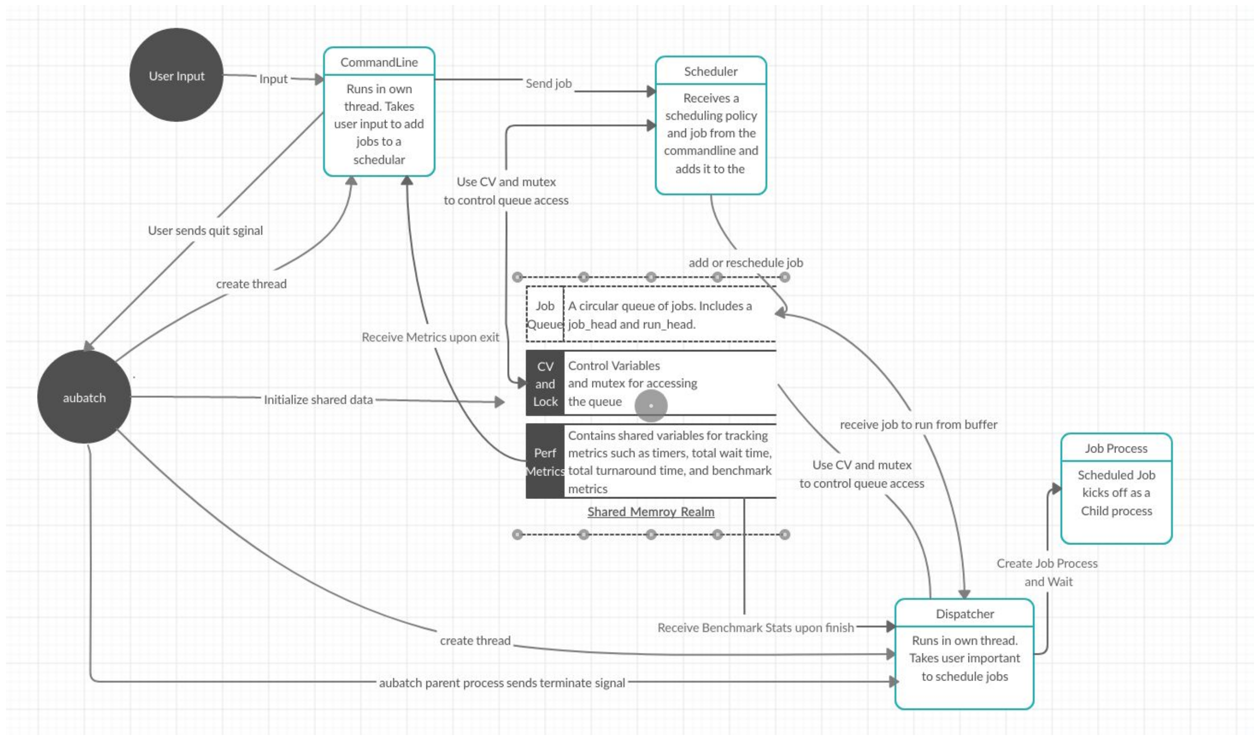
William Christopher White
Comp7500

- <u>Please note the length of the report is due to listing of benchmark results and a long script session</u>

Design, Implementation, and DFD Diagram:

DTD:


User Input
Input
CommandLine
Runs in own thread. Takes user input to add jobs to a schedular
Send job
Scheduler
Receives a scheduling policy and job from the commandline and adds it to the
Use CV and mutex to control queue access
User sends quit sginal
create thread
add or reschedule job
Job Queue
A circular queue of jobs. Includes a job_head and run_head.
Receive Metrics upon exit
CV and Lock
Control Variables and mutex for accessing the queue
aubatch
Initialize shared data
receive job to run from buffer
Perf Metrics
Contains shared variables for tracking metrics such as timers, total wait time, total turnaround time, and benchmark metrics
Use CV and mutex to control queue access
Job Process
Scheduled Job kicks off as a Child process
Shared Memroy Realm
Create Job Process and Wait
Dispatcher
Runs in own thread. Takes user important to schedule jobs
Receive Benchmark Stats upon finish
create thread
aubatch parent process sends terminate signal

The above Data Flow Diagram gives the outline of the flow of the program. The program has a parent process called aubatch. The aubatch controls three separate parts: a shared memory region, a job dispatcher, and a command line interface. The command line interface in turn controls the scheduler. All of the modules have access to a shared memory region. The aubatch process starts off by initializing the shared memory variables, the control variables, and the mutex. It then creates two threads. One thread for the command line interface and one thread for the dispatcher.

The shared memory is an important piece and can be accessed by every module in the program. It allows for the command line/scheduling thread and the dispatch thread to share variables with each other. The shared memory includes the jobs queue, a mutex, and two control variables to indicate that the queue is empty or full. The jobs queue uses two variables to track the head of the queue for the scheduler and the head of the queue for the dispatcher, called job_head and run_head (respectively). Additionally the shared memory holds many metric variables. There are two sets of time variables, one measures the total run time of the application and one measures how long a process has actually been running. On top of this, counts are kept for number of processed jobs, total turnaround time, total wait time, and total execution time. These counts are kept for all processed jobs. There are also the same counting variables that measure only the specific benchmark instead of all jobs processed. This region also stores the range of the benchmark, allowing the dispatcher to know where in the queue a benchmark starts and where in the queue the benchmark ends. The definition for the job structure is also here. The job structure contains name, arrival time as a number and string, waiting time, burst time, priority, and its run status (Which can either waiting, running, or complete). The job queue is an array of job. The jobs queue is implemented as a circular queue. The job queue is locked on all writes, but can have simultaneous reads.

The command line interface thread is the thread that takes in user input. The scheduling module also runs on the same thread and is controlled by the command line module. The command line interface allows for a user to add a single job to the queue, run a batch test, list jobs currently in the queue, and change scheduling priority. When the user exits the command line, the statistics of all ran jobs are printed. These statistics are grabbed from the shared memory region. These statistics include job count, turnaround time, waiting time and throughput. Once the quit command is signaled, the parent process kills both the command line thread and the dispatcher thread. The benchmark command will send multiple jobs to the scheduler to be queued. It is important to note that the program is designed to handle only one benchmark at a time. The command line will not take in a new benchmark until a shared variable indicates that the previous benchmark is finished. The benchmark tool generates priority and burst time based on a randomly generated number falling within the users provided range.The command line updates the shared memory so that the dispatcher knows how far up the queue a benchmark will last. The default scheduling algorithm on launch is FCFS.

The next piece is the scheduler. The schedular’s main task is to take in a single job at a time and place it into the jobs queue based on the policy given. When a job is received, the scheduler checks to see if the queue is full and waits if it is. If the queue is not full, then the scheduler attempts to lock the mutex. At this point the scheduler creates and populates all the information specified by the job structure. For arrival time and waiting time, timers from the shared memory region are used. The total-time timer is used to calculate arrival time as an integer. This number is also stored as a string to allow easy printing. The waiting time is also calculated by first calculating the remaining time on the running job and then by adding the execution time of each job ahead of it. The current process remaining time is measured using a time in the shared region that contains the start time of each process. Once a job is added, the control variable is signaled and the mutex unlocked, allowing the dispatcher module to access the job queue. The scheduler will also detect a new policy change and reorganize all waiting jobs in the queue. It’s important to note that this is a nonpreemptive scheduler so the running job will continue running at the top of the queue. When a new policy is detected, the same condition variable and mutex criteria are applied before making any changes. When the queue is rescheduled, the running thread is not preempted and continues to run at the top of the queue. The scheduler sleeps for one second while adding each job to simulate arrival time.

The final piece to the design is the dispatcher module. The dispatcher module runs in a separate thread than the command line and scheduling modules. It constantly loops through looking for new jobs. It uses a control variable to check if the queue is empty. Once it sees that there is a job, it locks the queue and gets the job out of the queue. It then unlocks the queue right before running the job. In order to run the job, execv is used to launch the batch_job file that simulated a full process. Batch_job takes in a process name and time as parameters. It then runs the process for x seconds. The dispatcher waits for the process to finish before moving on to the next job in the queue. The dispatcher also updates variables in a shared memory region that keep up with metrics. The dispatcher also uses the shared memory region to detect if a benchmark is being performed. If it sees there is a benchmark, it determines when the benchmark begins in the queue and where it ends. Once it detects that a benchmark is being performed, it starts tracking metrics for that benchmark. And once it detects that the benchmark is finished, it prints all the metrics. At this point the shared memory region is updated to indicate that a new benchmark may begin.

