```
//system calls

//cp


#include<stdio.h>

#include<sys/types.h>

#include<sys/stat.h>

#include<fcntl.h>

#include<stdlib.h>


int main()

{

  int f1;

  int f2;

  int n=0,buf[100];

  f1=open("f1.txt",O_RDWR);

  printf("File opened\n");

  f2=open("f2.txt",O_RDWR|O_CREAT);

  n=read(f1,buf,50);

    write(f2,buf, 50 );


               printf("CONTENTS COPIED");

                close(f1);

                close(f2);

  return 0;

}
```

```
//grep

#include<stdio.h>

#include<sys/types.h>

#include<sys/stat.h>

#include<stdlib.h>

#include<fcntl.h>

#include<string.h>


int main()

{

 char fn[10],pat[10],temp[200];

 FILE *fp;

 printf("Enter file name\n");

 scanf("%s",fn);

 printf("Enter pattern to be searched\n");

 scanf("%s",pat);

 fp=fopen(fn,"r");

 while(!feof(fp))

 {

  fgets(temp,1000,fp);

  if(strstr(temp,pat))

   printf("%s",temp);

 }

 fclose(fp);
}


//list

#include<stdio.h>

#include<dirent.h>

main()

{

char dirname[10];

DIR*p;

struct dirent *d;

printf("Enter directory name\n");

scanf("%s",dirname);

p=opendir(dirname);

if(p==NULL)

  {

  perror("Cannot find directory");

  exit(-1);

  }

while(d=readdir(p))

  printf("%s\n",d->d_name);

}


//SHELL

//arithmatic

#!/bin/bash
```

```
echo $2

case $2 in


"+") echo "Sum is ` expr $1 + $3 `" ;;


"-") echo "Difference is `expr $1 - $3`" ;;


"*") echo "Multiplication is `expr $1 \* $3`" ;;


"/") echo "Division is `expr $1 / $3`" ;;


esac




//* pyramid

clear

echo ---------------------------------------------

echo 'pyramid of number'

echo ---------------------------------------------

echo "Enter the number"

read n

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

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

do
```

```
 echo -e "* \c"

 done

 echo ""

 done


//existfile

#!/bin/bash


FILE=$1

if [ -f $FILE ];

then

echo "$1 FILE EXIST"

else

echo "$1 FILE DOES NOT EXIST"

fi


//pwd/datetime

clear

echo ---------------------------------------------

echo 'current time,date username and directory'

echo ---------------------------------------------

echo date

now="$(date)"

printf "Current date and time %s\n" "$now"

now="$(date +'%d/%m/%Y')"
```

```
printf "Current date in dd/mm/yyyy format %s\n" "$now"

echo current user

u="$USER"

echo "Current user name $u"

echo Current directory

mydir="$(pwd)"

echo "My Current directory:" "$mydir"



//bigfrom3

#!/bin/bash

echo "Enter first number :"

read first

echo "Enter second number :"

read sec

echo "Enter third number :"

read third

if [ $first -gt $sec ] ; then

if [ $first -gt $third ] ; then

echo -e " $first is greatest number "

else

echo -e " $third is greatest number "

fi

else

if [ $sec -gt $third ] ; then
```

```
echo -e " $sec is greatest number "

else

echo -e " $third is greatest number "

fi

fi




//printnum

clear

echo ------------------------

echo -e "\tPyramid"

echo ------------------------

makePyramid()

{

 n=$1;

var=1;

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

  do

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

   do

    echo -ne " ";

   done

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

      do

      echo -ne $var;
```

```
               echo -ne " ";
      done
      for((z=1;z<i;z++))
      do
              echo -ne $var;
 echo -ne " ";
      done
      echo;
var=$((var+1));
  done
}
makePyramid 9
```

```
//FCFSSJF
#include<stdio.h>
struct process
{
 int pid,bt,wt,at,tat;
};

int main()
{
 struct process p[10];
 int i,j,n,temp[10]={0},ch;
```

```
float totalwt=0.0,totaltat=0.0,awt,att;
printf("\nEnter no.of.process :");
scanf("%d",&n);
for(i=0;i<n;i++)
{
 printf("\nEnter the process id:");
 scanf("%d",&p[i].pid);
 printf("\nEnter the burst time:");
 scanf("%d",&p[i].bt);
 printf("\nEnter the arrival time:");
 scanf("%d",&p[i].at);
}
p[0].wt=0;
p[0].tat=0;
do
{
 printf("\n1.FCFS\n2.SJF\n3.exit");
 printf("\nEnter your choice:");
 scanf("%d",&ch);
 switch(ch)
 {
  case 1:
  {
   for(i=0;i<n;i++)
   {
```

```
 temp[i+1]=temp[i]+p[i].bt;
}
for(i=0;i<n;i++)
{
 p[i].wt=temp[i]-p[i].at;
  totalwt=totalwt+p[i].wt;
 p[i].tat=p[i].wt+p[i].bt;
 totaltat=totaltat+p[i].tat;
}
awt=totalwt/n;
att=totaltat/n;
          printf("\n*********FIRST COME FIRST SERVE********");
printf("\nProcess   Arrival time \t  Burst Time \t Waiting Time \t Turn A Time");
for(i=0;i<n;i++)
{
 printf("\nP[%d] \t|\t%d \t|\t%d \t|\t%d \t|\t%d",p[i].pid,p[i].at,p[i].bt,p[i].wt,p[i].tat);
}
printf("\nAverage Waiting Time==>%f",awt);
printf("\nAverage Turnaround Time==>%f\n",att);

//print gantt chart

puts(" ");
puts("      GANTT CHART       ");
puts("      ***********       ");
```

```
//gannt chart 1st line

    printf(" ");
    for(i=0;i<n;i++)
    {
     for(j=0;j<p[i].bt;j++)
       printf("--");
     printf(" ");
    }
    printf("\n|");

//gannt chart second line

    for(i=0;i<n;i++)
    {
     for(j=0;j<p[i].bt-1;j++) printf(" ");
     printf("P%d",p[i].pid);
     for(j=0;j<p[i].bt-1;j++) printf(" ");
     printf("|");
    }
    printf("\n");
    //printing bottom bar
    for(i=0;i<n;i++)
    {
```

```
  for(j=0;j<p[i].bt;j++) printf("--");
  printf(" ");
 }
          //printing timeline
 printf("\n");
 printf("%d",p[0].at);
 for(i=0;i<n;i++)
 {
  for(j=0;j<p[i].bt;j++) printf(" ");
  printf("%d",temp[i+1]+1);
 }
 printf("\n");
break;
}
                                        //***************************************
case 2:
{
 float totalwt=0.0,totaltat=0.0;
 struct process temp1;
 for(i=0;i<n-1;i++)
 {
  for(j=1;j<n-1-i;j++)
  {
   if(p[j].bt>p[j+1].bt)
   {
      temp1=p[j];
      p[j]=p[j+1];
      p[j+1]=temp1;
     }
    }
   }
   for(i=0;i<n;i++)
    temp[i+1]=temp[i]+p[i].bt;
   for(i=0;i<n;i++)
   {
    p[i].wt=temp[i]-p[i].at;
    totalwt=totalwt+p[i].wt;
    p[i].tat=p[i].wt+p[i].bt;
    totaltat=totaltat+p[i].tat;
   }
   awt=totalwt/n;
   att=totaltat/n;

//burst time sorting
for(i=0;i<n;i++)
{
for(j=i+1;j<n;j++)
{
 if(p[i].bt>p[j].bt)
  {
    temp[i]=p[i].bt;

    p[i].bt=p[j].bt;

    p[j].bt=temp[i];

            }

}

}


   /* SORTING

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

    {

        pos=i;

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

        {

            if(bt[j]<bt[pos])

                pos=j;

        }


        temp=bt[i];

        bt[i]=bt[pos];

        bt[pos]=temp;


        temp=p[i];

        p[i]=p[pos];

        p[pos]=temp;

    } */
```

```
          printf("\n*********SHORTEST JOB FIRST************");
 printf("\nProcess   Arrival time \t  Burst Time \t Waiting Time \t Turn A Time");
 for(i=0;i<n;i++)
 {
  printf("\nP[%d] \t|\t%d \t|\t%d \t|\t%d \t|\t%d",p[i].pid,p[i].at,p[i].bt,p[i].wt,p[i].tat);
 }
 printf("\nAverage Waiting Time==>%f",awt);
 printf("\nAverage Turnaround Time==>%f\n",att);

 puts(" ");
 puts("       GANTT CHART        ");
 puts("       ***********        ");

 printf(" ");
//gannt chart first line
 for(i=0;i<n;i++)
 {
  for(j=0;j<p[i].bt;j++)
    printf("--");
  printf(" ");
 }
 printf("\n|");
 //gannt chart second line
 for(i=0;i<n;i++)
 {
     for(j=0;j<p[i].bt-1;j++) printf(" ");
     printf("P%d",p[i].pid);
     for(j=0;j<p[i].bt-1;j++) printf(" ");
     printf("|");
    }
    printf("\n");
    //printing 3rd line
    for(i=0;i<n;i++)
    {
