COMP 7500: Advanced Operating Systems
Project 3: AUBatch , A Pthread-based Batch Scheduling System
Due March 12, 2018

by: Thomas Heckwolf

Professor: Dr. Xiao Qin
Department of Computer Science and Software Engineering
Samuel Ginn College of Engineering
Auburn University
Auburn, AL, U.S.A

Contents

1 Overview **3**
- 1.1 Requirements . . . 3
- 1.2 Implementation . . . 3
- 1.3 Compiling . . . 3

2 Design **4**
- 2.1 Initial Design Diagram . . . 4
- 2.2 Final Design Diagram . . . 5
- 2.3 Simple DataFlow Diagram . . . 6

3 AUbatch.c Source Code **7**
- 3.1 aubatch.h Prototype Functions . . . 7
- 3.2 aubatch.h Global Variables and Data Structures . . . 8
- 3.3 aubatch.c Main Function . . . 9
- 3.4 aubatch.c Print Functions . . . 9
- 3.5 aubatch.c Helper Functions . . . 10
- 3.6 aubatch.c Scheduler Functions . . . 11
- 3.7 aubatch.c Dispatcher Functions . . . 17

4 batch_job Source Code **20**
- 4.1 batch_job.c . . . 20

5 Build / Compile **21**
- 5.1 Code Repository . . . 21
- 5.2 Makefile . . . 21

6 Running Program / Commands **22**
- 6.1 Running Program . . . 22
- 6.2 AUbatch Commands . . . 22
  - 6.2.1 Help Command . . . 22
  - 6.2.2 Run Command . . . 23
  - 6.2.3 List Command . . . 23
  - 6.2.4 Quit Command . . . 25
  - 6.2.5 FCFS Command . . . 25
  - 6.2.6 SJF Command . . . 25
  - 6.2.7 PRI Command . . . 26
  - 6.2.8 Quit Command . . . 27
  - 6.2.9 Test Command . . . 28

7 Testing **29**
- 7.1 test mytest fcfs 3 2 1 5 . . . 29
- 7.2 test mytest pri 3 2 1 5 . . . 30
- 7.3 test mytest sjf 3 2 1 5 . . . 31

8 Conclusion **33**
- 8.1 Conclusion Program . . . 33
  - 8.1.1 Performance metrics and workload conditions . . . 33
  - 8.1.2 Performance evaluation of the three scheduling algorithms . . . 34
  - 8.1.3 Lessons Learned . . . 34

1 Overview

The goal of this project is to produce the AUbatch program that implements a Pthread-based Batch Scheduling System.

1.1 Requirements

- To design a batch scheduling system
- To evaluate three scheduling policies/algorithms
- To implement a scheduler where two threads are synchronized
- To learn POSIX Threads Programming
- To use the pthread ligrary and execv functions
- To study and apply the condition variables using the Pthread library
- To address synchronization issues in the scheduler
- To develop micro batch-job benchmarks
- To design of performance metrics
- To assess various workload conditions
- Use the GDB tool to debug your C program in Linux
- To strengthen your debugging skill
- To improve your software development skills
- To boost your operating systems research skills

1.2 Implementation

We implemented this program with a c program called aubatch. To implement this program we also had to create a simple job program called batch_job which which takes a basic integer input and sleeps for the desired time. To verify that the program was compiled for both programs to run on a desired machine a Makefile was implemented which compiles both programs to verify binary compatibility of the running machine. When compiling we used the -pthread , -lm and std c=99 compiler and linking flags.

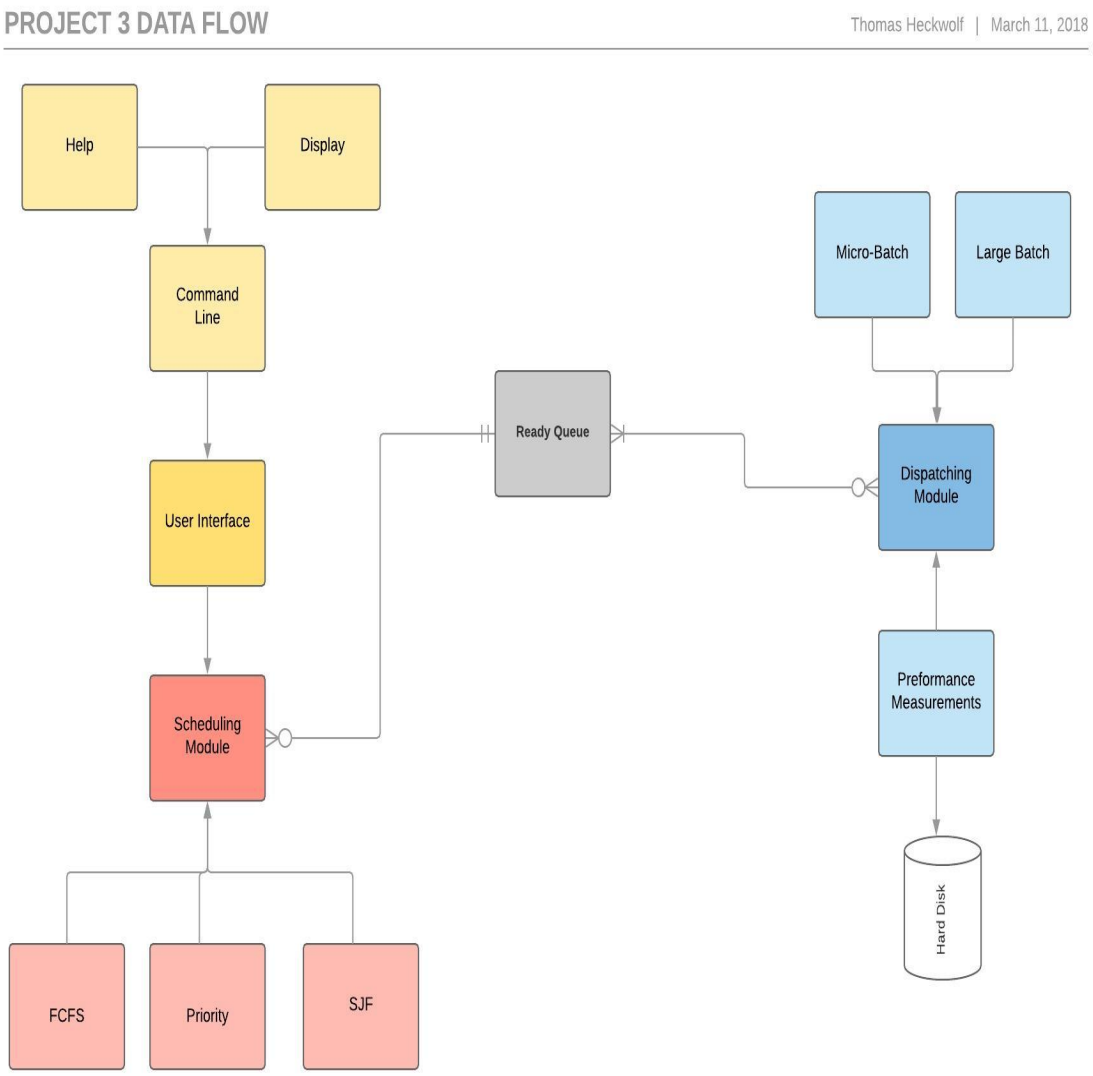
1.3 Compiling

This program has been built with CentOS 7 and Max OS X operating systems. When running this program is is essential to make within the enviroment that you wan the program to run. Both the batch job and aubatch programs need the binary executable file to be compatible with the host system running the programs.

