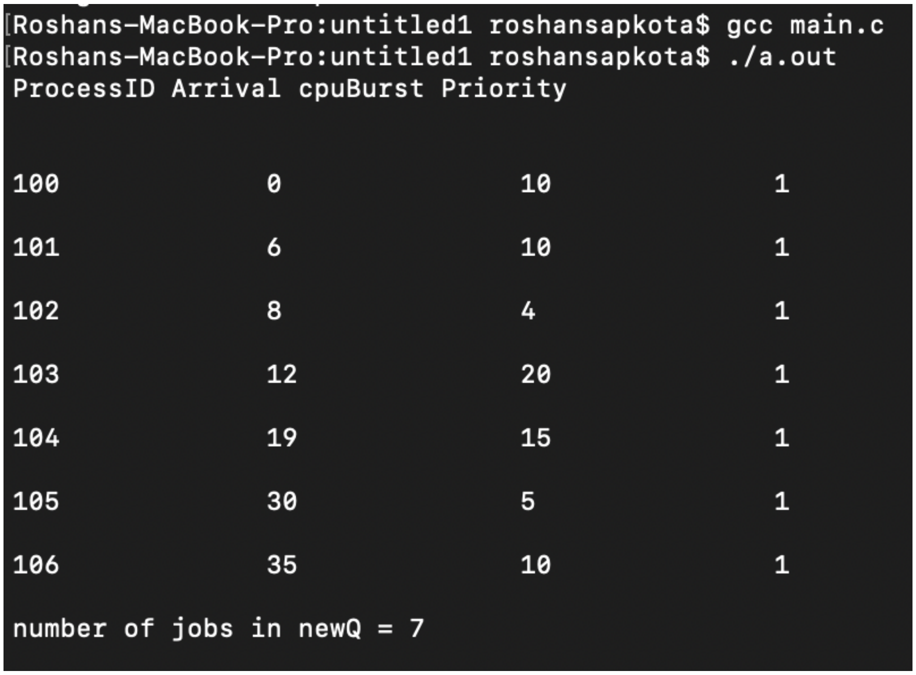
CMPS-431 Program 2- Job Scheduler

For this program, we made a simulation of job schedular for an OS. Major three scheduling algorithm are implemented in this program namely “First Come First Serve”, “Shortest Job First” and “Round Robin”. All of these programs read in a list of jobs from an input.txt file and output the completion time, response time, throughput and turnaround time.

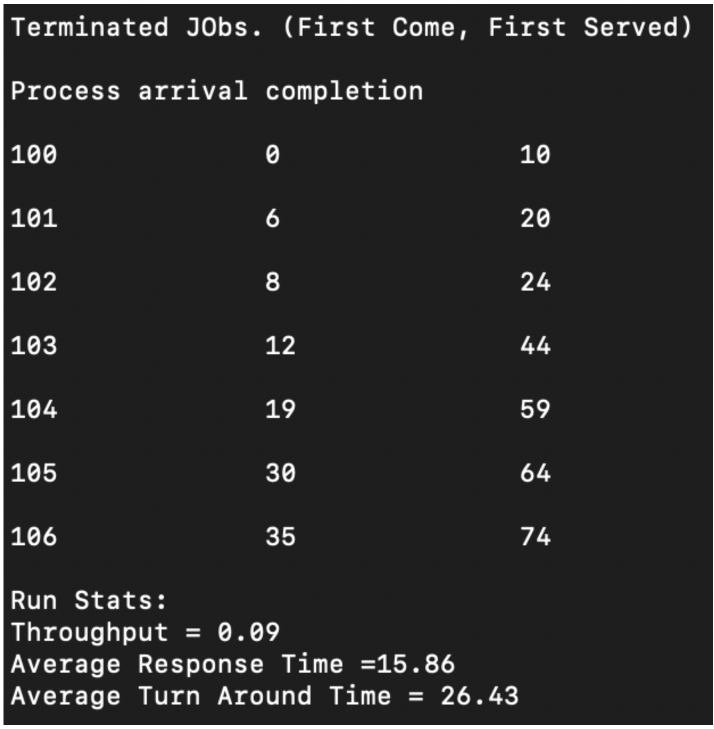
Below are the screenshots of the output of the program.

