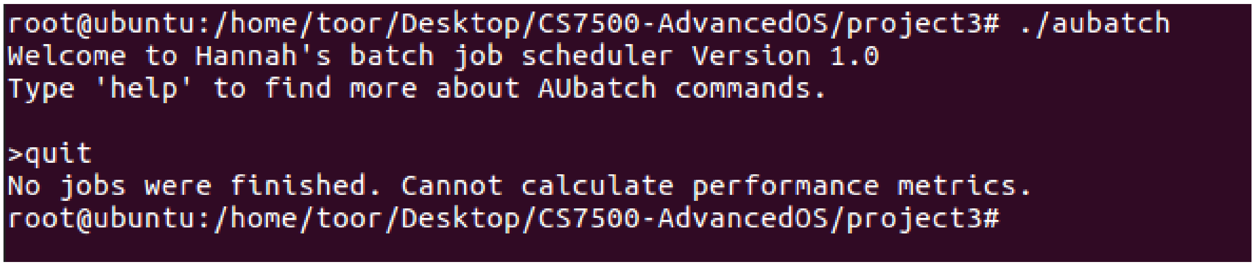
Project 3 Report

Name: Hannah Reinbolt
Date: 3-14-2021
Assignment: Project 3
Class: CS7500

1.1 Project and Design of AUbatch

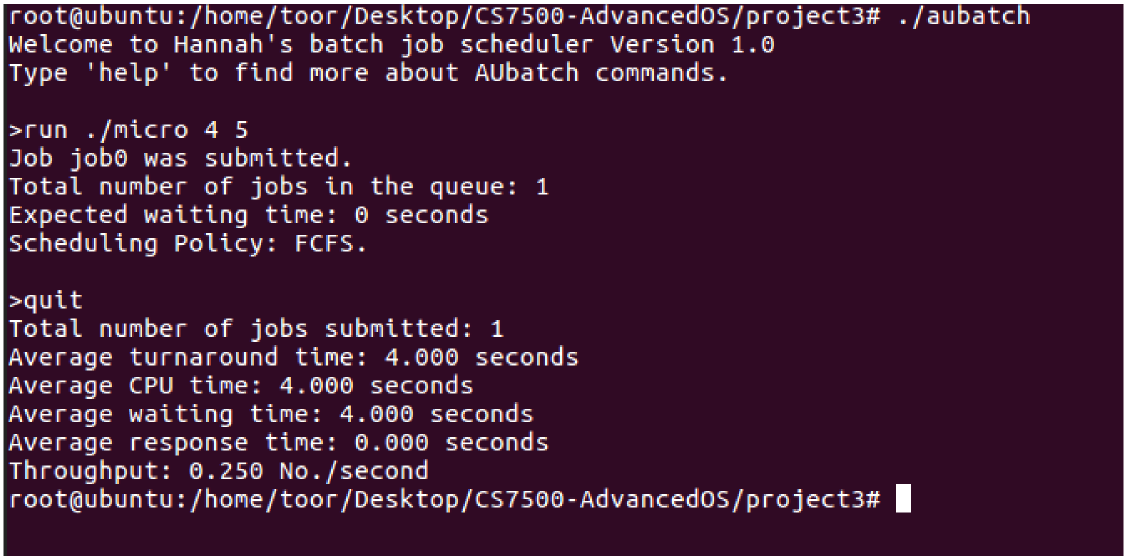
The AUbatch system project design is function-oriented, including a wide range of functions that help run the scheduling, dispatching, and command line functions. These functions are broken up into three main files; scheduling, which contains the scheduling and dispatching functionality; commandline, which contains the commandline functionality; and aubatch, which contains the main program that calls the scheduler and dispatcher from two threads. The program synchronizes with global variables located in the main aubatch function. The commandline functionality is called and runs forever until the user specifies quit. The "help" command can show more information about how to use the commandline. Here is an example of the start, help and quit menus.

