COVID-OS

Generated by Doxygen 1.9.0

Chapter 1

Class Index

1.1 Class List

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

Chapter 2

File Index

2.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 3

Class Documentation

3.1 Alarm Struct Reference

Alarm structure containing a message, hour, minute and second, as well as pointers to the next and previous alarms.

`#include <R4.h>`

Public Attributes

- char **msg** [50]
- int **hour**
- int **min**
- int **sec**
- struct Alarm * **next**
- struct Alarm * **prev**

3.1.1 Detailed Description

Alarm structure containing a message, hour, minute and second, as well as pointers to the next and previous alarms.

Definition at line 9 of file R4.h.

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

- mpx_core/modules/R3/R4.h

3.2 AlarmList Struct Reference

List that contains all alarms, with a head and tail pointer, and a counter.

`#include <R4.h>`

Public Attributes

- int **count**
- ala * **head**
- ala * **tail**

3.2.1 Detailed Description

List that contains all alarms, with a head and tail pointer, and a counter.

Definition at line 21 of file R4.h.

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

- mpx_core/modules/R3/R4.h

3.3 Block Struct Reference

Public Attributes

- int **size**
- cmcb * **head**
- cmcb * **tail**

3.3.1 Detailed Description

Definition at line 30 of file r5.h.

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

- mpx_core/modules/R5/r5.h

3.4 CMCB Struct Reference

Public Attributes

- int **type**
- int **size**
- int **memorySize**
- u32int **beginAddress**
- char * **name**
- struct CMCB * **next**
- struct CMCB * **prev**

3.4.1 Detailed Description

Definition at line 10 of file r5.h.

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

- mpx_core/modules/R5/r5.h

3.5 Context Struct Reference

Structure that contains the register values for context switching.

`#include <syscall.h>`

Public Attributes

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

3.5.1 Detailed Description

Structure that contains the register values for context switching.

Definition at line 12 of file syscall.h.

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

- mpx_core/modules/R3/syscall.h

3.6 DCB Struct Reference

Device Control Block struct.

`#include <R6.h>`

Public Attributes

- int **flag_status**
- int * **Event_Flag**
- int **Status**
- char * **In_Buffer**
- int * **In_Count**
- int **In_Position**
- char * **Out_Buffer**
- int * **Out_Count**
- int **Out_Position**
- char **Ring_Buffer** [300]
- int **input_index**
- int **output_index**
- int **count**

3.6.1 Detailed Description

Device Control Block struct.

Definition at line 68 of file R6.h.

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

- mpx_core/modules/R6/R6.h

3.7 gdt_descriptor_struct Struct Reference

Public Attributes

- u16int **limit**
- u32int **base**

3.7.1 Detailed Description

Definition at line 23 of file tables.h.

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

- mpx_core/include/core/tables.h

3.8 gdt_entry_struct Struct Reference

Public Attributes

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

3.8.1 Detailed Description

Definition at line 30 of file tables.h.

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

- mpx_core/include/core/tables.h

3.9 idt_entry_struct Struct Reference

Public Attributes

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

3.9.1 Detailed Description

Definition at line 6 of file tables.h.

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

- mpx_core/include/core/tables.h

3.10 idt_struct Struct Reference

Public Attributes

- u16int **limit**
- u32int **base**

3.10.1 Detailed Description

Definition at line 16 of file tables.h.

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

- mpx_core/include/core/tables.h

3.11 IOCB Struct Reference

Stores an event flag and count, with pointers to the IOD head, tail, and current.

```
#include <syscall.h>
```

Public Attributes

- int **event_flag**
- int **iocb_count**
- struct IOD * **head**
- struct IOD * **tail**
- struct IOD * **current**

3.11.1 Detailed Description

Stores an event flag and count, with pointers to the IOD head, tail, and current.

Definition at line 32 of file syscall.h.

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

- mpx_core/modules/R3/syscall.h

3.12 IOD Struct Reference

Stores a record of the device data transfer and its status.

```
#include <syscall.h>
```

Public Attributes

- char **name** [12]
- int **Op_Code**
- struct PCB * **pcb_id**
- char * **iod_buffer**
- int * **iod_count**
- struct IOD * **next**
- struct IOD * **prev**

3.12.1 Detailed Description

Stores a record of the device data transfer and its status.

