Christopher Richard Bilger
CIS 3207 – Professor Kwatny

Lab #1

The project that I built is essentially a simulator for simple computer processes, and the timing of them. For each of the data structures I will explain how they work and why I chose to implement them the way that I did. I will explain the same details about the main loop design, enum EVENT struct and testing that occurred during the duration of the project.

- **Data Structures**
  - Priority Queue (Array) – The initialize function creates a struct of type priority_queue and initializes it will NULL values. After this, it is given a variable amount of memory that is equal to the number of elements that should be maximally stored in the queue multiplied by the size of each element (priority_node) that will be in the queue. Other notable functions are the push_to_priority_queue function as well as the pop_from_priority_queue function. They are both very straight forward apart from lines 41-45 and 63-64. Lines 41-45 take the node that was last pushed onto the *stack* and swaps it with the previous top of the stack, and then iterates over this while decrementing the node number, until the newly entered node is in the correct priority position in the queue. Lines 63-64 do a similar process except all nodes end up swapping one position lower so that the queue is in the correct state after retrieving a node from it.
  - FIFO Queue (Linked-List) – The aptly named FIFO queue, or first-in first-out queue, works in a significantly different way that the priority queue does. The FIFO queue has a variable, essentially limitless memory size because every node that is added to the queue is allocated dynamically in memory and then *linked* to the previous node. The createQueue function

works in a very similar way to the initialize function of the priority queue, returning a pointer to the newly created FIFO queue. The enQueue and deQueue functions are actually very straightforward where a new node of “int job” is created and then linked to the previous node with line 49, which is “q->rear->next = temp;”, or a node is removed from the front, the front is changed to a different node, and then the node that was removed in returned.

- Configuration and Config Structs – The configuration file is very simple, whereas the parsing of it looks very messy. It is actually only one loop that runs and gathers all of the *parsed* data from the CONFIG.ini file. The line in reference is line 14 of config.c. Lines 14-17 iterate over the file one line at a time, find the location of the deliminator, “=”, and then parse the data after the deliminator until the line ends. The use of the C standard library “string.h” is how the “strstr” function is able to find the location of a substring.

- **Main Loop**
  - The main loop is the heart of this program. It ties together all of the data structures, enum, and various other functions to create the simulator. The simulator works on an event system which is simply a finite automata that has 10 states. The 10 states are stored in the enum labeled “EVENT” which is explained below. The loop pops a node from the priority queue and reads the EVENT enum associated with the node. After reading this, the event is put through a switch statement to identify that should happen based on that information given in the node. Once the switch case for that event is finished processing, the loop moves to the final portion which is the actual data processing. The data processing occurs in the process_cpu, process_disk1, and process_disk2 functions. Once each of those functions have been called, the loop starts from the beginning by pulling out another node from the

priority queue and reading it. If the node pulled from the priority queue is of EVENT "SIM_ENDING", then the simulation finished by printing out statistics for the program. All the while, each EVENT prints out information for the user to handle. The best way to handle all of the output is to use the command "filename > LOG.txt" which will neatly log all of the simulations information.

- **EVENT Enum**
  - The EVENT enum stores all of the possible states of the simulator. There are 10 possible states which are SIM_STARTING, SIM_ENDING, ARRIVED, FINISHED, CPU_ARRIVED, CPU_FINISHED, DISK1_ARRIVED, DISK1_FINISHED, DISK2_ARRIVED, and DISK2_FINISHED. They are all indicative of what they're functions are except for, I think, ARRIVED and FINISHED. ARRIVED is a state that is when a job is going to the CPU queue but has not yet arrived in the CPU_ARRIVED state. It's a pre-state for CPU_ARRIVED. FINISHED is a final state for a job. That is, FINISHED is the post-state for all CPU jobs that quit immediately before going through another loop of disk processing.
- **Testing**
  - The testing is well documented in the "tests.c" file which includes detailed comments on how the tests were written and performed.

Function Declaration List:

- struct priority_queue initialize_priority_queue(int number_of_elements);
- void destroy_priority_queue(struct priority_queue *queue);
- void swap_nodes(struct priority_node *a, struct priority_node *b);

- _Bool push_to_priority_queue(struct priority_queue *queue, int priority, int job, enum EVENT event);
- struct priority_node pop_from_priority_queue(struct priority_queue *queue);
- _Bool priority_queue_is_empty(struct priority_queue *queue);
- _Bool priority_queue_is_full(struct priority_queue *queue);
- void print_priority_queue(struct priority_queue *queue);
- _Bool process_cpu(struct Queue *queue, struct priority_queue *p_queue, _Bool idle, int current_time);
- _Bool process_disk1(struct Queue *queue, struct priority_queue *p_queue, _Bool idle, int current_time);
- _Bool process_disk2(struct Queue *queue, struct priority_queue *p_queue, _Bool idle, int current_time);
- int generate_int(int minimum, int maximum);
- _Bool probability_select(int percent);
- struct QNode* newNode(int k);
- struct Queue *createQueue();
- void destroyQueue(struct Queue *queue);
- void enQueue(struct Queue *q, int job);
- struct QNode *deQueue(struct Queue *q);
- struct Configuration read_config_file(char *filename);
- void print_config_struct(struct Configuration config);
- int main();

Christopher Richard Bilger
CIS 3207 – Professor Kwatny

Finite State Automata Diagram: