WEEK5

M.No	Date	IDX	PROGRAM	Sign & Marks
5		Deadlocks & Threads		
		5.1	Write a program to illustrate concurrent execution of threads using pthreads library	
		5.2	implement Bankers Algorithm for Dead Lock avoidance and prevention	
		5.3	Write a program to solve producer-consumer problem using Semaphores.	

5.1

AIM : Write a program to illustrate concurrent execution of threads using pthread library.

PROGRAM :

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

void *mythread1(void *vargp)
{
   int i;
   printf("thread1 Start \n");
   for(i=1;i<=10;i++)
      printf("from Thread1 i=%d\n",i);
   printf("exit from thread1\n");
   return NULL;
}

void *mythread2(void *vargp)
{
   int j;
   printf("thread2 Start\n");
   for(j=1;j<=10;j++)
      printf("From thread2 j=%d\n",j);
   printf("Exit from thread2\n");
   return NULL;
}
```

```
int main()
{
   pthread_t tid1, tid2;
   printf("before thread Start\n");
   pthread_create(&tid1, NULL, mythread1, NULL);
   pthread_create(&tid2, NULL, mythread2, NULL);
   pthread_join(tid1, NULL);
   pthread_join(tid2, NULL);
   return 0;
}
```

OUTPUT :

```
before thread Start
thread1 Start
from Thread1 i=1
thread2 Start
From thread2 j=1
From thread2 j=2
From thread2 j=3
From thread2 j=4
from Thread1 i=2
From thread2 j=5
from Thread1 i=3
From thread2 j=6
from Thread1 i=4
From thread2 j=7
from Thread1 i=5
From thread2 j=8
from Thread1 i=6
From thread2 j=9
from Thread1 i=7
From thread2 j=10
from Thread1 i=8
Exit from thread2
from Thread1 i=9
from Thread1 i=10
exit from thread1
```

5.2

AIM : lmplement Bankers Algorithm for Dead Lock avoidance and prevention

PROGRAM :

```
#include <stdio.h>

int main()
{
```

```
   int allocated[15][15], max[15][15], need[15][15], avail[15], tres[15],
work[15], flag[15];
   int pno, rno, i, j, prc, count, total;
   count = 0;
   printf("\n Enter number of processes: ");
   scanf("%d", &pno);
   printf("\n Enter number of resources: ");
   scanf("%d", &rno);
   for(i = 0; i < pno; i++)
      flag[i] = 0;
   printf("\n Enter Number of Instances of each resource: ");
   for(i = 0; i < rno; i++)
      scanf("%d", &tres[i]);
   printf("\n Enter Max resources required for each process: \n");
   for(i = 0; i < pno; i++)
   {
      printf(" For process %d : ", i);
      for(j = 0; j < rno; j++)
         scanf("%d", &max[i][j]);
   }


   printf("\n Enter allocated resources for each process: \n");
   for(i = 0; i < pno; i++)
   {
      printf("For process %d : ", i);
      for(j = 0; j < rno; j++)
         scanf("%d", &allocated[i][j]);
   }
   printf("\n Available resources : \n");
   for(j = 0; j < rno; j++)
   {
      total = 0;
```

```
    for(i = 0; i < pno; i++)
        total += allocated[i][j];
    avail[j] = tres[j] - total;
    work[j] = avail[j];
    printf(" %d \t", work[j]);
}
do
{
    for(i = 0; i < pno; i++)
        for(j = 0; j < rno; j++)
            need[i][j] = max[i][j] - allocated[i][j];
    printf("\n\n   Allocated\t\tMax\t\tNeed");
    for(i = 0; i < pno; i++)
    {
        printf("\n");
        for(j = 0; j < rno; j++)
            printf("%4d", allocated[i][j]);
        printf("\t");
        for(j = 0; j < rno; j++)
            printf("%4d", max[i][j]);
        printf("\t");
        for(j = 0; j < rno; j++)
            printf("%4d", need[i][j]);
    }
    prc = -1;
    for(i = 0; i < pno; i++)
    {
        if(flag[i] == 0)
        {
            prc = i;
            for(j = 0; j < rno; j++)
            {
                if(work[j] < need[i][j])
```

```
            {
               prc = -1;
               break;
            }
         }
      }
      if(prc != -1)
         break;
      }
   if(prc != -1)
   {
      printf("\n Process %d completed \n", prc);
      count++;
      for(j = 0; j < rno; j++)
      {
         work[j] += allocated[prc][j];
         allocated[prc][j] = 0;
         max[prc][j] = 0;
      }
      printf("\n Current available resources are :\t");
      for(j = 0; j < rno; j++)
            printf("%4d", work[j]);
   flag[prc] = 1;
   }
   }while(count != pno && prc != -1);
   printf("\n The system is %s \n", (count == pno) ? "in a safe state!" : "in
an unsafe state!");
   return 0;
}
```

OUTPUT :

