**ASSIGNMENT – 1**

CPU Scheduling

Name – Ashutosh Soni

Id – 2018ucp1505

**Question:** Simplified version of CPU Scheduling. (We shall be dealing with only one CPU burst). Write a C

program (preferably for Linux gcc compiler) to simulate CPU scheduling. Following CPU scheduling

mechanisms need to be implemented:

1. FCFS

2. SJF

3. Priority

4. MLFQ

5. Round Robin

6. Lottery (proportional share)

The process traces should be read from a file called "process.dat". Format for this file is as follows:

<number of processes> <pid> <arrival time> <priority> <share> <burst>

......

<pid> <arrival time> <priority> <share> <burst>

So first line describes number of processes (say N).

Each of next N lines describe process id <pid>, time this process is put into ready queue <arrival time>. <priority> is the priority assigned to a process. A lower value means higher priority. <share> means number of tickets assigned in lottery (proportional share) scheduling.

We are assuming one CPU.

<burst> shall be an integer value between 1 - 20

For Round Robin scheduling, quantum shall be one.

For MLFQ scheduling, we shall assume that there are three queues - high, medium and low priority. Also process priority (except for those in high priority queue) is incremented every 10 cycles.

For priority queue, both preemptive and non-preemptive versions need to be implemented.

In lottery scheduling, it can be assumed that system is aware of total number of tickets (irrespective of process's arrival time).

Output should be process wise Gantt chart, CPU wise Gantt chart

And

Turnaround Time: process-wise, total and average

Waiting Time: process-wise, total and average

Response Time: process-wise, total and average

**Answer:**

**1) FCFS**

**Program for FCFS CPU scheduling mechanism:**

#include<stdio.h>

#include<stdlib.h>

// Structure of process

typedef struct process{

char name[5];

int pid;

int arrival\_time;

int priority;

int share;

int burst\_time;

int waiting\_time;

int response\_time;

int turn\_around\_time;

}processes;

// Sorting the process according to arrival time

void arrival\_sort(processes temp[],int n){

processes t;

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

for(int j=0;j<n-i;j++){

if(temp[j].arrival\_time >temp[j+1].arrival\_time){

t=temp[j];

temp[j]=temp[j+1];

temp[j+1]=t;

}

}

}

}

// Print Table

void print\_table(processes p[], int n)

{

int i;

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

printf("| Pname | PID | Arrival Time | Burst Time | Waiting Time | Turnaround Time | Response Time |\n");

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

for(i=0; i<n; i++) {

printf("| %s | %d | %d | %d | %d | %d | %d |\n",

p[i].name,p[i].pid,p[i].arrival\_time,p[i].burst\_time,p[i].waiting\_time,p[i].turn\_around\_time,p[i].response\_time);

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

}

}

// Printing gantt chart

void print\_gantt\_chart(processes p[], int n)

{

int i, j;

// print top bar

printf(" ");

for(i=0; i<n; i++) {

for(j=0; j<p[i].burst\_time; j++) printf("--");

printf(" ");

}

printf("\n|");

// printing process id in the middle

for(i=0; i<n; i++) {

for(j=0; j<p[i].burst\_time - 1; j++) printf(" ");

printf("%s", p[i].name);

for(j=0; j<p[i].burst\_time - 1; j++) printf(" ");

printf("|");

}

printf("\n ");

// printing bottom bar

for(i=0; i<n; i++) {

for(j=0; j<p[i].burst\_time; j++) printf("--");

printf(" ");

}

printf("\n");

// printing the time line

int val =0;

printf("0");

for(i=1; i<=n; i++) {

val =val+ p[i-1].burst\_time;

for(j=0; j<p[i-1].burst\_time; j++) printf(" ");

// if(p[i].turn\_around\_time > 9) printf("\b"); // backspace : remove 1 space

printf("%d", val);

}

printf("\n");

}

// Implementation of FCFS Aldorithm

void FIFS(processes P[],int n){

processes temp[n];

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

temp[i]=P[i];

}

// Sort according to arrival\_time

arrival\_sort(temp,n);

// Calculating waiting ,turnaround time and response time

int sum\_waiting=0, sum\_turnaround=0,sum\_response=0;

sum\_waiting=temp[0].waiting\_time=0;

temp[0].turn\_around\_time=temp[0].burst\_time - temp[0].arrival\_time;

sum\_turnaround = temp[0].turn\_around\_time;

sum\_response=sum\_waiting;

for(int i=1;i<n;i++){

temp[i].waiting\_time=(temp[i-1].burst\_time + temp[i-1].arrival\_time+temp[i-1].waiting\_time) - temp[i].arrival\_time;

temp[i].turn\_around\_time = (temp[i].waiting\_time+temp[i].burst\_time);

temp[i].response\_time= temp[i].waiting\_time;

sum\_waiting += temp[i].waiting\_time;

sum\_turnaround += temp[i].turn\_around\_time;

sum\_response =sum\_waiting;

}

// Printing table

printf("Table showing info about the processes: \n");

print\_table(temp,n);

// Calculating Average waiting, Average turnaround and average response time

float average\_waiting\_time, average\_turnaround\_time, average\_response\_time;

average\_waiting\_time = (float)sum\_waiting/n;

average\_turnaround\_time = (float)sum\_turnaround/n;

average\_response\_time = (float)sum\_response/n;

printf("\n");

printf("Average waiting time : %f\n",average\_waiting\_time );

printf("Average turnaround time : %f\n",average\_turnaround\_time );

printf("Average response time: %f\n",average\_response\_time );

printf("\n");

// Printing CPU Gantt chart

printf("Gantt chart of CPU for these processes: \n");

print\_gantt\_chart(temp,n);

printf("\n");

}

int main(int args,char \*argv[]){

FILE \*fp = fopen("process.dat","r");

if(fp==NULL){

printf("No such file exists.. Unable to open the file....\n");

exit(-1);

}

// Taking number of process thorugh file

int n;

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

processes P[n];

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

fscanf(fp,"%s",P[i].name);

fscanf(fp,"%d",&P[i].pid);

fscanf(fp,"%d",&P[i].arrival\_time);

fscanf(fp,"%d",&P[i].priority);

fscanf(fp,"%d",&P[i].share);

fscanf(fp,"%d",&P[i].burst\_time);

P[i].waiting\_time = P[i].response\_time = P[i].turn\_around\_time = 0;

}

// FIFS Algorithm

FIFS(P,n);

fclose(fp);

return 0;

}

**2) SJF**

**Program for SJF CPU scheduling mechanism:**

// Non preemptive approach

#include<stdio.h>

#include<stdlib.h>

// Structure of process

typedef struct process{

char name[5];

int pid;

int arrival\_time;

int priority;

int share;

int burst\_time;

int waiting\_time;

int response\_time;

int turn\_around\_time;

}processes;

// Sorting the process according to arrival time

void arrival\_sort(processes temp[],int n){

processes t;

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

for(int j=0;j<n-i;j++){

if(temp[j].arrival\_time >temp[j+1].arrival\_time){

t=temp[j];

temp[j]=temp[j+1];

temp[j+1]=t;

}

}

}

}

// Print Table

void print\_table(processes p[], int n)

{

int i;

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

printf("| Pname | PID | Arrival Time | Burst Time | Waiting Time | Turnaround Time | Response Time |\n");

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

for(i=0; i<n; i++) {

printf("| %s | %d | %d | %d | %d | %d | %d |\n",

p[i].name,p[i].pid,p[i].arrival\_time,p[i].burst\_time,p[i].waiting\_time,p[i].turn\_around\_time,p[i].response\_time);

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

}

}

// Printing gantt chart

void print\_gantt\_chart(processes p[], int n)

{

int i, j;

// print top bar

printf(" ");

for(i=0; i<n; i++) {

for(j=0; j<p[i].burst\_time; j++) printf("--");

printf(" ");

}

printf("\n|");

// printing process id in the middle

for(i=0; i<n; i++) {

for(j=0; j<p[i].burst\_time - 1; j++) printf(" ");

printf("%s", p[i].name);

for(j=0; j<p[i].burst\_time - 1; j++) printf(" ");

printf("|");

}

printf("\n ");

// printing bottom bar

for(i=0; i<n; i++) {

for(j=0; j<p[i].burst\_time; j++) printf("--");

printf(" ");

}

printf("\n");

// printing the time line

int val =0;

printf("0");

for(i=1; i<=n; i++) {

val =val+ p[i-1].burst\_time;

for(j=0; j<p[i-1].burst\_time; j++) printf(" ");

// if(p[i].turn\_around\_time > 9) printf("\b"); // backspace : remove 1 space

printf("%d", val);

}

printf("\n");

}

// Algorithm of Short Job First

void SJF(processes P[],int n){

processes temp[n];

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

temp[i]=P[i];

}

// Sorting on the basis of arrival time

arrival\_sort(temp,n);

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

printf("%s ",temp[i].name );

}

printf("\n");

