OS Godz – R6 – Programmer's Manual

Nathan Fielding – Logan Yokum – Dylan Smith – Ethan Boddy

Structs

pcb

Struct to represent the process control blocks of the system. Each PCB has a character array to represent its name (name), an integer to represent its class (class), an integer to represent its priority (priority), an integer to represent its state (state), a void pointer to represent the top of the process stack (stack), a void pointer to represent the other processes in the stack (stack_ptr), and a struct pointer to represent the next **pcb** (next) as members.

mcb

Struct to represent the memory control blocks of the system. Each MCB has a void pointer to represent its starting memory address (start_addr), a void pointer to represent the size of the block (size), a **mcb** struct pointer to represent the previous MCB in the stack (prev), and a **mcb** struct pointer to represent the next MCB in the stack (next) as members.

iocb

Struct to represent the I/O control blocks of the system. Each IOCB has an op_code struct to represent the current operation read/write (cur_op), a character pointer to represent the ring buffer input (buffer), a size_t to represent the ring buffer index (buf_idx), a size_t to represent the size of the ring buffer (buf_len), a **pcb** struct pointer to represent the associated PCB (process), and an **iocb** struct pointer to represent the next IOCB in the IOCB queue (next) as members.

dcb

Struct to represent the device control blocks of the system. Each DCB has an integer to indicate its busy flag (busy_flag), an integer to indicate its open flag (open_flag), an integer to indicate its event flag (event_flag), an op_code struct to represent the current operation read/write (cur_op), a character pointer to represent the ring buffer input (buffer), a size_t to represent the size of the ring buffer (buf_len), an integer to represent the starting index of the ring buffer (buf_start), an integer to represent the terminating index of the ring buffer (buf_end), and an **iocb** struct pointer to represent the associated IOCB (iocb_queue) as members.

context

Struct to represent the context of a process. Each context has 15 unsigned 32 bit integer members to represent the segment registers, general registers, and control registers.

alarm_t

Struct to represent an alarm created within the system. Each alarm has an integer to represent the hour (hour), an integer to represent the minute (minute), an integer to represent the second (second), a character pointer to represent the message to describe if the alarm was set (message), and an alarm struct pointer to represent the next alarm in the list (next) as members.

Functions

void commhand()

Function to manage and execute commands based on input from the serial port. Takes in no parameters and the return type is void. Function will compare the input from the serial port to each of the standard commands (help, shutdown, date, time, version) and call that command if the input was verified. If the input does not match any valid command then an error message will be displayed and the user will be prompted to enter a valid command.

void help(char* args)

Function to display the list of commands that can be executed. The parameter is a character pointer that will determine which options and instructions are output. If there are no arguments passed into this function then it will print a numbered list of commands in the order of help, shutdown, time, date and version. Any input from the user will be compared to these commands and if a valid command is input then an instructional prompt will be displayed explaining how to use the selected command. If an invalid command is input then a message will be displayed prompting the user to enter a valid command.

int shutdown()

Function to exit the MPX system. No arguments should be passed into it and it will return an integer value indicating whether the shutdown was successful meaning the user typed shutdown twice after being prompted to confirm. A 1 will be returned if unsuccessful (too many arguments passed or no shutdown confirmation) and a 0 will be returned if the shutdown was successful. The input from the user is loaded into a character array, sanitized (trailing whitespace, extra arguments) and compared with the confirmation input before executing.

void time(char* args)

Function to read and write the time from the Real Time Clock. Takes in a character pointer to determine if the time is to be accessed or mutated. If no arguments are passed and a new line “\n” is detected then the current time will be displayed. The return type is void. The hours, minutes and seconds are accessed individually via the bytes from the RTC, converted to integers, and separated by commas before being output. If arguments are passed into this function then the input is parsed by the colon delimiters and each element (hours, minutes, seconds) is stored and checked for validity. If the arguments are valid then the new time is set. If the arguments are invalid then a message will be displayed containing valid input bounds and a prompt to retry.

void date(char* args)

Function to read and write the date from the Real Time Clock. Takes in a character pointer to determine if the date is to be accessed or mutated. If no arguments are passed and a new line “\n” is detected then the current date will be displayed. The return type is void. The month, day and year are accessed individually via the bytes from the RTC, converted to integers, and separated by commas before being output. If arguments are passed into this function then the input is parsed by the colon delimiters and each element (month, day, year) is stored and checked for validity. If the arguments are valid then the new date is set. If the arguments are invalid then a message will be displayed containing valid input bounds and a prompt to retry.

void version()

Function to output the current milestone (R1, R2, etc.) in the MPX development. Takes in no parameters and the return type is void.

int serial_poll(device dev, char* buffer, size_t len)

Function to read input from the serial port. Takes in a device struct to read the input from, a character pointer as the user-provided buffer, and a size_t as the length of the user-provided buffer. If successful then it will return the number of bytes read from the device as an integer. If unsuccessful then it will return a negative integer.

char* gettime()

Helper function used within time(char* args) to fetch the current time from the RTC. Does not take in any arguments and will return the current time as character pointer. The hours, minutes and seconds are accessed individually via the bytes from the RTC, converted to integers, and separated by commas before being output.

char* getdate()

Helper function used within date(char* args) to fetch the current date from RTC. Does not take in any arguments and will return the current date as a character pointer. The month, day and year are accessed individually via the bytes from the RTC, converted to integers, and separated by commas before being output.

char* itoa(int n)

Function to convert an integer into a null-terminated string. Takes in the integer to be converted and returns the string version of the integer as a character pointer.

int dtoh(int dec)

Function to convert a decimal number to a hexadecimal number. Takes in the decimal integer to be converted and returns the hexadecimal version as an integer.

int htod(int hex)

Function to convert a hexadecimal number to a decimal number. Takes in the hexadecimal integer to be converted and returns the decimal version as an integer.

int validnum(const char* s)

Function to check if a number is valid. Takes in a constant character pointer and checks each index of the string to verify that it is a numerical digit ranging from 0-9. If any of the digits fall out of this range then a 0 will be returned. If all digits are verified to be valid then a 1 is returned.

int println(const char* message)

Function to print a string on its own new line in output. Takes in a constant character pointer containing the string to be output and returns the length of the string as an integer.

void pcb_op(char* pcb_str)

Function to manage user commands associated with process control blocks. Takes in a character pointer as the command and returns nothing. The function compares the parameter string with each of the user process commands and determines which command to execute. If an invalid command is input then the user is prompted to enter a valid command.

void pcb_delete(const char* name)

Function to remove a process from the queue. Takes in a constant character pointer as the name of the process and returns nothing. The process name is used to find the process, check the suspension status, and finally remove it. If an invalid or nonexistent name is passed into it then an error message will be displayed saying that the system process could not be deleted or the process does not exist.

void pcb_suspend(const char* name)

Function to change the state of a process to suspended. Takes in a constant character pointer as the name of the process and returns nothing. The process name is used to find the process, verify that the status is not already suspended, and insert into the suspended queue. If an invalid or nonexistent process is passed into it then an error message will be displayed saying that the process is already suspended, the process cannot be suspended, or the process does not exist.

void pcb_resume(const char* name)

Function to ensure that a process is ready and not blocked or suspended. Takes in a constant character pointer as the name of the process and returns nothing. The process name is used to find the process, verify that the class status is not already ready, and insert into the ready queue. If an invalid or nonexistent process is passed into it then an error message will be displayed saying the process is not suspended, the process cannot be deleted, or the process does not exist.

void pcb_set_priority(const char* name, int priority)

Function to set the priority value of a process. Takes in a constant character pointer as the name of the process and an integer as the priority value to be set. The function returns nothing. The process name is used to find the process and the integer parameter is validated to be between 0 and 9. The new priority member is then set and the process is inserted into the appropriate queue. If an invalid or nonexistent process is passed into it then an error message will be displayed saying the priority value is invalid or the process does not exist.

void pcb_show_one(const char* name)

Function to display all of the members of a given process. Takes in a constant character pointer as the name of the process and returns nothing. The process name is used to find the process and access all of its members. sys_rec is then used to display each of the members of the process (name, class, priority, state). If an invalid or nonexistent process is passed into it then an error message will be displayed saying the process does not exist.

struct pcb* pcb allocate()

Function to allocate memory for a process. Takes in no parameters and returns the struct for which the memory was allocated. sys_alloc_mem is called to allocate memory. If the PCB pointer or the stack top pointer are null then the function will return null.

int pcb free(struct pcb* p)

Function to free a process from memory. Takes in a process as a struct pointer and returns an integer flag value to confirm that the memory was freed. sys_free_mem is called to free memory. If the parameter PCB pointer is null then the function will return null.

struct pcb* pcb setup(const char* name, int class, int priority)

Function to set up a struct with the appropriate members. Takes in a constant character pointer as the name, an integer to designate the class, and an integer to set the priority and returns the process as a struct. Memory is allocated for the process and each member (class, priority, state) is assigned. If the length of the name is greater than 16, the name already exists, the class is not between 0 and 2, or the priority is not between 0 and 9 then the function will return null. If memory cannot be allocated for the PCB then the function will return null.

struct pcb* pcb find(const char* name)

Function to find a process in the system. Takes in a constant character pointer as the name of the process and returns the process as a struct. list_find is called to check the head node for each possible state (ready, blocked, suspended/ready, suspended/blocked). If the PCB pointer created cannot be found in any of the state queues (ready, blocked, suspended/ready, suspended/blocked) then the function will return null.