Diagram: Start Menu and Quit without Running a Job



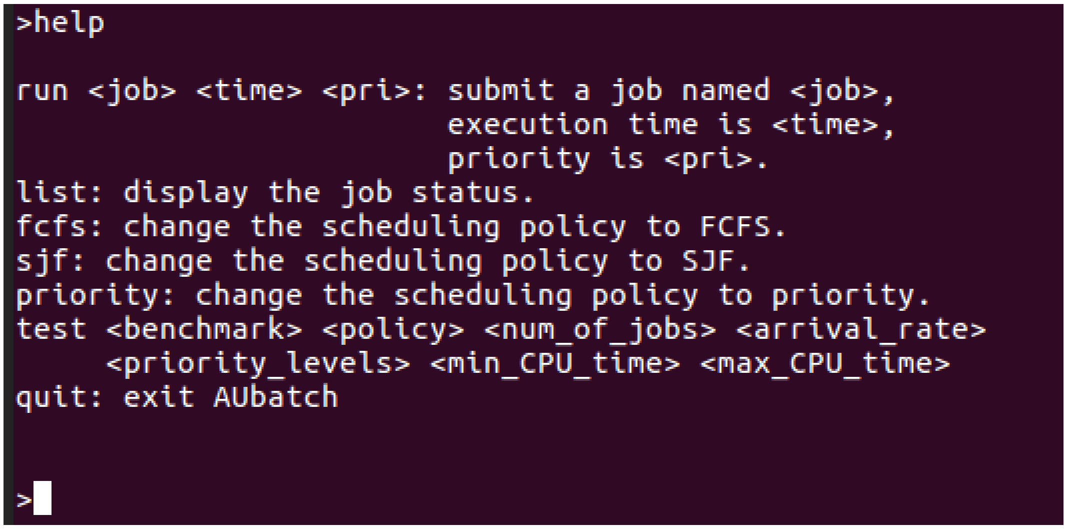
```
root@ubuntu:/home/toor/Desktop/CS7500-AdvancedOS/project3# ./aubatch
Welcome to Hannah's batch job scheduler Version 1.0
Type 'help' to find more about AUbatch commands.

>quit
No jobs were finished. Cannot calculate performance metrics.
root@ubuntu:/home/toor/Desktop/CS7500-AdvancedOS/project3#
```

Diagram: Quit after Running a Job and Job has Finished



```
root@ubuntu:/home/toor/Desktop/CS7500-AdvancedOS/project3# ./aubatch
Welcome to Hannah's batch job scheduler Version 1.0
Type 'help' to find more about AUbatch commands.

>run ./micro 4 5
Job job0 was submitted.
Total number of jobs in the queue: 1
Expected waiting time: 0 seconds
Scheduling Policy: FCFS.

>quit
Total number of jobs submitted: 1
Average turnaround time: 4.000 seconds
Average CPU time: 4.000 seconds
Average waiting time: 4.000 seconds
Average response time: 0.000 seconds
Throughput: 0.250 No./second
root@ubuntu:/home/toor/Desktop/CS7500-AdvancedOS/project3#
```

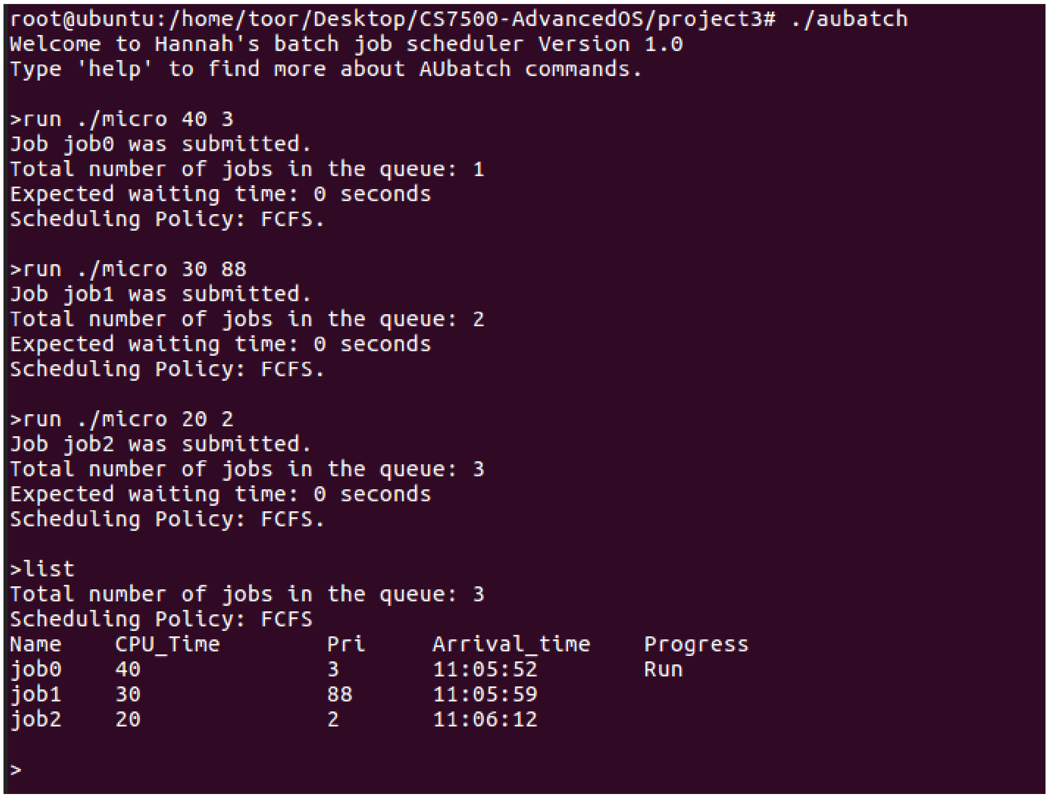
Diagram: Help Menu



```
>help

run <job> <time> <pri>: submit a job named <job>,
                        execution time is <time>,
                        priority is <pri>.
list: display the job status.
fcfs: change the scheduling policy to FCFS.
sjf: change the scheduling policy to SJF.
priority: change the scheduling policy to priority.
test <benchmark> <policy> <num_of_jobs> <arrival_rate>
     <priority_levels> <min_CPU_time> <max_CPU_time>
quit: exit AUbatch


>
```

