   //setup the kernel heap
149   kheap = make_heap(KHEAP_BASE, KHEAP_SIZE, KHEAP_BASE+KHEAP_MIN);
150 }
```

5.16.1.6 load_page_dir()

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

Definition at line 160 of file paging.c.

```
161 {
162   cdir = new_dir;
163   asm volatile ("mov %0,%%cr3":: "b"(&cdir->tables_phys[0]));
164   u32int cr0;
165   asm volatile ("mov %%cr0,%0": "=b"(cr0));
166   cr0 |= 0x80000000;
167   asm volatile ("mov %0,%%cr0":: "b"(cr0));
168 }
```

5.16.1.7 new_frame()

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

Definition at line 175 of file paging.c.

```
176 {
177   u32int index;
178   if (page->frameaddr != 0) return;
179   if ( (u32int)(-1) == (index=find_free()) ) kpanic("Out of memory");
180
181   //mark a frame as in-use
182   set_bit(index*page_size);
183   page->present   = 1;
184   page->frameaddr = index;
185   page->writeable = 1;
186   page->usermode  = 0;
187 }
```

5.16.1.8 set_bit()

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

Definition at line 34 of file paging.c.

```
35 {
36   u32int frame  = addr/page_size;
37   u32int index  = frame/32;
38   u32int offset = frame%32;
39   frames[index] |= (1 « offset);
40 }
```

5.16.2 Variable Documentation

5.16.2.1 cdir

```
page_dir* cdir = 0
```

Definition at line 24 of file paging.c.

5.16.2.2 frames

```
u32int* frames
```

Definition at line 21 of file paging.c.

5.16.2.3 kdir

```
page_dir* kdir = 0
```

Definition at line 23 of file paging.c.

5.16.2.4 kheap

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

Definition at line 16 of file heap.c.

5.16.2.5 mem_size

```
u32int mem_size = 0x4000000
```

Definition at line 17 of file paging.c.

5.16.2.6 nframes

```
u32int nframes
```

Definition at line 20 of file paging.c.

5.16.2.7 page_size

```
u32int page_size = 0x1000
```

Definition at line 18 of file paging.c.

5.16.2.8 phys_alloc_addr

```
u32int phys_alloc_addr  [extern]
```

Definition at line 24 of file heap.c.

5.17 mpx_core/lib/string.c File Reference

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

Functions

- int strlen (const char *s)

  Description: Returns the length of a string.
- char * strcpy (char *s1, const char *s2)

  Description: Copy one string to another.
- int atoi (const char *s)

  Description: Convert an ASCII string to an integer.
- int strcmp (const char *s1, const char *s2)

  Description: String comparison.
- char * strcat (char *s1, const char *s2)

  Description: Concatenate the contents of one string onto another.
- int isspace (const char *c)

  Description: Determine if a character is whitespace.
- void * memset (void *s, int c, size_t n)

  Description: Set a region of memory.
- char * strtok (char *s1, const char *s2)

  Description: Split string into tokens.

5.17.1 Function Documentation

5.17.1.1 atoi()

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

Description: Convert an ASCII string to an integer.

Parameters

s	String

Definition at line 50 of file string.c.

```
51 {
52   int res=0;
53    int charVal=0;
54    char sign = ' ';
55    char c = *s;
56
57
58    while(isspace(&c)){ ++s; c = *s;} // advance past whitespace
59
60
61    if (*s == '-' || *s == '+') sign = *(s++); // save the sign
62
63
64    while(*s != '\0'){
65         charVal = *s - 48;
66     res = res * 10 + charVal;
67     s++;
68
69    }
70
71
72    if ( sign == '-') res=res * -1;
73
74   return res; // return integer
75 }
```

5.17.1.2 isspace()

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

Description: Determine if a character is whitespace.

Parameters

c	character to check

Definition at line 121 of file string.c.

```
122 {
123   if (*c == ' '  ||
124       *c == '\n' ||
125       *c == '\r' ||
126       *c == '\f' ||
127       *c == '\t' ||
128       *c == '\v'){
129     return 1;
130   }
131   return 0;
132 }
```

5.17.1.3 memset()

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

Description: Set a region of memory.

Parameters

s	destination
c	byte to write
n	count

Definition at line 139 of file string.c.

```
140 {
141   unsigned char *p = (unsigned char *) s;
142   while(n--){
143     *p++ = (unsigned char) c;
144   }
145   return s;
146 }
```

5.17.1.4 strcat()

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

Description: Concatenate the contents of one string onto another.

Parameters

s1	destination
s2	source

Definition at line 108 of file string.c.

```
109 {
110   char *rc = s1;
111   if (*s1) while(*++s1);
112   while( (*s1++ = *s2++) );
113   return rc;
114 }
```

5.17.1.5 strcmp()

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

Description: String comparison.

Parameters

s1	string 1
s2	string 2

Definition at line 81 of file string.c.

```
82 {
83
84   // Remarks:
85   // 1) If we made it to the end of both strings (i. e. our pointer points to a
86   //    '\0' character), the function will return 0
87   // 2) If we didn't make it to the end of both strings, the function will
88   //    return the difference of the characters at the first index of
89   //    indifference.
90   while ( (*s1) && (*s1==*s2) ){
91     ++s1;
92     ++s2;
93   }
94   return ( *(unsigned char *)s1 - *(unsigned char *)s2 );
95 }
```

5.17.1.6 strcpy()

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

Description: Copy one string to another.

Parameters

s1	destination
s2	source

Definition at line 38 of file string.c.

```
39 {
40   char *rc = s1;
41   while( (*s1++ = *s2++) );
42   return rc; // return pointer to destination string
43 }
```

5.17.1.7 strlen()

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

Description: Returns the length of a string.

Parameters

s	input string

Definition at line 26 of file string.c.

```
27 {
28   int r1 = 0;
29   if (*s) while(*s++) r1++;
30   return r1;//return length of string
31 }
```

5.17.1.8 strtok()

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

Description: Split string into tokens.

Parameters

s1	String
s2	delimiter

Definition at line 153 of file string.c.

```
154 {
155   static char *tok_tmp = NULL;
156   const char *p = s2;
157
158   //new string
159   if (s1!=NULL){
160     tok_tmp = s1;
161   }
162   //old string cont'd
163   else {
164     if (tok_tmp==NULL){
165       return NULL;
166     }
167     s1 = tok_tmp;
168   }
169
170   //skip leading s2 characters
171   while ( *p && *s1 ){
172     if (*s1==*p){
173       ++s1;
174       p = s2;
175       continue;
176     }
177     ++p;
178   }
179
180   //no more to parse
181   if (!*s1){
182     return (tok_tmp = NULL);
183   }
184
185   //skip non-s2 characters
186   tok_tmp = s1;
187   while (*tok_tmp){
188     p = s2;
189     while (*p){
190       if (*tok_tmp==*p++){
191     *tok_tmp++ = '\0';
192     return s1;
193       }
194     }
195     ++tok_tmp;
196   }
197
198   //end of string
199   tok_tmp = NULL;
200   return s1;
201 }
```

5.18 mpx_core/modules/mpx_supt.c File Reference

```
#include "mpx_supt.h"
#include <mem/heap.h>
#include <string.h>
#include <core/serial.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 ()

Variables

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

5.18.1 Function Documentation

5.18.1.1 idle()

```
void idle ( )
```

Definition at line 175 of file mpx_supt.c.

```
176 {
177   char msg[30];
178   int count=0;
179 
180     memset( msg, '\0', sizeof(msg));
181     strcpy(msg, "IDLE PROCESS EXECUTING.\n");
182     count = strlen(msg);
183 
184   while(1){
185     sys_req( WRITE, DEFAULT_DEVICE, msg, &count);
186     sys_req(IDLE, DEFAULT_DEVICE, NULL, NULL);
187   }
188 }
```

5.18.1.2 mpx_init()

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

Definition at line 108 of file mpx_supt.c.

```
109 {
110 
111   current_module = cur_mod;
112   if (cur_mod == MEM_MODULE)
113         mem_module_active = TRUE;
114 
115   if (cur_mod == IO_MODULE)
116         io_module_active = TRUE;
117 }
```

5.18.1.3 sys_alloc_mem()

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

Definition at line 146 of file mpx_supt.c.

```
147 {
148   if (!mem_module_active)
149     return (void *) kmalloc(size);
150   else
151     return (void *) (*student_malloc)(size);
152 }
```

5.18.1.4 sys_free_mem()

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

Definition at line 160 of file mpx_supt.c.

```
161 {
162   if (mem_module_active)
163     return (*student_free)(ptr);
164   // otherwise we don't free anything
165   return -1;
166 }
```

5.18.1.5 sys_req()

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

Definition at line 51 of file mpx_supt.c.

```
56 {
57     int return_code =0;
58
59   if (op_code == IDLE || op_code == EXIT){
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     // validate buffer pointer and count pointer
68     if (buffer_ptr == NULL)
69       return_code = INVALID_BUFFER;
70     else if (count_ptr == NULL || *count_ptr <= 0)
71       return_code = INVALID_COUNT;
72
73     // if parameters are valid store in the params structure
74     if ( return_code == 0){
75       params.op_code = op_code;
76       params.device_id = device_id;
77       params.buffer_ptr = buffer_ptr;
78       params.count_ptr = count_ptr;
79
80       if (!io_module_active){
81         // if default device
82         if (op_code == READ)
83           return_code = *(polling(buffer_ptr, count_ptr));
84
```

```
85          else //must be WRITE
86            return_code = serial_print(buffer_ptr);
87
88        } else {// I/O module is implemented
89          asm volatile ("int $60");
90        } // NOT IO_MODULE
91      }
92    } else return_code = INVALID_OPERATION;
93
94    return return_code;
95 }// end of sys_req
```

5.18.1.6 sys_set_free()

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

Definition at line 136 of file mpx_supt.c.

```
137 {
138   student_free = func;
139 }
```

5.18.1.7 sys_set_malloc()

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

Definition at line 126 of file mpx_supt.c.

```
127 {
128   student_malloc = func;
129 }
```

5.18.2 Variable Documentation

5.18.2.1 current_module

```
int current_module = -1
```

Definition at line 20 of file mpx_supt.c.

5.18.2.2 params

```
param params
```

Definition at line 17 of file mpx_supt.c.

5.18.2.3 student_free

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

Definition at line 30 of file mpx_supt.c.

5.18.2.4 student_malloc

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

Definition at line 26 of file mpx_supt.c.

5.19 mpx_core/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 31 of file mpx_supt.h.

5.19.1.2 DEFAULT_DEVICE

```
#define DEFAULT_DEVICE 111
```

Definition at line 30 of file mpx_supt.h.

5.19.1.3 EXIT

```
#define EXIT 0
```

Definition at line 8 of file mpx_supt.h.

5.19.1.4 FALSE

```
#define FALSE 0
```

Definition at line 15 of file mpx_supt.h.

5.19.1.5 IDLE

```
#define IDLE 1
```

Definition at line 9 of file mpx_supt.h.

5.19.1.6 INVALID_BUFFER

```
#define INVALID_BUFFER 1000
```

Definition at line 27 of file mpx_supt.h.

5.19.1.7 INVALID_COUNT

```
#define INVALID_COUNT 2000
```

Definition at line 28 of file mpx_supt.h.

5.19.1.8 INVALID_OPERATION

```
#define INVALID_OPERATION 4
```

Definition at line 12 of file mpx_supt.h.

5.19.1.9 IO_MODULE

```
#define IO_MODULE 10
```

Definition at line 23 of file mpx_supt.h.