void pcb insert(struct pcb* p)

Function to insert a process into the appropriate queue. Takes in a process as a struct pointer and returns nothing. Checks for the state of the process and calls list_insert to insert the process into the appropriate queue.

int pcb remove(struct pcb* p)

Function to remove a process from a queue. Takes in a process as a struct pointer and returns an integer flag to confirm that the removal was successful. Checks each state of the head process in the appropriate queue and calls list_remove to remove a node from the queue. If the PCB cannot be removed from any of the state queues then the function will return -1.

int list_free(struct pcb* head)

Function to free a process from memory. Takes in a process as a struct pointer and returns an integer to confirm that the memory was freed successfully. The function starts at the head node and checks each process until the appropriate node is found and the memory is freed. If the PCB cannot be found then the function will return -1 and it will return 0 if it can be found.

pcb *list_find(struct pcb* head, const char* name)

Function to find a process in a queue. Takes in a struct pointer as the head node and a constant character pointer as the name of the process. The function returns the process as a struct pointer. The function iterates through a queue to find the parameter process. If the PCB is found then it will return it. If the PCB is not found then it will return null.

void list_insert_ready(struct pcb** head, struct pcb* p)

Function to insert a process into the ready queue. Takes in a struct pointer to a pointer as the head node and struct pointer as the process. The function returns nothing. The queue is iterated through until the appropriate position is reached for the process to be inserted. If the head pointer of the queue is null then the function returns.

void list_insert_blocked(struct pcb** head, struct pcb* p)

Function to insert a process into the blocked queue. Takes in a struct pointer to a pointer as the head node and struct pointer as the process. The function returns nothing. The queue is iterated through until the appropriate position is reached for the process to be inserted. If the head pointer of the queue is null then the function returns.

int list_remove(struct pcb** head, struct pcb* p)

Function to remove a process from the queue. Takes in a struct pointer to a pointer as the head node and a struct pointer as the process. The function returns an integer flag value to confirm that the removal was successful. The queue is iterated through until the appropriate position is reached for the process to be removed. If the head pointer of the queue is null then the function will return 0. When the function reaches the end of the queue then it will return 1.

void pcb_show_ready()

Function to show the processes in a ready state. Takes in no parameters and returns nothing. The function calls sys_rec to display all of the processes associated with a ready state. If there are no ready processes then a message will be displayed saying as such.

void pcb_show_blocked()

Function to show the processes in a blocked state. Takes in no parameters and returns nothing. The function calls sys_rec to display all of the processes associated with a blocked state. If there are no blocked processes then a message will be displayed saying as such.

void pcb_show_all()

Function to display all of the processes in the system. Takes in no parameters and returns nothing. The function calls sys_rec to iterate through and display all of the processes in the system. If there are no processes in the system then a message will be displayed saying as such.

void yield()

Function to trigger the command handler to give up control of the CPU. Takes in no parameters and returns nothing. Passes IDLE into sys_req.

void loadr3()

Function to load the test processes from R3 into the non-suspended queue and initialize a context for each. Takes in no parameters and returns nothing. The context for each process is saved and inserted onto the top of the PCB and the registers are inserted accordingly.

void alarm_setup(char* time, char* message)

Function to create an alarm to be executed at a certain time. Takes in a character pointer as the time to set and character pointer as the message to display. The function returns nothing. If the input passed is not in HH:MM:SS format, if any characters that should be numeric are non-numeric, or if any of the digits are out of the range (0-23, 0-59, 0-59 for hour, minute, second, respectively) then a message is displayed detailing the error. If a valid time and message is input then the alarm is sent to **alarm_insert()** to be placed into the appropriate queue.

void alarm_insert(alarm_t* a)

Function to insert an alarm into the alarm list based on the time set within the alarm. Takes in an alarm_t as the alarm to be inserted. The function returns nothing.

int alarm_remove()

Function to remove the first alarm from the alarm list. Takes in no parameters and returns an integer value to indicate whether the removal was successful (0 for success, 1 for failure).

void alarm_exec()

Function to execute all of the scheduled alarms between the current time and previous time the function was called. Takes in no parameters and returns nothing.

context sys_call(context* c)

Function to find the next context to be loaded. Takes in a context struct pointer as the current process and returns the next context to be loaded. If the context sent into it as the parameter is null then a new created context is returned.

void kmain(void)

Function to direct control to the Command Handler after system components are initialized. Takes in no parameters and returns nothing.

sys_call_isr

Assembly function to act as an interrupt handler and to manage the stack of registers associated with each process. Calls **sys_call** after certain registers are pushed and then proceeds to pop certain registers from the stack.

void initialize_heap(size_t size)

