

Team 3: Jack Lin, Kolbe Williams, Jared Ricks

Operating System Project

Team Member 1 Tasks (FCFS Algorithm on schedule_arrive_zero.txt)

Q1. Demonstrate FCFS execution:

- Update the code to implement the FCFS algorithm for task scheduling.

```
1 #include <stdlib.h>
2 #include <stdio.h>
3
4 #include "task.h"
5 #include "list.h"
6 #include "cpu.h"
7
8 // Reference to the head of the list
9 struct node *head = NULL;
10
11 // Sequence counter of next available thread identifier
12 int nextTid = 0;
13
14 Task *selectNextTask();
15
16
17 // Add a new task to the list of tasks
18 void add(char *name, int arrivalTime, int burst) {
19     // First, create the new task
20     Task *newTask = (Task *) malloc(sizeof(Task));
21
22     newTask->name = name;
23     newTask->tid = nextTid++;
24     newTask->arrivalTime = arrivalTime;
25     newTask->burst = burst;
26
27     // Insert the new task into the list of tasks
28     insert(&head, newTask);
29 }
30
31 // Function to reverse the linked list
32 void reverseList() {
33     struct node *prev = NULL;
34     struct node *current = head;
35     struct node *next = NULL;
36
37     while (current != NULL) {
38         next = current->next;
39         current->next = prev;
40         prev = current;
```

```

40     prev = current;
41     current = next;
42 }
43 head = prev;
44 }
45
46 /**
47  * Run the FCFS scheduler
48  */
49 void schedule() {
50     Task *current;
51
52     traverse(head);
53     // Reverse the list before scheduling
54     reverseList();
55
56     while (head != NULL) {
57         current = selectNextTask();
58
59         run(current, current->burst);
60         // Delete the task from the list
61         delete(&head, current);
62     }
63 }
64
65
66
67 /**
68  * Returns the next task selected to run.
69  */
70 Task *selectNextTask() {
71     // Find the task with the smallest arrival time (i.e., the first task in the list)
72     struct node *temp = head;
73     struct node *returnNode = head;
74
75     while (temp != NULL) {
76         if (temp->task->arrivalTime < returnNode->task->arrivalTime) {
77             returnNode = temp;
78         }
79         temp = temp->next;
80     }
81
82     return returnNode->task;
83 }

```

void reverseList() { // Function to reverse the linked list

 struct node *prev = NULL; // Pointer to previous node, initialized to NULL

 struct node *current = head; // Pointer to current node, initialized to the head of the list

 struct node *next = NULL; // Pointer to next node, initialized to NULL

 while (current != NULL) { // Loop until current node is NULL (end of the list)

 next = current->next; // Store the next node

 current->next = prev; // Reverse the link of current node to point to the previous node

 prev = current; // Move prev to current node

 current = next; // Move current to next node

```

    }
    head = prev; // Update the head of the list to point to the last node (which was the first node
before reversal)
}

Task *selectNextTask() { // Function to select the next task with the smallest arrival time
    // Find the task with the smallest arrival time (i.e., the first task in the list)
    struct node *temp = head; // Pointer to traverse the list, initialized to the head of the list
    struct node *returnNode = head; // Pointer to store the node with the smallest arrival time,
initialized to the head of the list

    while (temp != NULL) { // Loop until temp reaches the end of the list
        if (temp->task->arrivalTime < returnNode->task->arrivalTime) { // Check if the arrival
time of the current task is smaller than the arrival time of the stored node
            returnNode = temp; // If true, update the returnNode pointer to point to the current node
        }
        temp = temp->next; // Move to the next node in the list
    }

    return returnNode->task; // Return the task with the smallest arrival time
}

```

- Compile the program using the provided makefile (make fcfs).

```

labuser1@ML-RefVm-535928:~/Desktop/Project05$ make fcfs
gcc -Wall -c schedule_fcfs.c
gcc -Wall -o fcfs driver.o schedule_fcfs.o list.o CPU.o

```

- Run the program with schedule_arrive_zero.txt input file.

```

labuser1@ML-RefVm-535928:~/Desktop/Project05$ ./fcfs schedule_arrive_zero.txt
[T1] [0] [20]
[T2] [0] [25]
[T3] [0] [25]
[T4] [0] [15]
[T5] [0] [20]
[T6] [0] [10]
[T7] [0] [30]
[T8] [0] [25]
Running task = [T1] [0] [20] for 20 units.
Running task = [T2] [0] [25] for 25 units.
Running task = [T3] [0] [25] for 25 units.
Running task = [T4] [0] [15] for 15 units.
Running task = [T5] [0] [20] for 20 units.
Running task = [T6] [0] [10] for 10 units.
Running task = [T7] [0] [30] for 30 units.
Running task = [T8] [0] [25] for 25 units.
labuser1@ML-RefVm-535928:~/Desktop/Project05$ █

```

- Verify the execution order matches the FCFS criteria.

