# ECE 354: Project Part 2

## RTX Initialization

RTX Initialization does three main initializations, processes, priority queues, and interrupts. The process initialization is called in `init_processes()`. All the test processes have a PCB setup for them which is added to the process of list of the kernel. This goes through all processes in the system and initializes their stack pointers and sets up an entry point to each process by putting an exception frame on its stack. At this point no process is running. When initializing priority queues all processes are added to their respective priority queue. When initializing interrupts an interrupt handler for the system call is installed at autovector level 0.

```
void init_processes(){
    init_test_procs(){
        for each test proc{
            setup PCB;
            add to proccess list;
        }
    }

    for each process in process list{
        initialize stack pointer;
        put exception frame on stack;
    }
}

void init_priority_queues(){
    for each process{
        add process to correct priority queue;
    }
}

void init_interrupts(){

    initialize VBR into real memory;
    install system call ISR at vector 0;
}
```

## Release Processor

This function allows the kernel to make a scheduling decision by receiving alerts that a process is finished running for the time being. It is initially called in `rtx.c` and executed in `k_release_processor()` in `kernel.c`. Initially all process are in a stopped state (STATE_STOPPED) until they are run. The priority queues with two queues for each priority level, a ready queue and a done queue. Initially all processes are in the ready queue for their respective priority level. After it has been run it is moved into the done queue for its priority level. After all processes have been run (all ready queues are empty) the done queues are copied to their respective ready queues and this process begins from the start.

```
int release_processor(){
    mode switch to kernel mode;

    //retrieve next process from process queue
    nextProcess = dequeue next process from highest
        non-empty priority queue;

    if(proccess is running){

        Enqueue running process to the appropriate done priority queue;

        perform context switch{
            save all of the currently running
                processes registers to its stack;
            save stack pointer to PCB;
            set process.state = STATE_READY;
        }
    }
        switch to nextProccess{
            load the selected proccesses stack;
            if(nextProcess.state == STATE_STOPPED){
                //at this point the processes stack contains only an
                //exception frame pointing to entry point of processs
                rte;
            }
        }
    }



    if (context switch is successful){
        put RTX_SUCCESS into return value register;
        mode switch to user mode;
        return contents of return value register;
    }else{
        put RTX_ERROR into return value register;
        mode switch to user mode;
        return contents of return value register;
    }
}
```

## Priority Set and Get

The get and set priority functions have the ability to read and update the priority of a process given a specific priority ID. The program switches to kernel mode, makes any necessary reads or writes, and then returns to user mode and gives the result. These functions are initially called in `rtx.c` with the majority of the work happening in `kernel.c`.

```
int set_process_priority(int pid, int priority) {

    read arguments into registers;
    enter kernel mode via exception;
    read variables from registers;

    if(pid or priority is not valid){
        put RTX_ERROR into return value register;
        mode switch to user mode;
        return contents of return value register;
    }

    set priority to the priority argument value;
    move process to correct process queue;
    put RTX_SUCCESS into return value register;
    mode switch to user mode;
    return contents of return value register;
}

int get_process_priority(int pid){

    read arguments into registers;
    enter kernel mode;
    read variables from registers;

    if(pid is not valid){
        put RTX_ERROR into return value register;
        mode switch to user mode;
        return contents of return value register;
    }

    retrieve process from process list;
    put process priority into return value register;
    mode switch to user mode;
    return contents of return value register;
}
```

## Null Process

The null process was implemented to adhere to the project specifications. It has a priority of four and a process ID of zero. The process method is defined in `system_process.c` as an infinite loop that constantly calls `release_processor()`. It is added to the process table through a call to `init_null_process()` found in init.c. This call should always be the first call made to add processes to the process tables as it assumes that it is at the 0th index of the process table array. This will be fine as the initialization of this process is handled by the main function of the RTX.