MPX-OS VER 4.66 Programmers Manual

PREPARED BY

Username programming group

Brandon Steel

James Jackman

Chelsey Randolph

Mohammad Alzayer

Kawthar Abushaheen

How To Use This Manual

This manual is intended for programmers who intend to write their own programs with this OS and give insight into what each function in every file is intended to do and give example inputs and outputs.

This OS utilizes the C-programming language

Table of Contents

e. insert_pcb
f. remove_pcb
g. create_pcb
h. delete_pcb
i. block_pcb
j. unblock_pcb
k. suspend_pcb
l. resume _pcb
m. set_pcb_priority
n. show_pcb
o. show_ready
p. show_blocked
q. show_all
r. resume_all

3.1 Dispatcher.c

a. sys_call
b. yield
c. loadProcess
d. loadr3
e. infinite
f. sys_load_proc

3.2 Test_process.c

g. proc1
h. proc2
i. proc3
j. proc4
k. proc5

4.1 Alarm.c

a. remove_alarm
b. setAlarm
c. checkAlarms

4.2 colortext.c

d. Write_text
e. Write_text_red
f. Write_text_bold_red
g. Write_text_green
h. Write_text_bold_green
i. Write_text_yellow
j. Write_text_bold_yellow
k. Write_text_blue
l. Write_text_bold_blue
m. Write_text_magenta
n. Write_text_bold_magenta
o. Write_text_cyan
p. Write_text_bold_cyan

5.1 Memory_man.c

a. Init_heap
b. Alloc_mem
c. findCMCB
d. Free_mem
e. Merge_free_blocks
f. IsEmpty
g. Show_free_mem
h. Show_alloc_mem
i. Unlink
j. Insert_mem

5.2 Hex.c

k. printDecToHex

6.1 String.c

a. printf

1.1 Command Handler - comhand.c

a. **comhand()**

This function uses a command buffer and a menu based system in order to execute commands the user inputs. The input is a name of a function that will be executed.

The function itself has no defined input, though it does read user input from the system requirement write in order to execute the users command.

b. **version()**

Version is a function with no input and a write to the terminal for an output, it outputs a character array for the current version of the MPX operating system.

c. **displayAllCommands()**

This function will display all the commands that are available for the user to use.

d. **inputHelp(char helpBuffer[])**

inputHelp is the help function that is menu based similar to the command handler. The input is the word “help” + function name. This function will describe what the other functions do.

Parameters:

char helpBuffer[]: the buffer containing the input from the user

e. **shutDown()**

   This function first prompts the user with a confirmation message, then either exits back to comhand or shuts down the system.

f. **displayMenu()**

   At the start of our system, this will display our “Username” logo with the menu once.

g. **clear()**

   This function clears the terminal’s window of previous inputs and outputs.

h. **showHistory**

   This function displays the current user functions in history.

i. **addCmdToHistory(char[ ] command)**

   This function adds the most recent user command to the command history.

   Parameters:

   Char [ ] command: command being added

j. **getCommand(int index)**

   This function gets the command from the command history list.

   Parameters:

   Int index: the index of the command

k. **getHistorySize**

   This function gets the command’s size from the command history list.

l. **substring(char [ ] string, int start, int end)**

This function creates a substring from the command for autocomplete.

Parameters:

Char [ ] string: string from autocomplete to create substring from

Int start: the start index

Int end: the end index

m. **Auto_complete(char [ ])**

This function takes the user's input after tab is hit and tries to find a match within the user commands to autocomplete the rest of the word.

Parameters:

Char [ ] partial_str: buffer from polling

1.2 Date - date.c

n. **getDate()**

This function gets the current date that is set on the system.

o. **setDate()**

This function allows the user to set a new date on the system.

1.3 Time- time.c

p. **setTime()**

This function allows the user to set a new time on the system.

q. **getTime()**

This function gets the current time that is set on the system.

r. **BCDToDEC(int num)**

This function converts a binary number to a decimal number.

Parameters:

int num: the number to be converted

s. **DECToBCD(int num)**

This function converts a decimal number to a binary number.

Parameters:

int num: the number to be converted

1.4 Polling - polling.c

t. **init_polling(char *buffer, int *count)**

This function collects input from the user's keyboard.

Parameters:

char *buffer: a pointer that represents the input buffer

int *count: pointer that represents the size of the input buffer

2.1 PCB - pcb.c

a. **allocate_pcb()**

Allocate_pc is the allocate function that is type of internal functions to set up the pcb in memory and returns the size of the pcb.

b. **setup_pcb(char name[ ], int pclass, int priority)**

Setup_pcb is an internal function that includes a name of a process, the process class, and a priority. This function is used to allocate the memory to the new pcb by calling allocate_pcb(). The process name it must be at least 8 characters. The priority must be a number from 0 through 9. The process class must be either 0 for the user or 1 for the system.