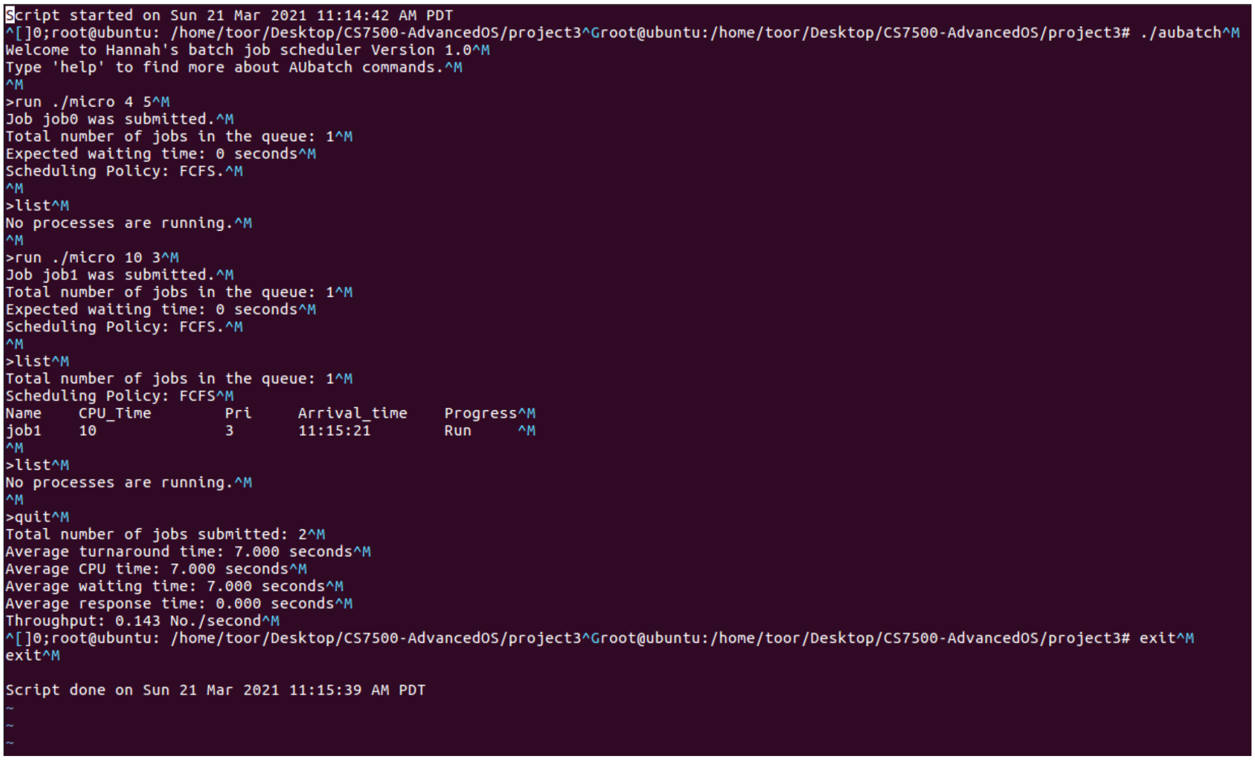
Jobs are specified on the AUbatch commandline in the format of “run <microbatch> <cpu time> <priority>”. This will execute the microbatch program selected for “cpu time” amount of seconds. These jobs can be shown in the commandline with the list function. See below for an example of job creation and the job list menu.

