Programmer’s Manual

R6

12/8/2021

By:
Will Ferrell, Jarett Hardek, Anthony Lowery, Emily Waechter

Table of Contents

void version():

-Prints the current version of MPX and its completion date
-Void function, prints the version and date to the terminal

void help():

-Provides usage instructions for each command available to the user
-Void function, prints each command on a new line

int shutdown():

-Exits command handler loop, shuts down the MPX machine
-Execution returns to kmain()
-Requires confirmation by users

Return - (integer): that is assigned to “quit”, the main command handler control loop variable

int comhand():

-Collects input from the user and validates that it is valid
-Controls the design of the user interface
-Menu driven, listing each command implemented in the MPX

Return - (integer-pointer): arbitrarily returns 0 to terminate the command handler

int print(char* str):

-Prints a string to the terminal

Parameter - str (char pointer)
Return - (integer): arbitrarily returns 0 to terminate the command handler

int println(char* str):

-Prints a string on a new line to the terminal

Parameter - str (char pointer)
Return - (integer): arbitrarily returns 0 to terminate the command handler

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

-Gathers keyboard input from the user and passes it into the command buffer which is stored as a char pointer
-It validates each keystroke including all alphanumeric characters, backslash, forward slash, comma, and period. It also handles special keys such as backspace, delete, the home and end key, and the arrow keys

Parameter - buffer (char-pointer): the data that is stored in the command buffer (usually empty)
Parameter - count (integer-pointer): the value of the size of the buffer (this is the value that remains unchanged and returns)
Return - (integer-pointer): the value of the size of the buffer

int gettime():

-Prints the time that was accessed through the outb function
calls bcdtoStr on the time to convert from BCD to a char and place it in the time char array

Return - (integer): arbitrarily returns 0

int settime(char *time):

-The time has already been inputted after printing a prompt in the command handler
-Calls check_time_str to make sure the time entered was valid
sets the time using the outB and StrtoBCD function to convert the char to a BCD

Parameter - time (char pointer)
Return - (integer): 0 if time format is valid or 1 if the time is invalid

int check_time_str(char* time_str):

-Makes sure the time is of valid length and valid format in military time
-If 0 is returned the settime function continues if 1 is returned a invalid format message is returned

Parameter - time_str (char pointer)
Return - (integer): 0 if time format is valid or 1 if the time is invalid

int getdate():

-Prints the date that was accessed through the outb function
-Calls bcdtoStr on the date to convert from BCD to a char and place it in the date char array

Return - (integer): arbitrarily returns 0

int setdate():

-Reads in the date entered by the user after printing a prompt
-Calls checkDate to make sure the date entered was valid
-Sets the date using the outB and StrtoBCD function to convert the char to a BCD

Return - (integer): arbitrarily returns 0

int checkDate(char* date):

-Makes sure the date is valid
-If 0 is returned the setdate function continues if 1 is returned a invalid format message is returned

Parameter - date (char pointer)
Return - (integer): 0 if time format is valid or 1 if the time is invalid

int intToBCD(int val):

-Converts an int value to a BCD

Parameter - val (integer)
Return - a BCD

int BCDToInt(int val):

-Converts a BCD to an int value

Parameter - val (integer)
Return - an int

int BCDtoStr(int val, char *str):

Parameter - val (integer): BCD value
Parameter - str (char array pointer)
Return - 0 but puts the BCD into the char array pointer

int StrtoBCD(int val, char *str):

-Converts a char pointer to binary coded decimal

Parameter - val (integer): BCD value
Parameter - str (char array pointer)
Return - the BCD of the number that was in the char array

void fifo_enqueue(queue *q, PCB *pcb):

-puts a new pcb into a fifo queue

Parameter - q (queue pointer)
Parameter - pcb (PCB- process control block pointer)

void priority_enqueue(queue *q, PCB *pcb):

-puts a new pcb into a priority queue

Parameter - q (queue pointer)
Parameter - pcb (PCB- process control block pointer)

PCB* AllocatePCB():

-allocates memory for a new PCB, and performs initialization

Return - PCB Pointer if successful, or NULL if an error occurs during allocation

void FreePCB():

-frees all memory for a given PCB

Return - Success or Error Code

PCB* SetupPCB(char* processName, unsigned char processClass, int priority):

-calls AllocatePCB() to create an empty pcb, performs initialization, and sets it to ready not suspended

Parameter - processName (char pointer)
Parameter - processClass (unsigned char)
Parameter - priority (integer)
Return - PCB Pointer if successful, or NULL if an error occurs

PCB* FindPCB(char* processName):

-searches all queues for a pcb with a given process name

Parameter - processName (char pointer)
Return - PCB Pointer if successful, or NULL if an error occurs

void InsertPCB(PCB *pcb):

-inserts a PCB into the appropriate queue

Parameter - pcb (PCB pointer)

void RemovePCB():

-removes a PCB from its current queue

Parameter - pcb (PCB pointer)

void CreatePCB(char* processName, unsigned char processClass, int priority):

-calls SetupPCB() and inserts the PCB into the appropriate queue

Parameter - processName (char pointer)
Parameter - processClass (unsigned char)
Parameter - priority (integer)

void DeletePCB(char* processName):

-removes the pcb from the appropriate queue, and free all associated memory

Parameter - processName (char pointer)

void BlockPCB(char* processName):

-Finds the given PCB, sets it to blocked, then reinserts it into its queue

Parameter - processName (char pointer)

void UnblockPCB(char* processName):

-Finds the given PCB, sets it to unblocked, then reinserts it into its queue

Parameter - processName (char pointer)

void SuspendPCB(char* processName):

-Finds the given PCB, sets it to suspended, then reinserts it into its queue

Parameter - processName (char pointer)

void ResumePCB(char* processName):

-Finds the given PCB, sets it to not suspended, then reinserts it into its queue

Parameter - processName (char pointer)

void SetPCBPriority(char* processName, int priority):

-sets the priority of the pcb

Parameter - processName (char pointer)
Parameter - priority (integer)

void ShowPCB(char* processName):

-Displays the process name, class, state, suspended status and priority of all PCB's in the ready queue

Parameter - processName (char pointer)

void ShowReady():

-Displays process name, class, state, suspended status, and priority

void ShowBlocked():

-Displays the process name, class, state, suspended status and priority of all PCB’s in the blocked queue

void ShowAll():

-Displays the process name, class, state, suspended status and priority of all PCB’s in both the ready and blocked queues

sys_call_isr():

-the system call interrupt service routine is written in assembly language. This pushes registers to the stack, calls sys_call, and pops the registers from the stack

sys_call(context* register):

-Prepares MPX for the next ready process to begin/resume execution.

Parameter - register (context pointer)

void yield():

-Causes the command handler to yield to other processes, if any processes are in the ready queue.

void loadr3():

-Loads all r3 processes into memory in a suspended ready state at any priority.

void createAlarmProcess():

-Creates a PCB for the alarm.

void addAlarm(char* message, int alarmTime):

-Creates a new alarm that contains a specified message and outputs the message at a given time

Parameter - message (char pointer)

Parameter - alarmTime (char pointer)

u32int initialize_heap(u32int size):

-initializes the heap as free memory and declares start_of_memory and total_size as global variables.

Parameter - size (u32int)

u32int allocate_memory(u32int bytes):

-Allocates the specified block of bytes/memory if there is enough room in free memory for bytes+sizeof(CMCB) and inserts the CMCB into the allocated list.

Parameter - bytes (u32int)

int free_memory(void* address):

-Frees the particular block of allocated memory and inserts the specified block into the free memory list, if there are two adjacent blocks of free memory they are merged. Returns 1 upon success.

Parameter - address (u32int)

int IsEmpty():

-Checks if the allocated list is empty to indicate whether or not the heap only contains free memory. If a 1 is returned the heap is empty, if a 0 is returned the heap contains allocated memory.

void Show_Allocated_Memory():

-Throws an error if the allocated list is empty or prints all of the beginning addresses and sizes of the allocated list in order of beginning address.

void Show_Free_Memory():

-Prints all of the beginning addresses and sizes of the allocated list in order of beginning address.

int com_open(int baud_rate):

-Called to initialize the serial port. This function has two parameters. The second is an integer value representing the desired baud rate.

Parameter - baud_rate (integer)
Return - integer 0 or error code

int com_close(void):

-Called to close the serial port. Clears the open indicator in the DCB, disables the appropriate level in the PIC mask register, Disables all necessary interrupts in the ACC, and restores the original saved input vector

Return - integer 0 or error code

int com_read(char* buf_p, int* count_p):

-Obtains input characters and loads them into the requestor's buffer.

Parameter - buf_p (char pointer)
Parameter - count_p (integer pointer)
Return - integer 0 or error code

int com_write(char* buf_p, int* count_p):

-Initiates the transfer of a block of data to the serial port.

Parameter - buf_p (char pointer)
Parameter - count_p (integer pointer)
Return - integer 0 or error code

void top_handler():

-If the port is open reads the interrupt id register, calls the appropriate second level handler,then clears the interrupt. If the port is not open it clears the interrupt and returns

void output_h():

-If the current status is writing and all characters have been transferred it resets the status to idle, sets the event flag and returns the count value; if all characters have not been transferred, it gets the next char from the requestors output buffer and stores it in the output register then returns without signalling completion

void input_h():

-If the current status is not reading, the character is stored in the ring buffer, and discards the character if the ring buffer is full. Then returns to the first level handler.

void enqueue(iqueue *q, iocb *iocb):

-Puts a new iocb into a fifo queue

Parameter - q (iqueue pointer)
Parameter - iocb (iocb - i/o control block pointer)

void dequeue(iqueue *q):

-Removes and returns the iocb at the top of the queue

Parameter - q (iqueue pointer)
Return - iocb or NULL if the queue is empty

void disable_interupt():

-Disables all interrupts

void enable_interupt():

-Enables all interrupts