2CS403: OPERATING SYSTEM

INNOVATIVE ASSIGNMENT

CPU SCHEDULING ALGORITHMS

SEMESTER - IV

Prepared By:- 21BCE076: Keyaba Gohil
21BCE019: Ayushi Shah

Code-

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

#define MAX_PROCESSES 100

struct process {
    int pid;
    int arrival_time;
    int burst_time;
    int priority;
    int remaining_time;
    int TAT;
    int waiting_time;
    int is_completed;
    int completion_time;
};

//FCFS Code
void FCFS()
{
    struct process processes[MAX_PROCESSES];
    int num_processes = 0;

    FILE *input_file = fopen("FCFS.txt", "r");
    if (input_file == NULL) {
        printf("Error opening input file!\n");
        return ;
    }

    while (fscanf(input_file, "%d %d %d",
&processes[num_processes].pid,
&processes[num_processes].arrival_time,
&processes[num_processes].burst_time) == 3) {
        num_processes++;
    }
```

```
    fclose(input_file);

    for (int i = 0; i < num_processes - 1; i++) {
        for (int j = 0; j < num_processes - i - 1;
j++) {
            if (processes[j].arrival_time >
processes[j+1].arrival_time) {
                struct process temp = processes[j];
                processes[j] = processes[j+1];
                processes[j+1] = temp;
            }
        }
    }

    int current_time = 0;
    int total_waiting_time = 0;
    int total_turnaround_time = 0;
    int gantt_chart[MAX_PROCESSES * 2];
    int gc_index = 0;
    FILE *output_file = fopen("FCFSoutput.txt",
"w");
    for (int i = 0; i < num_processes; i++) {
        while (current_time <
processes[i].arrival_time) {
            gantt_chart[gc_index++] = -1;
            current_time++;
        }
        gantt_chart[gc_index++] = processes[i].pid;
        total_waiting_time += (current_time -
processes[i].arrival_time);
        total_turnaround_time +=(current_time -
processes[i].arrival_time +
processes[i].burst_time);
        current_time += processes[i].burst_time;

    }
```

```
    if(output_file==NULL)
    {
        printf("Error:Failed to open output
file.\n");
        return ;
    }
    printf("Gantt Chart for FCFS is: ");
    for(int k=0;k<gc_index;k++)
    {
        printf("P%d ",gantt_chart[k]);
    }
    printf("\n");


    fprintf(output_file,"Average turn-around time :
%.2f\n",(float)(total_turnaround_time)/num_processes
);
    fprintf(output_file,"Average waiting time :
%.2f\nGantt_chart:\n",(float)(total_waiting_time)/nu
m_processes);
    for(int k=0;k<gc_index;k++)
    {
        fprintf(output_file,"P%d  ",
gantt_chart[k]);
    }
    fclose(output_file);
}

//Priority scheduling code
void priority()
{
    struct process processes[MAX_PROCESSES];
    int num_processes = 0;

    FILE *input_file = fopen("Priority.txt", "r");
    if (input_file == NULL) {
        printf("Error opening input file!\n");
        return ;
```

```
    }

    while (fscanf(input_file, "%d %d %d",
&processes[num_processes].pid,
&processes[num_processes].burst_time,&processes[num_
processes].priority) == 3) {
        num_processes++;
    }

    fclose(input_file);

    for (int i = 0; i < num_processes - 1; i++) {
        for (int j = 0; j < num_processes - i - 1;
j++) {
            if (processes[j].priority >
processes[j+1].priority) {
                struct process temp = processes[j];
                processes[j] = processes[j+1];
                processes[j+1] = temp;
            }
        }
    }

    int current_time = 0;
    int total_waiting_time = 0;
    int total_turnaround_time = 0;
    int gantt_chart[MAX_PROCESSES * 2];
    int gc_index = 0;

    for (int i = 0; i < num_processes; i++) {

total_turnaround_time+=(current_time+processes[i].bu
rst_time);

        int temp=processes[i].burst_time;
        while (temp--) {
            current_time++;
        }
```

```
        gantt_chart[gc_index++] = processes[i].pid;


        if(i!=num_processes-1)
        total_waiting_time += (current_time);

    }

    FILE *output_file = fopen("Priorityoutput.txt",
"w");
    if(output_file==NULL)
    {
        printf("Error:Failed to open output
file.\n");
        return ;
    }
    printf("Gantt Chart for Priority Scheduling is:
");
    for(int k=0;k<gc_index;k++)
    {
        printf("P%d ",gantt_chart[k]);
    }
    printf("\n");
    fprintf(output_file,"Average turn-around time :
%.2f\n",(float)(total_turnaround_time)/num_processes
);
    fprintf(output_file,"Average waiting time :
%.2f\nGantt_chart:\n",(float)(total_waiting_time)/nu
m_processes);
    for(int k=0;k<gc_index;k++)
    {
        fprintf(output_file,"P%d  ",
gantt_chart[k]);
    }
    fclose(output_file);
}

//Round Robbin Scheduling Code
```

```
void RR()
{
    struct process processes[MAX_PROCESSES];
    struct process processes_queue[MAX_PROCESSES*2];
    int num_processes = 0;

    FILE *input_file = fopen("RoundRobin.txt", "r");
    if (input_file == NULL) {
        printf("Error opening input file!\n");
        return ;
    }

    while (fscanf(input_file, "%d %d %d",
&processes[num_processes].pid,
&processes[num_processes].arrival_time,
&processes[num_processes].burst_time) == 3) {

processes[num_processes].remaining_time=processes[nu
m_processes].burst_time;
        num_processes++;
    }

    fclose(input_file);
    long long int current_time = 0;
    int num_processes_left=num_processes;
    int total_waiting_time = 0;
    int total_turnaround_time = 0;
    int gantt_chart[MAX_PROCESSES * 2];
    int gc_index = 0;


    for (int i = 0; i < num_processes - 1; i++) {
        for (int j = 0; j < num_processes - i - 1;
j++) {
            if (processes[j].arrival_time >
processes[j+1].arrival_time) {
                struct process temp = processes[j];
                processes[j] = processes[j+1];
```

```
                processes[j+1] = temp;
            }
        }
    }
    FILE *output_file =
fopen("RoundRobinoutput.txt", "w");
    if(output_file==NULL)
    {
        printf("Error:Failed to open output
file.\n");
        return ;
    }
    int quanttime=6;
    int i=0;//pointer to the current process queue
    int j=0;//pointer to the end of the process
queue
    int process_pointer=0;
    int flag=0;//to add processes to the queue based
on the arrival time

    fprintf(output_file," PID\tBurst Time\t Arrival
Time\t TAT\t\t Waiting Time\n");
    do {
        current_time++;
        while (( process_pointer!=num_processes)&&
current_time >=
processes[process_pointer].arrival_time) {

processes_queue[j++]=processes[process_pointer++];
            // if(process_pointer==num_processes)
flag=1;
            }
        if( processes_queue[i].remaining_time==0 &&
j!=i+1){
            // printf("the value of i is
here:%d\n",i);
            i++;
            continue;
```

```
        }
        else{

            gantt_chart[gc_index++] =
processes_queue[i].pid;


if(processes_queue[i].remaining_time>quanttime) {
                current_time+=quanttime;

            }
            else

current_time+=processes_queue[i].remaining_time;


processes_queue[i].remaining_time-=quanttime;

            while ((
process_pointer!=num_processes)&& current_time >=
processes[process_pointer].arrival_time) {


processes_queue[j++]=processes[process_pointer++];
            // if(process_pointer==num_processes)
flag=1;
            }
            if(processes_queue[i].remaining_time<=0)
{
                processes_queue[i].remaining_time=0;

total_turnaround_time+=processes_queue[i].TAT;

total_waiting_time+=processes_queue[i].waiting_time;
                fprintf(output_file," %d\t\t
%d\t\t\t %d \t\t\t\t%d \t\t\t\t\t%d
\n",processes_queue[i].pid,processes_queue[i].burst_
```

```
time,processes_queue[i].arrival_time,processes_queue
[i].TAT,processes_queue[i].waiting_time);
                num_processes_left--;
            }
            else{

processes_queue[j++]=processes_queue[i];
            }
            i++;
        }

    }
    while(num_processes_left!=0);

    printf("Gantt Chart for Round Robin is: ");
    for(int k=0;k<gc_index;k++)
    {
        printf("P%d ",gantt_chart[k]);
    }
    printf("\n");
    fprintf(output_file,"Average turn-around time :
%.2f\n",(float)(total_turnaround_time)/num_processes
);
    fprintf(output_file,"Average waiting time :
%.2f\nGantt_chart:\n",(float)(total_waiting_time)/nu
m_processes);

    for(int k=0;k<gc_index;k++)
    {
        fprintf(output_file,"P%d  ",
gantt_chart[k]);
    }
    fclose(output_file);
}

//Shortest Job First Code
void SJF(){
```

```
    int n, time = 0, smallest, completed = 0,
total_turnaround_time = 0, total_waiting_time = 0;
    struct process processes[MAX_PROCESSES], temp;
    FILE *fp;

    fp = fopen("SJB.txt", "r");
    if (fp == NULL) {
        printf("Error opening file.\n");
        return;
    }

    fscanf(fp, "%d", &n);

    for (int i = 0; i < n; i++) {
        processes[i].pid = i+1;
        fscanf(fp, "%d %d",
&processes[i].arrival_time,
&processes[i].burst_time);
        processes[i].is_completed = 0;
    }

    fclose(fp);
    fp = fopen("SJBoutput.txt", "w");
    if (fp == NULL) {
        printf("Error opening file.\n");
        return;
    }
    // Sort processes by arrival time
    for (int i = 0; i < n; i++) {
        for (int j = i+1; j < n; j++) {
            if (processes[i].arrival_time >
processes[j].arrival_time) {
                temp = processes[i];
                processes[i] = processes[j];
                processes[j] = temp;
            }
        }
    }
```

```
    // Run SJF algorithm
     fprintf(fp,"Gantt Chart : \n");
    while (completed != n) {
        smallest = -1;
        for (int i = 0; i < n; i++) {
            if (processes[i].arrival_time <= time &&
processes[i].is_completed == 0) {
                if (smallest == -1 ||
processes[i].burst_time <
processes[smallest].burst_time) {
                    smallest = i;
                }
            }
        }


        if (smallest == -1) {
            time++;
        } else {
            processes[smallest].completion_time =
time + processes[smallest].burst_time;
            processes[smallest].TAT =
processes[smallest].completion_time -
processes[smallest].arrival_time;
            processes[smallest].waiting_time =
processes[smallest].TAT -
processes[smallest].burst_time;
            total_turnaround_time +=
processes[smallest].TAT;
            total_waiting_time +=
processes[smallest].waiting_time;
            processes[smallest].is_completed = 1;
            completed++;
            time =
processes[smallest].completion_time;
```

```
        for (int j = 0; j <
processes[smallest].burst_time; j++) {
            fprintf(fp,"P%d ",
processes[smallest].pid);
        }
        }
    }

    fprintf(fp,"\n");
    fprintf(fp,"Average turn-around time: %.2f\n",
(float)total_turnaround_time / n);
    fprintf(fp,"Average waiting time: %.2f\n",
(float)total_waiting_time / n);

    printf("Gantt Chart for SJF is: ");
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < processes[i].burst_time;
j++) {
            printf("P%d ", processes[i].pid);

        }
    }



    printf("\n");
    fclose(fp);
}

//Shortest Remaining Job First Code
void SRJF(){

    int n, time = 0, smallest, completed = 0,
total_turnaround_time = 0, total_waiting_time = 0;
    struct process processes[MAX_PROCESSES], temp;
    FILE *fp;

    fp = fopen("SRJF.txt", "r");
    if (fp == NULL) {
```

```
        printf("Error opening file.\n");
        return;
    }

    fscanf(fp, "%d", &n);
    for (int i = 0; i < n; i++) {
        fscanf(fp, "%d %d %d",&processes[i].pid,
&processes[i].arrival_time,
&processes[i].burst_time);

processes[i].remaining_time=processes[i].burst_time;
        //
printf("%d:\n",processes[i].remaining_time);
        processes[i].is_completed = 0;
    }

    fclose(fp);
    fp = fopen("SRJFoutput.txt", "w");
    if (fp == NULL) {
        printf("Error opening file.\n");
        return;
    }
    int gantt_chart[MAX_PROCESSES*2];

    fprintf(fp," PID\tBurst Time\tArrival
Time\tTAT\tWaiting Time\n");
    // Sort processes by arrival time
    for (int i = 0; i < n; i++) {
        for (int j = i+1; j < n; j++) {
            if (processes[i].arrival_time >
processes[j].arrival_time) {
                temp = processes[i];
                processes[i] = processes[j];
                processes[j] = temp;
            }
        }
    }
```

```
    // Run SRJF algorithm
    while (completed != n) {
        smallest = -1;
        for (int i = 0; i < n; i++) {
            if (processes[i].arrival_time <= time &&
processes[i].is_completed == 0) {
                if (smallest == -1 ||
processes[i].remaining_time <
processes[smallest].remaining_time) {
                    smallest = i;
                }
            }

        }


        if (smallest == -1) {
            time++;
        } else {
            processes[smallest].remaining_time-=1;

if(processes[smallest].remaining_time==0)
            {
                completed++;
                processes[smallest].is_completed=1;
                processes[smallest].completion_time
= (time+1);
                processes[smallest].TAT =
processes[smallest].completion_time -
processes[smallest].arrival_time;
                processes[smallest].waiting_time =
processes[smallest].TAT -
processes[smallest].burst_time;

                total_turnaround_time +=
processes[smallest].TAT;
                total_waiting_time +=
processes[smallest].waiting_time;
```

```
fprintf(fp,"%d\t\t\t%d\t\t\t%d\t\t\t%d\t\t\t%d\n",pr
ocesses[smallest].pid,processes[smallest].burst_time
,processes[smallest].arrival_time,processes[smallest
].TAT,processes[smallest].waiting_time);
            }

gantt_chart[time]=processes[smallest].pid;
            time++;
        }
    }

    fprintf(fp,"Average turn-around time: %.2f\n",
(float)total_turnaround_time / n);
    fprintf(fp,"Average waiting time: %.2f\n",
(float)total_waiting_time / n);

    printf("Gantt Chart for SRTF is: ");
    for (int j = 0; j <time; j++) {
        printf("P%d ", gantt_chart[j]);
    }
     fprintf(fp,"Gnatt Chart: \n");
    for (int j = 0; j <time; j++) {
        fprintf(fp,"P%d ", gantt_chart[j]);
    }

    printf("\n");
    fclose(fp);

}

//Longest Remaining Job First Code
void LRJF(){
    int n, time = 0, largest, completed = 0,
total_turnaround_time = 0, total_waiting_time = 0;
    struct process processes[MAX_PROCESSES], temp;
    FILE *fp;
```

```
    fp = fopen("LRTF.txt", "r");
    if (fp == NULL) {
        printf("Error opening file.\n");
        return;
    }

    fscanf(fp, "%d", &n);
    for (int i = 0; i < n; i++) {
        fscanf(fp, "%d %d %d",&processes[i].pid,
&processes[i].arrival_time,
&processes[i].burst_time);

processes[i].remaining_time=processes[i].burst_time;

        processes[i].is_completed = 0;
    }

    fclose(fp);
    fp = fopen("LRTFoutput.txt", "w");
    if (fp == NULL) {
        printf("Error opening file.\n");
        return;
    }
    int gantt_chart[MAX_PROCESSES*2];
    fprintf(fp," PID\tBurst Time\tArrival
Time\tTAT\tWaiting Time\n");
    // Sort processes by arrival time
    for (int i = 0; i < n; i++) {
        for (int j = i+1; j < n; j++) {
            if (processes[i].arrival_time >
processes[j].arrival_time) {
                temp = processes[i];
                processes[i] = processes[j];
                processes[j] = temp;
            }
        }
    }
```

```
    while (completed != n) {
        largest = -1;
        for (int i = 0; i < n; i++) {
            if (processes[i].arrival_time <= time &&
processes[i].is_completed == 0) {
                if (largest == -1 ||
processes[i].remaining_time >
processes[largest].remaining_time) {
                    largest = i;
                }
            }

        }


        if (largest == -1) {
            time++;
        } else {
            processes[largest].remaining_time-=1;
            if(processes[largest].remaining_time==0)
            {
                completed++;
                processes[largest].is_completed=1;
                processes[largest].completion_time =
(time+1);
                processes[largest].TAT =
processes[largest].completion_time -
processes[largest].arrival_time;
                processes[largest].waiting_time =
processes[largest].TAT -
processes[largest].burst_time;

                total_turnaround_time +=
processes[largest].TAT;
                total_waiting_time +=
processes[largest].waiting_time;

fprintf(fp,"%d\t\t\t%d\t\t\t%d\t\t\t%d\t\t\t%d\n",pr
```

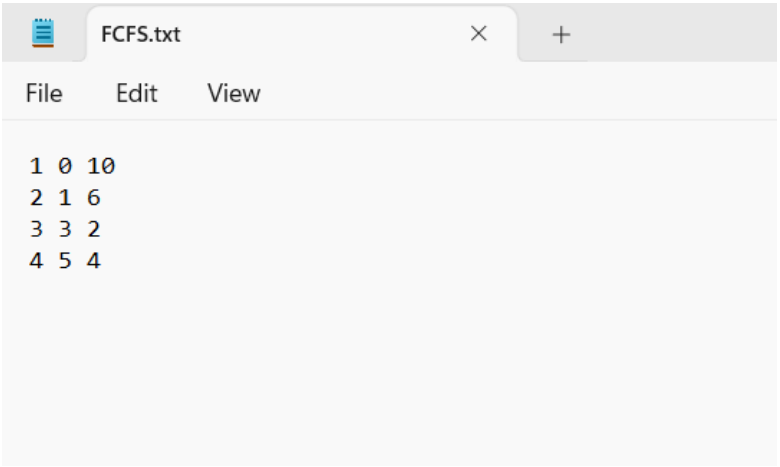
```
ocesses[largest].pid,processes[largest].burst_time,p
rocesses[largest].arrival_time,processes[largest].TA
T,processes[largest].waiting_time);
            }

gantt_chart[time]=processes[largest].pid;
            time++;
        }
    }


    fprintf(fp,"Average turn-around time: %.2f\n",
(float)total_turnaround_time / n);
    fprintf(fp,"Average waiting time: %.2f\n",
(float)total_waiting_time / n);

    printf("Gantt Chart for LRJF is: ");
     for (int j = 0; j <time; j++) {
        printf("P%d ", gantt_chart[j]);
    }
    fprintf(fp,"Gnatt Chart: \n");
    for (int j = 0; j <time; j++) {
        fprintf(fp,"P%d ", gantt_chart[j]);
    }

    printf("\n");
    fclose(fp);
}

//Main Function
int main() {
    int a;
    do
    {
        printf("\nSchdeuling algorithms :\n");
        printf("\t1. FCFS\n");
        printf("\t2. SJF\n");
        printf("\t3. RR \n");
```

```
        printf("\t4. Priority\n");
        printf("\t5. SRJF\n");
        printf("\t6. LRJF\n");
        printf("\t7. Exit\n");
        printf("Enter your choice(1-7): ");
        scanf("%d",&a);
        printf("\n");
        switch(a)
        {
            case 1:
            FCFS();
            break;
            case 2:
            SJF();
            break;
            case 3:
            RR();
            break;
            case 4:
            priority();
            break;
            case 5:
            SRJF();
            break;
            case 6:
            LRJF();
            break;
            case 7:
            return 0;
            break;
            default:
            printf("\t Choose from (1-7) only! ");
            break;
        }
    }while(a!=0);
}
```

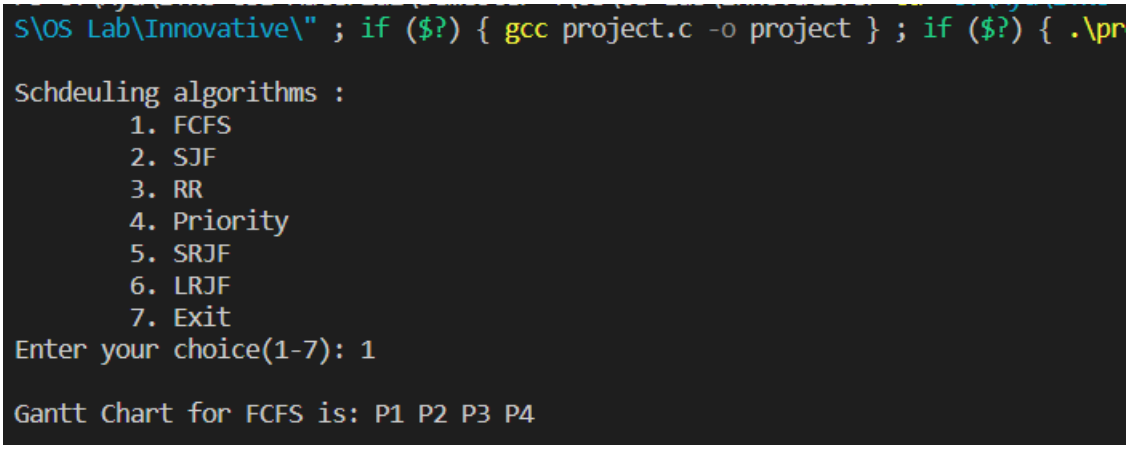
Screenshots of Output-

1) First-Come First-Served

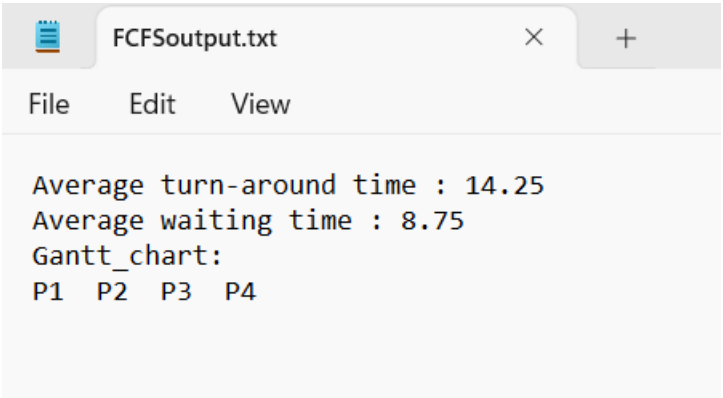
Input Text File:



C Program:

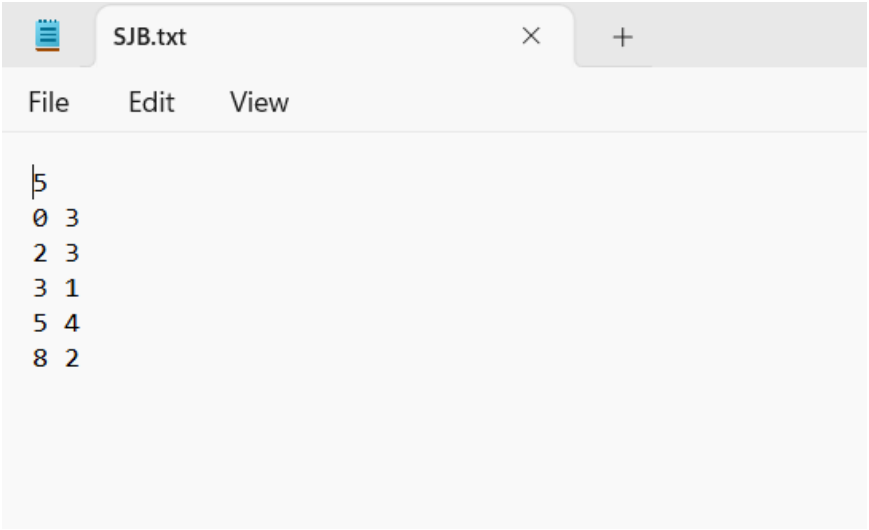


Output Text File:



2) Shortest Job First

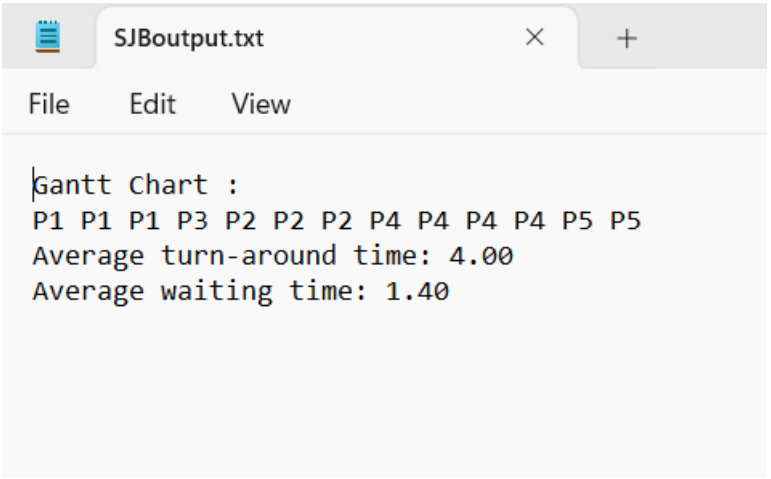
Input Text File:



C Program:

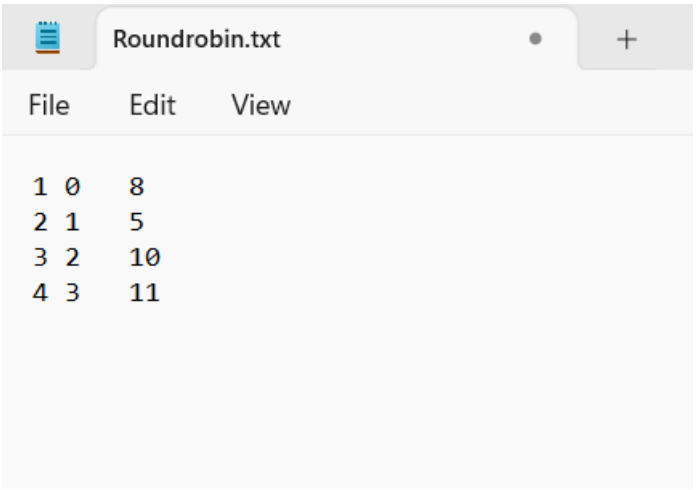
```
Schdeuling algorithms :
        1. FCFS
        2. SJF
        3. RR
        4. Priority
        5. SRJF
        6. LRJF
        7. Exit
Enter your choice(1-7): 2

Gantt Chart for SJF is: P1 P1 P1 P2 P2 P2 P3 P4 P4 P4 P4 P5 P5
```

Output Text File:



3) Round Robbin

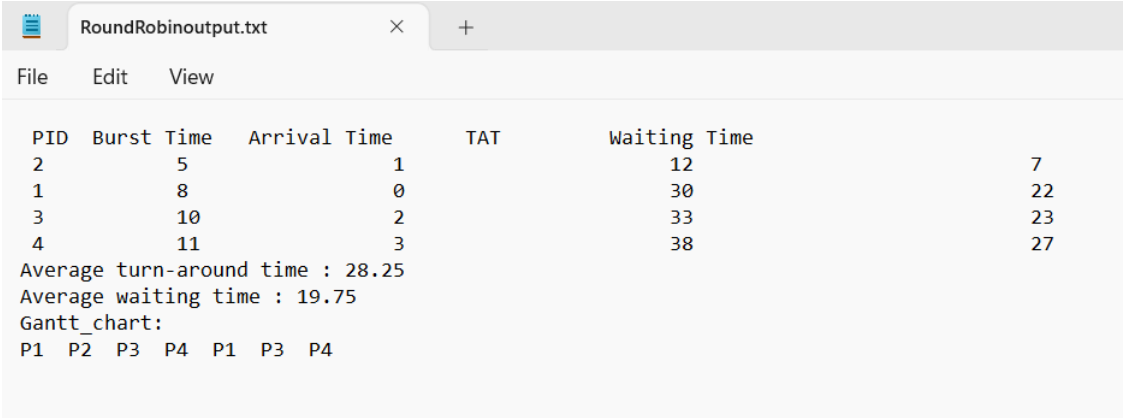
Input Text File:



C Program:

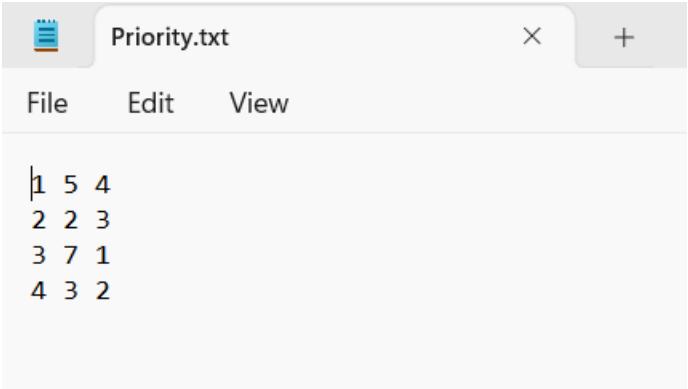
```
Schdeuling algorithms :
        1. FCFS
        2. SJF
        3. RR
        4. Priority
        5. SRJF
        6. LRJF
        7. Exit
Enter your choice(1-7): 3

Gantt Chart for Round Robin is: P1 P2 P3 P4 P1 P3 P4
```

Output Text File:



4) Priority Scheduling

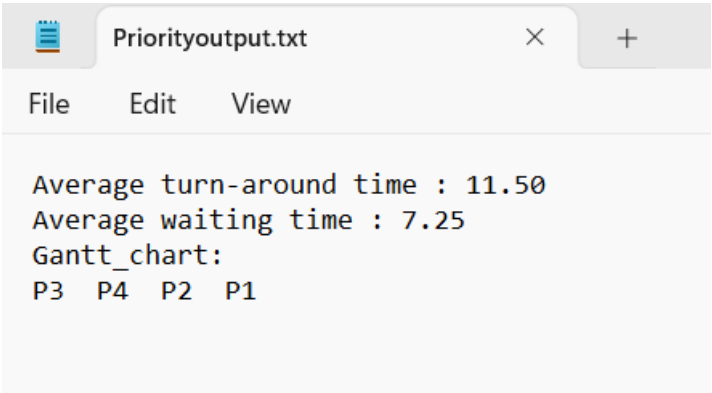
Input Text File:



C Program:

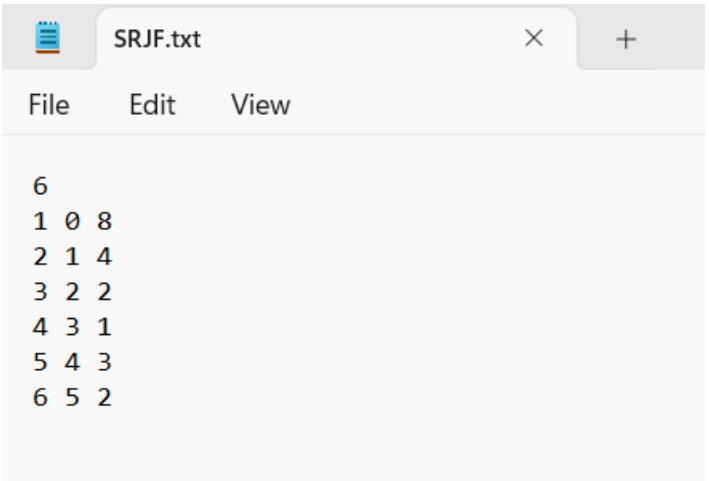
```
Schdeuling algorithms :
        1. FCFS
        2. SJF
        3. RR
        4. Priority
        5. SRJF
        6. LRJF
        7. Exit
Enter your choice(1-7): 4

Gantt Chart for Priority Scheduling is: P3 P4 P2 P1
```

Output Text File:



5) Shortest Remaining Time First

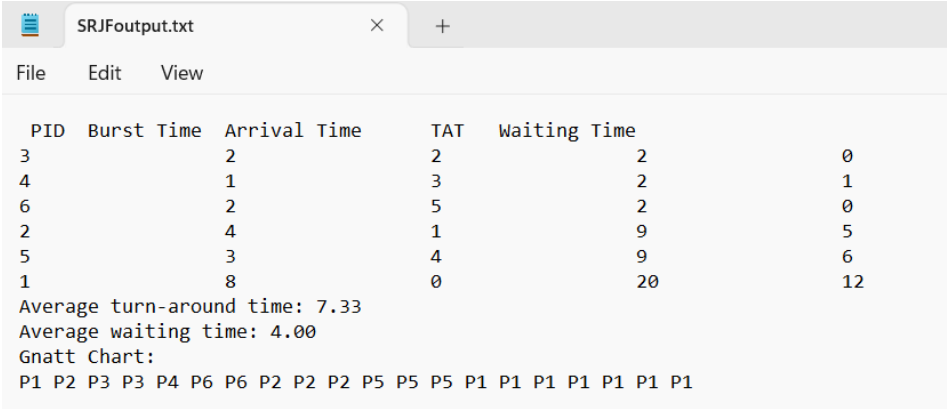
Input Text File:



C Program:

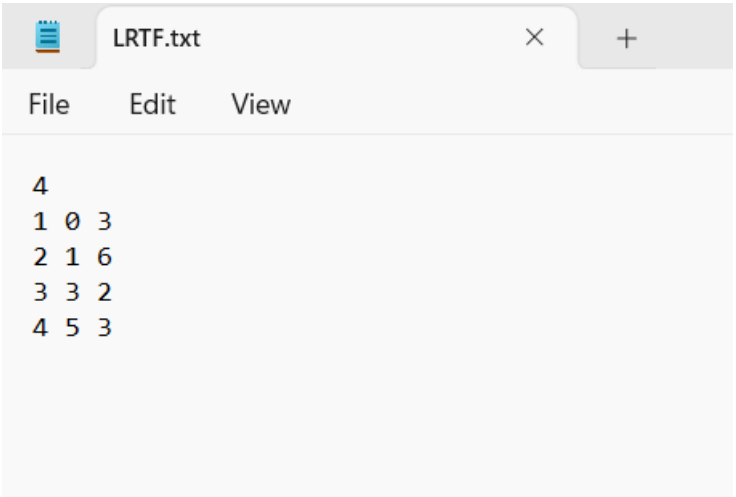
```
Schdeuling algorithms :
        1. FCFS
        2. SJF
        3. RR
        4. Priority
        5. SRJF
        6. LRJF
        7. Exit
Enter your choice(1-7): 5

Gantt Chart for SRTF is: P1 P2 P3 P3 P4 P6 P6 P2 P2 P2 P5 P5 P5 P1 P1 P1 P1 P1 P1 P1
```

Output Text File:



6) Longest Remaining time first

Input Text File:



C Program:

```
Schdeuling algorithms :
        1. FCFS
        2. SJF
        3. RR
        4. Priority
        5. SRJF
        6. LRJF
        7. Exit
Enter your choice(1-7): 6

Gantt Chart for LRJF is: P1 P2 P2 P2 P2 P4 P1 P2 P3 P4 P1 P2 P3 P4

Schdeuling algorithms :
        1. FCFS
        2. SJF
        3. RR
        4. Priority
        5. SRJF
        6. LRJF
        7. Exit
Enter your choice(1-7): 7
```

Output Text File: