SLIP 1

QUE1

#include<stdio.h>

#include<unistd.h>

int main()

{

int pid,retnice,i;

pid=fork();

for(i=0;i<3;i++)

{

if(pid==0)

{

retnice=nice(-5);

printf("Child gets higher CPU priority%d\n",retnice);

sleep(1);

}

else

{

retnice=nice(4);

printf("Parent gets lower CPU Priority%d\n",retnice);

sleep(1);

}

}

}

QUE2 (FIFO)

#include<stdio.h>

#include<string.h>

int RefString[20],PT[10],nof,nor;

void Accept(){

int i;

printf("Enter Reference String: \n");

for(i=0;i<nor;i++)

{

printf("[%d]=",i);

scanf("%d",&RefString[i]);

}

}

int Search(int s){

int i;

for(i=0;i<nof; i++)

if(PT[i]==s)

return(i);

return(-1);

}

void FIFO(){

int i,j,k=0,Faults=0;

for(i=0;i<nor;i++)

{

printf("\n%2d",RefString[i]);

if(Search(RefString[i])==-1)

{

PT[k]=RefString[i];

for(j=0;j<nof;j++)

{

if(PT[j])

{

printf("\t%2d",PT[j]);

}

}

Faults++;

k=(k+1)%nof;

}

}

printf("\nTotal Page Faults: %d",Faults);

}

void main(){

printf("Enter Length of reference string: ");

scanf("%d",&nor);

printf("Enter No. of Frames: ");

scanf("%d",&nof);

Accept();

FIFO();

}

QUE2 (BANKER’S ALGO)

#include<stdio.h>

int nop,nor,A[10][10],M[10][10],Av[10],N[10][10],finish[10];

void acceptdata(int x[10][10]){

int i,j;

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

printf("P%d\n",i);

for(j=0;j<nor;j++)

{

printf("%c: ",65+j);

scanf("%d",&x[i][j]);

}

}

}

void acceptav(){

int i;

for(i=0;i<nor;i++)

{

printf("%c: ",65+i);

scanf("%d",&Av[i]);

}

}

void calcneed(){

int i,j;

for(i=0;i<nop;i++)

for(j=0;j<nor;j++)

N[i][j]=M[i][j]-A[i][j];

}

void displaydata(){

int i,j;

printf("\n\tAllocation \t\tMax\t\tNeed\n\t");

for(i=0;i<3;i++)

{

for(j=0;j<nor;j++)

printf("%4c",65+j);

printf("\t");

}

for(i=0;i<nop;i++)

{

printf("\nP%d\t",i);

for(j=0;j<nor;j++)

printf("%4d",A[i][j]);

printf("\t");

for(j=0;j<nor;j++)

printf("%4d",M[i][j]);

printf("\t");

for(j=0;j<nor;j++)

printf("%4d",N[i][j]);

}

printf("\navailable");

for(i=0;i<nor;i++)

printf("%4d",Av[i]);

}

int checkneed(int pno)

{

int i;

for(i=0;i<nor;i++)

if(N[pno][i]>Av[i])

return 0;

return 1;

}

void banker(){

int p=0,j=0,k=0,flag=0,safe[10];

while(flag<2)

{

if(!finish[p])

{

printf("\n\nNeed of process P%d (,",p);

for(j=0;j<nor;j++)

printf("%d,",N[p][j]);

if(checkneed(p))

{

printf(") <= available (");

for(j=0;j<nor;j++)

printf("%d,",Av[j]);

printf(")");

printf("\nNeed is Satsified, So process P%d can be granted requiered resources.\n After P%d finishes, it will realease all the resources.",p,p);

for(j=0;j<nor;j++)

Av[j]=Av[j]+A[p][j];

printf("New Availble=");

for(j=0;j<nor;j++)

printf("%d ",Av[j]);

finish[p]=1;

safe[k++]=p;

}

else{

printf(") > available (");

for(j=0;j<nor;j++)

printf("%d,",Av[j]);

printf(")");

printf("\nNeed is not Satsified, So process P%d cannot be granted required resources.\n process P%d has to wait.",p,p);

}

}

if((p+1)%nop==0)

flag++;

p=(p+1)%nop;

}//while

if(k==nop){

printf("\nSystem is in safe state...");

printf("\nSafe Sequence: ");

for(j=0;j<k;j++)

printf("P%d->",safe[j]);

}

else

printf("\nSystem is not in safe state....");

}

Int main(){

printf("\nEnter No of Processes: ");

scanf("%d",&nop);

printf("\nEnter No. of Resources: ");

scanf("%d",&nor);

printf("\nEnter Allocation Matrix: ");

acceptdata(A);

printf("\nEnter Max Matrix: ");

acceptdata(M);

printf("\nEnter Availability:");

acceptav();

calcneed();

displaydata();

banker();

}

SLIP 2

QUE1

#include<stdio.h>

#include<unistd.h>

void childprocess(){

printf("\n HELLO WORLD");

}

void parentprocess(){

printf("\nHI");

}

int main(){

int pid;

pid=fork();

if(pid==0)

{

printf("Child Process ID=%d",pid);

childprocess();

}

else

{

printf("\n Parent Process Id=%d",pid);

parentprocess();

}

}

QUE2 (SJF NON-PREEMPTIVE)

#include<stdio.h>

struct input{

char pname[10];

int bt,at,tbt,ft;

}tab[10];

struct gantt{

int start,end;

char pname[10];

}g[50],g1[10];

int n,i,k,time,prev;

void getinput(){

printf("\nEnter No of Processes: ");

scanf("%d",&n);

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

printf("\nProcess Name: ");

scanf("%s",tab[i].pname);

printf("Burst Time: ");

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

tab[i].tbt=tab[i].bt;

printf("Arrival Time:" );

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

}

}

void printinput(){

printf("\nPname\tBT\tAT");

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

printf("\n%s\t%d\t%d",tab[i].pname,tab[i].tbt,tab[i].at);

}

void sort(){

struct input temp;

int j;

for(i=1;i<n;i++)//pass

for(j=0;j<n-1;j++)//Comp

if(tab[j].at>tab[j+1].at)

{

temp=tab[j];

tab[j]=tab[j+1];

tab[j+1]=temp;

}

}

int arrived(int time){

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

if(tab[i].at<=time && tab[i].tbt!=0)

return 1;

return 0;

}

int getsmallburst(int time){

int min=99,mini;

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

if(tab[i].tbt<min && tab[i].at<=time && tab[i].tbt!=0){

min=tab[i].tbt;

mini=i;

}

}

return mini;

}

void processinput(){

int j,finish=0;

// time=tab[0].at;

while(finish!=n)

{

if(arrived(time))

{

i=getsmallburst(time);

for(j=0;j<tab[i].bt;j++)

{

time++;

tab[i].tbt--;

g[k].start=prev;

g[k].end=time;

// printinput();

prev=time;

tab[i].ft=time;

strcpy(g[k++].pname,tab[i].pname);

if(tab[i].tbt==0)

{

finish++;

break;

}

}

}

else

{

time++;

g[k].start=prev;

g[k].end=time;

prev=time;

strcpy(g[k++].pname,"idle");

}

}

// printinput();

}

void printoutput(){

int TTAT=0,TWT=0;

float ATAT,AWT;

printf("\n\*\*\*\*\*\*Final Table\*\*\*\*\*");

printf("\nPname\tAT\tBT\tFT\tTAT\tWT");

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

printf("\n%s\t%d\t%d\t%d\t%d\t%d",tab[i].pname,tab[i].at,tab[i].bt,tab[i].ft,tab[i].ft-tab[i].at,tab[i].ft-tab[i].at-tab[i].bt);

TTAT=TTAT+(tab[i].ft-tab[i].at);

TWT=TWT+(tab[i].ft-tab[i].at-tab[i].bt);

}

ATAT=(float)TTAT/n;

AWT=(float)TWT/n;

printf("\nTotal TAT=%d",TTAT);

printf("\nTotal WT=%d",TWT);

printf("\nAverage TAT=%f",ATAT);

printf("\nAverage WT=%f",AWT);

}

void printganttchart(){

int j=0;

g1[0]=g[0];

for(i=1;i<k;i++)

{

if(strcmp(g1[j].pname,g[i].pname)==0)

g1[j].end=g[i].end;

else

{

j++;

g1[j]=g[i];

}

}

printf("\n\*\*\*\*\*\*Each unit Gantt chart\*\*\*\*\*\*");

for(i=0;i<k;i++)

printf("\n%d\t%s\t%d",g[i].start,g[i].pname,g[i].end);

printf("\n\*\*\*\*\*\*Final Gantt chart\*\*\*\*\*\*");

for(i=0;i<=j;i++)

printf("\n%d\t%s\t%d",g1[i].start,g1[i].pname,g1[i].end);

}

int main()

{

getinput();

printf("\nEntered data is: ");

printinput();

sort();

printf("\nData after Sorting" );

printinput();

processinput();

printoutput();

printganttchart();