     for(j=0;j<p[i].bt;j++) printf("--");
     printf(" ");
    }
//Calculated Timeline
    printf("\n");
    printf("%d",p[0].at);
    for(i=0;i<n;i++)
    {
     for(j=0;j<p[i].bt;j++) printf(" ");
     printf("%d",temp[i+1]+1);
    }
    printf("\n");
   break;
  }
  case 3: return 0;
  }
```

```
 }while(ch!=3);
 return 0;
}


//RRPRI
#include<stdio.h>

int main()
{
  int bt[20],p[20],wt[20],tat[20],pr[20],i,total=0,pos,temp,avg_wt,avg_tat;
  int count,j,n,time,remain,flag=0,time_quantum;
  int wait_time=0,turnaround_time=0,at[10],rt[10];
  int ch;
  do
  {
   printf("\n1.priority\n2.round-robin\n3.exit");
   printf("\nEnter your choice");
   scanf("%d",&ch);
  switch(ch)
  {
    case 1:
    printf("Enter Total Number of Process:");
    scanf("%d",&n);
```

```
printf("\nEnter Burst Time ,Arrival time and Priority\n");

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

{

    printf("\nP[%d]\n",i+1);

    printf("Burst Time:");

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

    printf("\nArrival time:");

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

    printf("Priority:");

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

    p[i]=i+1;           //contains process number

}


//sorting burst time, priority and process number in ascending order using selection sort

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

{

    pos=i;

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

    {

        if(pr[j]<pr[pos])

            pos=j;

    }


    temp=pr[i];

    pr[i]=pr[pos];
    pr[pos]=temp;

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

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

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

wt[0]=0;    //waiting time for first process is zero

//calculate waiting time
for(i=1;i<n;i++)
{
    wt[i]=0;
    for(j=0;j<i;j++)
        wt[i]+=bt[j];

    total+=wt[i];
```

```
  }


  avg_wt=total/n;      //average waiting time

  total=0;


  printf("\nProcess\t    Burst Time    \tWaiting Time\tTurnaround Time");

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

  {

     tat[i]=bt[i]+wt[i];     //calculate turnaround time

     total+=tat[i];

     printf("\nP[%d]\t\t  %d\t\t    %d\t\t\t%d",p[i],bt[i],wt[i],tat[i]);

  }


  avg_tat=total/n;     //average turnaround time

  printf("\n\nAverage Waiting Time=%d",avg_wt);

  printf("\nAverage Turnaround Time=%d\n",avg_tat);

  break;


case 2:

     printf("Enter Total Process:\t ");

     scanf("%d",&n);

     remain=n;

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

     {

      printf("Enter Arrival Time and Burst Time for Process Process Number %d :",count+1);
```

```
 scanf("%d",&at[count]);

 scanf("%d",&bt[count]);

 rt[count]=bt[count];

}

      printf("Enter Time Quantum:\t");

      scanf("%d",&time_quantum);

      printf("\n\nProcess\t|Turnaround Time|Waiting Time\n\n");

      for(time=0,count=0;remain!=0;)

      {

       if(rt[count]<=time_quantum && rt[count]>0)

       {

        time+=rt[count];

        rt[count]=0;

       flag=1;

       }

       else if(rt[count]>0)

       {

        rt[count]-=time_quantum;

  time+=time_quantum;

       }

       if(rt[count]==0 && flag==1)

 {

        remain--;

        printf("P[%d]\t|\t%d\t|\t%d\n",count+1,time-at[count],time-at[count]-bt[count]);

  wait_time+=time-at[count]-bt[count];
```

```
               turnaround_time+=time-at[count];
               flag=0;
              }
              if(count==n-1)
         count=0;
              else if(at[count+1]<=time)
               count++;
              else
               count=0;
             }
