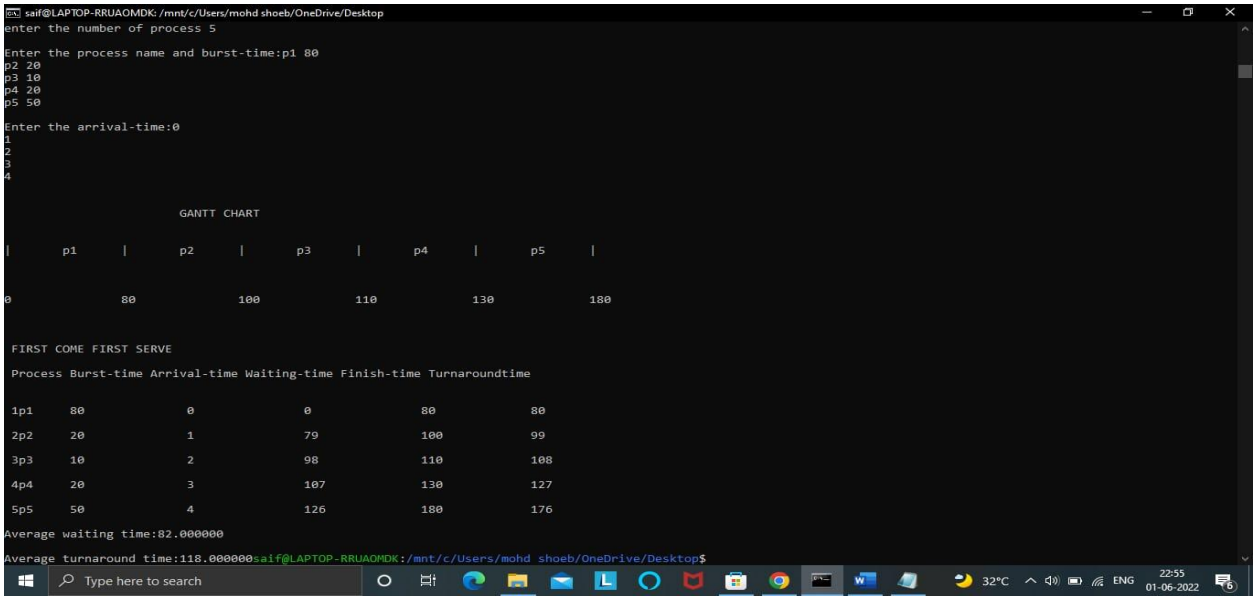
1.FCFS Program:

```
#include<stdio.h>
void main()
{
int bt[50],wt[80],at[80],wat[30],ft[80],tat[80];
int i,n;
float awt,att,sum=0,sum1=0;
char p[10][5];
printf("\nenter the number of process ");
scanf("%d",&n);
printf("\nEnter the process name and burst-time:");
for(i=0;i<n;i++)
scanf("%s%d",p[i],&bt[i]);
printf("\nEnter the arrival-time:");
for(i=0;i<n;i++)
scanf("%d",&at[i]);
wt[0]=0;
for(i=1;i<=n;i++)
wt[i]=wt[i-1]+bt[i-1];
ft[0]=bt[0];
for(i=1;i<=n;i++)
ft[i]=ft[i-1]+bt[i];
printf("\n\n\t\t\tGANTT CHART\n");
printf("\n \n");
for(i=0;i<n;i++)
printf("|\t%s\t",p[i]);
printf("|\t\n");
printf("\n \n");
```

```
printf("\n");
for(i=0;i<n;i++)
printf("%d\t\t",wt[i]);
printf("%d",wt[n]+bt[n]);
printf("\n \n");
printf("\n");
for(i=0;i<n;i++)
wat[i]=wt[i]-at[i];
for(i=0;i<n;i++)
tat[i]=ft[i]-at[i];
printf("\n FIRST COME FIRST SERVE\n");
printf("\n Process Burst-time Arrival-time Waiting-time Finish-time Turnaroundtime\n");
for(i=0;i<n;i++)
printf("\n\n %d%s \t %d\t\t %d \t\t %d\t\t %d \t\t%d",i+1,p[i],bt[i],at[i],wat[i],ft[i],tat[i]);
for(i=0;i<n;i++)
sum=sum+wat[i];
awt=sum/n;
for(i=0;i<n;i++)
sum1=sum1+tat[i];
att=sum1/n;
printf("\n\nAverage waiting time:%f",awt);
printf("\n\nAverage turnaround time:%f",att);
}
```

