5x5 Operating system

Programmers Manual

File term_util.c/.h

Function Description: Check if a string contains only numbers which can be converted into an integer.
Parameter: const char* string -> An input string
Parameter: size_t size -> The size of the input string.
Returns: 0 if the string cannot be parsed, 1 if the string can be parsed into an integer.
char intParsable(const char* string, size_t size)

Enum Description: Lists some color strings for sending over terminal to change the terminal text color for further output.
Reset
Yellow
White
Red
Purple
Blue
enum Color

Function Description: Takes a color and sets the terminal color accordingly.
Parameter: enum Color -> Color to set the next terminal output to.
void setTerminalColor(enum Color color)

File comhand.c/.h

Function Description: Check if a string is hexadecimal formatted and able to be converted into an integer.
Parameter: string -> An input string
Parameter: size -> The size of the input string.
Returns: 0 if the string cannot be parsed, 1 if the string can be parsed into an integer.
char hexParsable(const char* string, size_t size)

Function Description: Translates an integer to a null-terminated hexadecimal string.
Parameter: string -> A user provided string buffer to output the representation of the input integer.
Parameter: int integer -> An integer to translate into a hexadecimal string.
void addressToHex(char string[], int integer)

Function Description: Translates a null-terminated string to a hexadecimal address/integer.
Parameter: s -> A string to convert to an address/integer.
Returns: An address/integer.
void* hexToAddress(const char* s)

Function Description: Takes a PCB class as parameter and then returns the corresponding PCB class string for output.
Parameter: cls -> A process class.
Returns: A human readable string representing the input class.
const char* class_str(enum ProClass cls)

Function Description: Takes PCB state as parameter and then returns the corresponding PCB execution state string for output.
Parameter: state -> A process execution state.
Returns: A human readable string representing the input execution state.
const char* execstate_str(enum ProState state)

Function Description: Takes pcb dispatch state as parameter and then returns the corresponding PCB dispatch state string for output.
Parameter: state -> A process dispatch state.
Returns: A human readable string representing the input dispatch state.
const char* dispatchstate_str(enum ProState state)

Struct Description: Map states or classes of PCBs to strings for user input or output.
Field: class -> An enum value that can be a state or class.
Field: str -> A common string to compare to user input for the associated state or class.
Field: str_out -> A common string for outputting the associated state or class as a human readable string.
struct str_pcbprop_map

Variable Description: Contains process class values and associated strings for output and input.
avail_pcb_class[0] = System
avail_pcb_class[1] = User
const struct str_pcbprop_map avail_pcb_class[]

Variable Description: Contains pairs of process execution state values and associated strings for output and input.
avail_pcb_execstate[0] = Blocked
avail_pcb_execstate[1] = Ready
avail_pcb_execstate[2] = Running
const struct str_pcbprop_map avail_pcb_execstate[]

Variable Description: Contains pairs of process dispatch state values and associated strings for output and input.
avail_pcb_dpatchstate[0] = Suspended
avail_pcb_dpatchstate[1] = Active
const struct str_pcbprop_map avail_pcb_dpatchstate[]

Variable Description: Common global buffer for storing input from the user after a read.
char user_input[128]

Variable Description: Stores the length of user_input after a read.
int user_input_len

Function Description: Enters the prompt for user input and then stores input text in user_input.
void user_input_promptread()

Function Description: Manually clears the global user_input buffer via memset.
void user_input_clear()

Function Description: Shows the main menu and enters the command loop, gets the user input as a number and then compares it to which method number and then calls the method.
void comhand()

Commands

Function Description: Ask the user to give the time (hours, minutes, seconds) to then call "setTime()".
Parameter: *Hour* -> Hour of the day, must be 0-23.
Parameter: *Minute* -> Minute of the hour, must be 0-59.
Parameter: *Second* -> Second of the minute. Must be 0-59.
Returns: 0 always
int setTimeCommand()

