Course Name : Operating System

Course Code : 2CS403

Date : 20/03/2023

Group Members : 21BCE248, 21BCE259

Assignment Question:

Write a **single** menu driven C program to simulate the following CPU scheduling algorithms to find turnaround time and waiting time.
a) First-Come First-Served
b) Shortest Job First
c) Round Robin Scheduling
d) Priority Scheduling
e) Shortest Remaining Time First
f) Longest remaining time first(the decision to preempt or not to preempt is to be taken after each unit/cycle of CPU)

Note: Your program should be dynamic that automatically identifies the number of processes and other information from the file i.e. you can not initialize these variables. These are dynamic as you change the input in the file you did not need to change anything in the program.

Input file: Input for the respective algorithm

Output file:
Average turn-around time:
Average waiting time:
Print Gantt Chart in the form of array
e.g. (each cell consists of 1 unit CPU burst)
P1 P1 P2 P2 P1 P3 P3 …….

Methodology Followed :

We have created a menu-driven C program that asks users to enter input file names and output file names and asks them to select one of the six CPU scheduling algorithms.

We have created a structure named process. This structure contains various attributes to store process id, arrival time, burst time, etc. It maintains the data of all processes.

Input file format :

Number of processes

Process id, Process Arrival time, CPU Burst time, Priority

Output file format :

Gantt Chart

Average turnaround time

Average waiting time

The program contains various functions to carry out different tasks.

Sr no.	Function Name	Data type	Operation / Description
1	read_input	**struct process**	Reads the data of processes from the input file.
2	write_output	**struct process**	Writes the Gantt chart, Average turn around time and Average waiting time into the output file.
3	fcfs	**void**	Arranges the processes and prepare the output according to the First Come First Serve CPU scheduling algorithm.
4	sjf	**void**	Arranges the processes and prepare the output according to the Shortest Job First CPU scheduling algorithm.
5	rr	**void**	Arranges the processes and prepare the output according to the Round Robin CPU scheduling algorithm.
6	priority	**void**	Arranges the processes and prepare the output according to the Priority of the processes.
7	srtf	**void**	Arranges the processes and prepare the output according to the Shortest Remaining Time First CPU scheduling algorithm.
8	lrtp	**void**	Arranges the processes and prepare the output according to the Longest

			Remaining Time first CPU scheduling algorithm.
9	compute_avg_times	**void**	It computes the Average Turn around time and Average Waiting time and prints it into the output file.
10	main	**int**	It is the driver function for all of the above functions.

Code :

```
#include <stdio.h>
#include <stdlib.h>
#include <limits.h>
#include <string.h>

// Process structure
struct process
{
    int pid;              // process ID
    int burst_time;       // burst time
    int priority;         // priority
    int arrival_time;     // arrival time
    int start_time;       // start time
    int finish_time;      // finish time
    int waiting_time;     // waiting time
    int turnaround_time; // turnaround time
    int remaining_time;  // remaining time for Shortest Remaining Time First and
Longest Remaining Time First
    int completed;        // Flag to indicate if the process has completed
execution
    int preempted;        // Flag to indicate if the process was preempted during
execution
};

// Function prototypes
void fcfs(struct process *processes, int n, char *outputFilename);
void sjf(struct process *processes, int n, char *outputFilename);
void rr(struct process *processes, int n, int quantum, char *outputFilename);
void priority(struct process *processes, int n, char *outputFilename);
void srtf(struct process *processes, int n, char *outputFilename);
void lrtf(struct process *processes, int n, char *outputFilename);
// void print_gantt_chart(struct process *processes, int n);
void compute_avg_times(struct process *processes, int n, float
*avg_turnaround_time, float *avg_waiting_time);

// Function to read input from file and create an array of processes
```

```c
struct process *read_input(char *filename, int *n)
{
    FILE *input_file = fopen(filename, "r");
    if (input_file == NULL)
    {
        printf("\n --> Error: Unable to open input file %s\n", filename);
        exit(EXIT_FAILURE);
    }

    fscanf(input_file, "%d", n);
    // *n -> number of processes mention in the file
    struct process *processes = (struct process *)malloc(*n * sizeof(struct
process));

    for (int i = 0; i < *n; i++)
    {
        fscanf(input_file, "%d %d %d %d", &processes[i].pid,
&processes[i].arrival_time, &processes[i].burst_time, &processes[i].priority);
        processes[i].start_time = -1;                          // initialize to -1
        processes[i].finish_time = -1;                         // initialize to -1
        processes[i].waiting_time = -1;                        // initialize to 0
        processes[i].turnaround_time = -1;                     // initialize to 0
        processes[i].remaining_time = processes[i].burst_time; // initialize
remaining time
        processes[i].completed = 0;
        processes[i].preempted = 0;
    }

    fclose(input_file);
    return processes;
}

// Function to compute average turnaround time and average waiting time
void compute_avg_times(struct process *processes, int n, float
*avg_turnaround_time, float *avg_waiting_time)
{
    *avg_turnaround_time = 0;
    *avg_waiting_time = 0;
    for (int i = 0; i < n; i++)
    {
        processes[i].turnaround_time = processes[i].finish_time -
processes[i].arrival_time;
        processes[i].waiting_time = processes[i].turnaround_time -
processes[i].burst_time;
        *avg_turnaround_time += processes[i].turnaround_time;
        *avg_waiting_time += processes[i].waiting_time;
    }
    *avg_turnaround_time /= n;
    *avg_waiting_time /= n;
}
```

```c
// Function to simulate the First-Come First-Served (FCFS) CPU scheduling
algorithm
void fcfs(struct process *processes, int n, char *outputFilename)
{
    FILE *fpout = fopen(outputFilename, "w");
    if (fpout == NULL)
    {
        printf("\n --> Error: Unable to open output file %s\n", outputFilename);
        exit(1);
    }

    // Print Gantt chart
    fprintf(fpout, "Gantt Chart:\n");

    printf("\n --> Running First-Come First-Served (FCFS) CPU scheduling
algorithm...\n");
    int current_time = 0;
    for (int i = 0; i < n; i++)
    {
        if (current_time < processes[i].arrival_time)
        {
            for (int idle = 1; idle <= processes[i].arrival_time - current_time;
idle++)
            {
                fprintf(fpout, "| Idle ");
            }
            current_time = processes[i].arrival_time;
        }
        printf(" Processing P%d...\n", processes[i].pid);

        processes[i].finish_time = current_time + processes[i].burst_time;
        current_time = processes[i].finish_time;

        for (int k = 1; k <= processes[i].burst_time; k++)
        {
            fprintf(fpout, "| P%d ", processes[i].pid);
        }
    }
    fprintf(fpout, "\n\n");
    printf("\n --> All processes have finished executing.\n");
    float avg_turnaround_time, avg_waiting_time;
    compute_avg_times(processes, n, &avg_turnaround_time, &avg_waiting_time);
    printf("\n --> Average turnaround time: %.2f\n", avg_turnaround_time);
    printf(" --> Average waiting time: %.2f\n", avg_waiting_time);

    // Get the output in output file
    fprintf(fpout, "Average turn-around time: %.2f\n", avg_turnaround_time);
    fprintf(fpout, "Average waiting time: %.2f\n\n", avg_waiting_time);
    fclose(fpout);
```

```
}

// Function to simulate the Shortest Job First (SJF) CPU scheduling algorithm
void sjf(struct process *processes, int n, char *outputFilename)
{
    FILE *fpout = fopen(outputFilename, "w");
    if (fpout == NULL)
    {
        printf("\n --> Error: Unable to open output file %s\n", outputFilename);
        exit(1);
    }

    // Print Gantt chart
    fprintf(fpout, "Gantt Chart:\n");

    printf("\n --> Running Shortest Job First (SJF) CPU scheduling
algorithm...\n");
    int current_time = 0, completed = 0;
    struct process *shortest_job = NULL;
    while (completed < n)
    {
        shortest_job = NULL;
        for (int i = 0; i < n; i++)
        {
            if (processes[i].arrival_time <= current_time &&
processes[i].remaining_time > 0)
            {
                if (shortest_job == NULL || processes[i].burst_time <
shortest_job->burst_time)
                {
                    shortest_job = &processes[i];
                }
            }
        }
        if (shortest_job == NULL)
        {
            current_time++;
            fprintf(fpout, "| Idle ");
        }
        else
        {
            for (int i = 1; i <= shortest_job->burst_time; i++)
            {
                fprintf(fpout, "| P%d ", shortest_job->pid);
                shortest_job->remaining_time--;
                current_time++;
            }
            printf("Processing P%d...\n", shortest_job->pid);
            shortest_job->finish_time = current_time + 1;
            completed++;
```

```
        }
    }
    fprintf(fpout, "\n\n");
    printf("\n --> All processes have finished executing.\n");

    float avg_turnaround_time, avg_waiting_time;
    compute_avg_times(processes, n, &avg_turnaround_time, &avg_waiting_time);
    printf("\n --> Average turnaround time: %.2f\n", avg_turnaround_time);
    printf(" --> Average waiting time: %.2f\n", avg_waiting_time);

    // Get the output in output file
    fprintf(fpout, "Average turn-around time: %.2f\n", avg_turnaround_time);
    fprintf(fpout, "Average waiting time: %.2f\n\n", avg_waiting_time);
    fclose(fpout);
}

// Function to simulate the Round Robin (RR) CPU scheduling algorithm
// This algorithm still under development
void rr(struct process *processes, int n, int quantum, char *outputFilename)
{
    FILE *fpout = fopen(outputFilename, "w");
    if (fpout == NULL)
    {
        printf("\n --> Error: Unable to open output file %s\n", outputFilename);
        exit(1);
    }

    // Print Gantt chart
    fprintf(fpout, "Gantt Chart:\n");

    printf("\n --> Running Round Robin (RR) CPU scheduling algorithm with quantum
%d...\n", quantum);
    int current_time = 0, completed = 0;
    int *remaining_time = (int *)malloc(n * sizeof(int));
    for (int i = 0; i < n; i++)
    {
        remaining_time[i] = processes[i].burst_time;
    }
    while (completed < n)
    {
        int flag = 1; // to check idle condition
        for (int i = 0; i < n; i++)
        {
            if (remaining_time[i] > 0 && processes[i].arrival_time <=
current_time)
            {
                flag = 0;
                if (remaining_time[i] > quantum)
                {
                    for (int k = 1; k <= quantum; k++)
```

```
                    {
                        fprintf(fpout, "| P%d ", processes[i].pid);
                    }
                    printf("Processing P%d...\n", processes[i].pid);
                    remaining_time[i] -= quantum;
                    current_time += quantum;
                }
                else
                {
                    for (int k = 1; k <= remaining_time[i]; k++)
                    {
                        fprintf(fpout, "| P%d ", processes[i].pid);
                    }
                    printf("Processing P%d...\n", processes[i].pid);
                    current_time += remaining_time[i];
                    processes[i].finish_time = current_time;
                    remaining_time[i] = 0;
                    completed++;
                }
            }
        }
        if(flag)
        {
            current_time++;
            fprintf(fpout, "| Idle ");
        }
    }
    fprintf(fpout, "\n\n");

    printf("\n --> All processes have finished executing.\n");
    // print_gantt_chart(processes, n);
    float avg_turnaround_time, avg_waiting_time;
    compute_avg_times(processes, n, &avg_turnaround_time, &avg_waiting_time);
    printf("\n --> Average turnaround time: %.2f\n", avg_turnaround_time);
    printf(" --> Average waiting time: %.2f\n", avg_waiting_time);
    free(remaining_time);

    // Get the output in output file
    fprintf(fpout, "Average turn-around time: %.2f\n", avg_turnaround_time);
    fprintf(fpout, "Average waiting time: %.2f\n\n", avg_waiting_time);
    fclose(fpout);
}

// Function to simulate the Priority Scheduling CPU scheduling algorithm
void priority(struct process *processes, int n, char *outputFilename)
{
    FILE *fpout = fopen(outputFilename, "w");
    if (fpout == NULL)
    {
        printf("\n --> Error: Unable to open output file %s\n", outputFilename);
```

```
        exit(1);
    }

    // Print Gantt chart
    fprintf(fpout, "Gantt Chart:\n");

    printf("Running Priority Scheduling CPU scheduling algorithm...\n");
    int current_time = 0, completed = 0;
    while (completed < n)
    {
        int min_priority = INT_MAX, index = -1;
        for (int i = 0; i < n; i++)
        {
            if (processes[i].arrival_time <= current_time &&
!processes[i].completed && processes[i].priority < min_priority)
            {
                min_priority = processes[i].priority;
                index = i;
            }
        }
        if (index == -1)
        {
            fprintf(fpout, "| Idle ");
            current_time++;
            continue;
        }

        for (int k = 1; k <= processes[index].burst_time; k++)
        {
            fprintf(fpout, "| P%d ", processes[index].pid);
        }
        printf("Processing P%d...\n", processes[index].pid);
        processes[index].start_time = current_time;
        processes[index].finish_time = current_time + processes[index].burst_time;
        processes[index].waiting_time = processes[index].start_time -
processes[index].arrival_time;
        processes[index].turnaround_time = processes[index].finish_time -
processes[index].arrival_time;
        current_time = processes[index].finish_time;
        processes[index].completed = 1;
        completed++;
    }

    fprintf(fpout, "\n\n");
    printf("\n --> All processes have finished executing.\n");
    // print_gantt_chart(processes, n);
    float avg_turnaround_time, avg_waiting_time;
    compute_avg_times(processes, n, &avg_turnaround_time, &avg_waiting_time);
    printf("\n --> Average turnaround time: %.2f\n", avg_turnaround_time);
    printf(" --> Average waiting time: %.2f\n", avg_waiting_time);
```

```
    // Get the output in output file
    fprintf(fpout, "Average turn-around time: %.2f\n", avg_turnaround_time);
    fprintf(fpout, "Average waiting time: %.2f\n\n", avg_waiting_time);
    fclose(fpout);
}

// Function to simulate the Shortest Remaining Time First (SRTF) CPU scheduling
algorithm
void srtf(struct process *processes, int n, char *outputFilename)
{
    FILE *fpout = fopen(outputFilename, "w");
    if (fpout == NULL)
    {
        printf("\n --> Error: Unable to open output file %s\n", outputFilename);
        exit(1);
    }

    // Print Gantt chart
    fprintf(fpout, "Gantt Chart:\n");

    printf("\n --> Running Shortest Remaining Time First (SRTF) CPU scheduling
algorithm...\n");
    int current_time = 0, completed = 0;
    while (completed < n)
    {
        int min_burst_time = INT_MAX, index = -1;
        for (int i = 0; i < n; i++)
        {
            if (processes[i].arrival_time <= current_time &&
!processes[i].completed && processes[i].burst_time < min_burst_time)
            {
                min_burst_time = processes[i].burst_time;
                index = i;
            }
        }
        if (index == -1)
        {
            fprintf(fpout, "| Idle ");
            current_time++;
            continue;
        }
        fprintf(fpout, "| P%d ", processes[index].pid);
        printf("Processing P%d...\n", processes[index].pid);
        processes[index].start_time = current_time;
        processes[index].burst_time--;
        current_time++;
        if (processes[index].burst_time == 0)
        {
            processes[index].finish_time = current_time;
```

```
            processes[index].waiting_time = processes[index].start_time -
processes[index].arrival_time;
            processes[index].turnaround_time = processes[index].finish_time -
processes[index].arrival_time;
            processes[index].completed = 1;
            completed++;
        }
    }

    fprintf(fpout, "\n\n");
    printf("\n --> All processes have finished executing.\n");

    float avg_turnaround_time, avg_waiting_time;
    compute_avg_times(processes, n, &avg_turnaround_time, &avg_waiting_time);

    printf("\n --> Average turnaround time: %.2f\n", avg_turnaround_time);
    printf(" --> Average waiting time: %.2f\n", avg_waiting_time);

    // Get the output in output file
    fprintf(fpout, "Average turn-around time: %.2f\n", avg_turnaround_time);
    fprintf(fpout, "Average waiting time: %.2f\n\n", avg_waiting_time);
    fclose(fpout);
}

// Function to simulate the Longest Remaining Time First (LRTF) CPU scheduling
algorithm with preemption
void lrtf(struct process *processes, int n, char *outputFilename)
{
    FILE *fpout = fopen(outputFilename, "w");
    if (fpout == NULL)
    {
        printf("\n --> Error: Unable to open output file %s\n", outputFilename);
        exit(1);
    }

    // Print Gantt chart
    fprintf(fpout, "Gantt Chart:\n");

    printf("\n --> Running Longest Remaining Time First (LRTF) CPU scheduling
algorithm with preemption...\n");
    int current_time = 0, completed = 0, prev_process_index = -1;
    while (completed < n)
    {
        int max_burst_time = INT_MIN, index = -1;
        for (int i = 0; i < n; i++)
        {
            if (processes[i].arrival_time <= current_time &&
!processes[i].completed && processes[i].burst_time > max_burst_time)
            {
                max_burst_time = processes[i].burst_time;
```

```
                index = i;
            }
        }
        if (index == -1)
        {
            fprintf(fpout, "| Idle ");
            current_time++;
            continue;
        }
        if (prev_process_index != -1 && index != prev_process_index)
        {
            printf("Preempting P%d...\n", processes[prev_process_index].pid);
            processes[prev_process_index].preempted = 1;
        }

        fprintf(fpout, "| P%d ", processes[index].pid);
        printf("Processing P%d...\n", processes[index].pid);
        processes[index].start_time = (processes[index].start_time == -1) ?
current_time : processes[index].start_time;
        processes[index].burst_time--;
        current_time++;
        if (processes[index].burst_time == 0)
        {
            processes[index].finish_time = current_time;
            processes[index].waiting_time = processes[index].start_time -
processes[index].arrival_time;
            processes[index].turnaround_time = processes[index].finish_time -
processes[index].arrival_time;
            processes[index].completed = 1;
            completed++;
        }
        prev_process_index = index;
    }

    fprintf(fpout, "\n\n");
    printf("\n --> All processes have finished executing.\n");

    float avg_turnaround_time, avg_waiting_time;
    compute_avg_times(processes, n, &avg_turnaround_time, &avg_waiting_time);
    printf("\n --> Average turnaround time: %.2f\n", avg_turnaround_time);
    printf(" --> Average waiting time: %.2f\n", avg_waiting_time);

    // Get the output in output file
    fprintf(fpout, "Average turn-around time: %.2f\n", avg_turnaround_time);
    fprintf(fpout, "Average waiting time: %.2f\n\n", avg_waiting_time);
    fclose(fpout);
}

int main()
{
```

```
    char InputFilename[30];
    char OutputFilename[30];
    printf("\n|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
|||\n");
    printf("|||||               WELCOME TO THE SK CPU
SCHEDULER               |||||\n");
    printf("|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
|\n");

    int choice, n;

    int quantum = 0;
    do
    {
        printf("\n --> Select a CPU scheduling algorithm:\n");
        printf("1. First-Come First-Served (FCFS)\n");
        printf("2. Shortest Job First (SJF)\n");
        printf("3. Round Robin Scheduling (RR)\n");
        printf("4. Priority Scheduling (P)\n");
        printf("5. Shortest Remaining Time First (SRTF)\n");
        printf("6. Longest Remaining Time First (LRTF)\n");
        printf("0. Exit\n");
        printf("\n --> Enter your choice: ");
        scanf("%d", &choice);

        struct process *processes = NULL;
        if (choice)
        {
            printf("\n --> Enter the input file name (contains all information of
processes): ");
            scanf("%s", InputFilename);
            processes = read_input(InputFilename, &n);

            printf("\n --> Enter the output file name where you want to get the
cpu sheduling solution: ");
            scanf("%s", OutputFilename);
        }

        switch (choice)
        {
        case 0:
            printf("\n --> Exiting...\n");
            printf("\n|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
|||||||||||\n");
            printf("|||||           THANK YOU FOR USING SK CPU
SCHEDULER               |||||\n");
            printf("|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
|||||||||\n\n\n");
            break;
        case 1:
```