5.19.1.10 MEM_MODULE

```
#define MEM_MODULE 11
```

Definition at line 24 of file mpx_supt.h.

5.19.1.11 MODULE_F

```
#define MODULE_F 9
```

Definition at line 22 of file mpx_supt.h.

5.19.1.12 MODULE_R1

```
#define MODULE_R1 0
```

Definition at line 17 of file mpx_supt.h.

5.19.1.13 MODULE_R2

```
#define MODULE_R2 1
```

Definition at line 18 of file mpx_supt.h.

5.19.1.14 MODULE_R3

```
#define MODULE_R3 2
```

Definition at line 19 of file mpx_supt.h.

5.19.1.15 MODULE_R4

```
#define MODULE_R4 4
```

Definition at line 20 of file mpx_supt.h.

5.19.1.16 MODULE_R5

```
#define MODULE_R5 8
```

Definition at line 21 of file mpx_supt.h.

5.19.1.17 READ

```
#define READ 2
```

Definition at line 10 of file mpx_supt.h.

5.19.1.18 TRUE

```
#define TRUE 1
```

Definition at line 14 of file mpx_supt.h.

5.19.1.19 WRITE

```
#define WRITE 3
```

Definition at line 11 of file mpx_supt.h.

5.19.2 Function Documentation

5.19.2.1 idle()

```
void idle ( )
```

Definition at line 175 of file mpx_supt.c.

```
176 {
177   char msg[30];
178   int count=0;
179 
180     memset( msg, '\0', sizeof(msg));
181     strcpy(msg, "IDLE PROCESS EXECUTING.\n");
182     count = strlen(msg);
183 
184   while(1){
185     sys_req( WRITE, DEFAULT_DEVICE, msg, &count);
186     sys_req(IDLE, DEFAULT_DEVICE, NULL, NULL);
187   }
188 }
```

5.19.2.2 mpx_init()

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

Definition at line 108 of file mpx_supt.c.

```
109 {
110
111   current_module = cur_mod;
112   if (cur_mod == MEM_MODULE)
113         mem_module_active = TRUE;
114
115   if (cur_mod == IO_MODULE)
116         io_module_active = TRUE;
117 }
```

5.19.2.3 sys_alloc_mem()

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

Definition at line 146 of file mpx_supt.c.

```
147 {
148   if (!mem_module_active)
149     return (void *) kmalloc(size);
150   else
151     return (void *) (*student_malloc)(size);
152 }
```

5.19.2.4 sys_free_mem()

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

Definition at line 160 of file mpx_supt.c.

```
161 {
162   if (mem_module_active)
163     return (*student_free)(ptr);
164   // otherwise we don't free anything
165   return -1;
166 }
```

5.19.2.5 sys_req()

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

Definition at line 51 of file mpx_supt.c.

```
56 {
57     int return_code =0;
58
59   if (op_code == IDLE || op_code == EXIT){
```

```
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     // validate buffer pointer and count pointer
68     if (buffer_ptr == NULL)
69       return_code = INVALID_BUFFER;
70     else if (count_ptr == NULL || *count_ptr <= 0)
71       return_code = INVALID_COUNT;
72
73     // if parameters are valid store in the params structure
74     if ( return_code == 0){
75       params.op_code = op_code;
76       params.device_id = device_id;
77       params.buffer_ptr = buffer_ptr;
78       params.count_ptr = count_ptr;
79
80       if (!io_module_active){
81         // if default device
82         if (op_code == READ)
83           return_code = *(polling(buffer_ptr, count_ptr));
84
85         else //must be WRITE
86           return_code = serial_print(buffer_ptr);
87
88       } else {// I/O module is implemented
89         asm volatile ("int $60");
90       } // NOT IO_MODULE
91     }
92   } else return_code = INVALID_OPERATION;
93
94   return return_code;
95 }// end of sys_req
```

5.19.2.6 sys_set_free()

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

Definition at line 136 of file mpx_supt.c.

```
137 {
138   student_free = func;
139 }
```

5.19.2.7 sys_set_malloc()

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

Definition at line 126 of file mpx_supt.c.

```
127 {
128   student_malloc = func;
129 }
```

5.20 mpx_core/modules/R1/comHand.c File Reference

```
#include <stdint.h>
#include <string.h>
#include <system.h>
#include <core/serial.h>
#include <core/io.h>
#include "../mpx_supt.h"
#include "userFunctions.h"
```

Functions

- int comHand ()

  Description: Interprets user input to call the appropriate user functions.

5.20.1 Function Documentation

5.20.1.1 comHand()

```
int comHand ( )
```

Description: Interprets user input to call the appropriate user functions.

Definition at line 22 of file comHand.c.

```
22                      {
23
24          Help("\0");
25
26          char cmdBuffer[100];
27          int bufferSize = 99;
28          int quit = 0;
29          int shutdown = 0;
30
31          while(quit != 1)    {
32              memset(cmdBuffer, '\0', 100);
33            sys_req(READ, DEFAULT_DEVICE, cmdBuffer, &bufferSize);
34              char* FirstToken = strtok(cmdBuffer, "-");
35          char* SecondToken = strtok(NULL, "-");
36              char* ThirdToken = strtok(NULL, "-");
37              char* FourthToken = strtok(NULL, "-");
38              char* FifthToken = strtok(NULL, "-");
39              if(shutdown == 0){
40 /*****************************************************************************
41                  R1 comHand
42 *****************************************************************************/
43                  if(strcmp(FirstToken,"help") == 0 && strcmp(SecondToken,NULL) == 0)     {
44                      Help("\0");
45                  }
46                  //R1 Commands
47                  else if(strcmp(FirstToken,"help") == 0 && strcmp(SecondToken,"version") == 0 &&
       strcmp(ThirdToken,NULL) == 0) {
48                      Help("Version");
49                  }
50                  else if(strcmp(FirstToken,"help") == 0 && strcmp(SecondToken,"getDate") == 0 &&
       strcmp(ThirdToken,NULL) == 0) {
51                      Help("GetDate");
52                  }
53                  else if(strcmp(FirstToken,"help") == 0 && strcmp(SecondToken,"setDate") == 0 &&
       strcmp(ThirdToken,NULL) == 0) {
54                      Help("SetDate");
55                  }
56                  else if(strcmp(FirstToken,"help") == 0 && strcmp(SecondToken,"getTime") == 0 &&
       strcmp(ThirdToken,NULL) == 0) {
57                      Help("GetTime");
58                  }
59                  else if(strcmp(FirstToken,"help") == 0 && strcmp(SecondToken,"setTime") == 0 &&
       strcmp(ThirdToken,NULL) == 0) {
60                      Help("SetTime");
61                  }
62                  // R2 Commands
63                  else if(strcmp(FirstToken,"help") == 0 && strcmp(SecondToken,"suspend") == 0 &&
       strcmp(ThirdToken,NULL) == 0) {
64                      Help("suspend");
65                  }
66                  else if(strcmp(FirstToken,"help") == 0 && strcmp(SecondToken,"resume") == 0 &&
       strcmp(ThirdToken,NULL) == 0)  {
67                      Help("resume");
68                  }
69                  else if(strcmp(FirstToken,"help") == 0 && strcmp(SecondToken,"setPriority") == 0 &&
       strcmp(ThirdToken,NULL) == 0) {
70                      Help("setPriority");
71                  }
```

```
72                 else if(strcmp(FirstToken,"help") == 0 && strcmp(SecondToken,"showPCB") == 0 &&
       strcmp(ThirdToken,NULL) == 0) {
73                     Help("showPCB");
74                 }
75                 else if(strcmp(FirstToken,"help") == 0 && strcmp(SecondToken,"showAll") == 0 &&
       strcmp(ThirdToken,NULL) == 0) {
76                     Help("showAll");
77                 }
78                 else if(strcmp(FirstToken,"help") == 0 && strcmp(SecondToken,"showReady") == 0 &&
       strcmp(ThirdToken,NULL) == 0)   {
79                     Help("showReady");
80                 }
81                 else if(strcmp(FirstToken,"help") == 0 && strcmp(SecondToken,"showBlocked") == 0 &&
       strcmp(ThirdToken,NULL) == 0) {
82                     Help("showBlocked");
83                 }
84                 // Temporary R2 commands
85                 else if(strcmp(FirstToken,"help") == 0 && strcmp(SecondToken,"createPCB") == 0 &&
       strcmp(ThirdToken,NULL) == 0)   {
86                     Help("createPCB");
87                 }
88                 else if(strcmp(FirstToken,"help") == 0 && strcmp(SecondToken,"deletePCB") == 0 &&
       strcmp(ThirdToken,NULL) == 0)   {
89                     Help("deletePCB");
90                 }
91                 else if(strcmp(FirstToken,"help") == 0 && strcmp(SecondToken,"block") == 0 &&
       strcmp(ThirdToken,NULL) == 0)   {
92                     Help("block");
93                 }
94                 else if(strcmp(FirstToken,"help") == 0 && strcmp(SecondToken,"unblock") == 0 &&
       strcmp(ThirdToken,NULL) == 0) {
95                     Help("unblock");
96                 }
97
98                 else if(strcmp(FirstToken,"version") == 0 && strcmp(SecondToken,NULL) == 0)
99                     Version();
100
101                 else if(strcmp(FirstToken,"getDate") == 0 && strcmp(SecondToken,NULL) == 0)
102                     GetDate();
103
104                 else if(strcmp(FirstToken,"setDate") == 0){
105                     if (EdgeCase(SecondToken) == 1 && EdgeCase(ThirdToken) == 1 && EdgeCase(FourthToken)
       == 1 && EdgeCase(FifthToken) == 1)     {
106                         SetDate(atoi(SecondToken), atoi(ThirdToken), atoi(FourthToken),
       atoi(FifthToken));
107                     }
108                     else
109                         printf("\x1b[31m""\nERROR: Invalid parameters for setDate \n""\x1b[0m");
110                 }
111                 else if(strcmp(FirstToken,"getTime") == 0 && strcmp(SecondToken,NULL) == 0) //Return the
       current time held by the registers.
112                     GetTime();
113                 else if(strcmp(FirstToken,"setTime") == 0 && strcmp(FifthToken,NULL) == 0){
114                     if (EdgeCase(SecondToken) == 1 && EdgeCase(ThirdToken) == 1 && EdgeCase(FourthToken)
       == 1)     {
115                             SetTime(atoi(SecondToken), atoi(ThirdToken), atoi(FourthToken));     //input
       as Hour-Minute-Seconds
116                     }
117                     else
118                         printf("\x1b[31m""\nERROR: Invalid parameters for setTime \n""\x1b[0m");
119                 }
120 /*****************************************************************************
121                 R2 comHand
122 *****************************************************************************/
123                 if(strcmp(FirstToken,"suspend") == 0 && strcmp(SecondToken,NULL) == 0) {
124                     Suspend();
125                 }
126                 else if(strcmp(FirstToken,"resume") == 0 && strcmp(SecondToken,NULL) == 0) {
127                     Resume();
128                 }
129                 else if(strcmp(FirstToken,"setPriority") == 0 && strcmp(SecondToken,NULL) == 0) {
130                     setPriority();
131                 }
132                 else if(strcmp(FirstToken,"showPCB") == 0 && strcmp(SecondToken,NULL) == 0) {
133                     Show_PCB();
134                 }
135                 else if(strcmp(FirstToken,"showAll") == 0 && strcmp(SecondToken,NULL) == 0) {
136                     showAll();
137                 }
138                 else if(strcmp(FirstToken,"showReady") == 0 && strcmp(SecondToken,NULL) == 0) {
139                     showReady();
140                 }
141                 else if(strcmp(FirstToken,"showBlocked") == 0 && strcmp(SecondToken,NULL) == 0) {
142                     showBlocked();
143                 }
144
145                 /********** R2 Temp Commands **********/
```

```
146                 else if(strcmp(FirstToken,"createPCB") == 0 && strcmp(SecondToken,NULL) == 0) {
147                     Create_PCB();
148                 }
149                 else if(strcmp(FirstToken,"deletePCB") == 0 && strcmp(SecondToken,NULL) == 0) {
150                     Delete_PCB();
151                 }
152                 else if(strcmp(FirstToken,"block") == 0 && strcmp(SecondToken,NULL) == 0) {
153                     Block();
154                 }
155                 else if(strcmp(FirstToken,"unblock") == 0 && strcmp(SecondToken,NULL) == 0) {
156                     Unblock();
157                 }
158
159 /*****************************************************************************
160                 shutdown comHand
161 *****************************************************************************/
162                 else if(strcmp(FirstToken,"shutdown") == 0 && strcmp(SecondToken,NULL) == 0){
163                     printf("\x1b[33m""\nAre you sure you want to shutdown? [yes/no]\n""\x1b[0m");
164                     shutdown = 1;
165                 }
166                 else
167                     printf("\x1b[31m""\nERROR: Not a valid command \n""\x1b[0m");
168             }
169             else{
170                 if(strcmp(FirstToken,"yes") == 0 && shutdown == 1)    {
171                     quit = 1;
172                 }
173                 else if(strcmp(FirstToken,"no") == 0){
174                     printf("\x1b[33m""\nShutdown Cancelled\x1b[0m");
175                     shutdown = 0;
176                 }
177                 else
178                     printf("\x1b[31m""\nERROR: Please enter \"yes\" or \"no\" \n""\x1b[0m");
179             }
180         }
181         return 0;   //shutdown procedure
182     }
```

5.21 mpx_core/modules/R1/comHand.h File Reference

Functions

- int comHand ()

  Description: Interprets user input to call the appropriate user functions.

5.21.1 Function Documentation

5.21.1.1 comHand()

```
int comHand ( )
```

Description: Interprets user input to call the appropriate user functions.

Definition at line 22 of file comHand.c.

```
22                  {
23
24         Help("\0");
25
26         char cmdBuffer[100];
27         int bufferSize = 99;
28         int quit = 0;
29         int shutdown = 0;
30
31         while(quit != 1)    {
32             memset(cmdBuffer, '\0', 100);
33           sys_req(READ, DEFAULT_DEVICE, cmdBuffer, &bufferSize);
```

```
34              char* FirstToken = strtok(cmdBuffer, "-");
35          char* SecondToken = strtok(NULL, "-");
36              char* ThirdToken = strtok(NULL, "-");
37              char* FourthToken = strtok(NULL, "-");
38              char* FifthToken = strtok(NULL, "-");
39              if(shutdown == 0){
40 /*****************************************************************************
41                  R1 comHand
42 *****************************************************************************/
43                  if(strcmp(FirstToken,"help") == 0 && strcmp(SecondToken,NULL) == 0)     {
44                      Help("\0");
45                  }
46                  //R1 Commands
47                  else if(strcmp(FirstToken,"help") == 0 && strcmp(SecondToken,"version") == 0 &&
       strcmp(ThirdToken,NULL) == 0) {
48                      Help("Version");
49                  }
50                  else if(strcmp(FirstToken,"help") == 0 && strcmp(SecondToken,"getDate") == 0 &&
       strcmp(ThirdToken,NULL) == 0) {
51                      Help("GetDate");
52                  }
53                  else if(strcmp(FirstToken,"help") == 0 && strcmp(SecondToken,"setDate") == 0 &&
       strcmp(ThirdToken,NULL) == 0) {
54                      Help("SetDate");
55                  }
56                  else if(strcmp(FirstToken,"help") == 0 && strcmp(SecondToken,"getTime") == 0 &&
       strcmp(ThirdToken,NULL) == 0) {
57                      Help("GetTime");
58                  }
59                  else if(strcmp(FirstToken,"help") == 0 && strcmp(SecondToken,"setTime") == 0 &&
       strcmp(ThirdToken,NULL) == 0) {
60                      Help("SetTime");
61                  }
62                  // R2 Commands
63                  else if(strcmp(FirstToken,"help") == 0 && strcmp(SecondToken,"suspend") == 0 &&
       strcmp(ThirdToken,NULL) == 0) {
64                      Help("suspend");
65                  }
66                  else if(strcmp(FirstToken,"help") == 0 && strcmp(SecondToken,"resume") == 0 &&
       strcmp(ThirdToken,NULL) == 0)  {
67                      Help("resume");
68                  }
69                  else if(strcmp(FirstToken,"help") == 0 && strcmp(SecondToken,"setPriority") == 0 &&
       strcmp(ThirdToken,NULL) == 0) {
70                      Help("setPriority");
71                  }
72                  else if(strcmp(FirstToken,"help") == 0 && strcmp(SecondToken,"showPCB") == 0 &&
       strcmp(ThirdToken,NULL) == 0) {
73                      Help("showPCB");
74                  }
75                  else if(strcmp(FirstToken,"help") == 0 && strcmp(SecondToken,"showAll") == 0 &&
       strcmp(ThirdToken,NULL) == 0) {
76                      Help("showAll");
77                  }
78                  else if(strcmp(FirstToken,"help") == 0 && strcmp(SecondToken,"showReady") == 0 &&
       strcmp(ThirdToken,NULL) == 0)   {
79                      Help("showReady");
80                  }
81                  else if(strcmp(FirstToken,"help") == 0 && strcmp(SecondToken,"showBlocked") == 0 &&
       strcmp(ThirdToken,NULL) == 0) {
82                      Help("showBlocked");
83                  }
84                  // Temporary R2 commands
85                  else if(strcmp(FirstToken,"help") == 0 && strcmp(SecondToken,"createPCB") == 0 &&
       strcmp(ThirdToken,NULL) == 0)   {
86                      Help("createPCB");
87                  }
88                  else if(strcmp(FirstToken,"help") == 0 && strcmp(SecondToken,"deletePCB") == 0 &&
       strcmp(ThirdToken,NULL) == 0)   {
89                      Help("deletePCB");
90                  }
91                  else if(strcmp(FirstToken,"help") == 0 && strcmp(SecondToken,"block") == 0 &&
       strcmp(ThirdToken,NULL) == 0)   {
92                      Help("block");
93                  }
94                  else if(strcmp(FirstToken,"help") == 0 && strcmp(SecondToken,"unblock") == 0 &&
       strcmp(ThirdToken,NULL) == 0) {
95                      Help("unblock");
96                  }
97
98                  else if(strcmp(FirstToken,"version") == 0 && strcmp(SecondToken,NULL) == 0)
99                      Version();
100
101                  else if(strcmp(FirstToken,"getDate") == 0 && strcmp(SecondToken,NULL) == 0)
102                      GetDate();
103
104                  else if(strcmp(FirstToken,"setDate") == 0){
```

```
105                     if (EdgeCase(SecondToken) == 1 && EdgeCase(ThirdToken) == 1 && EdgeCase(FourthToken)
    == 1 && EdgeCase(FifthToken) == 1)    {
106                         SetDate(atoi(SecondToken), atoi(ThirdToken), atoi(FourthToken),
    atoi(FifthToken));
107                     }
108                     else
109                         printf("\x1b[31m""\nERROR: Invalid parameters for setDate \n""\x1b[0m");
110                 }
111                 else if(strcmp(FirstToken,"getTime") == 0 && strcmp(SecondToken,NULL) == 0) //Return the
    current time held by the registers.
112                     GetTime();
113                 else if(strcmp(FirstToken,"setTime") == 0 && strcmp(FifthToken,NULL) == 0){
114                     if (EdgeCase(SecondToken) == 1 && EdgeCase(ThirdToken) == 1 && EdgeCase(FourthToken)
    == 1)    {
115                         SetTime(atoi(SecondToken), atoi(ThirdToken), atoi(FourthToken));     //input
    as Hour-Minute-Seconds
116                     }
117                     else
118                         printf("\x1b[31m""\nERROR: Invalid parameters for setTime \n""\x1b[0m");
119                 }
120 /*****************************************************************************
121                 R2 comHand
122 *****************************************************************************/
123                 if(strcmp(FirstToken,"suspend") == 0 && strcmp(SecondToken,NULL) == 0) {
124                     Suspend();
125                 }
126                 else if(strcmp(FirstToken,"resume") == 0 && strcmp(SecondToken,NULL) == 0) {
127                     Resume();
128                 }
129                 else if(strcmp(FirstToken,"setPriority") == 0 && strcmp(SecondToken,NULL) == 0) {
130                     setPriority();
131                 }
132                 else if(strcmp(FirstToken,"showPCB") == 0 && strcmp(SecondToken,NULL) == 0) {
133                     Show_PCB();
134                 }
135                 else if(strcmp(FirstToken,"showAll") == 0 && strcmp(SecondToken,NULL) == 0) {
136                     showAll();
137                 }
138                 else if(strcmp(FirstToken,"showReady") == 0 && strcmp(SecondToken,NULL) == 0) {
139                     showReady();
140                 }
141                 else if(strcmp(FirstToken,"showBlocked") == 0 && strcmp(SecondToken,NULL) == 0) {
142                     showBlocked();
143                 }
144 
145                 /********** R2 Temp Commands **********/
146                 else if(strcmp(FirstToken,"createPCB") == 0 && strcmp(SecondToken,NULL) == 0) {
147                     Create_PCB();
148                 }
149                 else if(strcmp(FirstToken,"deletePCB") == 0 && strcmp(SecondToken,NULL) == 0) {
150                     Delete_PCB();
151                 }
152                 else if(strcmp(FirstToken,"block") == 0 && strcmp(SecondToken,NULL) == 0) {
153                     Block();
154                 }
155                 else if(strcmp(FirstToken,"unblock") == 0 && strcmp(SecondToken,NULL) == 0) {
156                     Unblock();
157                 }
158 
159 /*****************************************************************************
160                 shutdown comHand
161 *****************************************************************************/
162                 else if(strcmp(FirstToken,"shutdown") == 0 && strcmp(SecondToken,NULL) == 0){
163                     printf("\x1b[33m""\nAre you sure you want to shutdown? [yes/no]\n""\x1b[0m");
164                     shutdown = 1;
165                 }
166                 else
167                     printf("\x1b[31m""\nERROR: Not a valid command \n""\x1b[0m");
168             }
169             else{
170                 if(strcmp(FirstToken,"yes") == 0 && shutdown == 1)    {
171                     quit = 1;
172                 }
173                 else if(strcmp(FirstToken,"no") == 0){
174                     printf("\x1b[33m""\nShutdown Cancelled\x1b[0m");
175                     shutdown = 0;
176                 }
177                 else
178                     printf("\x1b[31m""\nERROR: Please enter \"yes\" or \"no\" \n""\x1b[0m");
179             }
180         }
181         return 0;   //shutdown procedure
182     }
```

5.22 mpx_core/modules/R1/userFunctions.c File Reference

```
#include <stdint.h>
#include <string.h>
#include <system.h>
#include <core/serial.h>
#include <core/io.h>
#include "../mpx_supt.h"
#include "userFunctions.h"
```

