Michael Blakley mzb0155@auburn.edu

Project 3 Report

COMP 7500 - AUbatch **21 MAR 2021**

CONTENTS

I. The Design of AUbatch

AUbatch is designed to perform as a batch scheduling system. To achieve job scheduling and execution, AUbatch uses two threads with a queue as a critical section. Both threads are launched with the execution of AUbatch. Additionally, AUbatch utilizes a command line parser in conjunction with the scheduler, a performance information accumulator to display metrics upon exiting the program, and an automated performance module to process batch jobs utilizing the existing AUbatch structure.

The scheduling thread launches the command line parser. As such, the command line parser contains the logic to lock and unlock the critical section containing the queue. If the queue is not full, it unlocks the critical section and waits for user input. Once input is received, it reacquires the lock and processes given command. There are two informational commands, list and help, that display a list of the jobs in the queue and a help menu respectively. The scheduling policies can be changed between first come first serve, shortest job first, and priority. All policies were designed to be non-preemptive - all jobs except the job in current execution will be reordered per the policy on a policy change or new job submission. New jobs can be manually submitted with the run command, where the scheduler will place the new job at the back of the queue and then reorganize the queue if necessary. The automated performance module is accessed through the test command and is further described in Section III. Metrics and Workloads. Finally a user can exit the program with the quit command where the jobs will be immediately terminated and all prior performance metrics will be displayed.

The queue in AUbatch is an array of pointers to structures containing relevant job information. The queue is available globally, but managed between the two threads as a critical section so that only one thread can access and update the queue at a time.

The dispatching thread launches on program execution and waits for a job to enter the queue. Once a job is in the queue, the dispatcher keeps the lock on the queue and forks a child process. The child process executes the job and terminates. The parent process unlocks the queue and waits for the child process/job execution to finish. When finished, the parent reacquires the lock and sends basic execution times to the performance accumulator.

The performance accumulator will be described in Section II. Performance Evaluation. Other design improvements and lessons learned are described in Section IV. Lessons Learned.

II. Performance Evaluation

To run jobs in AUbatch, use the run command followed by the path to the executable file, the execution time and the priority. You can run a job in the *jobs* folder with:

> run ./jobs/job1 30 3

AUbatch captures the performance of each job that runs. As mentioned in Section I, the dispatcher thread passes the basic performance information to a performance accumulator. The basic information is the job arrival time, start time, and finish time. Using those measurements, AUbatch calculates CPU time, turnaround time, waiting time, response time and throughput. These values are recorded into a metrics structure that holds the averages of each of those values with the exception of throughput which is the number of jobs per second. These values are displayed to the user upon exiting AUbatch with the quit command.

For this project, I chose to use wall time vs CPU_time. Initially I had implemented CPU_time using the clock cycles, but I felt the output was misleading depending on what type of job you submitted. A 30 second sleep job for example might only consume a second of actual CPU processing. I felt that the results were more appropriate for this project of manually sending in jobs. If we wanted to measure specific jobs for specific CPU time, then we should change the time calculations back to the clock cycle measurements.

Performance Output

These jobs were run in this sequence for FCFS, SJF and Priority:
> run ./jobs/job3 100 5
> run ./jobs/job2 30 4
> run ./jobs/job4 10 3
> run ./jobs/job1 20 1

The execution times are comparable to how long each of these jobs actually takes. Priorities were set to mix up execution order. From the output shown below, we get a fairly accurate depiction of our scheduling policies.

For SJF, you notice a lower waiting time and higher throughput than FCFS due to the quick jobs running first and improving their response times. If we queued several longer jobs that were starved at the back of the queue for a while, the waiting time would begin to rise. For this quick scenario though you can see the benefit for short jobs in SJF.

When looking at FCFS, we can observe the highest waiting and turnaround times along with a supporting lowered throughput. This would be typically

expected from larger randomized sets of jobs with varying execution times. Shorter jobs will get stuck behind longer running jobs and drive up the waiting time.

	FCFS	SJF	Priority
Average Turnaround Time (s)	182.25	170.75	164.75
Average Waiting Time (s)	118.25	107.00	101.00
Throughput (#jobs/sec)	0.0055	0.0059	0.0061

III. Metrics and Workloads

The automated performance module was designed to process a batch of jobs after invoking the test command with the appropriate arguments. **This module was specifically designed to run jobs from the *benchmark*s folder.** The concept was to load executable files into benchmarks and then run them with an example command as such:

> test ./benchmarks fcfs 4 5 10 40

The module will first set the scheduling algorithm passed as an argument. Then will cycle through each file in *benchmarks*, generate a random priority in the range of 1 to the fifth argument, generate a random execution time in the range of the sixth and seventh arguments, and build the path to the executable job in benchmarks. These steps essentially build the arguments for the run command. These arguments are passed to the scheduler for processing in the same manner that individual jobs are processed.