I used Dr. QIn’s command line example, aubatch_sample example, and the execv example to form the basis of my project. I then rewrote it so that it would work with a shared memory region and added some of my own logic to it.

Issues with Implementation.

There were a few issues with the implementation. One of the most notable issues was with the circular buffer. There is an edge case where if a extremely long process is running, and a very large number of processes come in that resets the circular buffer. If this happens, then there is a chance that the job_head reaches the run_head. Right now, if this were to happen the scheduler would start overwriting jobs that have not been processed yet. In order to stop this, I could have used a third condition variable that causes the scheduler to pause when it sees that the job_head is equal to the run_head and the job_head's current job's status is set to run. Another problem with my implementation is that only one benchmark could be run time. I did this to greatly reduce the complexity of gathering benchmark stats. This could have been expanded by creating a structure that kept up with individual benchmarks stats, queue start position, and queue finish location. This structure would be stored in an array that represents all running benchmarks. The other issue with benchmarking was that if jobs are manually added into the queue before launching a benchmark, then the benchmark picks up the manual jobs in the queue in its statistic monitoring. I attempted to code it to have the dispatcher wait to record the stats until it gets to the benchmark's first stored job, but this caused a bug I did not have time to fix. There is also a bug with printing of arrival time that cause the hour ticker to go up too soon. While this does not affect the integer arrival time used in metrics and comparison, it does make the list function to occasionally print weird numbers.

Performance Evaluation

Performance Metrics and Workload Conditions

There are three main performance criteria for evaluating workload conditions. The number of submitted jobs, the arrival rate, and the load distribution. The arrival rate for this program was done in intervals of 1 second. Having a consistent arrival time interval made the arrival time consistent between multiple benchmark runs. The load distribution is a range of burst times provided by the user. The load distribution consists of minimum and maximum cpu time time. The cpu time assigned to a process is randomly generated between that range. The nondeterministic nature of this range can cause some problems when comparing performance between benchmarks under different scheduling policies. This is because a benchmark with all the same parameters but different policy as another will have different average burst times. This will cause different average execution times between benchmarks, which could give inconsistent results. Multiple benchmarks under the same workload conditions could have been performed and the average taken to get a more realistic picture of differences between policies. One important note about distribution load: A longer distribution load range gives for better comparison results. This may seem counterproductive because similar or the same burst time for each job will give consistent results. The problem is that if each job takes the same amount of time to execute, all policies will handle it about the same. By providing a large range (Example 1 to 40), you get the most useful results.

The main performance criteria used to evaluate different policies were average turnaround time, average execution time, average waiting time, and average throughput. The average turnaround time measures the entire time taken to complete a process from its arrival on average. As job number and average execution time go up, this normally also goes up with it. Average execution time is simply the average of all cpu burst times. The average waiting time is the average time a process spends waiting after its arrival into the queue. Like turnaround time, when the job number of execution time increases, this will increase with it. The last metric is throughput. The throughput is simply how many jobs were processed in a given time unit. In my benchmark, this is how many jobs completed per second.

Performance Evaluation Between Policies

Below there are a bunch of benchmark results used to come to these conclusions. The benchmarks were broken up between 5 and 10 jobs with time ranges going from small, medium, large, and wide distribution. The Wide distribution gave the best results because it highlighted the biggest difference between policies. It should be noted that this could be due to the random way the times are generated. As mentioned in the previous section, this could have been mitigated by taking the average over the same work conditions. I will be using the results from the bottom of the provided data to draw my comparisons.The first set of results use the range 1 to 20. When looking at these results, it appears that SJF is more efficient than the rest. The turnaround time and waiting time for SJF is in the 70s, while for the other two policies sit between 90 and 100 on waiting time and 120 and 130 on turnaround time. Additionally, SFJ has a higher throughput handling 0.082 jobs per second. When the range is increased to 1 to 40 things change a little bit. The turnaround time and waiting time fall behind the priority policy, however the priority policy has a lower cpu burst time which may have played a factor in that. Even with these differences in turnaround time and waiting time, SJF still has the best throughput. Some of the other results produce results that were too similar to have a good comparison.