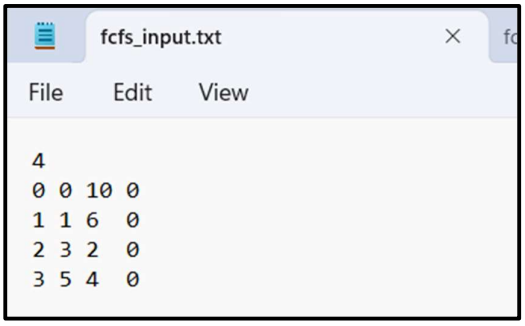
```
            fcfs(processes, n, OutputFilename);
            free(processes);
            break;
        case 2:
            sjf(processes, n, OutputFilename);
            free(processes);
            break;
        case 3:
            printf("\n --> Enter the quantum time for round robin sheduling: ");
            scanf("%d", &quantum);
            rr(processes, n, quantum, OutputFilename);
            free(processes);
            break;
        case 4:
            priority(processes, n, OutputFilename);
            free(processes);
            break;
        case 5:
            srtf(processes, n, OutputFilename);
            free(processes);
            break;
        case 6:
            lrtf(processes, n, OutputFilename);
            free(processes);
            break;
        default:
            printf("\n --> Invalid choice. Please try again...\n");
            break;
        }
    } while (choice != 0);

    return 0;
}
```