// Sorting on the basis of minimum job

processes t;

for(int i=2;i<n;i++){

for(int j=1;j<n-i+1;j++){

if(temp[j].burst\_time > temp[j+1].burst\_time){

t = temp[j];

temp[j] = temp[j+1];

temp[j+1] = t;

}

}

}

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

printf("%s ",temp[i].name );

}

int sum\_waiting=0 , sum\_turnaround=0, sum\_response=0;

//Calculating waiting, turnaround and response time

sum\_waiting = temp[0].waiting\_time = 0;

sum\_turnaround = temp[0].turn\_around\_time = temp[0].burst\_time - temp[0].arrival\_time;

sum\_response = sum\_waiting;

for(int i=1;i<n;i++){

temp[i].waiting\_time = (temp[i-1].burst\_time + temp[i-1].arrival\_time+temp[i-1].waiting\_time) - temp[i].arrival\_time;

temp[i].turn\_around\_time = temp[i].waiting\_time+ temp[i].burst\_time;

temp[i].response\_time = temp[i].waiting\_time;

sum\_waiting += temp[i].waiting\_time;

sum\_turnaround +=temp[i].turn\_around\_time;

sum\_response=sum\_waiting;

}

// Printing table

printf("Table showing info about the processes: \n");

print\_table(temp,n);

// Calculating Average waiting , Average turnaround and average response time

float average\_waiting\_time, average\_turnaround\_time, average\_response\_time;

average\_waiting\_time = (float)sum\_waiting/n;

average\_turnaround\_time = (float)sum\_turnaround/n;

average\_response\_time = (float)sum\_response/n;

printf("\n");

printf("Average waiting time : %f\n",average\_waiting\_time );

printf("Average turnaround time : %f\n",average\_turnaround\_time );

printf("Average response time: %f\n",average\_response\_time );

printf("\n");

// Printing CPU Gantt chart

printf("Gantt chart of CPU for these processes: \n");

print\_gantt\_chart(temp,n);

printf("\n");

}

int main(int args,char \*argv[]){

FILE \*fp = fopen("process.dat","r");

if(fp==NULL){

printf("No such file exists.. Unable to open the file....\n");

exit(-1);

}

// Taking number of process thorugh file

int n;

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

processes P[n];

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

fscanf(fp,"%s",P[i].name);

fscanf(fp,"%d",&P[i].pid);

fscanf(fp,"%d",&P[i].arrival\_time);

fscanf(fp,"%d",&P[i].priority);

fscanf(fp,"%d",&P[i].share);

fscanf(fp,"%d",&P[i].burst\_time);

P[i].waiting\_time = P[i].response\_time = P[i].turn\_around\_time = 0;

}

fclose(fp);

// SJF Algorithm

SJF(P,n);

return 0;

}

**3) Priority**

**Program for Priority CPU scheduling mechanism:**

// Priority Scheduling

#include<stdio.h>

#include<stdlib.h>

// Structure of process

typedef struct process{

char name[5];

int pid;

int arrival\_time;

int priority;

int share;

int burst\_time;

int waiting\_time;

int response\_time;

int turn\_around\_time;

int flag;

}processes;

// Sorting the process according to arrival time

void arrival\_sort(processes temp[],int n){

processes t;

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

for(int j=0;j<n-i;j++){

if(temp[j].arrival\_time >temp[j+1].arrival\_time){

t=temp[j];

temp[j]=temp[j+1];

temp[j+1]=t;

}

}

}

}

// Print Table

void print\_table(processes p[], int n)

{

int i;

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

printf("| Pname | PID | Arrival Time | Burst Time | Priority | Waiting Time | Turnaround Time | Response Time |\n");

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

for(i=0; i<n; i++) {

printf("| %s | %d | %d | %d | %d | %d | %d | %d |\n",

p[i].name,p[i].pid,p[i].arrival\_time,p[i].burst\_time,p[i].priority,p[i].waiting\_time,p[i].turn\_around\_time,p[i].response\_time);

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

}

}

// Printing gantt chart

void print\_gantt\_chart(processes p[], int n)

{

int i, j;

// print top bar

printf(" ");

for(i=0; i<n; i++) {

for(j=0; j<p[i].burst\_time; j++) printf("--");

printf(" ");

}

printf("\n|");

// printing process id in the middle

for(i=0; i<n; i++) {

for(j=0; j<p[i].burst\_time - 1; j++) printf(" ");

printf("%s", p[i].name);

for(j=0; j<p[i].burst\_time - 1; j++) printf(" ");

printf("|");

}

printf("\n ");

// printing bottom bar

for(i=0; i<n; i++) {

for(j=0; j<p[i].burst\_time; j++) printf("--");

printf(" ");

}

printf("\n");

// printing the time line

int val =0;

printf("0");

for(i=1; i<=n; i++) {

val =val+ p[i-1].burst\_time;

for(j=0; j<p[i-1].burst\_time; j++) printf(" ");

// if(p[i].turn\_around\_time > 9) printf("\b"); // backspace : remove 1 space

printf("%d", val);

}

printf("\n");

}

// Priority Scheduling pre-emptive approach

void PRT\_P(processes P[],int n){

int t\_total =0; // Total time of CPU burst

processes temp[n];

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

temp[i] = P[i];

t\_total += temp[i].burst\_time;

temp[i].response\_time=-1;

}

// Sorting on the basis of arrival time

arrival\_sort(temp,n);

int burst[n];

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

burst[i]=temp[i].burst\_time;

}

int tcurr=0; // Current time

int i=0,j=0;

int min\_pr; // Minimum priority

int sum\_waiting=0,sum\_turnaround=0,sum\_response=0;

printf("\n Gantt chart of CPU for these processes: \n\n %d %s",i,temp[i].name);

for(tcurr=0;tcurr<t\_total;tcurr++)

{

if(burst[i] > 0 && temp[i].arrival\_time <= tcurr)

burst[i]--;

if(i!=j)

printf(" %d %s",tcurr,temp[i].name);

if(burst[i]<=0 && temp[i].flag != 1)

{

temp[i].flag = 1;

temp[i].waiting\_time = (tcurr+1) - temp[i].burst\_time - temp[i].arrival\_time;

temp[i].turn\_around\_time = (tcurr+1) - temp[i].arrival\_time;

sum\_waiting+=temp[i].waiting\_time;

sum\_turnaround+=temp[i].turn\_around\_time;

}

if(temp[i].response\_time == -1){

temp[i].response\_time = tcurr- temp[i].arrival\_time;

sum\_response +=temp[i].response\_time;

}

j=i;

min\_pr = 999;

for(int x=0;x<n;x++){

if(temp[x].arrival\_time <= (tcurr+1) && temp[x].flag != 1){

if(min\_pr != temp[x].priority && min\_pr > temp[x].priority){

min\_pr = temp[x].priority;

i=x;

}

}

}

}

printf(" %d",tcurr);

printf("\n");

printf("\n");

printf("\n");

printf("Table showing info about the processes: \n");

print\_table(temp,n);

// Calculating Average waiting , Average turnaround and average response time

float average\_waiting\_time, average\_turnaround\_time, average\_response\_time;

average\_waiting\_time = (float)sum\_waiting/n;

average\_turnaround\_time = (float)sum\_turnaround/n;

average\_response\_time = (float)sum\_response/n;

printf("\n");

printf("Average waiting time : %f\n",average\_waiting\_time );

printf("Average turnaround time : %f\n",average\_turnaround\_time );

printf("Average response time: %f\n",average\_response\_time );

printf("\n");

}

// Priority Scheduling using non-pre-emptive approach

void PRT\_NP(processes P[],int n){

processes temp[n];

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

temp[i]=P[i];

}

// Sorting on the basis of arrival time

arrival\_sort(P,n);

// Now Sorting on the basis if priority

processes t;

for(int i=2;i<n;i++){

for(int j=1;j<n-i+1;j++){

if(temp[j].priority > temp[j+1].priority){

t=temp[j];

temp[j]=temp[j+1];

temp[j+1]=t;

}

}

}

int sum\_waiting=0, sum\_turnaround=0, sum\_response=0;

sum\_waiting = temp[0].waiting\_time =0;

sum\_turnaround = temp[0].turn\_around\_time = temp[0].burst\_time - temp[0].arrival\_time;

sum\_response = sum\_waiting;

for(int i=1;i<n;i++){

temp[i].waiting\_time = (temp[i-1].burst\_time + temp[i-1].arrival\_time+temp[i-1].waiting\_time) - temp[i].arrival\_time;

temp[i].turn\_around\_time = temp[i].waiting\_time+ temp[i].burst\_time;

temp[i].response\_time = temp[i].waiting\_time;

sum\_waiting += temp[i].waiting\_time;

sum\_turnaround +=temp[i].turn\_around\_time;

sum\_response=sum\_waiting;

}

printf("Table showing info about the processes: \n");

print\_table(temp,n);

// Calculating Average waiting , Average turnaround and average response time

float average\_waiting\_time, average\_turnaround\_time, average\_response\_time;

average\_waiting\_time = (float)sum\_waiting/n;

average\_turnaround\_time = (float)sum\_turnaround/n;

average\_response\_time = (float)sum\_response/n;

printf("\n");

printf("Average waiting time : %f\n",average\_waiting\_time );

printf("Average turnaround time : %f\n",average\_turnaround\_time );

printf("Average response time: %f\n",average\_response\_time );

printf("\n");

// Printing CPU Gantt chart

printf("Gantt chart of CPU for these processes: \n");

print\_gantt\_chart(temp,n);

printf("\n");

}

int main(int args,char \*argv[]){

FILE \*fp = fopen("process.dat","r");

if(fp==NULL){

printf("No such file exists.. Unable to open the file....\n");

exit(-1);

}

// Taking number of process thorugh file

int n;

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

processes P[n];

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

fscanf(fp,"%s",P[i].name);

fscanf(fp,"%d",&P[i].pid);

fscanf(fp,"%d",&P[i].arrival\_time);

fscanf(fp,"%d",&P[i].priority);

fscanf(fp,"%d",&P[i].share);

fscanf(fp,"%d",&P[i].burst\_time);

P[i].waiting\_time = P[i].response\_time = P[i].turn\_around\_time = 0;

}

fclose(fp);

// SJF Algorithm

printf("\nFrom pre-emptive Priority Scheduling: \n");

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

PRT\_P(P,n);

printf("From non-pre-emptive Priority Scheduling: \n");

printf("\n");

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

PRT\_NP(P,n);

return 0;

}

**4) MLFQ**

**Program for MLFQ CPU scheduling mechanism:**

// Non preemptive approach

#include<stdio.h>

#include<stdlib.h>

// Structure of process

typedef struct process{

char name[5];

int pid;

int arrival\_time;

int priority;

int share;

int burst\_time;

int waiting\_time;

int response\_time;

int turn\_around\_time;

int flag;

}processes;

// Sorting the process according to arrival time

void arrival\_sort(processes temp[],int n){

processes t;

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

for(int j=0;j<n-i;j++){

if(temp[j].arrival\_time >temp[j+1].arrival\_time){

t=temp[j];

temp[j]=temp[j+1];

temp[j+1]=t;

}

}

}

}

// Print Table

void print\_table(processes p[], int n)

{

int i;

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

printf("| Pname | PID | Arrival Time | Burst Time | Waiting Time | Turnaround Time | Response Time |\n");

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

for(i=0; i<n; i++) {

printf("| %s | %d | %d | %d | %d | %d | %d |\n",

p[i].name,p[i].pid,p[i].arrival\_time,p[i].burst\_time,p[i].waiting\_time,p[i].turn\_around\_time,p[i].response\_time);

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

}

}

// Printing gantt chart

void print\_gantt\_chart(processes p[], int n)

{

int i, j;

// print top bar

printf(" ");

for(i=0; i<n; i++) {

for(j=0; j<p[i].burst\_time; j++) printf("--");

printf(" ");

}

printf("\n|");

// printing process id in the middle

for(i=0; i<n; i++) {

for(j=0; j<p[i].burst\_time - 1; j++) printf(" ");

printf("%s", p[i].name);

for(j=0; j<p[i].burst\_time - 1; j++) printf(" ");

printf("|");

}

printf("\n ");

// printing bottom bar

for(i=0; i<n; i++) {

for(j=0; j<p[i].burst\_time; j++) printf("--");

printf(" ");

}

printf("\n");

// printing the time line

int val =0;

printf("0");

for(i=1; i<=n; i++) {

val =val+ p[i-1].burst\_time;

for(j=0; j<p[i-1].burst\_time; j++) printf(" ");

// if(p[i].turn\_around\_time > 9) printf("\b"); // backspace : remove 1 space

printf("%d", val);

}

printf("\n");

}

// Algorithm for Round Robin scheduling

void RR(processes P[],int n){

processes temp1[n],temp2[n];

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

temp1[i]=P[i];

temp1[i].response\_time =-1;

}

// Sorting on the basis of arrival time

arrival\_sort(temp1,n);

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

temp2[i] = temp1[i];

}

int quantum\_time = 1;

int sum\_waiting=0, sum\_turnaround=0, sum\_response=0;

printf("Quantum time as given it is 1 . \n\n");

// Printing Gantt chart and calculating others

printf("Gantt chart of CPU for these processes: \n\n");

int t,tcurr=0,pflag=0;

for(int k=0; ; k++){

if(k>n-1){

k=0;

}

if(temp1[k].burst\_time >0){

printf(" %d %s ",tcurr , temp1[k].name);

}

t=0;

while(t<quantum\_time && temp1[k].burst\_time>0){

t++;

tcurr++;

temp1[k].burst\_time--;

}

if(temp1[k].burst\_time <= 0 && temp1[k].flag !=1 ){

temp1[k].waiting\_time = tcurr - temp2[k].burst\_time -temp1[k].arrival\_time;

temp1[k].turn\_around\_time = tcurr - temp1[k].arrival\_time;

pflag++;

temp1[k].flag=1;

sum\_waiting += temp1[k].waiting\_time;

sum\_turnaround += temp1[k].turn\_around\_time;

}

if(temp1[k].response\_time == -1){

temp1[k].response\_time = tcurr -1 - temp1[k].arrival\_time;

sum\_response +=temp1[k].response\_time;

}

if(pflag==n){

break;

}

}

printf(" %d\n\n\n",tcurr );

printf("Table showing info about the processes: \n\n");

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

temp1[i].burst\_time = temp2[i].burst\_time;

}

print\_table(temp1,n);

// Calculating Average waiting , Average turnaround and average response time

float average\_waiting\_time, average\_turnaround\_time, average\_response\_time;

average\_waiting\_time = (float)sum\_waiting/n;

average\_turnaround\_time = (float)sum\_turnaround/n;

average\_response\_time = (float)sum\_response/n;

printf("\n");

printf("Average waiting time : %f\n",average\_waiting\_time );

printf("Average turnaround time : %f\n",average\_turnaround\_time );

printf("Average response time: %f\n",average\_response\_time );

printf("\n");

}

// Algorithm for Multi Level Feedback Scheduling

// Lets suppose the process having

// quantum time less than or equal to 2 go to first queue

// quantum time less than or equal to 4 go to second queue

// rest go to third queue

void MLFQ(processes P[],int n){

processes temp[n],temp1[n],temp2[n],temp3[n];

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

temp[i] = P[i];

}

arrival\_sort(temp,n);

int tcurr=0;

int size1=0,size2=0,size3=0;

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

if(temp[i].burst\_time <=2){

temp1[size1]=temp[i];

size1++;

}

else if(temp[i].burst\_time <=4){

temp2[size2]=temp[i];

size2++;

}

else{

temp3[size3]=temp[i];

size3++;

}

}

printf("size1 = %d size2 = %d size3 = %d\n",size1,size2,size3 );

// RR(temp1,size1);

// RR(temp2,size2);

// RR(temp3,size3);

}

int main(int args,char \*argv[]){

FILE \*fp = fopen("process.dat","r");

if(fp==NULL){

printf("No such file exists.. Unable to open the file....\n");

exit(-1);

}

// Taking number of process thorugh file

int n;

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

processes P[n];

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

fscanf(fp,"%s",P[i].name);

fscanf(fp,"%d",&P[i].pid);

fscanf(fp,"%d",&P[i].arrival\_time);

fscanf(fp,"%d",&P[i].priority);

fscanf(fp,"%d",&P[i].share);

fscanf(fp,"%d",&P[i].burst\_time);

P[i].waiting\_time = P[i].response\_time = P[i].turn\_around\_time = 0;

}

fclose(fp);

// SJF Algorithm

printf("\n");

MLFQ(P,n);

return 0;

}

**5) Round Robin**

**Program for Round Robin CPU scheduling mechanism:**

// Non preemptive approach

#include<stdio.h>

#include<stdlib.h>

// Structure of process

typedef struct process{

char name[5];

int pid;

int arrival\_time;

int priority;

int share;

int burst\_time;

int waiting\_time;

int response\_time;

int turn\_around\_time;

int flag;

}processes;

// Sorting the process according to arrival time

void arrival\_sort(processes temp[],int n){

processes t;

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

for(int j=0;j<n-i;j++){

if(temp[j].arrival\_time >temp[j+1].arrival\_time){

t=temp[j];

temp[j]=temp[j+1];

temp[j+1]=t;

}

}

}

}

// Print Table

void print\_table(processes p[], int n)

{

int i;

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

printf("| Pname | PID | Arrival Time | Burst Time | Waiting Time | Turnaround Time | Response Time |\n");

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

for(i=0; i<n; i++) {

printf("| %s | %d | %d | %d | %d | %d | %d |\n",

p[i].name,p[i].pid,p[i].arrival\_time,p[i].burst\_time,p[i].waiting\_time,p[i].turn\_around\_time,p[i].response\_time);

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

}

}

// Printing gantt chart

void print\_gantt\_chart(processes p[], int n)

{

int i, j;

// print top bar

printf(" ");

for(i=0; i<n; i++) {

for(j=0; j<p[i].burst\_time; j++) printf("--");

printf(" ");

}

printf("\n|");

// printing process id in the middle

for(i=0; i<n; i++) {

for(j=0; j<p[i].burst\_time - 1; j++) printf(" ");

printf("%s", p[i].name);

for(j=0; j<p[i].burst\_time - 1; j++) printf(" ");

printf("|");

}

printf("\n ");

// printing bottom bar

for(i=0; i<n; i++) {

for(j=0; j<p[i].burst\_time; j++) printf("--");

printf(" ");

}

printf("\n");

// printing the time line

int val =0;

printf("0");

for(i=1; i<=n; i++) {

val =val+ p[i-1].burst\_time;

for(j=0; j<p[i-1].burst\_time; j++) printf(" ");

// if(p[i].turn\_around\_time > 9) printf("\b"); // backspace : remove 1 space

printf("%d", val);

}

printf("\n");

}

// Algorithm for Round Robin scheduling

void RR(processes P[],int n){

processes temp1[n],temp2[n];

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

temp1[i]=P[i];

temp1[i].response\_time =-1;

}

// Sorting on the basis of arrival time

arrival\_sort(temp1,n);

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

temp2[i] = temp1[i];

}

int quantum\_time = 1;

int sum\_waiting=0, sum\_turnaround=0, sum\_response=0;

printf("Quantum time as given it is 1 . \n\n");

// Printing Gantt chart and calculating others

printf("Gantt chart of CPU for these processes: \n\n");

int t,tcurr=0,pflag=0;

for(int k=0; ; k++){

if(k>n-1){

k=0;

}

if(temp1[k].burst\_time >0){

printf(" %d %s ",tcurr , temp1[k].name);

}

t=0;

while(t<quantum\_time && temp1[k].burst\_time>0){

t++;

tcurr++;

temp1[k].burst\_time--;

}

if(temp1[k].burst\_time <= 0 && temp1[k].flag !=1 ){

temp1[k].waiting\_time = tcurr - temp2[k].burst\_time -temp1[k].arrival\_time;

temp1[k].turn\_around\_time = tcurr - temp1[k].arrival\_time;

pflag++;

temp1[k].flag=1;

sum\_waiting += temp1[k].waiting\_time;

sum\_turnaround += temp1[k].turn\_around\_time;

}

if(temp1[k].response\_time == -1){

temp1[k].response\_time = tcurr -1 - temp1[k].arrival\_time;

sum\_response +=temp1[k].response\_time;

}

if(pflag==n){

break;

}

}

printf(" %d\n\n\n",tcurr );

printf("Table showing info about the processes: \n\n");

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

temp1[i].burst\_time = temp2[i].burst\_time;

}

print\_table(temp1,n);

// Calculating Average waiting , Average turnaround and average response time

float average\_waiting\_time, average\_turnaround\_time, average\_response\_time;

average\_waiting\_time = (float)sum\_waiting/n;

average\_turnaround\_time = (float)sum\_turnaround/n;

average\_response\_time = (float)sum\_response/n;

printf("\n");

printf("Average waiting time : %f\n",average\_waiting\_time );

printf("Average turnaround time : %f\n",average\_turnaround\_time );

printf("Average response time: %f\n",average\_response\_time );

printf("\n");

}

int main(int args,char \*argv[]){

FILE \*fp = fopen("process.dat","r");

if(fp==NULL){

printf("No such file exists.. Unable to open the file....\n");

exit(-1);

}

// Taking number of process thorugh file

int n;

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

processes P[n];

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

fscanf(fp,"%s",P[i].name);

fscanf(fp,"%d",&P[i].pid);

fscanf(fp,"%d",&P[i].arrival\_time);

fscanf(fp,"%d",&P[i].priority);

fscanf(fp,"%d",&P[i].share);

fscanf(fp,"%d",&P[i].burst\_time);

P[i].waiting\_time = P[i].response\_time = P[i].turn\_around\_time = 0;

}

fclose(fp);

// SJF Algorithm

printf("\n");

RR(P,n);

return 0;

}

**6) Lottery (Proportional Share)**

**Program for Lottery CPU scheduling mechanism:**

#include<stdio.h>

#include<stdlib.h>

// Structure of process

typedef struct process{

char name[5];

int pid;

int arrival\_time;

int priority;

int share;

int burst\_time;

int waiting\_time;

int response\_time;

int turn\_around\_time;

int flag;

int ticket\_start;

int ticket\_end;

}processes;

// Sorting the process according to arrival time

void arrival\_sort(processes temp[],int n){

processes t;

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

for(int j=0;j<n-i;j++){

if(temp[j].arrival\_time >temp[j+1].arrival\_time){

t=temp[j];

temp[j]=temp[j+1];

temp[j+1]=t;

}

}

}

}

// Print Table

void print\_table(processes p[], int n)

{

int i;

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

printf("| Pname | PID | Arrival Time | Burst Time | Waiting Time | Turnaround Time | Response Time |\n");

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

for(i=0; i<n; i++) {

printf("| %s | %d | %d | %d | %d | %d | %d |\n",

p[i].name,p[i].pid,p[i].arrival\_time,p[i].burst\_time,p[i].waiting\_time,p[i].turn\_around\_time,p[i].response\_time);

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

}

}

// Implementation of FCFS Aldorithm

void lottery(processes P[],int n){

processes temp[n],temp1[n];

int total\_ticket=0;

int start=0;

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

temp[i]=P[i];

total\_ticket+=temp[i].burst\_time;

temp[i].response\_time=-1;

}

total\_ticket+=temp[n-1].burst\_time;

// Sort according to arrival time

arrival\_sort(temp,n);

// Now Sorting on the basis if priority

processes t;

for(int i=2;i<n;i++){

for(int j=1;j<n-i+1;j++){

if(temp[j].priority > temp[j+1].priority){

t=temp[j];

temp[j]=temp[j+1];

temp[j+1]=t;

}

}

}

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

temp[i].ticket\_start=start;

start+=temp[i].burst\_time+1;

temp[i].ticket\_end=start-1;

temp1[i]=temp[i];

}

int tcurr=0;

int sum\_waiting=0, sum\_turnaround=0, sum\_response=0;

int count=0;

// Gantt Chart for lottery scheduling

printf("\nGantt chart of CPU for these processes: \n");

printf("CPU taking time per second as follow\n\n");

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

int random= rand()%(total\_ticket);

// printf("%d\n",random );

for(int j=0;j<n;j++){

if(random>temp1[j].ticket\_start && random<temp1[j].ticket\_end){

if(temp1[j].burst\_time>0){

temp1[j].burst\_time--;

if(temp[j].response\_time==-1){

temp[j].response\_time=tcurr;

sum\_response+=temp[j].response\_time;

}

tcurr++;

printf("%s ",temp[j].name );

}

else if(temp1[j].burst\_time==0 && temp[j].flag!=1){

temp[j].flag=1;

temp[j].turn\_around\_time= tcurr - temp[j].arrival\_time;

temp[j].waiting\_time=temp[j].turn\_around\_time-temp[j].burst\_time;

sum\_waiting+=temp[j].waiting\_time;

sum\_turnaround+=temp[j].turn\_around\_time;

count++;

}

}

}

if(count==n){

break;

}

}

printf("\n");

printf("\n");

printf("Table showing info about the processes: \n\n");

print\_table(temp,n);

printf("\n");

printf("\n");

// Calculating Average waiting , Average turnaround and average response time

float average\_waiting\_time, average\_turnaround\_time, average\_response\_time;

average\_waiting\_time = (float)sum\_waiting/n;

average\_turnaround\_time = (float)sum\_turnaround/n;

average\_response\_time = (float)sum\_response/n;

printf("\n");

printf("Average waiting time : %f\n",average\_waiting\_time );

printf("Average turnaround time : %f\n",average\_turnaround\_time );

printf("Average response time: %f\n",average\_response\_time );

printf("\n");

}

int main(int args,char \*argv[]){

FILE \*fp = fopen("process.dat","r");

if(fp==NULL){

printf("No such file exists.. Unable to open the file....\n");

exit(-1);

}

// Taking number of process thorugh file

int n;

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

processes P[n];

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

fscanf(fp,"%s",P[i].name);

fscanf(fp,"%d",&P[i].pid);

fscanf(fp,"%d",&P[i].arrival\_time);

fscanf(fp,"%d",&P[i].priority);

fscanf(fp,"%d",&P[i].share);

fscanf(fp,"%d",&P[i].burst\_time);

P[i].waiting\_time = P[i].response\_time = P[i].turn\_around\_time = 0;

}

// FIFS Algorithm

lottery(P,n);

fclose(fp);

return 0;

}