Round Robin Project
CSCI 330-M02
Operating Systems
Kendall Molas

CPU Scheduling

CPU scheduling is a necessary for dealing with processes that constantly enter the operating system. CPU scheduling deals with processes that run on the CPU to execute its instructions. There are several different scheduling algorithms such as:

1. First Come First Serve (FCFS)
2. Shortest Job First (SJF)
3. Priority Scheduling
4. Round Robin

The goals of CPU Scheduling is as follows:

1. CPU is constantly busy.
2. Maximize throughput of the processes
3. Minimize turnaround time
4. Minimize waiting time
5. Minimize number of context switches

In this assignment, the Round Robin scheduling algorithm was focused on solely. The algorithm of Round Robin goes as follows, processes will enter the operating system one by one and be put into the ready queue to wait for CPU to be available. When CPU is available, the process will run for some amount of time. This time that the process will run on the CPU is called the time quantum.

Implementation

For this implementation of Round Robin, I used C. There are three many structures: Queue, CPU, and Process. The queue structure contains the array size, front item, rear item, original size, and its current size. This queue would be filled after read csv method is executed. The CPU structure contains information that would be used for displaying the final results of the algorithm at the end. Some of the information includes total waiting time and total context switches. Lastly, the process structure contains information of the arrival time, burst time, turnaround time, etc.

After the queue is filled with the processes read from the csv file, it would go into execution. It would constantly loop through the queue as long as the front process is not null. During the loop cycles, it would subtract the burst time from the time quantum that the user will input. After the difference between the burst time and the time quantum is computed, it would check if the burst time is zero. If it is zero, it would compute other information such as the waiting time and turnaround time. Information such as the waiting time and turnaround time are then passed to the CPU variables for later analysis. If burst time is not zero, the process' context switch would then be incremented by one. The process would then be dequeued, and it would repeat the process.

Analysis of Output Data

In regards to the actual output data, the CPU was constantly busy as shown in the screenshots. Other information was outputted as well including the total time it took to run based off the time quantum, the total idle time, average waiting time, total throughput, and the total amount of context switches. These times were determined based off the system clock which I included through the header time.h.

Time Quantum = 5:

```
kui-io@kuio ~/Documents/Programming/OS $ ./output pid.csv
Enter the time quantum of your choice: 5
Your time quantum is: 5
Size is: 4
Processes loaded into ready queue:
Process id#: 1 | Arrival Time: 0 | Burst Time 5
Process id#: 3 | Arrival Time: 0 | Burst Time 3
Process id#: 2 | Arrival Time: 1 | Burst Time 7
Process id#: 4 | Arrival Time: 2 | Burst Time 6

Processing...
Running on CPU... Burst Time Decremented...
Running on CPU... Burst Time Decremented...
Running on CPU... Burst Time Decremented...
Running on CPU... Burst Time Decremented...
Running on CPU... Burst Time Decremented...

Process id#: 1 | Arrival Time: 0 | Burst Time 0
Process id#: 3 | Arrival Time: 0 | Burst Time 3
Process id#: 2 | Arrival Time: 1 | Burst Time 7
Process id#: 4 | Arrival Time: 2 | Burst Time 6

Current time in seconds: 5.00s
Process Completion Time: 5s
Process Waiting Time: 0s
Process Turnaround time: 5s
Removing from queue

Current total waiting time: 0
Process has been removed from queue
Idle time of process is: 0
Process ran for time quantum of: 5
Original size of the queue was: 4
Current size of the queue is: 3
After dequeue, state of processes looks as follows.
Process id#: 3 | Arrival Time: 0 | Burst Time 3
Process id#: 2 | Arrival Time: 1 | Burst Time 7
Process id#: 4 | Arrival Time: 2 | Burst Time 6

Processing...
Running on CPU... Burst Time Decremented...
Running on CPU... Burst Time Decremented...
Running on CPU... Burst Time Decremented...

Process id#: 3 | Arrival Time: 0 | Burst Time 0
Process id#: 2 | Arrival Time: 1 | Burst Time 7
Process id#: 4 | Arrival Time: 2 | Burst Time 6
```

Current time in seconds: 8.00s
Process Completion Time: 3s
Process Waiting Time: 5s
Process Turnaround time: 8s
Removing from queue

Current total waiting time: 5
Process has been removed from queue
Idle time of process is: 0
Process ran for time quantum of: 5
Original size of the queue was: 4
Current size of the queue is: 2
After dequeue, state of processes looks as follows.
Process id#: 2 | Arrival Time: 1 | Burst Time 7
Process id#: 4 | Arrival Time: 2 | Burst Time 6

Processing...
Running on CPU... Burst Time Decremented...
Running on CPU... Burst Time Decremented...
Running on CPU... Burst Time Decremented...
Running on CPU... Burst Time Decremented...
Running on CPU... Burst Time Decremented...