Definition at line 20 of file syscall.h.

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

- mpx_core/modules/R3/syscall.h

3.13 LMCB Struct Reference

Public Attributes

- int **type**
- int **size**
- int **memorySize**

3.13.1 Detailed Description

Definition at line 22 of file r5.h.

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

- mpx_core/modules/R5/r5.h

3.14 param Struct Reference

To simulate system calls to request services from the kernel.

`#include <mpx_supt.h>`

Public Attributes

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

3.14.1 Detailed Description

To simulate system calls to request services from the kernel.

Definition at line 37 of file mpx_supt.h.

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

- mpx_core/modules/mpx_supt.h

3.15 PCB Struct Reference

PCB structure with a process name, class, priority, state and suspended value.

`#include <pcb.h>`

Public Attributes

- char **Process_Name** [12]
- int **Process_Class**
- int **Priority**
- int **State**
- int **Suspended**
- stack **Stack**
- struct PCB * **next**
- struct PCB * **prev**

3.15.1 Detailed Description

PCB structure with a process name, class, priority, state and suspended value.

Definition at line 24 of file pcb.h.

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

- mpx_core/modules/R2/pcb.h

3.16 Queue Struct Reference

Queue structure with a pointer to it's head and tail, as well as a counter.

`#include <pcb.h>`

Public Attributes

- int **count**
- pcb * **head**
- pcb * **tail**

3.16.1 Detailed Description

Queue structure with a pointer to it's head and tail, as well as a counter.

Definition at line 41 of file pcb.h.

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

- mpx_core/modules/R2/pcb.h

3.17 Stack Struct Reference

Stack structure with a pointer to it's top and base.

`#include <pcb.h>`

Public Attributes

- unsigned char * **stack_top**
- unsigned char * **stack_base**

3.17.1 Detailed Description

Stack structure with a pointer to it's top and base.

Definition at line 18 of file pcb.h.

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

- mpx_core/modules/R2/pcb.h

Chapter 4

File Documentation

4.1 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)

  Gathers keyboard input and stores it in the buffer.

4.2 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/R2/pcb.h"
#include "../modules/R2/user_commands.h"
#include "../modules/R5/r5.h"
#include "../modules/R6/R6.h"
#include "../modules/R3/syscall.h"
```

Functions

- void **kmain** (void)

Variables

- int **eventFlag** = 1

4.3 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 *buffer, int *count)

  Gathers keyboard input and stores it in the buffer.

Variables

- int **serial_port_out** = 0
- int **serial_port_in** = 0

4.4 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 *)

4.5 mpx_core/modules/mpx_supt.h File Reference

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

Classes

- struct param

  To simulate system calls to request services from the kernel.

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** ()
- void comhand ()

  Processes user input and delivers appropriate output.

4.6 mpx_core/modules/R1/comhand.h File Reference

```
#include "date.h"
#include "../modules/mpx_supt.h"
#include <system.h>
#include "../modules/R2/pcb.h"
#include "../modules/R3/syscall.h"
#include "../modules/R3/procsr3.h"
#include <string.h>
#include <core/serial.h>
#include <core/io.h>
#include "../modules/R5/r5.h"
#include <stdint.h>
```

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 **INVALID_COMMAND** 2001
- #define **DEFAULT_DEVICE** 111
- #define **COM_PORT** 222

Functions

- void comhand ()

  Processes user input and delivers appropriate output.
- void print_startup ()

  Prints the startup menu.
- void version ()

  Prints the current OS version.
- void exit ()

  Exits the OS after a confirmation.
- void help ()

  Prints a description of each command.
- void clear_screen ()

  Clears the current screen.
- void print_error (int err)

  Prints the corresponding error message.

4.7 mpx_core/modules/R1/date.h File Reference

Functions

- void getDate ()

  Display's the date (MM/DD/YYYY)
- void getTime ()

  Display's the time (HH:MM:SS)
- int **Time** ()
- void setDate ()

  Prompt's user for the date (MM/DD/YYYY)
- void setTime ()

  Prompt's user for the time (HH:MM:SS)
- void itoa (int num, char *str)

  Converts an integer to a string.
- unsigned int decToBcd (int temp)

  Converts decimal value to binary coded decimal value.

4.8 mpx_core/modules/R2/pcb.h File Reference

