

# Habib University Operating Systems - CS232

## Assignment 02 - Report Simulating A Scheduler



**Instructor:** Munzir Zafar

Ali Muhammad Asad - aa07190

## Contents

<b>1</b>	<b>Introduction</b>	<b>3</b>
<b>2</b>	<b>Makefile</b>	<b>3</b>
<b>3</b>	<b>Input</b>	<b>3</b>
<b>4</b>	<b>Output</b>	<b>4</b>
<b>5</b>	<b>Scheduling Algorithms</b>	<b>5</b>
5.1	FIFO - First In First Out . . . . .	5
5.2	SJF - Shortest Job First . . . . .	7
5.3	STCF - Shortest Time to Completion First . . . . .	8
5.4	RR - Round Robin . . . . .	10
<b>6</b>	<b>Performance Metrics of Scheduling Algorithms</b>	<b>12</b>
6.1	Testing Performance Metrics . . . . .	12
6.1.1	Test Case 0 / Test Case 2 (same test case) . . . . .	12
6.1.2	Test Case 5 . . . . .	13
6.1.3	Test Case 10 . . . . .	13
6.1.4	Test Case 13 . . . . .	14
6.2	Results - Comparison of Performance Metrics . . . . .	14
<b>7</b>	<b>Takeaway and Reflections</b>	<b>15</b>
<b>8</b>	<b>References</b>	<b>16</b>
<b>A</b>	<b>Appendix</b>	<b>17</b>
A.1	Outputs for Test Cases . . . . .	17
A.1.1	Test Case 0 / Test Case 2 . . . . .	17
A.1.2	Test Case 5 . . . . .	19
A.1.3	Test Case 10 . . . . .	22
A.1.4	Test Case 13 . . . . .	27
A.2	Program Code . . . . .	34

## 1 Introduction

The assignment is to write a simulator for a process scheduler using C language. Processes will be given to the program on the command line terminal, along with the scheduling algorithm to use, then the program will select the given processing algorithm to manage the processes with. A queue data structure has been made to store the processes based on time of arrival, and this queue will be passed as an argument to the scheduler along with the system time (by default it is always 0 thus a scheduler always starts from 0 system time).

The scheduling algorithms are FIFO (First-In-First-Out), SJF (Shortest Job First), STCF (Shortest Time to Completion First), and RR (Round Robin).

## 2 Makefile

The accompanying makefile has also been provided with the program, and supports the following commands:

- **make build**: builds the program and creates an executable file **scheduler**
- **make run**: runs the program, input has to be provided on the command line (input format is discussed in the next section)
- **make clean**: removes the executable file **scheduler**

## 3 Input

The input is given in the command line / terminal. The first line will be the total number  $N$  of processes (**int**), followed by the scheduling policy (**string**) in the second line which could be one of FIFO, SJF, STCF, RR. Then  $N$  lines of input will follow, each containing the following data separated by colon (:); Process Name **pname**, Process ID **pid**, Process total runtime **duration**, and Process Arrival Time **arrivaltime**. **pname** will be a string of upto 10 chars, all other fields will be integers.

A Sample Input is as follows:

```
3
RR
P1:12:7:3
P2:15:3:5
P3:1:6:2
end
```

The last line **end** is just there to show there are no more processes, you can replace it with any other string and the program will still work. But you do need to provide something after you have given the last process else it won't move forward (courtesy of hackerrank and given code that I won't change).

## 4 Output

The program simulates the scheduler at every step, showing the state of the system in the following format:

**time: running name: ready queue names comma separated:**

- **time** represents the clock-ticks passed since the system started (assuming a clock-tick lasts 1 millisecond). System starts at 0 ticks, and a clock tick between 0 and 1 ms is considered as 1 tick.
- **running name** represents the name of the process currently running on the CPU, if no process is running then it will be **idle**.
- **ready queue names** represents the names of the processes in the ready queue, if the ready queue is empty then it will be **empty**.

Sample Output for the given sample input:

```
1:idle:empty:
2:idle:empty:
3:P3:empty:
4:P3:P1(7),:
5:P1:P3(4),:
6:P3:P1(6),P2(3),:
7:P1:P2(3),P3(3),:
8:P2:P3(3),P1(5),:
9:P3:P1(5),P2(2),:
10:P1:P2(2),P3(2),:
11:P2:P3(2),P1(4),:
12:P3:P1(4),P2(1),:
13:P1:P2(1),P3(1),:
14:P2:P3(1),P1(3),:
15:P3:P1(3),:
16:P1:empty:
17:P1:empty:
18:P1:empty:
```

## 5 Scheduling Algorithms

Before running into the scheduling algorithms, a change was made to the Process Control Block (PCB) struct, in order to efficiently, and effectively calculate the response time. The PCB struct now contains a boolean variable `isfirsttime` which is set to true by default, and is set to false when the process is run for the first time. This way, the response time is calculated only once, and not every time the process is run. The PCB struct now looks like this:

```

1 struct pcb{                                // stores info on a process
2     unsigned int pid;                      // pid
3     char pname[20];                       // pname
4     unsigned int ptimeleft;               // time left to complete
5     unsigned int ptimearrival;           // time of arrival
6     bool isfirsttime;                    // flag for first run time check -> true
7     unsigned int pfirstruntime;          // time when process ran for the first time
8 }; typedef struct pcb pcb;
```

Listing 1: Struct PCB - Process Control Block

### 5.1 FIFO - First In First Out

```

1 void sched_FIFO(dlq *const p_fq, int *p_time){
2     // initialize a queue to manage processes that are ready to run
3     dlq queue; queue.head = queue.tail = NULL;
4     // initialize the first process node from the head of the queue
5     dlq_node *process = remove_from_head(p_fq);
6
7     // initialize performance metrics
8     int num_processes = 1;
9     int rtl = 0, response_time = 0;
10    int turnaround_time = 0;
11    int first_arrival = process->data->ptimearrival;
12
13    while(1){
14        (*p_time)++; // increment the system time
15        // if no processes left to run, and both queues empty, break the loop
16        if(!(process->data->ptimeleft) && is_empty(&queue) && is_empty(p_fq))
17            break;
18
19        // if processes are left and a process has arrived, add it to the
20        // queue and increment the number of processes
21        if(!is_empty(p_fq) && p_fq->head->data->ptimearrival < (*p_time)){
22            add_to_tail(&queue, remove_from_head(p_fq));
23            num_processes++;
24        }
25
26        if(process->data->isfirsttime == true){ // if the process has not run
27            // before, set its first runtime to the system time and set isfirsttime
28            // to false
29            process->data->isfirsttime = false;
30            process->data->pfirstruntime = (*p_time);
31            rtl = (*p_time) - process->data->ptimearrival;
```

```

29     if (rt1 < 0) rt1 = 1;
30     response_time += rt1;
31 }
32
33 printf("%d:", (*p_time));
34 // if the process still has to arrive, print idle, else decrement its
   time and print its name
35 if(process->data->ptimearrival >= (*p_time))
36     printf("idle:");
37 else{
38     process->data->ptimeleft--;
39     printf("%s:", process->data->pname);
40 }
41 // if the queue is empty, print empty, else show contents of the
   queue
42 if(is_empty(&queue)) printf("empty:\n");
43 else{
44     print_q(&queue);
45     printf(":\n");
46 }
47
48 // if process has finished, calculate the turnaround time and add it
   to the total turnaround time, then remove another process from the
   queue if queue is not empty
49 if(process->data->ptimeleft == 0){
50     turnaround_time += (*p_time) - process->data->ptimearrival;
51     if (!is_empty(&queue))
52         process = remove_from_head(&queue);
53 }
54 } free(process);
55 float throughput = (float)num_processes / ((*p_time) - first_arrival);
56 float avg_turnaround_time = (float)turnaround_time / num_processes;
57 float avg_response_time = (float)response_time / num_processes;
58 printf("Throughput: %.3f\n", throughput);
59 printf("Average turnaround time: %.3f\n", avg_turnaround_time);
60 printf("Average response time: %.3f\n", avg_response_time);
61 return;
62 }

```

Listing 2: FIFO Scheduling Algorithm

The FIFO scheduling algorithm is the simplest of all, it just runs the processes in the order they arrive. The processes are stored in a queue, and the first process is removed from the head of the queue and run. If the process has not run before, its first runtime is set to the current system time, and the response time is calculated. The process is then run for 1 tick, and if it has finished, the turnaround time is calculated and the next process is removed from the queue. If the process has not finished, it is kept running until it finishes. If the queue is empty, the process is kept running until it finishes. If the queue is not empty, the next process is removed from the queue and run. This process is repeated until all processes have finished running, and both queues become empty. Then the performance metrics are used to calculate the average turnaround time, average response time, and throughput which is then printed out, and the function returns.

