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.

```c
#include <stdlib.h>
#include <stdio.h>

#include "task.h"
#include "list.h"
#include "cpu.h"

// Reference to the head of the list
struct node *head = NULL;

// Sequence counter of next available thread identifier
int nextTid = 0;

Task *selectNextTask();


// Add a new task to the list of tasks
void add(char *name, int arrivalTime, int burst) {
    // First, create the new task
    Task *newTask = (Task *) malloc(sizeof(Task));

    newTask->name = name;
    newTask->tid = nextTid++;
    newTask->arrivalTime = arrivalTime;
    newTask->burst = burst;

    // Insert the new task into the list of tasks
    insert(&head, newTask);
}

// Function to reverse the linked list
void reverseList() {
    struct node *prev = NULL;
    struct node *current = head;
    struct node *next = NULL;

    while (current != NULL) {
        next = current->next;
        current->next = prev;
        prev = current;
```

```
        prev = current;
        current = next;
    }
    head = prev;
}

/**
 * Run the FCFS scheduler
 */
void schedule() {
    Task *current;

traverse(head);
    // Reverse the list before scheduling
    reverseList();


    while (head != NULL) {
        current = selectNextTask();

run(current,current->burst);
        // Delete the task from the list
        delete(&head, current);
    }
}


/**
 * Returns the next task selected to run.
 */
Task *selectNextTask() {
    // Find the task with the smallest arrival time (i.e., the first task in the list)
    struct node *temp = head;
    struct node *returnNode = head;

    while (temp != NULL) {
        if (temp->task->arrivalTime < returnNode->task->arrivalTime) {
            returnNode = temp;
        }
        temp = temp->next;
    }

    return returnNode->task;
}
```

```
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/ProjectOS$ 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/ProjectOS$ ./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/ProjectOS$
```

- 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)

```c
#include <stdlib.h>
#include <stdio.h>

#include "task.h"
#include "list.h"
#include "cpu.h"

// Reference to the head of the list
struct node *head = NULL;

// Sequence counter of next available thread identifier
int nextTid = 0;

Task *selectNextTask();

// Global variables to store statistics
float totalTurnaroundTime = 0;
float totalWaitingTime = 0;
int totalTasks = 0;

/*added
Task *arr[8];
int arr2[8];*/

// Add a new task to the list of tasks
void add(char *name, int arrivalTime, int burst) {
    // First, create the new task
    Task *newTask = (Task *) malloc(sizeof(Task));

    newTask->name = name;
    newTask->tid = nextTid++;
    newTask->arrivalTime = arrivalTime;
    newTask->burst = burst;

    // Insert the new task into the list of tasks
    insert(&head, newTask);
}

// Function to reverse the linked list
void reverseList() {
    struct node *prev = NULL;
    struct node *current = head;
    struct node *next = NULL;

    while (current != NULL) {
        next = current->next;
        current->next = prev;
```

```
        prev = current;
        current = next;
    }
    head = prev;
}

/**
 * Run the FCFS scheduler
 */
void schedule() {
    Task *current;

    traverse(head);
    // Reverse the list before scheduling
    reverseList();

    // Start system time
    int systemTime = 0;

    printf("-------------------------------------------------------------------------------------------------------\n");
    printf("| %-10s | %-10s | %-10s | %-10s | %-12s | %-12s | %-12s |\n", "Task Name", "Arrival", "Burst", "Turnaround", "Waiting", "Response", "Response Ratio");
    printf("-------------------------------------------------------------------------------------------------------\n");

    while (head != NULL) {
        current = selectNextTask();

        //run(current,current->burst);
        /*added
        int i = 0;
        arr[i] = current;
        arr2[i] = current->burst;
        i++;*/

        // Update waiting time for all tasks waiting in the queue
        totalWaitingTime += systemTime - current->arrivalTime;

        // Calculate response time
        int responseTime = systemTime - current->arrivalTime;

        // Calculate turnaround time
        int turnaroundTime = responseTime + current->burst;

        // Calculate response ratio
        float responseRatio = (float) turnaroundTime / current->burst;

        // Print task statistics
        printf("| %-10s | %-10d | %-10d | %-10d | %-12d | %-12d | %-12.2f   |\n", current->name, current->arrivalTime, current->burst, turnaroundTime, systemTime - current->arrivalTime, responseTime, responseRatio);

        // Update system time
        systemTime += current->burst;

        // Update total turnaround time
        totalTurnaroundTime += turnaroundTime;

        // Delete the task from the list
        delete(&head, current);

        // Increment total tasks
        totalTasks++;
    }

    printf("-------------------------------------------------------------------------------------------------------\n");

    // Calculate and print statistics
    float avgTurnaroundTime = totalTurnaroundTime / totalTasks;
    float avgWaitingTime = totalWaitingTime / totalTasks;
    float throughput = (float) systemTime / totalTasks;

    printf("| %-30s | %-30s | %-30s |\n", "Average Turnaround Time", "Average Waiting Time", "Throughput");
    printf("| %-30.2f | % -30.2f | %-30.2f |\n", avgTurnaroundTime, avgWaitingTime, throughput);
    printf("-------------------------------------------------------------------------------------------------------\n");

    /*added
    for(int i = 0; i < 8; i++)
    {
        run(arr[i],arr2[i]);
    }*/
}

/**
 * Returns the next task selected to run.
 */
Task *selectNextTask() {
    // Find the task with the smallest arrival time (i.e., the first task in the list)
    struct node *temp = head;
    struct node *returnNode = head;

    while (temp != NULL) {
        if (temp->task->arrivalTime < returnNode->task->arrivalTime) {
            returnNode = temp;
        }
        temp = temp->next;
    }