Yes, the processes run in alphabetical order since they all arrive at the same time.

Q2. Print Process Metrics:

- Modify code to calculate and print turnaround time, waiting time, response time, and response ratio for each process. (Click here: [Formula Sheet Actions](#))

```

1 #include <stdlib.h>
2 #include <stdio.h>
3
4 #include "task.h"
5 #include "list.h"
6 #include "cpu.h"
7
8 // Reference to the head of the list
9 struct node *head = NULL;
10
11 // Sequence counter of next available thread identifier
12 int nextTid = 0;
13
14 Task *selectNextTask();
15
16 // Global variables to store statistics
17 float totalTurnaroundTime = 0;
18 float totalWaitingTime = 0;
19 int totalTasks = 0;
20
21 /*added
22 Task *arr[8];
23 int arr2[8];*/
24
25 // Add a new task to the list of tasks
26 void add(char *name, int arrivalTime, int burst) {
27     // First, create the new task
28     Task *newTask = (Task *) malloc(sizeof(Task));
29
30     newTask->name = name;
31     newTask->tid = nextTid++;
32     newTask->arrivalTime = arrivalTime;
33     newTask->burst = burst;
34
35     // Insert the new task into the list of tasks
36     insert(&head, newTask);
37 }
38
39 // Function to reverse the linked list
40 void reverseList() {
41     struct node *prev = NULL;
42     struct node *current = head;
43     struct node *next = NULL;
44
45     while (current != NULL) {
46         next = current->next;
47         current->next = prev;

```

```

48     prev = current;
49     current = next;
50 }
51 head = prev;
52 }
53
54 /**
55  * Run the FCFS scheduler
56  */
57 void schedule() {
58     Task *current;
59
60     traverse(head);
61     // Reverse the list before scheduling
62     reverseList();
63
64     // Start system time
65     int systemTime = 0;
66
67     printf("-----\n");
68     printf("| %-10s | %-10s | %-10s | %-10s | %-12s | %-12s | %-12s |\n", "Task Name", "Arrival", "Burst", "Turnaround", "Waiting", "Response", "Response Ratio");
69     printf("-----\n");
70
71     while (head != NULL) {
72         current = selectNextTask();
73
74         //run(current,current->burst);
75         /*added
76         int i = 0;
77         arr[i] = current;
78         arr2[i] = current->burst;
79         i++;*/
80
81         // Update waiting time for all tasks waiting in the queue
82         totalWaitingTime += systemTime - current->arrivalTime;
83
84         // Calculate response time
85         int responseTime = systemTime - current->arrivalTime;
86
87         // Calculate turnaround time
88         int turnaroundTime = responseTime + current->burst;
89
90         // Calculate response ratio
91         float responseRatio = (float) turnaroundTime / current->burst;
92

```

```

93     // Print task statistics
94     printf("| %-10s | %-10d | %-10d | %-10d | %-12d | %-12d | %-12.2f | \n", current->name, current->arrivalTime, current->burst, turnaroundTime, systemTime - current->arrivalTime, responseTime,
responseRatio);
95
96     // Update system time
97     systemTime += current->burst;
98
99     // Update total turnaround time
100     totalTurnaroundTime += turnaroundTime;
101
102     // Delete the task from the list
103     delete(&head, current);
104
105     // Increment total tasks
106     totalTasks++;
107 }
108
109 printf("-----\n");
110
111 // Calculate and print statistics
112 float avgTurnaroundTime = totalTurnaroundTime / totalTasks;
113 float avgWaitingTime = totalWaitingTime / totalTasks;
114 float throughput = (float) systemTime / totalTasks;
115
116 printf("| %-30s | %-30s | %-30s |\n", "Average Turnaround Time", "Average Waiting Time", "Throughput");
117 printf("| %-30.2f | %-30.2f | %-30.2f |\n", avgTurnaroundTime, avgWaitingTime, throughput);
118 printf("-----\n");
119
120 /*added
121 for(int i = 0; i < 8; i++)
122 {
123     run(arr[i],arr2[i]);
124 }*/
125 }
126
127 /**
128  * Returns the next task selected to run.
129  */
130 Task *selectNextTask() {
131     // Find the task with the smallest arrival time (i.e., the first task in the list)
132     struct node *temp = head;
133     struct node *returnNode = head;
134