2 Design

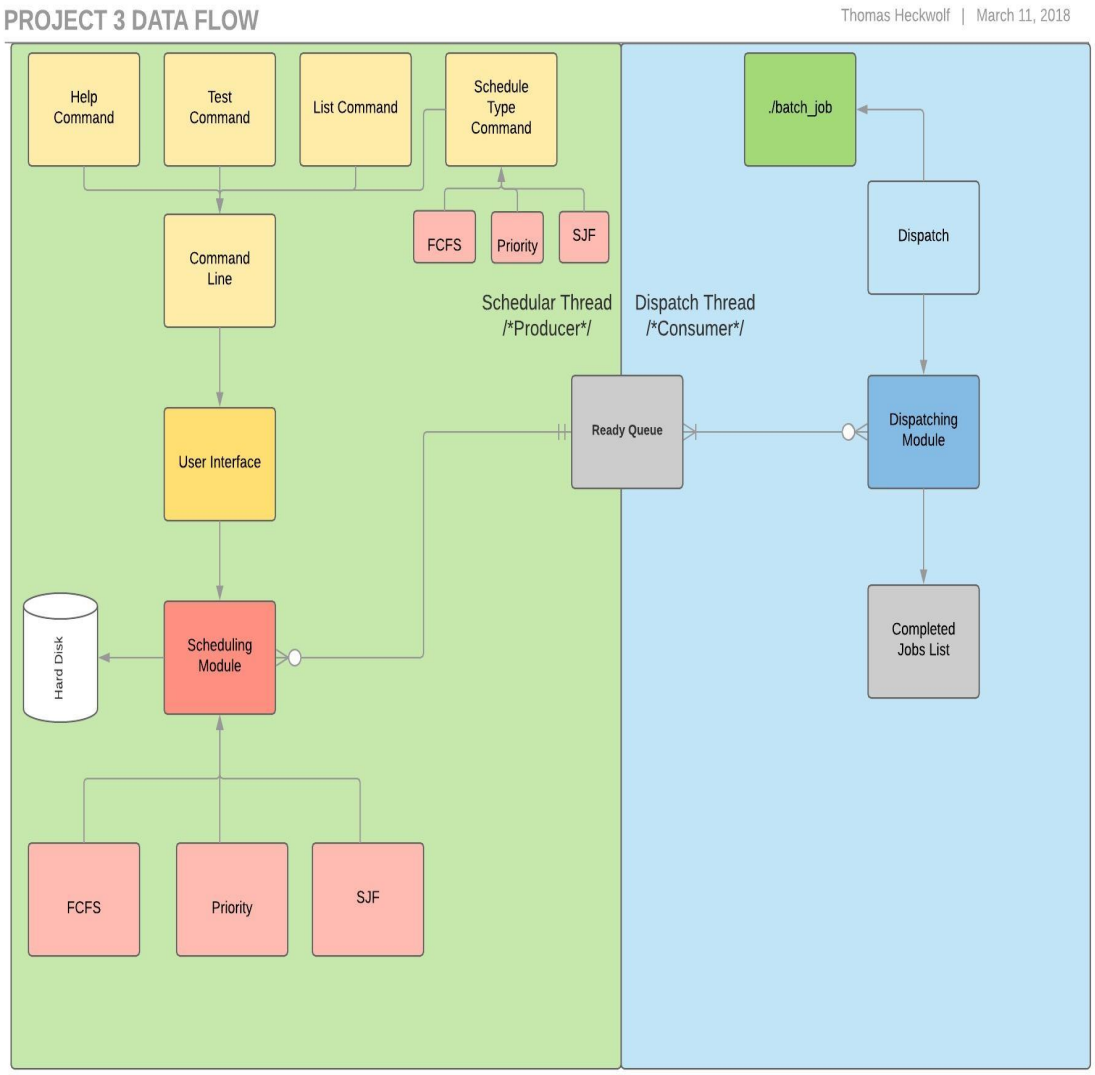
2.1 Initial Design Diagram

This is the initial data flow design diagram submitted.



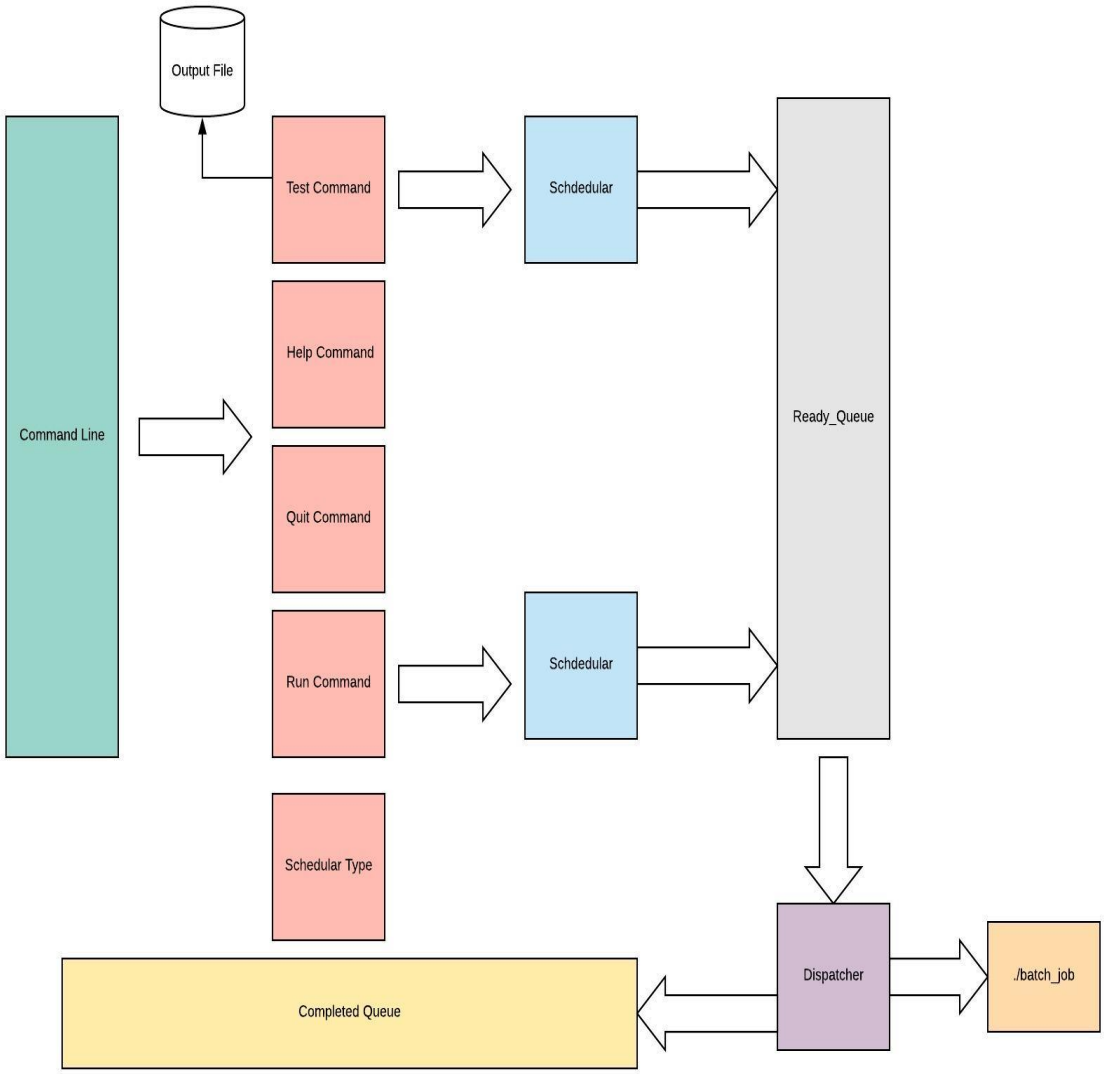
2.2 Final Design Diagram

This is the final design diagram which represents the AUbatch program.



2.3 Simple DataFlow Diagram

This is a simple data flow diagram showing the flow of execution threw the AUbatch program.



3 AUbatch.c Source Code

3.1 aubatch.h Prototype Functions

Prototype Functions located in the aubatch.h header file.

```
1
2  /************ Function Prototypes ********************/
3  /*Helper Functions*/
4  void initReadyQueue();
5  void calculateJobStats();
6  void clearJob(struct Job *job);
7  void clearComplete_queue();
8  void clearStats();
9  void inc_reset_tail();
10 /*Print Functions*/
11 void printReadyQueue();
12 void printStats();
13 void printCompleteQueue();
14 void printFileStats(FILE *fp);
15
16 /*Producer - Scheduler*/
17 void *scheduler();
18
19 void schedule(struct Job *job);
20
21 void helpInfo();
22
23 void commandline();
24
25 void commandParser(char *argument, char *param[], int *paramSize);
26
27 void insertionSortSJF(int *currentHead);
28
29 void insertionSortPri(int *currentHead);
30
31 void insertionSortFCFS(int *currentHead);
32
33 /*Consumer - Dispatcher*/
34 void *dispatcher();
35
36 void dispatch(struct Job *job);
37
38 void insertCompleteQueue(struct Job *job);
39
40 void execute(JobPtr currentJobPtr, struct Job *completedJobPtr, JobPtr jobPtr);
41
42 /*Commands*/
43 void run_command(char *const *commandv, int commandc);
44
45 void quit_command();
46
47 void sjf_command(enum Commands *command, int *tempHead);
48
49 void pri_command(enum Commands *command, int *tempHead);
50
51 void fcfs_command(enum Commands *command, int *tempHead);
52
53 void list_command(char *commandv, int commandc);
54
55 void help_command(char *commandv, int commandc);
56
57 /************ End Function Prototypes ********************/
58
59 void test_command(char *const *commandv, size_t bufsize, int commandc);
60
61 void send_job(char *const *commandv);
```

```
62
63 JobPtr createJob(char *const *commandv);
64
65
66 #endif //AUBATCH_AUBATCH_H
```

3.2 aubatch.h Global Variables and Data Structures

This is the location where the global variables and data structures used by the AUbatch program reside. Both the scheduler and dispatcher thread can access the data and structures.

```
1  #define MAX_JOB_NUMBER  400
2
3  enum Commands {
4      init, help, run, list, fcfs, sjf, pri, test, end
5  };
6  enum Scheduler {
7      FCFS, SJF, PRI
8  };
9  enum Commands command = init;
10 enum Scheduler prev_scheduleType = FCFS;
11 enum Scheduler scheduleType = FCFS;
12
13 struct Job {
14     char name[50];
15     int cputime;
16     int actualCpuTime;
17     int priority;
18     time_t arrival_time;
19     float turnarount_time;
20     float wait_time;
21     struct tm time;
22     char status[10];
23     int number;
24 };
25 struct Benchmark {
26     char name[50];
27     char sType[50];
28     int num_of_jobs;
29     int priority_levels;
30     int min_CPU_time;
31     int max_CPU_time;
32 };
33 struct Job ready_queue[MAX_JOB_NUMBER];
34 struct complete_queue {
35     struct Job job;
36     struct complete_queue *next_job;
37 };
38 struct complete_queue *chead = NULL;
39 struct Job *RunningJob = NULL;
40 typedef struct Job *JobPtr;
41 /*Global Definitions*/
42 int totalcount = 0;
43 int testDone = 0;
44 int count = 0;
45 int head = 0;
46 int tail = 0;
47 int testRunning = 0;
48 float avgTurnaround = 0;
49 float avgWaitTime = 0;
50 float avgCpuTime = 0;
51 int totoalCompletedJobs = 0;
52 float avgThroughput = 0;
53 FILE *fp;
```

3.3 aubatch.c Main Function

This is the main function of the program. It spawns the threads using pthread_create and waits for the threads to exit the program with pthread_join.

```
/************ Main Function ********************/
int main(int argc, char *argv[]) {
    //int i;
    //JobPtr RunningJobPtr = (JobPtr) malloc(sizeof(struct Job));
    initReadyQueue();
    pthread_t threads[3];
    pthread_attr_t attr;
/* Initialize mutex and condition variable objects */
    pthread_mutex_init(&queue_mutex, NULL);
    pthread_cond_init(&queue_threshold_cv, NULL);
    pthread_mutex_init(&test_mutex, NULL);
    pthread_cond_init(&test_threshold_cv, NULL);
/* For portability, explicitly create threads in a joinable state */
    pthread_attr_init(&attr);
    pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_JOINABLE);
    pthread_create(&threads[1], &attr, scheduler, NULL);
    pthread_create(&threads[2], &attr, dispatcher, NULL);

    pthread_join(threads[1], NULL);
    pthread_join(threads[2], NULL);

/* Clean up and exit */
    pthread_attr_destroy(&attr);
    pthread_mutex_destroy(&queue_mutex);
    pthread_cond_destroy(&queue_threshold_cv);
    pthread_exit(NULL);
}
```

3.4 aubatch.c Print Functions

```
/*Print Functions*/
void printReadyQueue() {
    int tempCount;
    printf("Total number of jobs in the queue: %d\n", count);
    char tempStr[5];
    switch (scheduleType) {
        case FCFS:
            strcpy(tempStr, "FCFS");
            break;
        case SJF:
            strcpy(tempStr, "SJF");
            break;
        case PRI:
            strcpy(tempStr, "PRI");
            break;
    }
    printf("Scheduling Policy: %s.\n", tempStr);


    if (RunningJob != NULL) {
        if (strcmp(RunningJob->status, "complete") != 0) {
            printf("Running Job:\n");
            printf("Name \t\tCPU_Time \tPri \tProgress\n");
            printf("%s %10d %14d %12s\n", RunningJob->name, RunningJob->cputime,
                   RunningJob->priority,
                   RunningJob->status);
            printf("\n");
        } else {

        }
    }

    for (int i = 0; i < MAX_JOB_NUMBER; i++) {
```

```
34          if (strcmp(ready_queue[i].status, "") == 0 || strcmp(ready_queue[i].status, "complete") == 0
     ||
35              strcmp(ready_queue[i].status, "init") == 0) {
36              continue;
37          } else {
38              printf("Ready Queue:\n");
39              printf("Name \t\t\tCPU_Time \tPri \tArrival_time \tProgress\n");
40              printf("%s %20d %10d %10d:%d:%d %18s\n", ready_queue[i].name, ready_queue[i].cputime,
41                      ready_queue[i].priority,
42                      ready_queue[i].time.tm_hour, ready_queue[i].time.tm_min, ready_queue[i].time.
    tm_sec,
43                      ready_queue[i].status);
44          }
45      }
46      tempCount++;
47
48  }
49
50  void printFileStats(FILE *fp) {
51      fprintf(fp, "Total number of job submitted: %d\n", totalcount);
52      fprintf(fp, "Average turnaround time: %f seconds\n", avgTurnaround);
53      fprintf(fp, "Average CPU time: %f seconds\n", avgCpuTime);
54      fprintf(fp, "Average waiting time: %f seconds\n", avgWaitTime);
55      fprintf(fp, "Throughput: %f No./second*/\n", avgThroughput);
56  }
57
58  void printCompleteQueue() {
59      int tempCount;
60      struct complete_queue *temp = chead;
61      if (temp == NULL) {}
62      else {
63          printf("Completed Jobs:\n");
64          printf("Name \t\t\tCPU_Time \tPri \tArrival_time \tProgress\n");
65          while (temp != NULL) {
66              printf("%s %20d %10d %10d:%d:%d %12s\n", temp->job.name, temp->job.cputime,
67                      temp->job.priority,
68                      temp->job.time.tm_hour, temp->job.time.tm_min, temp->job.time.tm_sec,
69                      temp->job.status);
70              temp = temp->next_job;
71          }
72      }
73  }
74
75  void printStats() {
76      printf("\nTotal number of job submitted: %d\n", totalcount);
77      printf("Total number of job completed: %d\n", totoalCompletedJobs);
78      printf("Average turnaround time: %f seconds\n", avgTurnaround);
79      printf("Average CPU time:%f seconds\n", avgCpuTime);
80      printf("Average waiting time:%f seconds\n", avgWaitTime);
81      printf("Throughput: %f No./second\n", avgThroughput);
82      return;
83  }
```

3.5 aubatch.c Helper Functions

Helper functions are created to allow for better readability of the code

```
1   /*Helper Functions*/
2   void initReadyQueue() {
3       pthread_mutex_lock(&queue_mutex);
4       for (int i = 0; i < MAX_JOB_NUMBER; i++) {
5           strcpy(ready_queue[i].name, "init");
6           ready_queue[i].cputime = 99999;
7           ready_queue[i].priority = 99999;
8       }
9       pthread_mutex_unlock(&queue_mutex);
10  }
11
12  void calculateJobStats() {
13      int tempCount = 0;
14      float tempTurn = 0;
15      float tempWait = 0;
```

```
16      float tempCpu = 0;
17
18      struct complete_queue *temp = chead;
19      if (temp == NULL) {}
20      else {
21          while (temp != NULL) {
22              tempCpu += temp->job.cputime;
23              tempWait += temp->job.wait_time;
24              tempTurn += temp->job.turnaround_time;
25              tempCount++;
26              temp = temp->next_job;
27          }
28          totalcount = tempCount;
29          if (totalcount == 0) {}
30          else {
31              avgTurnaround = tempTurn / totalcount;
32              avgWaitTime = tempWait / totalcount;
33              avgCpuTime = tempCpu / totalcount;
34              totoalCompletedJobs = totalcount;
35              avgThroughput = (1 / (avgTurnaround / totalcount));
36          }
37      }
38
39
40  }
41
42  void clearJob(struct Job *job) {
43      strcpy(job->name, "");
44      job->cputime = 99999;
45      job->priority = 99999;
46      job->arrival_time = 0;
47      job->turnaround_time = 0;
48      job->wait_time = 0;
49      strcpy(job->status, "init");
50      job->number = 0;
51  };
52
53  void clearStats() {
54      avgTurnaround = 0;
55      avgWaitTime = 0;
56      avgCpuTime = 0;
57      totoalCompletedJobs = 0;
58      avgThroughput = 0;
59      totalcount = 0;
60  }
61
62  void clearComplete_queue() {
63      struct complete_queue *next;
64      struct complete_queue *current = chead;
65      struct complete_queue *temp = NULL;
66      while (current != NULL) {
67          next = current->next_job;
68          free(current);
69          current = next;
70      }
71      chead = temp;
72  };
73  void inc_reset_tail() {
74      tail++;
75      if (tail == MAX_JOB_NUMBER) {
76          tail = 0;
77      }
78  }
```