Diagram: Job creation and List



```
root@ubuntu:/home/toor/Desktop/CS7500-AdvancedOS/project3# ./aubatch
Welcome to Hannah's batch job scheduler Version 1.0
Type 'help' to find more about AUbatch commands.

>run ./micro 40 3
Job job0 was submitted.
Total number of jobs in the queue: 1
Expected waiting time: 0 seconds
Scheduling Policy: FCFS.

>run ./micro 30 88
Job job1 was submitted.
Total number of jobs in the queue: 2
Expected waiting time: 0 seconds
Scheduling Policy: FCFS.

>run ./micro 20 2
Job job2 was submitted.
Total number of jobs in the queue: 3
Expected waiting time: 0 seconds
Scheduling Policy: FCFS.

>list
Total number of jobs in the queue: 3
Scheduling Policy: FCFS
Name    CPU_Time        Pri     Arrival_time    Progress
job0    40              3       11:05:52        Run
job1    30              88      11:05:59
job2    20              2       11:06:12

>
```

Diagram: Sample Script Tool Input/Output from AUbatch



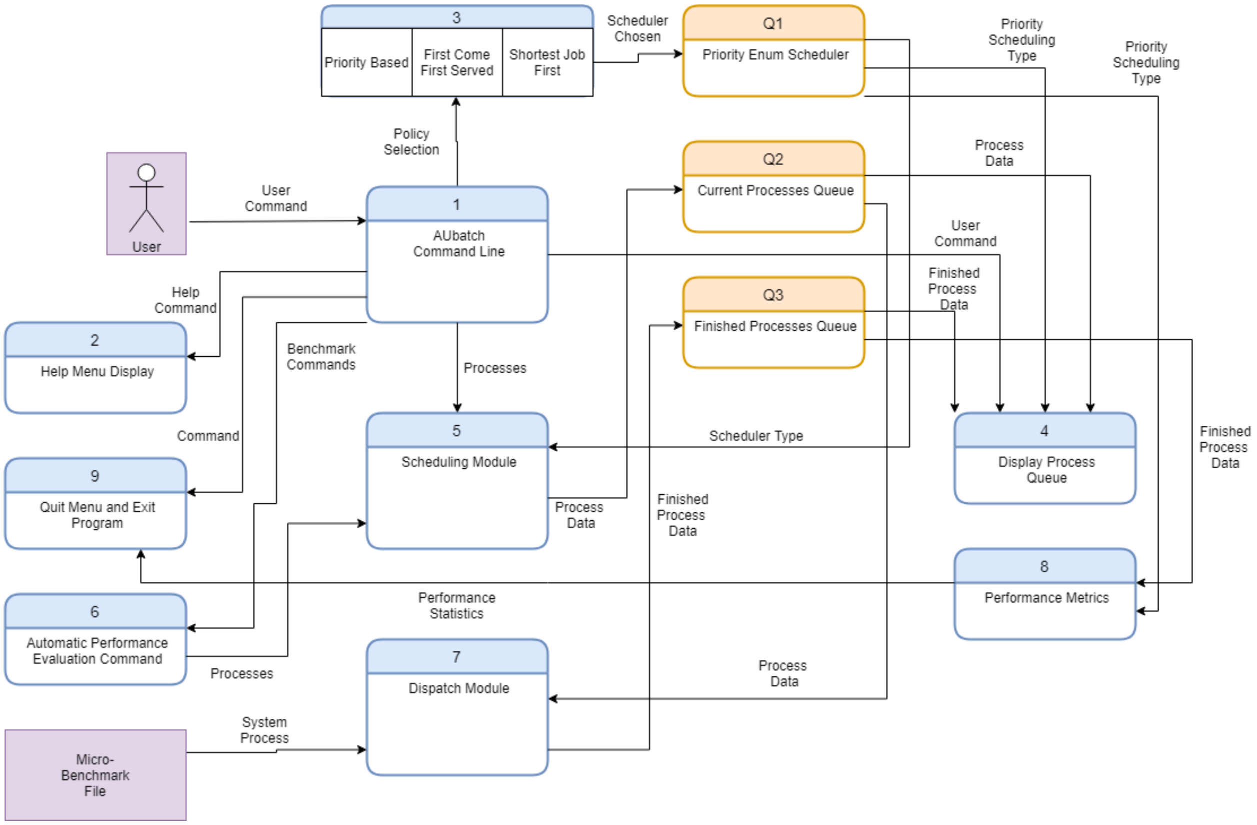
```
Script started on Sun 21 Mar 2021 11:14:42 AM PDT
^[]0;root@ubuntu: /home/toor/Desktop/CS7500-AdvancedOS/project3^Groot@ubuntu:/home/toor/Desktop/CS7500-AdvancedOS/project3# ./aubatch^M
Welcome to Hannah's batch job scheduler Version 1.0^M
Type 'help' to find more about AUbatch commands.^M
^M
>run ./micro 4 5^M
Job job0 was submitted.^M
Total number of jobs in the queue: 1^M
Expected waiting time: 0 seconds^M
Scheduling Policy: FCFS.^M
^M
>list^M
No processes are running.^M
^M
>run ./micro 10 3^M
Job job1 was submitted.^M
Total number of jobs in the queue: 1^M
Expected waiting time: 0 seconds^M
Scheduling Policy: FCFS.^M
^M
>list^M
Total number of jobs in the queue: 1^M
Scheduling Policy: FCFS^M
Name    CPU_Time        Pri     Arrival_time    Progress^M
job1    10              3       11:15:21        Run     ^M
^M
>list^M
No processes are running.^M
^M
>quit^M
Total number of jobs submitted: 2^M
Average turnaround time: 7.000 seconds^M
Average CPU time: 7.000 seconds^M
Average waiting time: 7.000 seconds^M
Average response time: 0.000 seconds^M
Throughput: 0.143 No./second^M
^[]0;root@ubuntu: /home/toor/Desktop/CS7500-AdvancedOS/project3^Groot@ubuntu:/home/toor/Desktop/CS7500-AdvancedOS/project3# exit^M
exit^M

Script done on Sun 21 Mar 2021 11:15:39 AM PDT
~
~
~
```

