20/6/24

Write a C program to simulate the concept of Dining-Philosophers problem

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

#define N 5
#define THINKING 2
#define HUNGRY 1
#define EATING 0
#define LEFT (phnum+4)%N
#define RIGHT (phum+1)%N

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

sem_t mutex;
sem_t s[N];

void test (int phum)
{
    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);
        sem_post(&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);
    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("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_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 5 is Hungry
Philosopher 4 is Hungry
Philosopher 3 is Hungry
Philosopher 1 is Hungry
Philosopher 2 is Hungry
Philosopher 2 takes fork 1 and 2
Philosopher 2 is Eating
Philosopher 2 putting fork 1 and 2 down
Philosopher 2 is thinking
Philosopher 1 takes fork 5 and 1
```

Sur 2016/24

Write a C program to simulate Bankers algorithm for the purpose of deadlock avoidance

```
#include <stdio.h>
int main()
{
    int n, m, i, j, k;
    n = 5;
    m = 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++)
```

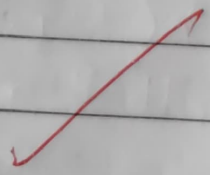
```
{
    for(i=0 ; i<n ; i++)
    {
        if (f[i] == 0) {
            int flag = 0;
            for(j=0 ; j<m ; j++)
            {
                if(need[i][j] > avail[j]){
                    flag = 1;
                    break;
                }
            }
            if (flag == 0)
                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 :-

Following is the SAFE Sequence

P1 → P3 → P4 → P0 → P2



SURYA Gold

Date Page

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],
        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 number of each type of
    resource for each process:\n");
for(i=0;i<n;i++){
  for(j=0;j<m;j++){
    scanf("%d",&all[i][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;
            deadlock++;
        }
    }

    if(deadlock>0){
        printf("Deadlock has occured:\n");
        printf("The 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 the number of processes and number of types of resources: 5 4

Enter the maximum number 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 number 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 number of each types of resource:

0 3 0 1

Deadlock has occured:

The deadlocked processes are:

P0 P4