**DINNING PHILOSOPHERS**

#include<stdio.h>

#include<stdlib.h>

#include<conio.h>

int i,tph,howhung,cho;

#include<stdio.h>

#include<stdlib.h>

#include<conio.h>

int i,tph,howhung,cho;

int philname[100];

int status[100];

int hu[100];

void main()

{

printf("\n\nDINING PHILOSOPHER PROBLEM");

printf("\nEnter the total no. of philosophers: ");

scanf("%d",&tph);

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

{

philname[i] = (i+1);

status[i]=1;

}

printf("How many are hungry : ");

scanf("%d", &howhung);

if(howhung==tph)

{

printf("\nAll are hungry..\nDead lock stage will occur");

printf("\nExiting..");

}

else

{

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

{

printf("Enter philosopher %d position: ",(i+1));

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

status[hu[i]]=2;

}

do

{

printf("1.One can eat at a time\t2.Two can eat at a time\t3.Exit\nEnter your choice:");

scanf("%d", &cho);

switch(cho)

{

case 1: one();

break;

case 2: two();

break;

case 3: exit(0);

default: printf("\nInvalid option..");

}}while(1);

}

}

one()

{

int pos=0, x, i;

printf("\nAllow one philosopher to eat at any time\n");

for(i=0;i<howhung; i++, pos++)

{

printf("\nP %d is granted to eat", philname[hu[pos]]);

for(x=pos;x<howhung;x++)

printf("\nP %d is waiting", philname[hu[x]]);

}

}

two()

{

int i, j, s=0, t, r, x;

printf("\n Allow two philosophers to eat at same time\n");

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

{

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

{

if(abs(hu[i]-hu[j])>=1&& abs(hu[i]-hu[j])!=4)

{

printf("\n\ncombination %d \n", (s+1));

t=hu[i];

r=hu[j];

s++;

printf("\nP %d and P %d are granted to eat", philname[hu[i]],

philname[hu[j]]);

for(x=0;x<howhung;x++)

{

if((hu[x]!=t)&&(hu[x]!=r))

printf("\nP %d is waiting", philname[hu[x]]);

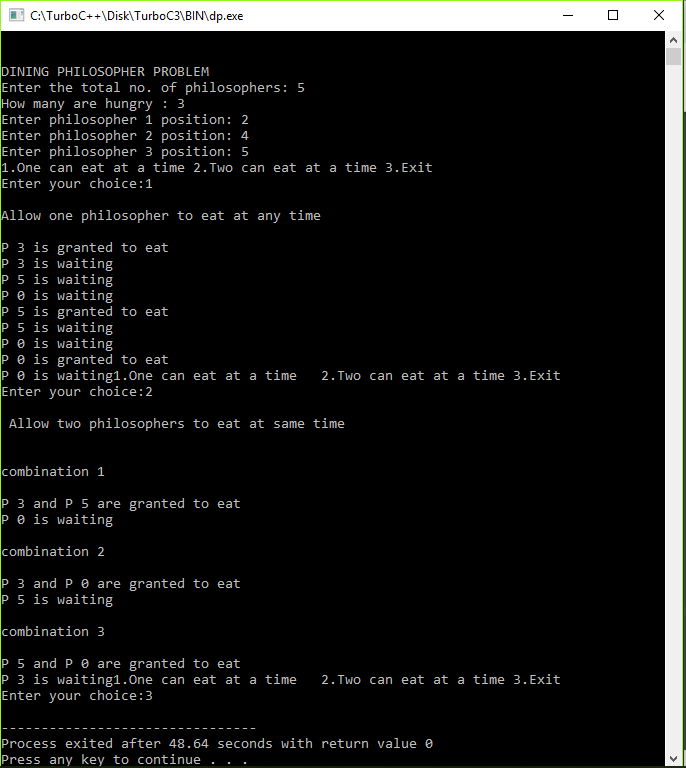
}

}

}

}

}



**BANKER’S ALGORITHM**

#include<stdio.h>

#include<stdlib.h>

#include<conio.h>

struct file

{

int all[10];

int max[10];

int need[10];

int flag;

};

void main()

{

struct file f[10];

int fl;

int i, j, k, p, b, n, r, g, cnt=0, id, newr;

int avail[10],seq[10];

printf("Enter number of processes -- ");

scanf("%d",&n);

printf("Enter number of resources -- ");

scanf("%d",&r);

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

{

printf("Enter details for P%d",i);

printf("\nEnter allocation\t -- \t");

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

scanf("%d",&f[i].all[j]);

printf("Enter Max\t\t -- \t");

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

scanf("%d",&f[i].max[j]);

f[i].flag=0;