OUTPUT:



ii) Write a C program to illustrate the following IPC mechanisms
a) Pipes b) FIFOs c) Message Queue d)Shared Memory

a)Pipes

```
#include<stdio.h>
#include<unistd.h>
int main() {
  int pipefds[2];
  int returnstatus;
  int pid;
  char writemessages[2][20]={"Hi", "Hello"};
  char readmessage[20];
  returnstatus = pipe(pipefds);
  if (returnstatus == -1) {
    printf("Unable to create pipe\n");
    return 1;
  }
  pid = fork();
 if (pid == 0) {
```

```
    read(pipefds[0], readmessage, sizeof(readmessage));
    printf("Child Process - Reading from pipe – Message 1 is %s\n", readmessage);
    read(pipefds[0], readmessage, sizeof(readmessage));
    printf("Child Process - Reading from pipe – Message 2 is %s\n", readmessage);
  } else { //Parent process
    printf("Parent Process - Writing to pipe - Message 1 is %s\n", writemessages[0]);
    write(pipefds[1], writemessages[0], sizeof(writemessages[0]));
    printf("Parent Process - Writing to pipe - Message 2 is %s\n", writemessages[1]);
    write(pipefds[1], writemessages[1], sizeof(writemessages[1]));
  }
  return 0;
}
```

OUTPUT:

b)FIFO

fifo1.c:

```
#include <stdio.h>
#include <string.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>

int main()
{
   int fd;


   char * myfifo = "/tmp/myfifo";

   mkfifo(myfifo, 0666);
```

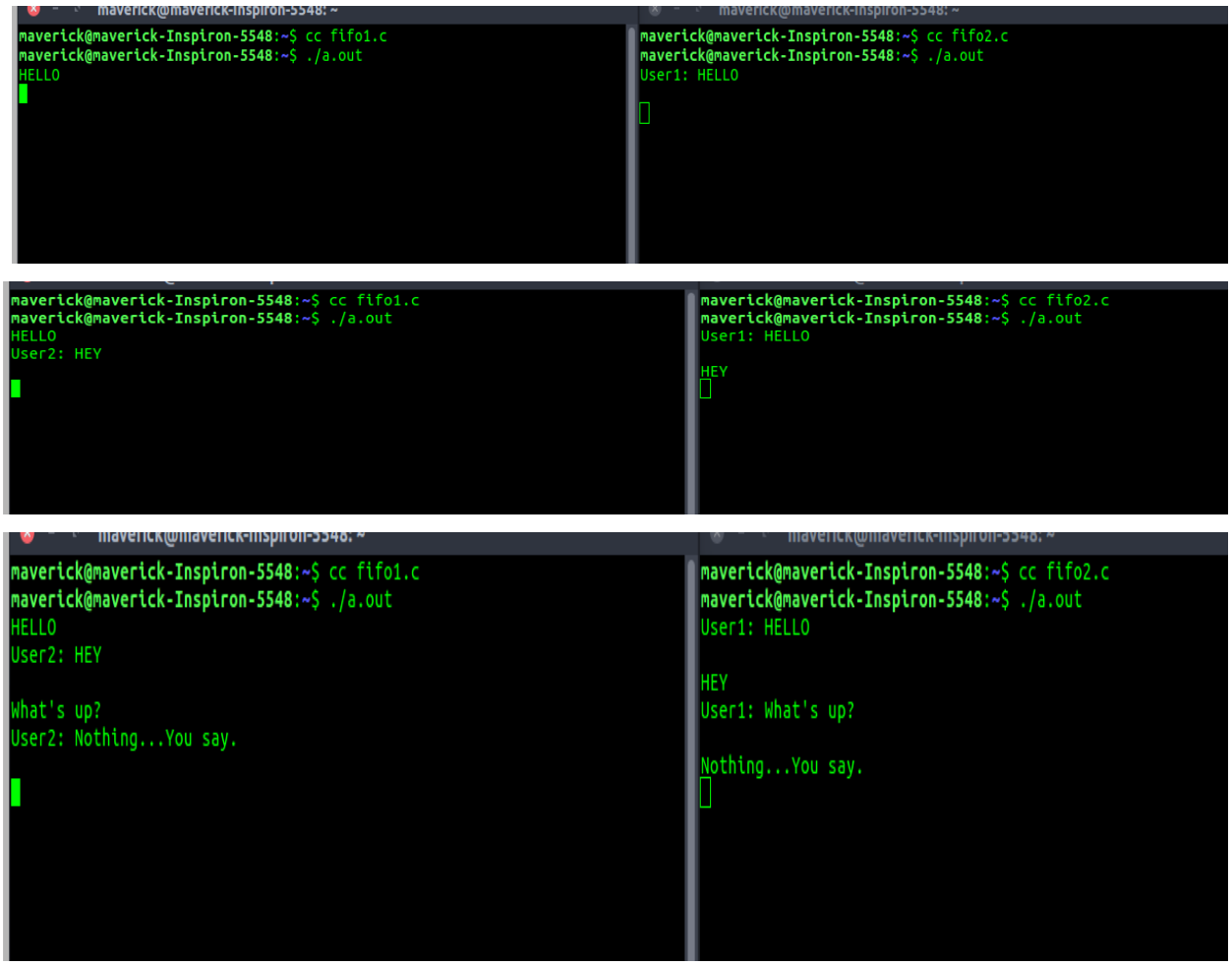
```
    char arr1[80], arr2[80];
    while (1)
    {

        fd = open(myfifo, O_WRONLY);

        fgets(arr2, 80, stdin);



        write(fd, arr2, strlen(arr2)+1);
        close(fd);


        fd = open(myfifo, O_RDONLY);


        read(fd, arr1, sizeof(arr1));


        printf("User2: %s\n", arr1);
        close(fd);
    }
    return 0;
}
```

Fifo2.c:

```
 #include <stdio.h>
#include <string.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <sys/types.h>
```

```
#include <unistd.h>

int main()
{
   int fd1;


   char * myfifo = "/tmp/myfifo";


   mkfifo(myfifo, 0666);

   char str1[80], str2[80];
   while (1)
   {

      fd1 = open(myfifo,O_RDONLY);
      read(fd1, str1, 80);


      printf("User1: %s\n", str1);
      close(fd1);


      fd1 = open(myfifo,O_WRONLY);
      fgets(str2, 80, stdin);
      write(fd1, str2, strlen(str2)+1);
      close(fd1);
   }
   return 0;
}
```

OUTPUT:



C. Message Queues

(i)Message_send

```
#include <stdio.h>
#include <sys/ipc.h>
#include <sys/msg.h>

struct mesg_buffer {
    long mesg_type;
    char mesg_text[100];
} message;

int main()
{
    key_t key;
```

```
    int msgid;

    key = ftok("somefile", 65);

    msgid = msgget(key, 0666 | IPC_CREAT);
    message.mesg_type = 1;

    printf("Insert message  : ");
    gets(message.mesg_text);

    msgsnd(msgid, &message, sizeof(message), 0);



    printf("Message sent to server : %s\n", message.mesg_text);

    return 0;
}
```

OUTPUT:

(ii) message_recieve

```
#include <stdio.h>
#include <sys/ipc.h>
#include <sys/msg.h>

struct mesg_buffer {
    long mesg_type;
    char mesg_text[100];
} message;

int main()
{
    key_t key;
    int msgid;


    key = ftok("somefile", 65);

    msgid = msgget(key, 0666 | IPC_CREAT);
```

```
    printf("Waiting for a message from client...\n");

    msgrcv(msgid, &message, sizeof(message), 1, 0);


    printf("Message received from client : %s\n",message.mesg_text);


    msgctl(msgid, IPC_RMID, NULL);

    return 0;
}
```

OUTPUT:

d) SHARED MEMORY FOR WRITER PROCESS

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/shm.h>
#include <string.h>

int main()
{
        void *shared_memory;
        char buff[100];
        int shmid;
        shmid=shmget((key_t)1122,1024,0666|IPC_CREAT);
        printf("Key of Shared Memory is %d\n",shmid);
        shared_memory=shmat(shmid,NULL,0);
        printf("Process attached at %p\n",shared_memory);
        read(0,buff,100);
        strcpy(shared_memory,buff);
        printf("You wrote : %s\n",(char*)shared_memory);
}
```

OUTPUT:

SHARED MEMORY FOR READER PROCESS

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/shm.h>
#include <string.h>

int main()
{
void *shared_memory;
char buff[100];
int shmid;
```

```
shmid=shmget((key_t)1122,1024,0666);
printf("Key of shared memory is %d\n",shmid);
shared_memory=shmat(shmid,NULL,0);
printf("Process attached at %p\n",shared_memory);
printf("Data read from the shared memory is : %s\n",(char*)shared_memory);
}
```

OUTPUT:

EXPERIMENT-3

i) Write a c program to solve producer-consumer problem using mutex and semaphore.