## 5.2 SJF - Shortest Job First

```

1 void sched_SJF(dlq *const p_fq, int *p_time){
2     dlq queue; queue.head = queue.tail = NULL;
3     dlq_node *process = remove_from_head(p_fq);
4
5     int num_processes = 1, rt1 = 0, response_time = 0, turnaround_time = 0,
6         first_arrival = process->data->ptimearrival;
7
8     while(1){
9         (*p_time)++;
10
11         if(!(process->data->ptimeleft) && is_empty(&queue) && is_empty(p_fq))
12             break;
13         // if processes left with arrival time less than system time, add
14         // them to the tail of the queue, and sort by time to completion
15         if(!is_empty(p_fq) && p_fq->head->data->ptimearrival < (*p_time)){
16             add_to_tail(&queue, remove_from_head(p_fq));
17             num_processes++;
18             sort_by_timetocompletion(&queue);
19         }
20
21         if(process->data->isfirsttime == true){
22             process->data->isfirsttime = false;
23             process->data->pfirstruntime = (*p_time);
24             rt1 = (*p_time) - process->data->ptimearrival;
25             if (rt1 < 0) rt1 = 1;
26             response_time += rt1;
27         }
28
29         printf("%d:", *p_time);
30         // if process still has to arrive, print idle, else decrement its
31         // time and print its name
32         if(process->data->ptimearrival >= (*p_time)) printf("idle:");
33         else{
34             process->data->ptimeleft--;
35             printf("%s:", process->data->pname);
36         }
37
38         // if queue is empty, print empty, else show contents of the queue
39         if(is_empty(&queue)) printf("empty:\n");
40         else{
41             print_q(&queue);
42             printf(":\n");
43         }
44
45         if(process->data->ptimeleft == 0){
46             turnaround_time += (*p_time) - process->data->ptimearrival;
47             if(!is_empty(&queue))
48                 process = remove_from_head(&queue);
49         }
50     }
51     free(process);
52     float throughput = (float)num_processes / ((*p_time) - first_arrival);

```

```

49 float avg_turnaround_time = (float)turnaround_time / num_processes;
50 float avg_response_time = (float)response_time / num_processes;
51 printf("Throughput: %.3f\n", throughput);
52 printf("Average turnaround time: %.3f\n", avg_turnaround_time);
53 printf("Average response time: %.3f\n", avg_response_time);
54 return;
55 }

```

Listing 3: SJF Scheduling Algorithm

The Shortest Job First (SJF) algorithm derives from the FIFO algorithm. When a process arrives and is added to the ready queue, the queue is sorted by time to completion. Therefore, when the current process ends, the next process to run will be the one that has the shortest completion time. This simulates the shortest job first algorithm as the shortest job will always be the next one to run. The rest of the algorithm is the same as FIFO, and the performance metrics are calculated and printed at the end.

### 5.3 STCF - Shortest Time to Completion First

```

1 void sched_STCF(dlq *const p_fq, int *p_time){
2     dlq queue; queue.head = queue.tail = NULL;
3     dlq_node *process = remove_from_head(p_fq);
4
5     int num_processes = 1, rt1 = 0, response_time = 0, turnaround_time = 0,
6         first_arrival = process->data->ptimearrival;
7
8     while(1){
9         (*p_time)++;
10        if(!(process->data->ptimeleft) && is_empty(&queue) && is_empty(p_fq))
11            break;
12
13        if(!is_empty(p_fq) && p_fq->head->data->ptimearrival < (*p_time)){
14            add_to_tail(&queue, remove_from_head(p_fq));
15            num_processes++;
16        }
17
18        if(process->data->isfirsttime == true){
19            process->data->isfirsttime = false;
20            process->data->pfirstruntime = (*p_time);
21            rt1 = (*p_time) - process->data->ptimearrival;
22            if (rt1 < 0) rt1 = 1;
23            response_time += rt1;
24        }
25
26        sort_by_timetocompletion(&queue); // sort the queue by time to
27        completion
28
29        printf("%d:", *p_time);
30        if(process->data->ptimearrival >= (*p_time))
31            printf("idle:");
32        else{
33            if(!is_empty(&queue) && (process->data->ptimeleft > queue.head->
34                data->ptimeleft)){

```



```

32     add_to_tail(&queue, process);
33     process = remove_from_head(&queue);
34 }
35 if(process->data->isfirsttime == false){
36     process->data->isfirsttime = true;
37     process->data->pfirstruntime = (*p_time);
38     rt1 = (*p_time) - process->data->ptimearrival;
39     if (rt1 < 0) rt1 = 1;
40     response_time += rt1;
41 }
42 process->data->ptimeleft--;
43 printf("%s:", process->data->pname);
44 }
45
46 if(is_empty(&queue))
47     printf("empty:\n");
48 else{
49     sort_by_timetocompletion(&queue);
50     print_q(&queue);
51     printf(":\n");
52 }
53
54 if(process->data->ptimeleft == 0){
55     turnaround_time += (*p_time) - process->data->ptimearrival;
56     if(!is_empty(&queue))
57         process = remove_from_head(&queue);
58 }
59 }
60 free(process);
61 float throughput = (float)num_processes / ((*p_time) - first_arrival);
62 float avg_turnaround_time = (float)turnaround_time / num_processes;
63 float avg_response_time = (float)response_time / num_processes;
64 printf("Throughput: %.3f\n", throughput);
65 printf("Average turnaround time: %.3f\n", avg_turnaround_time);
66 printf("Average response time: %.3f\n", avg_response_time);
67 return;
68 }

```

Listing 4: STCF Scheduling Algorithm

The Shortest Time to Completion First (STCF) is a modification of the SJF algorithm. The ready queue is sorted by time to completion, and the process with the shortest time to completion is run. If a process arrives with a shorter time to completion than the current process, the current process is added to the tail of the queue and the new process is run (the queue is again sorted based on time to completion so that the shortest algorithm is run next). The performance metrics are then calculated and printed at the end.

## 5.4 RR - Round Robin

```

1 void sched_RR(dlq *const p_fq, int *p_time){
2     dlq queue; queue.head = queue.tail = NULL;
3     dlq_node *process = remove_from_head(p_fq);
4     int quantum_time = 1, process_time = 0; // quantum is the time slice
        for each process, process_time is the time the process has been
        running
5
6     int num_processes = 1, rt1 = 0, response_time = 0, turnaround_time = 0,
        first_arrival = process->data->ptimearrival;
7
8     while(1){
9         (*p_time)++;
10        if (!(process->data->ptimeleft) && is_empty(&queue) && is_empty(p_fq)
11            )
12            break;
13
14        if(!is_empty(p_fq) && p_fq->head->data->ptimearrival < (*p_time)){
15            add_to_tail(&queue, remove_from_head(p_fq));
16            num_processes++;
17        }
18
19        if(process->data->isfirsttime == true){
20            process->data->isfirsttime = false;
21            process->data->pfirstruntime = (*p_time);
22            rt1 = (*p_time) - process->data->ptimearrival;
23            if (rt1 < 0) rt1 = 1;
24            response_time += rt1;
25        }
26
27        printf("%d:", *p_time);
28        if(process->data->ptimearrival >= (*p_time))
29            printf("idle:");
30        else{ // if process has arrived, decrement its time, update the time
31            it has run for, and print its name
32            process->data->ptimeleft--;
33            process_time++;
34            printf("%s:", process->data->pname);
35        }
36
37        if(is_empty(&queue))
38            printf("empty:\n");
39        else{
40            print_q(&queue);
41            printf(":\n");
42        }
43
44        if(process->data->ptimeleft == 0){
45            turnaround_time += (*p_time) - process->data->ptimearrival;
46            if (!is_empty(&queue))
47                process = remove_from_head(&queue);
48        }
49    }

```

```
47
48     if(process_time == quantum_time){ // if process has run for the time
49         slice, add it to the tail of the queue and set process_time to 0
50         add_to_tail(&queue, process);
51         process = remove_from_head(&queue);
52         process_time = 0;
53     }
54 }
55 free(process);
56 float throughput = (float)num_processes / ((*p_time) - first_arrival);
57 float avg_turnaround_time = (float)turnaround_time / num_processes;
58 float avg_response_time = (float)response_time / num_processes;
59 printf("Throughput: %.3f\n", throughput);
60 printf("Average turnaround time: %.3f\n", avg_turnaround_time);
61 printf("Average response time: %.3f\n", avg_response_time);
62 return;
63 }
```

Listing 5: RR Scheduling Algorithm

The Round Robin (RR) is a modification of the FIFO code. A quantum time slice is set to 1 tick, and each process is run for the time slice. (The quantum time slice can be changed as per preference). On each iteration, the process time is set to 0, and incremented once a process has run. If the process has run for the time slice, it is added to the tail of the queue and the next process is run. If the process has completed, it is removed from the queue and the next process is run (process time is set to 0). The performance metrics are then calculated and printed at the end.

## 6 Performance Metrics of Scheduling Algorithms

Each of the four scheduling algorithms are run over some sample test cases, and the performance metrics are calculated for each of them. More specifically, the metrics being measured are:

1. Average Throughput of the scheduler
2. Average Response Time of the scheduler
3. Average Turnaround Time of the scheduler

Outputs have been attached in the appendix for each of the test cases, and the performance metrics have been calculated and tabulated in the following sections.

*\*For each test case, green highlight indicates best performance, and red highlight indicates worst performance.*

### 6.1 Testing Performance Metrics

#### 6.1.1 Test Case 0 / Test Case 2 (same test case)

Sample Input:

3

*Scheduling Algorithm*

P1:12:7:3

P2:15:3:5

P3:1:6:2

Metric \ Scheduling Alg	FIFO	SJF	STCF	RR
Throughput	0.176	0.176	0.176	0.176
Average Response Time	6.000	4.667	4.667	2.000
Average Turnaround Time	10.333	9.000	9.000	12.667

Table 1: Test Case 0 / Test Case 2 Performance Metrics

The above table shows that the throughput remains the same throughout which is logical as the number of processes and the time taken by each process, in essence, remains same. In terms of average response time, RR performs better for this test case, while SJF and STCF have the same average turnaround time for this test case.

### 6.1.2 Test Case 5

Sample Input:

6

*Scheduling Algorithm*

P1:1:5:0

P2:2:7:2

P3:3:6:3

P4:4:9:4

P5:5:8:5

P6:6:4:7

Metric \ Scheduling Alg	FIFO	SJF	STCF	RR
Throughput	0.150	0.150	0.150	0.150
Average Response Time	13.667	11.333	11.333	4.000
Average Turnaround Time	19.167	16.833	16.833	25.333

Table 2: Test Case 5 Performance Metrics

Again, the throughput remains same throughout, average response time is best for RR, and average turnaround time is best for SJF and STCF.