Function to allocate all available memory as a single freed block to the memory manager. Takes in a size_t as the total size of the heap and returns nothing. Memory is allocated using **kmalloc()** and an MCB is created for the freed list.

void mcb_insert(mcb_t** head, mcb_t* m)

Function to insert a MCB into either the freed or allocated memory list. Takes in a mcb pointer to a pointer to as the beginning block of the memory list and a mcb pointer as the MCB to insert. The function returns nothing. If any null pointers are passed into the function then nothing is inserted.

int mcb_remove(mcb_t** head, mcb_t* m)

Function to remove a MCB from either the freed or allocated memory list. Takes in a mcb pointer to a pointer as the beginning block of the memory list and a mcb pointer as the MCB to remove. The function returns an integer flag value to indicate whether or not the removal was successful (0 for failure, 1 for success).

void* allocate_memory(size_t size)

Function to allocate memory from the heap. Takes in a size_t as the size of the allocation (in bytes). The function returns null if the size passed into it is less than 1 or greater than 50000 bytes or if the current MCB from the free list is null.

int free_memory(void* addr)

Function to free allocated memory. Takes in a void pointer as the starting address of an allocated block. The function returns an integer flag value to indicate whether or not the allocation was successful (0 for success, non-zero for failure).

void allocate(char* mem_size)

Function to allocate memory and print address of new memory block. Takes in a character pointer as the size of the allocation in decimal. The function returns nothing. The function calls **allocate_memory()** and prints an error message if the allocation fails (invalid/null memory size).

void free(char* address)

Function to free memory. Takes in a character pointer as the address of the memory block in hexadecimal. The function returns nothing. The function calls **free_memory()** and prints an error message if the free operation fails (invalid/null address).

void show_allocated()

Function to print the starting address in hexadecimal and size in decimal of each memory block in the allocated list. The function takes in no parameters and returns nothing.

void show_free()

Function to print the starting address in hexadecimal and size in decimal of each memory block in the freed list. The function takes in no parameters and returns nothing.

void serial_input_interrupt(struct dcb* dcb)

Function to read a device character and store it in either the ring buffer or the IOCB buffer. Takes in a dcb struct pointer as the device to read from. The function returns nothing. If the operation state of the DCB is read then the device character is stored in the appropriate IOCB queue. If the operation state of the DCB is write then the device character is stored in the ring buffer. If the passed DCB is invalid then the function returns.

void serial_output_interrupt(struct dcb* dcb)

Function to check for additional data within an IOCB queue and write the appropriate device character. Takes in a dcb struct pointer as the DCB to check. The function returns nothing. If the operation state of the DCB is not write then the function returns. If the IOCB associated with the device is null then the function returns.

void serial_interrupt()

Function to pause interrupts to check which device caused an interrupt and to determine if it was an input or output type. Takes in no parameters and returns nothing. Calls either **serial_output_interrupt()** or **serial_input_interrupt()** on the device depending on if the interrupt was input or output.

int serial_open(device dev, int speed)

Function to open a device serial port with a specific speed. Takes in a device to determine the corresponding serial port and an integer as the speed to be initialized to the serial port. The function returns an integer flag to indicate successful execution (-103 for already open, -1 for failure, 0 for success). If successful, serial port interrupts will be enabled.

int serial_close(device dev)

Function to close a device serial port. Takes in a device to determine the corresponding serial port to close. The function returns an integer flag to indicate successful execution (-201 for already closed, -1 for failure, 0 for success). If successful, serial port interrupts will be disabled.

int serial_read(device dev, char* buf, size_t len)

Function to read input from a serial port device. Takes in a device to read from, a character pointer as a buffer to read into, and a size_t as a byte read limit. The function returns an integer to indicate successful execution or to return the byte limit (-301 for closed device, -302 for a null buffer, -303 for byte size less than 1, -304 for busy device).

int serial_write(device dev, char* buf, size_t len)

Function to write data to a serial port device. Takes in a device to write into, a character pointer as a buffer to write from, and a size_t as a byte write limit. The function returns an integer to indicate successful execution (-401 for closed device, -402 for null buffer, -403 for byte size less than 1, -404 for busy device, 0 for success).

int iocb_dequeue(iocb** io_queue)

Function to remove an IOCB from the IOCB queue. Takes in an iocb struct pointer to a pointer as the IOCB queue. The function returns an integer value to indicate successful execution of the dequeue operation (-1 for failure/null queue, 0 for success).

void iocb_enqueue(iocb** io_queue, iocb* io)

Function to add an IOCB to the IOCB queue. Takes in an iocb struct pointer to a pointer as the IOCB queue and an iocb struct pointer as the IOCB to queued. The function returns nothing. If the passed IOCB queue is null then the IOCB to be queued is set as the head of the queue.

void serial_isr()

Function to access assembly code for interrupt service routine. Takes in no parameters and returns nothing.