Input | Output :

(1) FCFS :

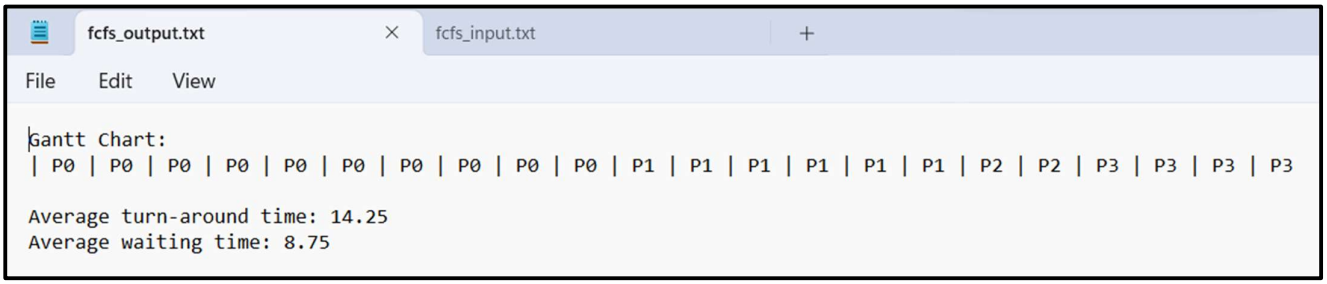
```
||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
|||||                WELCOME TO THE SK CPU SCHEDULER                    |||||
||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||

 --> Select a CPU scheduling algorithm:
1. First-Come First-Served (FCFS)
2. Shortest Job First (SJF)
3. Round Robin Scheduling (RR)
4. Priority Scheduling (P)
5. Shortest Remaining Time First (SRTF)
6. Longest Remaining Time First (LRTF)
0. Exit

 --> Enter your choice: 1

 --> Enter the input file name (contains all information of processes): fcfs_input.txt

 --> Enter the output file name where you want to get the cpu sheduling solution: fcfs_output.txt

 --> Running First-Come First-Served (FCFS) CPU scheduling algorithm...
Processing P0...
Processing P1...
Processing P2...
Processing P3...

 --> All processes have finished executing.

 --> Average turnaround time: 14.25
 --> Average waiting time: 8.75
```

Content of file ‘fcfs_input.txt’ :



The file ‘fcfs_output.txt’ was created automatically…

Content of file ‘fcfs_input.txt’ while running the code:



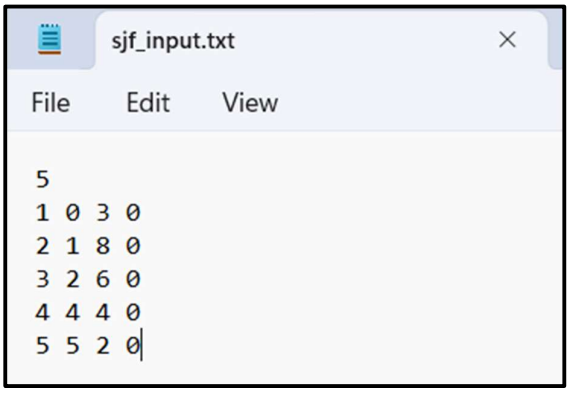
The file ‘input2.txt’ was not created so it will give error.

```
 --> Select a CPU scheduling algorithm:
1. First-Come First-Served (FCFS)
2. Shortest Job First (SJF)
3. Round Robin Scheduling (RR)
4. Priority Scheduling (P)
5. Shortest Remaining Time First (SRTF)
6. Longest Remaining Time First (LRTF)
0. Exit

 --> Enter your choice: 2

 --> Enter the input file name (contains all information of processes): input2.txt

 --> Error: Unable to open input file input2.txt
```

