FIRST COME FIRST SERVE SCHEDULING

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

struct process{
        int pid;
        int bt;
        int wt,tt;
  }p[10];

int main()
{
        int i,n,totwt,tottt,avg1,avg2;
        printf("enter the no of process\n");
        scanf("%d",&n);
for(i=1;i<=n;i++)
{
        p[i].pid=i;
        printf("enter the burst time ");
        scanf("%d",&p[i].bt);
}
        p[1].wt=0;
        p[1].tt=p[1].bt+p[1].wt;
        i=2;
```

```
while(i<=n)
{
        p[i].wt=p[i-1].bt+p[i-1].wt;
        p[i].tt=p[i].bt+p[i].wt;
        i ++;
}
        i=1;
        totwt=tottt=0;
        printf("\n processid\t bt\t wt\t tt\n");

while(i<=n)
{
        printf("\n\t%d\t%d\t%d\t%d",p[i].pid,p[i].bt,p[i].wt,p[i].tt);
        totwt=p[i].wt+totwt;
        tottt=p[i].tt+tottt;
        i++;
}
        avg1=totwt/n;
        avg2=tottt/n;
        printf("\navg1=%d\t avg2=%d\t",avg1,avg2);
        return 0;
}
```

SHORTEST JOB FIRST

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
struct process
{
        int pid,bt,wt,tt;
}p[10],temp;
int main()
{
        int i,j,n,totwt,tottt;
        float avg1,avg2;
        printf("\nEnter the number of process:\t");
        scanf("%d",&n);
for(i=1;i<=n;i++) {
        p[i].pid=i;
        printf("\nEnter the burst time:\t");
        scanf("%d",&p[i].bt);
}
for(i=1;i<n;i++) {
        for(j=i+1;j<=n;j++) {
if(p[i].bt>p[j].bt) {
        temp.pid=p[i].pid;
        p[i].pid=p[j].pid;
        p[j].pid=temp.pid;
        temp.bt=p[i].bt;p[i].bt=p[j].bt;
```

```
        p[j].bt=temp.bt;
}}}
        p[1].wt=0;
        p[1].tt=p[1].bt+p[1].wt;
        i=2;
while(i<=n){
        p[i].wt=p[i-1].bt+p[i-1].wt;
        p[i].tt=p[i].bt+p[i].wt;
        i++;
}
i=1;
        totwt=tottt=0;
        printf("\nProcess id \tbt \twt \ttt");

while(i<=n){
        printf("\n\t%d \t%d \t%d \t%d\n",p[i].pid,p[i].bt,p[i].wt,p[i].tt);
        totwt=p[i].wt+totwt;
        tottt=p[i].tt+tottt;
        i++;
}
        avg1=(float)totwt/n;
        avg2=(float)tottt/n;
        printf("\nAVG1=%f\t AVG2=%f",avg1,avg2);
        return 0;
 }
```

PRIORITY SCHEDULING

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
struct process
{
        int pid;
        int bt;
        int wt;
        int tt;
        int prior;
}
        p[10],temp;
int main()
{
        int i,j,n,totwt,tottt;
        float arg1,arg2;
        printf("enter the number of process");
        scanf("%d",&n);
for(i=1;i<=n;i++) {
        p[i].pid=i;
        printf("enter the burst time");
        scanf("%d",&p[i].bt);
        printf("\n enter the priority");
        scanf("%d",&p[i].prior);
}
```

```
for(i=1;i<n;i++) {
for(j=i+1;j<=n;j++) {
if(p[i].prior>p[j].prior)
{
        temp.pid=p[i].pid;
        p[i].pid=p[j].pid;
        p[j].pid=temp.pid;
        temp.bt=p[i].bt;
        p[i].bt=p[j].bt;
        p[j].bt=temp.bt;
        temp.prior=p[i].prior;
        p[i].prior=p[j].prior;
        p[j].prior=temp.prior;
}}}
        p[i].wt=0;
        p[1].tt=p[1].bt+p[1].wt;
        i=2;
while(i<=n)
{
        p[i].wt=p[i-1].bt+p[i-1].wt;
        p[i].tt=p[i].bt+p[i].wt;
        i++;
}
        i=1;
```

```
        totwt=tottt=0;
        printf("\n process to \t bt \t wt \t tt");
while(i<=n)
{
        printf("\n%d\t\t  %d\t  %d\t  %d\t",p[i].pid,p[i].bt,p[i].wt,p[i].tt);
        totwt=p[i].wt+totwt;
        tottt=p[i].tt+tottt;
        i++;
}
        arg1=(float)totwt/n;
        arg2=(float)tottt/n;
        printf("\n arg1=%f \t arg2=%f\t",arg1,arg2);
        return 0;
}
```

PRODUCER-CONSUMER PROBLEM

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
void main()
{
        int buffer[10], bufsize, in, out, produce, consume, choice=0;
        in = 0;
        out = 0;
        bufsize = 10;
while(choice !=3) {
        printf("\n1. Produce\t 2. Consume \t3. Exit");
        printf("\nEnter your choice:=");
        scanf("%d", &choice);
switch(choice)
{
case 1:
        if((in+1)%bufsize==out)
                printf("\nBuffer is Full");
else
{
        printf("\nEnter the value: ");
        scanf("%d", &produce);
        buffer[in] = produce;
        in = (in+1)%bufsize;
```

```
} break;
case 2:
        if(in == out)
                printf("\nBuffer is Empty");
else
{
        consume = buffer[out];
        printf("\nThe consumed value is %d", consume);
        out = (out+1)%bufsize;
}
break;
}
        }
                }
```