### 6.1.3 Test Case 10

Sample Input:

6

*Scheduling Algorithm*

P1:1:6:0

P2:2:12:2

P3:3:8:4

P4:4:15:5

P5:5:5:7

P6:6:10:9

Metric \ Scheduling Alg	FIFO	SJF	STCF	RR
Throughput	0.105	0.105	0.105	0.105
Average Response Time	19.333	14.667	13.500	3.333
Average Turnaround Time	27.667	23.000	22.667	34.667

Table 3: Test Case 10 Performance Metrics

Again, the throughput remains same throughout, average response time is best for RR, and average turnaround time is almost the same for SJF and STCF, however, STCF performs slightly better. Hence, STCF has the best average response time for this test case8.

### 6.1.4 Test Case 13

Sample Input:

8

*Scheduling Algorithm*

P1:1:10:0

P2:2:15:2

P3:3:8:4

P4:4:12:6

P5:5:6:9

P6:6:4:11

P7:7:7:13

P8:8:9:15

Metric \ Scheduling Alg	FIFO	SJF	STCF	RR
Throughput	0.111	0.111	0.111	0.111
Average Response Time	28.625	19.500	18.875	4.500
Average Turnaround Time	36.500	27.375	27.250	49.000

Table 4: Test Case 13 Performance Metrics

Again, the throughput remains same throughout, average response time is best for RR, and average turnaround time is best for STCF.

## 6.2 Results - Comparison of Performance Metrics

From the above results, a cumulative results table can be made:

Algorithms	Metrics \ Test Case #	Test Case 0 / 2	Test Case 5	Test Case 10	Test Case 13	Cumulative
FIFO	Throughput	0.176	0.150	0.105	0.111	0.542
	Avg Response Time	6.000	13.667	19.333	28.625	67.625
	Avg Turnaround Time	10.333	19.167	27.667	36.500	93.667
SJF	Throughput	0.176	0.150	0.105	0.111	0.542
	Avg Response Time	4.667	11.333	14.667	19.500	50.167
	Avg Turnaround Time	9.000	16.833	23.000	27.375	76.208
STCF	Throughput	0.176	0.150	0.105	0.111	0.542
	Avg Response Time	4.667	11.333	13.500	18.875	48.375
	Avg Turnaround Time	9.000	16.833	22.667	27.250	75.750
RR	Throughput	0.176	0.150	0.105	0.111	0.542
	Avg Response Time	2.000	4.000	3.333	4.500	13.833
	Avg Turnaround Time	12.667	25.333	34.667	49.000	121.667

Table 5: Cumulative Overall Performance Metrics

- **Best Throughput:** Remains same for all which is logical since cumulatively, all processes in a test case will have the same total time regardless of the algorithm used.
- **Best Avg Response Time:** *Round Robin*, which is logical since RR ensures that each process gets a fair share of the CPU time as it executes in a round robin manner.
- **Best Avg Turnaround Time:** *STCF*; again logical since STCF ensures a process with lesser runtime runs first even if a process is currently running.

## 7 Takeaway and Reflections

This homework was significantly, comparatively, much easier. It was a great learning experience about implementing the different scheduling algorithms and then modifying them to compute their performance metrics to find out which algorithms performs better with regards to which aspect. It was fun and interesting.



## 8 References

### References

- [1] Remzi H. Arpaci-Desseau and Andrea C. Arpaci-Desseau. *Operating Systems: Three Easy Pieces*. Arpaci-Dusseau Books, LLC, 2015.
- [2] *ChatGPT*. [Online]. Available: <https://chat.openai.com/> mainly for understanding, resolving errors, and commentation.
- [3] *Bing AI ChatBot*. [Online]. Available: <https://www.bing.com/search?toWww=1&redig=0011FC72B09C43A9A833284287211CB9&q=Bing+AI&showconv=1> mainly for understanding, resolving errors, and commentation. (Open with Microsoft Edge)



## A Appendix

### A.1 Outputs for Test Cases

#### A.1.1 Test Case 0 / Test Case 2

##### FIFO

```
1:idle:empty:
2:idle:empty:
3:P3:empty:
4:P3:P1(7),:
5:P3:P1(7),:
6:P3:P1(7),P2(3),:
7:P3:P1(7),P2(3),:
8:P3:P1(7),P2(3),:
9:P1:P2(3),:
10:P1:P2(3),:
11:P1:P2(3),:
12:P1:P2(3),:
13:P1:P2(3),:
14:P1:P2(3),:
15:P1:P2(3),:
16:P2:empty:
17:P2:empty:
18:P2:empty:
Throughput: 0.176
Average turnaround time: 10.333
Average response time: 6.000
```

##### SJF

```
1:idle:empty:
2:idle:empty:
3:P3:empty:
4:P3:P1(7),:
5:P3:P1(7),:
6:P3:P2(3),P1(7),:
7:P3:P2(3),P1(7),:
8:P3:P2(3),P1(7),:
9:P2:P1(7),:
10:P2:P1(7),:
11:P2:P1(7),:
12:P1:empty:
13:P1:empty:
14:P1:empty:
15:P1:empty:
16:P1:empty:
17:P1:empty:
18:P1:empty:
Throughput: 0.176
```

Average turnaround time: 9.000

Average response time: 4.667

### STCF

1:idle:empty:

2:idle:empty:

3:P3:empty:

4:P3:P1(7),:

5:P3:P1(7),:

6:P3:P2(3),P1(7),:

7:P3:P2(3),P1(7),:

8:P3:P2(3),P1(7),:

9:P2:P1(7),:

10:P2:P1(7),:

11:P2:P1(7),:

12:P1:empty:

13:P1:empty:

14:P1:empty:

15:P1:empty:

16:P1:empty:

17:P1:empty:

18:P1:empty:

Throughput: 0.176

Average turnaround time: 9.000

Average response time: 4.667

### RR

1:idle:empty:

2:idle:empty:

3:P3:empty:

4:P3:P1(7),:

5:P1:P3(4),:

6:P3:P1(6),P2(3),:

7:P1:P2(3),P3(3),:

8:P2:P3(3),P1(5),:

9:P3:P1(5),P2(2),:

10:P1:P2(2),P3(2),:

11:P2:P3(2),P1(4),:

12:P3:P1(4),P2(1),:

13:P1:P2(1),P3(1),:

14:P2:P3(1),P1(3),:

15:P1:P3(1),:

16:P3:P1(2),:

17:P1:empty:

18:P1:empty:

Throughput: 0.176

Average turnaround time: 12.667

Average response time: 2.000

**A.1.2 Test Case 5****FIFO**

```
1:P1:empty:
2:P1:empty:
3:P1:P2(7),:
4:P1:P2(7),P3(6),:
5:P1:P2(7),P3(6),P4(9),:
6:P2:P3(6),P4(9),P5(8),:
7:P2:P3(6),P4(9),P5(8),:
8:P2:P3(6),P4(9),P5(8),P6(4),:
9:P2:P3(6),P4(9),P5(8),P6(4),:
10:P2:P3(6),P4(9),P5(8),P6(4),:
11:P2:P3(6),P4(9),P5(8),P6(4),:
12:P2:P3(6),P4(9),P5(8),P6(4),:
13:P3:P4(9),P5(8),P6(4),:
14:P3:P4(9),P5(8),P6(4),:
15:P3:P4(9),P5(8),P6(4),:
16:P3:P4(9),P5(8),P6(4),:
17:P3:P4(9),P5(8),P6(4),:
18:P3:P4(9),P5(8),P6(4),:
19:P4:P5(8),P6(4),:
20:P4:P5(8),P6(4),:
21:P4:P5(8),P6(4),:
22:P4:P5(8),P6(4),:
23:P4:P5(8),P6(4),:
24:P4:P5(8),P6(4),:
25:P4:P5(8),P6(4),:
26:P4:P5(8),P6(4),:
27:P4:P5(8),P6(4),:
28:P5:P6(4),:
29:P5:P6(4),:
30:P5:P6(4),:
31:P5:P6(4),:
32:P5:P6(4),:
33:P5:P6(4),:
34:P5:P6(4),:
35:P5:P6(4),:
36:P6:empty:
37:P6:empty:
38:P6:empty:
39:P6:empty:
Throughput: 0.150
Average turnaround time: 19.167
Average response time: 13.667
```

**SJF**

```
1:P1:empty:
2:P1:empty:
```

3:P1:P2(7),:  
4:P1:P3(6),P2(7),:  
5:P1:P3(6),P2(7),P4(9),:  
6:P3:P2(7),P5(8),P4(9),:  
7:P3:P2(7),P5(8),P4(9),:  
8:P3:P6(4),P2(7),P5(8),P4(9),:  
9:P3:P6(4),P2(7),P5(8),P4(9),:  
10:P3:P6(4),P2(7),P5(8),P4(9),:  
11:P3:P6(4),P2(7),P5(8),P4(9),:  
12:P6:P2(7),P5(8),P4(9),:  
13:P6:P2(7),P5(8),P4(9),:  
14:P6:P2(7),P5(8),P4(9),:  
15:P6:P2(7),P5(8),P4(9),:  
16:P2:P5(8),P4(9),:  
17:P2:P5(8),P4(9),:  
18:P2:P5(8),P4(9),:  
19:P2:P5(8),P4(9),:  
20:P2:P5(8),P4(9),:  
21:P2:P5(8),P4(9),:  
22:P2:P5(8),P4(9),:  
23:P5:P4(9),:  
24:P5:P4(9),:  
25:P5:P4(9),:  
26:P5:P4(9),:  
27:P5:P4(9),:  
28:P5:P4(9),:  
29:P5:P4(9),:  
30:P5:P4(9),:  
31:P4:empty:  
32:P4:empty:  
33:P4:empty:  
34:P4:empty:  
35:P4:empty:  
36:P4:empty:  
37:P4:empty:  
38:P4:empty:  
39:P4:empty:  
Throughput: 0.150  
Average turnaround time: 16.833  
Average response time: 11.333