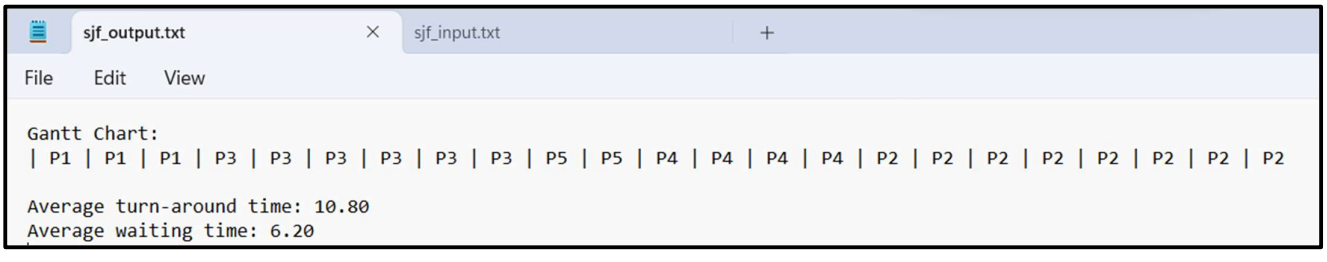
(2) SJF :

```
||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
|||||                WELCOME TO THE SK CPU SCHEDULER                    |||||
||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||

 --> Select a CPU scheduling algorithm:
1. First-Come First-Served (FCFS)
2. Shortest Job First (SJF)
3. Round Robin Scheduling (RR)
4. Priority Scheduling (P)
5. Shortest Remaining Time First (SRTF)
6. Longest Remaining Time First (LRTF)
0. Exit

 --> Enter your choice: 2

 --> Enter the input file name (contains all information of processes): sjf_input.txt

 --> Enter the output file name where you want to get the cpu sheduling solution: sjf_output.txt

 --> Running Shortest Job First (SJF) CPU scheduling algorithm...
Processing P1...
Processing P3...
Processing P5...
Processing P4...
Processing P2...

 --> All processes have finished executing.

 --> Average turnaround time: 10.80
 --> Average waiting time: 6.20
```

Content of file ‘sjf_input.txt’ :



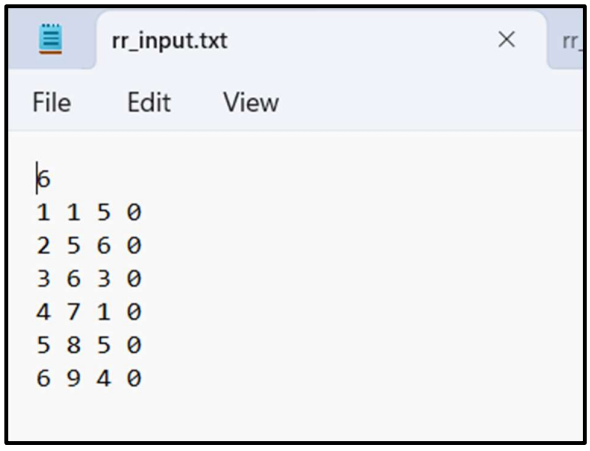
Content of file ‘sjf_output.txt’ :



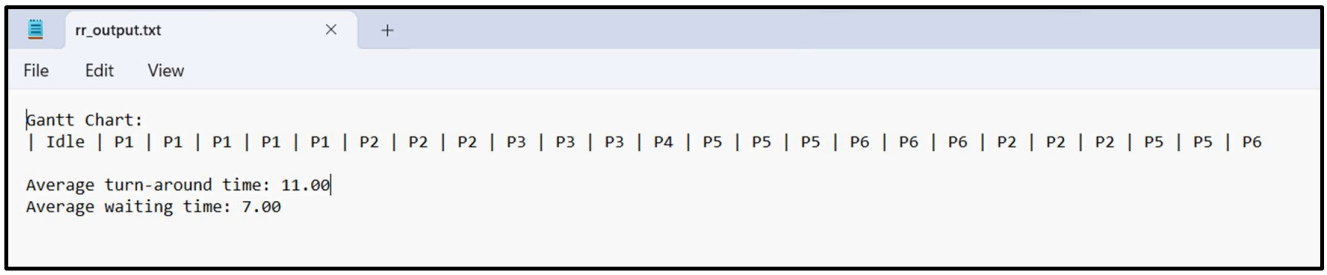
(3) RR :

```
||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
|||||                WELCOME TO THE SK CPU SCHEDULER                    |||||
||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||

 --> Select a CPU scheduling algorithm:
1. First-Come First-Served (FCFS)
2. Shortest Job First (SJF)
3. Round Robin Scheduling (RR)
4. Priority Scheduling (P)
5. Shortest Remaining Time First (SRTF)
6. Longest Remaining Time First (LRTF)
0. Exit

 --> Enter your choice: 3

 --> Enter the input file name (contains all information of processes): rr_input.txt

 --> Enter the output file name where you want to get the cpu sheduling solution: rr_output.txt

 --> Enter the quantum time for round robin sheduling: 3

 --> Running Round Robin (RR) CPU scheduling algorithm with quantum 3...
Processing P1...
Processing P1...
Processing P2...
Processing P3...
Processing P4...
Processing P5...
Processing P6...
Processing P2...
Processing P5...
Processing P6...

 --> All processes have finished executing.

 --> Average turnaround time: 11.00
 --> Average waiting time: 7.00
```

Content of file ‘rr_input.txt’ :



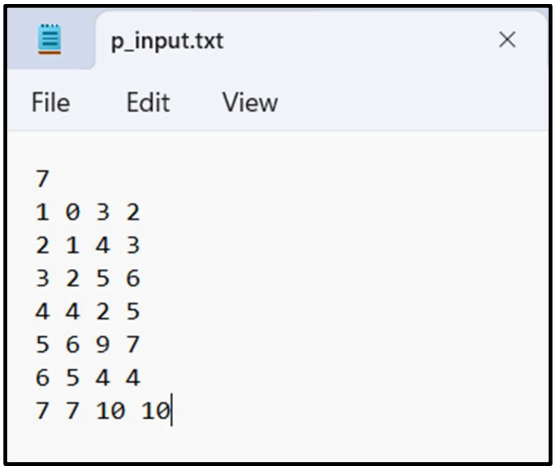
Content of file ‘rr_output.txt’ :



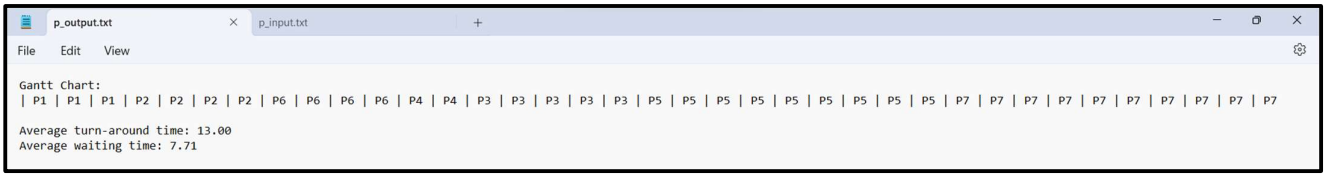
(4) Priority :

```
 --> Select a CPU scheduling algorithm:
1. First-Come First-Served (FCFS)
2. Shortest Job First (SJF)
3. Round Robin Scheduling (RR)
4. Priority Scheduling (P)
5. Shortest Remaining Time First (SRTF)
6. Longest Remaining Time First (LRTF)
0. Exit

 --> Enter your choice: 4

 --> Enter the input file name (contains all information of processes): p_input.txt

 --> Enter the output file name where you want to get the cpu sheduling solution: p_output.txt
Running Priority Scheduling CPU scheduling algorithm...
Processing P1...
Processing P2...
Processing P6...
Processing P4...
Processing P3...
Processing P5...
Processing P7...

 --> All processes have finished executing.

 --> Average turnaround time: 13.00
 --> Average waiting time: 7.71
```

Content of file ‘p_input.txt’ :



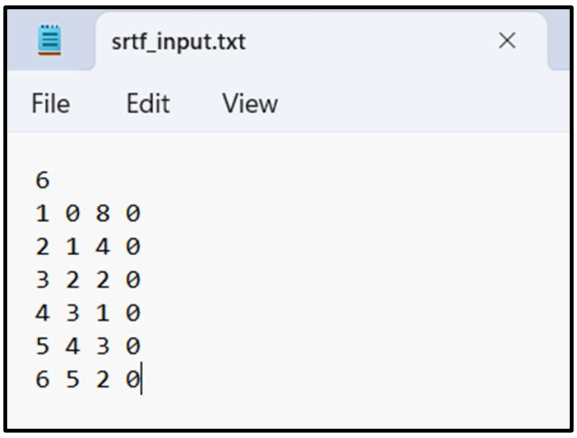
Content of file ‘p_output.txt’ :