PRODUCER_CONSUMER ** For compiling->(gcc producer_consumer.c -pthread)

```
#include <pthread.h>

#include <semaphore.h>

#include <stdlib.h>

#include <stdio.h>

#define MaxItems 5

#define BufferSize 5

sem_t empty;

sem_t full;

int in = 0;

int out = 0;

int buffer[BufferSize];

pthread_mutex_t mutex;


void *producer(void *pno)

{

   int item;
```

```
    for(int i = 0; i < MaxItems; i++) {
        item = rand();
        sem_wait(&empty);
        pthread_mutex_lock(&mutex);
        buffer[in] = item;
        printf("Producer %d: Insert Item %d at %d\n", *((int *)pno),buffer[in],in);
        in = (in+1)%BufferSize;
        pthread_mutex_unlock(&mutex);
        sem_post(&full);
    }
}
void *consumer(void *cno)
{
    for(int i = 0; i < MaxItems; i++) {
        sem_wait(&full);
        pthread_mutex_lock(&mutex);
        int item = buffer[out];
        printf("Consumer %d: Remove Item %d from %d\n",*((int *)cno),item, out);
        out = (out+1)%BufferSize;
        pthread_mutex_unlock(&mutex);
        sem_post(&empty);
    }
}

int main()
{
```

```
pthread_t pro[5],con[5];
pthread_mutex_init(&mutex, NULL);
sem_init(&empty,0,BufferSize);
sem_init(&full,0,0);

int a[5] = {1,2,3,4,5};

for(int i = 0; i < 5; i++) {
    pthread_create(&pro[i], NULL, (void *)producer, (void *)&a[i]);
}
for(int i = 0; i < 5; i++) {
    pthread_create(&con[i], NULL, (void *)consumer, (void *)&a[i]);
}

for(int i = 0; i < 5; i++) {
    pthread_join(pro[i], NULL);
}
for(int i = 0; i < 5; i++) {
    pthread_join(con[i], NULL);
}

pthread_mutex_destroy(&mutex);
sem_destroy(&empty);
sem_destroy(&full);

return 0;
```

```
}
```

FIRST FIT

```
#include<stdio.h>

void main()
{
int bsize[10], psize[10], bno, pno, flags[10], allocation[10], i, j;

for(i = 0; i < 10; i++)
{
flags[i] = 0;
allocation[i] = -1;
}
printf("Enter no. of blocks: ");
scanf("%d", &bno);
printf("\nEnter size of each block: ");
for(i = 0; i < bno; i++)
scanf("%d", &bsize[i]);

printf("\nEnter no. of processes: ");
scanf("%d", &pno);
printf("\nEnter size of each process: ");
for(i = 0; i < pno; i++)
scanf("%d", &psize[i]);
for(i = 0; i < pno; i++)
for(j = 0; j < bno; j++)
if(flags[j] == 0 && bsize[j] >= psize[i])
{
allocation[j] = i;
flags[j] = 1;
break;
}
printf("\nBlock no.\tsize\t\tprocess no.\t\tsize");
for(i = 0; i < bno; i++)
{
printf("\n%d\t\t%d\t\t", i+1, bsize[i]);
if(flags[i] == 1)
printf("%d\t\t\t%d",allocation[i]+1,psize[allocation[i]]);
else
printf("Not allocated");
}
```

}

Output:

Enter the number of blocks: 3

Enter the size of each block: 12

7

4

Enter the number of processes: 3

Enter the size of each process: 7

4

9

Block no. Size Process No. Size

1 12 1 7

2 7 2 4

3 4 Not Allocated

BEST FIT

```
#include<stdio.h>
void main()
{

        int fragment[20],b[20],p[20],i,j,nb,np,tem,low=9999;
        static int barray[20],parray[20];
        printf("Memory Management Scheme - Best Fit");

        printf("Enter the number of processes:");
```

```
        scanf("%d",&np);

        printf("\nEnter the number of blocks:");
        scanf("%d",&nb);

        printf("\nEnter the size of the blocks:-\n");
        for(i=1;i<=nb;i++)
        {
                printf("Block no.%d:",i);
                scanf("%d",&b[i]);
        }

        printf("\nEnter the size of the processes :-\n");
        for(i=1;i<=np;i++)
{
   printf("Process no.%d:",i);

   scanf("%d",&p[i]);
}
        for(i=1;i<=np;i++)
        {
                for(j=1;j<=nb;j++)
                {
                        if(barray[j]!=1)
                        {
                                tem=b[j]-p[i];
                                if(tem>=0)
```

```
                                        if(low>tem)
                                        {
                                                parray[i]=j;
                                                low=tem;
                                        }
                                }
                        }
                        fragment[i]=low;
                        barray[parray[i]]=1;
                        low=10000;
                }

                 printf("\nProcess_number \tProcess_size\tBlock_number \tBlock_size\tFragment");
                for(i=1;i<=np && parray[i]!=0;i++)
                        printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d",i,p[i],parray[i],b[parray[i]],fragment[i]);
}
```

Output

Enter the number of blocks: 5

Enter the number of processes: 4

Enter the size of the blocks:

Block number 1: 10

Block number 2: 15

Block number 3: 5

Block number 4: 9?

Block number 5: 3

Enter the size if the process

Process number 1: 1

Process number 2: 4

Process number 3: 7

Process number 4: 12

Process number	Process size	Block number	Block size	Fragment
1	1	5	3	2
2	4	3	5	1
3	7	4	9	2
4	12	2	15	3

EXPERIMENT 5

Write a C program to simulate following memory management techniques.

a)Paging b)segmentation

a)Paging

```
#include<stdio.h>
void main()
{
int ms, ps, nop, np, rempages, i, j, x, y, pa, offset;
```

```
int s[10], fno[10][20];
printf("\nEnter the memory size -- ");
scanf("%d",&ms);
printf("\nEnter the page size -- ");
scanf("%d",&ps);
nop = ms/ps;
printf("\nThe no. of pages available in memory are -- %d ",nop);
printf("\nEnter number of processes -- ");
scanf("%d",&np);
rempages = nop;
for(i=1;i<=np;i++)
{
printf("\nEnter no. of pages required for p[%d]-- ",i);
scanf("%d",&s[i]);
if(s[i] >rempages)
{
printf("\nMemory is Full");
break;
}
rempages = rempages - s[i];
printf("\nEnter pagetable for p[%d] --- ",i);

for(j=0;j<s[i];j++)
scanf("%d",&fno[i][j]);
}
printf("\nEnter Logical Address to find Physical Address ");
printf("\nEnter process no. and pagenumber and offset -- ");
```

```
scanf("%d %d %d",&x,&y, &offset);
if(x>np || y>=s[i] || offset>=ps)
printf("\nInvalid Process or Page Number or offset");
else
{
pa=fno[x][y]*ps+offset;
printf("\nThe Physical Address is -- %d",pa);
}
}
```

