```
#include<omp.h>
#include<stdio.h>
#define n 15
int main(int argc,char *argv[])
{
int numthreads,i,t;
int a[100],b[100],c[100]={0},d[100]={0};
for(i=0;i<n;i++){
a[i]=i*3;
b[i]=i*2;
}
#pragma omp parallel shared(a,b,c,d,numthreads) private(t,i)
{
t=omp_get_thread_num();
if(t==0){
    numthreads=omp_get_num_threads();
    printf("%d is the number of threads\n",numthreads);
}
printf("thread %d is running..\n",t);
#pragma omp sections nowait
{
    #pragma omp section
    {
    printf("thread %d is running section1\n",t);
    for(i=0;i<n;i++){
    c[i]=a[i]-b[i];
    }
    for(i=0;i<n;i++)
    {
    printf("%d\t",c[i]);
    }
    }
    #pragma omp section
    {
    printf("thread %d is running section2\n",t);
    for(i=0;i<n;i++){
    d[i]=a[i]+b[i];
    }
    for(i=0;i<n;i++)
    {
    printf("%d\t",d[i]);
    }
    }
}
printf("thread %d stopped running\n",t);
}
}
===========================
fcfs without arrival

#include<iostream>
using namespace std;
int main()
{
    int n;
    cout<<"Enter Number of Processes\n";
    cin>>n;
    int burst_time[n];
    int wait_time[n];
    int turn_around_time[n];
    double sum_waiting_time = 0;
    double sum_turn_around_time = 0;
    cout<<"Enter Burst Time of Processes\n";
    for(int i=0;i<n;i++)
        cin>>burst_time[i];

    // Calculating Waiting Time
    wait_time[0] = 0;
    for(int i=1;i<n;i++)
        wait_time[i] = burst_time[i-1] + wait_time[i-1];

    //Calculating TAT
    for(int i=0;i<n;i++)
        turn_around_time[i] = burst_time[i] + wait_time[i];

    //Gantt Chart
    cout<<"\nGantt Chart :";
        for(int i=0;i<n;i++)
               cout<<"P["<<i+1<<"]\t";
    cout<<"\n\t\t";
     for(int i=0;i<n;i++)
        cout<<wait_time[i]<<"\t";
    cout<<turn_around_time[n-1]<<endl;

 // Computing Avg Wating time and TAT
    for(int i=0;i<n;i++)
    {
        sum_waiting_time += wait_time[i];
        sum_turn_around_time += turn_around_time[i];
    }
    cout<<"Average Waiting Time = "<<sum_waiting_time/n<<endl;
    cout<<"Average Turn Around Time = "<<sum_turn_around_time/n<<endl;

}

==================================================================================
FCFS ARRIVAL

#include<stdio.h>
int main(){
int n,i,j,bt[10],wt[10],tat[10],at[10],gt[10],pos,p[10],temp;
float avgwt=0.0,avgtat=0.0;

printf("Enter the number of processes :");
scanf("%d",&n);
printf("\nEnter the burst times :");
for(i=0;i<n;i++){
    printf("\nP[%d] :",i+1);
    scanf("%d",&bt[i]);
}
printf("\nEnter the arrival times : ");
for(i=0;i<n;i++){
    printf("\nP[%d] :",i+1);
    scanf("%d",&at[i]);
    p[i]=i;
}

for(i=0;i<n;i++)
    {
        pos=i;
        for(j=i+1;j<n;j++)
        {
            if(at[j]<at[pos])
                pos=j;
        }

        temp=at[i];
        at[i]=bt[pos];
        at[pos]=temp;

        temp=bt[i];
        bt[i]=bt[pos];
        bt[pos]=temp;

        temp=p[i];
        p[i]=p[pos];
        p[pos]=temp;
    }


gt[0]=0;
for(i=0;i<n;i++){
    gt[i+1]=gt[i]+bt[i];
}
```

```
printf("\n\nProcess\tWait time\tTat\tBurst\n");
for(i=0;i<n;i++){
    tat[i]=gt[i+1]-at[i];
    wt[i]=gt[i]-at[i];
    avgwt+=wt[i];
    avgtat+=tat[i];
    printf("\nP[%d]\t%d\t%d\t%d",p[i],wt[i],tat[i],bt[i]);
}
avgwt/=i;
avgtat/=i;
printf("\nAvg wait time = %f",avgwt);
printf("\nAvg tat = %f ",avgtat);
return 0;
}
====================
ROUNDROBIN

#include<iostream>

using namespace std;

struct process
{
    int arrival,burst;
    int wait,turnAroundTime,sTime,ID;
};

int main()  {
    cout<<"Enter Number of processes: ";
    int n,i,counts,timeQ,temp,totalServiceTime=0;
    cin>>n;
    process p[n];
    cout<<"Enter the Burst Time:-\n";
    for(i = 0 ; i < n ; i++)    {
        p[i].ID=i;
        cout<<"\nProcess:"<<i;
        cout<<"\nBurst Time:";
        cin>>p[i].burst;
        p[i].sTime=p[i].burst;
        p[i].wait = 0;
    }
    cout<<"\n Enter the time quantum : ";
    cin>>timeQ;
    while(true) {
        for(i = 0 ,counts = 0 ; i < n; i++)    {
            temp = timeQ;
            if( p[i].sTime == 0 )   {
                counts++;
                continue;
            }
            if(p[i].sTime >= timeQ) {
                p[i].sTime -= timeQ;
            }
            else    {
                temp = p[i].sTime;
                p[i].sTime = 0;
            }
            totalServiceTime += temp;
            p[i].turnAroundTime = totalServiceTime;
        }
        if(counts == n)
            break;
    }
    for(i = 0; i < n; i++)  {
        p[i].wait = p[i].turnAroundTime - p[i].burst;
    }
    float avgtime=0.0,avgwait=0.0;
    for(i=0;i<n;i++)
    {
        avgtime+=p[i].turnAroundTime;
        avgwait+=p[i].wait;
    }
    avgtime/=n;
    avgwait/=n;
    cout<<"\n\nProc\t|Burst\t|Wait\t|tTime\t|sTime\n";
    for(i=0;i<n;i++)
    {
        cout<<p[i].ID<<"\t|"<<p[i].burst<<"\t|"<<p[i].wait<<"\t|"<<p[i].turnAroundTime<<"\t|"<<p
[i].sTime;
        cout<<endl;
    }
    cout<<"\n\nAvg Turnaround Time: "<<avgtime;
    cout<<"\nAvg Wait Time: "<<avgwait;
    cout<<"\nTotal Time: "<<totalServiceTime;
    return 0;
}
==============================================================================================================
contiguous memory

#include <iostream>
#include <cstdlib>
using namespace std;
int main(){
    int i,j,mat[5][6],blocks,filesize,q,n[50][50],temp = 0, arr[150], p=0, k=0;
    cout<<"\nEnter file size :";
    cin>>filesize;
    if(filesize%512==0)
            blocks=filesize/512;
        else if(filesize<=0)
        {
            cout<<"Error in size of file\n";
            return -1;
    }
        else
            blocks=filesize/512+1;
    cout<<"\nNo of blocks needed = "<<blocks;
    for(i=0;i<10;i++)
        for(j=0;j<10;j++)
                mat[i][j]=rand()%2;
    cout<<"\nMemory initially is \n";
    for(i=0;i<10;i++){
        for(j=0;j<10;j++)
                cout<<mat[i][j]<<" ";
        cout<<endl;
    }
    for(i=0;i<10;i++)
        for(j=0;j<10;j++,p++)
                arr[p]=mat[i][j];
    for(i=0;i<p;i++){
        if(arr[i]==1){
                k++;
                if(k==blocks)
                    break;
            }
        else
            k=0;
    }
    cout<<"Start addr :"<<512*(i-blocks+1);
    q=i-blocks+1;
    cout<<"\n"<<q<<"\t"<<q+blocks;
    for(j=q;j<q+blocks;j++)
            arr[j]=0;
    for(i=0;i<10;i++)
            for(j=0;j<10;j++,temp++)
                n[i][j]=arr[temp];
    cout<<"\n\nNew matrix\n";
    for(i=0;i<10;i++){
            for(j=0;j<10;j++)
                cout<<n[i][j]<<" ";
            cout<<endl;
    }
    return 0;
}
==================================================================
```