```
#include <system.h>
#include "../modules/R3/syscall.h"
```

Classes

- struct Stack

  Stack structure with a pointer to it's top and base.
- struct PCB

  PCB structure with a process name, class, priority, state and suspended value.
- struct Queue

  Queue structure with a pointer to it's head and tail, as well as a counter.

Macros

- #define **READY** 0
- #define **RUNNING** 1
- #define **BLOCKED** 2
- #define **NOT_SUSPENDED** 0
- #define **SUSPENDED** 1
- #define **STACK_SIZE** 1024

Typedefs

- typedef struct Stack stack

  Stack structure with a pointer to it's top and base.
- typedef struct PCB pcb

  PCB structure with a process name, class, priority, state and suspended value.
- typedef struct Queue queue

  Queue structure with a pointer to it's head and tail, as well as a counter.

Functions

- void freePCB ()

  Allocates memory for the PCB and initializes stack to null. Returns pointer to new PCB if successful, NULL otherwise.
- void enQueue ()

  adds item to the front of the queue.
- pcb * deQueue ()

  removes item from the front of the queue
- pcb * findPCB ()

  finds pcb from approriate queue. returns pcb pointer if sucessfull else returns NULL
- pcb * findInQueue ()

  finds pcb from given queue. returns pcb pointer if sucessfull else returns NULL
- pcb * allocatePCB ()

  Allocates memory for the PCB and initializes stack to null. Returns pointer to new PCB if successful, NULL otherwise.
- pcb * setupPCB ()

  Calls allocatePCB() and declares new PCB with given parameters. Returns pcb pointer if successful, NULL otherwise or invalid parameters.
- void insertPCB ()

  Adds PCB to correct queue based block.state. Takes pointer to pcb as parameter.
- void createPCB ()

  user commands
- int removePCB ()

  removes PCB from appropriate queue. takes queue and pcb pointeras parameter.returns "success" or "failure" .
- void show_pcb ()

  shows all the information contained by the pcb. Name,class,status,state, priority
- void show_ready ()

  shows all the pcb present in the ready queue
- void show_blocked ()

  shows all the pcb present in the blocked queue
- void show_susRed ()

  shows all the pcb present in the suspended_ready queue
- void show_all ()

shows all the pcb present in the ready queue and blocked queue

- int removeFromQueue ()

  removes PCB from given queue.takes queue and pcb pointeras parameter. returns "success" or "failure" .
- void deletePCB ()

  removes pcb from appropritate queue and frees associated memory
- void show_Queue ()

  shows all the items present in a particular queue
- void set_pri (char *name, int priority)

  sets the priority of a pcb.
- void blockPCB ()

  block the pcb
- void unblockPCB ()

  unblock the pcb
- void suspendPCB ()

  suspend the current pcb
- void resumePCB ()

  Resume the current pcb.

Variables

- int **bufferSize**
- queue **ready**
- queue **suspended_ready**
- queue **blocked**
- queue **suspended_blocked**

4.9 mpx_core/modules/R2/tempCommands.c File Reference

Functions

- void createPCB (char *name, int class, int priority)

  creates a new PCB by calling setup PCB and inserts into appropriate queue -> insertPCB

4.10 mpx_core/modules/R3/procsr3.h File Reference

Functions

- void proc1 ()

  initiate proc1
- void proc2 ()

  initiate proc2
- void proc3 ()

  initiate proc3
- void proc4 ()

  initiate proc4
- void proc5 ()

  initiate proc5

4.11 mpx_core/modules/R3/R4.h File Reference

Classes

- struct Alarm

  Alarm structure containing a message, hour, minute and second, as well as pointers to the next and previous alarms.
- struct AlarmList

  List that contains all alarms, with a head and tail pointer, and a counter.

Typedefs

- typedef struct Alarm ala

  Alarm structure containing a message, hour, minute and second, as well as pointers to the next and previous alarms.
- typedef struct AlarmList alist

  List that contains all alarms, with a head and tail pointer, and a counter.

Functions

- void checkAlarm ()

  Check's status of any current alarms.
- int checkTime ()

  Check's remaining time of alarm.
- void addAla ()

  Adds an alarm to the system.
- void createAlarm ()

  Creates an alarm for the system.
- void removeAla ()

  Removes the current alarm.

Variables

- alist **list**

4.12 mpx_core/modules/R3/syscall.h File Reference

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

Classes

- struct Context

  Structure that contains the register values for context switching.
- struct IOD

  Stores a record of the device data transfer and its status.
- struct IOCB

  Stores an event flag and count, with pointers to the IOD head, tail, and current.

Typedefs

- typedef struct Context context

  Structure that contains the register values for context switching.
- typedef struct IOD iod

  Stores a record of the device data transfer and its status.
- typedef struct IOCB iocb

  Stores an event flag and count, with pointers to the IOD head, tail, and current.

Functions

- u32int * sys_call (context *registers)

  interrupt that calls the context registers
- void infinity ()

  creates the infinite process
- void run_infinite ()

  Runs the infinite process.
- void io_scheduler ()
- void **enQ** (iocb *Queue, iod *temp)
- iod * **deQ** (iocb *Queue)
- iod * **createIOD** ()

Variables

- iocb * **active**
- iocb * **complete**

4.12.1 Function Documentation

4.12.1.1 io_scheduler()