The following are all of the stats I gathered to compare different results among policies.

## 5 jobs and 10 to 20 seconds ##

```
test mybenchmark fcfs 5 5 10 20

Total number of jobs submitted: 5
Average turn around time: 37.600000 seconds
Average execution time: 12.000000 seconds
Average waiting time: 21.600000 seconds
Throughput: 0.083333
```

test mybenchmark sjf 5 5 10 20

```
Total number of jobs submitted: 5
Average turn around time: 35.000000 seconds
Average execution time: 12.400000 seconds
Average waiting time: 24.600000 seconds
Throughput: 0.080645
```

test mybenchmark priority 5 5 10 20

```
Average turn around time: 54.400000 seconds
Average execution time: 17.400000 seconds
Average waiting time: 33.600000 seconds
Throughput: 0.057471
```

## 5 jobs and 20 to 40 seconds ##

test mybenchmark fcfs 5 5 20 40

```
Total number of jobs submitted: 5
Average turn around time: 97.000000 seconds
Average execution time: 31.000000 seconds
Average waiting time: 62.000000 seconds
Throughput: 0.032258
```

test mybenchmark sjf 5 5 20 40

```
Average turn around time: 96.200000 seconds
Average execution time: 33.400000 seconds
Average waiting time: 62.000000 seconds
Throughput: 0.029940
```

test mybenchmark priority 5 5 20 40

```
Total number of jobs submitted: 5
Average turn around time: 82.000000 seconds
Average execution time: 27.000000 seconds
Average waiting time: 50.400000 seconds
Throughput: 0.037037
```

## 20 jobs and 1 to 3 seconds ##

test mybenchmark fcfs 20 5 1 3

```
Total number of jobs submitted: 20
```

```
Average turn around time: 28.400000 seconds
Average execution time: 1.700000 seconds
Average waiting time: 7.700000 seconds
Throughput: 0.588235
```

test mybenchmark sjf 20 5 1 3

```
Total number of jobs submitted: 20
Average turn around time: 28.900000 seconds
Average execution time: 2.150000 seconds
Average waiting time: 10.250000 seconds
Throughput: 0.465116
```

test mybenchmark priority 20 5 1 3

```
Total number of jobs submitted: 20
Average turn around time: 26.300000 seconds
Average execution time: 1.650000 seconds
Average waiting time: 5.550000 seconds
Throughput: 0.606061
```

## 10 jobs and 40 to 50 seconds ##

test mybenchmark fcfs 20 5 40 50

```
Total number of jobs submitted: 20
Average turn around time: 479.950000 seconds
Average execution time: 44.300000 seconds
Average waiting time: 416.650000 seconds
Throughput: 0.022573
```

test mybenchmark sjf 20 5 40 50

```
Total number of jobs submitted: 20
Average turn around time: 469.400000 seconds
Average execution time: 45.250000 seconds
Average waiting time: 415.250000 seconds
Throughput: 0.022099
```

test mybenchmark priority 20 5 40 50

```
Total number of jobs submitted: 20
Average turn around time: 495.800000 seconds
Average execution time: 46.500000 seconds
Average waiting time: 434.550000 seconds
Throughput: 0.021505
```

## 10 jobs with a range of 1 to 20 (Best comparison)##

```
test mybenchmark fcfs 20 5 1 20

Average turn around time: 132.850000 seconds
Average execution time: 11.350000 seconds
Average waiting time: 102.500000 seconds
Throughput: 0.088106
```

```
test mybenchmark sjf 20 5 1 20
Average turn around time: 77.250000 seconds
Average execution time: 9.300000 seconds
Average waiting time: 73.500000 seconds
Throughput: 0.107527
```

```
test mybenchmark priority 20 5 1 20
Total number of jobs submitted: 20
Average turn around time: 123.400000 seconds
Average execution time: 12.150000 seconds
Average waiting time: 91.400000 seconds
Throughput: 0.082305
```

