==

PROGRAMMER'S MANUAL

==

This is the programmers manual page for the MPX OS Version R3&R4. It contains information necessary to manipulate the code and therefore the system.

This manual will be organized by file. The file name will be the section, and the contents will be the function header, description, parameters, and any returned values.

==

SERIAL.C

==

FUNCTION CALL:

    serial_poll();

DESCRIPTION:

Continuously polls (checks) the LSR (Line Status Register) for data and stores it one bit at a time in to a user-instantiated buffer, until either the buffer is full or a null terminator is encountered

PARAMETERS

    device dev - device to read from
    char *buffer - the user-provided buffer to write into
    size_t len - size of the buffer

RETURN

    size_t bytes_read - the amount of bytes read

COMMHAND.C

==

FUNCTION CALL:

    commhand();

DESCRIPTION:

Checks buffer from polling function to see if user input matches desired function input. If it does, proceed with command. If not, exit.

PARAMETERS

void

RETURN

void

ERRORS

“Invalid input, please try again.”

User’s input does not match any of the commands used to trigger the functions. Immediately allows the user to attempt to put the input command indefinitely.

“Buffer Overflow.”

User’s input exceeded the amount of space allocated in the buffer. Immediately allow the user to attempt to input the command indefinitely.

FUNCTION CALL:

help();

DESCRIPTION:

Provides instructions to the user upon request on how to operate the terminal.

PARAMETERS

void

RETURN

void

ERRORS

none

FUNCTION CALL:

shutdown

DESCRIPTION:

The "shutdown" function prompts the user to initiate a system shutdown by writing a message to a communication port (COM1), waits for user input, and performs system-wide cleanup and shutdown if the user inputs "yes." It provides feedback to the user and handles cases of "no" or invalid input.

PARAMETERS
	void

RETURN
	void

ERRORS
	none

FUNCTION CALL:
	get_time();

DESCRIPTION:
	Reads the binary coded decimal stored in the Real Time Clock Register 0x71.

PARAMETERS
	void

RETURN
	void

ERRORS
	none

FUNCTION CALL:
	set_time();

DESCRIPTION:
	Disables interrupts, writes a new value to the RTC register 0x71 using outb(), then re-enables interrupts

PARAMETERS
	void

RETURN
void

ERRORS
"Input buffer overflow."
User entered a time that caused overflow in the buffer. User has to re-enter the command and the time correctly. System immediately allows the user to try again indefinitely.
"Invalid input format."
User entered a time that was not HH:MM formatting, the hours or minutes were not within the range of 00-24 and 00-59 respectively.

FUNCTION CALL:
get_date();

DESCRIPTION:
Reads date value stored in system register

PARAMETERS
void

RETURN
void

ERRORS
none

FUNCTION CALL:
set_date();

DESCRIPTION:
Writes new value to date value stored in register

PARAMETERS
void

RETURN
void

ERRORS

“Input buffer overflow.”

User entered input that exceeded the amount of space in the buffer. System forces the user to re-enter the command and the time correctly, the cycle happens indefinitely.

“Invalid date format. Please use DD/MM/YY.”

User entered a date where the day, month and year were not within the set bounds of 01-31, 01-12 or 00-99. System assumes the year starts at 2000.

FUNCTION CALL:

delete_pcb(const char *name)

DESCRIPTION:

Finds the input process name and removes it from the queue using pcb_remove().

PARAMETERS

const char *name

RETURN

void

ERRORS

“Invalid PCB name.”

User attempted to delete a PCB with a name that did not fit into the requirements given for PCB names.

“PCB not found.”

User attempted to delete a PCB with a name that fit the requirements, but did not match the name of a PCB in the queue.

“System processes cannot be deleted.”

System cannot delete the process, immediately will return to the menu for the user to re-enter their desired commands indefinitely.

“Failed to free PCB memory.”

System cannot free the memory, immediately will return to the menu for the user to re-enter their desired commands indefinitely.

FUNCTION CALL:

block_pcb(const char *name);

DESCRIPTION:

Puts the input process into a blocked state, and then moves it to the blocked queue.

PARAMETERS

const char *name

RETURN

void

ERRORS

"Invalid process name."

User attempted to block a PCB with a name that did not fit into the requirements given for PCB names.

"Process not found."

User attempted to block a PCB with a name that fit the requirements, but did not match the name of a PCB in the queue.

FUNCTION CALL:

unblock_pcb(const char *name);

DESCRIPTION:

Takes input process, finds it in the blocked queue and moves it into the ready queue.

PARAMETERS

const char *name

RETURN

void

ERRORS

"Invalid process name."

User attempted to unblock a PCB with a name that did not fit into the requirements given for PCB names.

"Process not found."

User attempted to unblock a PCB with a name that fit the requirements, but did not match the name of a PCB in the queue.

FUNCTION CALL:
    suspend_pcb(const char* name);

DESCRIPTION:
    Suspends process by putting it in suspended queue and updating state number to match suspended state.

