2024/06/20
Thrusday

Lab-7

1) Write a C program to simulate the concept of Banker's algorithm for the purpose of deadlock avoidance.

```
#include<stdio.h>
 int main()
 {
    //P0,P1,P2,P3,P4 are the process names here
    int n,m,i,j,k;
     n=5;
     m=3;
     int alloc[5][3]={{0,1,0}, //Allocation matrix
                      {2,0,0},
                      {3,0,2},
                      {2,1,1},
                      {0,0,2}};

     int max[5][3]{{7,5,3}, // max matrix
                   {3,2,2},
                   {9,0,2},
                   {2,2,2},
                   {4,3,3}};

     int avail[3]={3,3,2};
     int F[n],ans[n],ind=0;
     For(k=0;k<n;k++){
         F[k]=0;
     }
     int need[n][m];
     For(i=0;i<n;i++){
         For(j=0;j<m;j++)
             need[i][j]=max[i][j]-alloc[i][j];
      }
      int y=0;
      For(i=0;i<n;i++){
```

```
    For(i=0;i<n;i++){
       if(F[i]==0){
          int Flag=0;
          For(J=0; J<m;J++){
             if(need[i][J] > avail[J]){
                Flag=1;
                break;
          } }

       if (flag==0){
          ans[ind++]=i;
           For(y=0; y<m;y++)
               avail[y]+=alloc[i][y];
           F[i]=1;
  } } }
}
  int flag=1;
  For(int i=0;i<n;i++)
 {
  if(F[i]==0)
   {
    Flag=0;
      printf("The following system is not safe");
      break;
    }
  }
   if(flag==1)
  {
   printf("following is the SAFE Sequence\n");
   For(i=0; i<n-1;i++)
      printf("p%d->",ans[i]);
   printf("p%d",ans[n-1]);
  }
  return(0);
}
```

Output:

The following safe sequence P1->P2->P4->P0->P2

② WAP to simulate the concept of Dining-philosphers problem.

```
#include<pthead.h>
#include<semaphore.h>
# inclule<stdio.h>
# defineN5
# define THINKING 2
# define HUNGRy 1
# define EATING0
# define LEFT(phnum+4)%N
# define RIGHT(phnum+1)%N

 int state[N];
 int phil[N]={0,1,2,3,4};

 sem_t mutex;
 sem_t S[N];

 void test(int phnum)
 {
     if(state[phnum]=HUNGER
        && state[LEFT]!=EATING
        && state[RIGHT]!=EATING){
        state[phnum]=EATING;
         sleep(2);
        printf("philosopher %d takes Fork%d and %d\n",
                    phnum+1, LEFT+1, phnum+1);
       Printf("philosopher %d is Eating\n", phnum+1);
         sem_post(&S[phnum]);
     }
 }
 void take_fork(int phnum)
 {
    sem_wait(&mutex);
     state[phnum]=Hungry;
     Printf("Phiosopher %d is Hungry\n", phnum+1;
       test(phnum);
```

```
  sem_post(&mutex);
  Sem-wait(&s[phnum]);
   sleep(1);
 }
void put-Fork(int phnum)
 {
  sem-wait(&mutex);
  State[phnum] = Thinking;
  printf("Philosopher %d putting Fork %d and %d down\n",
        phnum+1, LEFT+1, phnum+1);
  printf("philospher %d is thinking\n", phnum+1);
   test(LEFT);
   test(RIGHT);
   Sem-post(&mutex);
 }
void* philosopher(void* num)
 { while(1){
    int* i = num;
    Sleep(1);
    take-Fork(*i);
    Sleep(0);
    Put-fork(*i);
   }
 }
int main()
{
 int i;
 pthread-t thread-id[N];
  sem-init(&mutex,0,1);
  For(i=0;i<N;i++)
    Sem-init(&S[i],0,0);
  For(i=0;i<N;i++){
    pthread-create(&thread-id[i],NULL,
             Philosopher,&phil[i]);
    printf("philosopher %d is thinking\n",i+1);
  }
```

```
    For(i=0;i<N;i++)
     Pthread-Join(thread-id[i],NULL);
}
```

Output:

```
Philosopher 1 is thinking
Philosopher 2 is thinking
Philosopher 3 is thinking
Philosopher 4 is thinking
Philosopher 5 is thinking

Philosopher 2 is Hungry
Philosopher 1 is Hungry.
Philosopher 3 takes fork 2 and 3.
 Philosopher 3 is thinking.
```

11) Write a C program to simulate deadlock detection.

```
#include <Stdio.h>
void main()
{ int n,m,i,j;
  printf("Enter the number of processes and number of
          types of resources:\n");
    Scanf("%d %d",&n,&m);
    int max[n][m], need[n][m], all[n][m], ava[m]
    Flag=1, Finish[n], dead[n], c=0;
    printf("Enter the maximum number of each
      types of resources needed by each process:\n");
     For(i=0;i<n;i++)
    { For(j=0;j<m;j++)
      { Scanf("%d",&max[i][j]);
      }
  }
   printf("Enter the allocated number of each types
    of resource needed by each process\n");
      For(i=0;i<n;i++)
      {
       For(j=0;j<m;j++)
        {
          Scanf("%d",&all[i][j]);
        }
    }
    printf("Enter the available number of each
       types of resource:\n");
       For(j=0;j<m;j++)
      { Scanf("%d",&ava[j]);
      }
      For(i=0;i<n;i++)
      {
```

```
     For(J=0; J<m; J++)
     {
       need[i][J]=maxir[i][J]-all[i][J];
     }
   }
   For(i=0;i<n;i++)
   {
     Finish[i]=0;
   }
   while(flag)
   {
      flag=0;
      for(i=0;i<n;i++)
      {
         c=0;
         for(J=0;J<m;J++)
         {
            if(finish[i]==0 && need[i][J] <=ava[J])
        {
          c++;
          if(c==m)
          {
             for(J=0;J<m;J++)
             {
                 ava[J]+=all[i][J];
                 finish[i]=1;
                 flag=1;
             }
             if(finish[i]==1)
             {
                i=n;
             }
          }
        }
      }
    }
   }
   J=0;
   Flag=0;
```

```
For(i=0;i<n;i++)
{
  if(finish[i]==0)
  {
    dead[j]=i;
     j++;
    flag=1;
   }
 }
 if(flag==1)
 {
    printf("Deadlock has occure:\n");
    printf("The Deadlock processes are:\n");
    for(i=0;i<n;i++)
    {
      printf("P%d",dead[i]);
    }
 }
 else
 printf("no deadlock has occured!\n");
}
```

output:-

```
Enter the no of processes and number of type
   of resources  5 4
Enter the maximum no. of each type of resource
  0012            Allocation        Available
  1756            0012              1100.
  2358            1420
  0652            1364
  0656            0532
                  0014
Deadlock occured
 The deadlock process are   P1. P2 P3 P4 P0 .
```