Functions

- char * itoa (int num)

  Description: An integer is taken and seperated into individual chars and then all placed into a character array.
- int BCDtoDec (int BCD)

  Description: Changes binary number to decimal numbers.
- int DectoBCD (int Decimal)

  Description: Changes decimal numbers to binary numbers.
- void printf (char msg[ ])
- int EdgeCase (char *pointer)

  Description: Compares pointer char to validate if it is a number or not.
- void SetTime (int hours, int minutes, int seconds)

  Description: sets the time register to the new values that the user inputed, all values must be inputed as SetTime(↩ Hours, Minutes, Seconds).
- void GetTime ()

  Description: retrieve and return the time values for hours, minutes, and seconds form the clock register using inb(Port,address).
- void SetDate (int day, int month, int millennium, int year)

  Description: Sets the date register to the new values that the user inputed, all values must be inputed as SetDime(day, month, millenial, year).
- void GetDate ()

  Description: Returns the full date back to the user in decimal form.
- void Version ()

  Description: Simply returns a char containing "Version: R(module).
- char toLowercase (char c)

  Description: If a letter is uppercase, it changes it to lowercase.
- void Help (char *request)

  Brief Description: Gives helpful information for one of the functions.
- void Suspend (Char *Process_Name)

  Brief Description: Places a PCD in the suspended state and reinserts it into the appropriate queue.
- void Resume (Char *Process_Name)

  Brief Description: Places a PCD in the not suspended state and reinserts it into the appropriate queue.
- void Set_Priority (Char *Process_Name, int Priority)

  Brief Description: Sets PCB priority and reinserts the process into the correct place in the correct queue.
- void Show_PCB (char *Process_Name)

  Brief Description: Displays the process name, class, state, suspended status, and priority of a PCB.
- void Show_All ()

  Brief Description: Displays the process name, class, state, suspended status, and priority of all PCB in the ready and blocked queues.
- void Show_Ready ()

Brief Description: Displays the process name, class, state, suspended status, and priority of all PCB in the ready queue.

- void Show_Blocked ()

  Brief Description: Displays the process name, class, state, suspended status, and priority of all PCB in the blocked queue.

5.22.1 Function Documentation

5.22.1.1 BCDtoDec()

```
int BCDtoDec (
            int BCD )
```

Description: Changes binary number to decimal numbers.

Parameters

value	Binary number to be changed to decimal

Definition at line 64 of file userFunctions.c.

```
64                              {
65          return (((BCD»4)*10) + (BCD & 0xF));
66      }
```

5.22.1.2 DectoBCD()

```
int DectoBCD (
            int Decimal )
```

Description: Changes decimal numbers to binary numbers.

Parameters

Decimal	Decimal number to be changed to binary

Definition at line 71 of file userFunctions.c.

```
71                                  {
72           return (((Decimal/10) « 4) | (Decimal % 10));
73      }
```

5.22.1.3 EdgeCase()

```
int EdgeCase (
            char * pointer )
```

Description: Compares pointer char to validate if it is a number or not.

Parameters

Compares	pointer char to validate if it is a number or not.

Definition at line 83 of file userFunctions.c.

```
83                                  {
84      int valid = 0;
85      if (strcmp(pointer, "00") == 0){
86        valid = 1;
87        return valid;
88      }
89      int i, j;
90      for (i = 0; i < strlen(pointer); i++){
91          valid = 0;
92          for(j = 0; j <= 99; j++){
93              if(strcmp(pointer,itoa(j)) == 0)
94                  valid = 1;
95          }
96          if(valid == 0){
97            return valid;
98          }
99      }
100      return valid;
101    }
```

5.22.1.4 GetDate()

```
void GetDate ( )
```

Description: Returns the full date back to the user in decimal form.

No parameters.

Definition at line 225 of file userFunctions.c.

```
225                     {
226       int check = 2;
227       outb(0x70,0x07);
228         unsigned char day = BCDtoDec(inb(0x71));
229         outb(0x70,0x08);
230         unsigned char month = BCDtoDec(inb(0x71));
231         outb(0x70,0x32);
232         unsigned char millennium = BCDtoDec(inb(0x71));
233         char msg[2] = "-";
234         char msg3[10] = "Date: ";
235         printf(msg3);
236         sys_req(WRITE, COM1, itoa(day), &check);
237         printf(msg);
238         sys_req(WRITE, COM1, itoa(month), &check);
239         printf(msg);
240         sys_req(WRITE, COM1, itoa(millennium), &check);
241      outb(0x70,0x09);
242      if(BCDtoDec(inb(0x71)) == 0){
243        sys_req(WRITE, COM1, "00", &check);
244      }
245      else {
246          unsigned char year = BCDtoDec(inb(0x71));
247          sys_req(WRITE, COM1, itoa(year), &check);
248      }
249          printf("\n");
250      }
```

5.22.1.5 GetTime()

```
void GetTime ( )
```

Description: retrieve and return the time values for hours, minutes, and seconds form the clock register using inb(Port,address).

No parameters.

Definition at line 147 of file userFunctions.c.

```
147                     {
148     int check = 2;
149     int hour;
150     int minute;
151     int second;
152         outb(0x70,0x04);
153         unsigned char hours = inb(0x71);
154         outb(0x70,0x02);
155         unsigned char minutes = inb(0x71);
156         outb(0x70,0x00);
157         unsigned char seconds = inb(0x71);
158         char msg1[2] = ":";
159         char msg2[10] = "Time: ";
160         printf(msg2);
161         hour = BCDtoDec(hours);
162         sys_req(WRITE, COM1, itoa(hour), &check);
163         printf(msg1);
164         minute = BCDtoDec(minutes);
165         sys_req(WRITE, COM1, itoa(minute), &check);
166         printf(msg1);
167         second = BCDtoDec(seconds);
168         sys_req(WRITE, COM1, itoa(second), &check);
169       printf("\n");
170     }
```

5.22.1.6 Help()

```
void Help (
              char * request )
```

Brief Description: Gives helpful information for one of the functions.

Description: Can except a string as a pointer, if the pointer is null then the function will print a complete list of avaliable commands to the console. If the pointer is a avaliable commands then instructions on how to use the command will be printed. If the command does not exist then a message explaining that it is not a valid command will be displayed.

Parameters

request	Character pointer that matches the name of the function that you need help with.

Definition at line 274 of file userFunctions.c.

```
274                             {
275     int check = 1;
276         if (request[0] == '\0') {
277                 printf("\n to chain commands and parameters, please use \"-\" between keywords \n");
278             printf("\n getDate \n setDate \n getTime \n setTime \n version \n shutdown \n\n");
279         }
280         else if (strcmp(request, "GetDate") == 0) {
281                 printf("\n getDate returns the current date that is loaded onto the operating
       system.\n");
282         }
283         else if (strcmp(request, "SetDate") == 0) {
```

```
284                 printf("\n setDate allows the user to reset the correct date into the system, as follows
    setDate-"BLU"day"RESET"-"BLU"month"RESET"-"BLU"year"RESET".\n Time must be inputed as a two digit
    number, Example 02 or 00");
285         }
286         else if (strcmp(request, "GetTime") == 0) {
287                 printf("\n getTime returns the current time as hours, minutes, seconds that is loaded
    onto the operating system.\n");
288         }
289         else if (strcmp(request, "SetTime") == 0) {
290                 printf("\n setTime allows the user to reset the correct time into the system, as follows
    setTime-"BLU"hour"RESET"-"BLU"minute"RESET"-"BLU"second"RESET".\n Time must be inputed as a two digit
    number, Example 02 or 00");
291             }
292         else if (strcmp(request, "Version") == 0) {
293                 printf("\n version returns the current operating software version that the system is
    running.\n");
294         }
295      else if(strcmp(request, "shutdown") == 0) {
296        printf("\n shutdown shuts down the system.\n");
297      }
298
299 /************************R2 Commands*******************************************************/
300     else if(strcmp(request,"suspend") == 0) {
301     printf("\n Suspend takes in the name of a PCB then places it into the suspended state and reinserts
    it into the correct queue.\n");
302     }
303     else if(strcmp(FirstToken,"resume") == 0) {
304     printf("\n Resume takes in the name of a PCB then removes it from the suspended state and adds it to
    the correct queue.\n");
305     }
306     else if(strcmp(FirstToken,"setPriority") == 0) {
307     printf("\n SetPriority takes in the name of a PCB and the priority it needs to be set to then
    reinstates the specified PCB into a new location by priority.\n");
308     }
309     else if(strcmp(FirstToken,"showPCB") == 0) {
310     printf("\n ShowPCB takes in the name of a PCB and returns all the associated attributes to the
    user.\n");
311     }
312     else if(strcmp(FirstToken,"showAll") == 0) {
313     printf("\n ShowAll takes no parameters but returns all PCB's that are currently in any of the
    queues.\n");
314     }
315     else if(strcmp(FirstToken,"showReady") == 0) {
316     printf("\n ShowReady takes in no parameters but returns all PCB's and there attributes that
    currently are in the ready state.\n");
317     }
318     else if(strcmp(FirstToken,"showBlocked") == 0) {
319     printf("\n ShowBlocked takes in no parameters but returns all PCB's and there attributes that
    currently are in the blocked state.\n");
320     }
321
322 /******************************** R2 Temp Commands
    *******************************************************/
323     else if(strcmp(FirstToken,"createPCB") == 0) {
324     printf("\n  CreatePCB takes in the process_name, process_class, and process_priority. Then assigns
    this new process into the correct queue.\n");
325     }
326     else if(strcmp(FirstToken,"deletePCB") == 0) {
327     printf("\n DeletePCB takes in the process_name then deletes it from the queue and free's all the
    memory that was previously allocated to the specified PCB.\n");
328     }
329     else if(strcmp(FirstToken,"block") == 0) {
330     printf("\n Block takes in the process_name then sets it's state to blocked and reinserts it back
    into the correct queue.\n");
331     }
332     else if(strcmp(FirstToken,"unblock") == 0) {
333     printf("\n Unblock takes in the process_name then sets it's state to ready and reinserts it back
    into the correct queue.\n");
334     }
335     else    {
336      printf("\x1b[31m""\nThe requested command does not exist please refer to the Help function for a
    full list of commands.\n""\x1b[0m");
337     }
338 }
```

5.22.1.7 itoa()

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

Description: An integer is taken and seperated into individual chars and then all placed into a character array.

Adapted from geeksforgeeks.org.

Parameters

num	integer to be put into array Title: itoa Author: Neha Mahajan Date: 29 May, 2017 Availability: https://www.geeksforgeeks.org/implement-itoa/

Definition at line 33 of file userFunctions.c.

```
34      {
35              int i,j,k,count;
36              i = num;
37              j = 0;
38              count = 0;
39          while(i){ // count number of digits
40              count++;
41              i /= 10;
42          }
43
44          char* arr1;
45          char arr2[count];
46          arr1 = (char*)sys_alloc_mem(count); //memory allocation
47
48          while(num){ // seperate last digit from number and add ASCII
49              arr2[++j] = num%10 + '0';
50              num /= 10;
51          }
52
53          for(k = 0; k < j; k++){ // reverse array results
54              arr1[k] = arr2[j-k];
55          }
56          arr1[k] = '\0';
57
58          return(char*)arr1;
59      }
```

5.22.1.8 printf()

```
void printf (
            char msg[] )
```

Definition at line 75 of file userFunctions.c.

```
75                              {
76        int check =strlen(msg);
77          sys_req(WRITE, COM1, msg, &check);
78      }
```

5.22.1.9 Resume()

```
void Resume (
            Char * Process_Name )
```

Brief Description: Places a PCD in the not suspended state and reinserts it into the appropriate queue.

