Functions

- void init_comhand(void)
    - **Parameters**: N/A
    - **Returns**: N/A
    - **Description**: Will initiate the command handler function. Calls sys_req(READ) to get buffer user input from polling, executing code based on what the buffer reads.
- void comhand_version(void)
    - **Parameters**: N/A
    - **Returns**: N/A
    - **Description**: Will print the current project version to the user.
- void comhand_shutdown(void)
    - **Parameters**: N/A
    - **Returns**: N/A
    - **Description**: Will prompt the user for shutdown. If the user inputs 'yes', the system will initiate the shutdown procedure.
- void comhand_help(void)
    - **Parameters**: N/A
    - **Returns**: N/A
    - **Description**: Will print all currently implemented help files. Shows the user *all* commands the OS can run.
- void comhand_rtc(void)
    - **Parameters**: N/A
    - **Returns**: N/A
    - **Description**: Will print and display the current time of the real time clock to the user.
- void comhand_setTime(void)
    - **Parameters**: N/A
    - **Returns**: N/A
    - **Description**: Will prompt the user for input, asking them to enter a formatted (HH:MM:SS) text input that shows the
- getTime()
    - **Parameters**: N/A
    - **Returns**: char*
    - **Description**: will get time from ports, convert from BCD to decimal, and return as a char*

- getDate()
    - Parameters: N/A
    - Returns: char*
    - Description: gets the date from ports, converts it from bcd to int to char* and formats in it MM/DD/YY format
- int serial_poll(device dev, char* buffer, size_t len)
    - **Parameters**: device dev: the device to be read from. char* buffer: user-provided buffer. size_t len: the size of the buffer.
    - **Returns**: int the number of bytes read from the device, or a negative number on error
    - **Description**: Data is read one character at a time from the device's base register (same address as the device) using the inb() macro. When data is available, the least significant bit in the Line Status Register (device + LSR) is set
- setTime(char* newTime)
    - **Parameters**: char* newTime (containing new time to write)
    - **Returns**: N/A
    - **Description**: Takes parameter of char* and writes to the ports to set the new time
- setDate(char* newDate)
    - **Parameters**: char* newDate, the new date in MM/DD/YY format
    - **Returns**: N/A
    - **Description**: takes the date contained in newDate and writes to ports
- formatTime(int timeToFormat, char* stringPtr)
    - **Parameters**: int timeToFormat containing the time as an int read from the port. char* stringPtr as the pointer to the string to output the string to
    - **Returns**: N/A
    - **Description**: A helper function to getTime() that will take the output the int value from the ports and the pointer to the string to output, and will correctly format the time so that it shows up as ##.

- char* toupper(char* string)
    - **Parameters**: char* string (string to convert to uppercase)
    - **Returns**: New string, with each alphabetical character being uppercase.
    - **Description**: This function will turn whatever string is set as a parameter into an uppercase version of the same string. Returns a new string in memory that is fully uppercase.
- int isdigit(int c)
    - **Parameters**: int c, the character to check
    - **Returns:** 1 if c is a digit, 0 otherwise
    - **Description**: checks if c is a numeric ASCII character
- char* itoa(int i, char* dest)
    - **Parameters**: int i: the number to convert to ASCII, char* dest: location to store the resulting char* in
    - **Returns**: ASCII representation of i
    - **Description**: converts an integer to ASCII and stores the result in the memory address specified, as well as returning a reference to it.

- char* formatCore(char* format, …)
  - **Parameters**: char* format: string containing the desired string with placeholders for the rest of the arguments. Va_list: a variable length list of any elements that are to be inserted into the string.
  - **Returns**: the resulting string.
  - **Description:** takes the format string and replaces the placeholders with the specified elements
    - Use %s to insert a char*
    - Use %c to insert a char
    - Use %d or %i to insert an integer
    - Use %f to insert a float
    - Use %x to display an integer in hexadecimal
    - Use %o to display an integer in octal
    - Place a number between the % and the specified data type to specify the length of the string that is placed in the spot of the placeholder, if the padding size is shorter than the length of the string, the original length will be preserved. If a '.' is placed before the padding number, it will be padded with zeros as necessary.
    - For floats, the number after a decimal point determines the precision of the display, if this is not specified the precision is 6.
- Int sprintf(char* dest, char* format, …)
  - **Parameters**: dest: char* to store the result in. format: string containing placeholder for values, see formatCore for more details. va_list: a variable argument list of the elements to be inserted.
  - **Returns**: the length of the resulting string.
  - **Description**: wrapper that takes the result of formatCore and stores it in a char*. See formatCore description
- Void printf(char* format, …)
  - **Parameters**: format: string containing placeholder for values, see formatCore for more details. va_list: a variable argument list of the elements to be inserted.
  - **Returns**: N/A
  - **Description**: wrapper that takes the result of formatCore and prints it to the console. See formatCore description for formatting details.

- ➢ Void puts(char* buf)
    - **Parameters**: char* buf: a string.
    - **Returns:** N/A
    - **Description:** Prints the buf to the console.
- ➢ Void comhand_pcbCreate(void)
    - **Parameters**: N/A
    - **Returns:** N/A
    - **Description:** Assists the user in creating a new PCB with specific parameters.
- ➢ Void comhand_pcbDelete(void)
    - **Parameters**: N/A
    - **Returns:** N/A
    - **Description:** Assists the user in deleting a specific PCB.
- ➢ Void comhand_pcbSuspend(void)
    - **Parameters**: N/A
    - **Returns:** N/A
    - **Description:** Assists the user in changing a specific PCB to the [SUSPENDED] state
- ➢ Void comhand_pcbResume(void)
    - **Parameters**: N/A
    - **Returns:** N/A
    - **Description:** Assists the user in changing a specific PCB to the [NOT SUSPENDED] state
- ➢ Void comhand_pcbPriority(void)
    - **Parameters**: N/A
    - **Returns:** N/A
    - **Description:** Assists the user in changing a specific PCB's priority.
- ➢ Void comhand_pcbShow(int entry)
    - **Parameters**: int entry : Tells the function which type of show to perform.
    - **Returns:** N/A
    - **Description:** Shows either a specific PCB, all ready PCBs, all blocked PCBs, or all PCBs.

- Void comhand_pcbShowHelper(pcb* target)
  - **Parameters**: pcb* target the pcb to display
  - **Returns:** N/A
  - **Description:** Assists the comhand_pcbShow function in the print statements of PCBs.
- Void comhand_pcbprintPcbList(list* li)
  - **Parameters**: list* li the list to print
  - **Returns:** N/A
  - **Description:** iterates through the given pcb list, displaying them with comhand_pcbShowHelper
- list* getList(int entry)
  - **Parameters**: int entry : Specifies which list to return.
  - **Returns:** the user-specified list
  - **Description:** Returns a list specified by the int entry parameter.
- pcb* pcb_allocate(void)
  - **Parameters**: N/A
  - **Returns:** a pcb* to a new pointer, or NULL on error;
  - **Description:** allocates memory for a pcb and initialized the name char*, stack, and stackPtr
- int pcb_free(pcb* target)
  - **Parameters**: pcb* target the pcb* to free
  - **Returns:** 0 on success, non-zero on error
  - **Description:** frees the memory associated to the target pcb
- pcb* pcb_setup(const char* name, int class, int priority)
  - **Parameters**: const char* name the name for the new pcb
    - Int class USER or SYSTEM
    - Int priority the priority level of the new PCB, 0-9 from highest to lowest priority
  - **Returns:** the created PCB
  - **Description:** allocates a PCB with pcb_allocate and initializes it with the given values
- pcb* pcb_find(const char* name)
  - **Parameters**: const char* name, the name of the pcb to search for
  - **Returns:** the specified pcb, or NULL if not found
  - **Description:** searches through each list of Pcbs' to locate a pcb with the specified name

- void pcb_insert(pcb* newPCB)
    - **Parameters**: pcb* newPCB the pcb to add to the queue
    - **Returns:** N/A
    - **Description:** adds the given PCB to it's proper queue
- int pcb_remove(pcb* target)
    - **Parameters**: pcb* target the pcb to remove
    - **Returns:** 0 on success, non-zero on error
    - **Description:** removes the given pcb from its' queue without freeing associated memory
- int pcb_createcheck(char* procName)
    - **Parameters**: char* name for a new Pcb
    - **Returns:** a negative number if name is taken, 1 if the name is available
    - **Description:** searches for an existing PCB with the given name
- list* createList(void)
    - **Parameters:** N/A
    - **Returns:** list* pointer to new list
    - **Description:** creates and allocated memory for the linked list, and returns pointer
- node* createNode(void* data)
    - **Parameters:** void* data pointer for data to store in node
    - **Returns:** node pointer to node created
    - **Description:** creates and allocated memory for node and stores void pointer in parameter to data, and returns pointer to created node
- node* get(list* listPtr, int index)
    - **Parameters:** list* listPtr which is the pointer to the list, int index which is the index within the list
    - **Returns:** node pointer at index in list, NULL if DNE
    - **Description:** takes pointer to list and index as an int and will return the node pointer of node at index, or NULL if node DNE at index
- node* getHead(list* listPtr)
    - **Parameters:** list* listPtr pointer to list
    - **Returns:** node pointer of head of list
    - **Description:** returns node pointer of head of linked list within parameter

- node* add(list* listPtr, node* nodePtr)
    - **Parameters:** list* listPtr which is the pointer to list, node* nodePtr which is the pointer to the node to add
    - **Returns:** node pointer that was added to the list
    - **Description:** takes node pointer and adds to list in parameter and returns pointer to added node
- node* addToHead(list* listPtr, node* nodePtr)
    - **Parameters:** list* listPtr which is the pointer to the list, node* nodePtr which is the pointer to the node to add
    - **Returns:** node pointer that was added to the head of the list
    - **Description:** adds node pointer to the head of the list in the parameter and returns the pointer to the node added
- node* remove(list* listPtr, node* nodePtr)
    - **Parameters:** list* listPtr which is the pointer to the list, node* nodePtr which is the node to remove from the list
    - **Returns:** node pointer of node removed from the list
    - **Description**: finds and removes node from the list in the parameter, and returns the removed node
- node* removeHead(list* listPtr)
    - **Parameters:** list* listPtr which is the pointer to the list
    - **Returns:** node pointer of removed head from the list
    - **Description:** removes head of list in parameter, sets head to the next node in the list, and returns pointer to removed node
- int contains(list* listPtr, node* nodePtr)
    - **Parameters:** list* listPtr which is the pointer to the list, node* nodePtr which is the pointer to the node to find
    - **Returns:** int which is the nodes location in the list (headPtr == 0), of -1 if node DNE in list
    - **Description:** Finds location of node in list found in parameter, and returns int index of node, or -1 if node DNE in list
- void* getData(node*)
    - **Parameters:** node pointer
    - **Returns:** void pointer to data in node
    - **Description:** gets and returns the void pointer (data) within the node pointer

- context* sys_call(context* currentContext)
  - **Parameters:** pointer to context of current process
  - **Returns:** pointer to context of process to be loaded
  - **Description:** takes in the parameter of the current process, then returns the context of the process to be loaded
- void yield()
  - **Parameters:** N/A
  - **Returns:** N/A
  - **Description:** causes the command handler to yield to the CPU and will execute any processes in the queue
- void load()
  - **Parameters:** N/A
  - **Returns:** N/A
  - **Description:** loads all of the test processes for R3
- void comhand_load()
  - **Parameters:** N/A
  - **Returns:** N/A
  - **Description:** runs the load function in r3_commands to load the processes
- void comhand_alarm()
  - **Parameters:** N/A
  - **Returns:** N/A
  - **Description:** gets the information from the user including the alarm message and time and then creates and loads the alarm
- struct alarm* createAlarm(char* alarmName, char* time)
  - **Parameters:** alarmName which is the message for the alarm and time, which is the time for the alarm to be set to
  - **Returns:** alarm* which is a pointer to the created alarm structure
  - **Description:** takes the message and time, and allocates and creates the alarm, adds to the list of alarms, and returns the creates alarm pointer

- struct pcb* loadAlarm(void* function)
    - **Parameters:** function which is a void pointer to the function to be turned into a process
    - **Returns:** pointer to the pcb created
    - **Description:** will take in the function, and will then create a pcb with that function of the alarm, and return the pointer to the created pcb for the alarm
- void runAlarm()
    - **Parameters:** N/A
    - **Returns**: N/A
    - **Description:** is the function used for the alarm to see if the alarm should go off or not
- void removeAlarm(alarm* alarm)
    - **Parameters:** N/A
    - **Returns:** N/A
    - **Description:** goes through the list of alarms and finds the alarm to remove, removes the alarm, and frees the allocated memory to that alarm
- int compareTime(char* alarmTime)
    - **Parameters:** alarmTime which is the time of the alarm
    - **Returns:** an int value that shows if the alarm should go off or not. 1 meaning no, 0 meaning yes.
    - **Description:** compares the alarm time to the current time and determines if the alarm should go off or not
- void extractingTime(char* array[], char* time)
    - **Parameters:** char* array[] which is an array of chars that is already allocated and char* time which is the time to be extracting
    - **Returns**: N/A
    - **Description:** will take the time and separates into hours, minutes, and seconds, and sets it to the array in that order
- int free_memory(void* data)
    - **Parameters:** void* data which is data to be freed
    - **Returns**: 0 on success, 1 on failure
    - **Description:** Will take in a void pointer that corresponds to a memory address, clearing that memory from the system.

- void initialize_heap(size_t data)
    - **Parameters:** size_t data which indicates what size to make a heap of memory.
    - **Returns**: N/A
    - **Description:** Will create a memory heap block that is stored for future memory usage.
- void* allocate_memory(size_t data)
    - **Parameters:** size_t data which indicates what size to make an allocated block of memory.
    - **Returns**: Void pointer to location of allocated memory
    - **Description:** Will allocate memory from the free blocks, if available.
- int hexToInt(char* hexNumber)
    - **Parameters:** char pointer which has the hexadecimal number to be converted
    - **Returns:** int value that the decimal value of the hexadecimal number
    - **Description:** converts hexadecimal to integer
- int power(int x, int y)
    - **Parameters:** x int value that is the base number and y int value that is the power
    - **Returns:** the value of x to the y power
    - **Description:** gets the x to the y power
- void comhand_allocateMem
    - **Parameters:** N/A
    - **Returns:** N/A
    - **Description:** Prompts the user to enter an amount of memory they want to allocate on the system, and gives them the hexadecimal memory address of that allocated memory.
- void comhand_freeMem
    - **Parameters:** N/A
    - **Returns:** N/A
    - **Description:**Prompts the user to enter a hexadecimal address of memory they want to free on the system, and frees that memory.

- void comhand_showMemory(int entry)
  - **Parameters:** entry int to select if memory shown is freed memory or allocated memory.
  - **Returns:** N/A
  - **Description:** Prints out each block of memory currently freed or allocated, specified by the int entry.
- mcb* getHeadMcb(void)
  - **Parameters:** N/A
  - **Returns:** mcb* head of the memory control block list
  - **Description:** Returns the head of the memory control block list
- int serial_open(device dev, int speed)
  - **Parameters:** device dev the serial port to open, int speed the desired port speed
  - **Returns:** -1 on failure, 0 on success
  - **Description:** Initializes the port specified by device with the speed specified by speed.
- int serial_close(device dev)
  - **Parameters:** device dev the serial port to close
  - **Returns:** 201 on failure, 1 on success
  - **Description:** Will end a serial port session of specified device.
- int serial_read(device dev, char *buf, size_t len)
  - **Parameters:** device dev the serial port to read, char *buf the data to read, size_t len the amount of bytes to read
  - **Returns:** 301 if device is closed, 302 if invalid buffer address, 303 if invalid count, 1 if success
  - **Description:** Will read input from specified serial device and place characters in its associated buffer, reading no more than len bytes
- int serial_write(device dev, char *buf, size_t len)
  - **Parameters:** device dev the serial port to write to, char *buf the data to write, size_t len the amount of bytes to write
  - **Returns:** 401 if device is closed, 302 if invalid buffer address, 403 if invalid count, 404 if device is in use, 1 on success
  - **Description:** Will write data specified by *buf to serial device specified by dev.

- void serial_interrupt(void)
    - **Parameters:** N/A
    - **Returns:** N/A
    - **Description:** Handles I/O interrupts. Checks the type of interrupt and invokes the proper secondary handler.
- void serial_input_interrupt(dcb *dcb1)
    - **Parameters:** N/A
    - **Returns:** N/A
    - **Description:** secondary handler for input interrupts. If the device is currently writing, it writes the character input to the requestor buffer, otherwise it places it in the ring buffer for the device. Will trigger completion upon receiving a new line character.
- void serial_output_interrupt(dcb *dcb1)
    - **Parameters:** N/A
    - **Returns:** N/A
    - **Description:** Secondary handler for write I/O requests. If the current index of the buffer is less than the size, writes a byte, otherwise it triggers completion.
- void schedule_io(pcb* process, op_code op, device dev, char *buffer, size_t size)
    - **Parameters:**
        - Process: the pcb of the process requesting an I/O operation
        - Op: the operation being requested
        - Dev: the device for the request
        - Buffer: the buffer for the request
        - Size: the size of the buffer;
    - **Returns:** N/A
    - **Description:** creates an iocb for the I/O request and places it in the dcb. If the device is available, it calls serial read or write, otherwise it places the iocb in the waiting queue.
- void io_complete(void)
    - **Parameters:** N/A
    - **Returns:** N/A
    - **Description:** checks for I/O completion and moves any complete I/O requests back into the ready queue.

- alloc_status check_device_status(device dev)
    - **Parameters:** device dev: device to check status of.
    - **Returns:** the use status of the device, dev
    - **Description:** checks the use status of the device

<u>New data types</u>

- pcb : struct
  - char* name: name for the pcb
  - Int class: USER or SYSTEM
  - Int priority: priority value from 0-9, 0 being the highest priority
  - Int executionState: READY, RUNNING, or BLOCKED
  - Int dispatchingState: SUSPENDED or NOT_SUSPENDED
  - Char[] stack: 1024 byte array
  - char*stackPtr: pointer to the last few bytes of the stack
- list : struct
  - node * headPtr: pointer to the head node of the list
- node : struct
  - node* prevPtr: pointer to the previous node in the list
  - node* nextPtr: pointer to the next node in the list
  - void* data: void pointer to the data stored in the node
- context: struct
  - int gs: register
  - int fs: register
  - int es: register
  - int ds: register
  - int ss: register
  - int EDI: register
  - int ESI: register
  - int EBP: register
  - int EBX: register
  - int EDX: register
  - int ECX: register
  - int EAX: register
  - int EIP: register
  - int CS: register
  - int EFLAGS: register

- alarm: struct
  - char* alarmName: message for the alarm
  - char* alarmTime: time that the alarm is set for
- mcb: struct
  - size_t start_address: address where mcb is located
  - size_t size: defined size of a memory control block
  - struct mcb* nextPtr: pointer to the next mcb in the list
  - struct mcb* prevPtr: pointer to the previous mcb in the list
  - enum mem_flag flag: FREE or ALLOCATED
- ring_buffer: struct
  - char arr[16]: array kept in ring buffer memory
  - size_t size: current size of ring buffer
  - int head: current starting index of ring buffer
  - int tail: current ending index of ring buffer
- dcb: struct
  - device assoc_dev: serial port associated with dcb
  - alloc_status use_status: indicates whether device is being used
  - event_status event_status: indicates whether device has an event
  - op_code cur_op: indicates the current event of the device
  - struct iocb* iocb_head: head of the iocb list associated with device
  - ring_buffer* buffer: ring buffer associated with device
  - char* char_buffer: character buffer used by device
  - size_t buffer_len: current length of the buffer associated with device
  - size_t buffer_progress: current place in buffer associated with device
- iocb: struct
  - struct iocb* nextPtr: pointer to next iocb in list
  - pcb* assoc_pcb: pcb associated with iocb
  - dcb* assoc_dcb: dcb associated with iocb
  - size_t buffer_index: current placement in associated buffer
  - char* buffer: buffer associated with iocb
  - size_t buffer_len: current length of the buffer associated with iocb
  - op_code op_type: the operation to be performed by iocb