|  |  |
| --- | --- |
| **Name** | Manish Shashikant Jadhav |
| **UID no.** | 2023301005 |

|  |  |
| --- | --- |
| **Experiment 7** | |
| **AIM :** | C program implementing the solution to the Dining Philosopher Problem. The Dining Philosopher Problem states that there are five philosophers which do two thinks: think and eat. They share a table having a chair for each one of them. In the center of the table there is a bowl of rice and the table is laid with 5 single chopsticks .  When a philosopher thinks, he does not interact with others. When he gets hungry, he tries to pick up the two chopsticks that are near to him. For example, philosopher 1 will try to pick chopsticks 1 and 2. But the philosopher can pickup only one chopstick at a time. He can not take a chopstick that is already in the hands of his neighbor. The philosopher stars to eat when he has both his chopsticks in his hand. After eating the philosopher puts down both the chopsticks and starts to think again. |
| **Discussion & Output:** | **Program:**  #include <stdio.h>  #include <stdlib.h>  #include <pthread.h>  #include <semaphore.h>  #define num\_philopsophers 5  #define num\_chopsticks 5  void dine(int *n*);  pthread\_t philosopher[num\_philopsophers];  pthread\_mutex\_t chopstick[num\_chopsticks];  int main()  {  *// Define msg and status\_message*      int status\_message;      void \*msg;    *// Initialise the semaphore array*      for (int i = 1; i <= num\_chopsticks; i++)      {          status\_message = pthread\_mutex\_init(&chopstick[i], NULL);    *// Checking if the mutex was initialised successfully*          if (status\_message == -1)          {              printf("\n Mutex initialization failed");              exit(1);          }      }    *// Run the philosopher Threads using \*dine() function*      for (int i = 1; i <= num\_philopsophers; i++)      {          status\_message = pthread\_create(&philosopher[i], NULL, (void \*)dine, (int \*)i);          if (status\_message != 0)          {              printf("\n Thread creation error \n");              exit(1);          }      }  *// Wait for all philosophers threads to complete executing*  *// (finish dining) before closing the program*      for (int i = 1; i <= num\_philopsophers; i++)      {          status\_message = pthread\_join(philosopher[i], &msg);          if (status\_message != 0)          {              printf("\n Thread join failed \n");              exit(1);          }      }    *// Destroy the chopstick Mutex array*      for (int i = 1; i <= num\_chopsticks; i++)      {          status\_message = pthread\_mutex\_destroy(&chopstick[i]);          if (status\_message != 0)          {              printf("\n Mutex Destroyed \n");              exit(1);          }      }      return 0;  }  *// dine method*  void dine(int *n*)  {      printf("\nPhilosopher % d is thinking ", n);    *// picking up the left chopstick (wait)*      pthread\_mutex\_lock(&chopstick[n]);    *// picking up the right chopstick (wait)*      pthread\_mutex\_lock(&chopstick[(n + 1) % num\_chopsticks]);    *// both chopstick picked now starts eating*        printf("\nPhilosopher % d is eating ", n);      sleep(3);    *// places the left chopstick down (signal)*      pthread\_mutex\_unlock(&chopstick[n]);    *// places the  the right chopstick down (signal)*      pthread\_mutex\_unlock(&chopstick[(n + 1) % num\_chopsticks]);    *//  eating finishes*      printf("\nPhilosopher % d Finished eating ", n);  }  **Output:** |
| **CONCLUSION:** | Hence, by completing this experiment I came to know about implementing the solution to the Dining Philosopher Problem. |