## 10 jobs with a range of 1 to 40 (Best comparison)##

```
test mybenchmark fcfs 10 10 1 40
Total number of jobs submitted: 10
Average turn around time: 132.100000 seconds
Average execution time: 21.000000 seconds
Average waiting time: 102.100000 seconds
Throughput: 0.047619
```

```
test mybenchmark sjf 10 10 1 40

Total number of jobs submitted: 10
Average turn around time: 126.300000 seconds
Average execution time: 26.700000 seconds
Average waiting time: 125.200000 seconds
Throughput: 0.037453
```

```
test mybenchmark priority 10 10 1 40
```

```
Total number of jobs submitted: 10
Average turn around time: 118.600000 seconds
Average execution time: 24.100000 seconds
Average waiting time: 117.500000 seconds
Throughput: 0.041494


## Analysis of all jobs. Note that throughput is skewed##
## By time not spent with no jobs in queue##
Total number of jobs submitted: 104
Average turn around time: 15.788462 seconds
Average execution time: 4.644231 seconds
Average waiting time: 12.192308 seconds
Throughput: 0.020012
```

Lessons Learned:

There were a few lessons learned while working on this project. The first lesson is a pretty obvious one around time management. While it was not my intention to procrastinate, work and other world events cause me to push this off to the side. In the future, I plan to better organize my time so that one thing does not go to the wayside. Outside of that, I learned a lot about how a shared memory region works between multiple processes as I had never done that before. Additionally, my C programming skills are starting to get better. While I still have more to learn, I made a lot of progress during this project. In the future, I would like to make implementing unit testing each module a priority. For this project, I used manual testing. This was mostly due to there not being enough time at the beginning to write unit tests. This ended up being a mistake as the project got more and more complex. By the end of it, the number of tests I had to run for each completed step had increased and I had to waste time running manual tests. Having a unit test suite that quickly ran and told me if I had broken anything would have been very useful while I added in functionality.