Process id#: 2 | Arrival Time: 1 | Burst Time 2
Process id#: 4 | Arrival Time: 2 | Burst Time 6

Current time in seconds: 13.00s
Context switch...
Idle time of process is: 0
Process ran for time quantum of: 5
Original size of the queue was: 4
Current size of the queue is: 2
After dequeue, state of processes looks as follows.
Process id#: 4 | Arrival Time: 2 | Burst Time 6
Process id#: 2 | Arrival Time: 1 | Burst Time 2

Processing...
Running on CPU... Burst Time Decremented...
Running on CPU... Burst Time Decremented...
Running on CPU... Burst Time Decremented...
Running on CPU... Burst Time Decremented...
Running on CPU... Burst Time Decremented...

Process id#: 4 | Arrival Time: 2 | Burst Time 1
Process id#: 2 | Arrival Time: 1 | Burst Time 2

Current time in seconds: 18.00s

```
Context switch...
Idle time of process is: 0
Process ran for time quantum of: 5
Original size of the queue was: 4
Current size of the queue is: 2
After dequeue, state of processes looks as follows.
Process id#: 2 | Arrival Time: 1 | Burst Time 2
Process id#: 4 | Arrival Time: 2 | Burst Time 1

Processing...
Running on CPU... Burst Time Decremented...
Running on CPU... Burst Time Decremented...

Process id#: 2 | Arrival Time: 1 | Burst Time 0
Process id#: 4 | Arrival Time: 2 | Burst Time 1

Current time in seconds: 20.00s
Process Completion Time: 7s
Process Waiting Time: 12s
Process Turnaround time: 19s
Removing from queue

Current total waiting time: 17
Process has been removed from queue
Idle time of process is: 0
Process ran for time quantum of: 5
Original size of the queue was: 4
Current size of the queue is: 1
After dequeue, state of processes looks as follows.
Process id#: 4 | Arrival Time: 2 | Burst Time 1

Processing...
Running on CPU... Burst Time Decremented...

Process id#: 4 | Arrival Time: 2 | Burst Time 0

Current time in seconds: 21.00s
Process Completion Time: 6s
Process Waiting Time: 13s
Process Turnaround time: 19s
Removing from queue

Current total waiting time: 30
Process has been removed from queue
Idle time of process is: 0
Process ran for time quantum of: 5
Original size of the queue was: 4
Current size of the queue is: 0
```

After dequeue, state of processes looks as follows.

Done processing

This took 21.00s
Unit idle time:0.00s
CPU Utilization: 100.00%
Average Waiting Time: 7.00s
Total throughput: 0.19
Total amount of context switches: 2

