5x5 Operating system

Programmers Manual

File comhand.c

Function Description: Takes a PCB class as parameter and then returns the corresponding PCB class string for output.
Parameter: enum ProClass cls -> 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: enum ProState 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: enum ProState state -> A process dispatch state.
Returns: A human readable string representing the input dispatch state.

const char* dispatchstate_str(enum ProState state)

Enum Description: Contains color strings for sending over terminal to change the terminal text color for further output.

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)

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()

Function Description: Ask the user to put the time (hours, minutes, seconds) and then call setTime method in time.c file using while loops to handle errors.

int setTimeCommand()

Function Description: get the time from getTime() method

int getTimeCommand()

Function Description: Asks the user to put the date (month, day, year), and then call the method setDate in time.c. It uses loops to go through each asking for time to handle errors

int setDateCommand()

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

int getDateCommand()

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

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.

int helpCommand()

Function Description: Starts a prompt to turn the machine off, confirmed by sending a message to the user to enter 1.

int shutdownCommand()

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.

const struct str_pcbprop_map avail_pcb_class[]

Variable Description: Contains pairs of process execution state values and associated strings for output and input.

const struct str_pcbprop_map avail_pcb_execstate[]

Variable Description: Contains pairs of process dispatch state values and associated strings for output and input.

const struct str_pcbprop_map avail_pcb_dpatchstate[]

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)

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 block. If found, it will remove the PCB and set it to the block state before moving it to a ‘blocked’ 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 blocked.
Returns: 0 on success, 1 when the given process name was not found.

int blockPcbCommand()

Function Description: Command. Asks the user for the name of the process to unblock. If found, it will remove the PCB and set it to the ready state before moving it to a 'ready' 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 ready.
Returns: 0 on success, 1 when the given process name was not found.

int unblockPcbCommand()

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()

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()

File time.c

Function Description: Takes one integer parameter and converts it to BCD and then returns it.
Parameter: int integer -> An integer.
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: unsigned char 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: int hours -> Time in hours of a day (from 0 - 23).
Parameter: int minutes -> Time in minutes of an hour (from 0 - 59).
Parameter: int 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: int day -> Day of the month (must be from 1 to the last day of the provided month).
Parameter: int month -> Month of the year (must be from 1 to 12).
Parameter: int 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()

File serial.c

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)

File string.c

Function Description: Translates an integer to a null-terminated string.
Parameter: char 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

Struct Description: A process control block structure for maintaining process information for a process.
Field: char pname -> The current name of a process.
Field: var pcls -> The current class of a process.
Field: unsigned char ppri -> The current scheduling priority of a process. The value must be in [0-9], where 0 is the highest priority and increasing values indicate decreasing priority.
Field: pstate -> The current execution state of a process.
Field: pbp -> A pointer to the top of the stack for a process.
struct pcb

Struct Description: A node to hold queued PCB handles for a pcb_queue.
Field: pcb_elem -> A pointer to an existing PCB.
Field: p_next -> A pointer to the next entry in a queue. NULL if there is no next entry.
struct pcb_queue_node

Enum Description: Defines unique process state identifiers including independent execution and dispatch states.
enum ProcState

Enum Description: Defines unique process class identifiers.

enum ProcClass

Struct Description: A queue to hold queued PCB handles, linked list implementation.
Field: head -> If the queue is not empty, points to head/front of the queue which can be dequeued. NULL otherwise.
Field: 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)