Function Description: Gets the time from getTime() method.
Returns: 0 always.
int getTimeCommand()

Function Description: Asks the user to give the date (month, day, year) to then call "setDate()" with the inputs.
Parameter: *Month* -> Month of the year, must be from 1-12.
Parameter: *Day* -> Day of the month, must be within the span of the month.
Parameter: *Year* -> Two digits of a year in the 21st century, must be from 0-99.
Returns: 0 always.

int setDateCommand()

Function Description: Prints the date in a human readable format.
Returns: 0 always.

int getDateCommand()

Function Description: Prints the version of the command with the date, the date comes from the compiler.
Returns: 0 always.

int versionCommand()

Function Description: Uses the numbers from the menu to get the help that the user wants, or simply typing all inside help to see help for all of the methods. Moreover, the user can type the command name, then if he presses enter it will take him to the main menu.
Returns: 0 always.

int helpCommand()

Function Description: Starts a prompt to turn the machine off, confirmed by sending a message to the user to enter 1.
Returns: 1 if the shut down was canceled.

int shutdownCommand()

Function Description: Command. Asks the user for the name of a process and the priority to set on the process with the provided name. If the user inputs invalid creation information, it will prompt the user again for each parameter.
Parameter: *Process name* -> The name of the new process. Must be unique among the names of other existing processes.
Parameter: *Process class* -> The class of the new process. Must be either 'user' or 'kernel' for user or kernel classes accordingly.
Parameter: *New process priority* -> The priority to set the new process to. Must be a number from 0 - 9.
Returns: 0 on success, -1 if a PCB could not be created due to allocation errors.

int createPcbCommand()

Function Description: Command. Asks the user for the name of a process and the priority to set on the process with the provided name. If the PCB was not found or the provided priority is out of the allowed range, it will prompt the user again for each parameter.
Parameter: *Process name* -> The name of the process to find.
Parameter: *New process priority* -> The priority to set the process with the name to. Must be a number from 0 - 9.
Returns: 0 always.

int setPcbPriorityCommand()

Function Description: Command. Asks the user for the name of a process. If found, it will print the information about the process from the PCB. Otherwise, it will output an error.
Parameter: *Process name* -> The name of the process to find.
Returns: 0 on success, 1 when the given process name was not found.

int showPcbCommand()

Function Description: Command. Asks the user for the name of the process to delete. If found and not a kernel class PCB, it will remove the PCB and then free it. Otherwise, it will output an error.
Parameter: *Process name* -> The name of the process to find and delete.
Returns: 0 on success, 1 when the given process name was not found or if the queried PCB is classified as a kernel PCB.

int deletePcbCommand()

Function Description: Command. Asks the user for the name of the process to suspend. If found, it will remove the PCB and set it to the suspended state before moving it to a ‘suspended’ queue. Otherwise, it will output an error saying the PCB can not be found.
Parameter: *Process name* -> The name of the process to find and set suspended.
Returns: 0 on success, 1 when the given process name was not found.

int suspendPcbCommand()

Function Description: Command. Asks the user for the name of the process to make active. If found, it will remove the PCB and set it to the active state before moving it to an ‘active’ queue. Otherwise, it will output an error saying the PCB can not be found.
Parameter: *Process name* -> The name of the process to find and set active.
Returns: 0 on success, 1 when the given process name was not found.

int resumePcbCommand()

Function Description: Command. Prints all ready PCBs in the ready queues.
Returns: 0 always.

int showPcbReadyCommand()

Function Description: Command. Prints all blocked PCBs in the blocked queues.
Returns: 0 always.
int showPcbBlockedCommand()

Function Description: Command. Prints all PCBs in all queues.
Returns: 0 always.
int showPcbAllCommand()

Function Description: Command. Creates a new background alarm process to notify a user after a certain given time.
Parameters: *Month, Day, Year* -> Date to set the alarm for.
Parameters: *Hour, Minute, Second* -> Time of day to set the alarm for.
Parameter: *Message* -> The accompanying message to notify the user with as the alarm is triggered.
Returns: 0 always.
int alarmCommand()