Running processes can be sorted and using three scheduling algorithms. Please see section 1.3 for more information on usage of these algorithms and the benchmark stress testing for performance evaluation.

The project features error checking and proper documentation within all three usage files and header files. The system uses separate compilation and compiles with a make file for user-friendly compilation. The microbatch program takes in a user defined integer, forks a child process and sleeps for that amount of seconds. While the child is sleeping, the parent performs math equations on the CPU and exits once the child is done sleeping. This works the cpu for a user defined amount of time.

Finally, below is the AUbatch DFD Diagram for this program.

Diagram: DFD Flow Diagram



1.2 Performance Metrics and Workload

Workload conditions help provide parameters to the automatic testing functionality that overall help evaluate the performance of the three scheduling algorithms: FCFS, SJF and Priority. Below is a sample chart of workload conditions that can be used to in these stress tests.

Diagram: Workload conditions

Workload Parameters	Values
Number of Submitted Jobs	5, 10, 15, 20
Arrival Rate	1, 3, 5, 8
Load Distribution	[1, 10], [0, 5], [7, 20], [3, 15]

Performance metrics are included as such:

- Average Response Time
- Throughput
- Average Turnaround Time
- Average CPU Time
- Average Waiting Time

These metrics are included to help evaluate the performance of jobs performed on the user's CPU and to help evaluate the performance of included scheduling algorithms. Below is a chart on how each metric is calculated.

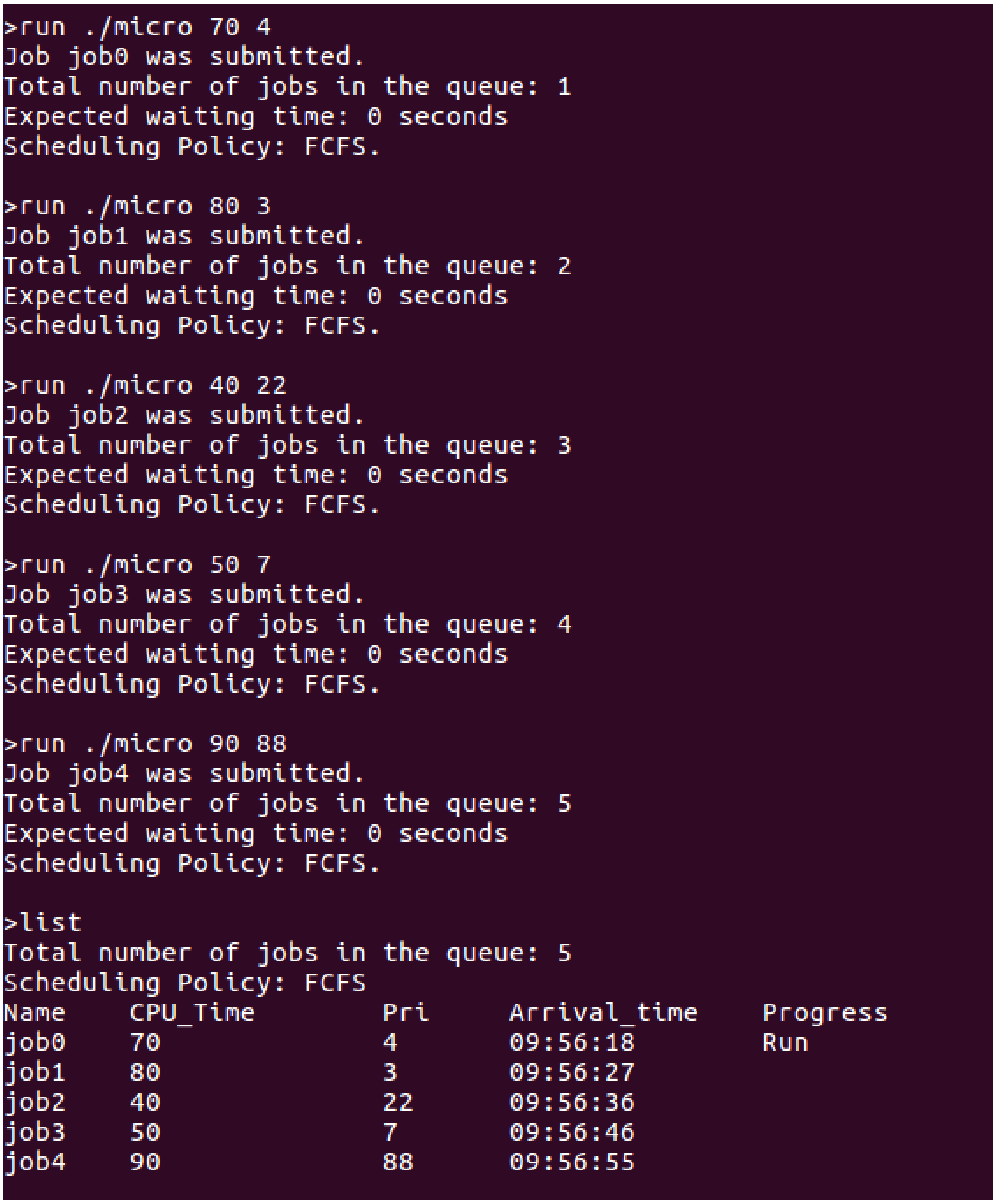
Diagram: Performance Metric Calculations