3.6 aubatch.c Scheduler Functions

Scheduler functions in the AUbatch program are the consumer producer thread of the program. The scheduler thread implements the user interface which is a command line interface, and schedules jobs for the dispatcher thread in the ready queue which is a round robin queue of structures.

```
/*Producder*/
void *scheduler() {
    printf("Welcome to Thomas Heckwolfs's batch job scheduler Version 1.0 \n Type ?help? to find more about AUbatch commands.\n");
    commandline();
    pthread_exit(NULL);
}

void schedule(struct Job *job) {
    time_t t = time(NULL);
    struct tm *tm = localtime(&t);
    int tempHead;
    job->arrival_time = t;
    job->time = *tm;
    pthread_mutex_lock(&queue_mutex);
    pthread_cond_signal(&queue_threshold_cv);
    count++;
    ready_queue[head] = *job;
    head++;
    if (head == MAX_JOB_NUMBER) {
        head = 0;
    }
    switch (scheduleType) {
        case SJF:
            tail = 0;
            insertionSortSJF(&tempHead);
            head = tempHead;
            break;
        case PRI:
            insertionSortPri(&tempHead);
            tail = 0;
            head = tempHead;
            break;
        case FCFS:
        default:
            break;
    }

    pthread_mutex_unlock(&queue_mutex);
    return;


}

void helpInfo() {
    command = help;
    printf("Run JOB:\n");
    printf("run <batch_job | job> <time> <pri>: submit a job named <job>,\n");
    printf("\texecution time is <time>,\n");
    printf("\tpriority is <pri>.\n");
    printf("list: display the job status.\n");
    printf("Scheduling policy:\n");
    printf("\tfcfs: change the scheduling policy to FCFS.\n");
    printf("\tsjf: change the scheduling policy to SJF.\n");
    printf("\tpriority: change the scheduling policy to priority.\n");
    printf("test <benchmark> <policy> <num_of_jobs> <priority_levels>\n\t<min_CPU_time> <max_CPU_time>\n");
    printf("Quit Program:\n");
    printf("quit");
    return;
}

/*Command*/
void fcfs_command(enum Commands *command, int *tempHead) {
    (*command) = fcfs;
    scheduleType = FCFS;
    insertionSortFCFS(tempHead);
    tail = 0;
    head = (*tempHead);
}

void pri_command(enum Commands *command, int *tempHead) {
    (*command) = pri;
```

```
72      scheduleType = PRI;
73      insertionSortPri(tempHead);
74      tail = 0;
75      head = (*tempHead);
76  }
77
78  void sjf_command(enum Commands *command, int *tempHead) {
79      (*command) = sjf;
80      scheduleType = SJF;
81      insertionSortSJF(tempHead);
82      tail = 0;
83      head = (*tempHead);
84  }
85
86  void quit_command() {
87      calculateJobStats();
88      printf("Total number of job submitted: %d\n", totalcount);
89      printf("Total number of job completed: %d\n", totoalCompletedJobs);
90      printf("Average turnaround time: %f seconds\n", avgTurnaround);
91      printf("Average CPU time:%f seconds\n", avgCpuTime);
92      printf("Average waiting time:%f seconds\n", avgWaitTime);
93      printf("Throughput: %f No./second\n", avgThroughput);
94      command = end;
95      pthread_cond_signal(&queue_threshold_cv);
96      pthread_exit(NULL);
97  }
98
99  void run_command(char *const *commandv, int commandc) {
100     command = run;
101     if (commandc < 4 || commandc > 4) {
102         printf("Error: Number of Run %d of 4 parameters entered.\n", commandc);
103         printf("run <job> <time> <pri>: submit a job named <job>,\n");
104     } else {
105         send_job(commandv);
106     }
107 }
108
109 void send_job(char *const *commandv) {
110     JobPtr job = createJob(commandv);
111     strcpy(job->status, "wait");
112     schedule(job);
113 }
114
115 JobPtr createJob(char *const *commandv) {
116     JobPtr job = (JobPtr) malloc(sizeof(struct Job));
117     char my_job[10];
118     strcpy(job->name, commandv[1]);
119     job->cputime = atoi(commandv[2]);
120     job->priority = atoi(commandv[3]);
121     if (strcmp(job->name, "batch_job") == 0){
122         sprintf(my_job, "%s", job->name);
123     }
124     else{
125         if (fp){
126             fprintf(fp,"Dispatcher only supports the ./batch_job program\n");
127             fprintf(fp,"%s replaced with ./batch_job program\n",job->name);
128         }
129         printf("Dispatcher only supports the ./batch_job program\n");
130         printf("%s replaced with ./batch_job program\n",job->name);
131         sprintf(my_job, "%s", "batch_job");
132     }
133     job->wait_time = 0;
134     job->arrival_time = 0;
135     job->turnaround_time = 0;
136     job->number = totalcount++;
137     return job;
138 }
139
140 void list_command(char *commandv, int commandc) {
141     command = list;
142     if (commandc == 2) {
143         if(strcmp(commandv, "-r") == 0)
144         {
```

```
145             while (count > 0) {
146                 printReadyQueue();
147                 printCompleteQueue();
148                 sleep(2);
149             }
150         }
151         else{
152             printReadyQueue();
153             printCompleteQueue();
154         }
155     } else {
156         printReadyQueue();
157         printCompleteQueue();
158     }
159 }
160
161 /*Command Line and Parser*/
162 void commandline() {
163     printf("\n>");
164     char *buffer;
165     char *commandv[7] = {NULL};
166     size_t bufsize = 32;
167     int commandc = -1;
168     buffer = (char *) malloc(bufsize * sizeof(char));
169     enum Commands command = init;
170     int tempHead;
171     do {
172         if (testRunning == 1 &&  count > 0) {
173             printf("\nBenchmark Test Currently Running\n");
174             printf("Limited Commands Allowed\n");
175             printf("list , help , quit\n");
176             printf("\n>");
177         }
178         if (testDone == 1) {
179             printf("Benchmark Test Done Running\n");
180             printStats();
181             printf("\n>");
182             testDone = 0;
183         }
184         fgets(buffer, bufsize, stdin);
185         commandParser(buffer, commandv, &commandc);
186
187         if (commandv[0] != NULL) {
188             if (strcmp(commandv[0], "help") == 0) {
189                 help_command(commandv[1], commandc);
190             } else if (strcmp(commandv[0], "run") == 0) {
191                 if (testRunning != 1) {
192                     run_command(commandv, commandc);
193                 } else { printf("Test Running: Command not accepted;"); }
194             } else if (strcmp(commandv[0], "quit") == 0 || strcmp(commandv[0], "exit") == 0) {
195                 if (fp) {
196                     fclose(fp);
197                 }
198                 quit_command();
199                 testRunning = 0;
200             } else if (strcmp(commandv[0], "sjf") == 0) {
201                 if (testRunning != 1) {
202                     sjf_command(&command, &tempHead);
203                 } else { printf("Test Running: Command not accepted;"); }
204             } else if (strcmp(commandv[0], "pri") == 0) {
205                 if (testRunning != 1) {
206                     pri_command(&command, &tempHead);
207                 } else { printf("Test Running: Command not accepted;"); }
208             } else if (strcmp(commandv[0], "list") == 0) {
209                 list_command(commandv[1], commandc);
210             } else if (strcmp(commandv[0], "fcfs") == 0) {
211                 if (testRunning != 1) {
212                     fcfs_command(&command, &tempHead);
213                 } else { printf("Test Running: Command not accepted;"); }
214             } else if (strcmp(commandv[0], "test") == 0) {
215                 test_command(commandv, bufsize, commandc);
216             } else if (strcmp(commandv[0], "job") == 0 || strcmp(commandv[0], "job") == 0) {
217                 printf("Error: No Command Job.\n");
```

```
218                 printf("run <job> <time> <pri>: submit a job named <job>,\n");
219             } else {
220                 printf("Error: Command Error Please Check Command Parameters.\n");
221             }
222         }