```
void io_scheduler ( )
```

+* io_scheduler() creates an io device for the PCB requesting IO. +*

Parameters

(the	params you give it depends on the design of your system)

Definition at line 126 of file syscall.c.

```
126                     {
127     // Check if there are any active or completed IO processes on the dcb.
128     // If completed,
129     // unblock the corresponding PCB and remove it from queue
130     // call com_read() or com_write() on the next iod depending on the op code.
131     //iocb* device;
132         iod *temp = createIOD(); //creat IOD
133
```

```
134         if (temp== NULL){
135             return;
136         }
137         else {
138             if ( params_ptr->device_id == DEFAULT_DEVICE){
139                     enQ(active, temp);
140                         if(params_ptr->op_code == READ){
141                             serial_io();
142                         }
143                         else if(params_ptr->op_code == WRITE){
144                             serial_io();
145 
146                         }
147 
148             }
149             else{
150                 enQ(complete, temp);
151                 if(params_ptr->op_code == READ){
152                     serial_io();
153 
154 
155                 }
156                 else if(params_ptr->op_code == WRITE){
157                     serial_io();
158 
159                 }
160 
161             }
162 
163         }
164 }
```

4.13 mpx_core/modules/R5/r5.h File Reference

```
#include <system.h>
#include "../modules/mpx_supt.h"
```

Classes

- struct CMCB
- struct LMCB
- struct Block

Typedefs

- typedef struct CMCB **cmcb**
- typedef struct LMCB **lmcb**
- typedef struct Block **Block**

Functions

- void init_heap ()

  Initializes the heap.
- cmcb * makeMemBlock ()

  Creates a memory block.
- u32int allocate (u32int size)

  Allocates memeory.
- void show_list ()

  Shows list of memory.

- void show_free ()

  Shows the free memory.
- void show_allocated ()

  Shows the allocated memory.
- int Free_mem ()

  Frees the memory.
- int IsEmpty ()

  Boolean for if the memory is heap is empty.

4.14 mpx_core/modules/R6/R6.h File Reference