```

```

135     while (temp != NULL) {
136         if (temp->task->arrivalTime < returnNode->task->arrivalTime) {
137             returnNode = temp;
138         }
139         temp = temp->next;
140     }
141

```

- Format output in a well-organized table format.

```
labuser1@ML-RefVm-535928:~/Desktop/Project05$ make fcfs
gcc -Wall -o fcfs driver.o schedule_fcfs.o list.o CPU.o
labuser1@ML-RefVm-535928:~/Desktop/Project05$ ./fcfs schedule_arrive_zero.txt
[T1] [0] [20]
[T2] [0] [25]
[T3] [0] [25]
[T4] [0] [15]
[T5] [0] [20]
[T6] [0] [10]
[T7] [0] [30]
[T8] [0] [25]
```

Task Name	Arrival	Burst	Turnaround	Waiting	Response	Response Ratio
T1	0	20	20	0	0	1.00
T2	0	25	45	20	20	1.80
T3	0	25	70	45	45	2.80
T4	0	15	85	70	70	5.67
T5	0	20	105	85	85	5.25
T6	0	10	115	105	105	11.50
T7	0	30	145	115	115	4.83
T8	0	25	170	145	145	6.80
Average Turnaround Time			Average Waiting Time		Throughput	
94.38			73.12		21.25	

```
labuser1@ML-RefVm-535928:~/Desktop/Project05$
```

Q3. Print Average Metrics:

- Extend code to calculate and print average turnaround time, average waiting time, and throughput.

```
111 // Calculate and print statistics
112 float avgTurnaroundTime = totalTurnaroundTime / totalTasks;
113 float avgWaitingTime = totalWaitingTime / totalTasks;
114 float throughput = (float) systemTime / totalTasks;
115
116 printf("| %-30s | %-30s | %-30s |\n", "Average Turnaround Time", "Average Waiting Time", "Throughput");
117 printf("| %-30.2f | %-30.2f | %-30.2f |\n", avgTurnaroundTime, avgWaitingTime, throughput);
118 printf("-----\n");
119
```

We put this in the last part too.

- Ensure output is presented clearly in a tabular format.

```
labuser1@ML-RefVm-535928:~/Desktop/Project05$ make fcfs
gcc -Wall -o fcfs driver.o schedule_fcfs.o list.o CPU.o
labuser1@ML-RefVm-535928:~/Desktop/Project05$ ./fcfs schedule_arrive_zero.txt
[T1] [0] [20]
[T2] [0] [25]
[T3] [0] [25]
[T4] [0] [15]
[T5] [0] [20]
[T6] [0] [10]
[T7] [0] [30]
[T8] [0] [25]
```

Task Name	Arrival	Burst	Turnaround	Waiting	Response	Response Ratio
T1	0	20	20	0	0	1.00
T2	0	25	45	20	20	1.80
T3	0	25	70	45	45	2.80
T4	0	15	85	70	70	5.67
T5	0	20	105	85	85	5.25
T6	0	10	115	105	105	11.50
T7	0	30	145	115	115	4.83
T8	0	25	170	145	145	6.80
Average Turnaround Time			Average Waiting Time		Throughput	
94.38			73.12		21.25	

```
labuser1@ML-RefVm-535928:~/Desktop/Project05$ █
```

Team Member 2 Tasks (FCFS Algorithm on schedule_arrive_diff.txt)

Q4. Implement FCFS for Different Arrival Times:

- Duplicate and modify the code for FCFS scheduling to handle the tasks with different arrival times.

```
1 #include <stdlib.h>
2 #include <stdio.h>
3
4 #include "task.h"
5 #include "list.h"
6 #include "cpu.h"
7
8 // Reference to the head of the list
9 struct node *head = NULL;
10
11 // Sequence counter of next available thread identifier
12 int nextTid = 0;
13
14 Task *selectNextTask();
15
16 // Global variables to store statistics
17 float totalTurnaroundTime = 0;
18 float totalWaitingTime = 0;
19 int firstResponseTime = 0;
20 int completionTime = 0;
21 int totalTasks = 0;
22 int totalBurst = 0;
23
24 // Add a new task to the list of tasks
25 void add(char *name, int arrivalTime, int burst) {
26     // First, create the new task
27     Task *newTask = (Task *) malloc(sizeof(Task));
28
29     newTask->name = name;
30     newTask->tid = nextTid++;
31     newTask->arrivalTime = arrivalTime;
32     newTask->burst = burst;
33
34     // Insert the new task into the list of tasks
35     insert(&head, newTask);
36 }
37
38 // Function to reverse the linked list
39 void reverseList() {
40     struct node *prev = NULL;
41     struct node *current = head;
42     struct node *next = NULL;
43 }
```



```

43
44     while (current != NULL) {
45         next = current->next;
46         current->next = prev;
47         prev = current;
48         current = next;
49     }
50     head = prev;
51 }
52
53 /**
54  * Run the FCFS scheduler
55  */
56 void schedule() {
57     Task *current;
58
59     // Reverse the list before scheduling
60     reverseList();
61     traverse(head);
62     //reverseList();
63
64     /*printf("-----\n");
65     printf("| %-10s | %-10s | %-10s | %-10s | %-12s | %-12s | %-12s | \n", "Task Name", "Arrival", "Burst", "Turnaround", "Waiting", "Response", "Response Ratio");
66     printf("-----\n");*/
67
68     while (head != NULL) {
69         current = selectNextTask();
70
71         run(current, current->burst);
72
73         if (firstResponseTime < current->arrivalTime)
74         {
75             firstResponseTime = current->arrivalTime;
76         }
77
78         completionTime = firstResponseTime + current->burst;
79
80         // Calculate response time
81         int responseTime = firstResponseTime - current->arrivalTime;
82
83         // Calculate turnaround time
84         int turnaroundTime = completionTime - current->arrivalTime;
85

```

```

86         // Update waiting time for all tasks waiting in the queue
87         int waitingTime = turnaroundTime - current->burst;
88
89         // Calculate response ratio
90         float responseRatio = (float) (waitingTime + current->burst) / current->burst;
91
92         /*// Print task statistics
93         printf("| %-10s | %-10d | %-10d | %-10d | %-12d | %-12d | %-12.2f | \n", current->name, current->arrivalTime, current->burst, turnaroundTime, waitingTime, responseTime,
94         responseRatio);*/
95
96         //Update the first response time
97         firstResponseTime += current->burst;
98
99         // Update total turnaround time
100         totalTurnaroundTime += turnaroundTime;
101
102         totalWaitingTime += waitingTime;
103
104         totalBurst += current->burst;
105
106         // Delete the task from the list
107         delete(&head, current);
108
109         // Increment total tasks
110         totalTasks++;
111     }
112
113     /*printf("-----\n");
114
115     // Calculate and print statistics
116     float avgTurnaroundTime = totalTurnaroundTime / totalTasks;
117     float avgWaitingTime = totalWaitingTime / totalTasks;
118     float throughput = (float) totalBurst / totalTasks;
119
120     printf("| %-30s | %-30s | %-30s | \n", "Average Turnaround Time", "Average Waiting Time", "Throughput");
121     printf("| %-30.2f | %-30.2f | %-30.2f | \n", avgTurnaroundTime, avgWaitingTime, throughput);
122     printf("-----\n");*/
123 }
124

```

```

125 /**
126  * Returns the next task selected to run.
127  */
128 Task *selectNextTask() {
129     // Find the task with the smallest arrival time (i.e., the first task in the list)
130     struct node *temp = head;
131     struct node *returnNode = head;
132
133     while (temp != NULL) {
134         if (temp->task->arrivalTime < returnNode->task->arrivalTime) {
135             returnNode = temp;
136         }
137         temp = temp->next;
138     }
139
140     return returnNode->task;
141 }

```

- Create a new file scheduler_fcfs_diff_arrival.c if necessary, and update the makefile accordingly.

```
# makefile for scheduling program
#

# make fcfs - for FCFS scheduling

CC=gcc
CFLAGS=-Wall

clean:
    rm -rf *.o
    rm -rf fcfs

fcfs: driver.o list.o CPU.o schedule_fcfs.o scheduler_fcfs_diff_arrival.o
    $(CC) $(CFLAGS) -o fcfs driver.o schedule_fcfs.o list.o CPU.o

fcfs2: driver.o list.o CPU.o scheduler_fcfs_diff_arrival.o
    $(CC) $(CFLAGS) -o fcfs2 driver.o scheduler_fcfs_diff_arrival.o list.o CPU.o

sjf: driver.o list.o CPU.o schedule_sjf.o
    $(CC) $(CFLAGS) -o sjf driver.o schedule_sjf.o list.o CPU.o

scheduler_fcfs_diff_arrival.o: scheduler_fcfs_diff_arrival.c
    $(CC) $(CFLAGS) -c scheduler_fcfs_diff_arrival.c

schedule_fcfs.o: schedule_fcfs.c
    $(CC) $(CFLAGS) -c schedule_fcfs.c

driver.o: driver.c
    $(CC) $(CFLAGS) -c driver.c

list.o: list.c list.h
    $(CC) $(CFLAGS) -c list.c

CPU.o: CPU.c cpu.h
    $(CC) $(CFLAGS) -c CPU.c
```