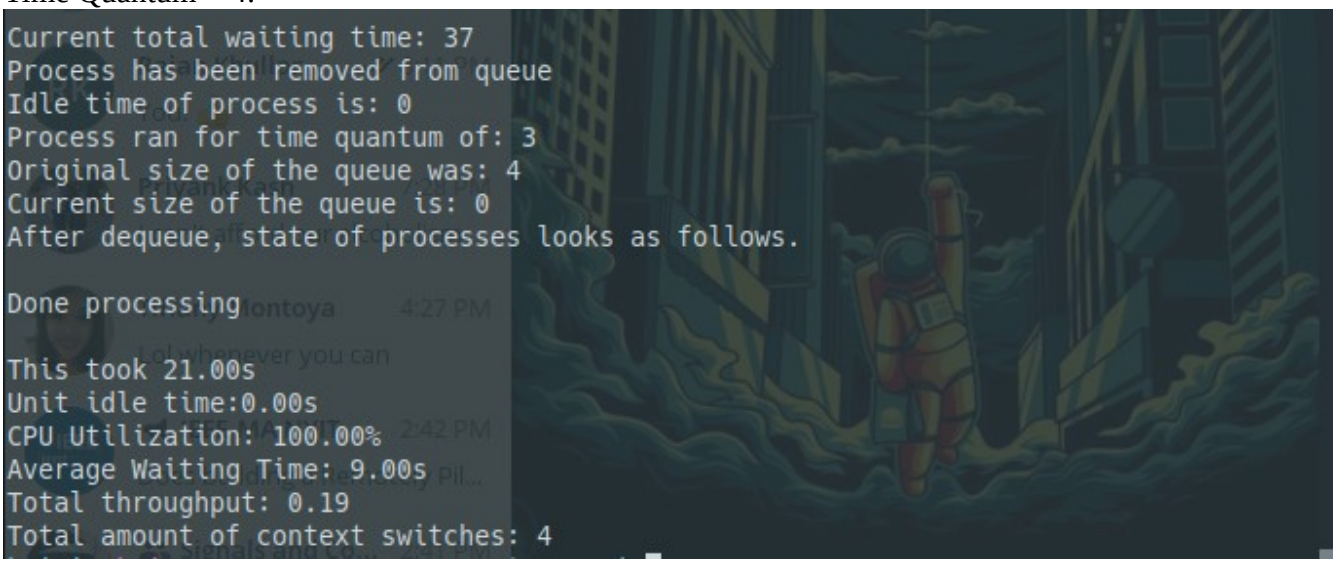
Time Quantum = 2:

```
Current total waiting time: 44
Process has been removed from queue
Idle time of process is: 0
Process ran for time quantum of: 2
Original size of the queue was: 4
Current size of the queue is: 0
After dequeue, state of processes looks as follows.

Done processing

This took 21.00s
Unit idle time:0.00s
CPU Utilization: 100.00%
Average Waiting Time: 11.00s
Total throughput: 0.19
Total amount of context switches: 8
```

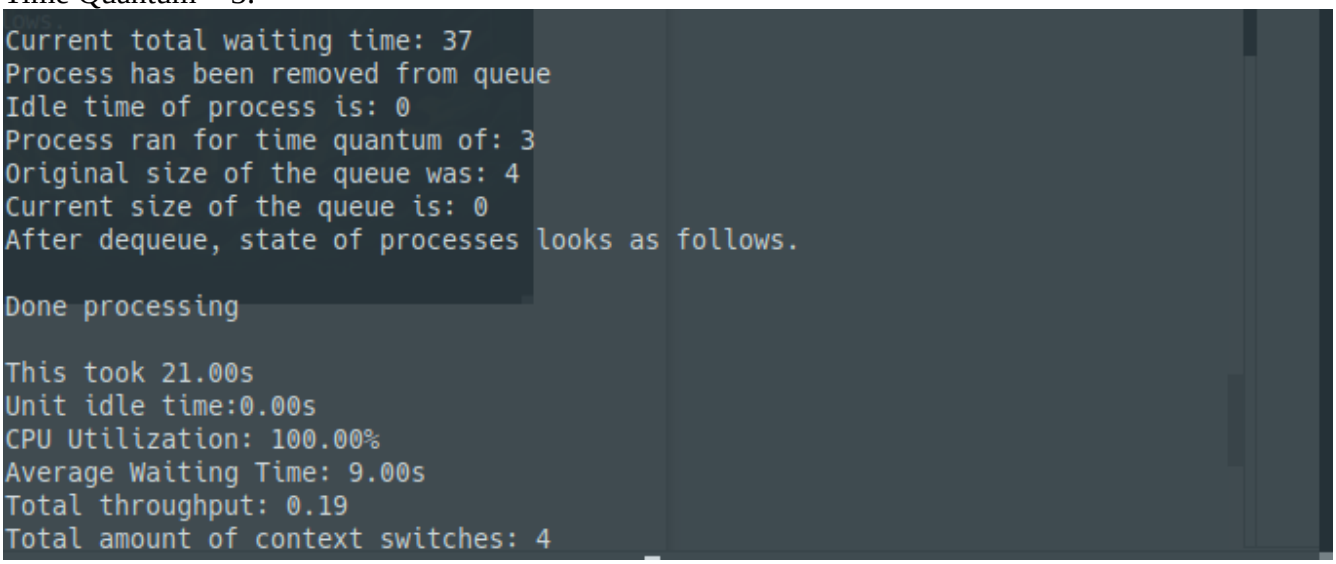
Time Quantum = 10:

```
Current total waiting time: 25
Process has been removed from queue
Idle time of process is: 0
Process ran for time quantum of: 10
Original size of the queue was: 4
Current size of the queue is: 0
After dequeue, state of processes looks as follows.

Done processing

This took 21.00s
Unit idle time:0.00s
CPU Utilization: 100.00%
Average Waiting Time: 6.00s
Total throughput: 0.19
Total amount of context switches: 0
```

Time Quantum = 4:



Time Quantum = 3:



```
// Kendall Molas
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <time.h>

typedef struct processes Process;
typedef struct queue Queue;
typedef struct cpu CPU;
unsigned time_quantum;

void delay(int ms);
void read_csv(Queue *q, char *file);
void enqueue(Queue *q, Process *p);
void dequeue(Queue *q, CPU *processor, time_t *exec_time);
void getCurrentState(Queue *q);
void start_scheduler(CPU *cpu, Queue *q, unsigned *tq);
void calculate_process_info(Process *p, int *exec_time);
void analysis(CPU *unit, Queue *ready_queue);

double cpu_maximum_utilization(CPU *cpu);
int check_current_size(Queue *q);
int compute_burst_time(Process *p, unsigned *tq);
Process *process_pids(unsigned *pid, unsigned *arrival, unsigned *burst);

//-----------------------------------------------------------------------------------------------
/* Structures:
 *    - Process
 *    - Queue
 *  - CPU */
//-----------------------------------------------------------------------------------------------

typedef struct processes {
   unsigned pid;
   unsigned arrival_time;
   unsigned burst_time;
   unsigned waiting_time;
   unsigned completion_time;
   unsigned turnaround_time;
   unsigned context_switch_count;
} Process;

typedef struct queue {
    Process *arr[4];     // Pointer to the array as we don't know what array we're specifying
    int front;
    int rear;
    size_t original_size, current_size;
} Queue;
```

```
typedef struct cpu{
	unsigned total_waiting_time;
	unsigned total_completion_time;
	unsigned total_context_switches;
	time_t total_time_running, idle_time;
} CPU;

//-----------------------------------------------------------------------------------------------
// Process Methods
//-----------------------------------------------------------------------------------------------

// Prints out the completion time, turnaround time, and waiting time for the process.
void calculate_process_info(Process *p, int *exec_time) {

	// Calculate Turnaround time & Waiting time
	p->turnaround_time = *exec_time - p->arrival_time;
	//printf("Exec time: %d and Arrival Time %d\n", *exec_time, p->arrival_time);
	p->waiting_time = p->turnaround_time - p->completion_time;

	printf("Process Completion Time: %ds\nProcess Waiting Time: %ds\nProcess Turnaround time:
%ds\n"
	"Removing from queue\n\n", p->completion_time, p->waiting_time, p->turnaround_time);
}

//-----------------------------------------------------------------------------------------------
// CPU Methods
//-----------------------------------------------------------------------------------------------
double cpu_maximum_utilization(CPU *unit) {
	printf("Unit idle time:%.2fs\n", (double) unit->idle_time);
	return ((double) (unit->total_time_running - unit->idle_time) / (double) (unit-
>total_time_running)) * 100;
}

//-----------------------------------------------------------------------------------------------
/* Queue methods
 * - Check current size of queue
 * - Enqueue
 * - Dequeue
 * - Get current state of queue
 * - Sort queue in ascending order based off of arrival time*/
//-----------------------------------------------------------------------------------------------

// This function returns amount of items currently in the ready queue
// @returns queue size
int check_current_size(Queue *q) {
   return q->current_size;
}

// This functions adds incoming processes or already processed process into queue
```

```
void enqueue(Queue *q, Process *process) {
	size_t size = sizeof(q->arr)/sizeof(q->arr[0]);
	unsigned currentSize = check_current_size(q);

	if (q->arr[currentSize] == NULL && currentSize < size) {
	  q->arr[currentSize] = process;
	  q->current_size++;
	}
	q->original_size = size;
}

// This function either:
// Shift processes in array by 1 and move front process to the end if process is not null OR
// Dequeues process if process' burst time is zero
// Problems: Does not make it null
void dequeue(Queue *q, CPU *processor, time_t *start_exec) {
	q->front = 0;
	q->rear = q->current_size-1;
	Process *tempProcess = q->arr[q->front];
	time_t start, end;

	start = clock();

	// Check if the process in front of the queue is not considered NULL
	// Move front process to the end
	if (tempProcess != NULL) {
		for (size_t i = 0; i < q->current_size-1; i++) {
			q->arr[i] = q->arr[i+1];
		}
		q->arr[q->rear] = tempProcess;
	}
	// Since it's null, dequeue it permanently
	else {
		for (size_t i = 0; i < q->current_size-1; i++) {
			q->arr[i] = q->arr[i+1];
		}
		q->current_size--;