}

QUE2

Menu driven need to do

SLIP3

QUE1

\*\*\*Doubt\*\*\*

QUE2 (FCFS)

#include<stdio.h>

#include<string.h>

struct input{

char pname[10];

int bt,at,tbt,ft;

}tab[10];

struct gantt{

int start,end;

char pname[10];

}g[50],g1[10];

int n,i,k,time,prev;

void getinput(){

printf("\nEnter No of Processes: ");

scanf("%d",&n);

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

{

printf("\nProcess Name: ");

scanf("%s",tab[i].pname);

printf("Burst Time: ");

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

tab[i].tbt=tab[i].bt;

printf("Arrival Time:" );

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

}}

void printinput(){

printf("\nPname\tBT\tAT");

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

printf("\n%s\t%d\t%d",tab[i].pname,tab[i].tbt,tab[i].at);

}

void sort(){

struct input temp;

int j;

for(i=1;i<n;i++)//pass

for(j=0;j<n-1;j++)//Comp

if(tab[j].at>tab[j+1].at)

{

temp=tab[j];

tab[j]=tab[j+1];

tab[j+1]=temp;

}

}

int arrived(int time)

{

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

if(tab[i].at<=time && tab[i].tbt!=0)

return 1;

return 0;

}

void processinput()

{

int j,finish=0;

// time=tab[0].at;

while(finish!=n)

{

if(arrived(time))

{

for(j=0;j<tab[i].bt;j++)

{

time++;

tab[i].tbt--;

g[k].start=prev;

g[k].end=time;

// printinput();

prev=time;

tab[i].ft=time;

strcpy(g[k++].pname,tab[i].pname);

if(tab[i].tbt==0)

{

finish++;

break;

}

}

else

{

time++;

g[k].start=prev;

g[k].end=time;

prev=time;

strcpy(g[k++].pname,"idle");

}

}

// printinput();

}

void printoutput()

{

int TTAT=0,TWT=0;

float ATAT,AWT;

printf("\n\*\*\*\*\*\*Final Table\*\*\*\*\*");

printf("\nPname\tAT\tBT\tFT\tTAT\tWT");

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

{

printf("\n%s\t%d\t%d\t%d\t%d\t%d",tab[i].pname,tab[i].at,tab[i].bt,tab[i].ft,tab[i].ft-tab[i].at,tab[i].ft-tab[i].at-tab[i].bt);

TTAT=TTAT+(tab[i].ft-tab[i].at);

TWT=TWT+(tab[i].ft-tab[i].at-tab[i].bt);

}

ATAT=(float)TTAT/n;

AWT=(float)TWT/n;

printf("\nTotal TAT=%d",TTAT);

printf("\nTotal WT=%d",TWT);

printf("\nAverage TAT=%f",ATAT);

printf("\nAverage WT=%f",AWT);

}

void printganttchart()

{

int j=0;

g1[0]=g[0];

for(i=1;i<k;i++)

{

if(strcmp(g1[j].pname,g[i].pname)==0)

g1[j].end=g[i].end;

else

{

j++;

g1[j]=g[i];

}

}

printf("\n\*\*\*\*\*\*Each unit Gantt chart\*\*\*\*\*\*");

for(i=0;i<k;i++)

printf("\n%d\t%s\t%d",g[i].start,g[i].pname,g[i].end);

printf("\n\*\*\*\*\*\*Final Gantt chart\*\*\*\*\*\*");

for(i=0;i<=j;i++)

printf("\n%d\t%s\t%d",g1[i].start,g1[i].pname,g1[i].end);

}

int main(){

getinput();

printf("\nEntered data is: ");

printinput();

sort();

printf("\nData after Sorting" );

printinput();

processinput();

printoutput();

printganttchart();

}

SLIP4

QUE1

#include<stdio.h>

#include<unistd.h>

int main()

{

int pid,a;

printf("\n the process id is%d",getpid());

pid=fork();

if(pid<0)

printf("\n Fork failed!");

else if(pid==0)

{

printf("\n I am Child process");

printf("\n Child process ID=%d",getpid());

}

else

{

wait(NULL);

sleep(5);

printf("\n I am Parent process");

printf("\n Child Parent ID=%d",getpid());

}

}

QUE2 (PRIORITY SCHEDULING NON-PREEMPTIVE)

#include<stdio.h>

#include<string.h>

struct input

{

char pname[10];

int bt,at,tbt,ft,p;

}tab[10];

struct gantt

{

char pname[10];

int start,end;

}g[30],g1[30];

int n,i,time,prev,k;

void getinput()

{

printf("\nEnter No of Process: ");

scanf("%d",&n);

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

{

printf("\nEnter Process Name: ");

scanf("%s",tab[i].pname);

printf("Arrival Time:");

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

printf("Burst Time: ");

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

tab[i].tbt=tab[i].bt;

printf("\nEnter the Priority:");

scanf("%d",&tab[i].p);

}

}

void printinput()

{

// int TWT=0,TTAT=0;

printf("\nPname\tAT\tBT");

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

printf("\n%s\t%d\t%d",tab[i].pname,tab[i].at,tab[i].bt);

}

void printoutput()

{

int TWT=0,TTAT=0;

float ATAT,AWT;

printf("\nPname\tAT\tBT\tFT\tWT\tTAT");

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

{

printf("\n%s\t%d\t%d\t%d\t%d\t%d",tab[i].pname,tab[i].at,tab[i].bt,tab[i].ft,tab[i].ft-tab[i].at-tab[i].bt,tab[i].ft-tab[i].at);

TWT=TWT+(tab[i].ft-tab[i].at-tab[i].bt);

TTAT=TTAT+(tab[i].ft-tab[i].at);

}

printf("\nTotal WT: %d",TWT);

printf("\nTotal TAT:%d",TTAT);

AWT=(float)TWT/n;

ATAT=(float)TTAT/n;

printf("\nAverage WT: %f",AWT);

printf("\nAverage TAT:%f",ATAT);

}

void sort()

{

int pass;

struct input temp;

for(pass=1;pass<n;pass++)

{

for(i=0;i<n-pass;i++)

{

if(tab[i].at>tab[i+1].at)

{

temp=tab[i];

tab[i]=tab[i+1];

tab[i+1]=temp;

}

}

}

}

int arrived(int time)

{

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

{

if(tab[i].at<=time && tab[i].tbt!=0)

return 1;

}

return 0;

}

int gethighpriority(int time)

{

int processpos,min=99;

for(i=0;i<n;i++)//i=0,1

{ // p1,p3 min=2

if(tab[i].at<=time && tab[i].tbt!=0 && tab[i].p<min)

{

min=tab[i].p;

processpos=i;

}

}

return processpos;

}

void processinput()

{

int finish=0,j;

k=0;

while(finish!=n)

{

if(arrived(time))

{

i=gethighpriority(time);

for(j=0;j<tab[i].bt;j++)

{

time++;

tab[i].tbt--;

g[k].start=prev;

g[k].end=time;

prev=time;

strcpy(g[k++].pname,tab[i].pname);

tab[i].ft=time;

if(tab[i].tbt==0)

{

finish++;

break;

}

}

}

else

{

time++;

g[k].start=prev;

g[k].end=time;

strcpy(g[k++].pname,"idle");

prev=time;

}

// i++;

}

}

void ganttchart()

{

int i,j=0;

printf("\n\*\*\*\*\*\*Each Unit Gantt chart\*\*\*\*\*\*");

printf("\nStart\tpname\tEnd");

for(i=0;i<k;i++)

{

printf("\n%d\t%s\t%d",g[i].start,g[i].pname,g[i].end);

}

printf("\n\*\*\*\*\*\*\*\*Final Gantt Chart\*\*\*\*\*\*\*");

g1[0]=g[0];

for(i=1;i<k;i++)

{

if(strcmp(g[i].pname,g1[j].pname)==0)

g1[j].end=g[i].end;

else

{

j++;

g1[j]=g[i];

}

}

printf("\nStart\tpname\tEnd");

for(i=0;i<=j;i++)

{

printf("\n%d\t%s\t%d",g1[i].start,g1[i].pname,g1[i].end);

}

}

int main()

{

getinput();

printinput();

sort();

printf("\nData After Sorting: ");

printinput();

processinput();

printoutput();

ganttchart();

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

{

tab[i].tbt=tab[i].bt=rand()%10+1;

tab[i].at=tab[i].ft+2;

}

printinput();

processinput();

printoutput();

ganttchart();

}

QUE2(BANKER’S ALGO)

#include<stdio.h>

int nop,nor,A[10][10],M[10][10],Av[10],N[10][10],finish[10];

void acceptdata(int x[10][10])

{

int i,j;

for(i=0;i<nop;i++)

{

printf("P%d\n",i);

for(j=0;j<nor;j++)

{

printf("%c: ",65+j);

scanf("%d",&x[i][j]);

}

}

}

void acceptav()

{

int i;

for(i=0;i<nor;i++)

{

printf("%c: ",65+i);

scanf("%d",&Av[i]);

}

}

void calcneed()

{

int i,j;

for(i=0;i<nop;i++)

for(j=0;j<nor;j++)

N[i][j]=M[i][j]-A[i][j];

}

void displaydata()

{

int i,j;

printf("\n\tAllocation \t\tMax\t\tNeed\n\t");

for(i=0;i<3;i++)

{

for(j=0;j<nor;j++)

printf("%4c",65+j);

printf("\t");

}

for(i=0;i<nop;i++)

{

printf("\nP%d\t",i);

for(j=0;j<nor;j++)

printf("%4d",A[i][j]);

printf("\t");

for(j=0;j<nor;j++)

printf("%4d",M[i][j]);

printf("\t");

for(j=0;j<nor;j++)

printf("%4d",N[i][j]);

}

printf("\navailable");

for(i=0;i<nor;i++)

printf("%4d",Av[i]);

}

int checkneed(int pno)

{

int i;

for(i=0;i<nor;i++)

if(N[pno][i]>Av[i])

return 0;

return 1;

}

void banker(){

int p=0,j=0,k=0,flag=0,safe[10];

while(flag<2)

{

if(!finish[p])

{

printf("\n\nNeed of process P%d (,",p);

for(j=0;j<nor;j++)

printf("%d,",N[p][j]);

if(checkneed(p))

{

printf(") <= available (");

for(j=0;j<nor;j++)

printf("%d,",Av[j]);

printf(")");

printf("\nNeed is Satsified, So process P%d can be granted requiered resources.\n After P%d finishes, it will realease all the resources.",p,p);

for(j=0;j<nor;j++)

Av[j]=Av[j]+A[p][j];

printf("New Availble=");

for(j=0;j<nor;j++)

printf("%d ",Av[j]);

finish[p]=1;

safe[k++]=p;

}

else

{

printf(") > available (");

for(j=0;j<nor;j++)

printf("%d,",Av[j]);

printf(")");

printf("\nNeed is not Satsified, So process P%d cannot be granted required resources.\n process P%d has to wait.",p,p);

}

}

if((p+1)%nop==0)

flag++;

p=(p+1)%nop;

}//while

if(k==nop)

{

printf("\nSystem is in safe state...");

printf("\nSafe Sequence: ");

for(j=0;j<k;j++)

printf("P%d->",safe[j]);

}

else

printf("\nSystem is not in safe state....");

}

main(){

printf("\nEnter No of Processes: ");

scanf("%d",&nop);

printf("\nEnter No. of Resources: ");

scanf("%d",&nor);

printf("\nEnter Allocation Matrix: ");

acceptdata(A);

printf("\nEnter Max Matrix: ");

acceptdata(M);

printf("\nEnter Availability:");

acceptav();

calcneed();

displaydata();

banker();

}

SLIP5

QUE1

#include<stdio.h>

int main()

{

int pid,retnice,i;

pid=fork();

for(i=0;i<3;i++)

{

if(pid==0)

{

retnice=nice(-5);

printf("Child gets higher CPU priority%d\n",retnice);

sleep(1);

}

else

{

retnice=nice(4);

printf("Parent gets lower CPU Priority%d\n",retnice);

sleep(1);

}

}

}

QUE2 (FIFO)

#include<stdio.h>

int RefString[20],PT[10],nof,nor;

void Accept()

{

int i;

printf("Enter Reference String: \n");

for(i=0;i<nor;i++)

{

printf("[%d]=",i);

scanf("%d",&RefString[i]);

}

}

int Search(int s)

{

int i;

for(i=0;i<nof; i++)

if(PT[i]==s)

return(i);

return(-1);

}

void FIFO(){

int i,j,k=0,Faults=0;

for(i=0;i<nor;i++)

{

printf("\n%2d",RefString[i]);

if(Search(RefString[i])==-1)

{

PT[k]=RefString[i];

for(j=0;j<nof;j++)

{

if(PT[j])

{

printf("\t%2d",PT[j]);

}

}

Faults++;

k=(k+1)%nof;

}

}

printf("\nTotal Page Faults: %d",Faults);

}

void main(){

printf("Enter Length of reference string: ");

scanf("%d",&nor);

printf("Enter No. of Frames: ");

scanf("%d",&nof);

Accept();

FIFO();}

QUE2

Menu driven bankers need to do

SLIP6

QUE1

#include <stdio.h>

#include <time.h>

// Function to perform the set of instructions

void performInstructions() {

// Replace this with the set of instructions you want to measure

for (int i = 0; i < 1000000; ++i) {

// Some time-consuming instructions

int result = i \* i;

}

}

int main() {

// Record the starting time

clock\_t start\_time = clock();

// Call the function with the set of instructions

performInstructions();

// Record the ending time

clock\_t end\_time = clock();

// Calculate the execution time in seconds

double execution\_time = ((double)(end\_time - start\_time)) / CLOCKS\_PER\_SEC;

// Print the execution time

printf("Execution time: %f seconds\n", execution\_time);

return 0;

}

QUE2 (FIFO)

#include<stdio.h>

int RefString[20],PT[10],nof,nor;

void Accept(){

int i;

printf("Enter Reference String: \n");

for(i=0;i<nor;i++)

{

printf("[%d]=",i);

scanf("%d",&RefString[i]);

}

}

int Search(int s){

int i;

for(i=0;i<nof; i++)

if(PT[i]==s)

return(i);

return(-1);

}

void FIFO(){

int i,j,k=0,Faults=0;

for(i=0;i<nor;i++)

{

printf("\n%2d",RefString[i]);

if(Search(RefString[i])==-1)

{

PT[k]=RefString[i];

for(j=0;j<nof;j++)

{

if(PT[j])

{

printf("\t%2d",PT[j]);

}

}

Faults++;

k=(k+1)%nof;

}

}

printf("\nTotal Page Faults: %d",Faults);

}

void main(){

printf("Enter Length of reference string: ");

scanf("%d",&nor);

printf("Enter No. of Frames: ");

scanf("%d",&nof);

Accept();

FIFO();

}

SLIP7

QUE1

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/wait.h>

int main() {

pid\_t pid;

// Fork a child process

pid = fork();

if (pid < 0) {

// Error occurred

fprintf(stderr, "Fork failed\n");

return 1;

} else if (pid == 0) {

// Child process

// Use execl() to execute the "ls" command

execl("/bin/ls", "ls", (char \*)NULL);

// If execl fails

perror("execl");

exit(1);

} else {

// Parent process

// Parent goes to sleep state

sleep(5);

// Parent process may continue its own execution here

printf("Parent process waking up after sleep.\n");

// Wait for the child process to finish

wait(NULL);

printf("Child process has terminated.\n");

}

return 0;

}

QUE2 (FCFS)

#include<stdio.h>

struct input

{

char pname[10];

int bt,at,tbt,ft;

}tab[10];

struct gantt

{

int start,end;

char pname[10];

}g[50],g1[10];

int n,i,k,time,prev;

void getinput({

printf("\nEnter No of Processes: ");

scanf("%d",&n);

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

{

printf("\nProcess Name: ");

scanf("%s",tab[i].pname);

printf("Burst Time: ");

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

tab[i].tbt=tab[i].bt;

printf("Arrival Time:" );

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

}

}

void printinput(){

printf("\nPname\tBT\tAT");

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

printf("\n%s\t%d\t%d",tab[i].pname,tab[i].tbt,tab[i].at);

}

void sort(){

struct input temp;

int j;

for(i=1;i<n;i++)//pass

for(j=0;j<n-1;j++)//Comp

if(tab[j].at>tab[j+1].at)

{

temp=tab[j];

tab[j]=tab[j+1];

tab[j+1]=temp;

}

}

int arrived(int time){

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

if(tab[i].at<=time && tab[i].tbt!=0)

return 1;

return 0;

}

void processinput({

int j,finish=0;

// time=tab[0].at;

while(finish!=n)

{

if(arrived(time))

{

for(j=0;j<tab[i].bt;j++)

{

time++;

tab[i].tbt--;

g[k].start=prev;

g[k].end=time;

// printinput();

prev=time;

tab[i].ft=time;

strcpy(g[k++].pname,tab[i].pname);

if(tab[i].tbt==0)

{

finish++;

break;

}

}

}

else

{

time++;

g[k].start=prev;

g[k].end=time;

prev=time;

strcpy(g[k++].pname,"idle");

}

}

// printinput();

}

void printoutput(){

int TTAT=0,TWT=0;

float ATAT,AWT;

printf("\n\*\*\*\*\*\*Final Table\*\*\*\*\*");

printf("\nPname\tAT\tBT\tFT\tTAT\tWT");

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

{

printf("\n%s\t%d\t%d\t%d\t%d\t%d",tab[i].pname,tab[i].at,tab[i].bt,tab[i].ft,tab[i].ft-tab[i].at,tab[i].ft-tab[i].at-tab[i].bt);

TTAT=TTAT+(tab[i].ft-tab[i].at);

TWT=TWT+(tab[i].ft-tab[i].at-tab[i].bt);

}

ATAT=(float)TTAT/n;