```

- Format output in a well-organized table format.

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

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

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.

```c
#include <stdlib.h>
#include <stdio.h>

#include "task.h"
#include "list.h"
#include "cpu.h"

// Reference to the head of the list
struct node *head = NULL;

// Sequence counter of next available thread identifier
int nextTid = 0;

Task *selectNextTask();

// Global variables to store statistics
float totalTurnaroundTime = 0;
float totalWaitingTime = 0;
int firstResponseTime = 0;
int completionTime = 0;
int totalTasks = 0;
int totalBurst = 0;

// Add a new task to the list of tasks
void add(char *name, int arrivalTime, int burst) {
    // First, create the new task
    Task *newTask = (Task *) malloc(sizeof(Task));

    newTask->name = name;
    newTask->tid = nextTid++;
    newTask->arrivalTime = arrivalTime;
    newTask->burst = burst;

    // Insert the new task into the list of tasks
    insert(&head, newTask);
}

// Function to reverse the linked list
void reverseList() {
    struct node *prev = NULL;
    struct node *current = head;
    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 }
```

- 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/ProjectOS$ make fcfs2
gcc -Wall -o fcfs2 driver.o scheduler_fcfs_diff_arrival.o list.o CPU.o
labuser1@ML-RefVm-535928:~/Desktop/ProjectOS$ ./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/ProjectOS$
```

- 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**.

```c
#include <stdlib.h>
#include <stdio.h>

#include "task.h"
#include "list.h"
#include "cpu.h"

// Reference to the head of the list
struct node *head = NULL;

// Sequence counter of next available thread identifier
int nextTid = 0;

Task *selectNextTask();

// Global variables to store statistics
float totalTurnaroundTime = 0;
float totalWaitingTime = 0;
int firstResponseTime = 0;
int completionTime = 0;
int totalTasks = 0;
int totalBurst = 0;

// Add a new task to the list of tasks
void add(char *name, int arrivalTime, int burst) {
    // First, create the new task
    Task *newTask = (Task *) malloc(sizeof(Task));

    newTask->name = name;
    newTask->tid = nextTid++;
    newTask->arrivalTime = arrivalTime;
    newTask->burst = burst;

    // Insert the new task into the list of tasks
    insert(&head, newTask);
}

// Function to reverse the linked list
void reverseList() {
    struct node *prev = NULL;
    struct node *current = head;
    struct node *next = NULL;

```

```
    while (current != NULL) {
        next = current->next;
        current->next = prev;
        prev = current;
        current = next;
    }
    head = prev;
}

/**
 * Run the FCFS scheduler
 */
void schedule() {
    Task *current;

    // Reverse the list before scheduling
    reverseList();
    traverse(head);
    //reverseList();

    printf("----------------------------------------------------------------------------------------------------\n");
    printf("| %-10s | %-10s | %-10s | %-10s | %-12s | %-12s | %-12s |\n", "Task Name", "Arrival", "Burst", "Turnaround", "Waiting", "Response", "Response Ratio");
    printf("----------------------------------------------------------------------------------------------------\n");

    while (head != NULL) {
        current = selectNextTask();

        //run(current,current->burst);

        if(firstResponseTime < current->arrivalTime)
        {
                firstResponseTime = current->arrivalTime;
        }

        completionTime = firstResponseTime + current->burst;

        // Calculate response time
        int responseTime = firstResponseTime - current->arrivalTime;

        // Calculate turnaround time
        int turnaroundTime = completionTime - current->arrivalTime;

        // Update waiting time for all tasks waiting in the queue
        int waitingTime = turnaroundTime - current->burst;

        // Calculate response ratio
        float responseRatio = (float) (waitingTime + current->burst) / current->burst;

        // Print task statistics
        printf("| %-10s | %-10d | %-10d | %-10d | %-12d | %-12d | %-12.2f   |\n", current->name, current->arrivalTime, current->burst, turnaroundTime, waitingTime, responseTime,
responseRatio);

        //Update the first response time
        firstResponseTime += current->burst;

        // Update total turnaround time
        totalTurnaroundTime += turnaroundTime;

        totalWaitingTime += waitingTime;

        totalBurst += current->burst;

        // Delete the task from the list
        delete(&head, current);

        // Increment total tasks
        totalTasks++;
    }

    printf("----------------------------------------------------------------------------------------------------\n");

    // Calculate and print statistics
    float avgTurnaroundTime = totalTurnaroundTime / totalTasks;
    float avgWaitingTime = totalWaitingTime / totalTasks;
    float throughput = (float) totalBurst / totalTasks;

    printf("| %-30s | %-30s | %-30s |\n", "Average Turnaround Time", "Average Waiting Time", "Throughput");
    printf("| %-30.2f | % -30.2f | %-30.2f |\n", avgTurnaroundTime, avgWaitingTime, throughput);
    printf("----------------------------------------------------------------------------------------------------\n");

}
```

```
/**
 * Returns the next task selected to run.
 */
Task *selectNextTask() {
    // Find the task with the smallest arrival time (i.e., the first task in the list)
    struct node *temp = head;
    struct node *returnNode = head;

    while (temp != NULL) {
        if (temp->task->arrivalTime < returnNode->task->arrivalTime) {
            returnNode = temp;
        }
        temp = temp->next;
    }

    return returnNode->task;
}
```

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

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.

schedule_sjf.c
/home/labuser1/Downloads

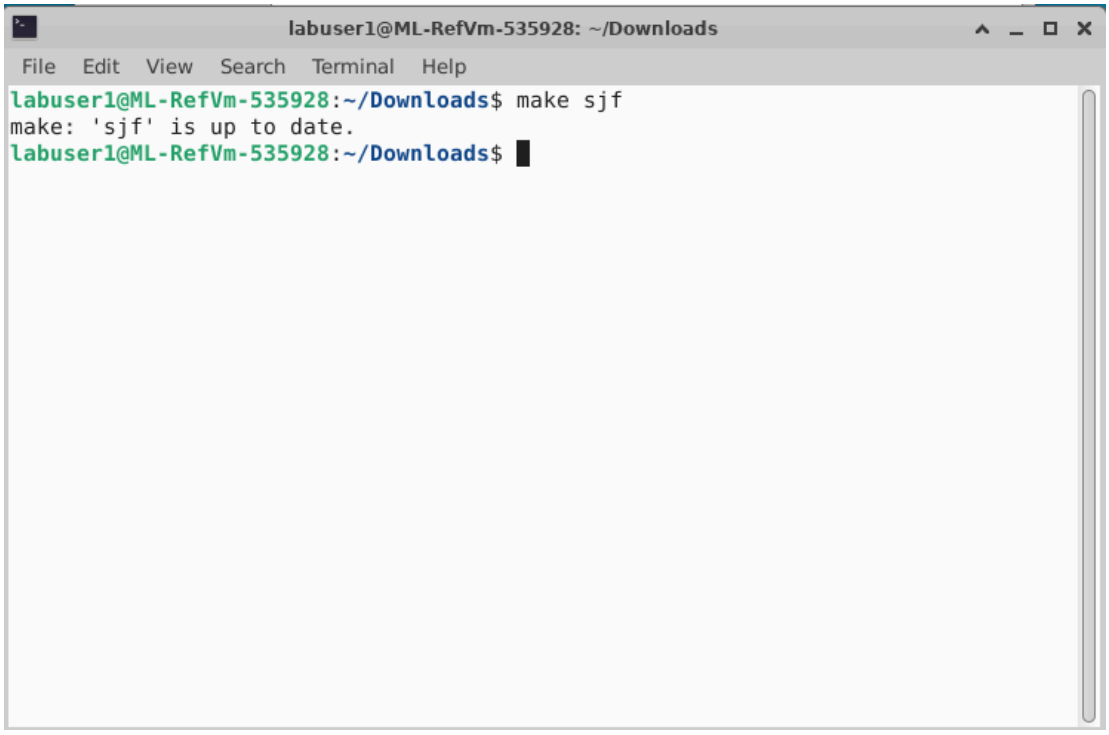
```c
#include <stdlib.h>
#include <stdio.h>

#include "task.h"
#include "list.h"
#include "cpu.h"

// Reference to the head of the list
struct node *head = NULL;

// Sequence counter of next available thread identifier
int nextTid = 0;

Task *selectNextTask();


// Add a new task to the list of tasks
void add(char *name, int arrivalTime, int burst) {
    // First, create the new task
    Task *newTask = (Task *) malloc(sizeof(Task));

    newTask->name = name;
    newTask->tid = nextTid++;
    newTask->arrivalTime = arrivalTime;
    newTask->burst = burst;

    // Insert the new task into the list of tasks
    insert(&head, newTask);
}

/**
 * Run the SJF scheduler
 */
void schedule() {
    Task *current;

```

C Tab Width: 8 Ln 67, Col 1 INS

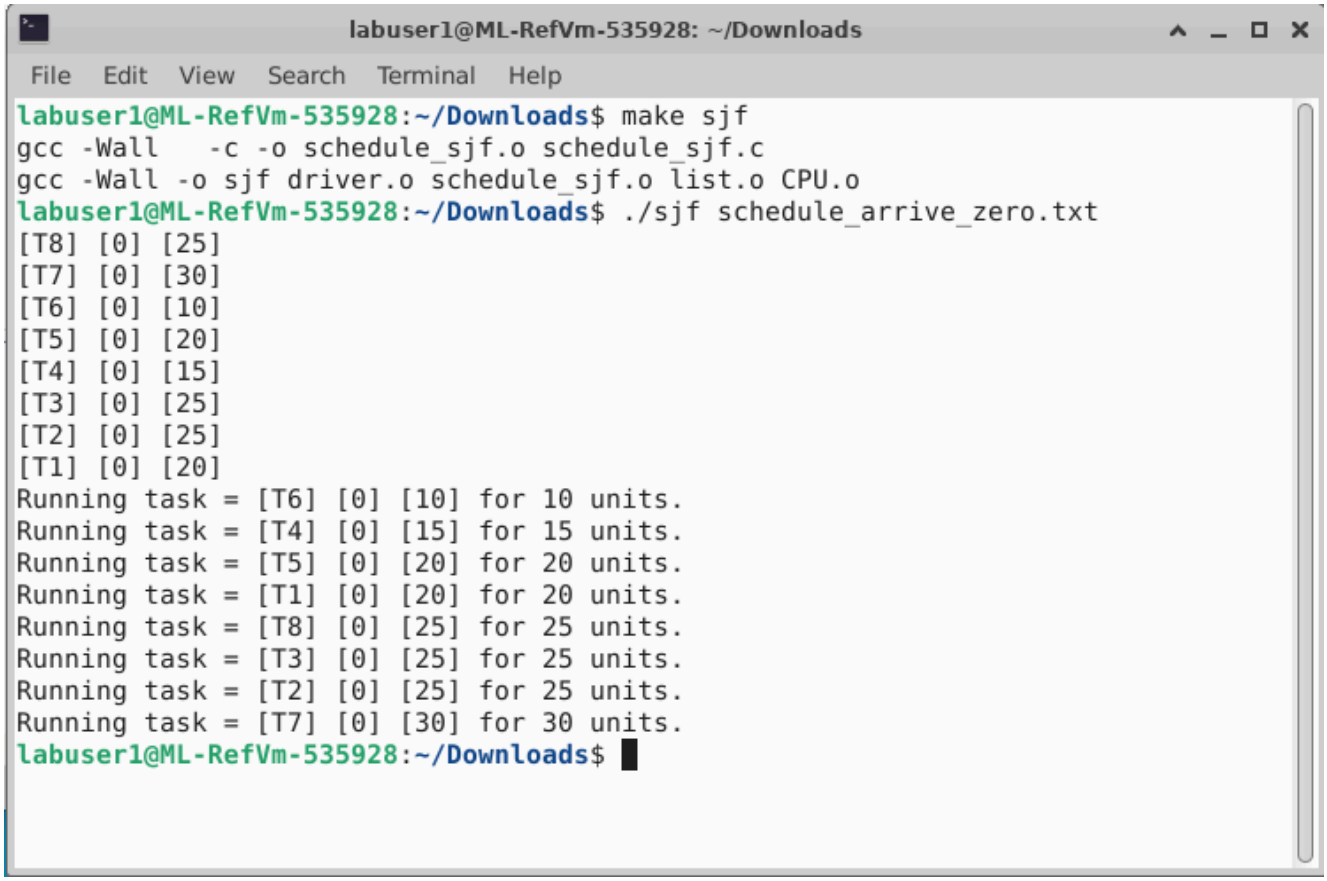
```c

    traverse(head);

    while (head != NULL) {
        current = selectNextTask();

        run(current, current->burst);

        // Delete the task from the list
        delete(&head, current);
    }
}

/**
 * Returns the next task selected to run.
 */
Task *selectNextTask() {
    // Find the task with the smallest burst time (i.e., the first task in the list)
    struct node *temp = head;
    struct node *returnNode = head;

    while (temp != NULL) {
        if (temp->task->burst < returnNode->task->burst) {
            returnNode = temp;
        }
        temp = temp->next;
    }

    return returnNode->task;
}

```

C Tab Width: 8 Ln 67, Col 1 INS

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



- Run the program with schedule_arrive_zero.txt input file.



- 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.

```c
#include <stdlib.h>
#include <stdio.h>

#include "task.h"
#include "list.h"
#include "cpu.h"

// Reference to the head of the list
struct node *head = NULL;

// Sequence counter of next available thread identifier
int nextTid = 0;

// Forward declaration of selectNextTask()
Task *selectNextTask();

// Global variables to store metrics
int totalTurnaroundTime = 0;
int totalWaitingTime = 0;
int totalResponseTime = 0;
int totalProcessesExecuted = 0;

// Add a new task to the list of tasks
void add(char *name, int arrivalTime, int burst) {
    // First, create the new task
    Task *newTask = (Task *) malloc(sizeof(Task));

    newTask->name = name;
    newTask->tid = nextTid++;
    newTask->arrivalTime = arrivalTime;
    newTask->burst = burst;

    // Insert the new task into the list of tasks
    insert(&head, newTask);
}

/**
 * Run the SJF scheduler
 */
void schedule() {
    Task *current;
    int currentTime = 0;

    traverse(head);

    printf("-------------------------------------------------------------------------------\n");
    printf("| %-8s | %-16s | %-13s | %-14s | %-15s |\n", "Process", "Turnaround Time", "Waiting Time", "Response Time", "Response Ratio");
    printf("-------------------------------------------------------------------------------\n");

```

```
    while (head != NULL) {
        current = selectNextTask();

        // Calculate metrics
        int turnaroundTime = currentTime + current->burst - current->arrivalTime;
        int waitingTime = currentTime - current->arrivalTime;
        int responseTime = currentTime - current->arrivalTime;

        // Print metrics for the current task
        printf("| %-8s | %-15d   | %-12d   | %-13d   | %-14.2f |\n", current->name, turnaroundTime, waitingTime, responseTime, (float) turnaroundTime / current->burst);

        // Run the current task
        //run(current, current->burst);

        // Update current time
        currentTime += current->burst;

        // Accumulate total metrics
        totalTurnaroundTime += turnaroundTime;
        totalWaitingTime += waitingTime;
        totalResponseTime += responseTime;
        totalProcessesExecuted++;

        // Delete the task from the list
        delete(&head, current);
    }

    // Print average metrics
    printf("-------------------------------------------------------------------------------\n");
    printf("| Average Turnaround Time: %-7.2f | Average Waiting Time: %-7.2f | Throughput: %-7.2f |\n", (float) totalTurnaroundTime / totalProcessesExecuted, (float) totalWaitingTime /
totalProcessesExecuted, (float) currentTime / totalProcessesExecuted);
    printf("-------------------------------------------------------------------------------\n");
}

/**
 * Returns the next task selected to run.
 */
Task *selectNextTask() {
    // Find the task with the smallest burst time (i.e., the first task in the list)
    struct node *temp = head;
    struct node *returnNode = head;

    while (temp != NULL) {
        if (temp->task->burst < returnNode->task->burst) {
            returnNode = temp;
        }
        temp = temp->next;
    }
```

```
/**
 * Returns the next task selected to run.
 */
Task *selectNextTask() {
    // Find the task with the smallest burst time (i.e., the first task in the list)
    struct node *temp = head;
    struct node *returnNode = head;

    while (temp != NULL) {
        if (temp->task->burst < returnNode->task->burst) {
            returnNode = temp;
        }
        temp = temp->next;
    }

    return returnNode->task;
}
```

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/ProjectOS$ make sjf
make: 'sjf' is up to date.
labuser1@ML-RefVm-535928:~/Desktop/ProjectOS$ ./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/ProjectOS$
```

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/ProjectOS$ make sjf
make: 'sjf' is up to date.
labuser1@ML-RefVm-535928:~/Desktop/ProjectOS$ ./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/ProjectOS$
```