Function Description: Calls showFreeMemory() from memory.c file.
Returns: 0 always.
int showFreeMemoryCommand()

Function Description: Calls showAllocatedMemory() from memory.c file.
Returns: 0 always.
int showAllocatedMemoryCommand()

Function Description: Takes an input from the user and checks if its valid input, then passes it to the function allocateMemory() in memory.c file.
Parameter: *Size* -> Amount of memory to be allocated.
Returns: 0 for success, 1 for failure to allocate memory.
int allocateMemoryCommand()

Function Description: Takes an input from the user and checks if its valid input, then passes it to the function freeMemory() in memory.c file.
Parameter: *Address* -> The starting address of the memory block to free in hexadecimal.
Returns: 0 for success, 1 for failure to free memory.
int freeMemoryCommand()

Function Description: Shows the current allocated memory in the system by listing the addresses and the sizes of all allocated memory blocks.
Parameter: none
Returns: 0 always.
int showAllocatedMemory()

Function Description: Shows the current free memory in the system by listing the addresses and the sizes of all memory blocks.
Parameter: none
Returns: 0 always.
int showFreeMemory()

Function Description: Command. Yields the CPU. It basically calls sys_req(IDLE).
Returns: 0 always.
int yield()

Function Description: Command. Loads associated R3 processes in a non-suspended and ready state.
Returns: 0 always.
int loadR3()

File time.c/.h

Function Description: Takes one integer parameter and converts it to BCD and then returns it.
Parameter: integer -> An integer to convert.
Returns: An equivalent BCD coded byte from the integer.
unsigned char decimalToBCD(int integer)

Function Description: Takes one integer parameter and converts it to decimal and then returns it.
Parameter: bcd -> A BCD coded byte.
Returns: An equivalent integer from the BCD coded byte.
int BCDtoDecimal(unsigned char bcd)

Function Description: Takes time values to set the system time.
Parameter: hours -> Time in hours of a day (from 0 - 23).
Parameter: minutes -> Time in minutes of an hour (from 0 - 59).
Parameter: seconds -> Time in seconds of a minute (from 0 - 59).
void setTime(int hours, int minutes, int seconds)

Function Description: Gets the current system time and then prints it in a human readable, 24-hour format.
void getTime()

Function Description: Takes date values to set the system date.
Parameter: day -> Day of the month (must be from 1 to the last day of the provided month).
Parameter: month -> Month of the year (must be from 1 to 12).
Parameter: year -> Year in the 21st century (must be from 0 to 99).
void setDate(int day, int month, int year)

Function Description: Gets the current system date and then prints it in a human readable (Month, DD, 20YY) format.
void getDate()

Function Description: A process that gets the time from the system and then compares to alarm time to check if the set time exceeds the alarm time. In between checks, it will idle until the system time is greater than the alarm time. When the alarm is set off, it will print a given message and then exit.
Parameter: args -> Arguments which include the passed time to set off and an alarm message.
void alarmProcess(struct alarmProcessParams args)

File serial.c/.h

Function Description: Polls for user input from a (serial) device while echoing back input. Accepts alphanumeric and symbolic keys as well as special input like backspace, arrows, and delete. Upon hitting the enter key or when user input reaches the maximum buffer size, polling exits and the function returns with (null-terminated) user input stored in the provided buffer.
Parameter: device dev -> The device to read input from.
Parameter: const char* buffer -> A pointer to a user provided buffer.
Parameter: size_t len -> The size of the user provided buffer.
Returns: The length of the user input provided, which may be less than the size.
int serial_poll(device dev, const char *buffer, size_t len)

Enum Description: A collection of register offsets to each UART register on a COM device.
RBR, THR, DLL = 0
IER, DLM = 1
IIR, FCR = 2
LCR = 3
MCR = 4
LSR = 5
MSR = 6
SCR = 7
enum uart_registers

