**FCFS(First come first serve)**

//Program For “First-Come-First-Serve "CPU Scheduling Algorithm

#include<stdio.h>

#include<string.h>

int main(void)

{

//VARIABLE DECLARATION

char pn[20][20], c[20][20]; //PN-PROGRAM NAMES

int n,i,j,at[20], bt[20], wt[20],tat[20], ct[20]; //bt-Burst Time ; wt-Waiting Time;

// tat-Turn Around Time

int twt=0, ttat=0, temp1, temp2;

printf("Enter number of processes:");

scanf("%d", &n);

//taking input values i.e., process-names, arrival-times and burst-times

printf("Enter <Process-name> <Arrival-time> <Burst-time> for processes:\n", (i+1));

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

scanf("%s%d%d",&pn[i],&at[i],&bt[i]);

//Sort The Processes According To Arrival Times

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

{

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

{

if(at[i]>at[j])

{

temp1 = bt[i];

temp2 = at[i];

bt[i] = bt[j];

at[i] = at[j];

bt[j] = temp1;

at[j] = temp2;

strcpy(c[i],pn[i]);

strcpy(pn[i],pn[j]);

strcpy(pn[j],c[i]);

}

}

if(i==0) ct[0]=at[0]+bt[0];

if(i>0)

{

if(at[i]>ct[i-1])

ct[i]=at[i]+bt[i];

else

ct[i]=ct[i-1]+bt[i];

}

}

//Calculating Waiting Time & Tat

wt[0]=0;

tat[0]=ct[0]-at[0];

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

{

tat[i] = ct[i]-at[i];

wt[i] = tat[i]-bt[i];

twt += wt[i];

ttat += tat[i];

}

//Printing The Values After All Preocesses Completed

printf("S.N.\tPN\tAT\tBT\tCT\tWT\tTAT\n");

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

printf("%d\t%s\t%d\t%d\t%d\t%d\t%d\n",(i+1),pn[i],at[i],bt[i],ct[i],wt[i],tat[i]);

printf("Total waiting time:%d\n", twt);

printf("Total Turn Around Time:%d", ttat);

}

**OPTIMAL**

include <stdio.h>

#include <conio.h>

int main()

{

 int n, rss, fa[20], rsa[50], ta[20]; //n-No\_of\_Frames

 //rss->Reference\_String\_Size::fa->Frame\_Array

 //rsa->Reference\_String\_Array::ta->Temporary\_Array

 int i,j,k, d,pfc=0,npf, cp,ff=0;

 //d-distance[How soon a page will be used again?]

 //cp->Current\_Position :: ff->Frames\_Filled ::pfc->Page\_Fault\_Count

 //npf:NO\_Page\_Faults [0-False, 1-True]

 printf("Enter number of frames: ");

 scanf("%d", &n);

 printf("Enter number of pages in reference string: ");

 scanf("%d", &rss);

 printf("Enter Reference string:\n");

 for(i=0; i<rss; i++)

 scanf("%d",&rsa[i]);

 for(i=0;i<n;i++)

 {

 fa[i]=-1;

 ta[i]=999;

 }

printf("\nCURRENT\_PAGE\t\tPAGES\_IN\_FRAME\t\tPAGE\_FAULT\_OCCURED?\n"

);

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

 {

 printf("\n\t%d\t\t",rsa[i]);

 npf=0;

 for(j=0;j<n;j++) //Checking for the page in FRAME ARRAY

 {

 if(fa[j]==rsa[i])

 {

 npf = 1;

 printf("\t\t\t\tNO");

 break;

 }

 }

 if(npf==0) // if page fault occurs

 {

 pfc++;

 if(ff<n)

 {

 fa[ff]=rsa[i];

 ff++;

 }

 else

 {

 for(k=0;k<n;k++)

 ta[k]=999;

 for(k=0; k<n; k++) //finding how near a page is

 {

 d = 0; // d-> distance

 for(j=i+1;j<rss;j++)

 {

 if(fa[k]==rsa[j])

 {

 ta[k]=d;

 break;

 }

 else

 d++;

 }

 }

 cp=0;

 for(j=1;j<n;j++)

 {

 if(ta[cp]<ta[j])

 cp=j; //cp->current position

 }

 fa[cp] = rsa[i];

 }

 for(j=0;j<n;j++)

 printf("%d\t",fa[j]);

 printf("\tYES");

 }

 }

 printf("\nTotal no of pagefaults: %d",pfc);

 return 0;

}

**SEQUENTIAL**

#include<stdio.h>

#include<conio.h>

struct fileTable

{

char name[20];

int sb, nob;

}ft[30];

void main()

{

int i, j, n;

char s[20];

clrscr();

printf("Enter no of files :");

scanf("%d",&n);

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

{

printf("\nEnter file name %d :",i+1);

scanf("%s",ft[i].name);

printf("Enter starting block of file %d :",i+1);

scanf("%d",&ft[i].sb);

printf("Enter no of blocks in file %d :",i+1);

scanf("%d",&ft[i].nob);

}

printf("\nEnter the file name to be searched-- ");

scanf("%s",s);

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

if(strcmp(s, ft[i].name)==0)

break;

if(i==n)

printf("\nFile Not Found");

else

{

printf("\nFILE NAME START BLOCK NO OF BLOCKS BLOCKS OCCUPIED\n");

printf("\n%s\t\t%d\t\t%d\t",ft[i].name,ft[i].sb,ft[i].nob);

for(j=0;j<ft[i].nob;j++)

printf("%d, ",ft[i].sb+j);

}

getch();

}

**DEADLOCK AVOIDANCE USING BANKERS ALGORITHM**

#include <stdio.h>

int main()

{

// P0, P1, P2, P3, P4 are the Process names here

int n, m, i, j, k;

n = 5; // Number of processes

m = 3; // Number of resources

int alloc[5][3] = { { 0, 1, 0 }, // P0 // Allocation Matrix

{ 2, 0, 0 }, // P1

{ 3, 0, 2 }, // P2

{ 2, 1, 1 }, // P3

{ 0, 0, 2 } }; // P4

int max[5][3] = { { 7, 5, 3 }, // P0 // MAX Matrix

{ 3, 2, 2 }, // P1

{ 9, 0, 2 }, // P2

{ 2, 2, 2 }, // P3

{ 4, 3, 3 } }; // P4

int avail[3] = { 3, 3, 2 }; // Available Resources

int f[n], ans[n], ind = 0;

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

f[k] = 0;

}

int need[n][m];

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

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

need[i][j] = max[i][j] - alloc[i][j];

}

int y = 0;

for (k = 0; k < 5; k++) {

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

if (f[i] == 0) {

int flag = 0;

for (j = 0; j < m; j++) {

if (need[i][j] > avail[j]){

flag = 1;

break;

}

}

if (flag == 0) {

ans[ind++] = i;

for (y = 0; y < m; y++)

avail[y] += alloc[i][y];

f[i] = 1;

}

}

}

}

int flag = 1;

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

{

if(f[i]==0)

{

flag=0;

printf("The following system is not safe");

break;

}

}

if(flag==1)

{

printf("Following is the SAFE Sequence\n");

for (i = 0; i < n - 1; i++)

printf(" P%d ->", ans[i]);

printf(" P%d", ans[n - 1]);

}

return (0);

}