20/6/24

PAGE NO.:
DATE / /

1) Write a C program to stimulate the concept of Dining-Philosophers problem.

```
#include <pthread.h>
#include <semaphore.h>
#include <stdio.h>
#define N 4
#define THINKING 2
#define HUNGRY 1
#define EATING 0
#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]= = HUNGRY && 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);
            sempost(&S[phnum]);
    }
}
void take_fork (int phnum){
    sem_wait (&mutex);
    state [phnum] = HUNGRY;
    printf (" Philosopher %d is Hungry \n", phnum+1);
    test (phnum);
    sem_post (& mutex );
```

laxy F54 5G

PAGE NO.:
DATE / /

```
        sem_wait (&S[phnum]);
        sleep (1);
}
void  put_fork(int  phnum) {
        sem_wait (&mutex);
        state [phnum] = THINKING;
        printf (" Philosopher %d  putting  fork %d  & %d down
                \n", phnum+1 , LEFT+1 , phnum+1);
        printf (" Philosopher %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_work (*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 4 is Hungry
philosopher 3 is Hungry
philosopher 3 takes fork 3 and 3
philosopher 3 is Eating
philosopher 2 is Hungry
philosopher 1 is Hungry
philosopher 1 takes fork 1 and 1
philosopher 1 is Eating
philosopher 3 putting fork 3 and 3 down
philosopher 3 is thinking
philosopher 3 is Hungry
philosopher 3 takes fork 3 and 3
```

[illegible]
20/6/24

Galaxy F54 5G

PAGE NO.:

DATE / /

2) Write a C program to stimulate Bankers Algorithm for the purpose of deadlock avoidance.

```
→ #include <stdio.h>
int main(){
        int n, m, i, j, k;
        n=5;
        n=3;
        int alloc [5][3] = { {0,1,0}, {2,0,0}, {3,0,2},
                              {2,1,1}, {0,0,2}};
        int max [5][3] = { {7,5,3}, {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(k=0; k<5; k++) {
             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;
                         }
                     }
```

PAGE NO.:
DATE / /

```
                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 SAFE Sequence\n");
        for(i=0; i<n-1; i++){
                printf("P%d →", ans[i]);
        }
        printf("P%d", ans[n-1]);
    }
    return 0;
}
```

→ Output:-

```
Following is safe sequence
P1 → P3 → P4 → P0 → P2.
```

alaxy F54 5G

PAGE NO.:
DATE / /

37 Write a C program to stimulate deadlock detection

```
→ #include <stdio.h>
void main(){
    int n,m,i,j;
    printf("Enter the nor of processes and nor of types
          of resources:\n");
    scanf("%d %d",&n,&m);
    int max[n][m] ,need[n][m], all[n][m], ava[m],
    finish[n], dead[n];
    int flag=1,c;
    for(i=0;i<n;i++){
            finish[i]=0;
    }
    printf("Enter the maximum number of each type
      of resource 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 nor of each type of
          resource for each process:\n");
    for(i=0;i<n;i++){
        for(j=0;j<m;j++){
            scanf("%d",&~~max~~all[i][j]);
        }
    }
    printf("Enter the available nor of each type 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] = max [i][j] - all [i][j];
    }
}
 while (flag){
        flag=0;
        for (i=0 ; i<n; i++){
            if (finish [i]==0){
              c=0;
              for(j=0 ; j<m; j++){
                if (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;
           }
        }
    }
}
int deadlock =0;
for (i=0 ; i<n ; i++){
    if (finish [i] ==0){
        dead[deadlock ]= i;
        dealock ++;
    }
}
```

PAGE NO.:
DATE / /

```
if (deadlock >0){
        printf("Deadlock has occured\n");
        printf(" Deadlocked processes are:\n");
        for(i=0; i<deadlock ; i++){
            printf("P %d", dead[i]);
        }
        printf("\n");
    }
    else{
        printf(" No deadlock has occured! \n");
    }
}
```

→ Output:-

```
Enter nor of processes & nor of types of resources: 5 4
Enter maximum nor of each type of resource needed by
each process:
5 1 1 7  3 2 1 1  3 3 2 1  4 6 1 2
6 3 2 5
Enter the allocated nor of each type of resource for
each process:
3 0 1 4  2 2 1 0  3 1 2 1  0 5 1 0
4 2 1 2
Enter the available nor of each type of resource:
0 3 0 1
Deadlock has occured:
The deadlocked processes are:
P0 P4
```

20/6/24