Below is a testing scenario in which I ran each policy five times generating random execution times and priorities. The same ranges were used throughout.

*FCFS had to be run manually to change the ordering in which the jobs enter the queue.

test ./benchmarks fcfs 4 10 30 150

FCFS	#1	#2	#3	#4	#5
Average Turnaround Time (s)	182.5	193.75	172.50	178.75	166.75
Average Waiting Time (s)	118.75	130.00	110.00	114.75	96.25
Throughput (#jobs/sec)	0.0055	0.0052	0.0059	0.0053	0.0061

test ./benchmarks sjf 4 10 30 150

SJF	#1	#2	#3	#4	#5
Average Turnaround Time (s)	182.5	186.25	148.75	130.00	186.25
Average Waiting Time (s)	118.75	122.50	85.00	66.25	122.50
Throughput (#jobs/sec)	0.0055	0.0054	0.0067	0.0077	0.0054

test ./benchmarks priority 4 10 30 150

Priority	#1	#2	#3	#4	#5
Average Turnaround Time (s)	137.50	193.75	175.25	193.75	175.00
Average Waiting Time (s)	73.75	130.00	111.25	130.00	111.25
Throughput (#jobs/sec)	0.0073	0.0052	0.0057	0.0052	0.0057

Here is the averages of the trials by policy type:

Per Policy Trial Averages

	FCFS	SJF	Priority
Average Turnaround Time (s)	178.85	166.75	175.05
Average Waiting Time (s)	113.95	103.00	111.25
Throughput (#jobs/sec)	0.0056	0.0061	0.0058

As from the previous section, the SJF continues to reduce the response time as compared to FCFS and Priority. FCFS is difficult to really get a good analysis of from this data because my AUbatch was not setup to randomize the order that jobs enter the queue so these were inserted manually and give a five second addition to compensate for the lag in manual entry. The data for FCFS still comes out pretty close to what you might expect, sitting slightly higher than other policies in turnaround time and waiting time since no execution or priority is taken into consideration in the scheduling. FCFS will reflect entry order and execution time, and can vary drastically based on those conditions. Priority stays near the average times because we have not designed the batch jobs to run in accordance with any specific priority. As the priorities are randomized amongst varying job execution times, you really are seeing results very close to FCFS.

IV. Lessons Learned

This project certainly pushed me to learn quite a bit more about the C language. I am not a C programmer, and prior to this class had never written code in C. During my undergrad 16+ years ago I used C++, but that's too far back to be of any use. Conceptually I understood this project, but had a very challenging time implementing the appropriate C code. I mainly struggle with pointers, their various syntax, and passing them between files or functions. I feel much more comfortable with them at this point though.

The most difficult part for me was the design of the queue, which I chose to build as an array of pointers to the job structs. I probably spent over 10hrs alone trying to properly setup the queue so that the jobs could be inserted and retrieved. It was very difficult incorporating the structs, pointers to structs, and making it available to multiple files. Once I had this in place, the remaining parts of the project came along pretty quickly up until the automated performance module.

Designing the scheduling algorithms wasn't overly complicated for me. Although I would redo these algorithms to be more efficient, and make SJF and Priority preemptive if this project were intended for real usage. I used an insertion sort type algorithm since I had fallen behind on the project from building the queue.

The quit command is another area that was discussed in class that I did not have a chance to build out more. If continuing with this project, I would add user flags to determine if you wanted to immediately quit or finish running the jobs in the queue and then quit.

I liked the size of this project and the forcing function to separate files and compile with the makefile. I spent several hours at the kickoff of the project reading various opinions on how to structure a C program with the .h file and makefile compilation. It certainly helped to organize the project and and have the job structs available throughout.

