CS 450 OPERATING SYSTEMS STRUCTURE

**MPX Programmer’s Manual**

WEST VIRGINIA UNIVERSITY

Prepared by:

*FAT Musketeers*

Module 6 Deliverable

10/6/2013

**TABLE OF CONTENTS**

1. General Information ……………………………………………………….. 3
   1. System Overview ..………………………………………………….. 3
   2. References …………………………………………………………... 3
   3. Points of Contact …………………………………………………... 3
2. The functions ………………………………………………………………... 4

2.1 main.c ……...…………………………………………………………. 4

2.2 date.c...……………………………………………………………..….4

2.3 commands.c ……………………………………………………….. 4

2.4 pcb.c ………………………………………………………………….6

**1. GENERAL INFORMATION**

**1.1 System Overview**

This manual provides detailed instructions for the Multi Programming Executive, or MPX project. MPX is an experimental platform designed to allow the study of realistic issues in the design of selected OS components. MPX operates in the command-line environment. The MPX project is still under development.

**1.2 References**

More information about this project can be found at the following site:

* csee.wvu.edu/~jdmooney/classes/cs450
* community.wvu.edu/~jad103/classes/CS450
* cs450.info

**1.3 Points of Contact**

Feel free to ask the programmers:

* Edgardo Antuna – [eantuna@mix.wvu.edu](mailto:eantuna@mix.wvu.edu)
* Bryan Turek – [turek.bryan@gmail.com](mailto:turek.bryan@gmail.com)
* Jake Ferguson - Jfergus7@mix.wvu.edu

**2. USING THE SYSTEM**

**2.1 main.c**

**FUNCTIONS**

**int main();** - initializes the system, queues, registers and IO devices. Loads the command handler and an IDLE process. Dispatches the command handler and IDLE process.When exit command is detected and the dispatcher is done it closes IO devices, cleans queues and exits.

**void loadComHandler();** - Loads the command handler as a system process with highest priority.

**void prompt();** executes a while loop prompting the user for input, then reads and parses the input. Next it calls the function executeCommand. If the command “Exit” is read the while loop breaks.

**void write(char \*string);** Takes a char\* as a parameter then calls sys\_req to print the screen on the terminal.

**void read(int size);** reads input from the terminal and returns a char\*

**char\* trim(char\* s);** takes a char\* as a parameter, then strips all leading and trailing spaces. It then returns the trimmed char\*

**2.2 date.c**

**FUNCTIONS**

**int isDateValid(int month, int day, int year);** It checks if the month, day and year a valid dates. It returns 1 if valid, 0 if not valid.

**int isLeapYear(int year);** Checks if the year is a leap year.

**Char\* getMonth(int month);** takes the number of the month as a parameter and returns the string of the month name.

**2.3 commands.c**

**FUNCTIONS**

**int executeCommand(int op, char\* command);** is called from prompt() in r1.c, compares the char\* command to match a function then calls the function with the op code of EXEC or HELP.

**void quit(int op);** only runs with the op code of HELP, prints help statement.

**void help(int op);** prints all of the help commands.

**void version(int op);** prints the current version or prints the help statement.

**void getDate(int op);** gets and prints the current date or the help statement.

**void setDate(int op);** splits the date into tokens by “/” then checks if the date is valid. If it is valid it sets the currentDate in MainShell, or displays the help statement

**void list(int op);** opens the directory given by the path or opens the current directory if no path was given. Then it loops through each entry by calling sys\_get\_entry printing the name and filesize. It then closes the directory.

**void create\_pcb(int op):** creates a pcb with the three parameters of name, priority, class by calling createPCB(char\*,int, int) in pcb.c

**void delete\_pcb(int op):** finds and deletes a pcb with the parameter of name by calling deletePCB(char \*) in pcb.c

**void block(int op):** finds by the name and sets a pcb to the blocked state by calling blockPCB(char \*) in pcb.c.

**void unblock(int op):** finds by the name and sets a pcb to the ready state by calling unblockPCB(char \*) in pcb.c.

**void suspend(int op):** finds by the name and sets a pcb to the suspended state by calling suspendPCB(char \*) in pcb.c.