- Compile and run the program with schedule_arrive_diff.txt input file.

```
labuser1@ML-RefVm-535928:~/Desktop/Project05$ make fcfs2
gcc -Wall -o fcfs2 driver.o scheduler_fcfs_diff_arrival.o list.o CPU.o
labuser1@ML-RefVm-535928:~/Desktop/Project05$ ./fcfs2 fcfs_schedule_arrive_diff.txt
[T1] [2] [20]
[T3] [3] [25]
[T4] [5] [15]
[T5] [5] [20]
[T8] [200] [25]
[T6] [12] [10]
[T7] [100] [30]
[T2] [3] [25]
Running task = [T1] [2] [20] for 20 units.
Running task = [T3] [3] [25] for 25 units.
Running task = [T2] [3] [25] for 25 units.
Running task = [T4] [5] [15] for 15 units.
Running task = [T5] [5] [20] for 20 units.
Running task = [T6] [12] [10] for 10 units.
Running task = [T7] [100] [30] for 30 units.
Running task = [T8] [200] [25] for 25 units.
labuser1@ML-RefVm-535928:~/Desktop/Project05$
```

- Verify if any adjustments are needed for correct task scheduling.

All of the tasks are running according to their arrival time.

- Do the same as **Q2** and **Q3**.

```

1 #include <stdlib.h>
2 #include <stdio.h>
3
4 #include "task.h"
5 #include "list.h"
6 #include "cpu.h"
7
8 // Reference to the head of the list
9 struct node *head = NULL;
10
11 // Sequence counter of next available thread identifier
12 int nextTid = 0;
13
14 Task *selectNextTask();
15
16 // Global variables to store statistics
17 float totalTurnaroundTime = 0;
18 float totalWaitingTime = 0;
19 int firstResponseTime = 0;
20 int completionTime = 0;
21 int totalTasks = 0;
22 int totalBurst = 0;
23
24 // Add a new task to the list of tasks
25 void add(char *name, int arrivalTime, int burst) {
26     // First, create the new task
27     Task *newTask = (Task *) malloc(sizeof(Task));
28
29     newTask->name = name;
30     newTask->tid = nextTid++;
31     newTask->arrivalTime = arrivalTime;
32     newTask->burst = burst;
33
34     // Insert the new task into the list of tasks
35     insert(&head, newTask);
36 }
37
38 // Function to reverse the linked list
39 void reverseList() {
40     struct node *prev = NULL;
41     struct node *current = head;
42     struct node *next = NULL;
43

```

```

44 while (current != NULL) {
45     next = current->next;
46     current->next = prev;
47     prev = current;
48     current = next;
49 }
50 head = prev;
51 }
52
53 /**
54  * Run the FCFS scheduler
55  */
56 void schedule() {
57     Task *current;
58
59     // Reverse the list before scheduling
60     reverseList();
61     traverse(head);
62     //reverseList();
63
64     printf("-----\n");
65     printf("| %-10s | %-10s | %-10s | %-10s | %-12s | %-12s | %-12s |\n", "Task Name", "Arrival", "Burst", "Turnaround", "Waiting", "Response", "Response Ratio");
66     printf("-----\n");
67
68     while (head != NULL) {
69         current = selectNextTask();
70
71         //run(current,current->burst);
72
73         if(firstResponseTime < current->arrivalTime)
74         {
75             firstResponseTime = current->arrivalTime;
76         }
77
78         completionTime = firstResponseTime + current->burst;
79
80         // Calculate response time
81         int responseTime = firstResponseTime - current->arrivalTime;
82
83         // Calculate turnaround time
84         int turnaroundTime = completionTime - current->arrivalTime;
85
86         // Update waiting time for all tasks waiting in the queue
87         int waitingTime = turnaroundTime - current->burst;
88
89         // Calculate response ratio
90         float responseRatio = (float) (waitingTime + current->burst) / current->burst;
91
92         // Print task statistics
93         printf("| %-10s | %-10d | %-10d | %-10d | %-12d | %-12d | %-12.2f |\n", current->name, current->arrivalTime, current->burst, turnaroundTime, waitingTime, responseTime,
responseRatio);
94
95         //Update the first response time
96         firstResponseTime += current->burst;
97
98         // Update total turnaround time
99         totalTurnaroundTime += turnaroundTime;
100
101         totalWaitingTime += waitingTime;
102
103         totalBurst += current->burst;
104
105         // Delete the task from the list
106         delete(head, current);
107
108         // Increment total tasks
109         totalTasks++;
110     }
111
112     printf("-----\n");
113
114     // Calculate and print statistics
115     float avgTurnaroundTime = totalTurnaroundTime / totalTasks;
116     float avgWaitingTime = totalWaitingTime / totalTasks;
117     float throughput = (float) totalBurst / totalTasks;
118
119     printf("| %-30s | %-30s | %-30s |\n", "Average Turnaround Time", "Average Waiting Time", "Throughput");
120     printf("| %-30.2f | %-30.2f | %-30.2f |\n", avgTurnaroundTime, avgWaitingTime, throughput);
121     printf("-----\n");
122 }
123 }
124