Description: Can except a string as a pointer that is the Process Name. Places a PCB in the not suspended state and reinserts it into the appropriate queue. An error check for valid Process Name.

Parameters

Process_Name	Character pointer that matches the name of process.

Definition at line 371 of file userFunctions.c.

```
371                                 {
372
373
374   // Name Error check
375   // Error check (Valid Name)
376   //if (Process_Name != valid name){
377   //  printf("\x1b[31m""\nERROR: Not a valid process name \n""\x1b[0m");
378   //}
379
380 }
```

5.22.1.10 Set_Priority()

```
void Set_Priority (
            Char * Process_Name,
            int Priority )
```

Brief Description: Sets PCB priority and reinserts the process into the correct place in the correct queue.

Description: Can except a string as a pointer that is the Process Name. Can accept and integer than is the Priority. Sets a PCB's priority and reinserts the process into the correct place in the correct queue. An error check for valid Process Name and an error check for a valid priority 1 - 9.

Parameters

Process_Name	Character pointer that matches the name of process.
Priority	integer that matches the priority number.

Definition at line 388 of file userFunctions.c.

```
388                                                 {
389   int i;
390
391   // Name Error check
392   // Error check (Valid Name)
393   //if (Process_Name != valid name){
394   //  printf("\x1b[31m""\nERROR: Not a valid process name \n""\x1b[0m");
395   //}
396   // Priority error check
397   for(i = 0; i < 9; i++){
398     if(Priority == i){
399       break;
400     }
401     else{
402       printf("\x1b[31m""\nERROR: Not a valid Priority \n""\x1b[0m")
403     }
404   }
405
406 }
```

5.22.1.11 SetDate()

```
void SetDate (
            int day,
```

```
            int month,
            int millennium,
            int year )
```

Description: Sets the date register to the new values that the user inputed, all values must be inputed as SetDime(day, month, millenial, year).

Parameters

day	Integer to be set in the Day position
month	Integer to be set in the Month position
millenial	Integer to be set in the Millenial position
year	Integer to be set in the Year position

Definition at line 178 of file userFunctions.c.

```
178                                                     {
179     outb(0x70,0x07);
180     int tempDay = BCDtoDec(inb(0x71));
181     outb(0x70,0x08);
182     int tempMonth = BCDtoDec(inb(0x71));
183     outb(0x70,0x32);
184     int tempMillennium = BCDtoDec(inb(0x71));
185     outb(0x70,0x09);
186     int tempYear = BCDtoDec(inb(0x71));
187     cli();
188         outb(0x70,0x07);
189         outb(0x71,DectoBCD (day));
190         outb(0x70,0x08);
191         outb(0x71,DectoBCD (month));
192         outb(0x70,0x32);
193         outb(0x71,DectoBCD (millennium));
194         outb(0x70,0x09);
195         outb(0x71,DectoBCD (year));
196         sti();
197     outb(0x70,0x07);
198     unsigned char newDay = BCDtoDec(inb(0x71));
199     outb(0x70,0x08);
200     unsigned char newMonth = BCDtoDec(inb(0x71));
201     outb(0x70,0x32);
202     unsigned char newMillennium = BCDtoDec(inb(0x71));
203     outb(0x70,0x09);
204     unsigned char newYear = BCDtoDec(inb(0x71));
205     if(newDay != day || newMonth != month || newMillennium != millennium || newYear != year){
206       printf("Your input was invalid\n");
207       cli();
208         outb(0x70,0x07);
209         outb(0x71,DectoBCD (tempDay));
210         outb(0x70,0x08);
211         outb(0x71,DectoBCD (tempMonth));
212         outb(0x70,0x32);
213         outb(0x71,DectoBCD (tempMillennium));
214         outb(0x70,0x09);
215         outb(0x71,DectoBCD (tempYear));
216         sti();
217     }
218     else
219       printf("Date Set\n");
220     }
```

5.22.1.12 SetTime()

```
void SetTime (
            int hours,
            int minutes,
            int seconds )
```

Description: sets the time register to the new values that the user inputed, all values must be inputed as SetTime(Hours, Minutes, Seconds).

Parameters

hours	Integer to be set in the Hour position
minutes	Integer to be set in the Minutes position
seconds	Integer to be set in the Seconds position

Definition at line 108 of file userFunctions.c.

```
108                                                     {
109     outb(0x70,0x04);
110     unsigned char tempHours = BCDtoDec(inb(0x71));
111     outb(0x70,0x02);
112     unsigned char tempMinutes = BCDtoDec(inb(0x71));
113     outb(0x70,0x00);
114     unsigned char tempSeconds = BCDtoDec(inb(0x71));
115         cli(); //outb(device + 1, 0x00); //disable interrupts
116         outb(0x70,0x04);
117         outb(0x71, DectoBCD(hours));// change to bcd
118         outb(0x70,0x02);
119         outb(0x71, DectoBCD(minutes));
120         outb(0x70,0x00);
121         outb(0x71, DectoBCD(seconds));
122         sti();  //outb(device + 4, 0x0B); //enable interrupts, rts/dsr set
123     outb(0x70,0x04);
124     unsigned char newHours = BCDtoDec(inb(0x71));
125     outb(0x70,0x02);
126     unsigned char newMinutes = BCDtoDec(inb(0x71));
127     outb(0x70,0x00);
128     unsigned char newSeconds = BCDtoDec(inb(0x71));
129     if(newHours != hours || newMinutes != minutes || newSeconds != seconds){
130       printf("Your input was invalid\n");
131       cli(); //outb(device + 1, 0x00); //disable interrupts
132         outb(0x70,0x04);
133         outb(0x71, DectoBCD(tempHours));// change to bcd
134         outb(0x70,0x02);
135         outb(0x71, DectoBCD(tempMinutes));
136         outb(0x70,0x00);
137         outb(0x71, DectoBCD(tempSeconds));
138         sti();  //outb(device + 4, 0x0B); //enable interrupts, rts/dsr set
139     }
140     else
141       printf("Time Set\n");
142     }
```

5.22.1.13 Show_All()

```
void Show_All ( )
```

Brief Description: Displays the process name, class, state, suspended status, and priority of all PCB in the ready and blocked queues.

Description: The process name, claas, state, suspend status, and priority of each of he PCB's in the ready and blocked queues.

Definition at line 439 of file userFunctions.c.

```
439                 {
440   int check = 20;
441   int i;
442   int j;
443   for(i = 0; i < sizeof(ready queue);i++)   {
444     char rProcess_Name = ready queue [i] Process_Name;
445     int rClass =  ready queue [i] class;
446     char rState = ready queue[i] state;
447     char rStatus = ready queue[i] status;
448     int rPriority = ready queue[i] priority;
449     sys_req(WRITE, COM1, rProcess_Name, &check);
450     sys_req(WRITE, COM1, itoa(rClass), &check);
451     sys_req(WRITE, COM1, rState, &check);
452     sys_req(WRITE, COM1, rStatus, &check);
453     sys_req(WRITE, COM1, itoa(rPriority), &check);
454   }
```

```
455   for(j = 0; j < sizeof(blocked queue); j++){
456     char bProcess_Name = blocked queue [j] Process_Name;
457     int bClass =  blocked queue [j] class;
458     char bState = blocked queue[j] state;
459     char bStatus = blocked queue[j] status;
460     int bPriority = blocked queue[j] priority;
461     sys_req(WRITE, COM1, bProcess_Name, &check);
462     sys_req(WRITE, COM1, itoa(bClass), &check);
463     sys_req(WRITE, COM1, bState, &check);
464     sys_req(WRITE, COM1, bStatus, &check);
465     sys_req(WRITE, COM1, itoa(bPriority), &check);
466   }
467 }
```

5.22.1.14 Show_Blocked()

```
void Show_Blocked ( )
```

Brief Description: Displays the process name, class, state, suspended status, and priority of all PCB in the blocked queue.

Description: The process name, claas, state, suspend status, and priority of each of he PCB's in the blocked queue.
Brief Description: Calls SetupPCB() and inserts PCB into appropriate queue.

Description: Can except a string as a pointer that is the Process Name. Can accept two integers, Priority and Class. SetupPCB() will be called and the PCB will be inserted into the appropriate queue. An error check for unique and valid Process Name, an error check for valid process class, and an error check for process priority.

Parameters

Process_Name	Character pointer that matches the name of process.
Priority	integer that matches the priority number.
Class	integer that matches the class number.

Brief Description: Removes PCB from appropriate queue and frees all associated memory.

Description: Can except a string as a pointer that is the Process Name. Removes PCB from the appropriate queue and then frees all associated memory. An error check to make sure process name is valid.

Parameters

Process_Name	Character pointer that matches the name of process.

Brief Description: Places a PCD in the blocked state and reinserts it into the correct queue.

Description: Can except a string as a pointer that is the Process Name. The specified PCB will be places in a blocked state and reinserted into the appropriate queue. An error check for a valid name occurs.

Parameters

Process_Name	Character pointer that matches the name of process.

Brief Description: Places a PCD in the unblocked state and reinserts it into the correct queue.

Description: Can except a string as a pointer that is the Process Name. The specified PCB will be places in an unblocked state and reinserted into the appropriate queue. An error check for a valid name occurs.

Parameters

Process_Name	Character pointer that matches the name of process.

Definition at line 492 of file userFunctions.c.

5.22.1.15 Show_PCB()

```
void Show_PCB (
            char * Process_Name )
```

Brief Description: Displays the process name, class, state, suspended status, and priority of a PCB.

Description: Can except a string as a pointer that is the Process Name. The process name, claas, state, suspend status, and priority of a PCB are displayed. An error check for a valid name occurs.

Parameters

Process_Name	Character pointer that matches the name of process

Definition at line 413 of file userFunctions.c.

```
413                                     {
414   int class, check, state, prior;
415   char[] name;
416   check = 10;
417   PCB* pcb = FindPCB(Process_Name);
418   class = pcb->Process_Class;
419   name = pcb->Process_Name;
420   state = pcb->ReadyState;
421   status = pcb->SuspendedState;
422   prior = pcb->Priority;
423
424   if(name == NULL){
425     printf("\x1b[31m""\nERROR: Not a valid process name \n""\x1b[0m");
426   } else{
427     sys_req(WRITE, COM1, name, &check);
428     sys_req(WRITE, COM1, itoa(class), &check);
429     sys_req(WRITE, COM1, itoa(state), &check);
430     sys_req(WRITE, COM1, itoa(status), &check);
431     sys_req(WRITE, COM1, itoa(priot), &check);
432 }
433 }
```

5.22.1.16 Show_Ready()

```
void Show_Ready ( )
```

Brief Description: Displays the process name, class, state, suspended status, and priority of all PCB in the ready queue.

Description: The process name, claas, state, suspend status, and priority of each of he PCB's in the ready queue.

Definition at line 472 of file userFunctions.c.

```
472                     {
473   int check = 20;
474   int i;
475   for(i = 0; i < sizeof(ready queue);i++)   {
476     char Process_Name = ready queue [i] Process_Name;
477     char Class =  ready queue [i] class;
478     char State = ready queue[i] state;
479     char Status = ready queue[i] status;
480     char Priority = ready queue[i] priority;
481     sys_req(WRITE, COM1, Process_Name, &check);
482     sys_req(WRITE, COM1, Class, &check);
483     sys_req(WRITE, COM1, State, &check);
484     sys_req(WRITE, COM1, Status, &check);
485     sys_req(WRITE, COM1, Priority, &check);
486   }
487 }
```

5.22.1.17 Suspend()

```
void Suspend (
              Char * Process_Name )
```

Brief Description: Places a PCD in the suspended state and reinserts it into the appropriate queue.