Parameters:

char name[ ] : character array that represents the name of the process.

int pclass : an integer that represents the class classes whether it's 0 or 1.

int priority: an integer that represents the priority number.

c. **free_pcb(PCB* pcb)**

free_pcb is an internal function that's allow the user to free the pcb in memory by sys_free_mem().

Parameters:

PCB* pcb : a pointer that points to the process control block in the stack function in head file.

d. **find_pcb(char process_name[ ])**

Find_pcb is an internal function that allows the user to find a specific pcb. This function checks the four linked lists (ready_queue, blocked_queue, suspend_ready_queue, and suspend_block queue) for the given pcb name.
Parameters:

char process_name[ ] : is a character array that is used to compare the name given by the user to the current names of processes.

e. **insert_pcb(PCB *pcb)**

Insert_pcb is an internal function that allows the user to insert a pcb in an appropriate queue.

Parameters:

PCB *pcb: a pointer to the pcb which the user wants to insert into a specific queue

f. **remove_pcb(PCB *pcb)**

remove_pcb is an internal function that allows the user to remove a pcb from the queue in which it is currently stored.

Parameters:

PCB *pcb: a pointer to the pcb which the user wants to remove from a specific queue.

g. **create_pcb()**

This function will call setup_pcb and insert_pcb to insert the new pcb in the appropriate queue. It also checks the data being sent in from the user to make sure it meets the requirements from the system.

h. **delete_pcb(char name[30])**

This function will take a pcb name, find it, remove it, and then free all associated memory.

Parameters:

char name[30]: name of the pcb the user wants to delete.

i. **block_pcb(char name[30])**

This function will take a pcb name the user has given and set its state to blocked.

Parameters:

char name[30]: name of the pcb the user wants to block.

j. **unblock_pcb(char name[30])**

This function will take a pcb name the user has given and set its state to ublocked.

Parameters:

char name[30]: name of the pcb the user wants to unblock.

k. **suspend_pcb(char name[30])**

This function will take a pcb name the user has given and set its state to suspended.

Parameters:

char name[30]: name of the pcb the user wants to suspend.

l. **resume_pcb(char name[30])**

This function will take a pcb name the user has given and set its state to not suspended.

Parameters:

char name[30]: name of the pcb the user wants to unsuspend.

m. **set_pcb_priority(char name[30], int new_priority)**

This function will take a pcb name the user has given and set its new priority to the given integer.

Parameters:

char name[30]: name of the pcb the user wants to set its priority.

int new_priority: integer of the new priority.

n. **show_pcb(char name[30])**

This function will take a pcb name the user has given and displays information about its state.

Parameters:

char name[30]: name of the pcb the user wants to display.

o. **show_ready()**

This function shows all the pcbs in the ready queue.

p. **show_blocked()**

This function shows all the pcbs in the blocked queue.

q. **show_all()**

This function shows all the pcbs in the system.aa

r. **resume_all()**

This function resumes all the processes that are currently suspended.

3.1 Dispatcher.c

a. **sys_call**

This function allows a context switch between the currently operating process and the next process to be executed.

b. **yield**

This command causes the comhand to yield to another process and if there is any process in the ready queue they will be executed.

c. **loadProcess**

It's a function that creates a process given the name ,class ,priority and functions to execute. It sets up the context with the specified registers within the system and returns the new process.

Parameters:

char name[ ] : character array that represents the name of the process.

int pclass : an integer that represents the class classes whether it's 0 or 1.

int priority: an integer that represents the priority number.

void function: the function that is being called.

d. **loadr3**

This function loads all of the R3 test _processes into memory in a suspended ready state at the priority of the users choosing.

e. **infinite**

This function runs until it is suspended from the ready queue. It must be suspended before the user can delete it. It also must be suspended before the user can shutdown the system. During execution, this function prints a message and then idles to comhand.

f. **sys_load_proc**

This function assists our load processes function to help load the new process into the correct queue.

3.2 Test_process.c

e. proc1

This function prints out a message depending on conditional statement, then idles. If function exceeds the conditional statement, it prints out an error message. Once the function has completed the conditional statement, it exits.

f. proc2

This function prints out a message depending on conditional statement, then idles. If function exceeds the conditional statement, it prints out an error message. Once the function has completed the conditional statement, it exits.

g. proc3

This function prints out a message depending on conditional statement, then idles. If function exceeds the conditional statement, it prints out an error message. Once the function has completed the conditional statement, it exits.

h. proc4

This function prints out a message depending on conditional statement, then idles. If function exceeds the conditional statement, it prints out an error message. Once the function has completed the conditional statement, it exits.

i. proc5

This function prints out a message depending on conditional statement, then idles. If function exceeds the conditional statement, it prints out an error message. Once the function has completed the conditional statement, it exits.

4.1 Alarm.c

a. remove_alarm

This function removes any alarm from the alarm queue held by the alarm PCB and subsequently frees the memory attached to it. Allowing the addition of more alarms in the future.

b. set_alarm

This is the function the user interacts with as its the one that displays the prompts, takes in the user input, and stores the information in an alarm structure. This alarm is then saved inside a queue that the PCB will use to determine what alarms has been created in the check_alarm function

c. check_alarm

This function will check the current time and compare it to all the alarm structures inside the alarm queue. If the alarm has executed, then the message displays and the alarm is removed via the remove_alarm function from the queue. If the alarm has not been activated, then the alarm will be printed with a “not time yet” message. If all alarms are executed, then the alarm pcb exits.

4.2 colortext.c

d. Write_text(char str[ ])

This function takes in a character array, allocates memory for the array, prints given string to terminal using sys_req(WRITE...), then performs garbage collection automatically.

Parameters:

char str[ ]: character array of string the print

e. Write_text_red(char str[ ])

This function uses write_text to print a character array in the color red.

Parameters:

char str[ ]: character array of string the print

f. Write_text_bold_red(char str[ ])

This function uses write_text to print a character array in bold in the color red.

Parameters:

char str[ ]: character array of string the print

g. Write_text_green(char str[ ])

This function uses write_text to print a character array in the color green.

Parameters:

char str[ ]: character array of string the print

h. Write_text_bold_green(char str[ ])

This function uses write_text to print a character array in bold in the color green.

Parameters:

char str[ ]: character array of string the print

i. Write_text_yellow(char str[ ])

This function uses write_text to print a character array in the color yellow.

Parameters:

char str[ ]: character array of string the print

j. Write_text_bold_yellow(char str[ ])

This function uses write_text to print a character array in bold in the color yellow.

Parameters:

char str[ ]: character array of string the print

k. Write_text_blue(char str[ ])

This function uses write_text to print a character array in the color blue.

Parameters:

char str[ ]: character array of string the print

l. Write_text_bold_blue(char str[ ])

This function uses write_text to print a character array in bold in the color blue.

Parameters:

char str[ ]: character array of string the print

m. Write_text_magenta(char str[ ])

This function uses write_text to print a character array in the color magenta.

Parameters:

char str[ ]: character array of string the print

n. Write_text_bold_magenta(char str[ ])

This function uses write_text to print a character array in bold in the color magenta.

Parameters:

char str[ ]: character array of string the print

o. Write_text_cyan(char str[ ])

This function uses write_text to print a character array in the color cyan.

Parameters:

char str[ ]: character array of string the print

p. Write_text_bold_cyan(char str[ ])

This function uses write_text to print a character array in bold in the color cyan.

Parameters:

char str[ ]: character array of string the print

5.1 Memory_man.c

a. **Init_heap(u32int heap_size_param)**

This function initializes the heap for our system.

Parameters:

u32int heap_size_param: amount of bytes we want to initialize.

b. **Alloc_mem(u32int num_bytes)**

This function allocates the memory from the heap .

Parameters:

u32int num_bytes: represent the number of bytes to allocate.

c. **findCMCB(char name[])**

This function finds an allocated CMCB within our list by its name.

Parameters:

Char name[]: Name of the CMCB.

d. **Free_mem(void *addr)**

This function frees a block of memory that has been allocated.

Parameters:

Void *addr: *address or pointer to the data in memory that we want to free.*

e. **Merge_free_blocks(CMCB *freeblock)**

This finds any free blocks that needs merged and merges them. If not it just inserts a free block into the free list.

Parameters:

CMCB *freeblock: The block that we are trying to merge.

f. **IsEmpty()**

This function returns true or false based on whether the heap is empty .

g. **Show_free_mem()**

This function traverses the free memory list to display each CMCB's size, start address, and type.

h. **Show_alloc_mem()**

This function traverses the allocated memory list to display each CMCB's name, size, start address, and type.

i. **Unlink(CMCB* mcb)**

This function takes the block we are trying to manipulate or remove and unlinks it from the list.

Parameters:

CMCB* mcb: The block we are trying to unlink.

j. **Insert_mem(CMCB* mcb)**

This function inserts the memory block into a free or allocated list.

Parameters:

CMCB* mcb: The block we are wanting to insert.

5.2 Hex.c

k. **printDecToHex(int dec)**

This function takes a decimal number and converts it to hexadecimal. This allows us to print the start address of each block.

Parameters:

Int dec: the number we want to convert.

6.1 String.c

a. **Printf(char[] string, ...)**

This function takes in multiple parameters. The first parameter is always a string, then next parameters could be any of the following: int (%d), char (%c), string (%s). It takes the beginning string and and inserts the following variables into the string following each percent sign.

Parameters:

Char [ ]: string containing the %_ for each variable to insert

... = (int)(string)(char): int - integer variable to insert

Char[ ] - string variable to insert

Char - character variable to insert