      printf("\nAverage Waiting Time= %f\n",wait_time*1.0/n);
      printf("Avg Turnaround Time = %f",turnaround_time*1.0/n);
      return 0;
     break;

 case 3: exit(0);
       break;

 default:printf("\nInvalid coice");
 }
}while(ch!=3);
}


//producerconsumersemaphore
```

```
//producer
#include<stdlib.h>
#include<string.h>
#include<semaphore.h>
#include<pthread.h>
#include<sys/ipc.h>
#include<sys/shm.h>
#include<sys/sem.h>
#include<sys/wait.h>
#include<sys/errno.h>
#include<sys/types.h>
#include<stdio.h>
#include<unistd.h>
#include<iostream>
using namespace std;
sem_t *mutex,*empty,*full;
int main()
{
int n;
cout<<"Enter total number of items to be produced: ";
cin>>n;
string s;
cout<<"\nEnter Item ";
cin>>s;
int i,j,k,id,eid,fid,mid;
```

```
id=shmget(111,50,IPC_CREAT | 00666);

eid=shmget(211,50,IPC_CREAT | 00666);

fid=shmget(311,50,IPC_CREAT | 00666);

mid=shmget(411,50,IPC_CREAT | 00666);

empty=(sem_t*)shmat(eid,NULL,0);

full=(sem_t*)shmat(fid,NULL,0);

mutex=(sem_t*)shmat(mid,NULL,0);

sem_init(empty,1,n);

sem_init(full,1,0);

sem_init(mutex,1,1);

for(i=0;i<s.length();++i)

{

cout<<"\nProducer trying to acquire empty";

sem_wait(empty);

cout<<"\nProducer successfully acquired empty";

cout<<"\nProducer trying to acquire mutex";

sem_wait(mutex);

cout<<"\nProducer successfully acquired mutex";

cout<<"\nProduced Item: "<<s[i];

char *a;

a=(char*)shmat(id,NULL,0);

a[i]=s[i];

a[i+1]='\0';

//shmdt(a);

sem_post(mutex);
cout<<"\nProducer released mutex";

sem_post(full);

cout<<"\nProducer released full";

sleep(10);

cout<<"\n\n";

}

sleep(10);

shmdt(full);

shmdt(empty);

shmdt(mutex);

sem_destroy(full);

sem_destroy(empty);

sem_destroy(mutex);

cout<<"\nProducer exited";

}


//consumer

#include<stdlib.h>

#include<string.h>

#include<semaphore.h>

#include<pthread.h>

#include<sys/ipc.h>

#include<sys/shm.h>

#include<sys/sem.h>

#include<sys/wait.h>
```

```
#include<sys/errno.h>

#include<sys/types.h>

#include<stdio.h>

#include<unistd.h>

#include<iostream>

using namespace std;

sem_t *mutex,*empty,*full;

int main()

{

int n;

int i,j,k,id,eid,fid,mid,x;

id=shmget(111,50,IPC_CREAT | 00666);

eid=shmget(211,50,IPC_CREAT | 00666);

fid=shmget(311,50,IPC_CREAT | 00666);

mid=shmget(411,50,IPC_CREAT | 00666);

empty=(sem_t*)shmat(eid,NULL,0);

full=(sem_t*)shmat(fid,NULL,0);

mutex=(sem_t*)shmat(mid,NULL,0);

sem_init(empty,1,100);

sem_init(full,1,0);

sem_init(mutex,1,1);

char *a;

i=0;

while(i!=5)

{
```

```
sleep(10);

a=(char*)shmat(id,NULL,0);

cout<<"\nConsumer trying to acquire full";

sem_wait(full);

cout<<"\nConsumer successfully acquired full";

cout<<"\nConsumer trying to acquire mutex";

sem_wait(mutex);

cout<<"\nConsumer successfully acquired mutex";

a=(char*)shmat(id,NULL,0);

cout<<"\nConsumed Item: "<<a[i++];

sem_post(mutex);

cout<<"\nConsumer released mutex";

sem_post(empty);

cout<<"\nConsumer released empty";

cout<<"\n\n";

}

sem_destroy(full);

sem_destroy(empty);

sem_destroy(mutex);

shmdt(full);

shmdt(empty);

shmdt(mutex);

cout<<"\nConsumer exited";

}
```

```
//Bankers

#include<stdlib.h>

#include<iostream>

#include<vector>

using namespace std;

int main()

{

int ch;

do

{

cout<<"\n1.Read Data\t2.Print Data\t3.Safety sequence\t4.Resource request\n";

int n,m,all[100][100],max[100][100],need[100]

[100],available[100],finish[100],ch1,t[100];

cin>>ch1;

if(ch1==1)

{

cout<<"\nEnter the number of processes\n";

cin>>m;

cout<<"\nEnter the number of resources\n";

cin>>n;

cout<<"\nEnter the allocated resources\n";

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

{

cout<<"\nEnter allocated resources for P"<<i<<endl;
```