223         if (count == 0 && testDone == 1) {
224             testRunning = 0;
225             calculateJobStats();
226             printFileStats(fp);
227             command = test;
228             if (fp) {
229                 fclose(fp);
230             }
231         }
232         printf("\n>");
233     } while (command != end);
234     return;
235 }
236
237 void help_command(char *commandv, int commandc) {
238     if (commandc == 1) {
239         helpInfo();
240     } else if (commandc == 2) {
241         if (strcmp(commandv, "-list") == 0 || strcmp(commandv, "list") == 0 || strcmp(commandv, "-l"
    ) == 0) {
242             printf("list: display the job status.\n");
243         } else if (strcmp(commandv, "run") == 0 || strcmp(commandv, "-run") == 0 || strcmp(commandv,
     "-r") == 0) {
244             printf("run <job> <time> <pri>: submit a job named <job>,\n");
245         } else if (strcmp(commandv, "pri") == 0 || strcmp(commandv, "-p") == 0 || strcmp(commandv, "
    -pri") == 0) {
246             printf("Scheduling policy:\n");
247             printf("\tfcfs: change the scheduling policy to FCFS.\n");
248             printf("\tsjf: change the scheduling policy to SJF.\n");
249             printf("\tpriority: change the scheduling policy to priority.\n");
250         } else if (strcmp(commandv, "fcfs") == 0 || strcmp(commandv, "-f") == 0 || strcmp(commandv,
    "-fcfs") == 0) {
251             printf("Scheduling policy:\n");
252             printf("\tfcfs: change the scheduling policy to FCFS.\n");
253         } else if (strcmp(commandv, "sjf") == 0 || strcmp(commandv, "-s") == 0 || strcmp(commandv, "
    -sjf") == 0) {
254             printf("Scheduling policy:\n");
255             printf("\tsjf: change the scheduling policy to SJF.\n");
256         } else if (strcmp(commandv, "test") == 0 || strcmp(commandv, "-test") == 0 || strcmp(
    commandv, "-t") == 0) {
257             printf("Test Command:\n");
258             printf("test <benchmark> <policy> <num_of_jobs> <priority_levels>\n\t<min_CPU_time> <
    max_CPU_time>");
259         } else if (strcmp(commandv, "quit") == 0 || strcmp(commandv, "-quit") == 0 || strcmp(
    commandv, "-q") == 0) {
260             printf("Quit Command:Exit the Program.\n");
261         }
262     } else {
263         printf("Error: Help Command.\n");
264         printf("help list , || run , sjf , pri , fcfs");
265         printf("help -l ,   || run , sjf , pri , fcfs");
266     }
267 }
268
269 void test_command(char *const *commandv, size_t bufsize, int commandc) {
270     command = test;
271     if (commandc < 7 || commandc > 7) {
272         printf("Error: Test Command.\n");
273         printf("test <benchmark> <policy> <num_of_jobs> <priority_levels>\n");
274         printf("\t<min_CPU_time> <max_CPU_time>");
275     } else {
276         printf("Warning: Test Command.\n");
277         printf("Previous Statistics and Queue will be cleared.\n");
278         printf("Only list and Help command will be allowed to be entered.\n");
279         testRunning = 1;
280         clearComplete_queue();
281         initReadyQueue();
282         clearStats();
```

```
283         char filename[25];
284         struct Benchmark *benchmark;
285         JobPtr job = (JobPtr) malloc(sizeof(struct Job));
286         benchmark = (struct Benchmark *) malloc(bufsize * sizeof(struct Benchmark));
287         strcpy(benchmark->name, commandv[1]);
288         strcpy(benchmark->sType, commandv[2]);
289         sprintf(filename, "%s_%s.txt", benchmark->name,benchmark->sType);
290         fp = fopen(filename, "w");
291         if (strcmp(benchmark->sType, "fcfs") == 0) {
292             scheduleType = FCFS;
293         } else if (strcmp(benchmark->sType, "pri") == 0) {
294             scheduleType = PRI;
295         } else if (strcmp(benchmark->sType, "sjf") == 0) {
296             scheduleType = SJF;
297         }
298         benchmark->num_of_jobs = atoi(commandv[3]);
299         benchmark->priority_levels = atoi(commandv[4]);
300         benchmark->min_CPU_time = atoi(commandv[5]);
301         benchmark->max_CPU_time = atoi(commandv[6]);
302         int pri_level = benchmark->priority_levels;
303         int max_cpu = benchmark->max_CPU_time;
304         int min_cpu = benchmark->min_CPU_time;
305         int randBurst;
306         srand(time(NULL));
307         fprintf(fp, "Submitted Test Jobs: %s\n",benchmark->sType);
308         fprintf(fp, "Number of Jobs:%d,Priority Levels:%d,Min CPU:%d Max CPU:%d\n",benchmark->
    num_of_jobs,benchmark->priority_levels,benchmark->min_CPU_time,benchmark->max_CPU_time);
309         fprintf(fp, "Name \t\tCPU_Time \tPri \tProgress\n");
310         for (int i = 0; i < benchmark->num_of_jobs; i++) {
311             if ((max_cpu - min_cpu) == 0){
312                 randBurst = rand() % max_cpu + 1;
313             }
314             else {
315                 randBurst = min_cpu + rand() % (max_cpu - min_cpu);
316             }
317             int randPri = rand() % pri_level + 1;
318             strcpy(job->name, "batch_job");
319             job->cputime = randBurst;
320             job->priority = randPri;
321             job->wait_time = 0;
322             job->arrival_time = 0;
323             job->turnarount_time = 0;
324             job->number = totalcount++;
325             strcpy(job->status, "wait");
326             schedule(job);
327             fprintf(fp, "%s %10d %14d %12s\n", job->name, job->cputime, job->priority, job->status);
328 
329         }
330         free(benchmark);
331 
332 
333     }
334 
335 
336 }
337 
338 void commandParser(char *argument, char *param[], int *paramSize) {
339     char *arg;
340     int i = 0;
341     argument[strcspn(argument, "\r\n")] = 0;
342     arg = strtok(argument, " ");
343     param[i] = arg;
344     while (arg != NULL) {
345         i++;
346         arg = strtok(NULL, " ");
347         param[i] = arg;
348     }
349     *paramSize = i;
350     return;
351 
352 }
353 
354 void insertionSortSJF(int *currentHead) {
```

```
355     int j;
356     JobPtr tempJob = (JobPtr) malloc(sizeof(struct Job));
357     for (int i = 0; i < MAX_JOB_NUMBER; i++) {
358         j = i;
359         while (j > 0 && ready_queue[j].cputime < ready_queue[j - 1].cputime) {
360             *tempJob = ready_queue[j];
361             ready_queue[j] = ready_queue[j - 1];
362             ready_queue[j - 1] = *tempJob;
363             j--;
364         }
365     }
366     j = 0;
367     for (int i = 0; i < MAX_JOB_NUMBER; i++) {
368         if (ready_queue[i].cputime != 99999) {
369             j++;
370         }
371     }
372     *currentHead = j;
373 
374     free(tempJob);
375 }
376 
377 void insertionSortPri(int *currentHead) {
378     int j;
379     JobPtr tempJob = (JobPtr) malloc(sizeof(struct Job));
380     for (int i = 0; i < MAX_JOB_NUMBER; i++) {
381         j = i;
382         while (j > 0 && ready_queue[j].priority < ready_queue[j - 1].priority) {
383             *tempJob = ready_queue[j];
384             ready_queue[j] = ready_queue[j - 1];
385             ready_queue[j - 1] = *tempJob;
386             j--;
387         }
388     }
389     j = 0;
390     for (int i = 0; i < MAX_JOB_NUMBER; i++) {
391         if (ready_queue[i].priority != 99999) {
392             j++;
393         }
394     }
395     *currentHead = j;
396 
397     free(tempJob);
398 }
399 
400 void insertionSortFCFS(int *currentHead) {
401     int j;
402     JobPtr tempJob = (JobPtr) malloc(sizeof(struct Job));
403     for (int i = 0; i < MAX_JOB_NUMBER; i++) {
404         j = i;
405         while (j > 0 && ready_queue[j].number < ready_queue[j - 1].number) {
406             *tempJob = ready_queue[j];
407             ready_queue[j] = ready_queue[j - 1];
408             ready_queue[j - 1] = *tempJob;
409             j--;
410         }
411     }
412     j = 0;
413     for (int i = 0; i < MAX_JOB_NUMBER; i++) {
414         if (ready_queue[i].number != 99999) {
415             j++;
416         }
417     }
418     *currentHead = j;
419     free(tempJob);
420 }
```