**STCF**

1:P1:empty:  
2:P1:empty:  
3:P1:P2(7),:  
4:P1:P3(6),P2(7),:  
5:P1:P3(6),P2(7),P4(9),:  
6:P3:P2(7),P5(8),P4(9),:  
7:P3:P2(7),P5(8),P4(9),:

```

8:P3:P6(4),P2(7),P5(8),P4(9),:
9:P3:P6(4),P2(7),P5(8),P4(9),:
10:P3:P6(4),P2(7),P5(8),P4(9),:
11:P3:P6(4),P2(7),P5(8),P4(9),:
12:P6:P2(7),P5(8),P4(9),:
13:P6:P2(7),P5(8),P4(9),:
14:P6:P2(7),P5(8),P4(9),:
15:P6:P2(7),P5(8),P4(9),:
16:P2:P5(8),P4(9),:
17:P2:P5(8),P4(9),:
18:P2:P5(8),P4(9),:
19:P2:P5(8),P4(9),:
20:P2:P5(8),P4(9),:
21:P2:P5(8),P4(9),:
22:P2:P5(8),P4(9),:
23:P5:P4(9),:
24:P5:P4(9),:
25:P5:P4(9),:
26:P5:P4(9),:
27:P5:P4(9),:
28:P5:P4(9),:
29:P5:P4(9),:
30:P5:P4(9),:
31:P4:empty:
32:P4:empty:
33:P4:empty:
34:P4:empty:
35:P4:empty:
36:P4:empty:
37:P4:empty:
38:P4:empty:
39:P4:empty:
Throughput: 0.150
Average turnaround time: 16.833
Average response time: 11.333

```

**RR**

```

1:P1:empty:
2:P1:empty:
3:P1:P2(7),:
4:P2:P1(2),P3(6),:
5:P1:P3(6),P2(6),P4(9),:
6:P3:P2(6),P4(9),P1(1),P5(8),:
7:P2:P4(9),P1(1),P5(8),P3(5),:
8:P4:P1(1),P5(8),P3(5),P2(5),P6(4),:
9:P1:P5(8),P3(5),P2(5),P6(4),P4(8),:
10:P3:P2(5),P6(4),P4(8),P5(8),:
11:P2:P6(4),P4(8),P5(8),P3(4),:
12:P6:P4(8),P5(8),P3(4),P2(4),:

```

13:P4:P5(8),P3(4),P2(4),P6(3),:  
 14:P5:P3(4),P2(4),P6(3),P4(7),:  
 15:P3:P2(4),P6(3),P4(7),P5(7),:  
 16:P2:P6(3),P4(7),P5(7),P3(3),:  
 17:P6:P4(7),P5(7),P3(3),P2(3),:  
 18:P4:P5(7),P3(3),P2(3),P6(2),:  
 19:P5:P3(3),P2(3),P6(2),P4(6),:  
 20:P3:P2(3),P6(2),P4(6),P5(6),:  
 21:P2:P6(2),P4(6),P5(6),P3(2),:  
 22:P6:P4(6),P5(6),P3(2),P2(2),:  
 23:P4:P5(6),P3(2),P2(2),P6(1),:  
 24:P5:P3(2),P2(2),P6(1),P4(5),:  
 25:P3:P2(2),P6(1),P4(5),P5(5),:  
 26:P2:P6(1),P4(5),P5(5),P3(1),:  
 27:P6:P4(5),P5(5),P3(1),P2(1),:  
 28:P5:P3(1),P2(1),P4(5),:  
 29:P3:P2(1),P4(5),P5(4),:  
 30:P4:P5(4),P2(1),:  
 31:P5:P2(1),P4(4),:  
 32:P2:P4(4),P5(3),:  
 33:P5:P4(4),:  
 34:P4:P5(2),:  
 35:P5:P4(3),:  
 36:P4:P5(1),:  
 37:P5:P4(2),:  
 38:P4:empty:  
 39:P4:empty:  
 Throughput: 0.150  
 Average turnaround time: 25.333  
 Average response time: 4.000

### A.1.3 Test Case 10

#### FIFO

1:P1:empty:  
 2:P1:empty:  
 3:P1:P2(12),:  
 4:P1:P2(12),:  
 5:P1:P2(12),P3(8),:  
 6:P1:P2(12),P3(8),P4(15),:  
 7:P2:P3(8),P4(15),:  
 8:P2:P3(8),P4(15),P5(5),:  
 9:P2:P3(8),P4(15),P5(5),:  
 10:P2:P3(8),P4(15),P5(5),P6(10),:  
 11:P2:P3(8),P4(15),P5(5),P6(10),:  
 12:P2:P3(8),P4(15),P5(5),P6(10),:  
 13:P2:P3(8),P4(15),P5(5),P6(10),:  
 14:P2:P3(8),P4(15),P5(5),P6(10),:  
 15:P2:P3(8),P4(15),P5(5),P6(10),:

```
16:P2:P3(8),P4(15),P5(5),P6(10),:
17:P2:P3(8),P4(15),P5(5),P6(10),:
18:P2:P3(8),P4(15),P5(5),P6(10),:
19:P3:P4(15),P5(5),P6(10),:
20:P3:P4(15),P5(5),P6(10),:
21:P3:P4(15),P5(5),P6(10),:
22:P3:P4(15),P5(5),P6(10),:
23:P3:P4(15),P5(5),P6(10),:
24:P3:P4(15),P5(5),P6(10),:
25:P3:P4(15),P5(5),P6(10),:
26:P3:P4(15),P5(5),P6(10),:
27:P4:P5(5),P6(10),:
28:P4:P5(5),P6(10),:
29:P4:P5(5),P6(10),:
30:P4:P5(5),P6(10),:
31:P4:P5(5),P6(10),:
32:P4:P5(5),P6(10),:
33:P4:P5(5),P6(10),:
34:P4:P5(5),P6(10),:
35:P4:P5(5),P6(10),:
36:P4:P5(5),P6(10),:
37:P4:P5(5),P6(10),:
38:P4:P5(5),P6(10),:
39:P4:P5(5),P6(10),:
40:P4:P5(5),P6(10),:
41:P4:P5(5),P6(10),:
42:P5:P6(10),:
43:P5:P6(10),:
44:P5:P6(10),:
45:P5:P6(10),:
46:P5:P6(10),:
47:P6:empty:
48:P6:empty:
49:P6:empty:
50:P6:empty:
51:P6:empty:
52:P6:empty:
53:P6:empty:
54:P6:empty:
55:P6:empty:
56:P6:empty:
Throughput: 0.105
Average turnaround time: 27.667
Average response time: 19.333
```

**SJF**

```
1:P1:empty:
2:P1:empty:
3:P1:P2(12),:
```

4:P1:P2(12),:  
5:P1:P3(8),P2(12),:  
6:P1:P3(8),P2(12),P4(15),:  
7:P3:P2(12),P4(15),:  
8:P3:P5(5),P2(12),P4(15),:  
9:P3:P5(5),P2(12),P4(15),:  
10:P3:P5(5),P6(10),P2(12),P4(15),:  
11:P3:P5(5),P6(10),P2(12),P4(15),:  
12:P3:P5(5),P6(10),P2(12),P4(15),:  
13:P3:P5(5),P6(10),P2(12),P4(15),:  
14:P3:P5(5),P6(10),P2(12),P4(15),:  
15:P5:P6(10),P2(12),P4(15),:  
16:P5:P6(10),P2(12),P4(15),:  
17:P5:P6(10),P2(12),P4(15),:  
18:P5:P6(10),P2(12),P4(15),:  
19:P5:P6(10),P2(12),P4(15),:  
20:P6:P2(12),P4(15),:  
21:P6:P2(12),P4(15),:  
22:P6:P2(12),P4(15),:  
23:P6:P2(12),P4(15),:  
24:P6:P2(12),P4(15),:  
25:P6:P2(12),P4(15),:  
26:P6:P2(12),P4(15),:  
27:P6:P2(12),P4(15),:  
28:P6:P2(12),P4(15),:  
29:P6:P2(12),P4(15),:  
30:P2:P4(15),:  
31:P2:P4(15),:  
32:P2:P4(15),:  
33:P2:P4(15),:  
34:P2:P4(15),:  
35:P2:P4(15),:  
36:P2:P4(15),:  
37:P2:P4(15),:  
38:P2:P4(15),:  
39:P2:P4(15),:  
40:P2:P4(15),:  
41:P2:P4(15),:  
42:P4:empty:  
43:P4:empty:  
44:P4:empty:  
45:P4:empty:  
46:P4:empty:  
47:P4:empty:  
48:P4:empty:  
49:P4:empty:  
50:P4:empty:  
51:P4:empty:  
52:P4:empty:



53:P4:empty:  
54:P4:empty:  
55:P4:empty:  
56:P4:empty:  
Throughput: 0.105  
Average turnaround time: 23.000  
Average response time: 14.667

**STCF**

1:P1:empty:  
2:P1:empty:  
3:P1:P2(12),:  
4:P1:P2(12),:  
5:P1:P3(8),P2(12),:  
6:P1:P3(8),P2(12),P4(15),:  
7:P3:P2(12),P4(15),:  
8:P5:P3(7),P2(12),P4(15),:  
9:P5:P3(7),P2(12),P4(15),:  
10:P5:P3(7),P6(10),P2(12),P4(15),:  
11:P5:P3(7),P6(10),P2(12),P4(15),:  
12:P5:P3(7),P6(10),P2(12),P4(15),:  
13:P3:P6(10),P2(12),P4(15),:  
14:P3:P6(10),P2(12),P4(15),:  
15:P3:P6(10),P2(12),P4(15),:  
16:P3:P6(10),P2(12),P4(15),:  
17:P3:P6(10),P2(12),P4(15),:  
18:P3:P6(10),P2(12),P4(15),:  
19:P3:P6(10),P2(12),P4(15),:  
20:P6:P2(12),P4(15),:  
21:P6:P2(12),P4(15),:  
22:P6:P2(12),P4(15),:  
23:P6:P2(12),P4(15),:  
24:P6:P2(12),P4(15),:  
25:P6:P2(12),P4(15),:  
26:P6:P2(12),P4(15),:  
27:P6:P2(12),P4(15),:  
28:P6:P2(12),P4(15),:  
29:P6:P2(12),P4(15),:  
30:P2:P4(15),:  
31:P2:P4(15),:  
32:P2:P4(15),:  
33:P2:P4(15),:  
34:P2:P4(15),:  
35:P2:P4(15),:  
36:P2:P4(15),:  
37:P2:P4(15),:  
38:P2:P4(15),:  
39:P2:P4(15),:  
40:P2:P4(15),:

41:P2:P4(15),:  
42:P4:empty:  
43:P4:empty:  
44:P4:empty:  
45:P4:empty:  
46:P4:empty:  
47:P4:empty:  
48:P4:empty:  
49:P4:empty:  
50:P4:empty:  
51:P4:empty:  
52:P4:empty:  
53:P4:empty:  
54:P4:empty:  
55:P4:empty:  
56:P4:empty:  
Throughput: 0.105  
Average turnaround time: 22.667  
Average response time: 13.500

**RR**

1:P1:empty:  
2:P1:empty:  
3:P1:P2(12),:  
4:P2:P1(3),:  
5:P1:P2(11),P3(8),:  
6:P2:P3(8),P1(2),P4(15),:  
7:P3:P1(2),P4(15),P2(10),:  
8:P1:P4(15),P2(10),P3(7),P5(5),:  
9:P4:P2(10),P3(7),P5(5),P1(1),:  
10:P2:P3(7),P5(5),P1(1),P4(14),P6(10),:  
11:P3:P5(5),P1(1),P4(14),P6(10),P2(9),:  
12:P5:P1(1),P4(14),P6(10),P2(9),P3(6),:  
13:P1:P4(14),P6(10),P2(9),P3(6),P5(4),:  
14:P6:P2(9),P3(6),P5(4),P4(14),:  
15:P2:P3(6),P5(4),P4(14),P6(9),:  
16:P3:P5(4),P4(14),P6(9),P2(8),:  
17:P5:P4(14),P6(9),P2(8),P3(5),:  
18:P4:P6(9),P2(8),P3(5),P5(3),:  
19:P6:P2(8),P3(5),P5(3),P4(13),:  
20:P2:P3(5),P5(3),P4(13),P6(8),:  
21:P3:P5(3),P4(13),P6(8),P2(7),:  
22:P5:P4(13),P6(8),P2(7),P3(4),:  
23:P4:P6(8),P2(7),P3(4),P5(2),:  
24:P6:P2(7),P3(4),P5(2),P4(12),:  
25:P2:P3(4),P5(2),P4(12),P6(7),:  
26:P3:P5(2),P4(12),P6(7),P2(6),:  
27:P5:P4(12),P6(7),P2(6),P3(3),:  
28:P4:P6(7),P2(6),P3(3),P5(1),:

29:P6:P2(6),P3(3),P5(1),P4(11),:  
 30:P2:P3(3),P5(1),P4(11),P6(6),:  
 31:P3:P5(1),P4(11),P6(6),P2(5),:  
 32:P5:P4(11),P6(6),P2(5),P3(2),:  
 33:P6:P2(5),P3(2),P4(11),:  
 34:P2:P3(2),P4(11),P6(5),:  
 35:P3:P4(11),P6(5),P2(4),:  
 36:P4:P6(5),P2(4),P3(1),:  
 37:P6:P2(4),P3(1),P4(10),:  
 38:P2:P3(1),P4(10),P6(4),:  
 39:P3:P4(10),P6(4),P2(3),:  
 40:P6:P2(3),P4(10),:  
 41:P2:P4(10),P6(3),:  
 42:P4:P6(3),P2(2),:  
 43:P6:P2(2),P4(9),:  
 44:P2:P4(9),P6(2),:  
 45:P4:P6(2),P2(1),:  
 46:P6:P2(1),P4(8),:  
 47:P2:P4(8),P6(1),:  
 48:P6:P4(8),:  
 49:P4:empty:  
 50:P4:empty:  
 51:P4:empty:  
 52:P4:empty:  
 53:P4:empty:  
 54:P4:empty:  
 55:P4:empty:  
 56:P4:empty:  
 Throughput: 0.105  
 Average turnaround time: 34.667  
 Average response time: 3.333

#### A.1.4 Test Case 13

##### FIFO

1:P1:empty:  
 2:P1:empty:  
 3:P1:P2(15),:  
 4:P1:P2(15),:  
 5:P1:P2(15),P3(8),:  
 6:P1:P2(15),P3(8),:  
 7:P1:P2(15),P3(8),P4(12),:  
 8:P1:P2(15),P3(8),P4(12),:  
 9:P1:P2(15),P3(8),P4(12),:  
 10:P1:P2(15),P3(8),P4(12),P5(6),:  
 11:P2:P3(8),P4(12),P5(6),:  
 12:P2:P3(8),P4(12),P5(6),P6(4),:  
 13:P2:P3(8),P4(12),P5(6),P6(4),:  
 14:P2:P3(8),P4(12),P5(6),P6(4),P7(7),:

```
15:P2:P3(8),P4(12),P5(6),P6(4),P7(7),:
16:P2:P3(8),P4(12),P5(6),P6(4),P7(7),P8(9),:
17:P2:P3(8),P4(12),P5(6),P6(4),P7(7),P8(9),:
18:P2:P3(8),P4(12),P5(6),P6(4),P7(7),P8(9),:
19:P2:P3(8),P4(12),P5(6),P6(4),P7(7),P8(9),:
20:P2:P3(8),P4(12),P5(6),P6(4),P7(7),P8(9),:
21:P2:P3(8),P4(12),P5(6),P6(4),P7(7),P8(9),:
22:P2:P3(8),P4(12),P5(6),P6(4),P7(7),P8(9),:
23:P2:P3(8),P4(12),P5(6),P6(4),P7(7),P8(9),:
24:P2:P3(8),P4(12),P5(6),P6(4),P7(7),P8(9),:
25:P2:P3(8),P4(12),P5(6),P6(4),P7(7),P8(9),:
26:P3:P4(12),P5(6),P6(4),P7(7),P8(9),:
27:P3:P4(12),P5(6),P6(4),P7(7),P8(9),:
28:P3:P4(12),P5(6),P6(4),P7(7),P8(9),:
29:P3:P4(12),P5(6),P6(4),P7(7),P8(9),:
30:P3:P4(12),P5(6),P6(4),P7(7),P8(9),:
31:P3:P4(12),P5(6),P6(4),P7(7),P8(9),:
32:P3:P4(12),P5(6),P6(4),P7(7),P8(9),:
33:P3:P4(12),P5(6),P6(4),P7(7),P8(9),:
34:P4:P5(6),P6(4),P7(7),P8(9),:
35:P4:P5(6),P6(4),P7(7),P8(9),:
36:P4:P5(6),P6(4),P7(7),P8(9),:
37:P4:P5(6),P6(4),P7(7),P8(9),:
38:P4:P5(6),P6(4),P7(7),P8(9),:
39:P4:P5(6),P6(4),P7(7),P8(9),:
40:P4:P5(6),P6(4),P7(7),P8(9),:
41:P4:P5(6),P6(4),P7(7),P8(9),:
42:P4:P5(6),P6(4),P7(7),P8(9),:
43:P4:P5(6),P6(4),P7(7),P8(9),:
44:P4:P5(6),P6(4),P7(7),P8(9),:
45:P4:P5(6),P6(4),P7(7),P8(9),:
46:P5:P6(4),P7(7),P8(9),:
47:P5:P6(4),P7(7),P8(9),:
48:P5:P6(4),P7(7),P8(9),:
49:P5:P6(4),P7(7),P8(9),:
50:P5:P6(4),P7(7),P8(9),:
51:P5:P6(4),P7(7),P8(9),:
52:P6:P7(7),P8(9),:
53:P6:P7(7),P8(9),:
54:P6:P7(7),P8(9),:
55:P6:P7(7),P8(9),:
56:P7:P8(9),:
57:P7:P8(9),:
58:P7:P8(9),:
59:P7:P8(9),:
60:P7:P8(9),:
61:P7:P8(9),:
62:P7:P8(9),:
63:P8:empty:
```

64:P8:empty:  
65:P8:empty:  
66:P8:empty:  
67:P8:empty:  
68:P8:empty:  
69:P8:empty:  
70:P8:empty:  
71:P8:empty:  
Throughput: 0.111  
Average turnaround time: 36.500  
Average response time: 28.625

### **SJF**

1:P1:empty:  
2:P1:empty:  
3:P1:P2(15),:  
4:P1:P2(15),:  
5:P1:P3(8),P2(15),:  
6:P1:P3(8),P2(15),:  
7:P1:P3(8),P4(12),P2(15),:  
8:P1:P3(8),P4(12),P2(15),:  
9:P1:P3(8),P4(12),P2(15),:  
10:P1:P5(6),P3(8),P4(12),P2(15),:  
11:P5:P3(8),P4(12),P2(15),:  
12:P5:P6(4),P3(8),P4(12),P2(15),:  
13:P5:P6(4),P3(8),P4(12),P2(15),:  
14:P5:P6(4),P7(7),P3(8),P4(12),P2(15),:  
15:P5:P6(4),P7(7),P3(8),P4(12),P2(15),:  
16:P5:P6(4),P7(7),P3(8),P8(9),P4(12),P2(15),:  
17:P6:P7(7),P3(8),P8(9),P4(12),P2(15),:  
18:P6:P7(7),P3(8),P8(9),P4(12),P2(15),:  
19:P6:P7(7),P3(8),P8(9),P4(12),P2(15),:  
20:P6:P7(7),P3(8),P8(9),P4(12),P2(15),:  
21:P7:P3(8),P8(9),P4(12),P2(15),:  
22:P7:P3(8),P8(9),P4(12),P2(15),:  
23:P7:P3(8),P8(9),P4(12),P2(15),:  
24:P7:P3(8),P8(9),P4(12),P2(15),:  
25:P7:P3(8),P8(9),P4(12),P2(15),:  
26:P7:P3(8),P8(9),P4(12),P2(15),:  
27:P7:P3(8),P8(9),P4(12),P2(15),:  
28:P3:P8(9),P4(12),P2(15),:  
29:P3:P8(9),P4(12),P2(15),:  
30:P3:P8(9),P4(12),P2(15),:  
31:P3:P8(9),P4(12),P2(15),:  
32:P3:P8(9),P4(12),P2(15),:  
33:P3:P8(9),P4(12),P2(15),:  
34:P3:P8(9),P4(12),P2(15),:  
35:P3:P8(9),P4(12),P2(15),:  
36:P8:P4(12),P2(15),:

37:P8:P4(12),P2(15),:  
38:P8:P4(12),P2(15),:  
39:P8:P4(12),P2(15),:  
40:P8:P4(12),P2(15),:  
41:P8:P4(12),P2(15),:  
42:P8:P4(12),P2(15),:  
43:P8:P4(12),P2(15),:  
44:P8:P4(12),P2(15),:  
45:P4:P2(15),:  
46:P4:P2(15),:  
47:P4:P2(15),:  
48:P4:P2(15),:  
49:P4:P2(15),:  
50:P4:P2(15),:  
51:P4:P2(15),:  
52:P4:P2(15),:  
53:P4:P2(15),:  
54:P4:P2(15),:  
55:P4:P2(15),:  
56:P4:P2(15),:  
57:P2:empty:  
58:P2:empty:  
59:P2:empty:  
60:P2:empty:  
61:P2:empty:  
62:P2:empty:  
63:P2:empty:  
64:P2:empty:  
65:P2:empty:  
66:P2:empty:  
67:P2:empty:  
68:P2:empty:  
69:P2:empty:  
70:P2:empty:  
71:P2:empty:

Throughput: 0.111

Average turnaround time: 27.375

Average response time: 19.500

### STCF

1:P1:empty:  
2:P1:empty:  
3:P1:P2(15),:  
4:P1:P2(15),:  
5:P1:P3(8),P2(15),:  
6:P1:P3(8),P2(15),:  
7:P1:P3(8),P4(12),P2(15),:  
8:P1:P3(8),P4(12),P2(15),:  
9:P1:P3(8),P4(12),P2(15),:

10:P1:P5(6),P3(8),P4(12),P2(15),:  
11:P5:P3(8),P4(12),P2(15),:  
12:P6:P5(5),P3(8),P4(12),P2(15),:  
13:P6:P5(5),P3(8),P4(12),P2(15),:  
14:P6:P5(5),P7(7),P3(8),P4(12),P2(15),:  
15:P6:P5(5),P7(7),P3(8),P4(12),P2(15),:  
16:P5:P7(7),P3(8),P8(9),P4(12),P2(15),:  
17:P5:P7(7),P3(8),P8(9),P4(12),P2(15),:  
18:P5:P7(7),P3(8),P8(9),P4(12),P2(15),:  
19:P5:P7(7),P3(8),P8(9),P4(12),P2(15),:  
20:P5:P7(7),P3(8),P8(9),P4(12),P2(15),:  
21:P7:P3(8),P8(9),P4(12),P2(15),:  
22:P7:P3(8),P8(9),P4(12),P2(15),:  
23:P7:P3(8),P8(9),P4(12),P2(15),:  
24:P7:P3(8),P8(9),P4(12),P2(15),:  
25:P7:P3(8),P8(9),P4(12),P2(15),:  
26:P7:P3(8),P8(9),P4(12),P2(15),:  
27:P7:P3(8),P8(9),P4(12),P2(15),:  
28:P3:P8(9),P4(12),P2(15),:  
29:P3:P8(9),P4(12),P2(15),:  
30:P3:P8(9),P4(12),P2(15),:  
31:P3:P8(9),P4(12),P2(15),:  
32:P3:P8(9),P4(12),P2(15),:  
33:P3:P8(9),P4(12),P2(15),:  
34:P3:P8(9),P4(12),P2(15),:  
35:P3:P8(9),P4(12),P2(15),:  
36:P8:P4(12),P2(15),:  
37:P8:P4(12),P2(15),:  
38:P8:P4(12),P2(15),:  
39:P8:P4(12),P2(15),:  
40:P8:P4(12),P2(15),:  
41:P8:P4(12),P2(15),:  
42:P8:P4(12),P2(15),:  
43:P8:P4(12),P2(15),:  
44:P8:P4(12),P2(15),:  
45:P4:P2(15),:  
46:P4:P2(15),:  
47:P4:P2(15),:  
48:P4:P2(15),:  
49:P4:P2(15),:  
50:P4:P2(15),:  
51:P4:P2(15),:  
52:P4:P2(15),:  
53:P4:P2(15),:  
54:P4:P2(15),:  
55:P4:P2(15),:  
56:P4:P2(15),:  
57:P2:empty:  
58:P2:empty:

59:P2:empty:  
60:P2:empty:  
61:P2:empty:  
62:P2:empty:  
63:P2:empty:  
64:P2:empty:  
65:P2:empty:  
66:P2:empty:  
67:P2:empty:  
68:P2:empty:  
69:P2:empty:  
70:P2:empty:  
71:P2:empty:  
Throughput: 0.111  
Average turnaround time: 27.250  
Average response time: 18.875

## RR

1:P1:empty:  
2:P1:empty:  
3:P1:P2(15),:  
4:P2:P1(7),:  
5:P1:P2(14),P3(8),:  
6:P2:P3(8),P1(6),:  
7:P3:P1(6),P2(13),P4(12),:  
8:P1:P2(13),P4(12),P3(7),:  
9:P2:P4(12),P3(7),P1(5),:  
10:P4:P3(7),P1(5),P2(12),P5(6),:  
11:P3:P1(5),P2(12),P5(6),P4(11),:  
12:P1:P2(12),P5(6),P4(11),P3(6),P6(4),:  
13:P2:P5(6),P4(11),P3(6),P6(4),P1(4),:  
14:P5:P4(11),P3(6),P6(4),P1(4),P2(11),P7(7),:  
15:P4:P3(6),P6(4),P1(4),P2(11),P7(7),P5(5),:  
16:P3:P6(4),P1(4),P2(11),P7(7),P5(5),P4(10),P8(9),:  
17:P6:P1(4),P2(11),P7(7),P5(5),P4(10),P8(9),P3(5),:  
18:P1:P2(11),P7(7),P5(5),P4(10),P8(9),P3(5),P6(3),:  
19:P2:P7(7),P5(5),P4(10),P8(9),P3(5),P6(3),P1(3),:  
20:P7:P5(5),P4(10),P8(9),P3(5),P6(3),P1(3),P2(10),:  
21:P5:P4(10),P8(9),P3(5),P6(3),P1(3),P2(10),P7(6),:  
22:P4:P8(9),P3(5),P6(3),P1(3),P2(10),P7(6),P5(4),:  
23:P8:P3(5),P6(3),P1(3),P2(10),P7(6),P5(4),P4(9),:  
24:P3:P6(3),P1(3),P2(10),P7(6),P5(4),P4(9),P8(8),:  
25:P6:P1(3),P2(10),P7(6),P5(4),P4(9),P8(8),P3(4),:  
26:P1:P2(10),P7(6),P5(4),P4(9),P8(8),P3(4),P6(2),:  
27:P2:P7(6),P5(4),P4(9),P8(8),P3(4),P6(2),P1(2),:  
28:P7:P5(4),P4(9),P8(8),P3(4),P6(2),P1(2),P2(9),:  
29:P5:P4(9),P8(8),P3(4),P6(2),P1(2),P2(9),P7(5),:  
30:P4:P8(8),P3(4),P6(2),P1(2),P2(9),P7(5),P5(3),:  
31:P8:P3(4),P6(2),P1(2),P2(9),P7(5),P5(3),P4(8),:



```
32:P3:P6(2),P1(2),P2(9),P7(5),P5(3),P4(8),P8(7),:
33:P6:P1(2),P2(9),P7(5),P5(3),P4(8),P8(7),P3(3),:
34:P1:P2(9),P7(5),P5(3),P4(8),P8(7),P3(3),P6(1),:
35:P2:P7(5),P5(3),P4(8),P8(7),P3(3),P6(1),P1(1),:
36:P7:P5(3),P4(8),P8(7),P3(3),P6(1),P1(1),P2(8),:
37:P5:P4(8),P8(7),P3(3),P6(1),P1(1),P2(8),P7(4),:
38:P4:P8(7),P3(3),P6(1),P1(1),P2(8),P7(4),P5(2),:
39:P8:P3(3),P6(1),P1(1),P2(8),P7(4),P5(2),P4(7),:
40:P3:P6(1),P1(1),P2(8),P7(4),P5(2),P4(7),P8(6),:
41:P6:P1(1),P2(8),P7(4),P5(2),P4(7),P8(6),P3(2),:
42:P2:P7(4),P5(2),P4(7),P8(6),P3(2),P1(1),:
43:P7:P5(2),P4(7),P8(6),P3(2),P1(1),P2(7),:
44:P5:P4(7),P8(6),P3(2),P1(1),P2(7),P7(3),:
45:P4:P8(6),P3(2),P1(1),P2(7),P7(3),P5(1),:
46:P8:P3(2),P1(1),P2(7),P7(3),P5(1),P4(6),:
47:P3:P1(1),P2(7),P7(3),P5(1),P4(6),P8(5),:
48:P1:P2(7),P7(3),P5(1),P4(6),P8(5),P3(1),:
49:P7:P5(1),P4(6),P8(5),P3(1),P2(7),:
50:P5:P4(6),P8(5),P3(1),P2(7),P7(2),:
51:P8:P3(1),P2(7),P7(2),P4(6),:
52:P3:P2(7),P7(2),P4(6),P8(4),:
53:P7:P4(6),P8(4),P2(7),:
54:P4:P8(4),P2(7),P7(1),:
55:P8:P2(7),P7(1),P4(5),:
56:P2:P7(1),P4(5),P8(3),:
57:P7:P4(5),P8(3),P2(6),:
58:P8:P2(6),P4(5),:
59:P2:P4(5),P8(2),:
60:P4:P8(2),P2(5),:
61:P8:P2(5),P4(4),:
62:P2:P4(4),P8(1),:
63:P4:P8(1),P2(4),:
64:P8:P2(4),P4(3),:
65:P4:P2(4),:
66:P2:P4(2),:
67:P4:P2(3),:
68:P2:P4(1),:
69:P4:P2(2),:
70:P2:empty:
71:P2:empty:
Throughput: 0.111
Average turnaround time: 49.000
Average response time: 4.500
```