Description: Can except a string as a pointer that is the Process Name. Places a PCB in the suspended state and reinserts it into the appropriate queue. An error check for valid Process Name.

Parameters

Process_Name	Character pointer that matches the name of process.

Definition at line 356 of file userFunctions.c.

```
356                                     {
357
358   // Name Error check
359   // Error check (Valid Name)
360   //if (Process_Name != valid name){
361   //  printf("\x1b[31m""\nERROR: Not a valid process name \n""\x1b[0m");
362   //}
363
364 }
```

5.22.1.18 toLowercase()

```
char toLowercase (
              char c )
```

Description: If a letter is uppercase, it changes it to lowercase.

(char)

Parameters

c	Character that is to be changed to its lowercase equivalent

Definition at line 262 of file userFunctions.c.

```
262                                     {
263         if((c >= 65) && (c <= 90))  {
264                 c = c + 32;
265             }
266         return c;
267         }
```

5.22.1.19 Version()

```
void Version ( )
```

Description: Simply returns a char containing "Version: R(module).

(the iteration that module is currently on).

No parameters.

Definition at line 255 of file userFunctions.c.

```
255                     {
256         printf("Version: R2.0 \n");
257     }
```

5.23 mpx_core/modules/R1/userFunctions.h File Reference

Macros

- #define RED "\x1B[31m"
- #define GRN "\x1B[32m"
- #define YEL "\x1B[33m"
- #define BLU "\x1B[34m"
- #define MAG "\x1B[35m"
- #define CYN "\x1B[36m"
- #define WHT "\x1B[37m"
- #define RESET "\x1B[0m"

Functions

- void SetTime (int hours, int minutes, int seconds)

  Description: sets the time register to the new values that the user inputed, all values must be inputed as SetTime(↩ Hours, Minutes, Seconds).
- void GetTime ()

  Description: retrieve and return the time values for hours, minutes, and seconds form the clock register using inb(Port,address).
- int DectoBCD (int Decimal)

  Description: Changes decimal numbers to binary numbers.
- char * itoa (int num)

  Description: An integer is taken and seperated into individual chars and then all placed into a character array.
- void SetDate (int day, int month, int millennium, int year)

  Description: Sets the date register to the new values that the user inputed, all values must be inputed as SetDime(day, month, millenial, year).
- int BCDtoDec (int BCD)

Description: Changes binary number to decimal numbers.

- void GetDate ()

  Description: Returns the full date back to the user in decimal form.

- void Version ()

  Description: Simply returns a char containing "Version: R(module).

- void Help (char *request)

  Brief Description: Gives helpful information for one of the functions.

- void printf (char msg[])
- int EdgeCase (char *pointer)

  Description: Compares pointer char to validate if it is a number or not.

- char toLowercase (char c)

  Description: If a letter is uppercase, it changes it to lowercase.

- void Suspend (Char *Process_Name)

  Brief Description: Places a PCD in the suspended state and reinserts it into the appropriate queue.

- void Resume (Char *Process_Name)

  Brief Description: Places a PCD in the not suspended state and reinserts it into the appropriate queue.

- void Set_Priority (Char *Process_Name, int Priority)

  Brief Description: Sets PCB priority and reinserts the process into the correct place in the correct queue.

- void Show_PCB (Char *Process_Name)
- void Show_All ()

  Brief Description: Displays the process name, class, state, suspended status, and priority of all PCB in the ready and blocked queues.

- void Show_Ready ()

  Brief Description: Displays the process name, class, state, suspended status, and priority of all PCB in the ready queue.

- void Show_Blocked ()

  Brief Description: Displays the process name, class, state, suspended status, and priority of all PCB in the blocked queue.

- void Create_PCB (char *Process_Name, int Priority, int Class)
- void Delete_PCB (Char *Process_Name)
- void Block (Char *Process_Name)
- void Unblock (Char *Process_Name)

5.23.1 Macro Definition Documentation

5.23.1.1 BLU

```
#define BLU "\x1B[34m"
```

Definition at line 9 of file userFunctions.h.

5.23.1.2 CYN

```
#define CYN "\x1B[36m"
```

Definition at line 11 of file userFunctions.h.

5.23.1.3 GRN

```
#define GRN "\x1B[32m"
```

Definition at line 7 of file userFunctions.h.

5.23.1.4 MAG

```
#define MAG "\x1B[35m"
```

Definition at line 10 of file userFunctions.h.

5.23.1.5 RED

```
#define RED "\x1B[31m"
```

Definition at line 6 of file userFunctions.h.

5.23.1.6 RESET

```
#define RESET "\x1B[0m"
```

Definition at line 13 of file userFunctions.h.

5.23.1.7 WHT

```
#define WHT "\x1B[37m"
```

Definition at line 12 of file userFunctions.h.

5.23.1.8 YEL

```
#define YEL "\x1B[33m"
```

Definition at line 8 of file userFunctions.h.

5.23.2 Function Documentation

5.23.2.1 BCDtoDec()

```
int BCDtoDec (
            int BCD )
```

Description: Changes binary number to decimal numbers.

Parameters

value	Binary number to be changed to decimal

Definition at line 64 of file userFunctions.c.

```
64                              {
65          return (((BCD»4)*10) + (BCD & 0xF));
66      }
```

5.23.2.2 Block()

```
void Block (
              Char * Process_Name )
```

5.23.2.3 Create_PCB()

```
void Create_PCB (
              char * Process_Name,
              int Priority,
              int Class )
```

5.23.2.4 DectoBCD()

```
int DectoBCD (
              int Decimal )
```

Description: Changes decimal numbers to binary numbers.

Parameters

Decimal	Decimal number to be changed to binary

Definition at line 71 of file userFunctions.c.

```
71                              {
72          return (((Decimal/10) « 4) | (Decimal % 10));
73      }
```

5.23.2.5 Delete_PCB()

```
void Delete_PCB (
              Char * Process_Name )
```

5.23.2.6 EdgeCase()

```
int EdgeCase (
            char * pointer )
```

Description: Compares pointer char to validate if it is a number or not.

Parameters

Compares	pointer char to validate if it is a number or not.

Definition at line 83 of file userFunctions.c.

```
83                                      {
84      int valid = 0;
85      if (strcmp(pointer, "00") == 0){
86        valid = 1;
87        return valid;
88      }
89      int i, j;
90      for (i = 0; i < strlen(pointer); i++){
91          valid = 0;
92          for(j = 0; j <= 99; j++){
93              if(strcmp(pointer,itoa(j)) == 0)
94                  valid = 1;
95          }
96          if(valid == 0){
97            return valid;
98          }
99      }
100      return valid;
101    }
```

5.23.2.7 GetDate()

```
void GetDate ( )
```

Description: Returns the full date back to the user in decimal form.

No parameters.

Definition at line 225 of file userFunctions.c.

```
225                         {
226       int check = 2;
227       outb(0x70,0x07);
228         unsigned char day = BCDtoDec(inb(0x71));
229         outb(0x70,0x08);
230         unsigned char month = BCDtoDec(inb(0x71));
231         outb(0x70,0x32);
232         unsigned char millennium = BCDtoDec(inb(0x71));
233         char msg[2] = "-";
234         char msg3[10] = "Date: ";
235         printf(msg3);
236         sys_req(WRITE, COM1, itoa(day), &check);
237         printf(msg);
238         sys_req(WRITE, COM1, itoa(month), &check);
239         printf(msg);
240         sys_req(WRITE, COM1, itoa(millennium), &check);
241     outb(0x70,0x09);
242     if(BCDtoDec(inb(0x71)) == 0){
243       sys_req(WRITE, COM1, "00", &check);
244     }
245     else {
246         unsigned char year = BCDtoDec(inb(0x71));
247         sys_req(WRITE, COM1, itoa(year), &check);
248     }
249         printf("\n");
250     }
```

5.23.2.8 GetTime()

```
void GetTime ( )
```

Description: retrieve and return the time values for hours, minutes, and seconds form the clock register using inb(Port,address).

No parameters.

Definition at line 147 of file userFunctions.c.

```
147                     {
148     int check = 2;
149     int hour;
150     int minute;
151     int second;
152         outb(0x70,0x04);
153         unsigned char hours = inb(0x71);
154         outb(0x70,0x02);
155         unsigned char minutes = inb(0x71);
156         outb(0x70,0x00);
157         unsigned char seconds = inb(0x71);
158         char msg1[2] = ":";
159         char msg2[10] = "Time: ";
160         printf(msg2);
161         hour = BCDtoDec(hours);
162         sys_req(WRITE, COM1, itoa(hour), &check);
163         printf(msg1);
164         minute = BCDtoDec(minutes);
165         sys_req(WRITE, COM1, itoa(minute), &check);
166         printf(msg1);
167         second = BCDtoDec(seconds);
168         sys_req(WRITE, COM1, itoa(second), &check);
169       printf("\n");
170     }
```

5.23.2.9 Help()

```
void Help (
            char * request )
```

Brief Description: Gives helpful information for one of the functions.

Description: Can except a string as a pointer, if the pointer is null then the function will print a complete list of avaliable commands to the console. If the pointer is a avaliable commands then instructions on how to use the command will be printed. If the command does not exist then a message explaining that it is not a valid command will be displayed.

Parameters

request	Character pointer that matches the name of the function that you need help with.