INPUT

Enter the memory size -- 1000

Enter the page size -- 100

The no. of pages available in memory are -- 10

Enter number of processes -- 3

Enter no. of pages required for p[1]-- 4

Enter pagetable for p[1] --- 8 6 9 5

Enter no. of pages required for p[2]-- 5

Enter pagetable for p[2] --- 1 4 5 7 3

Enter no. of pages required for p[3]-- 5

OUTPUT

Memory is Full

Enter Logical Address to find Physical Address

Enter process no. and page number and offset -- 2 3 60

The Physical Address is – 760

b)Segmentation: For compiling → (gcc segmentation.c -lm)

```
#include <stdio.h>
#include <math.h>
int sost;
void gstinfo();
void ptladdr();
struct segtab
{
int sno;
int baddr;
int limit;
int val[10];
}st[10];
void gstinfo()
{
int i,j;
printf("\n\tEnter the size of the segment table: ");
scanf("%d",&sost);
for(i=1;i<=sost;i++)
{
printf("\n\tEnter the information about segment: %d",i);
st[i].sno = i;
printf("\n\tEnter the base Address: ");
scanf("%d",&st[i].baddr);
printf("\n\tEnter the Limit: ");
scanf("%d",&st[i].limit);
for(j=0;j<st[i].limit;j++)
{
printf("Enter the %d address Value: ",(st[i].baddr + j));
scanf("%d",&st[i].val[j]);
}
}
}
void ptladdr()
{
int i,swd,d=0,n,s,disp,paddr;
printf("\n\n\t\t\t SEGMENT TABLE \n\n");
printf("\n\t SEG.NO\tBASE ADDRESS\t LIMIT \n\n");
for(i=1;i<=sost;i++)
printf("\t\t%d \t\t%d\t\t%d\n\n",st[i].sno,st[i].baddr,st[i].limit);
printf("\n\nEnter the logical Address: ");
scanf("%d",&swd);
n=swd;
while (n != 0)
{
```

```
n=n/10;
d++;
}
s = swd/pow(10,d-1);
disp = swd%(int)pow(10,d-1);
if(s<=sost)


{
if(disp < st[s].limit)
{
paddr = st[s].baddr + disp;
printf("\n\t\tLogical Address is: %d",swd);
printf("\n\t\tMapped Physical address is: %d",paddr);
printf("\n\tThe value is: %d",( st[s].val[disp] ) );
}
else
printf("\n\t\tLimit of segment %d is high\n\n",s);
}

else
printf("\n\t\tInvalid Segment Address \n");
}
void main()
{
char ch;
gstinfo();
do
{
ptladdr();
printf("\n\t Do U want to Continue(Y/N)");
scanf("%c",&ch);
}while (ch == 'Y' || ch == 'y' );
}
```

OUTPUT

INPUT AND OUTPUT:

Enter the size of the segment table: 3

Enter the information about segment: 1

Enter the base Address: 4

Enter the Limit: 5

Enter the 4 address Value: 11

Enter the 5 address Value: 12

Enter the 6 address Value: 13

Enter the 7 address Value: 14

Enter the 8 address Value: 15

Enter the information about segment: 2

Enter the base Address: 5

Enter the Limit: 4

Enter the 5 address Value: 21

Enter the 6 address Value: 31

Enter the 7 address Value: 41

Enter the 8 address Value: 51

Enter the information about segment: 3

Enter the base Address: 3

Enter the Limit: 4

Enter the 3 address Value: 31

Enter the 4 address Value: 41

Enter the 5 address Value: 41

Enter the 6 address Value: 51

SEGMENT TABLE

SEG.NO BASE ADDRESS LIMIT

1 4 5

2 5 4

3 3 4

Enter the logical Address: 3

Logical Address is: 3

Mapped Physical address is: 3

The value is: 31

Do U want to Continue(Y/N)

SEGMENT TABLE

SEG.NO BASE ADDRESS LIMIT

1 4 5

2 5 4

3 3 4

Enter the logical Address: 1

Logical Address is: 1

Mapped Physical address is: 4

The value is: 11

Do U want to Continue(Y/N)