Sample Usage

Below is a long session of the tool in action. This was placed at the bottom to not distract from the rest of the report. **Please note that to run this, it is best to use the “run.sh” command as it ensures that you are running everything from the proper directory.**

```
Script started on 2020-03-11 23:16:14-05:00 [TERM="xterm-256color"
TTY="/dev/pts/4" COLUMNS="238" LINES="29"]
]777;notify;Command
```

```
completed;exit\]777;precmd\]0;chwhite@localhost:/Shared/eclipse-workspace-cpp/AU
Batch/project3\]7;file://localhost.localdomain/Shared/eclipse-workspace-cpp/AUBatch/
project3\[chwhite@localhost project3]$ exit./run.sh
777;preexecWelcome to William (Chris) White's batch schedular Version 0.1 alpha
Type 'help' to find more about AUBatch commands.
> [? for menu]: list
Total number of jobs in the queue: 0
Scheduling Policy: FCFS
There are no jobs to display
> [? for menu]: test mybenchmark sjf 10 10 1 40
Job mybenchmark1 was submitted.
Total number of jobs in the queue: 1
Expected waiting time is: 0 seconds
Scheduling Policy: SJF
Running process mybenchmark1 for 18...
Job mybenchmark2 was submitted.
Total number of jobs in the queue: 2
Expected waiting time is: 17 seconds
Scheduling Policy: SJF
Job mybenchmark3 was submitted.
Total number of jobs in the queue: 3
Expected waiting time is: 46 seconds
Scheduling Policy: SJF
Job mybenchmark4 was submitted.
Total number of jobs in the queue: 4
Expected waiting time is: 79 seconds
Scheduling Policy: SJF
Job mybenchmark5 was submitted.
Total number of jobs in the queue: 5
Expected waiting time is: 114 seconds
Scheduling Policy: SJF
Job mybenchmark6 was submitted.
Total number of jobs in the queue: 6
Expected waiting time is: 147 seconds
Scheduling Policy: SJF
Job mybenchmark7 was submitted.
Total number of jobs in the queue: 7
Expected waiting time is: 183 seconds
Scheduling Policy: SJF
Job mybenchmark8 was submitted.
Total number of jobs in the queue: 8
Expected waiting time is: 208 seconds
Scheduling Policy: SJF
Job mybenchmark9 was submitted.
```

```
Total number of jobs in the queue: 9
Expected waiting time is: 229 seconds
Scheduling Policy: SJF
Job mybenchmark10 was submitted.
Total number of jobs in the queue: 10
Expected waiting time is: 229 seconds
Scheduling Policy: SJF
> [? for menu]: list
Total number of jobs in the queue: 10
Scheduling Policy: SJF
Name           CPU_Time Pri Arrival_time Progress
mybenchmark1       18      6   00:00:23   Running
mybenchmark9       1       1   00:00:31   Waiting!
mybenchmark8       22      5   00:00:30   Waiting!
mybenchmark7       26      6   00:00:29   Waiting!
mybenchmark10   29         4   00:00:32   Waiting!
mybenchmark2       30      4   00:00:24   Waiting!
mybenchmark3       34      1   00:00:25   Waiting!
mybenchmark5       34      9   00:00:27   Waiting!
mybenchmark4       36      9   00:00:26   Waiting!
mybenchmark6       37      8   00:00:28   Waiting!
> [? for menu]: Done ...
Running process mybenchmark9 for 1...
Done ...
Running process mybenchmark8 for 22...
Done ...
Running process mybenchmark7 for 26...
Done ...
Running process mybenchmark10 for 29...
Done ...
Running process mybenchmark2 for 30...
Done ...
Running process mybenchmark3 for 34...
Done ...
Running process mybenchmark5 for 34...
list
Total number of jobs in the queue: 3
Scheduling Policy: SJF
Name           CPU_Time Pri Arrival_time Progress
mybenchmark5       34      9   00:00:27   Running
mybenchmark4       36      9   00:00:26   Waiting!
mybenchmark6       37      8   00:00:28   Waiting!
> [? for menu]: Done ...
Running process mybenchmark4 for 36...
```

```
Done ...
Running process mybenchmark6 for 37...
Done ...
The benchmark mybenchmark is over
Total number of jobs submitted: 10
Average turn around time: 126.300000 seconds
Average execution time: 26.700000 seconds
Average waiting time: 125.200000 seconds
Throughput: 0.037453
test mybenchmark sjf 10 10 1 40
Job mybenchmark1 was submitted.
Total number of jobs in the queue: 1
Expected waiting time is: 0 seconds
Scheduling Policy: SJF
Running process mybenchmark1 for 25...
Job mybenchmark2 was submitted.
Total number of jobs in the queue: 2
Expected waiting time is: 24 seconds
Scheduling Policy: SJF
Job mybenchmark3 was submitted.
Total number of jobs in the queue: 3
Expected waiting time is: 46 seconds
Scheduling Policy: SJF
Job mybenchmark4 was submitted.
Total number of jobs in the queue: 4
Expected waiting time is: 76 seconds
Scheduling Policy: SJF
Job mybenchmark5 was submitted.
Total number of jobs in the queue: 5
Expected waiting time is: 113 seconds
Scheduling Policy: SJF
Job mybenchmark6 was submitted.
Total number of jobs in the queue: 6
Expected waiting time is: 136 seconds
Scheduling Policy: SJF
Job mybenchmark7 was submitted.
Total number of jobs in the queue: 7
Expected waiting time is: 166 seconds
Scheduling Policy: SJF
Job mybenchmark8 was submitted.