```

```

125 /**
126  * Returns the next task selected to run.
127  */
128 Task *selectNextTask() {
129     // Find the task with the smallest arrival time (i.e., the first task in the list)
130     struct node *temp = head;
131     struct node *returnNode = head;
132
133     while (temp != NULL) {
134         if (temp->task->arrivalTime < returnNode->task->arrivalTime) {
135             returnNode = temp;
136         }
137         temp = temp->next;
138     }
139
140     return returnNode->task;
141 }

```

```
labuser1@ML-RefVm-535928:~/Desktop/Project05$ make fcfs2
gcc -Wall -c scheduler_fcfs_diff arrival.c
gcc -Wall -o fcfs2 driver.o scheduler_fcfs_diff arrival.o list.o CPU.o
labuser1@ML-RefVm-535928:~/Desktop/Project05$ ./fcfs2 fcfs_schedule_arrive_diff.txt
[T1] [2] [20]
[T3] [3] [25]
[T4] [5] [15]
[T5] [5] [20]
[T8] [200] [25]
[T6] [12] [10]
[T7] [100] [30]
[T2] [3] [25]
-----
| Task Name | Arrival | Burst | Turnaround | Waiting | Response | Response Ratio |
|-----|-----|-----|-----|-----|-----|-----|
| T1 | 2 | 20 | 20 | 0 | 0 | 1.00 |
| T3 | 3 | 25 | 44 | 19 | 19 | 1.76 |
| T2 | 3 | 25 | 69 | 44 | 44 | 2.76 |
| T4 | 5 | 15 | 82 | 67 | 67 | 5.47 |
| T5 | 5 | 20 | 102 | 82 | 82 | 5.10 |
| T6 | 12 | 10 | 105 | 95 | 95 | 10.50 |
| T7 | 100 | 30 | 47 | 17 | 17 | 1.57 |
| T8 | 200 | 25 | 25 | 0 | 0 | 1.00 |
|-----|-----|-----|-----|-----|-----|
| Average Turnaround Time | Average Waiting Time | Throughput |
| 61.75 | 40.50 | 21.25 |
|-----|-----|-----|
labuser1@ML-RefVm-535928:~/Desktop/Project05$ █
```

Team Member 3 Tasks (SJF Algorithm on schedule_arrive_zero.txt)

Q5. Implement SJF Execution:

- Update the code to implement the Shortest Job First (SJF) algorithm for task scheduling.

```
Open + schedule_sjf.c /home/labuser1/Downloads Save - + x
1 #include <stdlib.h>
2 #include <stdio.h>
3
4 #include "task.h"
5 #include "list.h"
6 #include "cpu.h"
7
8 // Reference to the head of the list
9 struct node *head = NULL;
10
11 // Sequence counter of next available thread identifier
12 int nextTid = 0;
13
14 Task *selectNextTask();
15
16
17 // Add a new task to the list of tasks
18 void add(char *name, int arrivalTime, int burst) {
19     // First, create the new task
20     Task *newTask = (Task *) malloc(sizeof(Task));
21
22     newTask->name = name;
23     newTask->tid = nextTid++;
24     newTask->arrivalTime = arrivalTime;
25     newTask->burst = burst;
26
27     // Insert the new task into the list of tasks
28     insert(&head, newTask);
29 }
30
31 /**
32  * Run the SJF scheduler
33  */
34 void schedule() {
35     Task *current;
36
37     traverse(head);
38     while (head != NULL) {
39         current = selectNextTask();
40         run(current, current->burst);
41         // Delete the task from the list
42         delete(&head, current);
43     }
44 }
45
46 /**
47  * Returns the next task selected to run.
48  */
49 Task *selectNextTask() {
50     // Find the task with the smallest burst time (i.e., the first task in
51     // the list)
52     struct node *temp = head;
53     struct node *returnNode = head;
54
55     while (temp != NULL) {
56         if (temp->task->burst < returnNode->task->burst) {
57             returnNode = temp;
58         }
59         temp = temp->next;
60     }
61
62     return returnNode->task;
63 }
64
65
66
67
```

```
36
37
38     traverse(head);
39
40     while (head != NULL) {
41         current = selectNextTask();
42         run(current, current->burst);
43         // Delete the task from the list
44         delete(&head, current);
45     }
46 }
47
48 /**
49  * Returns the next task selected to run.
50  */
51 Task *selectNextTask() {
52     // Find the task with the smallest burst time (i.e., the first task in
53     // the list)
54     struct node *temp = head;
55     struct node *returnNode = head;
56
57     while (temp != NULL) {
58         if (temp->task->burst < returnNode->task->burst) {
59             returnNode = temp;
60         }
61         temp = temp->next;
62     }
63
64     return returnNode->task;
65 }
66
67
```

- Compile the program using the provided makefile (make sjf).

```
labuser1@ML-RefVm-535928: ~/Downloads
File Edit View Search Terminal Help
labuser1@ML-RefVm-535928:~/Downloads$ make sjf
make: 'sjf' is up to date.
labuser1@ML-RefVm-535928:~/Downloads$
```

- Run the program with `schedule_arrive_zero.txt` input file.

```
labuser1@ML-RefVm-535928: ~/Downloads
File Edit View Search Terminal Help
labuser1@ML-RefVm-535928:~/Downloads$ make sjf
gcc -Wall -c -o schedule_sjf.o schedule_sjf.c
gcc -Wall -o sjf driver.o schedule_sjf.o list.o CPU.o
labuser1@ML-RefVm-535928:~/Downloads$ ./sjf schedule_arrive_zero.txt
[T8] [0] [25]
[T7] [0] [30]
[T6] [0] [10]
[T5] [0] [20]
[T4] [0] [15]
[T3] [0] [25]
[T2] [0] [25]
[T1] [0] [20]
Running task = [T6] [0] [10] for 10 units.
Running task = [T4] [0] [15] for 15 units.
Running task = [T5] [0] [20] for 20 units.
Running task = [T1] [0] [20] for 20 units.
Running task = [T8] [0] [25] for 25 units.
Running task = [T3] [0] [25] for 25 units.
Running task = [T2] [0] [25] for 25 units.
Running task = [T7] [0] [30] for 30 units.
labuser1@ML-RefVm-535928:~/Downloads$
```

- Confirm the execution order adheres to the SJF criteria.

Yes, it is in order.

Q6. Print Process Metrics for SJF:

- Modify code to calculate and print turnaround time, waiting time, response time, and response ratio for each process under SJF scheduling.

```

1 #include <stdlib.h>
2 #include <stdio.h>
3
4 #include "task.h"
5 #include "list.h"
6 #include "cpu.h"
7
8 // Reference to the head of the list
9 struct node *head = NULL;
10
11 // Sequence counter of next available thread identifier
12 int nextTid = 0;
13
14 // Forward declaration of selectNextTask()
15 Task *selectNextTask();
16
17 // Global variables to store metrics
18 int totalTurnaroundTime = 0;
19 int totalWaitingTime = 0;
20 int totalResponseTime = 0;
21 int totalProcessesExecuted = 0;
22
23 // Add a new task to the list of tasks
24 void add(char *name, int arrivalTime, int burst) {
25     // First, create the new task
26     Task *newTask = (Task *) malloc(sizeof(Task));
27
28     newTask->name = name;
29     newTask->tid = nextTid++;
30     newTask->arrivalTime = arrivalTime;
31     newTask->burst = burst;
32
33     // Insert the new task into the list of tasks
34     insert(&head, newTask);
35 }
36
37 /**
38  * Run the SJF scheduler
39  */
40 void schedule() {
41     Task *current;
42     int currentTime = 0;
43
44     traverse(head);
45
46     printf("-----\n");
47     printf("| %-8s | %-16s | %-13s | %-14s | %-15s |\n", "Process", "Turnaround Time", "Waiting Time", "Response Time", "Response Ratio");
48     printf("-----\n");
49

