WEEK 6

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

1.1.1 Code:

```
#include <stdio.h>

#include <pthread.h>

#include <semaphore.h>


#define N 5

#define THINKING 2

#define HUNGRY 1

#define EATING 0

#define LEFT (num_of_philosopher + 4) % N

#define RIGHT (num_of_philosopher + 1) % N


int state[N];

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


sem_t mutex;

sem_t S[N];


void test(int num_of_philosopher)
```

```
{
        if (state[num_of_philosopher] == HUNGRY && state[LEFT] !=
EATING && state[RIGHT] != EATING)
        {
            state[num_of_philosopher] = EATING;

            sleep(2);

            printf("Philosopher %d takes fork %d and %d\n",
num_of_philosopher +1, LEFT +1, num_of_philosopher +1);

            printf("Philosopher %d is Eating\n", num_of_philosopher +1);

            sem_post(&S[num_of_philosopher]);
    }
}

void take_fork(int num_of_philosopher)
{
    sem_wait(&mutex);
    state[num_of_philosopher] = HUNGRY;
    printf("Philosopher %d is Hungry\n", num_of_philosopher +1);
```

```
        test(num_of_philosopher);


        sem_post(&mutex);
        sem_wait(&S[num_of_philosopher]);
        sleep(1);
}


void put_fork(int num_of_philosopher)
{
        sem_wait(&mutex);
        state[num_of_philosopher] = THINKING;


        printf("Philosopher %d putting fork %d and %d
down\n",num_of_philosopher +1, LEFT +1, num_of_philosopher +1);


        printf("Philosopher %d is thinking\n", num_of_philosopher +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 1 is Hungry
Philosopher 5 is Hungry
Philosopher 4 is Hungry
Philosopher 3 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
Philosopher 1 is Eating
Philosopher 3 takes fork 2 and 3
Philosopher 3 is Eating
Philosopher 1 putting fork 5 and 1 down
Philosopher 1 is thinking
Philosopher 5 takes fork 4 and 5
Philosopher 5 is Eating
Philosopher 2 is Hungry
Philosopher 3 putting fork 2 and 3 down
Philosopher 3 is thinking
Philosopher 2 takes fork 1 and 2
Philosopher 2 is Eating
Philosopher 1 is Hungry
Philosopher 5 putting fork 4 and 5 down
Philosopher 5 is thinking
Philosopher 4 takes fork 3 and 4
Philosopher 4 is Eating
Philosopher 3 is Hungry
Philosopher 2 putting fork 1 and 2 down
Philosopher 2 is thinking
Philosopher 1 takes fork 5 and 1
Philosopher 1 is Eating
```

25/7/23

```
#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}
                   {2, 1, 1}
                   {0, 0, 2}};
  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-i]);
}
return(0);
}
```

Output

Enter no of process & resources : 4 3
Enter claim matrix
3 2 2
6 1 3
3 1 4
4 2 2

Enter the allocation matrix
1 0 0
6 1 2
2 1 1
0 0 2

Resource vector 9 3 6
All the resources can be allocated to process 2
Available resource are : 6 2 3
Process executed : Y
All resources can be allocated to process 3
Available resource are : 8 3 4
Process 3 executed ? Y
All the resources can be allocated to process 1
Available resource are : 9 3 6
Process 1 executed : Y
System is in safe mode the given state is safe sta

10/10

21/5/23