Total number of jobs in the queue: 8
Expected waiting time is: 186 seconds
Scheduling Policy: SJF
Job mybenchmark9 was submitted.
```

```
Total number of jobs in the queue: 9
Expected waiting time is: 210 seconds
Scheduling Policy: SJF
Job mybenchmark10 was submitted.
Total number of jobs in the queue: 10
Expected waiting time is: 218 seconds
Scheduling Policy: SJF
> [? for menu]: Done ...
Running process mybenchmark9 for 9...
Done ...
Running process mybenchmark10 for 14...
list
Total number of jobs in the queue: 8
Scheduling Policy: SJF
Name          CPU_Time Pri Arrival_time Progress
mybenchmark10   14      9  01:00:43  Running
mybenchmark7    21      2  01:00:40  Waiting!
mybenchmark2    23      2  01:00:35  Waiting!
mybenchmark5    24      3  01:00:38  Waiting!
mybenchmark8    25      9  01:00:41  Waiting!
mybenchmark3    31      6  01:00:36  Waiting!
mybenchmark6    31      3  01:00:39  Waiting!
mybenchmark4    38      2  01:00:37  Waiting!
> [? for menu]: Done ...
Running process mybenchmark7 for 21...
Done ...
Running process mybenchmark2 for 23...
Done ...
Running process mybenchmark5 for 24...
Done ...
Running process mybenchmark8 for 25...
Done ...
Running process mybenchmark3 for 31...
Done ...
Running process mybenchmark6 for 31...
Done ...
Running process mybenchmark4 for 38...
Done ...
The benchmark mybenchmark is over
Total number of jobs submitted: 10
Average turn around time: 118.600000 seconds
Average execution time: 24.100000 seconds
Average waiting time: 117.500000 seconds
Throughput: 0.041494
```

```
test mybenchmark fcfs 5 5 20 40
Job mybenchmark1 was submitted.
Total number of jobs in the queue: 1
Expected waiting time is: 0 seconds
Scheduling Policy: FCFS
Running process mybenchmark1 for 25...
Job mybenchmark2 was submitted.
Total number of jobs in the queue: 2
Expected waiting time is: 24 seconds
Scheduling Policy: FCFS
Job mybenchmark3 was submitted.
Total number of jobs in the queue: 3
Expected waiting time is: 44 seconds
Scheduling Policy: FCFS
Job mybenchmark4 was submitted.
Total number of jobs in the queue: 4
Expected waiting time is: 68 seconds
Scheduling Policy: FCFS
Job mybenchmark5 was submitted.
Total number of jobs in the queue: 5
Expected waiting time is: 90 seconds
Scheduling Policy: FCFS
> [? for menu]: list
Total number of jobs in the queue: 5
Scheduling Policy: FCFS
Name           CPU_Time Pri Arrival_time Progress
mybenchmark1      25     2   02:01:07   Running
mybenchmark2      21     1   02:01:08   Waiting!
mybenchmark3      25     0   02:01:09   Waiting!
mybenchmark4      23     3   02:01:10   Waiting!
mybenchmark5      25     3   02:01:11   Waiting!
> [? for menu]: priority
> [? for menu]: list
Total number of jobs in the queue: 5
Scheduling Policy: priority
Name           CPU_Time Pri Arrival_time Progress
mybenchmark1      25     2   02:01:07   Running
mybenchmark3      25     0   02:01:09   Waiting!
mybenchmark2      21     1   02:01:08   Waiting!
mybenchmark4      23     3   02:01:10   Waiting!
mybenchmark5      25     3   02:01:11   Waiting!
> [? for menu]: Done ...
Running process mybenchmark3 for 25...
sjf
```

```
> [? for menu]: list
Total number of jobs in the queue: 4
Scheduling Policy: SJF
Name          CPU_Time Pri Arrival_time Progress
mybenchmark3      25      0   02:01:09  Running
mybenchmark2      21      1   02:01:08  Waiting!
mybenchmark4      23      3   02:01:10  Waiting!
mybenchmark5      25      3   02:01:11  Waiting!
> [? for menu]: fcfs
> [? for menu]: list
Total number of jobs in the queue: 4
Scheduling Policy: FCFS
Name          CPU_Time Pri Arrival_time Progress
mybenchmark3      25      0   02:01:09  Running
mybenchmark2      21      1   02:01:08  Waiting!
mybenchmark4      23      3   02:01:10  Waiting!
mybenchmark5      25      3   02:01:11  Waiting!
> [? for menu]: Done ...
Running process mybenchmark2 for 21...
Done ...
Running process mybenchmark4 for 23...
Done ...
Running process mybenchmark5 for 25...
Done ...
The benchmark mybenchmark is over
Total number of jobs submitted: 5
Average turn around time: 73.800000 seconds
Average execution time: 23.800000 seconds
Average waiting time: 45.200000 seconds
Throughput: 0.042017
quit
The program is terminating. Here are some stats about the jobs that have been
processed
Jobs have been processed. Below are the stats of all jobs since the program started
...
Total number of jobs submitted: 25
Average turn around time: 104.720000 seconds
Average execution time: 25.080000 seconds
Average waiting time: 106.120000 seconds
Throughput: 0.025536
]777;notify;Command completed;./run.sh
\]777;precmd\]0;chwhite@localhost:/Shared/eclipse-workspace-cpp/AUBatch/project3
\]7;file://localhost.localdomain/Shared/eclipse-workspace-cpp/AUBatch/project3\[chwh
ite@localhost project3]$ exit
```

```
777;preexecexit

Script done on 2020-03-11 23:32:38-05:00 [COMMAND_EXIT_CODE="0"]
```