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

cin>>all[i][j];

} cout<<"\nEnter the maximum resources\n";

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

{

cout<<"\nEnter maximum resources for P"<<i<<endl;

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

cin>>max[i][j];

}

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

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

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

cout<<"\nEnter the available resources\n";

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

cin>>available[i];

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

t[i]=available[i];

} else if(ch1==2)

{

cout<<"\nAlloc\tMax\tNeed\tAvail\n";

for(int i=0,c=0;i<(4*n);i++,c++)

{

if(c>n-1)

c-=n;

if(c==n-1)
```

```
cout<<(char)('A'+c)<<"\t";
else
cout<<(char)('A'+c)<<" ";
} cout<<endl;
int temp[100][100]={0};
for(int i=0;i<m;i++)
for(int j=0;j<n;j++)
temp[i][j]=all[i][j];
for(int i=0;i<m;i++)
for(int j=n;j<(2*n);j++)
temp[i][j]=max[i][j%n];
for(int i=0;i<m;i++)
for(int j=(2*n);j<(3*n);j++)
temp[i][j]=need[i][j%n];
for(int i=0;i<m;i++)
{
if(i==0)
for(int j=(3*n);j<(4*n);j++)
temp[i][j]=available[j%n];
else
for(int j=(3*n);j<(4*n);j++)
temp[i][j]=10000;
}
for(int i=0;i<m;i++)
{
```

```
for(int j=0;j<(4*n);j++)

if((j+1)%n==0)

{

if(temp[i][j]!=10000)

cout<<temp[i][j]<<"\t";

} else

{

if(temp[i][j]!=10000)

cout<<temp[i][j]<<" ";

} cout<<endl;

}

} else if(ch1==3)

{

vector<int> v;