3.7 aubatch.c Dispatcher Functions

The dispatcher functions handle the dispatching and retrieval of the executable jobs from the ready queue. The dispatcher calls the external batch_job program. Once the jobs complete the dispatcher inserts the jobs into a link list of

completed jobs.

```
/*Consumer*/
void *dispatcher() {
    JobPtr currentJobPtr = (JobPtr) malloc(sizeof(struct Job));
    JobPtr completedJobPtr = (JobPtr) malloc(sizeof(struct Job));
    JobPtr jobPtr = (JobPtr) malloc(sizeof(struct Job));
    pthread_mutex_lock(&queue_mutex);
    while (command != end) {
        if (count == 0) {
            pthread_cond_wait(&queue_threshold_cv, &queue_mutex);
            if (count == 0) {
                pthread_exit(NULL);
            }
        }
        count--;
        if (strcmp(ready_queue[tail].status, "wait") == 0) {
            execute(currentJobPtr, completedJobPtr, jobPtr);
        } else {
            inc_reset_tail();
        }
        if (testRunning == 1 &&  count == 0) {
            printf("Benchmark Test Done Running\n");
            printf("Please Press Enter for Statistics\n");
            testDone = 1;
        }
    }
    pthread_exit(NULL);
}

void execute(JobPtr currentJobPtr, struct Job *completedJobPtr, JobPtr jobPtr) {
    strcpy(ready_queue[tail].status, "run");
    RunningJob = currentJobPtr;
    *jobPtr = ready_queue[tail];
    memcpy(currentJobPtr, jobPtr, sizeof(struct Job));
    clearJob(jobPtr);
    ready_queue[tail] = *jobPtr;
    inc_reset_tail();

    pthread_mutex_unlock(&queue_mutex);

    time_t start = time(NULL);
    dispatch(currentJobPtr);
    time_t end = time(NULL);
    int cpu = end - start;

    strcpy(currentJobPtr->status, "complete");
    currentJobPtr->actualCpuTime = cpu;
    currentJobPtr->turnaround_time = end - currentJobPtr->arrival_time;
    currentJobPtr->wait_time = currentJobPtr->turnaround_time - cpu;
    memcpy(completedJobPtr, currentJobPtr, sizeof(struct Job));

    insertCompleteQueue(completedJobPtr);
}


void dispatch(struct Job *job) {
    pid_t pid, c_pid;
    int status;
    char my_job[10];
    char buffer[3]; // 3 digit buffer
    sprintf(buffer, "%d", job->cputime);

    if (strcmp(job->name, "batch_job") == 0){
        sprintf(my_job, "%s", job->name);
    }
    else{
        sprintf(my_job, "%s", "batch_job");
    }
    char *const parmList[] = {my_job, buffer, NULL};
    if ((pid = fork()) == -1)
        perror("fork error");
    else if (pid == 0) {
```

```
72          execv(my_job, parmList);
73      } else if (pid > 0) {
74          //from manual page of https://linux.die.net/man/2/wait
75          do {
76              c_pid = waitpid(pid, &status, WUNTRACED | WCONTINUED);
77              if (c_pid == -1) {
78                  perror("waitpid");
79                  exit(EXIT_FAILURE);
80              }
81          } while (!WIFEXITED(status) && !WIFSIGNALED(status));
82      }
83
84
85  }
86  void insertCompleteQueue(struct Job *job) {
87      struct complete_queue *job_node = (struct complete_queue *) malloc(sizeof(struct complete_queue)
        );
88      struct complete_queue *last = chead;
89      job_node->job = *job;
90      job_node->next_job = NULL;
91      if (chead == NULL) {
92          chead = job_node;
93          return;
94      }
95      while (last->next_job != NULL) {
96          last = last->next_job;
97      }
98
99      last->next_job = job_node;
100     return;
101 }
```

4 batch_job Source Code

4.1 batch_job.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
int main(int argc, char *argv[])
{
    int seconds = atoi(argv[1]);
    sleep(seconds);
    return 0;
}
```

Simple program that sleeps for an amount of time which is passed to the program.

5 Build / Compile

5.1 Code Repository

This code was source controlled with git and will be posted on github. The below link will be available after the grade for the assignment is posted.
https://github.com/theckwolf/AUbatch

5.2 Makefile

Makefile compiles and cleans the object files for the AUbatch program.

```
CC =gcc
CC_FLAGS = -std=c99 -pthread -lm
AU_FILES = aubatch.c
JOB_FILES = batch\_job.c
OUT_EXE = aubatch
JOB_OUT_EXE = batch\_job

all: aubatch job

aubatch: \$(AU_FILES)
  \$(CC)\$(CC_FLAGS) -o \$(OUT_EXE) \$(AU_FILES)

job: \$(JOB_FILES)
  \$(CC)\$(CC_FLAGS) -o \$(JOB_OUT_EXE) $(JOB_FILES)
clean:
  rm -f *.o aubatch
  rm -f *.o batch_job
```

Script started on Fri 12 Jan 2018 11:51:11 AM CST
[tbh0020@CentOS7]$ make
gcc -std=c99 -pthread -lm -o aubatch aubatch.c
gcc -std=c99 -pthread -lm -o batch_job batch_job.c

Script started on Fri 12 Jan 2018 11:51:11 AM CST
[tbh0020@CentOS7]$ make clean
rm -f *.o aubatch
rm -f *.o batch_job

6 Running Program / Commands

6.1 Running Program

The below output is an example of how to start the AUbatch program.

```
tbh0020@CentOS7 AuBatch]$ ./aubatch
Welcome to Thomas Heckwolfs's batch job scheduler Version 1.0
 Type 'help' to find more about AUbatch commands.
```

6.2 AUbatch Commands

AUbatch commands are Help,Quit,Run,FCFS,Pri,SJF,Test

6.2.1 Help Command

The help command will display the available options that the AUbatch program offers.

```
>help
Run JOB:
run <batch_job | job> <time> <pri>: submit a job named <job>,
execution time is <time>,
priority is <pri>.
list: display the job status.
Scheduling policy:
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> <priority_levels>
<min_CPU_time> <max_CPU_time>
Quit Program:
quit
>help -q
Quit Command:Exit the Program.
```

```
>help
Run JOB:
run <batch_job | job> <time> <pri>: submit a job named <job>,
execution time is <time>,
priority is <pri>.
list: display the job status.
Scheduling policy:
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> <priority_levels>
<min_CPU_time> <max_CPU_time>
Quit Program:
quit
>help -list
list: display the job status.
```

```
>help -r
run <job> <time> <pri>: submit a job named <job>,

>help -fcfs
Scheduling policy:
fcfs: change the scheduling policy to FCFS.

>
```

6.2.2 Run Command

The run command will dispatch jobs to the ready_queue once the queue starts to populate the dispatcher will begin to dispatch jobs to the batch_job program.
The run command will error if the number of options are not met. The run command will accept batch_job or job as the external program to run.

```
>run job 10 1
Dispatcher only supports the ./batch_job program
job replaced with ./batch_job program

>list
Total number of jobs in the queue: 0
Scheduling Policy: FCFS.
Running Job:
Name  CPU_Time  Pri  Progress
job         10              1          run

>run job 20 1
Dispatcher only supports the ./batch_job program
job replaced with ./batch_job program

>list
Total number of jobs in the queue: 0
Scheduling Policy: FCFS.
Running Job:
Name  CPU_Time  Pri  Progress
job         20              1          run

Completed Jobs:
Name  CPU_Time  Pri  Arrival_time  Progress
job                   10          1         20:20:49     complete

>run batch_job 3 1

>run batch_job 4 1 1
Error: Number of Run 5 of 4 parameters entered.
run <job> <time> <pri>: submit a job named <job>,

>run batch_job 3
Error: Number of Run 3 of 4 parameters entered.
run <job> <time> <pri>: submit a job named <job>,
```

6.2.3 List Command

The list command will display the current Ready Queue also it will display running jobs and completed jobs sent and run by the dispatcher. The time displayed on the arrival column is the local-time based on your system arrival.