Here, program reads the input file and show the total number of jobs available in the file.



```
[Roshans-MacBook-Pro:untitled1 roshansapkota$ gcc main.c
[Roshans-MacBook-Pro:untitled1 roshansapkota$ ./a.out
ProcessID Arrival cpuBurst Priority


100            0             10            1

101            6             10            1

102            8             4             1

103            12            20            1

104            19            15            1

105            30            5             1

106            35            10            1

number of jobs in newQ = 7
```

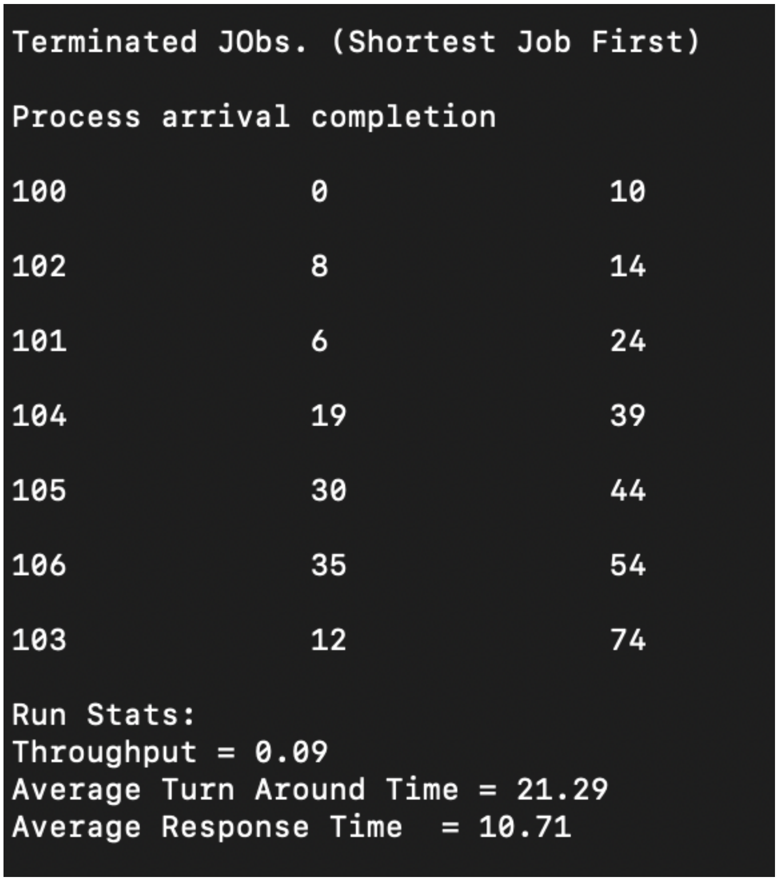
Below is the output of First Come First Serve algorithm. It calculates the completion time of the jobs based on their arrival time. The job with lowest arrival time is executed first. Then, it calculates the different stats required.



```
Terminated JObs. (First Come, First Served)

Process arrival completion

100             0               10

101             6               20

102             8               24

103             12              44

104             19              59

105             30              64

106             35              74

Run Stats:
Throughput = 0.09
Average Response Time =15.86
Average Turn Around Time = 26.43
```

.

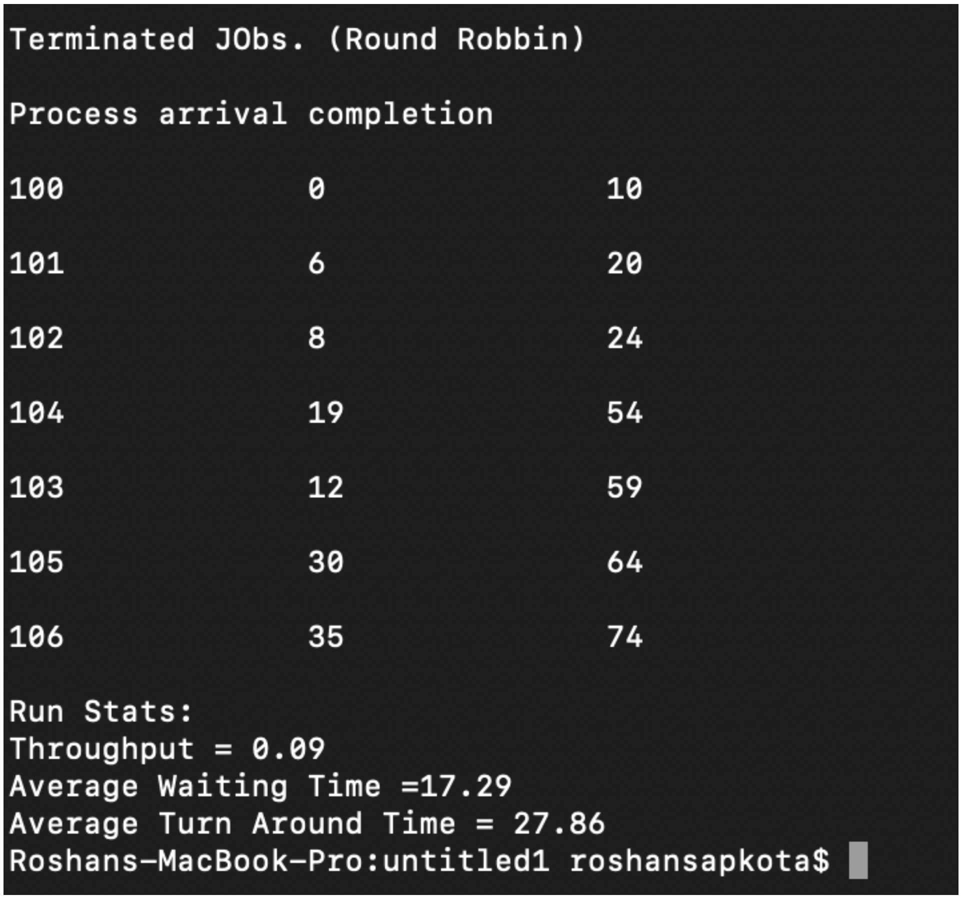
Another algorithm is Shortest Job First. It first sorts the table according to the lowest arrival. Then, it looks at the burst time and sort again according to its CPU Burst time. Then finally it calculates the completion time and all the required stats.



```
Terminated JObs. (Shortest Job First)

Process arrival completion

100             0               10

102             8               14

101             6               24

104             19              39

105             30              44

106             35              54

103             12              74

Run Stats:
Throughput = 0.09
Average Turn Around Time = 21.29
Average Response Time  = 10.71
```

.

Our last algorithm is Round Robin. For the less quantum time, it behaves exactly as “FCFS”. Here, quanta is 15. Our algorithm checks if the burst time is greater than quanta or not. If so, it subtracts the quanta from burst time and updates the burst time as well. It also then computes the completion time and all the stats required as below:



```
Terminated JObs. (Round Robbin)

Process arrival completion

100             0               10

101             6               20

102             8               24

104             19              54

103             12              59

105             30              64

106             35              74

Run Stats:
Throughput = 0.09
Average Waiting Time =17.29
Average Turn Around Time = 27.86
Roshans-MacBook-Pro:untitled1 roshansapkota$ 
```

CODE:

1: loadJObs():

```
int loadJobs(char *filename)
{
    FILE *jobs;
    char string[80];
    int pId, arrival, cpuBurst, priority;
    int pid[30], at[30], pt[30], bt[30];
    int j, completion;
    static int nJobs;
/* Open file of jobs to be put in the ready que. */
    jobs = fopen(filename, "r");
/* Load the ready que from the file. */
    fgets(string, 80, jobs);
    printf("%s \n", string);
    j= 0;
    while(fscanf(jobs, "%d %d %d %d", &pId, &arrival, &cpuBurst,
                 &priority) != EOF) {
        at[j] = arrival;
        pid[j] = pId;
        bt[j] = cpuBurst;
        pt[j] = priority;
        printf("\n%d\t\t%d\t\t%d\t\t%d\n", pId, arrival, cpuBurst, priority);
        j = j+1;
    }
    nJobs = j;
    printf("\n");
    printf("number of jobs in newQ = %d \n", nJobs);
    fclose(jobs);
    return nJobs;
}
```