Performance Metric	**Calculation**
Throughput	1 / ((Total Finish Time – Total Arrival Time) / Total Processes)
Average Response Time	(Total First Time on CPU – Total Arrival Time) / Total Processes
Average Turnaround Time	(Total Finish Time – Total Arrival Time) / Total Processes
Average CPU Time	Total CPU Burst Time / Total Processes
Average Waiting Time	(Total CPU burst Time + (Total Turnaround Time – Total CPU Time)) / Total Processes

1.3 Performance Evaluation of Three Scheduling Algorithms

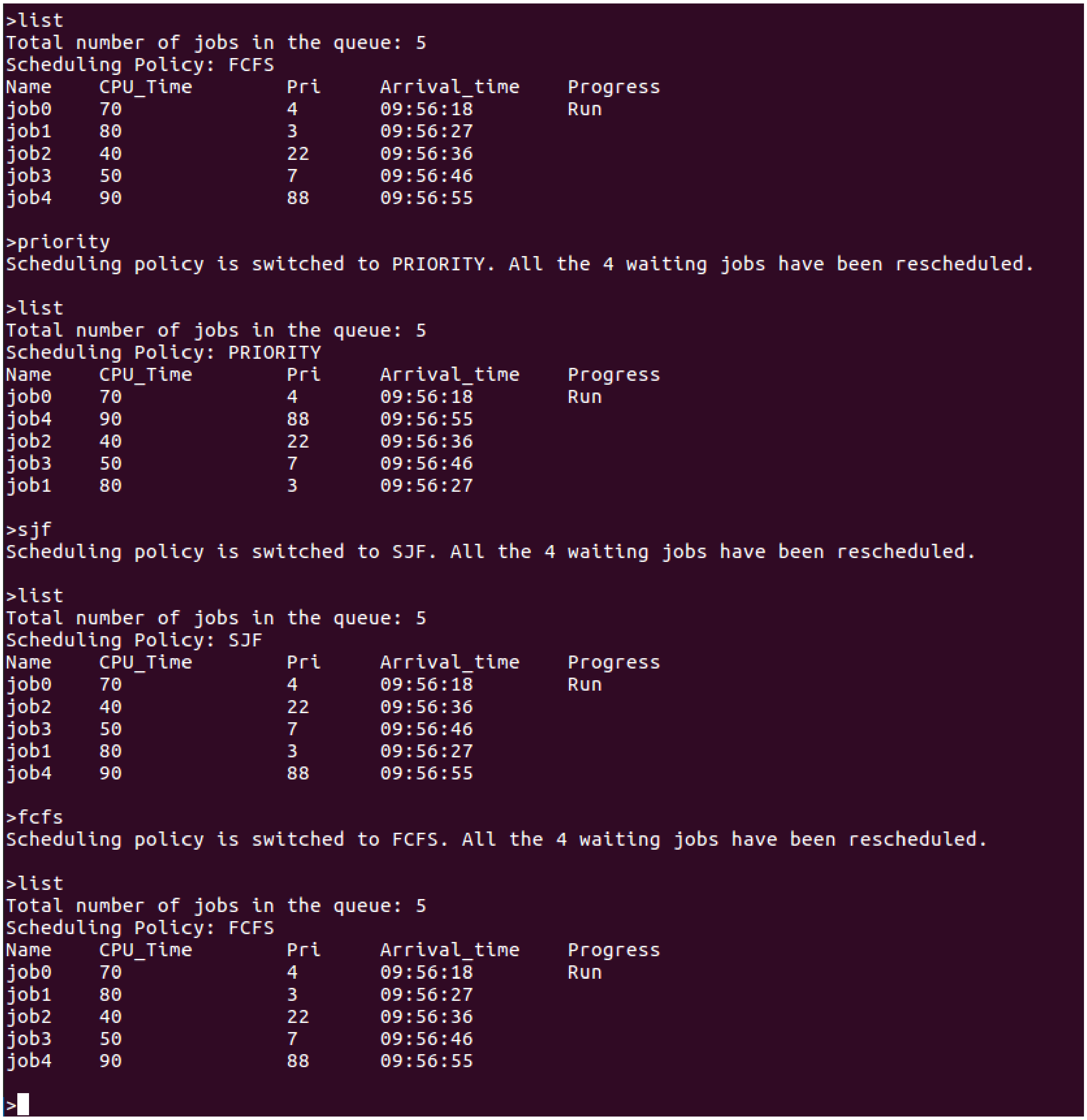
Performance between the three scheduling algorithms needs to be tested under certain workloads in order to study performance and efficiency. The three scheduling algorithms include First Come First Serve (FCFS), Shortest Job First (SJF) and Priority Based Scheduling (Priority). These benchtests are performed using the defined workload conditions specified in section 1.2. See the below diagrams of successful algorithm functionality and benchtests.

Diagram: Initial Queue



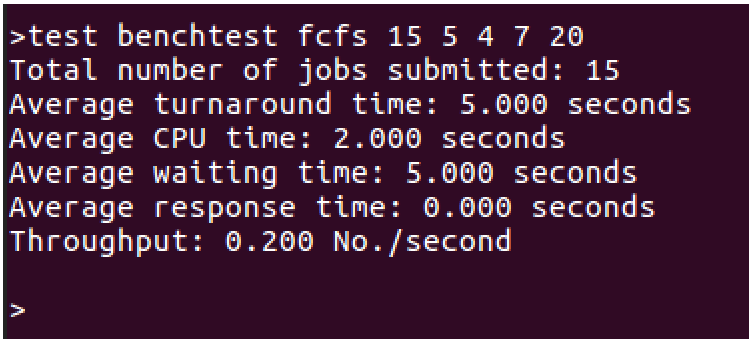
```
>run ./micro 70 4
Job job0 was submitted.
Total number of jobs in the queue: 1
Expected waiting time: 0 seconds
Scheduling Policy: FCFS.

>run ./micro 80 3
Job job1 was submitted.
Total number of jobs in the queue: 2
Expected waiting time: 0 seconds
Scheduling Policy: FCFS.

>run ./micro 40 22
Job job2 was submitted.
Total number of jobs in the queue: 3
Expected waiting time: 0 seconds
Scheduling Policy: FCFS.

>run ./micro 50 7
Job job3 was submitted.
Total number of jobs in the queue: 4
Expected waiting time: 0 seconds
Scheduling Policy: FCFS.

>run ./micro 90 88
Job job4 was submitted.
Total number of jobs in the queue: 5
Expected waiting time: 0 seconds
Scheduling Policy: FCFS.

>list
Total number of jobs in the queue: 5
Scheduling Policy: FCFS
Name    CPU_Time        Pri     Arrival_time    Progress
job0    70              4       09:56:18        Run
job1    80              3       09:56:27
job2    40              22      09:56:36
job3    50              7       09:56:46
job4    90              88      09:56:55
```

Diagram: Scheduling Sort Example



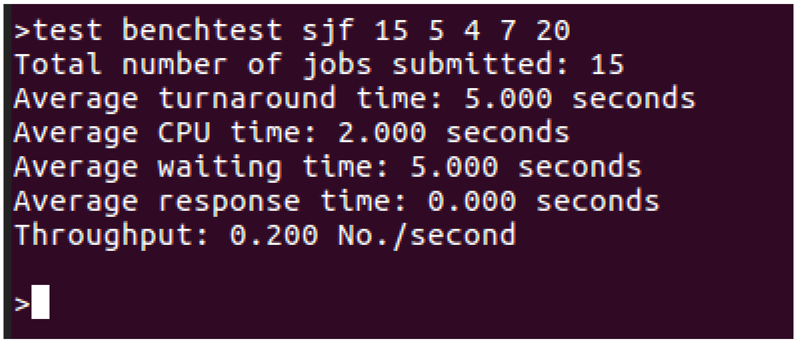
```
>list
Total number of jobs in the queue: 5
Scheduling Policy: FCFS
Name    CPU_Time        Pri     Arrival_time    Progress
job0    70              4       09:56:18        Run
job1    80              3       09:56:27
job2    40              22      09:56:36
job3    50              7       09:56:46
job4    90              88      09:56:55

>priority
Scheduling policy is switched to PRIORITY. All the 4 waiting jobs have been rescheduled.

>list
Total number of jobs in the queue: 5
Scheduling Policy: PRIORITY
Name    CPU_Time        Pri     Arrival_time    Progress
job0    70              4       09:56:18        Run
job4    90              88      09:56:55
job2    40              22      09:56:36
job3    50              7       09:56:46
job1    80              3       09:56:27

>sjf
Scheduling policy is switched to SJF. All the 4 waiting jobs have been rescheduled.

>list
Total number of jobs in the queue: 5
Scheduling Policy: SJF
Name    CPU_Time        Pri     Arrival_time    Progress
job0    70              4       09:56:18        Run
job2    40              22      09:56:36
job3    50              7       09:56:46
job1    80              3       09:56:27
job4    90              88      09:56:55

>fcfs
Scheduling policy is switched to FCFS. All the 4 waiting jobs have been rescheduled.

>list
Total number of jobs in the queue: 5
Scheduling Policy: FCFS
Name    CPU_Time        Pri     Arrival_time    Progress
job0    70              4       09:56:18        Run
job1    80              3       09:56:27
job2    40              22      09:56:36
job3    50              7       09:56:46
job4    90              88      09:56:55

>
```