```
#include <system.h>
#include "../modules/mpx_supt.h"
```

Classes

- struct DCB

  Device Control Block struct.

Macros

- #define **BASE** 0x3F8
- #define **BUFFERS** BASE + 0
- #define **BAUD_RATE_LSB** BASE +0
- #define **INTERRUPT_ENABLE** BASE + 1
- #define **BAUD_RATE_MSB** BASE + 1
- #define **INTERRUPT_ID** BASE + 2
- #define **LINE_CONTROL** BASE + 3
- #define **MODEM_CONTROL** BASE + 4
- #define **LINE_STATUS** BASE + 5
- #define **MODEM_STATUS** BASE + 6
- #define **INTVECT** 0x0C
- #define **RING_SIZE** 100
- #define **PIC_MASK** 0x21
- #define **PIC_REG** 0x20
- #define **PIC_EOI** 0x20
- #define **IRQ_COM1** 0x10
- #define **OPEN_SUCCESS** 0
- #define **OPEN_NULL** -101
- #define **OPEN_INV_BAUD** -102
- #define **ALREADY_OPEN** -103
- #define **CLOSE_SUCCESS** 0
- #define **CLOSE_NOT_OPEN** -201
- #define **WRITE_SUCCESS** 0
- #define **WRITE_NOT_OPEN** -401
- #define **WRITE_INV_BUFF** -402
- #define **WRITE_INV_COUNT** -403
- #define **WRITE_DEV_BUSY** -404
- #define **READ_SUCCESS** 0

- #define **READ_NOT_OPEN** -301
- #define **READ_INV_BUFF** -302
- #define **READ_INV_COUNT** -303
- #define **READ_DEV_BUSY** -304
- #define **WRITING** 5555
- #define **READING** 4444
- #define **WAITING** 6666
- #define **OPEN** 0
- #define **CLOSED** 1
- #define **Status_OPEN** 3

Typedefs

- typedef struct DCB dcb

  Device Control Block struct.

Functions

- void pic_mask (char enable)
- void disable_interrupts ()
- void enable_interrupts ()
- int com_open (int *e_flag, int baud_rate)
- int com_close (void)
- int com_read (char *buf_ptr, int *count_ptr)
- int com_write (char *buf_ptr, int *count_ptr)
- void serial_io ()
- void serial_write ()
- void serial_read ()

4.14.1 Function Documentation

4.14.1.1 com_close()

```
int com_close (
            void  )
```

+* com_close() Closes the communication port. +*

Returns

error code if port was not open, or a 0 for successful operation

Definition at line 85 of file R6.c.

```
85                  {
86   int Return_Val = CLOSE_SUCCESS;
87
88   if (!port->flag_status) {
89         Return_Val = CLOSE_NOT_OPEN;
90     }
91   else {
92     port->flag_status = FALSE;
93     // Reset Communication Protocol to original settings
94     outb(LINE_CONTROL, 0x00);         // Set DLAB = 0
95     outb(MODEM_CONTROL, 0x00);        // Disable Serial Interrupts
96            // Turn off interrupts
97     disable_interrupts();
98     outb(PIC_MASK, (inb(PIC_MASK) & 0x10)); // Set Programmable Interrupt Controller
99     enable_interrupts();
100     outb(INTVECT, old); // Restore Old Interrupt Vector
101   }
102   return Return_Val;
103 }
```

4.14.1.2 com_open()

```
int com_open (
              int * e_flag,
              int baud_rate )
```

+∗ com_open() Opens the communication port. +∗

Parameters

e_flag	event flag will be set to 1 if read/write +∗
baud_rate	the desired baud rate +∗

Returns

Returns three possible error codes, or a 0 for successful operation.

Definition at line 39 of file R6.c.