Definition at line 274 of file userFunctions.c.

```
274                             {
275     int check = 1;
276         if (request[0] == '\0') {
277                 printf("\n to chain commands and parameters, please use \"-\" between keywords \n");
278             printf("\n getDate \n setDate \n getTime \n setTime \n version \n shutdown \n\n");
279         }
280         else if (strcmp(request, "GetDate") == 0) {
281                 printf("\n getDate returns the current date that is loaded onto the operating
       system.\n");
282         }
283         else if (strcmp(request, "SetDate") == 0) {
```

```
284                 printf("\n setDate allows the user to reset the correct date into the system, as follows
    setDate-"BLU"day"RESET"-"BLU"month"RESET"-"BLU"year"RESET".\n Time must be inputed as a two digit
    number, Example 02 or 00");
285         }
286         else if (strcmp(request, "GetTime") == 0) {
287                 printf("\n getTime returns the current time as hours, minutes, seconds that is loaded
    onto the operating system.\n");
288         }
289         else if (strcmp(request, "SetTime") == 0) {
290                 printf("\n setTime allows the user to reset the correct time into the system, as follows
    setTime-"BLU"hour"RESET"-"BLU"minute"RESET"-"BLU"second"RESET".\n Time must be inputed as a two digit
    number, Example 02 or 00");
291             }
292         else if (strcmp(request, "Version") == 0) {
293                 printf("\n version returns the current operating software version that the system is
    running.\n");
294         }
295      else if(strcmp(request, "shutdown") == 0) {
296        printf("\n shutdown shuts down the system.\n");
297      }
298
299 /************************R2 Commands*********************************************************/
300     else if(strcmp(request,"suspend") == 0) {
301     printf("\n Suspend takes in the name of a PCB then places it into the suspended state and reinserts
    it into the correct queue.\n");
302     }
303     else if(strcmp(FirstToken,"resume") == 0) {
304     printf("\n Resume takes in the name of a PCB then removes it from the suspended state and adds it to
    the correct queue.\n");
305     }
306     else if(strcmp(FirstToken,"setPriority") == 0) {
307     printf("\n SetPriority takes in the name of a PCB and the priority it needs to be set to then
    reinstates the specified PCB into a new location by priority.\n");
308     }
309     else if(strcmp(FirstToken,"showPCB") == 0) {
310     printf("\n ShowPCB takes in the name of a PCB and returns all the associated attributes to the
    user.\n");
311     }
312     else if(strcmp(FirstToken,"showAll") == 0) {
313     printf("\n ShowAll takes no parameters but returns all PCB's that are currently in any of the
    queues.\n");
314     }
315     else if(strcmp(FirstToken,"showReady") == 0) {
316     printf("\n ShowReady takes in no parameters but returns all PCB's and there attributes that
    currently are in the ready state.\n");
317     }
318     else if(strcmp(FirstToken,"showBlocked") == 0) {
319     printf("\n ShowBlocked takes in no parameters but returns all PCB's and there attributes that
    currently are in the blocked state.\n");
320     }
321
322 /******************************** R2 Temp Commands
    *****************************************************/
323     else if(strcmp(FirstToken,"createPCB") == 0) {
324     printf("\n  CreatePCB takes in the process_name, process_class, and process_priority. Then assigns
    this new process into the correct queue.\n");
325     }
326     else if(strcmp(FirstToken,"deletePCB") == 0) {
327     printf("\n DeletePCB takes in the process_name then deletes it from the queue and free's all the
    memory that was previously allocated to the specified PCB.\n");
328     }
329     else if(strcmp(FirstToken,"block") == 0) {
330     printf("\n Block takes in the process_name then sets it's state to blocked and reinserts it back
    into the correct queue.\n");
331     }
332     else if(strcmp(FirstToken,"unblock") == 0) {
333     printf("\n Unblock takes in the process_name then sets it's state to ready and reinserts it back
    into the correct queue.\n");
334     }
335     else    {
336      printf("\x1b[31m""\nThe requested command does not exist please refer to the Help function for a
    full list of commands.\n""\x1b[0m");
337     }
338 }
```

5.23.2.10 itoa()

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

Description: An integer is taken and seperated into individual chars and then all placed into a character array.

Adapted from geeksforgeeks.org.

Parameters

num	integer to be put into array Title: itoa Author: Neha Mahajan Date: 29 May, 2017 Availability: https://www.geeksforgeeks.org/implement-itoa/

Definition at line 33 of file userFunctions.c.

```
34      {
35              int i,j,k,count;
36              i = num;
37              j = 0;
38              count = 0;
39          while(i){ // count number of digits
40              count++;
41              i /= 10;
42          }
43
44          char* arr1;
45          char arr2[count];
46          arr1 = (char*)sys_alloc_mem(count); //memory allocation
47
48          while(num){ // seperate last digit from number and add ASCII
49              arr2[++j] = num%10 + '0';
50              num /= 10;
51          }
52
53          for(k = 0; k < j; k++){ // reverse array results
54              arr1[k] = arr2[j-k];
55          }
56          arr1[k] = '\0';
57
58          return(char*)arr1;
59      }
```

5.23.2.11 printf()

```
void printf (
            char msg[] )
```

Definition at line 75 of file userFunctions.c.

```
75                              {
76        int check =strlen(msg);
77          sys_req(WRITE, COM1, msg, &check);
78      }
```

5.23.2.12 Resume()

```
void Resume (
            Char * Process_Name )
```

Brief Description: Places a PCD in the not suspended state and reinserts it into the appropriate queue.

Description: Can except a string as a pointer that is the Process Name. Places a PCB in the not suspended state and reinserts it into the appropriate queue. An error check for valid Process Name.

Parameters

Process_Name	Character pointer that matches the name of process.

Definition at line 371 of file userFunctions.c.

```
371                                 {
372
373
374   // Name Error check
375   // Error check (Valid Name)
376   //if (Process_Name != valid name){
377   //  printf("\x1b[31m""\nERROR: Not a valid process name \n""\x1b[0m");
378   //}
379
380 }
```

5.23.2.13 Set_Priority()

```
void Set_Priority (
            Char * Process_Name,
            int Priority )
```

Brief Description: Sets PCB priority and reinserts the process into the correct place in the correct queue.

Description: Can except a string as a pointer that is the Process Name. Can accept and integer than is the Priority. Sets a PCB's priority and reinserts the process into the correct place in the correct queue. An error check for valid Process Name and an error check for a valid priority 1 - 9.

Parameters

Process_Name	Character pointer that matches the name of process.
Priority	integer that matches the priority number.

Definition at line 388 of file userFunctions.c.

```
388                                                 {
389   int i;
390
391   // Name Error check
392   // Error check (Valid Name)
393   //if (Process_Name != valid name){
394   //  printf("\x1b[31m""\nERROR: Not a valid process name \n""\x1b[0m");
395   //}
396   // Priority error check
397   for(i = 0; i < 9; i++){
398     if(Priority == i){
399       break;
400     }
401     else{
402       printf("\x1b[31m""\nERROR: Not a valid Priority \n""\x1b[0m")
403     }
404   }
405
406 }
```

5.23.2.14 SetDate()

```
void SetDate (
            int day,
```

```
            int month,
            int millennium,
            int year )
```

Description: Sets the date register to the new values that the user inputed, all values must be inputed as SetDime(day, month, millenial, year).

Parameters

day	Integer to be set in the Day position
month	Integer to be set in the Month position
millenial	Integer to be set in the Millenial position
year	Integer to be set in the Year position

Definition at line 178 of file userFunctions.c.

```
178                                                     {
179     outb(0x70,0x07);
180     int tempDay = BCDtoDec(inb(0x71));
181     outb(0x70,0x08);
182     int tempMonth = BCDtoDec(inb(0x71));
183     outb(0x70,0x32);
184     int tempMillennium = BCDtoDec(inb(0x71));
185     outb(0x70,0x09);
186     int tempYear = BCDtoDec(inb(0x71));
187     cli();
188         outb(0x70,0x07);
189         outb(0x71,DectoBCD (day));
190         outb(0x70,0x08);
191         outb(0x71,DectoBCD (month));
192         outb(0x70,0x32);
193         outb(0x71,DectoBCD (millennium));
194         outb(0x70,0x09);
195         outb(0x71,DectoBCD (year));
196         sti();
197     outb(0x70,0x07);
198     unsigned char newDay = BCDtoDec(inb(0x71));
199     outb(0x70,0x08);
200     unsigned char newMonth = BCDtoDec(inb(0x71));
201     outb(0x70,0x32);
202     unsigned char newMillennium = BCDtoDec(inb(0x71));
203     outb(0x70,0x09);
204     unsigned char newYear = BCDtoDec(inb(0x71));
205     if(newDay != day || newMonth != month || newMillennium != millennium || newYear != year){
206       printf("Your input was invalid\n");
207       cli();
208         outb(0x70,0x07);
209         outb(0x71,DectoBCD (tempDay));
210         outb(0x70,0x08);
211         outb(0x71,DectoBCD (tempMonth));
212         outb(0x70,0x32);
213         outb(0x71,DectoBCD (tempMillennium));
214         outb(0x70,0x09);
215         outb(0x71,DectoBCD (tempYear));
216         sti();
217     }
218     else
219       printf("Date Set\n");
220     }
```

5.23.2.15 SetTime()

```
void SetTime (
            int hours,
            int minutes,
            int seconds )
```

Description: sets the time register to the new values that the user inputed, all values must be inputed as SetTime(Hours, Minutes, Seconds).

Parameters

hours	Integer to be set in the Hour position
minutes	Integer to be set in the Minutes position
seconds	Integer to be set in the Seconds position

Definition at line 108 of file userFunctions.c.

```
108                                                     {
109     outb(0x70,0x04);
110     unsigned char tempHours = BCDtoDec(inb(0x71));
111     outb(0x70,0x02);
112     unsigned char tempMinutes = BCDtoDec(inb(0x71));
113     outb(0x70,0x00);
114     unsigned char tempSeconds = BCDtoDec(inb(0x71));
115         cli(); //outb(device + 1, 0x00); //disable interrupts
116         outb(0x70,0x04);
117         outb(0x71, DectoBCD(hours));// change to bcd
118         outb(0x70,0x02);
119         outb(0x71, DectoBCD(minutes));
120         outb(0x70,0x00);
121         outb(0x71, DectoBCD(seconds));
122         sti();  //outb(device + 4, 0x0B); //enable interrupts, rts/dsr set
123     outb(0x70,0x04);
124     unsigned char newHours = BCDtoDec(inb(0x71));
125     outb(0x70,0x02);
126     unsigned char newMinutes = BCDtoDec(inb(0x71));
127     outb(0x70,0x00);
128     unsigned char newSeconds = BCDtoDec(inb(0x71));
129     if(newHours != hours || newMinutes != minutes || newSeconds != seconds){
130       printf("Your input was invalid\n");
131       cli(); //outb(device + 1, 0x00); //disable interrupts
132         outb(0x70,0x04);
133         outb(0x71, DectoBCD(tempHours));// change to bcd
134         outb(0x70,0x02);
135         outb(0x71, DectoBCD(tempMinutes));
136         outb(0x70,0x00);
137         outb(0x71, DectoBCD(tempSeconds));
138         sti();  //outb(device + 4, 0x0B); //enable interrupts, rts/dsr set
139     }
140     else
141       printf("Time Set\n");
142     }
```

5.23.2.16 Show_All()

```
void Show_All ( )
```

Brief Description: Displays the process name, class, state, suspended status, and priority of all PCB in the ready and blocked queues.

Description: The process name, claas, state, suspend status, and priority of each of he PCB's in the ready and blocked queues.

Definition at line 439 of file userFunctions.c.

```
439                 {
440   int check = 20;
441   int i;
442   int j;
443   for(i = 0; i < sizeof(ready queue);i++)   {
444     char rProcess_Name = ready queue [i] Process_Name;
445     int rClass =  ready queue [i] class;
446     char rState = ready queue[i] state;
447     char rStatus = ready queue[i] status;
448     int rPriority = ready queue[i] priority;
449     sys_req(WRITE, COM1, rProcess_Name, &check);
450     sys_req(WRITE, COM1, itoa(rClass), &check);
451     sys_req(WRITE, COM1, rState, &check);
452     sys_req(WRITE, COM1, rStatus, &check);
453     sys_req(WRITE, COM1, itoa(rPriority), &check);
454   }
```

```
455   for(j = 0; j < sizeof(blocked queue); j++){
456     char bProcess_Name = blocked queue [j] Process_Name;
457     int bClass =  blocked queue [j] class;
458     char bState = blocked queue[j] state;
459     char bStatus = blocked queue[j] status;
460     int bPriority = blocked queue[j] priority;
461     sys_req(WRITE, COM1, bProcess_Name, &check);
462     sys_req(WRITE, COM1, itoa(bClass), &check);
463     sys_req(WRITE, COM1, bState, &check);
464     sys_req(WRITE, COM1, bStatus, &check);
465     sys_req(WRITE, COM1, itoa(bPriority), &check);
466   }
467 }
```

5.23.2.17 Show_Blocked()

```
void Show_Blocked ( )
```

Brief Description: Displays the process name, class, state, suspended status, and priority of all PCB in the blocked queue.

Description: The process name, claas, state, suspend status, and priority of each of he PCB's in the blocked queue.
Brief Description: Calls SetupPCB() and inserts PCB into appropriate queue.