BANKERS ALGORITHM:

```
#include<stdio.h>

#include <stdbool.h>

bool check(int resources,int need[resources],int available[resources]);

void getSafeSequence(int processors,int resources,int allocated[processors][resources],int
max[processors][resources],int need[processors][resources],int *available);

void check_request(int process_number,int processors,int resources,int request[resources],int
allocated[processors][resources],int max[processors][resources],int need[processors][resources],int
available[resources]);

 int main(){

 int processors,resources;
```

```
 printf("Enter number of processors : "); scanf("%d",&processors);
 printf("Enter number of resources : "); scanf("%d",&resources);
 int allocated[processors][resources],max[processors][resources];
 printf("Enter Allocation Matrix\n");
 for(int i=0;i<processors;i++){
  for(int j=0;j<resources;j++){
    scanf("%d",&allocated[i][j]);
  }
 }
 printf("Enter Max Matrix\n");
 for(int i=0;i<processors;i++){
  for(int j=0;j<resources;j++){
    scanf("%d",&max[i][j]);
  }
 }
 int available[resources];
 for(int i=0;i<resources;i++){
  printf("\nEnter available of resource %c :",(i+65));
  scanf("%d",&available[i]);
 }
 printf("\nThe Number Of Instances Of Resource Present In The System Under Each Type Of
Resource are :\n");

 int instances[resources];
 for(int i=0;i<resources;i++){instances[i]=0;}
 for(int i=0;i<processors;i++){
  for(int j=0;j<resources;j++){
    instances[j]+=allocated[i][j];
```

```
  }
}
for(int i=0;i<resources;i++){instances[i]+=available[i];}
for(int i=0;i<resources;i++){printf("%c = %d\n",(i+65),instances[i]);}
printf("\nThe Need Matrix is \n");
int need[processors][resources];
for(int i=0;i<processors;i++){
  for(int j=0;j<resources;j++){
    need[i][j]=max[i][j]-allocated[i][j];
    printf("%d ",need[i][j]);
  }
  printf("\n");
}
int current_available[processors];
for(int i=0;i<processors;i++){current_available[i]=available[i];}
getSafeSequence(processors,resources,allocated,max,need,current_available);

printf("\n\nIf a request from process p1 arrives for (1,1,0,0), can the request be granted?");
int request1[4];
request1[0]=1; request1[1]=1;
int current_available1[processors];
for(int i=0;i<processors;i++){current_available1[i]=available[i];}
check_request(1,processors,resources,request1,allocated,max,need,current_available1);

printf("\n\nIf a request from process p4 arrives for (0,0,2,0), can the request be granted?\n");
int request2[4];
request2[2]=2;
```

```
  int current_available2[processors];
  for(int i=0;i<processors;i++){current_available2[i]=available[i];}
  check_request(4,processors,resources,request2,allocated,max,need,current_available2);
}
void getSafeSequence(int processors,int resources,int allocated[processors][resources],int
max[processors][resources],int need[processors][resources],int available[resources]){
  int computed=0;
  int computed_order[processors];
  int pointer_to_computed=0;
  bool processed[processors];
  for(int i=0;i<processors;i++){processed[i]=false;}
  while(computed<processors){
    bool any_process_computed=false;
    for(int i=0;i<processors;i++){
      if(processed[i]){continue;}
      if(check(resources,need[i],available)){
        for(int j=0;j<resources;j++){
          available[j]+=allocated[i][j];
        }
        processed[i]=true;
        any_process_computed=true;
        computed_order[pointer_to_computed++]=i;
        computed+=1;
      }
    }
    if(!any_process_computed){break;}
  }
  if(computed==processors){
```

```
   printf("\nThe System is in safe state and the safe sequence is :\n");
   for(int i=0;i<processors;i++){printf("P%d ",computed_order[i]);}
  }
  else{
   printf("The System is not in safe state and the processes will be in deadlock\n");
  }
}
bool check(int resources,int need[resources],int available[resources]){
  for(int i=0;i<resources;i++){
   if(need[i]>available[i]){return false;}
  }
  return true;
}
void check_request(int process_number,int processors,int resources,int request[resources],int
allocated[processors][resources],int max[processors][resources],int need[processors][resources],int
available[resources]){
  if(check(resources,request,available)){
   if(check(resources,request,need[process_number])){
     for(int i=0;i<resources;i++){
       available[i]-=request[i];
       allocated[process_number][i]+=request[i];
       need[process_number][i]=max[process_number][i]-allocated[process_number][i];
     }
     getSafeSequence(processors,resources,allocated,max,need,available);
     return;
   }
  }
  printf("Request cannot be granted\n");
```

}

OUTPUT

```
Enter number of processors :  5
Enter number of resources :  4
Enter Allocation Matrix

2 0 0 1
3 1 2 1
2 1 0 3
1 3 1 2
1 4 3 2

Enter Max Matrix

4 2 1 2
5 2 5 2
2 3 1 6
1 4 2 4
3 6 6 5

Enter available of resource A :3
Enter available of resource B :3
Enter available of resource C :2
Enter available of resource ❑ :1

The Number of Instances of Resource Present In The System Under Each Type of
Resource are

A = 12
B = 12
C = 8
D = 10

The Need Matrix is

2 2 1 1
2 1 3 1
0 2 1 3
0 1 1 2
2 2 3 3

The System is in safe state and the safe sequence is
P0 P3 P4 P1 P2

If a request from process p1 arrives for (1,1,0,0) can the request be granted?
The System is in safe state and the safe sequence is :

P0 P3 P4 P1 P2

If a request from process p4 arrives for (0,0,2,0) can the request be
granted? The System is not in safe state and the processes will be in deadlock
```