## A.2 Program Code

```

1  #include<stdio.h>
2  #include<stdlib.h>
3  #include<string.h>
4  #include<stdbool.h>
5
6  // process control block (PCB)
7  struct pcb{                                // stores info on a process
8      unsigned int pid;                      // pid
9      char pname[20];                       // pname
10     unsigned int ptimeleft;                // time left to complete
11     unsigned int ptimearrival;             // time of arrival
12     bool isfirsttime;                     // flag to check if process has run for the
        first time
13     unsigned int pfirstruntime; // time at which process ran for the first
        time
14 }; typedef struct pcb pcb;
15
16 // queue node
17 struct dlq_node{                          // doubly linked queue node
18     struct dlq_node *pfwd; // pointer to next node
19     struct dlq_node *pbck; // pointer to previous node
20     struct pcb *data;       // pointer to data which is a pcb struct
21 }; typedef struct dlq_node dlq_node;
22
23 // queue
24 struct dlq{                               // doubly linked queue
25     struct dlq_node *head; // pointer to head of queue
26     struct dlq_node *tail; // pointer to tail of queue
27 }; typedef struct dlq dlq;
28
29 // function to add a pcb to a new queue node -> creates a new doubly
        linked queue node and initialiezs its data to ndata - a pointer to a
        pcb struct
30 dlq_node *get_new_node(pcb *ndata){ // ndata is a pointer to a pcb struct
31     if(!ndata) return NULL; // if ndata is null, return null
32
33     dlq_node *new = malloc(sizeof(dlq_node)); // allocate memory for a new
        node
34     if(!new){ // if new is null, print error and exit
35         fprintf(stderr, "Error: allocating memory\n");
36         exit(1);
37     }
38     // set the pointers to null and set the data to ndata
39     new->pfwd = new->pbck = NULL;
40     new->data = ndata;
41     return new;
42 }
43
44 // function to add a node to the tail of queue
45 void add_to_tail(dlq *q, dlq_node *new){
46     if(!new) return; // if new is null, return

```

```
47
48 if(q->head == NULL){ // if queue is empty, set head and tail to the new
    node
49     if(q->tail != NULL){ // if tail is not null, print error and exit
        since queue is inconsistent
50         fprintf(stderr, "DLList inconsitent.\n");
51         exit(1);
52     }
53     q->head = new; q->tail = q->head;
54 }
55 else{ // if queue is not empty, set the new node to the tail and set
    the tail to the new node
56     new->pfwd = q->tail;
57     new->pbck = NULL;
58     new->pfwd->pbck = new;
59     q->tail = new;
60 }
61 }
62
63 // function to remove a node from the head of queue
64 dlq_node *remove_from_head(dlq *const q){
65     if(q->head == NULL){ // if queue is empty, return null
66         if(q->tail != NULL){
67             fprintf(stderr, "DLList inconsitent.\n");
68             exit(1);
69         } // if tail is not null, print error and exit since queue is
            inconsistent
70         return NULL;
71     } else if(q->head == q->tail){ // if queue has only one node
72         if(q->head->pbck != NULL || q->tail->pfwd != NULL){ // if head's
            previous or tail's next is not null, print error and exit since queue
            is inconsistent
73             fprintf(stderr, "DLList inconsitent.\n");
74             exit(1);
75         }
76         // set head and tail to null and return the node
77         dlq_node *p = q->head;
78         q->head = NULL; q->tail = NULL;
79
80         p->pfwd = p->pbck = NULL;
81         return p;
82     } else{ // set the head to the next node and return the node
83         dlq_node *p = q->head;
84         q->head = q->head->pbck;
85         q->head->pfwd = NULL;
86
87         p->pfwd = p->pbck = NULL;
88         return p;
89     }
90 }
91
92 // function to print our queue
93 void print_q(const dlq *q){
```

```
94     dlq_node *n = q->head;
95     if(n == NULL) return;
96     while(n){
97         printf("%s(%d)", n->data->pname, n->data->ptimeleft);
98         n = n->pbck;
99     }
100 }
101
102 // function to check if the queue is empty
103 int is_empty(const dlq *q){
104     if (q->head == NULL && q->tail == NULL) return 1;
105     else if (q->head != NULL && q->tail != NULL) return 0;
106     else{
107         fprintf(stderr, "Error: DLL queue is inconsistent.");
108         exit(1);
109     }
110 }
111
112 // function to sort the queue on completion time
113 void sort_by_timetocompletion(const dlq *q){
114     // bubble sort
115     dlq_node *start = q->tail;
116     dlq_node *end = q->head;
117
118     while(start != end){ // while start and end are not equal
119         dlq_node *node = start; // set node to start
120         dlq_node *next = node->pfwd; // set next to node's forward
121
122         while(next != NULL){ // while next is not null
123             if (node->data->ptimeleft < next->data->ptimeleft){ // if node's
124                 time left is less than next's time left, do a swap
125                 pcb *temp = node->data;
126                 node->data = next->data;
127                 next->data = temp;
128             }
129             node = next; // set node to next and next to node's forward
130             next = node->pfwd;
131         }
132         end = end->pbck; // set end to end's backward
133     }
134 }
135
136 // function to sort the queue on arrival time
137 void sort_by_arrival_time(const dlq *q){
138     // bubble sort
139     dlq_node *start = q->tail;
140     dlq_node *end = q->head;
141
142     while(start != end){
143         dlq_node *node = start;
144         dlq_node *next = node->pfwd;
145
146         while(next != NULL){
```

```
146     if (node->data->ptimearrival < next->data->ptimearrival){
147         // do a swap
148         pcb *temp = node->data;
149         node->data = next->data;
150         next->data = temp;
151     }
152     node = next;
153     next = node->pfwd;
154 }
155 end = end->pbck;
156 }
157 }
158
159 // function to tokenize the one row of data -> parses a line of input
160 // data and returns a pointer to a pcb struct from it
161 pcb *tokenize_pdata(char *buf){ // buf is a pointer to the line of input
162 // data containing the process data in the format pname:pid:duration:
163 // arrival time
164 pcb *p = (pcb *)malloc(sizeof(pcb)); // allocate memory for a pcb
165 struct
166 if(!p){
167     fprintf(stderr, "Error: allocating memory.\n");
168     exit(1);
169 }
170
171 char *token = strtok(buf, ":\n"); // tokenize the line of input data
172 if(!token){
173     fprintf(stderr, "Error: Expecting token pname\n");
174     exit(1);
175 }
176 strcpy(p->pname, token); // copy the token to pname
177
178 token = strtok(NULL, ":\n"); // tokenize the line of input data
179 if(!token){
180     fprintf(stderr, "Error: Expecting token pid\n");
181     exit(1);
182 }
183 p->pid = atoi(token); // convert the token to an integer and set it to
184 pid
185
186 token = strtok(NULL, ":\n");
187 if(!token){
188     fprintf(stderr, "Error: Expecting token duration\n");
189     exit(1);
190 }
191 p->ptimeleft = atoi(token); // convert the token to an integer and set
192 it to ptimeleft
193
194 token = strtok(NULL, ":\n");
195 if(!token){
196     fprintf(stderr, "Error: Expecting token arrival time\n");
197     exit(1);
198 }
199 }
```

```
193 p->ptimearrival = atoi(token); // convert the token to an integer and
    set it to ptimearrival
194
195 p->isfirsttime = true; // set isfirsttime to true
196
197 token = strtok(NULL, ":\n");
198 if(token){
199     fprintf(stderr, "Error: Oh, what've you got at the end of the line?\n
    ");
200     exit(1);
201 }
202 return p;
203 }
204
205 // implement the FIFO scheduling code
206 void sched_FIFO(dlq *const p_fq, int *p_time){
207     // initialize a queue to manage processes that are ready to run
208     dlq queue; queue.head = queue.tail = NULL;
209     // initialize the first process node from the head of the queue
210     dlq_node *process = remove_from_head(p_fq);
211
212     // initialize performance metrics
213     int num_processes = 1, rt1 = 0, response_time = 0, turnaround_time = 0,
        first_arrival = process->data->ptimearrival;
214     while(1){
215         (*p_time)++; // increment the system time
216         // if there are no processes left to run and the queue and ready
        queue is empty, break the loop
217         if(!(process->data->ptimeleft) && is_empty(&queue) && is_empty(p_fq))
            break;
218
219         // if there are processes left to run and the head of the queue has
        an arrival time less than the system time, add the head of the queue
        to the ready queue
220         if(!is_empty(p_fq) && p_fq->head->data->ptimearrival < (*p_time)){
221             add_to_tail(&queue, remove_from_head(p_fq)); num_processes++;
222         }
223
224         if(process->data->isfirsttime == true){ // if the process has not run
        before, set its first runtime to the system time and set isfirsttime
        to false
225             process->data->isfirsttime = false;
226             process->data->pfirstruntime = (*p_time);
227             rt1 = (*p_time) - process->data->ptimearrival;
228             if (rt1 < 0) rt1 = 1;
229             response_time += rt1;
230         }
231
232         printf("%d:", (*p_time));
233         // if the process still has to arrive, print idle, else decrement its
        time and print its name
234         if(process->data->ptimearrival >= (*p_time)) printf("idle:");
```

```

235     else{ process->data->ptimeleft--; printf("%s:", process->data->pname)
; }
236     // if the queue is empty, print empty, else show contents of the
queue
237     if(is_empty(&queue)) printf("empty:\n");
238     else{ print_q(&queue); printf(":\n");}
239
240     // if process has finished, calculate the turnaround time and add it
to the total turnaround time, then remove another process from the
queue if queue is not empty
241     if(process->data->ptimeleft == 0){
242         turnaround_time += (*p_time) - process->data->ptimearrival;
243         if (!is_empty(&queue)) process = remove_from_head(&queue);
244     }
245     free(process);
246     float throughput = (float)num_processes / ((*p_time) - first_arrival);
247     float avg_turnaround_time = (float)turnaround_time / num_processes;
248     float avg_response_time = (float)response_time / num_processes;
249     printf("Throughput: %.3f\n", throughput);
250     printf("Average turnaround time: %.3f\n", avg_turnaround_time);
251     printf("Average response time: %.3f\n", avg_response_time);
252     return;
253 }
254
255 // implement the SJF scheduling code
256 void sched_SJF(dlq *const p_fq, int *p_time){
257     dlq queue; queue.head = queue.tail = NULL;
258     dlq_node *process = remove_from_head(p_fq);
259
260     int num_processes = 1, rt1 = 0, response_time = 0, turnaround_time = 0,
first_arrival = process->data->ptimearrival;
261
262     while(1){
263         (*p_time)++;
264
265         if(!(process->data->ptimeleft) && is_empty(&queue) && is_empty(p_fq))
break;
266         // if processes left with arrival time less than system time, add
them to the tail of the queue, and sort by time to completion
267         if(!is_empty(p_fq) && p_fq->head->data->ptimearrival < (*p_time)){
268             add_to_tail(&queue, remove_from_head(p_fq));
269             num_processes++;
270             sort_by_timetocompletion(&queue);
271         }
272
273         if(process->data->isfirsttime == true){
274             process->data->isfirsttime = false;
275             process->data->pfirstruntime = (*p_time);
276             rt1 = (*p_time) - process->data->ptimearrival;
277             if (rt1 < 0) rt1 = 1;
278             response_time += rt1;
279         }
280