V. Sample Input and Output of AUbatch

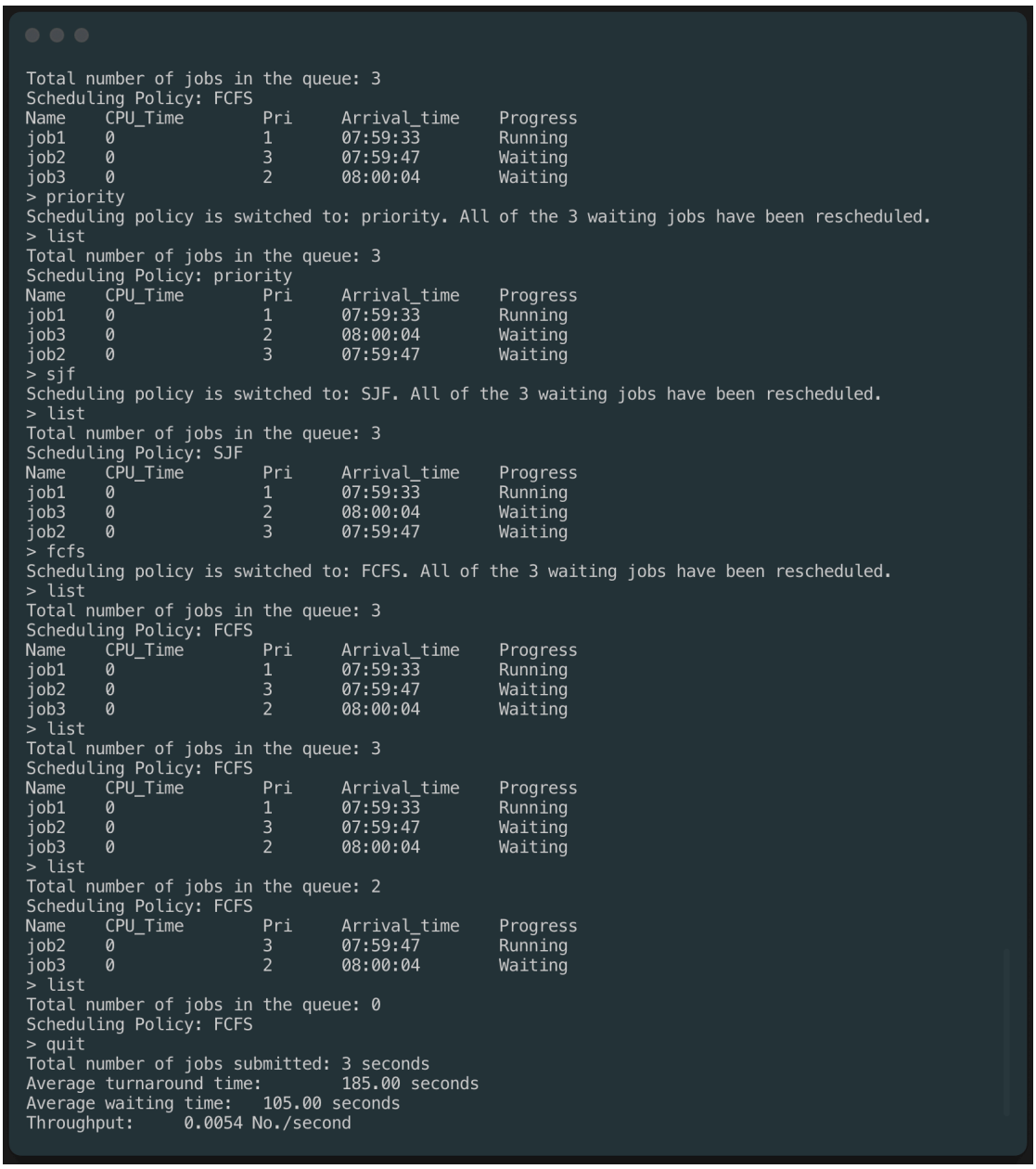
The script file for manual execution is included in the project with filename, in_out_example.script. A second script file specifically shows the benchmarks and is included as benchmarks_example.script.

Included jobs can be run with:

> run ./jobs/job1 25 3

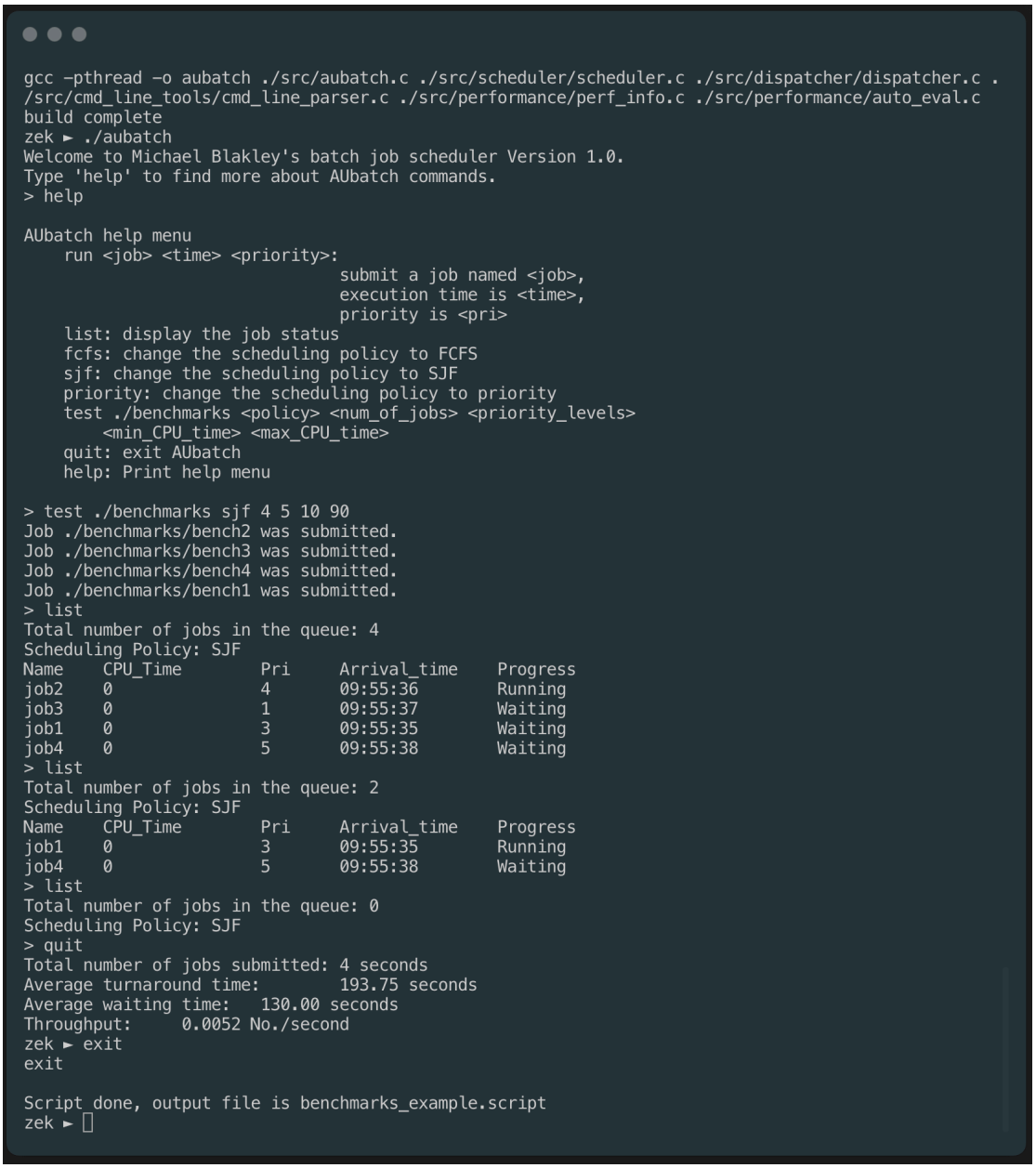
```
zek ► ./aubatch
Welcome to Michael Blakley's batch job scheduler Version 1.0.
Type 'help' to find more about AUbatch commands.
> help

AUbatch help menu
    run <job> <time> <priority>:
                                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 ./benchmarks <policy> <num_of_jobs> <priority_levels>
        <min_CPU_time> <max_CPU_time>
    quit: exit AUbatch
    help: Print help menu

> fcfs
The policy is already set to FCFS.
> sjf
Scheduling policy is switched to: SJF. All of the 0 waiting jobs have been rescheduled.
>

: Command not found
> sjf
The policy is already set to SJF.
> priority
Scheduling policy is switched to: priority. All of the 0 waiting jobs have been rescheduled.
> fcfs
Scheduling policy is switched to: FCFS. All of the 0 waiting jobs have been rescheduled.
> foo
foo
: Command not found
> run ../jobs/job3 10 1
Job ../jobs/job3 was submitted.
Expected waiting time: 0 seconds
Scheduling Policy: FCFS
> list
Total number of jobs in the queue: 1
Scheduling Policy: FCFS
Name    CPU_Time        Pri     Arrival_time    Progress
job1    0               1       07:59:33        Running
> run ../jobs/job3 30 3
Job ../jobs/job3 was submitted.
Expected waiting time: 0 seconds
Scheduling Policy: FCFS
> list
Total number of jobs in the queue: 2
Scheduling Policy: FCFS
Name    CPU_Time        Pri     Arrival_time    Progress
job1    0               1       07:59:33        Running
job2    0               3       07:59:47        Waiting
```