		printf("Process has been removed from queue\n");
	}

	// Get end clock of when context switch ends...
	end = clock();

	// Then assign the difference divided by the clocks per second to the idle time.
	float val = (float) (end - *start_exec) / CLOCKS_PER_SEC;

	processor->idle_time += val;
	printf("Idle time of process is: %d\n", processor->idle_time);
}
```

```
// After the program runs for specified time quantum, check the current state of all processes
void getCurrentState(Queue *q) {
	for (size_t i = 0; i < q->current_size; i++) {
		printf("Process id#: %d | Arrival Time: %d | Burst Time %d\n", q->arr[i]->pid,
		q->arr[i]->arrival_time, q->arr[i]->burst_time);
	}
}

// Sorts the queue based on the arrival time of the read processes from the csv file
void sortQueue(Queue *q) {
	for (size_t i = 0; i < q->current_size; i++) {
		for (size_t j = 0; j < q->current_size-1; j++) {
			if (q->arr[j+1]->arrival_time < q->arr[j]->arrival_time) {
				Process *p = q->arr[j];
				q->arr[j] = q->arr[j+1];
				q->arr[j+1] = p;
			}
		}
	}
}

//-----------------------------------------------------------------------------------------------
// Methods needed for starting Round Robin CPU Scheduling
// Process the processes in the ready queue with specified time quantum, etc
// @returns pointer to process with specified id, arrival time, and burst time
//-----------------------------------------------------------------------------------------------
Process *process_pids(unsigned *p_id, unsigned *arrival, unsigned *burst) {
    Process *p;
    p = malloc(sizeof(Process));
    p->pid = *p_id;
    p->arrival_time = *arrival;
    p->burst_time = *burst;
    p->turnaround_time = 0;
    p->completion_time = 0;
    p->waiting_time = 0;
    return p;
}

// Read csv file and check each character for an int.
// that int is then passed to the process in the queue array.
// @input - file name
void read_csv(Queue *ready_queue, char *fileName) {

    char c;
    char c_arr[4];
    FILE *file; // Pointer to file to be read
    unsigned pidFound = 0, arrivalTimeFound = 0, burstTimeFound = 0; // Acts as boolean
    unsigned process_id, arrival_time, burst_time;
```

```
    unsigned index = 0;

    file = fopen(fileName,"r"); // Read the csv file

        // Check if file exists and if it does, continue reading ntil end of file
        if (file) {
            while ((c = getc(file)) != EOF) {
                if (isdigit(c)) {     // Check if character read is a number

                    if (c != ',') {

                        // Read each number and store it after recognizing the ','
                        if (pidFound == 0 && (arrivalTimeFound == burstTimeFound)) {
                            process_id = c - '0';    // Needed to turn read character in a number
                            pidFound = 1;
                        }
                        else if (arrivalTimeFound == 0 && pidFound == 1) {
                            arrival_time = c - '0';
                            arrivalTimeFound = 1;
                        }
                        else if (burstTimeFound == 0 && (arrivalTimeFound == pidFound)) {
                            burst_time = c - '0';
                            burstTimeFound = 1;
                        }
                    }
                }

                // Begin enqueueing the processes
                else if (c == '\n' && (pidFound == 1 && burstTimeFound == 1 && arrivalTimeFound ==
1 )) {
                    enqueue(ready_queue, process_pids(&process_id, &arrival_time, &burst_time));
                    pidFound = 0, arrivalTimeFound = 0, burstTimeFound = 0;
                    index++;
                }
            }
            fclose(file);
        }
}

// Start Round Robin Scheduling Algorithm
void start_scheduler(CPU *processor, Queue *ready_queue, unsigned *time_quantum) {
    Process *newProcess = ready_queue->arr[ready_queue->front];
    time_t context_switch_start;
    while (newProcess != NULL) {
        if (newProcess->burst_time > 0) {
            printf("Processing...\n");

            // Run process on CPU
            newProcess->burst_time = compute_burst_time(newProcess, time_quantum);
```

```
            // Get time at which process finishes running on CPU
            time_t end_execution = clock() / CLOCKS_PER_SEC;

            printf("\n");

            // Get time for how long CPU is idle
            context_switch_start = clock();
            getCurrentState(ready_queue);
            printf("\n");
            printf("Current time in seconds: %.2fs\n", (double) (clock() - processor-
>total_time_running) / CLOCKS_PER_SEC);

            // Check if the burst time is zero
            if (newProcess->burst_time == 0) {

                // Fill in processes' turnaround time, completion time, etc.
                int execution_time = end_execution - (int) processor->total_time_running;
                calculate_process_info(newProcess, &execution_time);

                // If this process had some type of context switch event, add number of context
switches
                // to total amount that occured.
                if (newProcess->context_switch_count != 0)
                    processor->total_context_switches += newProcess->context_switch_count;

                // Add process' waiting time to total waiting time
                processor->total_waiting_time += newProcess->waiting_time;
                printf("Current total waiting time: %d\n", processor->total_waiting_time);

                // Make that item null now
                ready_queue->arr[ready_queue->front] = NULL;
            }
            else {
                printf("Context switch...\n");
                newProcess->context_switch_count += 1;
            }
            dequeue(ready_queue, processor, &context_switch_start);
            printf("Process ran for time quantum of: %d\nOriginal size of the queue was: %d\n"
            "Current size of the queue is: %d\nAfter dequeue, state of processes looks as follows.\n",
            *time_quantum, ready_queue->original_size,check_current_size(ready_queue));
            getCurrentState(ready_queue);

            // Set the process equal to whatever is in front now
            newProcess = ready_queue->arr[ready_queue->front];
            printf("\n");
        }
    }
}
```

```
// This function returns burst time after computing difference between process' burst time and the
// time quantum itself
int compute_burst_time(Process *p, unsigned *time_quantum) {
	for (size_t i = *time_quantum; i > 0; i--) {
		if (p->burst_time != 0) {
			p->burst_time -= 1;
			p->completion_time += 1;
			printf("Running on CPU... Burst Time Decremented...\n");
			delay(1000);
		}
	}
	return p->burst_time;
}

// Return CPU Utilization, average waiting time, throughput, and # of occurances of context switching
void analysis(CPU *unit, Queue *ready_queue) {
	// Get end duration of the clock and subtract the end from the beginning to get the total run time
	time_t end = clock() / CLOCKS_PER_SEC;
	unit->total_time_running = end - unit->total_time_running;
	double average_waiting_time = unit->total_waiting_time / ready_queue->original_size;

	printf("This took %.2lfs\n", (double)unit->total_time_running);
	printf("CPU Utilization: %.2f%\n", cpu_maximum_utilization(unit));
	printf("Average Waiting Time: %.2fs\n", average_waiting_time);
	printf("Total throughput: %.2f\n", ready_queue->original_size / (double) (unit-
>total_time_running));
	printf("Total amount of context switches: %d\n", unit->total_context_switches);
}

//-----------------------------------------------------------------------------------------------
// Simulations Tools
// * Delay - Allows a small delay depending on input
//-----------------------------------------------------------------------------------------------

// @input - Amount of milliseconds that it should be delayed for.
void delay(int ms) {
   long halt;
   clock_t current, previous;

   halt = ms*(CLOCKS_PER_SEC/1000);
   current = previous = clock();

   while((current-previous) < halt )
      current = clock();
}

int main(int argc, char *argv[]) {
```

```
    // Initialize items in ready_queue to be null, and the variables in CPU to be 0.
    Queue ready_queue = {NULL};
    CPU processing_unit = {0, 0, 0, 0 ,0};

    // Read the argument, which is a csv file.
    read_csv(&ready_queue, argv[1]);

    if (argc != 2) {
            printf("Please input a csv file when running the command ./output [insert file name here]\n");
            return 0;
      }
      sortQueue(&ready_queue);

      printf("Enter the time quantum of your choice: ");
      scanf("%d", &time_quantum);
      printf("Your time quantum is: %d\n", time_quantum);
      printf("Size is: %d\nProcesses loaded into ready queue:\n", check_current_size(&ready_queue));

      getCurrentState(&ready_queue);
      printf("\n");

      processing_unit.total_time_running = (int) clock() / CLOCKS_PER_SEC;
      start_scheduler(&processing_unit, &ready_queue, &time_quantum);
      printf("Done processing\n\n");

      analysis(&processing_unit, &ready_queue);
}
```