PARAMETERS
    const char* name

RETURN
    void

ERRORS
    "Invalid process name."
        User attempted to suspend a PCB with a name that did not fit into the requirements given for PCB names.
    "Process not found."
        User attempted to suspend a PCB with a name that fit the requirements, but did not match the name of a PCB in the queue.

FUNCTION CALL:
    resume_pcb(const char* name);

DESCRIPTION:
    Places the requested process into the ready state from being in the suspended queue.

PARAMETERS
    const char* name

RETURN
    void

ERRORS
    "Invalid process name."
        User attempted to resume a PCB with a name that did not fit into the requirements given for PCB names.
    "Process not found."

User attempted to resume a PCB with a name that fit the requirements, but did not match the name of a PCB in the queue.

FUNCTION CALL:

set_pcb_priority(const char* name);

DESCRIPTION:

Allows the user to set the PCB priority level between 0 and 9, and places the process appropriately in the queue.

PARAMETERS

const char* name
int new_priority

RETURN

void

ERRORS

"Invalid PCB name."

User attempted to resume a PCB with a name that did not fit into the requirements given for PCB names.

"Invalid priority, please assign a priority in range 0-9."

User attempted to assign a priority that was outside the range.

FUNCTION CALL:

show_pcb(const char* name);

DESCRIPTION:

Displays the requested process's name, class, state, suspended status and priority.

PARAMETERS

const char* name

RETURN

void

ERRORS

"Invalid PCB name."

User attempted to resume a PCB with a name that did not fit into the requirements given for PCB names.

“Invalid priority, please assign a priority in range 0-9.”

User attempted to assign a priority that was outside the range.

FUNCTION CALL:

show_ready(const char* name);

DESCRIPTION:

Displays the following information of all the processes in the ready state: name, class, state, suspended status and priority.

PARAMETERS

none

RETURN

void

ERRORS

none

FUNCTION CALL:

show_blocked(const char* name);

DESCRIPTION:

Displays the following information of all the processes in the blocked state: name, class, state, suspended status and priority.

PARAMETERS

none

RETURN

void

ERRORS

none

FUNCTION CALL:

    show_all(const char* name);

DESCRIPTION:

Displays the following information about all of the created processes, no matter their state: name, class, state, suspended status and priority.

PARAMETERS

    none

RETURN

    void

ERRORS

    none

FUNCTION CALL:

    alarm()

DESCRIPTION:

The "alarm" command creates independent processes that display user-defined messages at specified times. These processes idle until the specified time, ensuring late but not early triggering, and support concurrent execution for time-sensitive applications.

PARAMETERS

    void

RETURN

    void

ERRORS

    none

PCB.C

===

FUNCTION CALL:

pcb_free(struct pcb* old_pcb)

DESCRIPTION:

The "pcb_free" function is responsible for releasing memory associated with a Process Control Block (PCB) in a system. It first checks if the provided PCB pointer is not null. If it's not null, the function proceeds to free the memory allocated for the PCB's name (if it's not empty) and then frees the memory for the PCB structure itself. If the PCB pointer is null, an error message is written to a communication port (COM1), and the function returns an error code (1) to indicate that the PCB could not be found and freed

PARAMETERS

struct pcb* old_pcb

RETURN

int 0,1

ERRORS

PCB NULL

FUNCTION CALL:

pcb_find(struct pcb* old_pcb)

DESCRIPTION:

Searches all process queues for a process with the user defined name.

PARAMETERS

struct pcb* old_pcb

RETURN

void

ERRORS

none

FUNCTION CALL:

void pcb_insert()

DESCRIPTION:

Inserts a PCB into the appropriate queue based on the state and priority if the process is ready.

PARAMETERS

struct pcb* old_pcb

RETURN

void

ERRORS

none

FUNCTION CALL:

pcb_remove(struct pcb* old_pcb)

DESCRIPTION:

Removes a PCB from the current queue but does not free any associated memory or data structures.

PARAMETERS

struct pcb* old_pcb

RETURN

int 0,1

ERRORS

PCB NULL

===

SYS_CALL.C
==

FUNCTION CALL:

sys_call();

DESCRIPTION:

This function handles system calls related to process management. It checks the system call number in the provided context and performs actions accordingly. In the "IDLE" case, it ensures the process is ready for execution and inserts it into the ready queue. In the "EXIT" case, it removes and deallocates the currently running process. For other cases, it returns an error code. It then updates the context to the next process in the ready queue or returns the initial context if no other processes are ready.

PARAMETERS

Struct context* current_context - memory being allocated for the current context

RETURN

struct context*
==

SYS_CALL_ISR.S
==

FUNCTION CALL:

extern sys_call();

DESCRIPTION:

This assembly routine is the system call interrupt handler, designed to work with the sys_call C function for process management. It preserves registers and context, calls the C function, and then restores the state before returning from the interrupt. This facilitates the interaction between system calls and higher-level code.

PARAMETERS

NONE

RETURN

NONE