```
>run job 1 2
```

```
Dispatcher only supports the ./batch_job program
job replaced with ./batch_job program

>run batch_job 5 1

>run batch_job 2 2

>list
Total number of jobs in the queue: 0
Scheduling Policy: FCFS.
Completed Jobs:
Name  CPU_Time  Pri  Arrival_time  Progress
job                   1           2         20:29:5     complete
batch_job                    5           1         20:29:15     complete
batch_job                    2           2         20:29:25     complete

>run job 10 1
Dispatcher only supports the ./batch_job program
job replaced with ./batch_job program

>list
Total number of jobs in the queue: 0
Scheduling Policy: FCFS.
Running Job:
Name  CPU_Time  Pri  Progress
job         10              1          run

Completed Jobs:
Name  CPU_Time  Pri  Arrival_time  Progress
job                   1           2          20:29:5                      complete
batch_job                    5           1         20:29:15                    complete
batch_job                    2           2         20:29:25                     complete

>run job 10 1
Dispatcher only supports the ./batch_job program
job replaced with ./batch_job program

>run job 11 1
Dispatcher only supports the ./batch_job program
job replaced with ./batch_job program

>run job 32 1
Dispatcher only supports the ./batch_job program
job replaced with ./batch_job program

>list
Total number of jobs in the queue: 2
Scheduling Policy: FCFS.
Running Job:
Name  CPU_Time  Pri  Progress
job          10              1          run

Ready Queue:
Name  CPU_Time  Pri  Arrival_time  Progress
job                   11            1         20:32:17               wait
Ready Queue:
Name  CPU_Time  Pri  Arrival_time  Progress
job                      32          1         20:32:21               wait
```

```
>
```

6.2.4 Quit Command

The quit command exits the program and displayed the statistics of the scheduled jobs.

```
>run batch_job 5 1

>run batch_job 3 1

>run batch_job 2 1

>list
Total number of jobs in the queue: 0
Scheduling Policy: FCFS.
Completed Jobs:
Name  CPU_Time  Pri  Arrival_time  Progress
batch_job                   5          1         20:35:3     complete
batch_job                   3          1         20:35:7     complete
batch_job                   2          1         20:35:13     complete

>quit
Total number of job submitted: 3
Total number of job completed: 3
Average turnaround time: 3.666667 seconds
Average CPU time:3.333333 seconds
Average waiting time:0.333333 seconds
Throughput: 0.818182 No./second
```

6.2.5 FCFS Command

The fcfs command sets the schedule type to the first come first algorithm. This will also reorder the ready queue if jobs are still present in the queue.

```
>fcfs
>run batch_job 5 1

>run batch_job 2 1

>run batch_job 10 1

>list
Total number of jobs in the queue: 0
Scheduling Policy: FCFS.
Running Job:
Name  CPU_Time  Pri  Progress
batch_job         10              1          run

Completed Jobs:
Name  CPU_Time  Pri  Arrival_time  Progress
batch_job                   5          1         20:38:51     complete
batch_job                   2          1         20:38:56     complete
```

6.2.6 SJF Command

The sjf command sets the schedule type to the shortest job first algorithm. This will also reorder the ready queue if jobs are still present in the queue.

```
>sjf

>run batch_job 20 1

>run batch_job 10 2

>run batch_job 5 2

>list
Total number of jobs in the queue: 2
Scheduling Policy: SJF.
Running Job:
Name  CPU_Time  Pri  Progress
batch_job         20              1          run

Ready Queue:
Name  CPU_Time  Pri  Arrival_time  Progress
batch_job                    5          2         20:41:25               wait
Ready Queue:
Name  CPU_Time  Pri  Arrival_time  Progress
batch_job                   10          2         20:41:17               wait

>list
Total number of jobs in the queue: 1
Scheduling Policy: SJF.
Running Job:
Name  CPU_Time  Pri  Progress
batch_job          5              2          run

Ready Queue:
Name  CPU_Time  Pri  Arrival_time  Progress
batch_job                   10          2         20:41:17               wait
Completed Jobs:
Name  CPU_Time  Pri  Arrival_time  Progress
batch_job                   20          1         20:41:11     complete

>list
Total number of jobs in the queue: 0
Scheduling Policy: SJF.
Running Job:
Name  CPU_Time  Pri  Progress
batch_job         10              2          run

Completed Jobs:
Name  CPU_Time  Pri  Arrival_time  Progress
batch_job                   20          1         20:41:11     complete
batch_job                    5          2         20:41:25     complete

>
```

6.2.7 PRI Command

The pri command sets the schedule type to the shortest job first algorithm. This will also reorder the ready queue if jobs are still present in the queue.

```
>pri

>run batch_job 20 20
```

```
>run batch_job 20 2

>run batch_job 20 1

>list
Total number of jobs in the queue: 2
Scheduling Policy: PRI.
Running Job:
Name  CPU_Time  Pri  Progress
batch_job         20             20          run

Ready Queue:
Name  CPU_Time  Pri  Arrival_time  Progress
batch_job                  20          1         20:44:2               wait
Ready Queue:
Name  CPU_Time  Pri  Arrival_time  Progress
batch_job                  20          2         20:43:52               wait

>list
Total number of jobs in the queue: 1
Scheduling Policy: PRI.
Running Job:
Name  CPU_Time  Pri  Progress
batch_job         20              1          run

Ready Queue:
Name  CPU_Time  Pri  Arrival_time  Progress
batch_job                  20          2         20:43:52               wait
Completed Jobs:
Name  CPU_Time  Pri  Arrival_time  Progress
batch_job                  20         20         20:43:46     complete
```

6.2.8 Quit Command

The quit command exits the program and displays the statistics of the schedule algorithm.

```
>run batch_job 3 1

>run batch_job 2 1

>run batch_job 10 1

>list
Total number of jobs in the queue: 0
Scheduling Policy: FCFS.
Running Job:
Name  CPU_Time  Pri  Progress
batch_job         10              1          run

Completed Jobs:
Name  CPU_Time  Pri  Arrival_time  Progress
batch_job                   3          1         21:18:19     complete
batch_job                   2          1         21:18:24     complete

>list
Total number of jobs in the queue: 0
Scheduling Policy: FCFS.
```

```
Completed Jobs:
Name  CPU_Time  Pri  Arrival_time  Progress
batch_job                   3          1         21:18:19     complete
batch_job                   2          1         21:18:24     complete
batch_job                  10          1         21:18:28     complete

>quit
Total number of job submitted: 3
Total number of job completed: 3
Average turnaround time: 5.000000 seconds
Average CPU time:5.000000 seconds
Average waiting time:0.000000 seconds
Throughput: 0.600000 No./second
```

6.2.9 Test Command

The test command will test our scheduler and dispatcher and with random numbers to verify our algorithms. The test command will also output the results to the screen and to a output file.

```
>test mytest fcfs 5 3 2 1
Warning: Test Command.
Previous Statistics and Queue will be cleared.
Only list and Help command will be allowed to be entered.

>
Benchmark Test Currently Running
Limited Commands Allowed
list , help , quit

>Benchmark Test Done Running
Please Press Enter for Statistics


>Benchmark Test Done Running

Total number of job submitted: 6
Total number of job completed: 6
Average turnaround time: 7.000000 seconds
Average CPU time:2.000000 seconds
Average waiting time:5.000000 seconds
Throughput: 0.857143 No./second
```

7 Testing

We have seen that the commands and the AUbatch program works. However, we still need to verify and test the system with using the test command we are able to test the dispatcher and scheduler.

7.1 test mytest fcfs 3 2 1 5

```
Warning: Test Command.
Previous Statistics and Queue will be cleared.
Only list and Help command will be allowed to be entered.