linkedallocation.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
struct filestruct{
        int blockNum;
        struct filestruct* nextBlock;
};
struct filestruct* head= NULL;
int n=0;
void addBlock(int n){
        struct filestruct* temp =(struct filestruct*)malloc(sizeof(struct filestruct));
        temp->blockNum= n;
        temp -> nextBlock= NULL;
        if(head= NULL){
            head= temp;
            return;
        }
        else{
            temp -> nextBlock= head;
            head= temp;
        }
}
void printBlock(){
        struct filestruct* temp = head;
        while(temp!=NULL){
            printf("\nBlock number is : %d",temp->blockNum);
            temp=temp->nextBlock;
        }
}
int main(){
        char fname[30];
        int size, num, memory[10][10],i,j,i1,j1,k,count=1,index;
        float numBlocks;
        printf("\nEnter the filename: ");
        scanf("%s",fname);
        //srand(time(NULL));
        printf("\nEnter the size of the file is kB: ");
        scanf("%d",&size);
        if(size%512==0)
            numBlocks=size/512;
        else if(size<=0)
        {
            printf("\nError in size of file\n");
            return -1;
    }
        else
            numBlocks=size/512+1;
        if((numBlocks-(int)numBlocks)==0.0)
            num= (int)numBlocks;
        else
            num=(int)numBlocks+1;
    printf("%d blocks are allocated\n",num);
    for(i=0;i<10;i++)
            for(j=0;j<10;j++)
                memory[i][j]= rand()%2;
        printf("\nInitial memory map: \n");
        for(i=0;i<10;i++){
            for(j=0;j<10;j++)
                    printf("%d\t",memory[i][j]);
            printf("\n");
        }
        for(i=0;i<10;i++){
            for(j=0;j<10;j++){
                    if(memory[i][j]==1){
                        index=i*5+j;
                        addBlock(index);
                        memory[i][j]=0;
                        count++;
                    }
                    if(count>num) break;
            }
            if(count>num) break;
        }
        printf("\nFinal memory map: \n");
        for(i=0;i<10;i++){
            for(j=0;j<10;j++)
                    printf("%d\t",memory[i][j]);
            printf("\n");
        }
}
================================================================
fit.cpp