**void resume(int op):** finds by the name and sets a pcb to the unsuspended state by calling resumePCB(char \*) in pcb.c.

**void set\_priority(int op):** changes a pcb’s priority, takes in two parameters, char \* name and int priority.

**void show\_pcb(int op):** finds a pcb by the name and displays the information about that pcb.

**void show\_ready(int op):** shows all the pcbs in the ready and the suspended ready queue.

**void show\_blocked(int op):** shows all the pcbs in the blocked and the suspended block queue.

**void show\_all(int op):** shows all the pcbs by calling show\_ready and show\_blocked.

**2.5 pcb.c**

**FUNCTIONS**

**Void initR2():** allocates memory for the queues and initializes the four queues.

**Void cleanR2():** frees all the PCB’s and the queues.

**PCB\* allocatePCB():** allocates memory for the PCB

**int freePCB(struct PCB\*)** frees the memory for the PCB, returns an error code if there is one.

**PCB\* setupPCB(char \* name, int pri, int class):** initializes the PCB by the given parameters and calls insertPCB(struct PCB\*) to put the PCB in the queue.

**PCB\* findPCB(char \* name):** searches all the queues for a PCB that has the same name.

**void insertPCB(struct PCB\*):** inserts the PCB in the correct queue according to its state.

**void removesPCB(struct PCB\*):** removes the PCB from the queue

**2.6 R3.c**

**FUNCTIONS**

**Void dispatcher():** prepares the first process of the ready queue for execution by switching to its stack.

**Void interrupt sys\_call():** handler called by the system when the int 60h instruction is encountered. Saves the state of the currently operating process, checks if any I/O operations have been completed and calls dispatcher() to get the next process.

**Void initR3():** sets up temporary register variables for stack segment and stack pointer. Sets up control structures for I/O devices.

**Void loadProgram(char \*name, int priority):** Allocates and sets up new PCB with parameters *name, priority, APP*; allocates program memory and loads program into that memory.

**Void ioScheduler():** processes I/O requests by creating a new IOD for the I/O request and either processing it immediately if the device is ready or placing it in the waiting queue to be run later.

**Void openDeviceDrivers():** opens terminal and com\_port devices.

**Void closeDeviceDrivers():** closes terminal and com\_port devices.

**STRUCTURES**

**Context:** the top 24 bytes of PCB stack. 12 register values needed to begin/resume execution.

**Params:** parameters passed to the interrupt handler via the stack;

**Iod:** holds information about an I/O device.

**Iocb:** holds information about an I/O request.

**2.7 R5.c**

**FUNCTION**

**Int com\_write(char \*buff, int \*count):** Initiates the transfer of data to the serial port. Pass to it a buffer containing the string to send. It checks if the device is free. If it is it will indicate that the device is now busy and will write the first byte in the buffer *buff* to the serial port.

**Int com\_read(char \*buff, int \*count):** Loads input into the buffer. *Buff* is where the read characters can be placed. *Count* is a pointer to the number of characters to read.

**Int com\_open(int \*eflag\_p, int baud\_rate):** Initializes the serial port. Initializes the Device Control Block, sets the interrupt vector to the new handler address, computes and stores the baud rate divisor and enables necessary interrupts.

**Int com\_close():** closes the serial port. Checks that the port is open, closes the DCB, disables appropriate level in the PIC mask register and serial interrupts and restores the original interrupt vector

**Void interrupt handler():** Determine cause of the interrupt and call the appropriate second level interrupt. Called when an I/O interrupt on the com port is generated.

**Void input\_int():** Place character in the requestor’s input buffer. Called when a read interrupt is generated.

**Void output\_int():** If there is data in the output buffer in the DCB that needs to be copied it will copy the next character to the output register and this procedure will be repeated until all data has been written. If there is no data in the output buffer it will perform cleanup operations to indicate that the write operation is complete.

**STRUCTURES**

**DeviceControlBlock:** stores information about the com device.