2. ShorestsJobFirst();

```
void ShortestJobFirst() {
    FILE *jobs;
    char string[80];
    int pId, arrival, cpuBurst, priority, i, j, t, nJobs, pid[30], pt[30],
bt[30], at[30], wt[30], tat[30], comp[30];
    float  awt = 0, atat = 0, resTime = 0, tpt = 0;
    int countValue[7], responseTime[30];
    jobs = fopen("/Users/roshansapkota/CLionProjects/untitled1/input.txt",
"r");
    fgets(string, 80, jobs);

    j = 0;
    while (fscanf(jobs, "%d %d %d %d", &pId, &arrival, &cpuBurst, &priority)
!= EOF) {
        at[j] = arrival;
        pid[j] = pId;
        bt[j] = cpuBurst;
        pt[j] = priority;
        j += 1;
    }

    nJobs = j;
    for (i = 0; i < nJobs; i++) {
        for (j = i; j < nJobs; j++) {
            if (at[i] > at[j]) {
                t = at[i];
                at[i] = at[j];
                at[j] = t;

                t = bt[i];
                bt[i] = bt[j];
                bt[j] = t;

                t = pid[j];
                pid[j] = pid[i];
                pid[i] = t;
            }
        }
    }

    comp[0] = at[0] + bt[0];
    int numCount = 0;
    for (i = 0; i < nJobs; i++) {
```

```c
        int count = 0;
        for (j = i+1; j < nJobs; j++) {
            if (comp[i] > at[j]) {
                countValue[count] = j;
                count++;
            }
        }
        for (int a = 0; a < nJobs; a++) {
            int resVal = countValue[a];
            int resValSec = countValue[a+1];
//             wt[i] = wt[i] + bt[a];
            if (resVal != 0 && resValSec != 0){
                if (bt[resVal] > bt[resValSec]) {
                    t = at[resVal];
                    at[resVal] = at[resValSec];
                    at[resValSec] = t;

                    t = bt[resVal];
                    bt[resVal] = bt[resValSec];
                    bt[resValSec] = t;

                    t = pid[resValSec];
                    pid[resValSec] = pid[resVal];
                    pid[resVal] = t;
                }
                comp[numCount +1] = comp[numCount] + bt[numCount+1];
                ++numCount;
            }
        }
        tat[i] = comp[i] - at[i];
        atat = atat + tat[i];
        wt[i] = tat[i] - bt[i];
        awt = awt + wt[i];
        resTime = awt;

    }
    printf("\nTerminated JObs. (Shortest Job First)\n");
    printf("\nProcess\tarrival\tcompletion\n");
    for (i = 0; i < nJobs; i++) {
        printf("\n%d\t\t%d\t\t%d\n", pid[i], at[i], comp[i]);
    }

    resTime = resTime / nJobs;
    atat = atat / nJobs;
//        tpt = tpt/nJobs;
    printf("\nRun Stats:\n");
    printf("Throughput = 0.09\n");
    printf("Average Turn Around Time = %.2f\n", atat);
    printf("Average Response Time  = %.2f\n", resTime);
}
```