}

printf("\nEnter Available Resources\t -- \t");

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

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

printf("\nEnter New Request Details -- ");

printf("\nEnter pid \t -- \t");

scanf("%d",&id);

printf("Enter Request for Resources \t -- \t");

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

{

scanf("%d",&newr);

f[id].all[i] += newr;

avail[i]=avail[i] - newr;

}

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

{

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

{

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

if(f[i].need[j]<0)

f[i].need[j]=0;

}

}

cnt=0;

fl=0;

while(cnt!=n)

{

g=0;

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

{

if(f[j].flag==0)

{

b=0;

for(p=0;p<r;p++)

{

if(avail[p]>=f[j].need[p])

b=b+1;

else

b=b-1;

}

if(b==r)

{

printf("\nP%d is visited",j);

seq[fl++]=j;

f[j].flag=1;

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

avail[k]=avail[k]+f[j].all[k];

cnt=cnt+1;

printf("(");

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

printf("%3d",avail[k]);

printf(")");

g=1;

}

}

}

if(g==0)

{

printf("\n REQUEST NOT GRANTED -- DEADLOCK OCCURRED");

printf("\n SYSTEM IS IN UNSAFE STATE");

goto y;

}

}

printf("\nSYSTEM IS IN SAFE STATE");

printf("\nThe Safe Sequence is -- (");

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

printf("P%d ",seq[i]);

printf(")");

y: printf("\nProcess\t\tAllocation\t\tMax\t\t\tNeed\n");

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

{

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

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

printf("%6d",f[i].all[j]);

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

printf("%6d",f[i].max[j]);

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

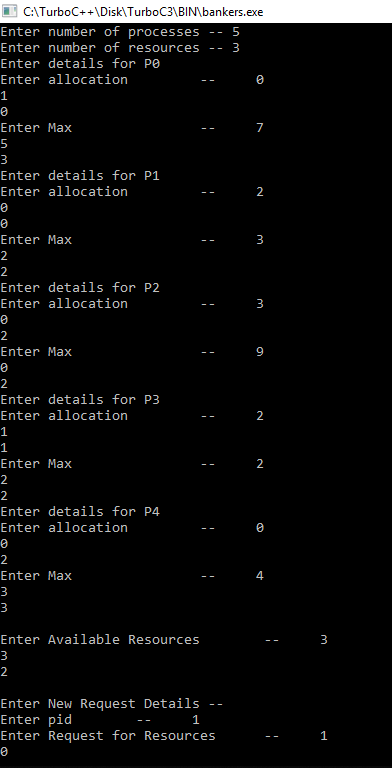
printf("%6d",f[i].need[j]);

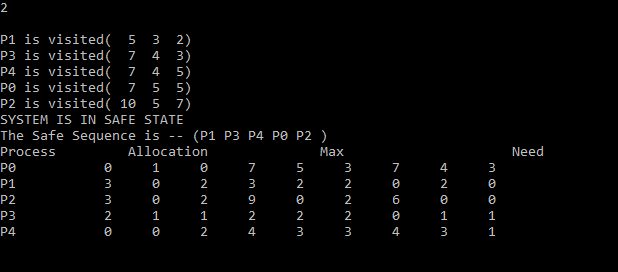
printf("\n");

}

getch();

}





**READER**

#include<stdio.h>

#include<pthread.h>

#include<semaphore.h>

sem\_t mutex;

sem\_t db;

int readercount=0;

pthread\_t reader1,reader2,writer1,writer2;

void \*reader(void \*);

void \*writer(void \*);

main()

{

sem\_init(&mutex,0,1);

sem\_init(&db,0,1);

while(1)

{

pthread\_create(&reader1,NULL,reader,"1");

pthread\_create(&reader2,NULL,reader,"2");

pthread\_create(&writer1,NULL,writer,"1");

pthread\_create(&writer2,NULL,writer,"2");

}

}

void \*reader(void \*p)

{

printf("prevoius value %dn",mutex);

sem\_wait(&mutex);

printf("Mutex acquired by reader %dn",mutex);

readercount++;

if(readercount==1) sem\_wait(&db);

sem\_post(&mutex);

printf("Mutex returned by reader %dn",mutex);

printf("Reader %s is Readingn",p);

//sleep(3);

sem\_wait(&mutex);

printf("Reader %s Completed Readingn",p);

readercount--;

if(readercount==0) sem\_post(&db);

sem\_post(&mutex);

}

void \*writer(void \*p)

{

printf("Writer is Waiting n");

sem\_wait(&db);

printf("Writer %s is writingn ",p);

sem\_post(&db);

//sleep(2);

}