```



```

50 while (head != NULL) {
51     current = selectNextTask();
52
53     // Calculate metrics
54     int turnaroundTime = currentTime + current->burst - current->arrivalTime;
55     int waitingTime = currentTime - current->arrivalTime;
56     int responseTime = currentTime - current->arrivalTime;
57
58     // Print metrics for the current task
59     printf("| %-8s | %-15d | %-12d | %-13d | %-14.2f |\n", current->name, turnaroundTime, waitingTime, responseTime, (float) turnaroundTime / current->burst);
60
61     // Run the current task
62     //run(current, current->burst);
63
64     // Update current time
65     currentTime += current->burst;
66
67     // Accumulate total metrics
68     totalTurnaroundTime += turnaroundTime;
69     totalWaitingTime += waitingTime;
70     totalResponseTime += responseTime;
71     totalProcessesExecuted++;
72
73     // Delete the task from the list
74     delete(&head, current);
75 }
76
77 // Print average metrics
78 printf("-----\n");
79 printf("Average Turnaround Time: %-7.2f | Average Waiting Time: %-7.2f | Throughput: %-7.2f |\n", (float) totalTurnaroundTime / totalProcessesExecuted, (float) totalWaitingTime /
80 totalProcessesExecuted, (float) currentTime / totalProcessesExecuted);
81 printf("-----\n");
82
83 /**
84  * Returns the next task selected to run.
85  */
86 Task *selectNextTask() {
87     // Find the task with the smallest burst time (i.e., the first task in the list)
88     struct node *temp = head;
89     struct node *returnNode = head;
90
91     while (temp != NULL) {
92         if (temp->task->burst < returnNode->task->burst) {
93             returnNode = temp;
94         }
95         temp = temp->next;
96     }
97 }

```

```

83 /**
84  * Returns the next task selected to run.
85  */
86 Task *selectNextTask() {
87     // Find the task with the smallest burst time (i.e., the first task in the list)
88     struct node *temp = head;
89     struct node *returnNode = head;
90
91     while (temp != NULL) {
92         if (temp->task->burst < returnNode->task->burst) {
93             returnNode = temp;
94         }
95         temp = temp->next;
96     }
97
98     return returnNode->task;
99 }

```

Task *selectNextTask() {

 // Find the task with the smallest burst time (i.e., the first task in the list)

 struct node *temp = head; // Initialize a temporary node pointer to the beginning of the list

 struct node *returnNode = head; // Initialize the node pointer to return with the first node

 // Traverse the list to find the task with the smallest burst time

 while (temp != NULL) { // Iterate until the end of the list is reached

 if (temp->task->burst < returnNode->task->burst) { // Check if current task has a smaller burst time

```

        returnNode = temp; // Update the returnNode pointer to point to the current node
    }
    temp = temp->next; // Move to the next node in the list
}

return returnNode->task; // Return the task with the smallest burst time
}

```

- Organize output in table format as required.

```

labuser1@ML-RefVm-535928:~/Desktop/Project0S$ make sjf
make: 'sjf' is up to date.
labuser1@ML-RefVm-535928:~/Desktop/Project0S$ ./sjf schedule_arrive_zero.txt
[T8] [0] [25]
[T7] [0] [30]
[T6] [0] [10]
[T5] [0] [20]
[T4] [0] [15]
[T3] [0] [25]
[T2] [0] [25]
[T1] [0] [20]

```

Process	Turnaround Time	Waiting Time	Response Time	Response Ratio
T6	10	0	0	1.00
T4	25	10	10	1.67
T5	45	25	25	2.25
T1	65	45	45	3.25
T8	90	65	65	3.60
T3	115	90	90	4.60
T2	140	115	115	5.60
T7	170	140	140	5.67

Average Turnaround Time: 82.50		Average Waiting Time: 61.25		Throughput: 21.25

```

labuser1@ML-RefVm-535928:~/Desktop/Project0S$ █

```

Q7. Print Average Metrics for SJF:

- Extend code to calculate and print average turnaround time, average waiting time, and throughput under SJF scheduling.

We did this in the last part.

- Format output in a clear tabular layout for better readability.

```

labuser1@ML-RefVm-535928:~/Desktop/Project0S$ make sjf
make: 'sjf' is up to date.
labuser1@ML-RefVm-535928:~/Desktop/Project0S$ ./sjf schedule_arrive_zero.txt
[T8] [0] [25]
[T7] [0] [30]
[T6] [0] [10]
[T5] [0] [20]
[T4] [0] [15]
[T3] [0] [25]
[T2] [0] [25]
[T1] [0] [20]

```

Process	Turnaround Time	Waiting Time	Response Time	Response Ratio
T6	10	0	0	1.00
T4	25	10	10	1.67
T5	45	25	25	2.25
T1	65	45	45	3.25
T8	90	65	65	3.60
T3	115	90	90	4.60
T2	140	115	115	5.60
T7	170	140	140	5.67
Average Turnaround Time: 82.50 Average Waiting Time: 61.25 Throughput: 21.25				

```

labuser1@ML-RefVm-535928:~/Desktop/Project0S$ █

```