#include<stdio.h>
#include<stdlib.h>
int part[100],avail[100],waste[3],sortpart[100];
int npa, npr;
int proc[100];
int main()
{
 int i,j,temp;
 printf("Enter the number partitions and processes\n");
 scanf("%d %d",&npa,&npr);
 if(npr>npa)
 {
printf("Number of partitions less than processes\n");
 exit(0);
 }
 printf("Enter the partition sizes\n");
for(i=0;i<npa;i++)
 scanf("%d",&part[i]);
 printf("Enter the process sizes\n");
 for(i=0;i<npr;i++)
 scanf("%d",&proc[i]);
 printf("FIRST FIT\n");
 for(i=0;i<npr;i++)
 {
 for(j=0;j<npa;j++)
 {
 if(avail[j]==0&&part[j]>=proc[i])
 {
 printf("%d<-%d\n",part[j],proc[i]);
 avail[j]=1;
 waste[0]+=(part[j]-proc[i]);
 break;
 }
 }
 if(j==npa)
 printf("%d does not fit\n",proc[i]);
 }
 printf("Wastage= %d\n",waste[0]);
 printf("\n");
 printf("BEST FIT\n");
 for(i=0;i<npa;i++)
 {
 avail[i]=0;
 sortpart[i]=part[i];
 }
 for(i=0;i<npa;i++)
 for(j=0;j<npa-1;j++)
 if(sortpart[j]>sortpart[j+1])
 {
 temp=sortpart[j];
 sortpart[j]=sortpart[j+1];
sortpart[j+1]=temp;
 }
 for(i=0;i<npr;i++)
 {
for(j=0;j<npa;j++)
 {
if(avail[j]==0&&sortpart[j]>=proc[i])
 {
```