```
39                                          {
40   int Baud_Rate_Div;
41   int Return_Val = OPEN_SUCCESS;
42
43   if(port->flag_status){
44         Return_Val = ALREADY_OPEN;
45     }
46   else if (eflag_p == NULL || eflag_p == 0){
47         Return_Val = OPEN_NULL;
48     }
49
50   else if (baud_rate < 0){
51         Return_Val = OPEN_INV_BAUD;
52     }
53   else {
54     // Initialize port DCB
55     port->flag_status = TRUE;
56     port->Event_Flag = eflag_p;
57     port->Status = IDLE;
58     port->input_index = 0;
59     port->output_index = 0;
60     port->count = 0;
61
62     // Set up New Interrupts
63     outb(LINE_CONTROL,  0x00);             // Set DLAB = 0
64     disable_interrupts();             // Turn off interrupts
65
66     old = (inb(INTVECT));     // Save Old Interrupt
67     outb(INTVECT, &serial_io);          // Set New Interrupt
68
69     Baud_Rate_Div = 115200 / (long) baud_rate;
70
71     outb(LINE_CONTROL,  0x80);             // Set DLAB = 1
72     outb(BAUD_RATE_LSB, Baud_Rate_Div & 0xFF);    // Set Baud rate - Divisor Latch Low Byte
73     outb(BAUD_RATE_MSB, (Baud_Rate_Div » 8) & 0xFF); // Set Baud rate - Divisor Latch High Byte
74     outb(LINE_CONTROL,  0x03);             // 8 Bits, No Parity, 1 Stop Bit
75     disable_interrupts();
76     outb(PIC_MASK, (inb(PIC_MASK) & ~0x80));// Set Programmable Interrupt Controller
77     enable_interrupts();
78     outb(MODEM_CONTROL, 0x08);             // Enable Serial Interrupts
79     outb(INTERRUPT_ENABLE,     0x01);             // Enable Input Ready Interrupts
80   }
81
82   return Return_Val;
83 }
```

4.14.1.3 com_read()

```
int com_read (
              char * buf_ptr,
              int * count_ptr )
```

+∗ com_read() Reads the buffer from the port. Non-blocking. +∗

Parameters

buf_ptr	buffer in which the read characters will be stored. +∗
count_ptr	the maximum number of bytes to read. After completion, +∗ this will contain the number of characters read. +∗

Returns

Returns four possible error codes, or a 0 for successful operation.

Definition at line 140 of file R6.c.

```
140                                     {
141   int Return_Value = READ_SUCCESS;
142
143   if(port->flag_status != OPEN){
144         Return_Value = READ_NOT_OPEN;
145     }
146   else if (port->Status != IDLE) {
147         Return_Value = READ_DEV_BUSY;
148     }
149   else if (buf_p == NULL)           {
150         Return_Value = READ_INV_BUFF;
151     }
152   else if (count_p == NULL)       {
153         Return_Value = READ_INV_COUNT;
154     }
155
156   port->In_Buffer = buf_p;
157   port->In_Count = count_p;
158   port->In_Position = 0;
159   *port->Event_Flag = 0;
160   disable_interrupts();
161   port->Status = READING;
162
163   while(port->count > 0){
164     if(*port->In_Count == port->In_Position){
165       port->Status = IDLE;
166       break;
167     }
168
169     if(port->Ring_Buffer[port->output_index] == '\r'){
170       port->Status = IDLE;
171       port->output_index++;
172       if(port->output_index > (RING_SIZE - 1)){
173         port->output_index = 0;
174       }
175       break;
176     }
177     port->In_Buffer[port->In_Position] = port->Ring_Buffer[port->output_index];
178     port->In_Position++;
179     port->count--;
180     port->output_index++;
181
182     if(port->output_index > (RING_SIZE - 1)){
183       port->output_index = 0;
184     }
185   }
186   enable_interrupts();
187
188   if(port->Status == IDLE){
189     port->In_Buffer[port->In_Position]  = '\0';
190     *port->In_Count = port->In_Position;
191     *port->Event_Flag = 1;
192   }
193   return Return_Value;
194 }
```

4.14.1.4 com_write()

```
int com_write (
            char * buf_ptr,
            int * count_ptr )
```

+∗ com_write() Writes the buffer to the port. Non-blocking. +∗

Parameters

buf_ptr	buffer in which the characters to write are stored. +∗
count_ptr	the number of characters from the buffer to write. +∗

Returns

Returns four possible error codes, or a 0 for successful operation.

Definition at line 107 of file R6.c.

```
107                                                 {
108   int Return_Value = WRITE_SUCCESS;
109
110   if      (port->flag_status != OPEN){
111         Return_Value = WRITE_NOT_OPEN;
112     }
113   else if (port->Status != IDLE) {
114         Return_Value = WRITE_DEV_BUSY;
115     }
116   else if (buf_p == 0)           {
117         Return_Value = WRITE_INV_BUFF;
118     }
119   else if (count_p == 0)        {
120         Return_Value = WRITE_INV_COUNT;
121     }
122   else {
123     port->Out_Buffer = buf_p;
124     port->Out_Count = count_p;
125     port->Out_Position = 0;
126     port->Status = WRITING;
127     *port->Event_Flag = 0;
128
129     // Write first char to com port
130     outb(BASE, *port->Out_Buffer);
131     port->Out_Buffer++;
132     port->Out_Position++;
133
134       outb(INTERRUPT_ENABLE, (inb(INTERRUPT_ENABLE) | 0x02));
135   }
136   return Return_Value;
137 }
```

4.14.1.5 disable_interrupts()

```
void disable_interrupts ( )
```

+∗ disable_interrupts() disables all interrupts to device.

Definition at line 21 of file R6.c.

```
21                            {
22   outb(INTERRUPT_ENABLE, 0x00); //disable interrupts
23 }
```

4.14.1.6 enable_interrupts()

```
void enable_interrupts ( )
```

+∗ enable_interrupts() enables interrupts to device.

Definition at line 25 of file R6.c.

```
25                           {
26     outb(COM1 + 4, 0x0B); //enable interrupts, rts/dsr set
27     mask = inb(INTERRUPT_ENABLE);
28     mask = mask & ~0x02;
29     outb(INTERRUPT_ENABLE, mask);
30
31 }
```

4.14.1.7 pic_mask()

```
void pic_mask (
            char enable )
```

+∗ struct iod represents an I/O Desriptor. +∗

Parameters

pcb_id	the process that this iod is representing. +∗
op_code	the operation that the process requested. +∗
com_port	the COM port. (You can omit this and just always use COM1) +∗
buffer_ptr	the buffer pointer to read to/write from. +∗
count_ptr	the amount of characters to be read/written. +∗
next	the next IOD in the IO queue after this one.

+∗ pic_mask() masks so only the desired PIC interrupt is enabled or disabled. +∗

Parameters

enable	1 to enable or 0 to disable.

4.14.1.8 serial_io()

```
void serial_io ( )
```

+∗ serial_io() is the interrupt C routine for serial IO.

Definition at line 200 of file R6.c.

```
200                  {
201     static int type;
202     if(port->flag_status == OPEN){
203         type = inb(INTVECT);
204         type = type & 0x07;
205         if(type == READ)
206                 serial_read();//1200 BAUD TO BE USED
207         else if (type == WRITE)
208             serial_write();
209     }
210     outb(PIC_REG, PIC_EOI);
211 }
```

4.14.1.9 serial_read()

```
void serial_read ( )
```

+∗ serial_read() provides interrupt routine for reading IO.

Definition at line 241 of file R6.c.

```
241                   {
242     unsigned char input = inb(BASE);
243     if(port->Status == READING){
```

```
244         if(input != '\r'){
245             port->In_Buffer[port->In_Position] = input;
246             port->In_Position++;
247         }
248         if(input == '\r' || port->In_Position == *(port->In_Count)){
249                 port->In_Buffer[port->In_Position] = '\0';
250                 port->Status = IDLE;
251                 *port->Event_Flag = 1;
252                 *port->In_Count = port->In_Position;
253         }
254     }
255     else{
256         if(port->count < RING_SIZE){
257             port->Ring_Buffer[port->input_index] = input;
258             port->input_index++;
259             port->count++;
260             if(port->input_index > (RING_SIZE - 1)){
261                 port->input_index = 0;
262             }
263         }
264     }
265 }
```

4.14.1.10 serial_write()

```
void serial_write ( )
```

+* serial_write() provides interrupt routine for writing IO.

Definition at line 217 of file R6.c.

```
217                     {
218     static char mask;
219     if(port->Status == WRITING){
220         if(port->Out_Position != *(port->Out_Count)){
221             io = port->Out_Buffer[port->Out_Position];
222             port->Out_Position++;
223             outb(BASE, io);
224         }
225         else{
226             port->Status = IDLE;
227             *port->Event_Flag = 1;
228             mask = inb(INTERRUPT_ENABLE);
229             mask = mask & ~0x02;
230             outb(INTERRUPT_ENABLE, mask);
231 
232         }
233     }
234 }
```

Index