Struct Description: A list of DCBs containing possible COM devices that can be opened. A list of indices and their associated DCBs is listed below and corresponding to the mapping provided by “serial_devno()”:

COM1 -> serial_dcb_list[0]
COM2 -> serial_dcb_list[1]
COM3 -> serial_dcb_list[2]
COM4 -> serial_dcb_list[3]

struct dcb serial_dcb_list[4]

Enum Description: A set of error codes for specific functions in the serial driver.

```
 SERIAL_ERR_DEV_NOT_FOUND        =   -1,
 SERIAL_O_ERR_INVALID_EVPTR      = -101,
 SERIAL_O_ERR_INVALID_SPEED            = -102,
 SERIAL_O_ERR_PORT_ALREADY_OPEN = -103,
 SERIAL_C_ERR_PORT_NOT_OPEN      = -201,
 SERIAL_C_ERR_DEV_BUSY                = -204,
 SERIAL_R_ERR_PORT_NOT_OPEN     = -301,
 SERIAL_R_ERR_INVALID_BUFFER       = -302,
 SERIAL_R_ERR_INVALID_BUF_LEN     = -303,
 SERIAL_R_ERR_DEV_BUSY                = -304,
 SERIAL_R_ERR_OUT_OF_MEM           = -305,
 SERIAL_W_ERR_PORT_NOT_OPEN    = -401,
 SERIAL_W_ERR_INVALID_BUFFER      = -402,
 SERIAL_W_ERR_INVALID_BUF_LEN    = -403,
 SERIAL_W_ERR_DEV_BUSY              = -404,
 SERIAL_W_ERR_OUT_OF_MEM        = -405,
 SERIAL_S_ERR_DEV_NOT_FOUND   = -500,
 SERIAL_S_ERR_PORT_NOT_OPEN   = -501,
 SERIAL_S_ERR_INVALID_BUFFER    = -502,
 SERIAL_S_ERR_INVALID_BUF_LEN  = -503,
 SERIAL_S_ERR_DEV_BUSY            = -504,
SERIAL_S_ERR_OUT_OF_MEM        = -505,
```

enum serial_errors

Function Description: Returns an integer corresponding to the given device/port address. In other words, it maps an integer to a device for the purpose of indexing into the “initialized” array.

Parameter: device dev -> A device

Returns: Returns the number for a valid COM device, so COM1 would return 0, COM4 returns 3. -1 is returned if no mapping exists for a given device or port address.

int serial_devno(device dev)

Function Description: Immediately prints the buffer that is passed to the function.
Parameter: dev -> The device to output to.
Parameter: buffer -> A pointer to the provided buffer to write out from.
Parameter: len -> The size of the provided buffer.
Returns: Returns the buffer size if successful, returns -1 otherwise if the device provided was not initialized or is invalid.

int serial_out(device dev, const char *buffer, size_t len)

Function Description: Polls a serial device for new characters until either a new line or the filling of the buffer.
Parameter: dev -> The device to poll.
Parameter: buffer -> A pointer to the provided buffer to write to.
Parameter: len -> The size of the provided buffer.
Returns: Returns the length of the read input if successful, returns -1 otherwise if the device provided was not initialized or is invalid.

int serial_poll(device dev, char *buffer, size_t len)

Function Description: Initializes the serial port on the device with the desired baud rate. The function only supports these values: 110, 150, 300, 600, 1200, 2400, 4800, 9600, 19200.
Parameter: device dev -> The device to open
Parameter: int speed -> The desired baud rate.
Returns: 0 for success, any other number indicates failure with an error code corresponding to an enum value from “serial_errors”: -101 indicates an invalid event pointer, -102 indicates an invalid baud rate input, and -103 indicates the device port is already open.

int serial_open(device dev, int speed)