```
 printf("%d<-%d\n",sortpart[j],proc[i]);
 avail[j]=1;
 waste[1]+=(sortpart[j]-proc[i]);
 break;
 }
 }
 if(j==npa)
 printf("%d does not fit\n",proc[i]);
 }
 printf("Wastage= %d\n",waste[1]);
 printf("\n");
 printf("WORST FIT\n");
 for(i=0;i<npa;i++)
 avail[i]=0;
for(i=0;i<npa;i++)
 for(j=0;j<npa-1;j++)
 if(sortpart[j]<sortpart[j+1])
 {
 temp=sortpart[j];
 sortpart[j]=sortpart[j+1];
 sortpart[j+1]=temp;
 }
 for(i=0;i<npr;i++)
{
 for(j=0;j<npa;j++)
 {
if(avail[j]==0&&sortpart[j]>=proc[i])
 {
 printf("%d<-%d\n",sortpart[j],proc[i]);
 avail[j]=1;
 waste[2]+=(sortpart[j]-proc[i]);
 break;
 }
 }
 if(j==npa)
 printf("%d does not fit\n",proc[i]);
 }
printf("Wastage=%d\n",waste[2]);
}
=========================================
deadlock

#include<iostream>
using namespace std;
int main(){
    int i,j,p,r,alloc[20][20],req[20][20],avail[20],success[20],flag=0,c1=0,c2=0,track=0;
    cout<<"Processes :";
    cin>>p;
    cout<<"\nResources :";
    cin>>r;
    cout<<"\nEnter the available vector :";
    for(i=0;i<r;i++){
        cin>>avail[i];
    }
    for(i=0;i<p;i++){
        success[i]=0;
    }
    cout<<"\nEnter the allocated matrix\n";
    for(i=0;i<p;i++){
        for(j=0;j<r;j++){
            cin>>alloc[i][j];
        }
    }
    cout<<"\nEnter the request matrix\n";
     for(i=0;i<p;i++){
        for(j=0;j<r;j++){
            cin>>req[i][j];
        }
    }
int process[100];
int pc=0;
    while(c1!=p){
        c2=c1;
        for(i=0;i<p;i++){
            for(j=0;j<r;j++){
                if(req[i][j]<=avail[j]){
                    track++;
                }
            }
            if(track==r && success[i]==0){
                success[i]=1;
        process[pc]=i;
        pc++;
                for(j=0;j<r;j++)
                    avail[j]=avail[j]+alloc[i][j];
                c1++;
            }
            track=0;
        }
        if(c1==c2){
            cout<<"\nDeadlock present!";
            flag=1;
            break;
        }
    }
    if(flag){
    cout<<"\n the following did not execute\n";
        for(j=0;j<p;j++){
            if(success[j]!=1){
                cout<<" P"<<j;
            }

        }

    }
    else{
        cout<<"No deadlock!";
    cout<<"\n safe sequence is \n";
    for(j=0;j<p;j++){

                cout<<" P"<<process[j];
            }
}


return 0;
}
==========================================================
banker.cpp

#include<iostream>
using namespace std;
int main(){
    int i,j,p,r,alloc[20][20],req[20][20],avail[20],success[20],flag=0,c1=0,c2=0,track=0;
    cout<<"Processes :";
    cin>>p;
    cout<<"\nResources :";
    cin>>r;
    cout<<"\nEnter the available vector :";
    for(i=0;i<r;i++){
        cin>>avail[i];
    }
    for(i=0;i<p;i++){
        success[i]=0;
    }
    cout<<"\nEnter the allocated matrix\n";
    for(i=0;i<p;i++){
        for(j=0;j<r;j++){
            cin>>alloc[i][j];
        }
    }
    cout<<"\nEnter the max matrix\n";
     for(i=0;i<p;i++){
        for(j=0;j<r;j++){
            cin>>req[i][j];
        }
    }
for(i=0;i<p;i++){
        for(j=0;j<r;j++){
    need[i][j]=req[i][j]-alloc[i][j];
}
}
cout<<"need matrix\n";
for(i=0;i<p;i++){
        for(j=0;j<r;j++)
    cout<<need[i][j];
    cout<<endl;
}
int process[100];
int pc=0;
    while(c1!=p){
        c2=c1;
        for(i=0;i<p;i++){
            for(j=0;j<r;j++){
                if(need[i][j]<=avail[j]){
                    track++;
                }
            }
            if(track==r && success[i]==0){
                success[i]=1;
        process[pc]=i;
        pc++;
                for(j=0;j<r;j++)
                    avail[j]=avail[j]+alloc[i][j];
                c1++;
            }
            track=0;
        }
        if(c1==c2){
            cout<<"\nDeadlock present!";
            flag=1;
            break;
        }
    }
    if(flag){
    cout<<"\n the following did not execute\n";
        for(j=0;j<p;j++){
            if(success[j]!=1){
                cout<<" P"<<j;
            }