AWT=(float)TWT/n;

printf("\nTotal TAT=%d",TTAT);

printf("\nTotal WT=%d",TWT);

printf("\nAverage TAT=%f",ATAT);

printf("\nAverage WT=%f",AWT);

}

void printganttchart({

int j=0;

g1[0]=g[0];

for(i=1;i<k;i++)

{

if(strcmp(g1[j].pname,g[i].pname)==0)

g1[j].end=g[i].end;

else

{

j++;

g1[j]=g[i];

}

}

printf("\n\*\*\*\*\*\*Each unit Gantt chart\*\*\*\*\*\*");

for(i=0;i<k;i++)

printf("\n%d\t%s\t%d",g[i].start,g[i].pname,g[i].end);

printf("\n\*\*\*\*\*\*Final Gantt chart\*\*\*\*\*\*");

for(i=0;i<=j;i++)

printf("\n%d\t%s\t%d",g1[i].start,g1[i].pname,g1[i].end);

}

int main()

{

getinput();

printf("\nEntered data is: ");

printinput();

sort();

printf("\nData after Sorting" );

printinput();

processinput();

printoutput();

printganttchart();

}

QUE2 (LRU)

#include<stdio.h>

int RefString[20],PT[10],nof,nor;

void Accept()

{

int i;

printf("Enter Reference String: \n");

for(i=0;i<nor;i++)

{

printf("[%d]=",i);

scanf("%d",&RefString[i]);

}

}

int Search(int s)

{

int i;

for(i=0;i<nof; i++)

if(PT[i]==s)

return(i);

return(-1);

}

int GetLRU(int e)

{

int i,j,Pos=99,Posi,k;

for(i=0;i<nof;i++)

{

for(j=e-1;j>=0;j--)

{

if(PT[i]==RefString[j])

{

if(j<Pos)

{

Pos=j;

Posi=i;

}

break;

}

}

}

return(Posi);

}

void LRU()

{

int i,j,k,Faults=0;

for(k=0,i=0; k<nof && i<nor; i++)

{

printf("\n\n%2d ",RefString[i]);

if(Search(RefString[i])==-1)

{

PT[k]=RefString[i];

for(j=0;j<nof;j++)

{

if(PT[j])

printf(" %2d",PT[j]);

}

Faults++;

k++;

}

}

while(i<nor)

{

printf("\n\n%2d ",RefString[i]);

if(Search(RefString[i])==-1)

{

k = GetLRU(i);

PT[k]=RefString[i];

for(j=0;j<nof;j++)

{

printf(" %2d",PT[j]);

}

Faults++;

}

i++;

}

printf("\nTotal Page Faults: %d",Faults);

}

void main()

{

printf("Enter Length of reference string: ");

scanf("%d",&nor);

printf("Enter No.of Frames: ");

scanf("%d",&nof);

Accept();

LRU();

}

SLIP8

QUE1

#include <stdio.h>

#include <stdlib.h>

void calculateNeedMatrix(int \*\*maxMatrix, int \*\*allocationMatrix, int \*\*needMatrix, int processes, int resources) {

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

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

needMatrix[i][j] = maxMatrix[i][j] - allocationMatrix[i][j];

}

}

}

void displayMatrix(int \*\*matrix, int rows, int cols) {

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

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

printf("%d ", matrix[i][j]);

}

printf("\n");

}

}

int main() {

int processes, resources;

// Accept the number of processes and resources

printf("Enter the number of processes: ");

scanf("%d", &processes);

printf("Enter the number of resources: ");

scanf("%d", &resources);

// Allocate memory for matrices

int \*\*maxMatrix = (int \*\*)malloc(processes \* sizeof(int \*));

int \*\*allocationMatrix = (int \*\*)malloc(processes \* sizeof(int \*));

int \*\*needMatrix = (int \*\*)malloc(processes \* sizeof(int \*));

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

maxMatrix[i] = (int \*)malloc(resources \* sizeof(int));

allocationMatrix[i] = (int \*)malloc(resources \* sizeof(int));

needMatrix[i] = (int \*)malloc(resources \* sizeof(int));

}

// Input the maximum matrix

printf("Enter the Maximum matrix:\n");

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

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

printf("Maximum[%d][%d]: ", i, j);

scanf("%d", &maxMatrix[i][j]);

}

}

// Input the allocation matrix

printf("Enter the Allocation matrix:\n");

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

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

printf("Allocation[%d][%d]: ", i, j);

scanf("%d", &allocationMatrix[i][j]);

}

}

// Calculate and display the need matrix

calculateNeedMatrix(maxMatrix, allocationMatrix, needMatrix, processes, resources);

printf("\nNeed Matrix:\n");

displayMatrix(needMatrix, processes, resources);

// Free allocated memory

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

free(maxMatrix[i]);

free(allocationMatrix[i]);

free(needMatrix[i]);

}

free(maxMatrix);

free(allocationMatrix);

free(needMatrix);

return 0;

}

QUE2 (OPTIMAL)

#include<stdio.h>

int RefString[20],PT[10],nof,nor;

void Accept()

{

int i;

printf("Enter Reference String: \n");

for(i=0;i<nor;i++)

{

printf("[%d]=",i);

scanf("%d",&RefString[i]);

}

}

int Search(int s)

{

int i;

for(i=0;i<nof; i++)

if(PT[i]==s)

return(i);

return(-1);

}

int Getoptimal(int e)

{

int i,j,Pos=0,Posi,k,flag;

for(i=0;i<nof;i++)

{

flag=0;

for(j=e+1;j<=nor;j++)

{

if(PT[i]==RefString[j])

{

flag=1;

if(j>Pos)

{

Pos=j;

Posi=i;

}

break;

}

}

if(flag==0)

{

Posi=i;

break;

}

}

return(Posi);

}

void optimal()

{

int i,j,k,Faults=0;

for(k=0,i=0; k<nof && i<nor; i++)

{

printf("\n%2d",RefString[i]);

if(Search(RefString[i])==-1)

{

PT[k]=RefString[i];

for(j=0;j<nof;j++)

{

if(PT[j])

printf("%2d",PT[j]);

}

Faults++;

k++;

}

}

while(i<nor)

{

printf("\n%2d",RefString[i]);

if(Search(RefString[i])==-1)

{

k = Getoptimal(i);

PT[k]=RefString[i];

for(j=0;j<nof;j++)

{

printf("%2d",PT[j]);

}

Faults++;

}

i++;

}

printf("\nTotal Page Faults: %d",Faults);

}

void main()

{

printf("Enter Length of reference string: ");

scanf("%d",&nor);

printf("Enter No.of Frames: ");

scanf("%d",&nof);

Accept();

optimal();

}

SLIP9

QUE1

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/wait.h>

int main() {

pid\_t pid;

// Fork a child process

pid = fork();

if (pid < 0) {

// Error occurred

fprintf(stderr, "Fork failed\n");

return 1;

} else if (pid == 0) {

// Child process

// Use execl() to execute the "ls" command

execl("/bin/ls", "ls", (char \*)NULL);

// If execl fails

perror("execl");

exit(1);

} else {

// Parent process

// Parent goes to sleep state

sleep(5);

// Parent process may continue its own execution here

printf("Parent process waking up after sleep.\n");

// Wait for the child process to finish

wait(NULL);

printf("Child process has terminated.\n");

}

return 0;

}

QUE2

Menu driven bankers need to do

QUE2 (ROUND ROBIN)

#include<stdio.h>

#include<string.h>

struct input

{

char pname[10];

int bt,at,tbt,ft;

}tab[10];

struct gantt

{

char pname[10];

int start,end;

}g[30],g1[30];

int n,time,prev,k,tq;

void getinput()

{

int i;

printf("\nEnter No of Process: ");

scanf("%d",&n);

printf("\nEnter Time quantum: ");

scanf("%d",&tq);

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

{

printf("\nEnter Process Name: ");

scanf("%s",tab[i].pname);

printf("Arrival Time:");

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

printf("Burst Time: ");

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

tab[i].tbt=tab[i].bt;

}

}

void printinput()

{

// int TWT=0,TTAT=0;

int i;

printf("\nPname\tAT\tBT");

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

printf("\n%s\t%d\t%d",tab[i].pname,tab[i].at,tab[i].tbt);

}

void printoutput()

{

int TWT=0,TTAT=0,i;

float ATAT,AWT;

printf("\nPname\tAT\tBT\tFT\tWT\tTAT");

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

{

printf("\n%s\t%d\t%d\t%d\t%d\t%d",tab[i].pname,tab[i].at,tab[i].bt,tab[i].ft,tab[i].ft-tab[i].at-tab[i].bt,tab[i].ft-tab[i].at);

TWT=TWT+(tab[i].ft-tab[i].at-tab[i].bt);

TTAT=TTAT+(tab[i].ft-tab[i].at);

}

printf("\nTotal WT: %d",TWT);

printf("\nTotal TAT:%d",TTAT);

AWT=(float)TWT/n;

ATAT=(float)TTAT/n;

printf("\nAverage WT: %f",AWT);

printf("\nAverage TAT:%f",ATAT);

}

void sort()

{

int pass,i;

struct input temp;

for(pass=1;pass<n;pass++)

{

for(i=0;i<n-pass;i++)

{

if(tab[i].at>tab[i+1].at)

{

temp=tab[i];

tab[i]=tab[i+1];

tab[i+1]=temp;

}

}

}

}

int arrived(int time)

{

int i;

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

{

if(tab[i].at<=time && tab[i].tbt!=0)

return 1;

}

return 0;

}

void processinput()

{

int finish=0,j;

int i=0;

k=0;

while(finish!=n)

{

if(arrived(time))

{

if(tab[i].tbt!=0)

{

for(j=0;j<tq;j++)

{

time++;

tab[i].tbt--;

g[k].start=prev;

g[k].end=time;

prev=time;

strcpy(g[k++].pname,tab[i].pname);

tab[i].ft=time;

if(tab[i].tbt==0)

{

finish++;

break;

}

}

}

}

else

{

time++;

g[k].start=prev;

g[k].end=time;

strcpy(g[k++].pname,"idle");

prev=time;

}

if(time<tab[(i+1)%n].at)

i=0;

else

i=(i+1)%n;

}

}

void ganttchart()

{

int i,j=0;

printf("\n\*\*\*\*\*\*Each Unit Gantt chart\*\*\*\*\*\*");

printf("\nStart\tpname\tEnd");

for(i=0;i<k;i++)

{

printf("\n%d\t%s\t%d",g[i].start,g[i].pname,g[i].end);

}

printf("\n\*\*\*\*\*\*\*\*Final Gantt Chart\*\*\*\*\*\*\*");

g1[0]=g[0];

for(i=1;i<k;i++)

{

if(strcmp(g[i].pname,g1[j].pname)==0)

g1[j].end=g[i].end;

else

{

j++;

g1[j]=g[i];

}

}

printf("\nStart\tpname\tEnd");

for(i=0;i<=j;i++)

{

printf("\n%d\t%s\t%d",g1[i].start,g1[i].pname,g1[i].end);

}

}

int main()

{

getinput();

printinput();

sort();

printf("\nData After Sorting: ");

printinput();

processinput();

printoutput();

ganttchart();

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

{

tab[i].tbt=tab[i].bt=rand()%10+1;

tab[i].at=tab[i].ft+2;

}

printinput();

processinput();

printoutput();

ganttchart();

}

SLIP10

QUE1

#include<stdio.h>

#include<unistd.h>

int main()

{

int pid,a;

printf("\n the process id is%d",getpid());

pid=fork();

if(pid<0)

printf("\n Fork failed!");

else if(pid==0)

{

printf("\n I am Child process");

printf("\n Child process ID=%d",getpid());

}

else

{

wait(NULL);

sleep(5);

printf("\n I am Parent process");

printf("\n Child Parent ID=%d",getpid());

}

}

QUE2 (OPTIMAL)

#include<stdio.h>

int RefString[20],PT[10],nof,nor;

void Accept()

{

int i;

printf("Enter Reference String: \n");

for(i=0;i<nor;i++)

{

printf("[%d]=",i);

scanf("%d",&RefString[i]);

}

}

int Search(int s)

{

int i;

for(i=0;i<nof; i++)

if(PT[i]==s)

return(i);

return(-1);

}

int Getoptimal(int e)

{

int i,j,Pos=0,Posi,k,flag;

for(i=0;i<nof;i++)

{

flag=0;

for(j=e+1;j<=nor;j++)

{

if(PT[i]==RefString[j])

{

flag=1;

if(j>Pos)

{

Pos=j;

Posi=i;

}

break;

}

}

if(flag==0)

{

Posi=i;

break;

}

}

return(Posi);

}void optimal()

{

int i,j,k,Faults=0;

for(k=0,i=0; k<nof && i<nor; i++)

{

printf("\n%2d",RefString[i]);

if(Search(RefString[i])==-1)

{

PT[k]=RefString[i];

for(j=0;j<nof;j++)

{

if(PT[j])

printf("%2d",PT[j]);

}

Faults++;

k++;

}

}

while(i<nor)

{

printf("\n%2d",RefString[i]);

if(Search(RefString[i])==-1)

{

k = Getoptimal(i);

PT[k]=RefString[i];

for(j=0;j<nof;j++)

{

printf("%2d",PT[j]);

}

Faults++;

}

i++;

}

printf("\nTotal Page Faults: %d",Faults);

}

void main()

{

printf("Enter Length of reference string: ");

scanf("%d",&nor);

printf("Enter No.of Frames: ");

scanf("%d",&nof);

Accept();

optimal();

}

QUE2 (FCFS)

#include<stdio.h>

struct input

{

char pname[10];

int bt,at,tbt,ft;

}tab[10];

struct gantt

{

int start,end;

char pname[10];

}g[50],g1[10];

int n,i,k,time,prev;

void getinput()

{

printf("\nEnter No of Processes: ");

scanf("%d",&n);

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

{

printf("\nProcess Name: ");

scanf("%s",tab[i].pname);

printf("Burst Time: ");

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

tab[i].tbt=tab[i].bt;

printf("Arrival Time:" );

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

}

}

void printinput()

{

printf("\nPname\tBT\tAT");

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

printf("\n%s\t%d\t%d",tab[i].pname,tab[i].tbt,tab[i].at);

}

void sort()

{

struct input temp;

int j;

for(i=1;i<n;i++)//pass

for(j=0;j<n-1;j++)//Comp

if(tab[j].at>tab[j+1].at)

{

temp=tab[j];

tab[j]=tab[j+1];

tab[j+1]=temp;

}

}

int arrived(int time)

{

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

if(tab[i].at<=time && tab[i].tbt!=0)

return 1;

return 0;

}

void processinput()

{

int j,finish=0;

// time=tab[0].at;

while(finish!=n)

{

if(arrived(time))

{

for(j=0;j<tab[i].bt;j++)

{

time++;

tab[i].tbt--;

g[k].start=prev;

g[k].end=time;

// printinput();

prev=time;

tab[i].ft=time;

strcpy(g[k++].pname,tab[i].pname);

if(tab[i].tbt==0)

{

finish++;

break;

}

}

}

else

{

time++;

g[k].start=prev;

g[k].end=time;

prev=time;

strcpy(g[k++].pname,"idle");

}

}

// printinput();

}

void printoutput()

{

int TTAT=0,TWT=0;

float ATAT,AWT;

printf("\n\*\*\*\*\*\*Final Table\*\*\*\*\*");

printf("\nPname\tAT\tBT\tFT\tTAT\tWT");

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

{

printf("\n%s\t%d\t%d\t%d\t%d\t%d",tab[i].pname,tab[i].at,tab[i].bt,tab[i].ft,tab[i].ft-tab[i].at,tab[i].ft-tab[i].at-tab[i].bt);

TTAT=TTAT+(tab[i].ft-tab[i].at);

TWT=TWT+(tab[i].ft-tab[i].at-tab[i].bt);

}

ATAT=(float)TTAT/n;

AWT=(float)TWT/n;

printf("\nTotal TAT=%d",TTAT);

printf("\nTotal WT=%d",TWT);

printf("\nAverage TAT=%f",ATAT);

printf("\nAverage WT=%f",AWT);

}

void printganttchart()

{

int j=0;

g1[0]=g[0];

for(i=1;i<k;i++)

{

if(strcmp(g1[j].pname,g[i].pname)==0)

g1[j].end=g[i].end;

else

{

j++;

g1[j]=g[i];

}

}

printf("\n\*\*\*\*\*\*Each unit Gantt chart\*\*\*\*\*\*");

for(i=0;i<k;i++)

printf("\n%d\t%s\t%d",g[i].start,g[i].pname,g[i].end);

printf("\n\*\*\*\*\*\*Final Gantt chart\*\*\*\*\*\*");

for(i=0;i<=j;i++)

printf("\n%d\t%s\t%d",g1[i].start,g1[i].pname,g1[i].end);

}

int main()

{

getinput();

printf("\nEntered data is: ");

printinput();

sort();

printf("\nData after Sorting" );

printinput();

processinput();

printoutput();

printganttchart();

}

SLIP11

QUE1

#include<stdio.h>

#include<unistd.h>

void childprocess(){

printf("\n HELLO WORLD");

}

void parentprocess(){

printf("\nHI");

}

int main(){

int pid;

pid=fork();

if(pid==0)

{

printf("Child Process ID=%d",pid);

childprocess();

}

else

{

printf("\n Parent Process Id=%d",pid);

parentprocess();

}

}

QUE2 (FIFO)

#include<stdio.h>

int RefString[20],PT[10],nof,nor;

void Accept()

{

int i;

printf("Enter Reference String: \n");

for(i=0;i<nor;i++)

{

printf("[%d]=",i);

scanf("%d",&RefString[i]);

}

}

int Search(int s)

{

int i;

for(i=0;i<nof; i++)

if(PT[i]==s)

return(i);

return(-1);

}void FIFO()

{

int i,j,k=0,Faults=0;

for(i=0;i<nor;i++)

{

printf("\n%2d",RefString[i]);

if(Search(RefString[i])==-1)

{

PT[k]=RefString[i];

for(j=0;j<nof;j++)

{

if(PT[j])

{

printf("\t%2d",PT[j]);

}

}

Faults++;

k=(k+1)%nof;

}

}

printf("\nTotal Page Faults: %d",Faults);

}

void main()

{

printf("Enter Length of reference string: ");

scanf("%d",&nor);

printf("Enter No. of Frames: ");

scanf("%d",&nof);

Accept();

FIFO();

}

SLIP12

\*\*\* SAME AS SLIP NO. 10 \*\*\*

SLIP 13

QUE1

#include<stdio.h>

int main(){

int pid,retnice,i;

pid=fork();

for(i=0;i<3;i++)

{

if(pid==0)

{

retnice=nice(-5);

printf("Child gets higher CPU priority%d\n",retnice);

sleep(1);

}

else

{

retnice=nice(4);

printf("Parent gets lower CPU Priority%d\n",retnice);

sleep(1);

}

}

QUE2 (BANKER’S ALGO)

#include<stdio.h>

int nop,nor,A[10][10],M[10][10],Av[10],N[10][10],finish[10];

void acceptdata(int x[10][10]){

int i,j;

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

printf("P%d\n",i);

for(j=0;j<nor;j++)

{

printf("%c: ",65+j);

scanf("%d",&x[i][j]);

}

}

}

void acceptav(){

int i;

for(i=0;i<nor;i++)

{

printf("%c: ",65+i);

scanf("%d",&Av[i]);

}

}

void calcneed()

{

int i,j;

for(i=0;i<nop;i++)

for(j=0;j<nor;j++)

N[i][j]=M[i][j]-A[i][j];

}

void displaydata(){

int i,j;

printf("\n\tAllocation \t\tMax\t\tNeed\n\t");

for(i=0;i<3;i++)

{

for(j=0;j<nor;j++)

printf("%4c",65+j);

printf("\t");

}

for(i=0;i<nop;i++)

{

printf("\nP%d\t",i);

for(j=0;j<nor;j++)

printf("%4d",A[i][j]);

printf("\t");

for(j=0;j<nor;j++)

printf("%4d",M[i][j]);

printf("\t");

for(j=0;j<nor;j++)

printf("%4d",N[i][j]);

}

printf("\navailable");

for(i=0;i<nor;i++)

printf("%4d",Av[i]);

}

int checkneed(int pno){

int i;

for(i=0;i<nor;i++)

if(N[pno][i]>Av[i])

return 0;

return 1;

}

void banker(){

int p=0,j=0,k=0,flag=0,safe[10];

while(flag<2){

if(!finish[p])

{

printf("\n\nNeed of process P%d (,",p);

for(j=0;j<nor;j++)

printf("%d,",N[p][j]);

if(checkneed(p))

{

printf(") <= available (");

for(j=0;j<nor;j++)

printf("%d,",Av[j]);

printf(")");

printf("\nNeed is Satsified, So process P%d can be granted requiered resources.\n After P%d finishes, it will realease all the resources.",p,p);

for(j=0;j<nor;j++)

Av[j]=Av[j]+A[p][j];

printf("New Availble=");

for(j=0;j<nor;j++)

printf("%d ",Av[j]);

finish[p]=1;

safe[k++]=p;

}

else{

printf(") > available (");

for(j=0;j<nor;j++)

printf("%d,",Av[j]);

printf(")");

printf("\nNeed is not Satsified, So process P%d cannot be granted required resources.\n process P%d has to wait.",p,p);

}

}

if((p+1)%nop==0)

flag++;

p=(p+1)%nop;

}//while

if(k==nop)

{

printf("\nSystem is in safe state...");

printf("\nSafe Sequence: ");

for(j=0;j<k;j++)

printf("P%d->",safe[j]);

}

else

printf("\nSystem is not in safe state....");

}

main()

{

printf("\nEnter No of Processes: ");

scanf("%d",&nop);

printf("\nEnter No. of Resources: ");

scanf("%d",&nor);

printf("\nEnter Allocation Matrix: ");

acceptdata(A);

printf("\nEnter Max Matrix: ");

acceptdata(M);

printf("\nEnter Availability:");

acceptav();

calcneed();

displaydata();

banker();

}

QUE2 (SJF NON-PREEMPTIVE)

#include<stdio.h>

struct inpu{

char pname[10];

int bt,at,tbt,ft;

}tab[10];

struct gantt

{

int start,end;

char pname[10];

}g[50],g1[10];

int n,i,k,time,prev;

void getinput()

{

printf("\nEnter No of Processes: ");

scanf("%d",&n);

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

{

printf("\nProcess Name: ");

scanf("%s",tab[i].pname);

printf("Burst Time: ");

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

tab[i].tbt=tab[i].bt;

printf("Arrival Time:" );

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

}

}

void printinput()

{

printf("\nPname\tBT\tAT");

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

printf("\n%s\t%d\t%d",tab[i].pname,tab[i].tbt,tab[i].at);

}

void sort()

{

struct input temp;

int j;

for(i=1;i<n;i++)//pass

for(j=0;j<n-1;j++)//Comp

if(tab[j].at>tab[j+1].at)

{

temp=tab[j];

tab[j]=tab[j+1];

tab[j+1]=temp;

}

}

int arrived(int time)

{

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

if(tab[i].at<=time && tab[i].tbt!=0)

return 1;

return 0;

}int getsmallburst(int time)

{

int min=99,mini;

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

{

if(tab[i].tbt<min && tab[i].at<=time && tab[i].tbt!=0)

{

min=tab[i].tbt;

mini=i;

}

}

return mini;

}

void processinput()

{

int j,finish=0;

// time=tab[0].at;

while(finish!=n)

{

if(arrived(time))

{

i=getsmallburst(time);

for(j=0;j<tab[i].bt;j++)

{

time++;

tab[i].tbt--;

g[k].start=prev;

g[k].end=time;

// printinput();

prev=time;

tab[i].ft=time;

strcpy(g[k++].pname,tab[i].pname);

if(tab[i].tbt==0)

{

finish++;

break;

}

}

}

else

{

time++;

g[k].start=prev;

g[k].end=time;

prev=time;

strcpy(g[k++].pname,"idle");

}

// printinput();

}

void printoutput()

{

int TTAT=0,TWT=0;

float ATAT,AWT;

printf("\n\*\*\*\*\*\*Final Table\*\*\*\*\*");

printf("\nPname\tAT\tBT\tFT\tTAT\tWT");

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

{

printf("\n%s\t%d\t%d\t%d\t%d\t%d",tab[i].pname,tab[i].at,tab[i].bt,tab[i].ft,tab[i].ft-tab[i].at,tab[i].ft-tab[i].at-tab[i].bt);

TTAT=TTAT+(tab[i].ft-tab[i].at);

TWT=TWT+(tab[i].ft-tab[i].at-tab[i].bt);

}

ATAT=(float)TTAT/n;

AWT=(float)TWT/n;

printf("\nTotal TAT=%d",TTAT);

printf("\nTotal WT=%d",TWT);

printf("\nAverage TAT=%f",ATAT);

printf("\nAverage WT=%f",AWT);

}

void printganttchart()

{

int j=0;

g1[0]=g[0];

for(i=1;i<k;i++)

{

if(strcmp(g1[j].pname,g[i].pname)==0)

g1[j].end=g[i].end;

else

{

j++;

g1[j]=g[i];

}

}

printf("\n\*\*\*\*\*\*Each unit Gantt chart\*\*\*\*\*\*");

for(i=0;i<k;i++)

printf("\n%d\t%s\t%d",g[i].start,g[i].pname,g[i].end);

printf("\n\*\*\*\*\*\*Final Gantt chart\*\*\*\*\*\*");

for(i=0;i<=j;i++)

printf("\n%d\t%s\t%d",g1[i].start,g1[i].pname,g1[i].end);

}

int main(){

getinput();

printf("\nEntered data is: ");

printinput();

sort();

printf("\nData after Sorting" );

printinput();

processinput();

printoutput();

printganttchart();

}

SLIP14

QUE1, QUE2

\*\* SAME AS SLIP NO.6 \*\*

SLIP15

QUE1, QUE2

\*\* SAME AS SLIP NO.7 \*\*

SLIP 16

QUE1

#include <stdio.h>

#include <time.h>

// Function to perform the set of instructions

void performInstructions() {

// Replace this with the set of instructions you want to measure

for (int i = 0; i < 1000000; ++i) {

// Some time-consuming instructions

int result = i \* i;

}

}

int main() {

// Record the starting time

clock\_t start\_time = clock();

// Call the function with the set of instructions

performInstructions();

// Record the ending time

clock\_t end\_time = clock();

// Calculate the execution time in seconds

double execution\_time = ((double)(end\_time - start\_time)) / CLOCKS\_PER\_SEC;

// Print the execution time

printf("Execution time: %f seconds\n", execution\_time);

return 0;

}

QUE2 (OPTIMAL)

#include<stdio.h>

int RefString[20],PT[10],nof,nor;

void Accept()

{

int i;

printf("Enter Reference String: \n");

for(i=0;i<nor;i++)

{

printf("[%d]=",i);

scanf("%d",&RefString[i]);

}

}

int Search(int s)

{

int i;

for(i=0;i<nof; i++)

if(PT[i]==s)

return(i);

return(-1);

}

int Getoptimal(int e)

{

int i,j,Pos=0,Posi,k,flag;

for(i=0;i<nof;i++)

{

flag=0;

for(j=e+1;j<=nor;j++)

{

if(PT[i]==RefString[j])

{

flag=1;

if(j>Pos)

{

Pos=j;

Posi=i;

}

break;

}

}

if(flag==0)

{

Posi=i;

break;

}

}

return(Posi);

}

void optimal()

{

int i,j,k,Faults=0;

for(k=0,i=0; k<nof && i<nor; i++)

{

printf("\n%2d",RefString[i]);

if(Search(RefString[i])==-1)

{

PT[k]=RefString[i];

for(j=0;j<nof;j++)

{

if(PT[j])

printf("%2d",PT[j]);

}

Faults++;

k++;

}

}

while(i<nor)

{

printf("\n%2d",RefString[i]);

if(Search(RefString[i])==-1)

{

k = Getoptimal(i);

PT[k]=RefString[i];

for(j=0;j<nof;j++)

{

printf("%2d",PT[j]);

}

Faults++;

}

i++;

}

printf("\nTotal Page Faults: %d",Faults);

}

void main()

{

printf("Enter Length of reference string: ");

scanf("%d",&nor);

printf("Enter No.of Frames: ");

scanf("%d",&nof);

Accept();

optimal();

}

SLIP 17

QUE1

QUE2 (OPTIMAL)

#include<stdio.h>

int RefString[20],PT[10],nof,nor;

void Accept()

{

int i;

printf("Enter Reference String: \n");

for(i=0;i<nor;i++)

{

printf("[%d]=",i);

scanf("%d",&RefString[i]);

}

}

int Search(int s)

{

int i;

for(i=0;i<nof; i++)

if(PT[i]==s)

return(i);

return(-1);

}

int Getoptimal(int e)

{

int i,j,Pos=0,Posi,k,flag;

for(i=0;i<nof;i++)

{

flag=0;

for(j=e+1;j<=nor;j++)

{

if(PT[i]==RefString[j])

{

flag=1;

if(j>Pos)

{

Pos=j;

Posi=i;

}

break;

}

}

if(flag==0)

{

Posi=i;

break;

}

}

return(Posi);

}

void optimal()

{

int i,j,k,Faults=0;

for(k=0,i=0; k<nof && i<nor; i++)

{

printf("\n%2d",RefString[i]);

if(Search(RefString[i])==-1)

{

PT[k]=RefString[i];

for(j=0;j<nof;j++)

{

if(PT[j])

printf("%2d",PT[j]);

}

Faults++;

k++;

}

}

while(i<nor)

{

printf("\n%2d",RefString[i]);

if(Search(RefString[i])==-1)

{

k = Getoptimal(i);

PT[k]=RefString[i];

for(j=0;j<nof;j++)

{

printf("%2d",PT[j]);

}

Faults++;

}

i++;

}

printf("\nTotal Page Faults: %d",Faults);

}

void main()

{

printf("Enter Length of reference string: ");

scanf("%d",&nor);

printf("Enter No.of Frames: ");

scanf("%d",&nof);

Accept();

optimal();

}

QUE2 (FCFS)

#include<stdio.h>

struct input{

char pname[10];

int bt,at,tbt,ft;

}tab[10];

struct gantt

{

int start,end;

char pname[10];

}g[50],g1[10];

int n,i,k,time,prev;

void getinput()

{

printf("\nEnter No of Processes: ");

scanf("%d",&n);

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

{

printf("\nProcess Name: ");

scanf("%s",tab[i].pname);

printf("Burst Time: ");

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

tab[i].tbt=tab[i].bt;

printf("Arrival Time:" );

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

}

}

void printinput()

{

printf("\nPname\tBT\tAT");

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

printf("\n%s\t%d\t%d",tab[i].pname,tab[i].tbt,tab[i].at);

}

void sort()

{

struct input temp;

int j;

for(i=1;i<n;i++)//pass

for(j=0;j<n-1;j++)//Comp

if(tab[j].at>tab[j+1].at)

{

temp=tab[j];

tab[j]=tab[j+1];

tab[j+1]=temp;

}

}

int arrived(int time)

{

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

if(tab[i].at<=time && tab[i].tbt!=0)

return 1;

return 0;

}

void processinput()

{

int j,finish=0;

// time=tab[0].at;

while(finish!=n)

{

if(arrived(time))

{

for(j=0;j<tab[i].bt;j++)

{

time++;

tab[i].tbt--;

g[k].start=prev;

g[k].end=time;

// printinput();

prev=time;

tab[i].ft=time;

strcpy(g[k++].pname,tab[i].pname);

if(tab[i].tbt==0)

{

finish++;

break;

}

}

}

else

{

time++;

g[k].start=prev;

g[k].end=time;

prev=time;

strcpy(g[k++].pname,"idle");

}

}

// printinput();

}

void printoutput()

{

int TTAT=0,TWT=0;

float ATAT,AWT;

printf("\n\*\*\*\*\*\*Final Table\*\*\*\*\*");

printf("\nPname\tAT\tBT\tFT\tTAT\tWT");

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

{

printf("\n%s\t%d\t%d\t%d\t%d\t%d",tab[i].pname,tab[i].at,tab[i].bt,tab[i].ft,tab[i].ft-tab[i].at,tab[i].ft-tab[i].at-tab[i].bt);

TTAT=TTAT+(tab[i].ft-tab[i].at);

TWT=TWT+(tab[i].ft-tab[i].at-tab[i].bt);

}

ATAT=(float)TTAT/n;

AWT=(float)TWT/n;

printf("\nTotal TAT=%d",TTAT);

printf("\nTotal WT=%d",TWT);

printf("\nAverage TAT=%f",ATAT);

printf("\nAverage WT=%f",AWT);

}

void printganttchart()

{

int j=0;

g1[0]=g[0];

for(i=1;i<k;i++)

{

if(strcmp(g1[j].pname,g[i].pname)==0)

g1[j].end=g[i].end;

else

{

j++;

g1[j]=g[i];

}

}

printf("\n\*\*\*\*\*\*Each unit Gantt chart\*\*\*\*\*\*");

for(i=0;i<k;i++)

printf("\n%d\t%s\t%d",g[i].start,g[i].pname,g[i].end);

printf("\n\*\*\*\*\*\*Final Gantt chart\*\*\*\*\*\*");

for(i=0;i<=j;i++)

printf("\n%d\t%s\t%d",g1[i].start,g1[i].pname,g1[i].end);

}

int main()

{

getinput();

printf("\nEntered data is: ");

printinput();

sort();

printf("\nData after Sorting" );

printinput();

processinput();

printoutput();

printganttchart();

}

SLIP18

QUE1, QUE2

\*\*\* SAME AS SLIP NO.8 \*\*\*

SLIP19

QUE1

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/wait.h>

int main() {

pid\_t pid;

// Fork a child process

pid = fork();

if (pid < 0) {

// Error occurred

fprintf(stderr, "Fork failed\n");

return 1;

} else if (pid == 0) {

// Child process

// Use execl() to execute the "ls" command

execl("/bin/ls", "ls", (char \*)NULL);

// If execl fails

perror("execl");

exit(1);

} else {

// Parent process

// Parent goes to sleep state

sleep(5);

// Parent process may continue its own execution here

printf("Parent process waking up after sleep.\n");

// Wait for the child process to finish

wait(NULL);

printf("Child process has terminated.\n");

}

return 0;

}

QUE2 (BANKER’S ALGO)

#include<stdio.h>

int nop,nor,A[10][10],M[10][10],Av[10],N[10][10],finish[10];

void acceptdata(int x[10][10])

{

int i,j;

for(i=0;i<nop;i++)

{

printf("P%d\n",i);

for(j=0;j<nor;j++)

{

printf("%c: ",65+j);

scanf("%d",&x[i][j]);

}

}

}

void acceptav()

{

int i;

for(i=0;i<nor;i++)

{

printf("%c: ",65+i);

scanf("%d",&Av[i]);

}

}

void calcneed()

{

int i,j;

for(i=0;i<nop;i++)

for(j=0;j<nor;j++)

N[i][j]=M[i][j]-A[i][j];

}

void displaydata()

{

int i,j;

printf("\n\tAllocation \t\tMax\t\tNeed\n\t");

for(i=0;i<3;i++)

{

for(j=0;j<nor;j++)

printf("%4c",65+j);

printf("\t");

}

for(i=0;i<nop;i++)

{

printf("\nP%d\t",i);

for(j=0;j<nor;j++)

printf("%4d",A[i][j]);

printf("\t");

for(j=0;j<nor;j++)

printf("%4d",M[i][j]);

printf("\t");

for(j=0;j<nor;j++)

printf("%4d",N[i][j]);

}

printf("\navailable");

for(i=0;i<nor;i++)

printf("%4d",Av[i]);

}

int checkneed(int pno)

{

int i;

for(i=0;i<nor;i++)

if(N[pno][i]>Av[i])

return 0;

return 1;

}

void banker()

{

int p=0,j=0,k=0,flag=0,safe[10];

while(flag<2)

{

if(!finish[p])

{

printf("\n\nNeed of process P%d (,",p);

for(j=0;j<nor;j++)

printf("%d,",N[p][j]);

if(checkneed(p))

{

printf(") <= available (");

for(j=0;j<nor;j++)

printf("%d,",Av[j]);

printf(")");

printf("\nNeed is Satsified, So process P%d can be granted requiered resources.\n After P%d finishes, it will realease all the resources.",p,p);

for(j=0;j<nor;j++)

Av[j]=Av[j]+A[p][j];

printf("New Availble=");

for(j=0;j<nor;j++)

printf("%d ",Av[j]);

finish[p]=1;

safe[k++]=p;

}

else

{

printf(") > available (");

for(j=0;j<nor;j++)

printf("%d,",Av[j]);

printf(")");

printf("\nNeed is not Satsified, So process P%d cannot be granted required resources.\n process P%d has to wait.",p,p);

}

}

if((p+1)%nop==0)

flag++;

p=(p+1)%nop;

}//while

if(k==nop)

{

printf("\nSystem is in safe state...");

printf("\nSafe Sequence: ");

for(j=0;j<k;j++)

printf("P%d->",safe[j]);

}

else

printf("\nSystem is not in safe state....");

}

main()

{

printf("\nEnter No of Processes: ");

scanf("%d",&nop);

printf("\nEnter No. of Resources: ");

scanf("%d",&nor);

printf("\nEnter Allocation Matrix: ");

acceptdata(A);

printf("\nEnter Max Matrix: ");

acceptdata(M);

printf("\nEnter Availability:");

acceptav();

calcneed();

displaydata();

banker();

}

QUE2 (NON-PREEMPTIVE PRIORITY SCHEDULING)

#include<stdio.h>

struct input

{

char pname[10];

int bt,at,tbt,ft,p;

}tab[10];

struct gantt

{

char pname[10];

int start,end;

}g[30],g1[30];

int n,i,time,prev,k;

void getinput()

{

printf("\nEnter No of Process: ");

scanf("%d",&n);

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

{

printf("\nEnter Process Name: ");

scanf("%s",tab[i].pname);

printf("Arrival Time:");

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

printf("Burst Time: ");

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

tab[i].tbt=tab[i].bt;

printf("\nEnter the Priority:");

scanf("%d",&tab[i].p);

}

}

void printinput()

{

// int TWT=0,TTAT=0;

printf("\nPname\tAT\tBT");

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

printf("\n%s\t%d\t%d",tab[i].pname,tab[i].at,tab[i].bt);

}

void printoutput()

{

int TWT=0,TTAT=0;

float ATAT,AWT;

printf("\nPname\tAT\tBT\tFT\tWT\tTAT");

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

{

printf("\n%s\t%d\t%d\t%d\t%d\t%d",tab[i].pname,tab[i].at,tab[i].bt,tab[i].ft,tab[i].ft-tab[i].at-tab[i].bt,tab[i].ft-tab[i].at);

TWT=TWT+(tab[i].ft-tab[i].at-tab[i].bt);

TTAT=TTAT+(tab[i].ft-tab[i].at);

}

printf("\nTotal WT: %d",TWT);

printf("\nTotal TAT:%d",TTAT);

AWT=(float)TWT/n;

ATAT=(float)TTAT/n;

printf("\nAverage WT: %f",AWT);

printf("\nAverage TAT:%f",ATAT);

}

void sort()

{

int pass;

struct input temp;

for(pass=1;pass<n;pass++)

{

for(i=0;i<n-pass;i++)

{

if(tab[i].at>tab[i+1].at)

{

temp=tab[i];

tab[i]=tab[i+1];

tab[i+1]=temp;

}

}

}

}

int arrived(int time)

{

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

{

if(tab[i].at<=time && tab[i].tbt!=0)

return 1;

}

return 0;

}

int gethighpriority(int time)

{

int processpos,min=99;

for(i=0;i<n;i++)//i=0,1

{ // p1,p3 min=2

if(tab[i].at<=time && tab[i].tbt!=0 && tab[i].p<min)

{

min=tab[i].p;

processpos=i;

}

}

return processpos;

}

void processinput()

{

int finish=0,j;

k=0;

while(finish!=n)

{

if(arrived(time))

{

i=gethighpriority(time);

for(j=0;j<tab[i].bt;j++)

{

time++;

tab[i].tbt--;

g[k].start=prev;

g[k].end=time;

prev=time;

strcpy(g[k++].pname,tab[i].pname);

tab[i].ft=time;

if(tab[i].tbt==0)

{

finish++;

break;

}

}

}

else

{

time++;

g[k].start=prev;

g[k].end=time;

strcpy(g[k++].pname,"idle");

prev=time;

}

// i++;

}

}

void ganttchart()

{

int i,j=0;

printf("\n\*\*\*\*\*\*Each Unit Gantt chart\*\*\*\*\*\*");

printf("\nStart\tpname\tEnd");

for(i=0;i<k;i++)

{

printf("\n%d\t%s\t%d",g[i].start,g[i].pname,g[i].end);

}

printf("\n\*\*\*\*\*\*\*\*Final Gantt Chart\*\*\*\*\*\*\*");

g1[0]=g[0];

for(i=1;i<k;i++)

{

if(strcmp(g[i].pname,g1[j].pname)==0)

g1[j].end=g[i].end;

else

{

j++;

g1[j]=g[i];

}

}

printf("\nStart\tpname\tEnd");

for(i=0;i<=j;i++)

{

printf("\n%d\t%s\t%d",g1[i].start,g1[i].pname,g1[i].end);

}

}

int main()

{

getinput();

printinput();

sort();

printf("\nData After Sorting: ");

printinput();

processinput();

printoutput();

ganttchart();

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

{

tab[i].tbt=tab[i].bt=rand()%10+1;

tab[i].at=tab[i].ft+2;

}

printinput();

processinput();

printoutput();

ganttchart();

}

SLIP 20

QUE1

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/wait.h>

int main() {

pid\_t pid;

// Fork a child process

pid = fork();

if (pid < 0) {

// Error occurred

fprintf(stderr, "Fork failed\n");

return 1;

} else if (pid == 0) {

// Child process

// Use execl() to execute the "ls" command

execl("/bin/ls", "ls", (char \*)NULL);

// If execl fails

perror("execl");

exit(1);

} else {

// Parent process

// Parent goes to sleep state

sleep(5);

// Parent process may continue its own execution here

printf("Parent process waking up after sleep.\n");

// Wait for the child process to finish

wait(NULL);

printf("Child process has terminated.\n");

}

return 0;

}

QUE2 (LRU)

#include<stdio.h>

int RefString[20],PT[10],nof,nor;

void Accept()

{

int i;

printf("Enter Reference String: \n");

for(i=0;i<nor;i++)

{

printf("[%d]=",i);

scanf("%d",&RefString[i]);

}

}

int Search(int s)

{

int i;

for(i=0;i<nof; i++)

if(PT[i]==s)

return(i);

return(-1);

}

int GetLRU(int e)

{

int i,j,Pos=99,Posi,k;

for(i=0;i<nof;i++)

{

for(j=e-1;j>=0;j--)

{

if(PT[i]==RefString[j])

{

if(j<Pos)

{

Pos=j;

Posi=i;

}

break;

}

}

}

return(Posi);

}

void LRU()

{

int i,j,k,Faults=0;

for(k=0,i=0; k<nof && i<nor; i++)

{

printf("\n\n%2d ",RefString[i]);

if(Search(RefString[i])==-1)

{

PT[k]=RefString[i];

for(j=0;j<nof;j++)

{

if(PT[j])

printf(" %2d",PT[j]);

}

Faults++;

k++;

}

}

while(i<nor)

{

printf("\n\n%2d ",RefString[i]);

if(Search(RefString[i])==-1)

{

k = GetLRU(i);

PT[k]=RefString[i];

for(j=0;j<nof;j++)

{

printf(" %2d",PT[j]);

}

Faults++;

}

i++;

}

printf("\nTotal Page Faults: %d",Faults);

}

void main()

{

printf("Enter Length of reference string: ");

scanf("%d",&nor);

printf("Enter No.of Frames: ");

scanf("%d",&nof);

Accept();

LRU();

}

QUE2 (FCFS)