**Lab Exercise 3: Implementation of CPU Scheduling Policies: FCFS and SJF**

***Code:***

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

#include<limits.h>

struct process

{

char pid[5];

int bt,at,wt,rt,ta,ct,id;

};

void swap(struct process \*xp,struct process \*yp)

{

struct process temp = \*xp;

\*xp=\*yp;

\*yp=temp;

}

void sort(struct process \*P,int n,int k)

{

int min\_i;

if(k==2) // to revert back to input order

{

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

{

min\_i=i;

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

{

if(P[j].id<P[min\_i].id)

min\_i=j;

}

if(min\_i!=i)

swap(&P[min\_i],&P[i]);

}

return;

}

for(int i=0;i<n;i++) // sort wrt to at

{

min\_i=i;

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

{

if(P[j].at<P[min\_i].at)

min\_i=j;

}

if(min\_i!=i)

swap(&P[min\_i],&P[i]);

}

for(int i=0;i<n;i++) // sort wrt to input order

{

min\_i=i;

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

{

if(P[j].at==P[min\_i].at && P[j].id<P[min\_i].id)

min\_i=j;

}

if(min\_i!=i)

swap(&P[min\_i],&P[i]);

}

}

int input(struct process \*P,int n) // to get input

{

int sorted=1;

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

{

printf("Enter pid: ");

scanf("%s",P[i].pid);

printf("Enter Arrival time: ");

scanf("%d",&P[i].at);

if(i!=0 && sorted)

sorted=P[i].at>P[i-1].at?1:0;

printf("Enter burst time: ");

scanf("%d",&P[i].bt);

P[i].id=i+1;

P[i].ct=P[i].rt=-1;

printf("\n");

}

return sorted;

}

void gantt\_chart(char str[])

{

int y=0;

for(int i=0;i<strlen(str);i++) // top line

printf("%c",str[i]=='|'?'+':'-');

printf("\n%s\n",str); // chart

for(int i=0;i<strlen(str);i++)

printf("%c",str[i]=='|'?'+':'-'); // bottom line

printf("\n");

int time=0,prev=0;

for(int i=0;i<strlen(str);i++) // to print time stamps

{

if(str[i]=='|')

{

printf("%\*d",i-prev,time);

prev=i;

y=1;

}

else if(str[i]==' ')

{

if(y)

time++;

}

else

y=0;

}

printf("\n");

}

void display(struct process P[],int n,float ata,float awt,float art) // display table

{

printf("-------------------------------------------------------------------------------------");

printf("\nPID\tArrival Time\tBurst Time Turnaround Time\tWaiting Time\tResponse Time\n");

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

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

printf("%3s\t%-12d %-10d %-15d\t%-12d\t%d\n",P[i].pid,P[i].at,P[i].bt,P[i].ta,P[i].wt,P[i].rt);

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

printf("%31s %-15.2f\t%-12.2f\t%0.2f\n","Average",ata,awt,art);

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

//printf("%s \t %0.2f \t %0.2f \t %0.2f \t %0.2f \t %0.2f\n",P[i].name,P[i].at,P[i].bt,P[i].ta,P[i].wt,P[i].rt);

}

void FCFS()

{

int n;

char buf[100],pid[5],gantt[1000];

strcpy(gantt,"|");

float tbt=0,tta=0,twt=0,trt=0;

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

printf("Enter no of processes: ");

scanf("%d",&n);

if(n<=0)

return;

struct process P[n];

int sorted=input(P,n);

if(!sorted)

sort(P,n,0); // sort wrt to at

int cur\_time=0;

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

{

if (cur\_time < P[i].at)

{

//idle

sprintf(buf,"%\*s%s%\*s%s",P[i].at-cur\_time," ","#",P[i].at-cur\_time," ","|");

strcat(gantt,buf);

/\*sprintf(buf,"%\*s",P[i].at-cur\_time," ");

sprintf(temp,"%s%s%s%s",buf,"#",buf,"|");

strcat(gantt,temp); \*/

/\*//for (int j = 0; j < (P[i].at-cur\_time); j ++)

// strcat(gantt," ");

strcat(gantt,"#");

for (int j = 0; j < (P[i].at-cur\_time); j ++)

strcat(gantt," ");

strcat(gantt,"|");

//\*/

cur\_time = P[i].at ;

i--;

}

else

{

cur\_time+=P[i].bt;

sprintf(buf,"%\*s%s%\*s%s",P[i].bt," ",P[i].pid,P[i].bt," ","|");

strcat(gantt,buf);

P[i].ct=cur\_time;

P[i].ta=P[i].ct-P[i].at;

P[i].wt=P[i].rt=P[i].ta-P[i].bt;

tta+=P[i].ta;

twt+=P[i].wt;

}

}

trt=twt; // cuz np

sort(P,n,2); // revert to input order

printf("\n");

gantt\_chart(gantt);

printf("\n");

display(P,n,tta/n,twt/n,trt/n);

}

void SJF()

{

int n;

char buf[100],pid[5],gantt[1000];

strcpy(gantt,"|");

float tbt=0,tta=0,twt=0,trt=0;

printf("\n---------------SCJ-NP Scheduler-----------\n");

printf("Enter no of processes: ");

scanf("%d",&n);

if(n<=0)

return;

struct process P[n];

int sorted=input(P,n);

if(!sorted)

sort(P,n,0); // sort wrt to at

int min,min\_i;

int i=0,cur\_time=0;

while(i<n) //order seq

{

if(cur\_time<P[i].at)

{

//idle

sprintf(buf,"%\*s%s%\*s%s",P[i].at-cur\_time," ","#",P[i].at-cur\_time," ","|");

strcat(gantt,buf);

//

cur\_time=P[i].at;

continue;

}

min=P[i].bt;

min\_i=i;

for(int j=i+1;j<n && i!=0;j++) // finding the shortest task

{

if(cur\_time>=P[j].at)

{

if(min>P[j].bt)

{

min=P[j].bt;

min\_i=j;

}

}

}

if(min\_i!=i) // pushing the completed task to front of the array

swap(&P[min\_i],&P[i]);

cur\_time+=P[i].bt;

P[i].ct=cur\_time;

P[i].ta=P[i].ct-P[i].at;

P[i].wt=P[i].rt=P[i].ta-P[i].bt;

tta+=P[i].ta;

twt+=P[i].wt;

sprintf(buf,"%\*s%s%\*s%s",P[i].bt," ",P[i].pid,P[i].bt," ","|");

strcat(gantt,buf);

i++;

}

sort(P,n,2); // revert to input order

trt=twt; // cuz np

printf("\n");

gantt\_chart(gantt);

printf("\n");

display(P,n,tta/n,twt/n,trt/n);

}

void SRTF()

{

int n;

char buf[200],pid[5],gantt[1000];

strcpy(gantt,"|");

float tbt=0,tta=0,twt=0,trt=0;

printf("\n---------------SJF-P Scheduler-----------\n");

printf("Enter no of processes: ");

scanf("%d",&n);

if(n<=0)

return;

struct process P[n+1];

int sorted=input(P,n);

if(!sorted)

sort(P,n,0);

int rbt[n+1]; // copy of burst time -> remaining time

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

{

rbt[i]=P[i].bt;

}

int shortest\_task,prev=0,k=0,comp=0,cur\_time=0;

rbt[n]=INT\_MAX;

strcpy(P[n].pid,"#");

for(cur\_time=0;comp!=n;cur\_time++)

{

int min=INT\_MAX;

shortest\_task=n;

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

{

if( (rbt[j]<rbt[shortest\_task]) && (P[j].at<=cur\_time && rbt[j]>0))

shortest\_task=j;

}

rbt[shortest\_task]-=1;

if(P[shortest\_task].rt==-1) // response time

P[shortest\_task].rt=cur\_time-P[shortest\_task].at;

if(shortest\_task==prev)

k++;

else

{

sprintf(buf,"%\*s%s%\*s%s",k," ",P[prev].pid,k," ","|");

strcat(gantt,buf);

k=1;

}

if(rbt[shortest\_task]==0)

{

comp++;

P[shortest\_task].ct=cur\_time+1;

P[shortest\_task].ta=P[shortest\_task].ct-P[shortest\_task].at;

P[shortest\_task].wt=P[shortest\_task].ta-P[shortest\_task].bt;

trt+=P[shortest\_task].rt;

tta+=P[shortest\_task].ta;

twt+=P[shortest\_task].wt;

}

prev=shortest\_task;

}

sprintf(buf,"%\*s%s%\*s%s",k," ",P[prev].pid,k," ","|");

strcat(gantt,buf);

printf("\n");

gantt\_chart(gantt);

printf("\n");

display(P,n,tta/n,twt/n,trt/n);

}

int main()

{

int ch;

char ch2;

do

{

printf("\nMenu:\n");

printf("1.FCFS\n");

printf("2.SJF\n");

printf("3.EXIT\n");

printf("Choice: ");

scanf("%d",&ch);

switch(ch)

{

case 1: FCFS();

break;

case 2: printf("\n---------------SJF Scheduler-----------\n");

do

{

printf("\na.Non-Preemptive\n");

printf("b.Preemptive\n");

printf("c.Go back\n");

printf("d.Exit\n");

printf("Choice: ");

scanf(" %c",&ch2);

switch(ch2)

{

case 'a':

case 'A': SJF();

break;

case 'b':

case 'B': SRTF();

break;

case 'c':

case 'C': break;

case 'd':

case 'D': printf("\nExiting...\n");

return 0;

default: printf("\nInavlid Input!\n");

}

}while(!(ch2-97>=0 && ch2-99<=0||ch2-65>=0 && ch2-67<=0));

break;

case 3: printf("\nExiting...\n");

break;

default:printf("\nInavlid Input!\n");

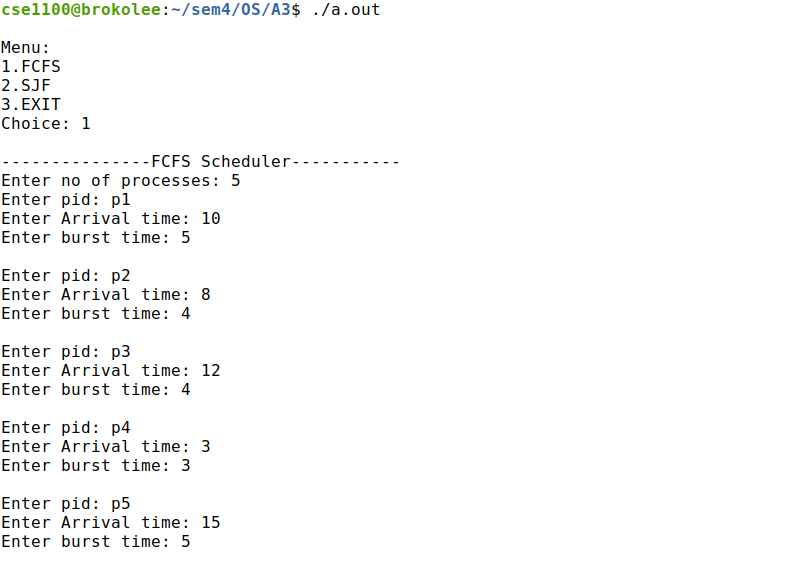
}

}while(ch!=3);

return 0;

}

***Output:***

******