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

finish[i]=0;

for(int i=0,c=0;;i++,c++)

{

if(i>=m)

i=i-m;

if(finish[i])

continue;

int flag=1;

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

if(need[i][j]>available[j])
```

```
{

flag=0;

break;

}

if(flag)

{

v.push_back(i);

finish[i]=1;

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

available[j]+=all[i][j];

} if(!flag&&c>m)

break;

if(v.size()==m)

break;

}

vector<int>::iterator it;

if(v.size()<m)

cout<<"\nNo safe sequence\n";

else

{

cout<<"\nSafety sequence\n<";

for(it=v.begin();it!=v.end();it++)

if((it+1)!=v.end())

cout<<*it<<",";

else
```

```
cout<<*it<<">"<<endl;

}

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

available[i]=t[i];

} else if(ch1==4)

{

cout<<"\nRequest from which process\n";

int k;

cin>>k;

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

finish[i]=0;

cout<<"\nEnter the request\n";

int request[100];

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

cin>>request[j];

int flag=1;

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

if(request[j]>need[k][j])

{

flag=0;

break;

}

if(!flag)

{

cout<<"\nRequest can't be granted\n";
```

```
continue;

}

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

if(request[j]>available[j])

{

flag=0;

break;

}

if(!flag)

{

cout<<"\nRequest can't be granted\n";

continue;

}

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

available[j]-=request[j];

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

all[k][j]+=request[j];

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

need[k][j]-=request[j];

vector<int> v;

for(int i=0,c=0;;i++,c++)

{

if(i>=m)

i=i-m;

if(finish[i])
```

```
continue;

int flag=1;

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

if(need[i][j]>available[j])

{

flag=0;

break;

}

if(flag)

{

v.push_back(i);

finish[i]=1;

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

available[j]+=all[i][j];

} if(!flag&&c>m)

break;

if(v.size()==m)

break;

}

vector<int>::iterator it;

if(v.size()<m)

cout<<"\nNo safe sequence\n";

else

{

cout<<"\nSafety sequence\n<";
```

```
for(it=v.begin();it!=v.end();it++)

if((it+1)!=v.end())

cout<<*it<<",";

else

cout<<*it<<">"<<endl;

}

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

available[i]=t[i];

} else

cout<<"\nChoice is invalid\n";

cout<<"\nDo you want to continue?\n";

cin>>ch;

}while(ch==1);

return 0;

}


//deadlock

#include<iostream>

#include<queue>

#include<stdlib.h>

using namespace std;

//n - No. Of Processes

//m - No. Of Resources

//max-alloc=need

//r--->available
```

```
struct process

{

int alloc[10],max[10],req[10],need[10];

string id;

};

void read(process a[],int n,int r[],int m)

{

cout<<"Enter available resources:";

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

{

cin>>r[i];

//r1[i]=r[i];

}

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

{

cout<<"Enter process id:";

cin>>a[i].id;

cout<<"Enter process allocation:";

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

cin>>a[i].alloc[j];

cout<<"Enter process maximum:";

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

cin>>a[i].max[j];

}

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

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

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

}

}

void write(process a[],int n,int r[],int m)

{

cout<<"Pid Allocation Maximum Need Available\n";

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

{

cout<<a[i].id<<" ";

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

cout<<a[i].alloc[j]<<" ";

cout<<" ";

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

cout<<a[i].max[j]<<" ";

cout<<" ";

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

cout<<a[i].need[j]<<" ";

if(i==0)

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

cout<<r[j]<<" ";

cout<<endl;

}

}
```

```
void safety_seq(process a[],int n,int r[],int m)

{

queue<string> q;

int counter=0,count=0,counting=0;

bool finish[10];

string buffer,str[10];

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

finish[i]=false;

int k=0,l=0,st=0;

while(1)

{

l=0;

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

{

if(finish[i]==false)

{

counter=0;

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

if(a[i].need[j]<=r[j])

counter++;

if(counter==m)