        }

    }
    else{
        cout<<"No deadlock!";
    cout<<"\n safe sequence is \n";
    for(j=0;j<p;j++){

                cout<<" P"<<process[j];
            }
}
int reqp,flag=0;
int request[50];
cout<<"enter the process number which wishes to request more\n"
cin>>reqp;
cout<<"\nenter its request vector\n";
for(i=0;i<r;i++){
cin>>request[reqp][i];
}
for(i=0;i<r;i++){
if(request[reqp][i]<need[reqp][r])
else
flag=1;
}
for(i=0;i<r;i++){
```

```
if(request[reqp][i]<avail[reqp][r])
else
flag=1;
}
if(flag==1)
cout<<"Cannot carry out request"<<endl;
else{
for(i=0;i<r;i++){
avail[reqp][r]-=request[reqp][r];
alloc[reqp][r]+=request[reqp][r];
need[reqp][r]-=request[reqp][r];
}


c1=0;
c2=0;
pc=0;
while(c1!=p){
        c2=c1;
        for(i=0;i<p;i++){
            for(j=0;j<r;j++){
                if(need[i][j]<=avail[j]){
                    track++;
                }
            }
            if(track==r && success[i]==0){
                success[i]=1;
        process[pc]=i;
        pc++;
                for(j=0;j<r;j++)
                    avail[j]=avail[j]+alloc[i][j];
                c1++;
            }
            track=0;
        }
        if(c1==c2){
            cout<<"\nDeadlock present!";
            flag=1;
            break;
        }
    }
    if(flag){
    cout<<"\n the following did not execute\n";
        for(j=0;j<p;j++){
            if(success[j]!=1){
                cout<<" P"<<j;
            }

        }

    }
    else{
        cout<<"No deadlock!";
    cout<<"\n safe sequence is \n";
    for(j=0;j<p;j++){

                cout<<" P"<<process[j];
            }
}
}
return 0;
}
==============================================================
fifo.c

#include<stdio.h>
int main()
{
 int i,j,n,a[50],frame[10],no,k,avail,count=0;
 printf("\n Enter the number of frames:");
 scanf("%d",&no);
 printf("\n Enter the number of pages:\n");
 scanf("%d",&n);
 printf("\n Enter the page numbers :\n");
 for(i=1;i<=n;i++)
 scanf("%d",&a[i]);
 for(i=0;i<no;i++)
 frame[i]= -1;
 j=0;
 printf("\tref string\t page frames\n");
 for(i=1;i<=n;i++)
 {
 printf("%d\t\t",a[i]);
 avail=0;
 for(k=0;k<no;k++)
if(frame[k]==a[i])
avail=1;
 if (avail==0)
 {
 frame[j]=a[i];
 j=(j+1)%no;
 count++;
 for(k=0;k<no;k++)
 printf("%d\t",frame[k]);
 }
 printf("\n");
 }
 printf("Page Fault Is %d",count);
return 0;
}

==========================================
lru

#include<iostream>
using namespace std;
int fr[20],pages[20];
int main(){
    int i,j,k=0,p,f,flag=0,miss=0;
    cout<<"\nEnter the number of pages :";
    cin>>p;
    cout<<"\nEnter the frame size :";
    cin>>f;
    cout<<"\nEnter the pages :";
    for(i=0;i<p;i++){
        cin>>pages[i];
    }
    for(j=0;j<f;j++){
        fr[j]=-1;
    }

    cout<<"\nCache status is\n";
    for(j=0;j<f;j++){
        fr[j]=pages[j];
        for(i=0;i<f;i++)
            cout<<fr[i]<<" ";
        cout<<endl;
    }
    miss=f;
    for(i=f;i<p;i++){
        k=0;
        for(j=0;j<f;j++){
            if(fr[j]==pages[i]){//if any fram has the req page
                continue;
            }
            else{
                k++;
            }
        }
        if(k==f){
            miss++;
            for(j=0;j<f;j++){
               if(fr[j]==pages[i-f]){//when f=3, i know that 1 and 2 where just refernced, so i gotta
check 3 steps back
```