Function Description: Closes the serial port on the desired device.
Parameter: dev -> The device to close
Returns: 0 for success, any other number indicates failure with an error code corresponding to an enum value from “serial_errors”: -201 indicates a failure to close because the serial port is not open, -204 indicates an inability to close as the device is busy.

int serial_close(device dev)

Function Description: Attempts to do a read/write operation on a device with the given buffer, attempting to queue an operation if the device is not idle.
Parameter: dev -> The device to schedule a read/write operation on.
Parameter: buffer -> The pointer to the buffer to be used for the operation.
Parameter: buffer_sz -> The size of the provided buffer, must be greater than 0 for both read and write operations.
Parameter: io_op -> The operation to be performed (read/write).
Returns: 0 for success, any other number indicates failure with an error code corresponding to an enum value from “serial_errors”: -500 indicates the device was not found, -501 indicating the port is not open, -502 indicates the buffer provided is invalid, -503

indicates the buffer size is invalid, -504 indicates the device is busy, -505 indicates an out of memory error.

int serial_schedule_io(device dev, unsigned char* buffer, size_t buffer_sz, unsigned char io_op)

Function Description: Called from “serial_isr” as a first level handler. It identifies the device which raised an interrupt and calls a corresponding second level handler according to the operation of the identified device.

void serial_interrupt(void *)

Function Description: Initiates a read operation on an idle device.
Parameter: dev -> The device to read from.
Parameter: buf -> A pointer to the provided buffer.
Parameter: len -> The size of the provided buffer.
Returns: 0 for success, any other number indicates failure with an error code corresponding to an enum value from “serial_errors”:

int serial_read(device dev, char* buf, size_t len)

Function Description: Initiates a write operation on an idle device.
Parameter: dev -> The device to write to.
Parameter: buf -> A pointer to the provided buffer.
Parameter: len -> The size of the provided buffer.
Returns: 0 for success, any other number indicates failure with an error code corresponding to an enum value from “serial_errors”:

int serial_write(device dev, char* buf, size_t len)

Function Description: A second level interrupt handler for “serial_isr”. If the DCB is set to a read operation, the handler attempts to read a character from the device specified in the DCB, appending it to the current IOCB buffer and signals completion upon filling the buffer or reading a new line. Attempts to store character in ring buffer if reading is not the current operation on the DCB.
Parameter: dcb -> The DCB to operate on.

void serial_input_interrupt(struct dcb* dcb)

Function Description: A second level interrupt handler for “serial_isr”. If the DCB is set to a write operation, the handler writes out a character from the current IOCB buffer of the given DCB and then may signal completion upon having written the whole buffer.
Parameter: dcb -> The DCB to operate on.

void serial_output_interrupt(struct dcb* dcb)

Function Description: Attempts to initialize the specified device.
Parameter: dev -> A serial device to be initialized.
Returns: 0 for success, -1 for failure if the given device is invalid.
int serial_init(device dev)

Function Description: Checks a device for completed I/O via event flags. If a DCB is finished with an operation, completion is performed before advancing and configuring a DCB to work on other queued operations.
Parameter: dev -> A device to check for completed I/O.
Returns: 0 if the DCB did not have completed I/O, 1 otherwise upon no error. -1 is returned if provided with an invalid device.
int serial_check_io(device dev)

File string.c/.h

Function Description: Translates an integer to a null-terminated string.
Parameter: string -> A user provided string buffer to output the representation of the input integer.
Parameter: int integer -> An integer to translate into a string.
void itoa(char string[], int integer)

File pcb.c/.h

Struct Description: A structure defining PCB state bitfields and priority.
Field: exec -> Execution state corresponding to a value from “ProcExecState”.
Field: dpatch -> Dispatch state corresponding to a value from “ProcDpatchState”.
Field: cls -> Execution state corresponding to a value from “ProcClassState”.
Field: pri -> The priority of a process.
struct pcb_state