>
Benchmark Test Currently Running
Limited Commands Allowed
list , help , quit

>list
Total number of jobs in the queue: 2
Scheduling Policy: FCFS.
Running Job:
Name  CPU_Time  Pri  Progress
batch_job          1              1          run

Ready Queue:
Name  CPU_Time  Pri  Arrival_time  Progress
batch_job                   3          2         21:20:26               wait
Ready Queue:
Name  CPU_Time  Pri  Arrival_time  Progress
batch_job                   2          1         21:20:26               wait
Completed Jobs:
Name  CPU_Time  Pri  Arrival_time  Progress
batch_job                   1          2         21:20:26     complete

>
Benchmark Test Currently Running
Limited Commands Allowed
list , help , quit

>list
Total number of jobs in the queue: 1
Scheduling Policy: FCFS.
Running Job:
Name  CPU_Time  Pri  Progress
batch_job          3              2          run

Ready Queue:
Name  CPU_Time  Pri  Arrival_time  Progress
batch_job                   2          1         21:20:26               wait
Completed Jobs:
Name  CPU_Time  Pri  Arrival_time  Progress
batch_job                   1          2         21:20:26     complete
batch_job                   1          1         21:20:26     complete

>
```

```
Benchmark Test Currently Running
Limited Commands Allowed
list , help , quit

>list
Total number of jobs in the queue: 1
Scheduling Policy: FCFS.
Running Job:
Name  CPU_Time  Pri  Progress
batch_job          3              2          run

Ready Queue:
Name  CPU_Time  Pri  Arrival_time  Progress
batch_job                    2          1         21:20:26               wait
Completed Jobs:
Name  CPU_Time  Pri  Arrival_time  Progress
batch_job                    1          2         21:20:26     complete
batch_job                    1          1         21:20:26     complete

>
Benchmark Test Currently Running
Limited Commands Allowed
list , help , quit

>Benchmark Test Done Running
Please Press Enter for Statistics


>Benchmark Test Done Running

Total number of job submitted: 4
Total number of job completed: 4
Average turnaround time: 3.750000 seconds
Average CPU time:1.750000 seconds
Average waiting time:2.000000 seconds
Throughput: 1.066667 No./second

>quit
Total number of job submitted: 4
Total number of job completed: 4
Average turnaround time: 3.750000 seconds
Average CPU time:1.750000 seconds
Average waiting time:2.000000 seconds
Throughput: 1.066667 No./second
[tbh0020@CentOS7 AuBatch]$
[tbh0020@CentOS7 AuBatch]$
[tbh0020@CentOS7 AuBatch]$
[tbh0020@CentOS7 AuBatch]$ ./aubatch
Welcome to Thomas Heckwolfs's batch job scheduler Version 1.0
 Type ?help? to find more about AUbatch commands.
```

7.2 test mytest pri 3 2 1 5

```
>test mytest pri 3 2 1 5
Warning: Test Command.
```

```
Previous Statistics and Queue will be cleared.
Only list and Help command will be allowed to be entered.

>
Benchmark Test Currently Running
Limited Commands Allowed
list , help , quit

>list
Total number of jobs in the queue: 2
Scheduling Policy: PRI.
Running Job:
Name  CPU_Time  Pri  Progress
batch_job          3              2          run

Ready Queue:
Name  CPU_Time  Pri  Arrival_time  Progress
batch_job                   1          2         21:21:45               wait
Ready Queue:
Name  CPU_Time  Pri  Arrival_time  Progress
batch_job                   4          2         21:21:45               wait
Completed Jobs:
Name  CPU_Time  Pri  Arrival_time  Progress
batch_job                   1          1         21:21:45     complete

>
Benchmark Test Currently Running
Limited Commands Allowed
list , help , quit

>Benchmark Test Done Running
Please Press Enter for Statistics


>Benchmark Test Done Running

Total number of job submitted: 4
Total number of job completed: 4
Average turnaround time: 4.750000 seconds
Average CPU time:2.250000 seconds
Average waiting time:2.500000 seconds
Throughput: 0.842105 No./second

>list
Total number of jobs in the queue: 0
Scheduling Policy: PRI.
Completed Jobs:
Name  CPU_Time  Pri  Arrival_time  Progress
batch_job                   1          1         21:21:45     complete
batch_job                   3          2         21:21:45     complete
batch_job                   1          2         21:21:45     complete
batch_job                   4          2         21:21:45     complete
```

7.3 test mytest sjf 3 2 1 5

```
>
Warning: Test Command.
```

```
Previous Statistics and Queue will be cleared.
Only list and Help command will be allowed to be entered.

>
Benchmark Test Currently Running
Limited Commands Allowed
list , help , quit

>list
Total number of jobs in the queue: 2
Scheduling Policy: SJF.
Running Job:
Name  CPU_Time  Pri  Progress
batch_job          3              1          run

Ready Queue:
Name  CPU_Time  Pri  Arrival_time  Progress
batch_job                    4          2         21:22:48               wait
Ready Queue:
Name  CPU_Time  Pri  Arrival_time  Progress
batch_job                    4          2         21:22:48               wait
Completed Jobs:
Name  CPU_Time  Pri  Arrival_time  Progress
batch_job                    2          2         21:22:48     complete

>
Benchmark Test Currently Running
Limited Commands Allowed
list , help , quit

>Benchmark Test Done Running
Please Press Enter for Statistics


>Benchmark Test Done Running

Total number of job submitted: 4
Total number of job completed: 4
Average turnaround time: 7.250000 seconds
Average CPU time:3.250000 seconds
Average waiting time:4.000000 seconds
Throughput: 0.551724 No./second
```

8 Conclusion

8.1 Conclusion Program

The design and implementation of your AUbatch.

- Performance metrics and workload conditions.
- The performance evaluation of the three scheduling algorithms.
- Lessons learned

8.1.1 Performance metrics and workload conditions

We see with basic test the results of the test run. Also note we save the runs to a file with the benchmark name given. We see with the SJF we had a overall better performance of smaller burst jobs. While priority came in second and first come first serve came in last.
Note: All test results are stored in filenames of the test job. Note: The below test files now show the scheduler type when run in the generated file. Note: This is with small cpu burst and only 5 jobs. The second test we gave more jobs to see if the results algorithms stayed the same.

```
>test mytest fcfs 5 3 1 4
Total number of job submitted: 5
Total number of job completed: 5
Average turnaround time: 6.400000 seconds
Average CPU time:2.400000 seconds
Average waiting time:4.000000 seconds
Throughput: 0.781250 No./second

>test mytest pri 5 3 1 4
Total number of job submitted: 5
Total number of job completed: 5
Average turnaround time: 5.200000 seconds
Average CPU time:1.600000 seconds
Average waiting time:3.600000 seconds
Throughput: 0.961538 No./second

>test mytest sjf 5 3 1 4
Total number of job submitted: 5
Total number of job completed: 5
Average turnaround time: 5.000000 seconds
Average CPU time:1.800000 seconds
Average waiting time:3.200000 seconds
Throughput: 1.000000 No./second

test mytest2 sjf 10 8 10 30
Total number of job submitted: 10
Total number of job completed: 10
Average turnaround time: 73.500000 seconds
Average CPU time:15.100000 seconds
Average waiting time:58.400002 seconds
Throughput: 0.136054 No./second

test mytest2 pri 10 8 10 30
Total number of job submitted: 10
```

```
Total number of job completed: 10
Average turnaround time: 91.400002 seconds
Average CPU time:17.900000 seconds
Average waiting time:73.500000 seconds
Throughput: 0.109409 No./second

test mytest2 fcfs 10 8 10 30
Total number of job submitted: 10
Total number of job completed: 10
Average turnaround time: 101.199997 seconds
Average CPU time:18.799999 seconds
Average waiting time:82.400002 seconds
Throughput: 0.098814 No./second

>test mytest3 fcfs  100 3 1 4
Total number of job submitted: 100
Total number of job completed: 100
Average turnaround time: 98.430000 seconds
Average CPU time:2.070000 seconds
Average waiting time:96.349998 seconds
Throughput: 1.015950 No./second

>test mytest3 pri  100 3 1 4
Total number of job submitted: 100
Total number of job completed: 100
Average turnaround time: 82.379997 seconds
Average CPU time:2.070000 seconds
Average waiting time:80.309998 seconds
Throughput: 1.213887 No./second

>test mytest3 sjf  100 3 1 4
Total number of job submitted: 100
Total number of job completed: 100
Average turnaround time: 98.839996 seconds
Average CPU time:1.950000 seconds
Average waiting time:96.889999 seconds
Throughput: 1.011736 No./second
```

8.1.2 Performance evaluation of the three scheduling algorithms

We see using the basic test operation, that in the results from the test runs, we have not been able to find a silver bullet. Due to the nature of the arrival time, we found that we can not enter jobs manually due to the fact that we introduce a delay in input which causes wait times to improperly calculate. Thus we have to use a test feature to inject jobs all at once which also is not a real world example. Though given the current abilities of this program, we have come to the conclusion that a simple, single algorithm is not the best approach when scheduling jobs. The type of jobs scheduled will determine the scheduling algorithm that is best suited for the system.

8.1.3 Lessons Learned

- How to use Pthreads to create multi-threaded programs utilizing arrays and linked list.
- When manually entering commands, we do not get accurate wait times due to the fact that lower burst jobs may be finished when the next job is entered.
- How to use excve to spawn child processes to allow for execution of separate programs.
- How to evaluate different scheduling algorithms. We have seen that depending on the variation of the given parameters, the best algorithm greatly depends on the type of jobs that you are receiving.