PAGE REPLACEMENT FIFO

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
int i,j,nof,nor,flag=0,ref[50],frm[50],pf=0,victim=-1;
void main()
{
        printf("\n \t\t\t FIFO PAGE REPLACEMENT ALGORITHM");
        printf("\n Enter no.of frames....");
        scanf("%d",&nof);
        printf("Enter number of Pages.\n");
        scanf("%d",&nor);
for(i=0;i<nor;i++) {
        printf("\n Enter the Page No...");
        scanf("%d",&ref[i]);
}
        printf("\nThe given Pages are:");
for(i=0;i<nor;i++) {
        printf("%4d",ref[i]);
}
for(i=1;i<=nof;i++) {
        frm[i]=-1;
        printf("\n");
}
```

```
for(i=0;i<nor;i++) {
        flag=0;
        printf("\n\t page no %d->\t",ref[i]);
for(j=0;j<nof;j++) {
        if(frm[j]==ref[i])
{
        flag=1;
        break;
}}
if(flag==0) {
        pf++;
        victim++;
        victim=victim%nof;
        frm[victim]=ref[i];
for(j=0;j<nof;j++){
printf("%4d",frm[j]);
}
        }
                }
printf("\n\n\t\t No.of pages faults...%d",pf);

}
```

PAGE REPLACEMENT LRU

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
        int i,j,nof,nor,flag=0,ref[50],frm[50],pf=0,victim=-1;
        int recent[10],lrucal[50],count=0;
        int lruvictim();
void main()
{
        printf("\n\t\t\t LRU PAGE REPLACEMENT ALGORITHM");
        printf("\n Enter no.of Frames....");
        scanf("%d",&nof);
        printf(" Enter no.of reference string..");
        scanf("%d",&nor);
        printf("\n Enter reference string..");
for(i=0;i<nor;i++) {
        scanf("%d",&ref[i]);
}
        printf("\n\n\t\t LRU PAGE REPLACEMENT ALGORITHM ");
        printf("\n\t The given reference string:");
        printf("\n.....");
for(i=0;i<nor;i++) {
        printf("%4d",ref[i]);
}
for(i=1;i<=nof;i++) {
```

```
        frm[i]=-1;
        lrucal[i]=0;
}
for(i=0;i<10;i++) {
        recent[i]=0;
        printf("\n");
}
for(i=0;i<nor;i++) {
        flag=0;
        printf("\n\t Reference NO %d->\t",ref[i]);
for(j=0;j<nof;j++) {
if(frm[j]==ref[i]) {
flag=1;
break;
        }
}
if(flag==0) {
        count++;
        if(count<=nof)
        victim++;
else
        victim=lruvictim();
        pf++;
        frm[victim]=ref[i];
for(j=0;j<nof;j++) {
printf("%4d",frm[j]);
```

```
        }
}
recent[ref[i]]=i;
}
        printf("\n\n\t No.of page faults...%d",pf);
        getch();
                }
int lruvictim() {
        int i,j,temp1,temp2;
for(i=0;i<nof;i++) {
        temp1=frm[i];
        lrucal[i]=recent[temp1];
}
        temp2=lrucal[0];
for(j=1;j<nof;j++) {
        if(temp2>lrucal[j])
        temp2=lrucal[j];
}
for(i=0;i<nof;i++) {
if(ref[temp2]==frm[i])
return i;
return 0;
        }
}
```

ROUND ROBIN SCHEDULING

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
struct process
{
        int pid,bt,tt,wt;
};
int main()
{
        struct process x[10],p[30];
        int i,j,k,tot=0,m,n;
        float wttime=0.0,tottime=0.0,a1,a2;
        printf("\nEnter the number of process:\t");
        scanf("%d",&n);
for(i=1;i<=n;i++){
        x[i].pid=i;
        printf("\nEnter the Burst Time:\t");
        scanf("%d",&x[i].bt);
        tot=tot+x[i].bt;
}
        printf("\nTotal Burst Time:\t%d",tot);
        p[0].tt=0;
        k=1;
        printf("\nEnter the Time Slice:\t");
```

```
        scanf("%d",&m);
for(j=1;j<=tot;j++) {
for(i=1;i<=n;i++) {
        if(x[i].bt !=0) {
                p[k].pid=i;
        if(x[i].bt-m<0) {
                p[k].wt=p[k-1].tt;
                p[k].bt=x[i].bt;
                p[k].tt=p[k].wt+x[i].bt;
                x[i].bt=0;
                k++;
}
        else
{
                p[k].wt=p[k-1].tt;
                p[k].tt=p[k].wt+m;
                x[i].bt=x[i].bt-m;
                k++;
}}}}
        printf("\nProcess id \twt \ttt");
for(i=1;i<k;i++){
        printf("\n\t%d \t%d \t%d",p[i].pid,p[i].wt,p[i].tt);
        wttime=wttime+p[i].wt;
        tottime=tottime+p[i].tt;
        a1=wttime/n;
        a2=tottime/n;
```

```
}


printf("\n\nAverage Waiting Time:\t%f",a1);
printf("\n\nAverage TurnAround Time:\t%f",a2);
return 0;
}
```