Struct Description: A process control block structure for maintaining process information for a process.
Field: p_next -> Points to another PCB or NULL if the PCB is in a queue.
Field: name -> The name of a process.
Field: state -> The contained state of a process
Field: pctxt -> Pointer to the saved context of a process.
Field: pstackseg -> A pointer to the memory allocated for the stack of a PCB.
struct pcb

Enum Description: Defines unique process execution states.
PCB_EXEC_BLOCKED
PCB_EXEC_READY
PCB_EXEC_RUNNING
enum ProcExecState

Enum Description: Defines unique process dispatch state identifiers.
PCB_DPATCH_SUSPENDED
PCB_DPATCH_ACTIVE
enum ProcDpatchState

Enum Description: Defines unique process class identifiers.
PCB_CLASS_SYSTEM
PCB_CLASS_USER
enum ProcClassState

Struct Description: A queue to hold queued PCB handles, linked list implementation.
Field: pcb_head -> If the queue is not empty, points to head/front of the queue which can be dequeued. NULL otherwise.
Field: pcb_tail -> If the queue is not empty, tail points to the tail/back of the queue to help enqueue an element. NULL otherwise.
Field: type_pri -> Identifies whether the queue is a priority queue. If so, inserting via pcb_insert().
struct pcb_queue

Function Description: Allocate memory for a new PCB.
Returns: A non-NULL pointer to a newly allocated PCB on success. NULL on error during allocation or initialization.
struct pcb* pcb_allocate(void)

Function Description: Frees all memory associated with a given PCB, including its stack.
Parameter: struct pcb* pcb -> A pointer to the pcb to free.
Returns: 0 on success or otherwise a negative value upon error. It returns -1 if there was an error with freeing the PCB or its associated stack.
int pcb_free(struct pcb* pcb)

Function Description: Allocates a new PCB, initializes it with data provided, and sets state to active-ready.
Parameter: name -> Name string for the new process. Must be a NULL-terminated string and no larger than the size defined by MPX_PCB_PROCNAME_SZ.
Parameter: enum ProcClass cls -> Class of the new process.
Parameter: unsigned char pri -> Priority of the new process.

Returns: A non-NULL pointer to the created PCB on success, NULL on error during allocation, initialization, or invalid parameters.

struct pcb* pcb_setup(const char* name, enum ProcClass cls, unsigned char pri)

Function Description: Searches all process queues for processes with the provided name.
Parameter: const char* name -> Name of the process to find.
Returns: A non-NULL pointer to the found PCB on success. NULL if the provided name was not found in any queue.

struct pcb* pcb_find(const char* name)

Function Description: Inserts a PCB into the appropriate queue based on state and priority.
Parameter: pcb -> A pointer to the PCB to enqueue. Assumed to be a valid handle to a PCB with a unique name.

void pcb_insert(struct pcb* pcb)

Function Description: Removes a PCB from its current queue without freeing memory or data structures.
Parameter: pcb -> A pointer to the PCB to dequeue. Assumed to be a valid handle.
Returns: 0 on success or a negative value if there was an error. A value of -1 indicates that the passed PCB handle does not match any other handles in the associated queue. A value of -2 indicates an error with freeing a node in a target queue.

int pcb_remove(struct pcb* pcb)

Function Description: Sets up the context and stack of a PCB to have an entry point such that it will be able to run as a process.
Parameter: pcb -> A pointer to the PCB to set up the context for. Assumed to be a valid handle.
Parameter: func -> A pointer to the function that will serve as an entry point for the process to run. Assumed to be valid.
Parameter: fargs -> A pointer to an argument buffer to copy arguments from to pass into the function. Contained arguments should match the signature of the passed function.
Parameter: fargc -> Size of the buffer pointed to by fargs.

void pcb_context_init(struct pcb* pcb, void* func, void* fargs, size_t fargc)

System Call Routines

File sys_call.c/.h

Function Description: Checks all devices for completed I/O and performs completion on each one that is completed.
Returns: 0 if no DCBs held completed I/O, 1 otherwise.
unsigned char sys_check_io()