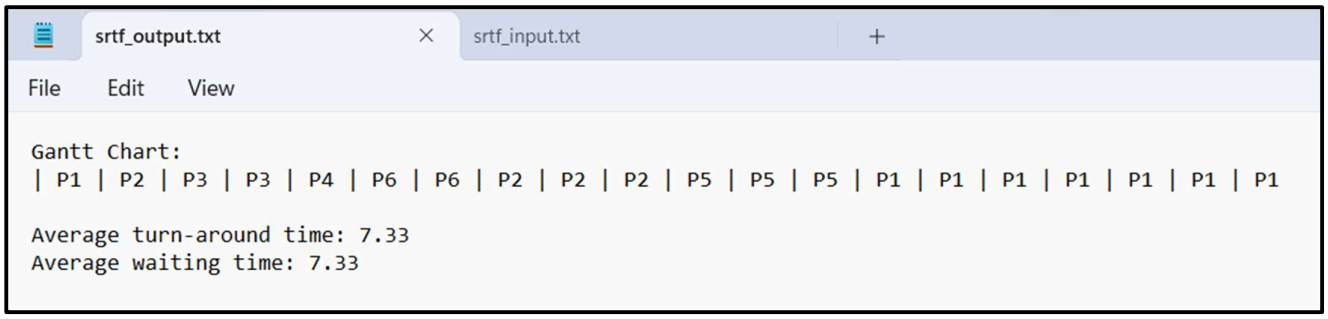
(5) SRTF :

```
 --> Select a CPU scheduling algorithm:
1. First-Come First-Served (FCFS)
2. Shortest Job First (SJF)
3. Round Robin Scheduling (RR)
4. Priority Scheduling (P)
5. Shortest Remaining Time First (SRTF)
6. Longest Remaining Time First (LRTF)
0. Exit

 --> Enter your choice: 5

 --> Enter the input file name (contains all information of processes): srtf_input.txt

 --> Enter the output file name where you want to get the cpu sheduling solution: srtf_output.txt

 --> Running Shortest Remaining Time First (SRTF) CPU scheduling algorithm...
Processing P1...
Processing P2...
Processing P3...
Processing P3...
Processing P4...
Processing P6...
Processing P6...
Processing P2...
Processing P2...
Processing P2...
Processing P5...
Processing P5...
Processing P5...
Processing P1...
Processing P1...
Processing P1...
Processing P1...
Processing P1...
Processing P1...
Processing P1...

 --> All processes have finished executing.

 --> Average turnaround time: 7.33
 --> Average waiting time: 7.33
```

Content of file ‘srtf_input.txt’ :



Content of file ‘srtf_output.txt’ :

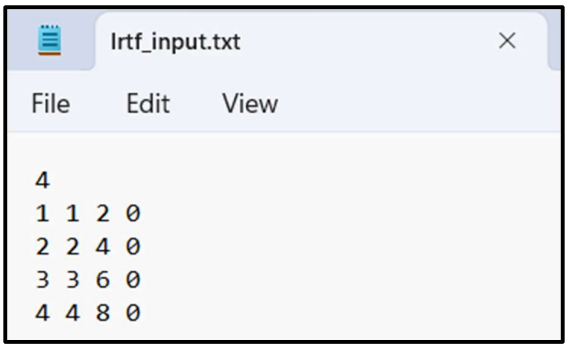


(6) LRTF :

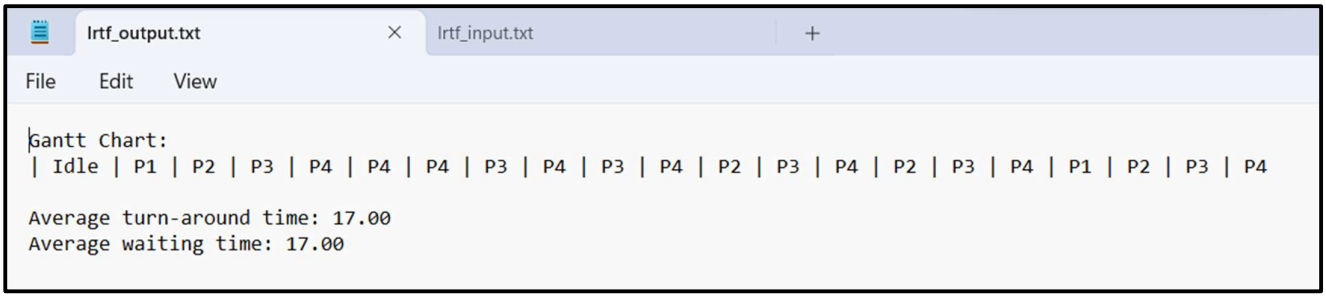
```
 --> Select a CPU scheduling algorithm:
1. First-Come First-Served (FCFS)
2. Shortest Job First (SJF)
3. Round Robin Scheduling (RR)
4. Priority Scheduling (P)
5. Shortest Remaining Time First (SRTF)
6. Longest Remaining Time First (LRTF)
0. Exit

 --> Enter your choice: 6

 --> Enter the input file name (contains all information of processes): lrtf_input.txt

 --> Enter the output file name where you want to get the cpu sheduling solution: lrtf_output.txt
```

```
 --> Running Longest Remaining Time First (LRTF) CPU scheduling algorithm with preemption...
Processing P1...
Preempting P1...
Processing P2...
Preempting P2...
Processing P3...
Preempting P3...
Processing P4...
Processing P4...
Processing P4...
Preempting P4...
Processing P3...
Preempting P3...
Processing P4...
Preempting P4...
Processing P3...
Preempting P3...
Processing P4...
Preempting P4...
Processing P2...
Preempting P2...
Processing P3...
Preempting P3...
Processing P4...
Preempting P4...
Processing P2...
Preempting P2...
Processing P3...
Preempting P3...
Processing P4...
Preempting P4...
Processing P1...
Preempting P1...
Processing P2...
Preempting P2...
Processing P3...
Preempting P3...
Processing P4...

 --> All processes have finished executing.

 --> Average turnaround time: 17.00
 --> Average waiting time: 17.00
```

Content of file 'lrtf_input.txt' :



Content of file 'lrtf_output.txt' :



"If CPU is the heart of the PC, then OS is the blood is of the PC..!!"

Signature of Teacher :

Thank you.