Screenshot #1: Start AUbatch - help, policy change, list, bad command, run job



```
Total number of jobs in the queue: 3
Scheduling Policy: FCFS
Name    CPU_Time        Pri     Arrival_time    Progress
job1    0               1       07:59:33        Running
job2    0               3       07:59:47        Waiting
job3    0               2       08:00:04        Waiting
> priority
Scheduling policy is switched to: priority. All of the 3 waiting jobs have been rescheduled.
> list
Total number of jobs in the queue: 3
Scheduling Policy: priority
Name    CPU_Time        Pri     Arrival_time    Progress
job1    0               1       07:59:33        Running
job3    0               2       08:00:04        Waiting
job2    0               3       07:59:47        Waiting
> sjf
Scheduling policy is switched to: SJF. All of the 3 waiting jobs have been rescheduled.
> list
Total number of jobs in the queue: 3
Scheduling Policy: SJF
Name    CPU_Time        Pri     Arrival_time    Progress
job1    0               1       07:59:33        Running
job3    0               2       08:00:04        Waiting
job2    0               3       07:59:47        Waiting
> fcfs
Scheduling policy is switched to: FCFS. All of the 3 waiting jobs have been rescheduled.
> list
Total number of jobs in the queue: 3
Scheduling Policy: FCFS
Name    CPU_Time        Pri     Arrival_time    Progress
job1    0               1       07:59:33        Running
job2    0               3       07:59:47        Waiting
job3    0               2       08:00:04        Waiting
> list
Total number of jobs in the queue: 3
Scheduling Policy: FCFS
Name    CPU_Time        Pri     Arrival_time    Progress
job1    0               1       07:59:33        Running
job2    0               3       07:59:47        Waiting
job3    0               2       08:00:04        Waiting
> list
Total number of jobs in the queue: 2
Scheduling Policy: FCFS
Name    CPU_Time        Pri     Arrival_time    Progress
job2    0               3       07:59:47        Running
job3    0               2       08:00:04        Waiting
> list
Total number of jobs in the queue: 0
Scheduling Policy: FCFS
> quit
Total number of jobs submitted: 3 seconds
Average turnaround time:        185.00 seconds
Average waiting time:   105.00 seconds
Throughput:     0.0054 No./second
```

Screenshot #2 - Adding more jobs to queue, changing policy to rearrange order, quit and display metrics



```
gcc -pthread -o aubatch ./src/aubatch.c ./src/scheduler/scheduler.c ./src/dispatcher/dispatcher.c .
/src/cmd_line_tools/cmd_line_parser.c ./src/performance/perf_info.c ./src/performance/auto_eval.c
build complete
zek ► ./aubatch
Welcome to Michael Blakley's batch job scheduler Version 1.0.
Type 'help' to find more about AUbatch commands.
> help

AUbatch help menu
    run <job> <time> <priority>:
                                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 ./benchmarks <policy> <num_of_jobs> <priority_levels>
        <min_CPU_time> <max_CPU_time>
    quit: exit AUbatch
    help: Print help menu

> test ./benchmarks sjf 4 5 10 90
Job ./benchmarks/bench2 was submitted.
Job ./benchmarks/bench3 was submitted.
Job ./benchmarks/bench4 was submitted.
Job ./benchmarks/bench1 was submitted.
> list
Total number of jobs in the queue: 4
Scheduling Policy: SJF
Name    CPU_Time        Pri     Arrival_time    Progress
job2    0               4       09:55:36        Running
job3    0               1       09:55:37        Waiting
job1    0               3       09:55:35        Waiting
job4    0               5       09:55:38        Waiting
> list
Total number of jobs in the queue: 2
Scheduling Policy: SJF
Name    CPU_Time        Pri     Arrival_time    Progress
job1    0               3       09:55:35        Running
job4    0               5       09:55:38        Waiting
> list
Total number of jobs in the queue: 0
Scheduling Policy: SJF
> quit
Total number of jobs submitted: 4 seconds
Average turnaround time:        193.75 seconds
Average waiting time:   130.00 seconds
Throughput:     0.0052 No./second
zek ► exit
exit

Script done, output file is benchmarks_example.script
zek ►
```

Screenshot #3 - Benchmarks execution