Function Description: This function takes pointer to the current process context and then returns the context of the process to be loaded if there needs to be a context switch. When the last process does an exit request, it will load the first context from where an IDLE request was issued.
Parameter: context_in -> A pointer to a struct that refers to the context of the current process.
Returns: A pointer to the context of the process to be loaded in the system. It returns -1 if the call was a READ or WRITE operation or otherwise unrecognized.
struct context* sys_call(struct context* context_in)

File syscalls.c/.h

Function Description: An alias of sys_req(WRITE).
Parameter: dev -> The device to output to.
Parameter: buffer_in -> A pointer to an allocated buffer to write out from.
Parameter: buffer_in_sz -> Size of the buffer buffer_in.
Returns: sys_req(WRITE, dev, buffer_in, buffer_in_sz)
int write(device dev, const void* buffer_in, size_t buffer_in_sz)

Function Description: An alias of sys_req(READ).
Parameter: dev -> The device to output to.
Parameter: buffer_inout -> A pointer to an allocated buffer to write into.
Parameter: buffer_inout_sz -> Size of the buffer buffer_inout.
Returns: sys_req(READ, dev, buffer_inout, buffer_inout_sz)
int read(device dev, const void* buffer_inout, size_t buffer_inout_sz)

File irq.s

Function Description: An assembly syscall ISR. It saves the current processor state and pushes it to stack, then restores processor state and returns from ISR under operations not involving a context switch. If sys_call() returns a value other than -1, it performs a context switch on the address pointed to by the context.
sys_call_isr

Function Description: An assembly serial ISR that calls "serial_interrupt()" to handle serial interrupts and operations.

serial_isr

File memory.c/.h

Function Description: Initialize a new heap and take size as parameter for the heap.
Parameter: size -> Size to create the heap with.
Returns: nothing.

void initialize_heap(size_t size)

Function Description: Allocate memory to the mcb if there is enough space in the mcb to be allocated in.
Parameter: size -> Size of memory to allocate.
Returns: A pointer to the location of the allocated memory block.

void* allocate_memory(size_t size)

Function Description: Free the block from memory by finding the allocated block then remove all lists that it has inside.
Parameter: ptr -> Pointer to a valid block to be freed.
Returns: 0 for success freeing the memory, 1 for failure to place the memory.

int free_memory(void* ptr)

File device.c/.h

Enum Description: Provides the associated port addresses for serial devices COM1 up to COM4.
COM1
COM2
COM3
COM4

enum device

Enum Description: Defines a set of possible operations on the serial devices.
IO_OP_READ
IO_OP_WRITE

enum io_op

Struct Description: Represents a serial or I/O operation to be done on a device. It helps form a queue for other pending operations.
Field: p_next -> Points to a next operation or NULL if none.
Field: pcb_rq -> The PCB requesting the operation associated with the IOCB.
Field: buffer -> The buffer provided by the requesting process for an I/O operation to use.
Field: buffer_sz -> The size of the provided buffer.
Field: io_op -> The code for the operation, i.e., it identifies the operation for an IOCB.

struct iocb

Struct Description: Represents an I/O device.
Field: dev -> The associated port address.
Field: iocb_queue_head -> The head, current operation being performed if not NULL.
Field: iocb_queue_tail -> The tail operation used for insertion of other operations.
Field: rbuffer -> A ring buffer to use for input operations.
Field: rbuffer_sz -> The size of the provided ring buffer.
Field: rbuffer_idx_begin -> The left boundary of the ring buffer, i.e., an index to read from next.
Field: rbuffer_idx_end -> The right boundary of the ring buffer, i.e., an index to write to next.
Field: buffer_idx -> Stores progression on the current operation.
Field: open -> Indicates if the device has been opened and configured.
Field: event -> Indicates completion of an operation to remove its associated IOCB for the next operation.

struct dcb