{

l++;

q.push(a[i].id);

count++;
```

```
for(int j=0;j<m;j++)
r[j]+=a[i].alloc[j];
finish[i]=true;
}
}
}
if(l==0 && count!=n)
{
k++;
}
if(k>2)
{
cout<<"\nProcess that can cause deadlock:";
buffer="no";
while (!q.empty())
{
str[st++]=q.front();
q.pop();
}
for(int ii=0;ii<n;ii++)
{
counting=0;
for(int jj=0;jj<st;jj++)
{
if(str[jj]==a[ii].id)
```

```
counting=1;

}

if(counting==0)

cout<<a[ii].id<<" ";

}

break;

}

if(count==n)

break;

}

if(buffer!="no")

{

cout<<"\nNo deadlock possibility\n";

cout<<endl;

}

}

void proc_req(process a[],int n,int r[],int m)

{

int req[10],r1[10],c=0;

string id1;

cout<<"Enter process id:";

cin>>id1;

cout<<"Enter the request:";

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

{
```

```
cin>>req[i];

}

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

{

if(a[j].id==id1)

{

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

{

if(req[i]>r[i] || req[i]>a[j].need[i])

{

c=-1;

cout<<"Request Cannot Be Granted\n";

break;

}

else

r[i]-=req[i];

}

}

}

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

if(a[i].id==id1)

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

a[i].alloc[j]+=req[j];

