19/7/23

Write a C-Program to Simulate the concept of Dining-Philosophers.

```
#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 (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 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 & %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)
Pf ("P %d is thinking \n", i+1);
  }
for (i=0; i<N; i++)
  pthread_joint (threadid[i], NULL);
  }
```

output.

P1 is thinking
P2 is thinking
P3 is thinking.
P4 is HUNGRY
P2 is Hungry
P5 is Hungry.
P3 takes fork 2 & 3
P3 is Eating
P1 is Hungry
P5 is Hungry.
P5 takes fork 4 & 5.

=> Producer-Consumer.

```
#include <stdio.h>
#include< stdlib.h>
int mutex = 1, full = 0, Empty = 3, x = 0;
int main()
{
  int n;
  void Producer();
  void Consumer();
  int wait(int);
  int Signal(int);
  Pf("\n1. Producer\n2. Consumer\n3. Exit");
  while(1)
{
  Pf("\n Enter your Choice:");
    Sf("%d", &n);
    Switch(n)
    {
  Case 1: if((mutex == 1) && (Empty != 0))
            Producer();
    else
     Printf("Buffer is full!!");
       break;
```

```
case 2: if ((mutex == 1) && (full != 0))
            consumer ();
        else
        Printf ("Buffer is Empty !!");
            break;

 case 3:
    Exit (0);
    break;

  }
 }
 return 0;
}
int wait (int s)
{
 return (--s);
}
int signal (int s)
{
  return (++s);
}
void Producer ()
{
  mutex = wait (mutex);
  full = Signal (full);
  Empty = wait (Empty);
     x++
Pf ("\n Producer Produces the item %d", x);
   mutex = Signal (mutex);
}
```

```
void Consumer()
{
    mutex = wait(mutex);
    full = wait(full);
    Empty = signal(Empty);
    Pf("\n Consumer consumes item-%d", x);
    x--;
    mutex = signal(mutex);
}
```

Output

1. Producer
2. Consumer
3. Exit

Enter your choice: 1

Producer Produces the item 1

Enter your choice: 2

Producer Produces the item 2

Enter your choice: 1

Producer Produces the item 3.

Enter your choice: 2.

Consumer Consumes item 3

Enter your choice: 2

Buffer is Empty !!