```
 Enter number of processes: 5

 Enter number of resources: 3

 Enter Number of Instances of each resource: 10 5 7

 Enter Max resources required for each process:
 For process 0 : 7 5 3
 For process 1 : 3 2 2
 For process 2 : 9 0 2
 For process 3 : 2 2 2
 For process 4 : 4 3 3

 Enter allocated resources for each process:
For process 0 : 0 1 0
For process 1 : 3 0 2
For process 2 : 3 0 2
For process 3 : 2 1 1
For process 4 : 0 0 2

 Available resources :
 2       3       0
```

```
  Allocated            Max              Need
  0   1   0        7   5   3        7   4   3
  3   0   2        3   2   2        0   2   0
  3   0   2        9   0   2        6   0   0
  2   1   1        2   2   2        0   1   1
  0   0   2        4   3   3        4   3   1
Process 1 completed

Current available resources are :         5   3   2

  Allocated            Max              Need
  0   1   0        7   5   3        7   4   3
  0   0   0        0   0   0        0   0   0
  3   0   2        9   0   2        6   0   0
  2   1   1        2   2   2        0   1   1
  0   0   2        4   3   3        4   3   1
Process 3 completed

Current available resources are :         7   4   3

  Allocated            Max              Need
  0   1   0        7   5   3        7   4   3
  0   0   0        0   0   0        0   0   0
  3   0   2        9   0   2        6   0   0
  0   0   0        0   0   0        0   0   0
  0   0   2        4   3   3        4   3   1
Process 0 completed

Current available resources are :         7   5   3
```

```
Current available resources are :        7   5   3

  Allocated            Max              Need
  0   0   0        0   0   0        0   0   0
  0   0   0        0   0   0        0   0   0
  3   0   2        9   0   2        6   0   0
  0   0   0        0   0   0        0   0   0
  0   0   2        4   3   3        4   3   1
Process 2 completed

Current available resources are :       10   5   5

  Allocated            Max              Need
  0   0   0        0   0   0        0   0   0
  0   0   0        0   0   0        0   0   0
  0   0   0        0   0   0        0   0   0
  0   0   0        0   0   0        0   0   0
  0   0   2        4   3   3        4   3   1
Process 4 completed

Current available resources are :       10   5   7
The system is in a safe state!
```

5.3

AIM: C program for producer consumer problem with semaphores

PROGRAM

```
#include<sys/shm.h>
#include<stdio.h>
#include<sys/sem.h>
#include<unistd.h>
#include<stdlib.h>
struct sembuf op;
int pid,sid,shid,cid;
char *ptr,*cn;
void producer();
void consu();
void sem_wait(int x);
void sem_sig(int x);

int main()
{
	sid=semget((key_t)0x30,1,IPC_CREAT|0666);
	shid=shmget((key_t)0x40,10,IPC_CREAT|0666);
	cid=shmget((key_t)0x50,2,IPC_CREAT|0666);
	semctl(sid,0,SETVAL,1);
	ptr=((char *)shmat(shid,0,0));
	cn=((char *)shmat(cid,0,0));
```

```
        cn[1]=0;
        system("clear");
          pid=fork();
        if(pid==0)
        {
         printf("procedure start\n");
         producer();
         printf("producer ends\n");
        }
        else
        {
                printf("consumer starts\n");
                consu();
                printf("consumer ends\n");
        }
}
void producer()
{
        int i=0,j,k;
        for(i=0;i<5;i++)
        {
                sem_wait(sid);
                j=cn[1];
                if(j>=5)
                {
                sem_sig(sid);
                i--;
               printf("buffer is full\n");
                }
                else
                {
                        printf("\nProducing....%d",i+1);
                        ptr[i]=i;
```

```
                j++;
                cn[1]=j;
                sem_sig(sid);
            }
        }
}

void consu()
{
        int i,j,k,v;

 for(i=0;i<5;i++)
        {
          sem_wait(sid);
            j=cn[1];
            if(j<=0)
            {
                sem_sig(sid);
                i--;
            printf("buffer is empty\n");
            }
            else
            {
                v=ptr[i];
                j--;
                cn[1]=j;
                printf("\nConsumed....%d",v+1);
                sem_sig(sid);
            }
            sleep(1);
        }
}
```

```
void sem_wait(int x)
{
	op.sem_num=0;
	op.sem_op=-1;
	op.sem_flg=0;
	semop(x,&op,1);
}
void sem_sig(int x)
{
	op.sem_num=0;
	op.sem_op=1;
	op.sem_flg=0;
	semop(x,&op,1);
}
```

OUTPUT :