```
                    fr[j]=pages[i];
                }
            }
        }
            for(j=0;j<f;j++)
                cout<<fr[j]<<" ";
            cout<<endl;


        }
        cout<<"\nFaults : "<<miss;
return 0;
}
=================================================================================
optimal.c

#include<stdio.h>
int n,f,i,page[20],fr[10];
void display(){
    for(i=0;i<f;i++)
        printf("%d ",fr[i]);
    printf("\n");
}

void request(){
printf("Enter the number of pages :");
scanf("%d",&n);
printf("\nEnter the size of page frame :");
scanf("%d",&f);
printf("\nEnter the pages : ");
for(i=0;i<n;i++){
    scanf("%d",&page[i]);
}
for(i=0;i<n;i++){
    fr[i]=-1;
}
}

void replace(){
    int j,nextp[10],flag=0,pf=0;
    int m,max,index;
    for(j=0;j<f;j++){
        fr[j]=page[j];
        pf++;
//        flag=1;
        display();
    }
    for(j=f;j<n;j++){
        flag=0;
        for(i=0;i<f;i++){
            if(fr[i]==page[j]){
                flag=1;
                break;
            }
        }
        if(flag==0){
            for(i=0;i<f;i++)
                nextp[i]=0;
            for(i=0;i<f;i++){
                for(m=j+1;m<n;m++){
                    if(fr[i]==page[m]){
                        nextp[i]=m-j;
                        break;
                    }
                }
            }
            max=nextp[0];
            index=0;
            for(i=0;i<f;i++){
                if(nextp[i]==0){
                    index=i;
                    break;
                }
                else{
                    if(max<nextp[i]){
                        max=nextp[i];
                        index=i;
                    }
                }
            }
            fr[index]=page[j];
            pf++;
            display();

        }
    }
    printf("\npage faults = %d ",pf);

}

int main(){
    request();
    printf("\nCache status is \n");
    replace();
    return 0;
}
======================================
producer-consumer

#include<stdio.h>
#include<semaphore.h>
#include<pthread.h>
#include<stdlib.h>
#include<time.h>
#include<unistd.h>
//#define buffersize 1

pthread_t tidP[20],tidC;
sem_t full,empty,mutex;
int counter,m,k,buffersize;
int buffer[100];

void initialize()
{
    sem_init(&full,0,0);
    sem_init(&empty,0,buffersize);
    sem_init(&mutex,0,1);
    counter=0;
}

void mywrite(int item)
{
    buffer[counter++]=item;
}

int myread()
{
    return(buffer[--counter]);
}

void * producer (void * param)
{
    int i;
    srand(time(NULL));
    while(1)
    {
        m=rand()%100;
        sem_wait(&empty);
        sem_wait(&mutex);
        printf("\nproducer no:%d got the lock\n",i);
        mywrite(m);
        printf("\nProducer put %d",m);
        printf("\nproducer no:%d returns the lock\n",i);
        sem_post(&mutex);
        //printf("\nproducer %ld returns the lock\n",tidP[i]);
        sem_post(&full);
        sleep(7);
    }
}

void * consumer (void * param)
{
    int i;

    while(1)
    {
        sem_wait(&full);
        sem_wait(&mutex);
        printf("\ncustomer no:%d got the lock\n",i);
        i=myread();
        printf("\nConsumer got %d",i);
        printf("\ncustomer no:%d returns the lock\n",i);
        sem_post(&mutex);
        sem_post(&empty);
        sleep(7);
    }
}

int main()
{
    printf("\nEnter the buffer size: ");
    scanf("%d",&buffersize);
    int n1,n2,i;
    initialize();
    printf("\nEnter the no of producers: ");
    scanf("%d",&n1);
    for(i=0;i<n1;i++)
        pthread_create(&tidP[i],NULL,producer,NULL);
    pthread_create(&tidC,NULL,consumer,NULL);
    for(i=0;i<n1;i++)
        pthread_join(tidP[i],NULL);
    pthread_join(tidC,NULL);
    exit(0);
}
```