3. FirstComeFirstServe():

```c
void FirstComeFirstServe(){
    FILE *jobs;
    char string[80];
    int  pId, arrival, cpuBurst, priority, i, j, nJobs, pid[30], pt[30],
bt[30], at[30], wt[30], tat[30], temp[30], comp[30];
    float awt = 0, atat = 0, resTime = 0, tpt = 0;

    jobs = fopen("/Users/roshansapkota/CLionProjects/untitled/input.txt",
"r");
    fgets(string, 80, jobs);

    j = 0;
    while(fscanf(jobs, "%d %d %d %d", &pId, &arrival, &cpuBurst,
                 &priority) != EOF) {
        at[j] = arrival;
        pid[j] = pId;
        bt[j] = cpuBurst;
        pt[j] = priority;
        j += 1;
    }
    nJobs = j;
    printf("\nTerminated JObs. (First Come, First Served)\n");
    printf("\nProcess\tarrival\tcompletion\n");
    temp[0] = 0;
    for(i = 0; i< nJobs; i++){
        wt[i] = 0;
        tat[i] = 0;
        temp[i+1] = temp[i] + bt[i];
        wt[i] = temp[i] - at[i];
        tat[i] = wt[i] + bt[i];
        awt = awt + wt[i];
        atat = atat + tat[i];
        comp[i] =tat[i] + at[i];
        resTime = awt;
        printf("\n%d\t\t%d\t\t%d\n", pid[i], at[i], comp[i]);
    }
//    awt = awt / nJobs;
    atat = atat / nJobs;
    resTime = resTime/nJobs;
    printf("\nRun Stats:");
    printf("\nThroughput = 0.09\n");
    printf("Average Response Time =%.2f\n", resTime);
    printf("Average Turn Around Time = %.2f\n", atat);
}
```

3. RoundRobbin():

```
void roundRobbin(int qt){
    FILE *jobs;
    char string[80];
    int pId, arrival, cpuBurst, priority, bt[30], wt[30], tat[30], rem_bt[30]
, pid[30], at[30], pt[30], comp[30];
    float  awt = 0, atat = 0, tpt =0 ;
    int i, j, k, resTime = 0;
    int t, completionTime = 0;
    int count;
    int nJobs;
    jobs = fopen("/Users/roshansapkota/CLionProjects/untitled/input.txt",
"r");
    fgets(string, 80, jobs);
    j = 0;
    while(fscanf(jobs, "%d %d %d %d", &pId, &arrival, &cpuBurst,
                 &priority) != EOF) {
        at[j] = arrival;
        pid[j] = pId;
        bt[j] = cpuBurst;
        pt[j] = priority;
        rem_bt[j] = bt[j];
        j += 1;
    }
    nJobs = j;
    comp[0] = at[0] + bt[0];
    int remainingValue = 0;
    for (k = 0, count = 0; k < nJobs; k++) {
        if (bt[k] > qt) {
            int intialRemainingValue = rem_bt[k];
            remainingValue = rem_bt[k] - qt;
            rem_bt[k] = rem_bt[k + 1];
            rem_bt[k + 1] = remainingValue;
            for (int i = k + 1; i < nJobs; i++) {
                if (bt[k] > at[i]) {
                    t = pid[k];
                    pid[k] = pid[k + 1];
                    pid[k + 1] = t;

                    t = at[k];
                    at[k] = at[k + 1];
                    at[k + 1] = t;

                    rem_bt[k] = rem_bt[k] + (intialRemainingValue -
remainingValue);
                }
            }
        }
        if (nJobs == count)
            break;

    }
    for (int i = 1; i < nJobs; i++ ) {
        comp[i] = comp[i-1] + rem_bt[i];
    }
    printf("\nTerminated JObs. (Round Robbin)\n");
    printf("\nProcess\tarrival\tcompletion\n");
    for (int i = 0; i < nJobs; i++) {
        tat[i] = comp[i]-at[i];
        wt[i]= tat[i] - bt[i];
        awt = awt + wt[i];
        atat = atat + tat[i];
        //completionTime = completionTime + comp[i];
        printf("\n%d\t\t%d\t\t%d\n", pid[i], at[i], comp[i]);

    }
    awt = awt / nJobs;
    atat = atat / nJobs;
//    tpt = tpt/nJobs;
//    averageResponseTime = resTime/nJobs;
    printf("\nRun Stats:\n");
    printf("Throughput = 0.09\n");
    printf("Average Waiting Time =%.2f\n", awt);
    printf("Average Turn Around Time = %.2f\n", atat);
//    getchar();
}
```

4. main():

```
int main() {
    loadJobs("/Users/roshansapkota/CLionProjects/untitled1/input.txt");
    FirstComeFirstServe();
    ShortestJobFirst();
    roundRobbin(15);
    return 0;
}
```