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

{
```

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

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

}

}

int main()

{

process a[10];

int choice,n,m,r[10],r1[10];

while(1)

{

cout<<"Enter 1)Read Data 2)Print Data 3)Run Safety Algorithm 4)Process Request 5)Exit:";

cin>>choice;

switch(choice)

{

case 1:cout<<"Enter number of processes:";

cin>>n;

cout<<"Enter number of resources:";

cin>>m;

read(a,n,r,m);

break;

case 2:write(a,n,r,m);

break;

case 3:safety_seq(a,n,r,m);

break;

case 4:proc_req(a,n,r,m);
```

```
break;

case 5:exit(1);

break;

}

}

}


//thread

#include<pthread.h>

#include<stdio.h>

#include<math.h>

#include<stdlib.h>

int sum,max,min,median,sum2,n;

float average,sd;

void *worker_average(void *p);

void *worker_max(void *p);

void *worker_min(void *p);

void *worker_median(void *p);

void *worker_sd(void *p);

struct input

{

char **data;

int count;

}in;

int main(int argc,char**argv)
```

```
{

pthread_t tid,tid1,tid2,tid3,tid4,tid5;

pthread_attr_t attr;

pthread_attr_init(&attr);

in.data=argv;

in.count=argc;

n=argc-1;

pthread_create(&tid,&attr,worker_average,&in);

pthread_create(&tid1,&attr,worker_max,&in);

pthread_create(&tid2,&attr,worker_min,&in);

pthread_create(&tid4,&attr,worker_median,&in);

pthread_create(&tid5,&attr,worker_sd,&in);

pthread_join(tid,NULL);

pthread_join(tid1,NULL);

pthread_join(tid2,NULL);

pthread_join(tid5,NULL);

pthread_join(tid4,NULL);

//printf("Sum=%d\n",sum);

printf("Max=%d\n",max);

printf("Average=%f\n",average);

printf("Min=%d\n",min);

printf("Median=%d\n",median);

printf("Standard deviation:%f\n",sd);

}

void *worker_average(void *param)
```

```
{

struct input *t=(struct input*)param;

int i;

sum=0;

for(i=0;i<t->count;i++)

sum+=atoi(t->data[i]);

average=(float)(sum)/(t->count-1);

pthread_exit(0);

}

void *worker_max(void *param)

{

struct input *t=(struct input*)param;

int i;

max=0;

for(i=1;i<t->count;i++)

if(atoi(t->data[i])>max)

max=atoi(t->data[i]);

pthread_exit(0);

}

void *worker_min(void *param)

{

struct input *t=(struct input*)param;

int i,j;

min=atoi(t->data[1]);

for(i=1;i<t->count;i++)
if(atoi(t->data[i])<min)

min=atoi(t->data[i]);

pthread_exit(0);

}

void *worker_median(void *param)

{

struct input *t=(struct input*)param;

int i,j;

median=0;

char* temp;

for(i=0;i<t->count-1;i++)

{ for(j=i+1;j<t->count;j++)

{

if(t->data[j]<t->data[i])

{

temp=t->data[j];

t->data[j]=t->data[i];

t->data[i]=temp;

}

}

}

median=atoi(t->data[t->count/2]);

pthread_exit(0);

}

void *worker_sd(void *param)
```

```
{
struct input *t=(struct input *)param;
int i,j;
float x;
sd=0;
float sum2=0;
float sum1=0;
for(i=1;i<t->count;i++)
{
x=(float)(atoi(t->data[i]));
sum1+=x;
sum2+=pow(x,2);
}
x=(sum2/n)-(sum1*sum1/(n*n));
sd=sqrt(x);
pthread_exit(0);
}



//paging
#include<iostream>
#include<stdlib.h>
#include<list>
#include<algorithm>
#include<string.h>
```

```
using namespace std;

struct process

{

string id;

int frames[20];

int size;

};

int main()

{

int pgmem,pgsize,framesize,ch,count =1,x;

cout<<"Enter the physical memory:"<<endl;

cin>>pgmem;

cout<<"Enter the page size:"<<endl;

cin>>pgsize;

framesize=pgmem/pgsize;

x=9;

cout<<"The page is divided into "<<framesize<<"frames"<<endl;

list<int> free_list;

for(int i=0;i<9;i++)

{

free_list.push_back(rand()%framesize);

}

list<int>::iterator it;

it=unique(free_list.begin(),free_list.end());

free_list.resize(distance(free_list.begin(),it));
int i=0;

struct process p[10];

while(ch!=5)

{

cout<<"Enter 1)Process request\n2)deallocation\n3)Page table display\n4)Free

frame list\n5)Exit";

cout<<endl;

cin>>ch;

if(ch==1)

{

cout<<"Enter the process Id and the size:";

cin>>p[i].id>>p[i].size;

if(p[i].size>x)

{

cout<<"Sorry..Your frame size is exceeded."<<endl;

exit(0);

}

for(int j=0;j<p[i].size;j++)

{

p[i].frames[j]=free_list.front();

free_list.pop_front();

}

x=x-p[i].size;

for(int j=0;j<p[i].size;j++)

{
```

```
cout<<"Page "<<j<<":"<<p[i].frames[j]<<endl;

}

i++;

count++;

cout<<endl;

}

if(ch==4)

{

cout<<"Free frame list:"<<endl;

list<int>::iterator it1;

for(it1=free_list.begin();it1!=free_list.end();it1++)

{

cout<<*it1<<"\t";

}

cout<<endl;

}

if(ch==3)

{

if(i==0)

{

cout<<"Sorry.First allocate the pages."<<endl;

continue;}

else

{

cout<<"Displaying page table for all process:"<<endl;
```

```
cout<<endl;

for(int k=0;k<count-1;k++)

{

if(p[k].frames[0]==-1)

{

continue;

}

else{

cout<<"Page table for "<<p[k].id<<endl;

for(int l=0;l<p[k].size;l++)

cout<<p[k].frames[l]<<"\t";

cout<<endl;

}

}

}

}

if(ch==2)

{

int k;

string id1;

cout<<"Enter the process to be deallocated \t";

cin>>id1;

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

{

if(id1==p[k].id)
```

```
{
break;
}
}
for(int j=0;j<p[k].size;j++)
{
free_list.push_back(p[k].frames[j]);
}
p[k].frames[0]=-1;
x=x+p[k].size;
i--;
cout<<"Process deallocated"<<endl;
cout<<endl;
}
}
}
```