Diagram: FCFS Benchtest Performance



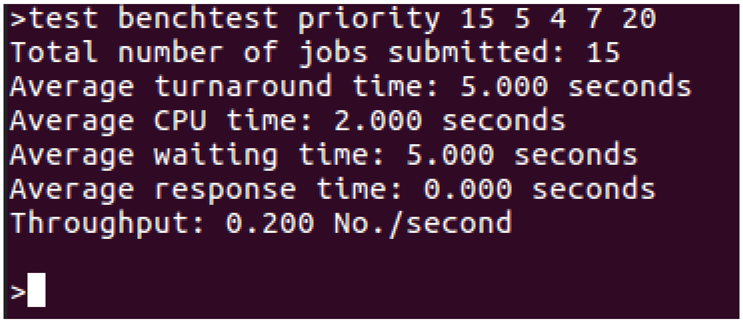
```
>test benchtest fcfs 15 5 4 7 20
Total number of jobs submitted: 15
Average turnaround time: 5.000 seconds
Average CPU time: 2.000 seconds
Average waiting time: 5.000 seconds
Average response time: 0.000 seconds
Throughput: 0.200 No./second

>
```

Diagram: SJF Benchtest Performance



```
>test benchtest sjf 15 5 4 7 20
Total number of jobs submitted: 15
Average turnaround time: 5.000 seconds
Average CPU time: 2.000 seconds
Average waiting time: 5.000 seconds
Average response time: 0.000 seconds
Throughput: 0.200 No./second

>
```

Diagram: Priority Benchtest Performance



```
>test benchtest priority 15 5 4 7 20
Total number of jobs submitted: 15
Average turnaround time: 5.000 seconds
Average CPU time: 2.000 seconds
Average waiting time: 5.000 seconds
Average response time: 0.000 seconds
Throughput: 0.200 No./second

>
```

After evaluation all the scheduling algorithms with the previous benchtest, it is difficult to see which algorithm would be more efficient under these circumstances. This is most likely due to processor speed and not enough time to see a difference. The differences in speed are very slight and would require more in-depth testing in order to possibly see performance variances. There are several

1.4 Lessons Learned

During the development of this project there were several learning moments. Below are some notes on those lessons.

1. EXECV takes over a process and exits when complete. This means it is necessary to create a child process with fork in order to successfully run and exit this function while not closing the AUbatch program upon execv completion. This took me several hours to figure out.

2. Performing a copy with MEMCPY is a lot easier than writing a whole other function to copy a running process to the finished process queue. But it is important to pass by reference, otherwise the pointer will only be copied.

3. When executing a program in the dispatcher, it is important to check to make sure the correct microbatch program is running. I created a microbatch program to run with AUbatch and have my program check if the microbatch program selected is my program, then it updates the correct CPU burst time. If a different microbatch program is selected, it will run that program for zero time. This was to minimize errors by not running it. If a custom microbatch program seems to never be running, this is why. Additionally I put checking in place to check for valid program files.

4. The microbatch program was specified as "running a user amount of time for x seconds on the CPU". The sleep function is great as a timer, but does not workout the cpu. Because of this, I created a fork to run sleep and while the parent process was waiting on the child to complete, it runs math calculations. This way the cpu is working for user defined amount of seconds. This took me a few hours to think up.

5. Don't procrastinate. This project took me a solid week of programming to finish. Because midterms hit at the same time, I wasn't able to work on this project much before it was due. Thankfully the extensions made it possible to complete the project.

These are the big lessons I learned while designing this AUbatch system.