```

```

281     printf("%d:", *p_time);
282     // if process still has to arrive, print idle, else decrement its
time and print its name
283     if(process->data->ptimearrival >= (*p_time)) printf("idle:");
284     else{
285         process->data->ptimeleft--;
286         printf("%s:", process->data->pname);
287     }
288     // if queue is empty, print empty, else show contents of the queue
289     if(is_empty(&queue)) printf("empty:\n");
290     else{
291         print_q(&queue);
292         printf(":\n");
293     }
294
295     if(process->data->ptimeleft == 0){
296         turnaround_time += (*p_time) - process->data->ptimearrival;
297         if(!is_empty(&queue)) process = remove_from_head(&queue);
298     }
299 }
300 free(process);
301 float throughput = (float)num_processes / ((*p_time) - first_arrival);
302 float avg_turnaround_time = (float)turnaround_time / num_processes;
303 float avg_response_time = (float)response_time / num_processes;
304 printf("Throughput: %.3f\n", throughput);
305 printf("Average turnaround time: %.3f\n", avg_turnaround_time);
306 printf("Average response time: %.3f\n", avg_response_time);
307 return;
308 }
309
310 // implement the RR scheduling code
311 void sched_RR(dlq *const p_fq, int *p_time){
312     dlq queue; queue.head = queue.tail = NULL;
313     dlq_node *process = remove_from_head(p_fq);
314     int quantum_time = 1, process_time = 0; // quantum is the time slice
for each process, process_time is the time the process has been
running
315
316     int num_processes = 1, rtl = 0, response_time = 0, turnaround_time = 0,
first_arrival = process->data->ptimearrival;
317
318     while(1){
319         (*p_time)++;
320         if (!(process->data->ptimeleft) && is_empty(&queue) && is_empty(p_fq)
) break;
321
322         if(!is_empty(p_fq) && p_fq->head->data->ptimearrival < (*p_time)){
323             add_to_tail(&queue, remove_from_head(p_fq));
324             num_processes++;
325         }
326
327         if(process->data->isfirsttime == true){
328             process->data->isfirsttime = false;

```



```
329     process->data->pfirstruntime = (*p_time);
330     rt1 = (*p_time) - process->data->ptimearrival;
331     if (rt1 < 0) rt1 = 1;
332     response_time += rt1;
333 }
334
335 printf("%d:", *p_time);
336 if(process->data->ptimearrival >= (*p_time)) printf("idle:");
337 else{ // if process has arrived, decrement its time, update the time
338     it has run for, and print its name
339     process->data->ptimeleft--;
340     process_time++;
341     printf("%s:", process->data->pname);
342 }
343
344 if(is_empty(&queue)) printf("empty:\n");
345 else{
346     print_q(&queue);
347     printf(":\n");
348 }
349
350 if(process->data->ptimeleft == 0){
351     turnaround_time += (*p_time) - process->data->ptimearrival;
352     if (!is_empty(&queue)) process = remove_from_head(&queue);
353 }
354
355 if(process_time == quantum_time){ // if process has run for the time
356     slice, add it to the tail of the queue and set process_time to 0
357     add_to_tail(&queue, process);
358     process = remove_from_head(&queue);
359     process_time = 0;
360 }
361
362 free(process);
363 float throughput = (float)num_processes / ((*p_time) - first_arrival);
364 float avg_turnaround_time = (float)turnaround_time / num_processes;
365 float avg_response_time = (float)response_time / num_processes;
366 printf("Throughput: %.3f\n", throughput);
367 printf("Average turnaround time: %.3f\n", avg_turnaround_time);
368 printf("Average response time: %.3f\n", avg_response_time);
369 return;
370 }
371
372 // implement the STCF scheduling code
373 void sched_STCF(dlq *const p_fq, int *p_time){
374     dlq queue; queue.head = queue.tail = NULL;
375     dlq_node *process = remove_from_head(p_fq);
376
377     int num_processes = 1, rt1 = 0, response_time = 0, turnaround_time = 0,
378         first_arrival = process->data->ptimearrival;
379
380     while(1){
381         (*p_time)++;
```

```
379     if(!(process->data->ptimeleft) && is_empty(&queue) && is_empty(p_fq))
380         break;
381
382     if(!is_empty(p_fq) && p_fq->head->data->ptimearrival < (*p_time)){
383         add_to_tail(&queue, remove_from_head(p_fq));
384         num_processes++;
385     }
386
387     if(process->data->isfirsttime == true){
388         process->data->isfirsttime = false;
389         process->data->pfirstruntime = (*p_time);
390         rt1 = (*p_time) - process->data->ptimearrival;
391         if (rt1 < 0) rt1 = 1;
392         response_time += rt1;
393     }
394
395     sort_by_timetocompletion(&queue); // sort the queue by time to
396     completion
397
398     printf("%d:", *p_time);
399     if(process->data->ptimearrival >= (*p_time)) printf("idle:");
400     else{
401         if(!is_empty(&queue) && (process->data->ptimeleft > queue.head->
402         data->ptimeleft)){
403             add_to_tail(&queue, process);
404             process = remove_from_head(&queue);
405         }
406         if(process->data->isfirsttime == false){
407             process->data->isfirsttime = true;
408             process->data->pfirstruntime = (*p_time);
409             rt1 = (*p_time) - process->data->ptimearrival;
410             if (rt1 < 0) rt1 = 1;
411             response_time += rt1;
412         }
413         process->data->ptimeleft--;
414         printf("%s:", process->data->pname);
415     }
416
417     if(is_empty(&queue)) printf("empty:\n");
418     else{
419         sort_by_timetocompletion(&queue);
420         print_q(&queue);
421         printf(":\n");
422     }
423
424     if(process->data->ptimeleft == 0){
425         turnaround_time += (*p_time) - process->data->ptimearrival;
426         if(!is_empty(&queue)) process = remove_from_head(&queue);
427     }
428
429     free(process);
430     float throughput = (float)num_processes / ((*p_time) - first_arrival);
431     float avg_turnaround_time = (float)turnaround_time / num_processes;
```

```
429 float avg_response_time = (float)response_time / num_processes;
430 printf("Throughput: %.3f\n", throughput);
431 printf("Average turnaround time: %.3f\n", avg_turnaround_time);
432 printf("Average response time: %.3f\n", avg_response_time);
433 return;
434 }
435
436 int main(){
437     /* Enter your code here. Read input from STDIN. Print output to STDOUT
438     */
439     int N = 0; // number of processes
440     char tech[20] = {'\0'}; // scheduling policy
441     char buffer[100] = {'\0'}; // buffer to store the input data
442     scanf("%d", &N); // read the number of processes
443     scanf("%s", tech); // read the scheduling policy
444
445     dlq queue; // create a queue
446     queue.head = NULL; // set the head and tail to null
447     queue.tail = NULL;
448     for (int i = 0; i < N; ++i){ // for each process, read the data,
449         tokenize it, and add it to the queue
450         scanf("%s\n", buffer);
451         pcb *p = tokenize_pdata(buffer);
452         add_to_tail(&queue, get_new_node(p));
453     }
454
455     unsigned int system_time = 0;
456     sort_by_arrival_time(&queue);
457
458     // run scheduler
459     if (!strcmp(tech, "FIFO", 4)) sched_FIFO(&queue, &system_time);
460     else if (!strcmp(tech, "SJF", 3)) sched_SJF(&queue, &system_time);
461     else if (!strcmp(tech, "STCF", 4)) sched_STCF(&queue, &system_time);
462     else if (!strcmp(tech, "RR", 2)) sched_RR(&queue, &system_time);
463     else fprintf(stderr, "Error: unknown POLICY\n");
464     return 0;
465 }
```