Description: Can except a string as a pointer that is the Process Name. Can accept two integers, Priority and Class. SetupPCB() will be called and the PCB will be inserted into the appropriate queue. An error check for unique and valid Process Name, an error check for valid process class, and an error check for process priority.

Parameters

Process_Name	Character pointer that matches the name of process.
Priority	integer that matches the priority number.
Class	integer that matches the class number.

Brief Description: Removes PCB from appropriate queue and frees all associated memory.

Description: Can except a string as a pointer that is the Process Name. Removes PCB from the appropriate queue and then frees all associated memory. An error check to make sure process name is valid.

Parameters

Process_Name	Character pointer that matches the name of process.

Brief Description: Places a PCD in the blocked state and reinserts it into the correct queue.

Description: Can except a string as a pointer that is the Process Name. The specified PCB will be places in a blocked state and reinserted into the appropriate queue. An error check for a valid name occurs.

Parameters

Process_Name	Character pointer that matches the name of process.

Brief Description: Places a PCD in the unblocked state and reinserts it into the correct queue.

Description: Can except a string as a pointer that is the Process Name. The specified PCB will be places in an unblocked state and reinserted into the appropriate queue. An error check for a valid name occurs.

Parameters

Process_Name	Character pointer that matches the name of process.

Definition at line 492 of file userFunctions.c.

5.23.2.18 Show_PCB()

```
void Show_PCB (
            Char * Process_Name )
```

5.23.2.19 Show_Ready()

```
void Show_Ready ( )
```

Brief Description: Displays the process name, class, state, suspended status, and priority of all PCB in the ready queue.

Description: The process name, claas, state, suspend status, and priority of each of he PCB's in the ready queue.

Definition at line 472 of file userFunctions.c.

```
472                     {
473   int check = 20;
474   int i;
475   for(i = 0; i < sizeof(ready queue);i++)   {
476     char Process_Name = ready queue [i] Process_Name;
477     char Class =  ready queue [i] class;
478     char State = ready queue[i] state;
479     char Status = ready queue[i] status;
480     char Priority = ready queue[i] priority;
481     sys_req(WRITE, COM1, Process_Name, &check);
482     sys_req(WRITE, COM1, Class, &check);
483     sys_req(WRITE, COM1, State, &check);
484     sys_req(WRITE, COM1, Status, &check);
485     sys_req(WRITE, COM1, Priority, &check);
486   }
487 }
```

5.23.2.20 Suspend()

```
void Suspend (
            Char * Process_Name )
```

Brief Description: Places a PCD in the suspended state and reinserts it into the appropriate queue.

Description: Can except a string as a pointer that is the Process Name. Places a PCB in the suspended state and reinserts it into the appropriate queue. An error check for valid Process Name.

Parameters

Process_Name	Character pointer that matches the name of process.

Definition at line 356 of file userFunctions.c.

```
356                                     {
357
358   // Name Error check
359   // Error check (Valid Name)
360   //if (Process_Name != valid name){
361   //  printf("\x1b[31m""\nERROR: Not a valid process name \n""\x1b[0m");
362   //}
363
364 }
```

5.23.2.21 toLowercase()

```
char toLowercase (
            char c )
```

Description: If a letter is uppercase, it changes it to lowercase.

(char)

Parameters

c	Character that is to be changed to its lowercase equivalent

Definition at line 262 of file userFunctions.c.

```
262                                     {
263         if((c >= 65) && (c <= 90))  {
264                 c = c + 32;
265             }
266         return c;
267         }
```

5.23.2.22 Unblock()

```
void Unblock (
            Char * Process_Name )
```

5.23.2.23 Version()

```
void Version ( )
```

Description: Simply returns a char containing "Version: R(module).

(the iteration that module is currently on).

No parameters.

Definition at line 255 of file userFunctions.c.

```
255                     {
256         printf("Version: R2.0 \n");
257     }
```

5.24 mpx_core/modules/R2/PCB.c File Reference

Classes

- struct Queue
- struct **ReadyQueue**
- struct PCB

Functions

- struct PCB AllocatePCB ()
- else return printf ("\nMemory cannot be released from the requested pcb->\n")
- PCB * SetupPCB (char[ ] Name, int Class, int Level)
- PCB * FindPCB (char[ ] Name)
- void InsertPCB (PCB **head)
- void RemovePCB ()

Variables

- ReadyQueue count = 0
- ReadyQueue head = NULL
- ReadyQueue tail = NULL
- unsigned char stack [1KMEM]
- unsigned char * stackTop
- struct PCB * prev
- struct PCB * next
- char Process_Name [10]
- int Process_Class
- int Priority
- int ReadyState
- int SuspendedState
- char FreePCB PCB

5.24.1 Function Documentation

5.24.1.1 AllocatePCB()

```
struct PCB AllocatePCB ( )
```

Definition at line 24 of file PCB.c.

```
39                      {
40     return sys_alloc_mem(sizeof(PCB));
41 }
```

5.24.1.2 FindPCB()

```
PCB* FindPCB (
            char[] Name )
```

Definition at line 65 of file PCB.c.

```
65                              {
66     while(temp != ReadyQueue->tail && temp != block->tail)  {
67         if(strcmp(stack->head->name,Name) == 0)
68             return &head
69         else{
70       printf("\n Error: There is no such PCB \n");
71             return NULL;
72     }
73   }
74 }
```

5.24.1.3 InsertPCB()

```
void InsertPCB (
            PCB ** head )
```

Definition at line 76 of file PCB.c.

```
76                              {
77     PCB*
78 }
```

5.24.1.4 printf()

```
else return printf (
            "\nMemory cannot be released from the requested pcb->\n"  )
```

5.24.1.5 RemovePCB()

```
void RemovePCB ( )
```

Definition at line 80 of file PCB.c.

```
80                      {
81     //if the Ready queue is empty->
82   if(Ready->count==0)   {
83           printf("Queue is Empty\n");
84           return;
85   }
86   //otherwise we can remove the specific pcb-> from the queue->
87   else  {
88     Ready->count--;
89     Q->front++;
90     if(Q->front==Q->capacity)   {
91       Q->front=0;
92     }
93   }
94 }
```

5.24.1.6 SetupPCB()

```
PCB* SetupPCB (
              char[] Name,
              int Class,
              int Level )
```

Definition at line 50 of file PCB.c.

```
50                                                  {
51          PCB* pcb-> = AllocatePCB();
52          pcb->stackTop = 1024 + pcb->stack;
53          memset(pcb->stack, 0, 1024);
54          pcb->prev = NULL;
55          pcb->next = NULL;
56          pcb->ReadyState = READY;
57          pcb->SuspendedState = NULL;
58          pcb->Priority = Level;
59          strcpy(pcb->Process_Name,Name);
60          pcb->Process_Class = Class;
61      return Name;
62  }
```

5.24.2 Variable Documentation

5.24.2.1 count

```
BlockedQueue count = 0
```

Definition at line 1 of file PCB.c.

5.24.2.2 head

```
BlockedQueue head = NULL
```

Definition at line 2 of file PCB.c.

5.24.2.3 next

```
struct PCB* next
```

Definition at line 43 of file PCB.c.

5.24.2.4 PCB

```
char FreePCB PCB
```

Initial value:

```
{
    if(sys_free_mem(PCB) != -1)
        return printf("\nMemory release successful \n")
```

Definition at line 43 of file PCB.c.

5.24.2.5 prev

```
struct PCB* prev
```

Definition at line 42 of file PCB.c.

5.24.2.6 Priority

```
int Priority
```

Definition at line 46 of file PCB.c.

5.24.2.7 Process_Class

```
int Process_Class
```

Definition at line 45 of file PCB.c.

5.24.2.8 Process_Name

```
char Process_Name[10]
```

Definition at line 44 of file PCB.c.

5.24.2.9 ReadyState

```
int ReadyState
```

Definition at line 47 of file PCB.c.

5.24.2.10 stack

```
unsigned char stack[1KMEM]
```

Definition at line 40 of file PCB.c.

5.24.2.11 stackTop

```
unsigned char* stackTop
```

Definition at line 41 of file PCB.c.

5.24.2.12 SuspendedState

```
int SuspendedState
```

Definition at line 48 of file PCB.c.

5.24.2.13 tail

```
BlockedQueue tail = NULL
```

Definition at line 3 of file PCB.c.

5.25 mpx_core/modules/R2/PCB.h File Reference

Classes

- struct struct

Functions

- Queue AllocatePCB ()
- PCB SetupPCB (char[ ] Name, int Class, int Level)
- PCB FindPCB (char[ ] Name)
- InsertPCBA ()
- RemovePCB ()

Variables

- char[10] Process_Name
- int Process_Class
- int Priority
- int State
- int Process_Stack
- int * PCB_Pointer
- char FreePCB * PCB

5.25.1 Function Documentation

5.25.1.1 AllocatePCB()

```
Queue AllocatePCB ( )
```

Definition at line 24 of file PCB.c.

```
39                      {
40     return sys_alloc_mem(sizeof(PCB));
41 }
```

5.25.1.2 FindPCB()

```
PCB FindPCB (
              char[] Name )
```

Definition at line 65 of file PCB.c.

```
65                              {
66     while(temp != ReadyQueue->tail && temp != block->tail)  {
67         if(strcmp(stack->head->name,Name) == 0)
68             return &head
69         else{
70       printf("\n Error: There is no such PCB \n");
71             return NULL;
72     }
73   }
74 }
```

5.25.1.3 InsertPCBA()

```
InsertPCBA ( )
```

5.25.1.4 RemovePCB()

```
RemovePCB ( )
```

Definition at line 80 of file PCB.c.

```
80                      {
81     //if the Ready queue is empty->
82   if(Ready->count==0)   {
83           printf("Queue is Empty\n");
84           return;
85   }
86   //otherwise we can remove the specific pcb-> from the queue->
87   else  {
88     Ready->count--;
89     Q->front++;
90     if(Q->front==Q->capacity)   {
91       Q->front=0;
92     }
93   }
94 }
```

5.25.1.5 SetupPCB()

```
PCB SetupPCB (
              char[] Name,
              int Class,
              int Level )
```

Definition at line 50 of file PCB.c.

```
50                                                          {
51         PCB* pcb-> = AllocatePCB();
52         pcb->stackTop = 1024 + pcb->stack;
53         memset(pcb->stack, 0, 1024);
54         pcb->prev = NULL;
55         pcb->next = NULL;
56         pcb->ReadyState = READY;
57         pcb->SuspendedState = NULL;
58         pcb->Priority = Level;
59         strcpy(pcb->Process_Name,Name);
60         pcb->Process_Class = Class;
61     return Name;
62 }
```

5.25.2 Variable Documentation

5.25.2.1 PCB

```
char FreePCB* PCB
```

Definition at line 32 of file PCB.h.

5.25.2.2 PCB_Pointer

```
int* PCB_Pointer
```

Definition at line 22 of file PCB.h.

5.25.2.3 Priority

```
int Priority
```

Definition at line 19 of file PCB.h.

5.25.2.4 Process_Class

```
int Process_Class
```

Definition at line 18 of file PCB.h.

5.25.2.5 Process_Name

```
char [10] Process_Name
```

Definition at line 17 of file PCB.h.

5.25.2.6 Process_Stack

```
int Process_Stack
```

Definition at line 21 of file PCB.h.

5.25.2.7 State

```
int State
```

Definition at line 20 of file PCB.h.

5.26 README.md File Reference