**IT-253: Operating Systems Laboratory Handout-Answers-lab-5,6**

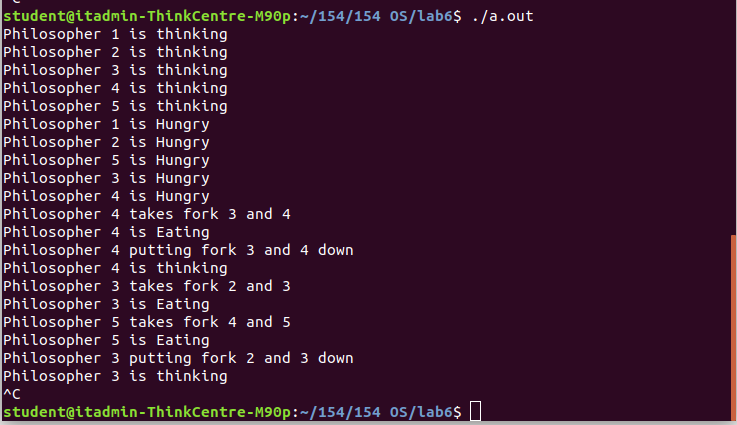
**Ashok Bhobhiya(181IT154)**

***Exercise 1​*** : ​ There are five philosophers, each identified by a number 0 to 4. Each philosopher enters into the three following states: ​ THINKING, HUNGRY and EATING and ​ will run each process as a separate thread. Use two semaphores in C (using Pthread library): ​ Mutex and a Semaphore array for the philosophers where, Mutex is used such that no two philosophers may access the pickup or put down at the same time and semaphore array is used to control the behavior of each philosopher.

***Ans-***

|  |
| --- |
| #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 that 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]) has no effect  // during takefork  // used to wake up hungry philosophers  // during putfork  sem\_post(&S[phnum]);  } }   // take up chopsticks void take\_fork(int phnum) {    sem\_wait(&mutex);    // state that hungry  state[phnum] = HUNGRY;    printf("Philosopher %d is Hungry\n", phnum + 1);    // eat if neighbours are not eating  test(phnum);    sem\_post(&mutex);    // if unable to eat wait to be signalled  sem\_wait(&S[phnum]);    sleep(1); }   // put down chopsticks void put\_fork(int phnum) {    sem\_wait(&mutex);    // state that thinking  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\* philospher(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];    // initialize the semaphores  sem\_init(&mutex, 0, 1);    for (i = 0; i < N; i++)    sem\_init(&S[i], 0, 0);    for (i = 0; i < N; i++) {    // create philosopher processes  pthread\_create(&thread\_id[i], NULL,  philospher, &phil[i]);    printf("Philosopher %d is thinking\n", i + 1);  }    for (i = 0; i < N; i++)    pthread\_join(thread\_id[i], NULL); } |

Screenshots-

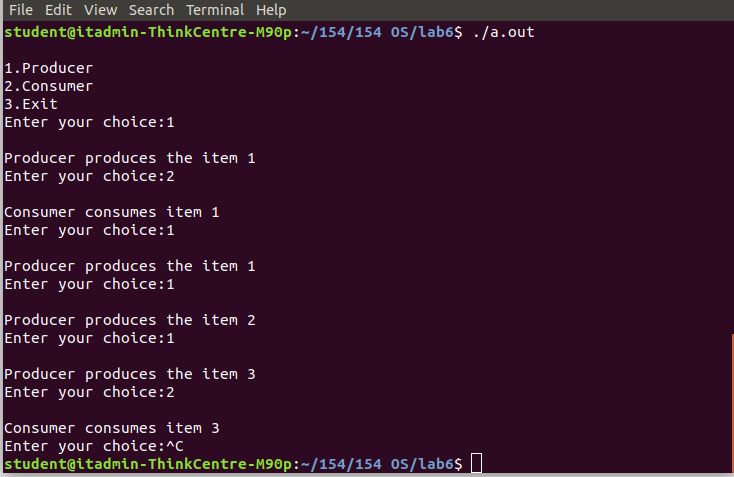


***Exercise 2***​ : Demonstrate how process synchronization problem in producer consumer problem is solved with semaphores.

***Ans-***

|  |
| --- |
| #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);  printf("\n1.Producer\n2.Consumer\n3.Exit");  while(1)  {  printf("\nEnter your choice:");  scanf("%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++;  printf("\nProducer produces the item %d",x);  mutex=signal(mutex); }   void consumer() {  mutex=wait(mutex);  full=wait(full);  empty=signal(empty);  printf("\nConsumer consumes item %d",x);  x--;  mutex=signal(mutex); } |

***Screenshots-***



***Exercise 3​*** : ​ There is a shared resource which should be accessed by multiple processes. There are two types of processes in the system namely, ​ reader​ and ​ writer​ . Any number of ​ readers​ can read from the shared resource simultaneously, but only one ​ writer​ can write to the shared resource. The same way, when a ​ writer ​ is writing data to the resource, no other process can access the resource. A ​ writer​ cannot write

to the resource if there are non-zero number of readers accessing the resource at that time. From the above problem statement, it is evident that readers have higher priority than writers. If a writer wants to write to the resource, it must wait until there are no readers currently accessing that resource. So, based on this find an appropriate solution to the problem given using Monitors as well as Semaphore.

***Ans-***

|  |
| --- |
| #include<stdio.h> #include<pthread.h> #include<semaphore.h>  sem\_t mutex,writeblock; int data = 0,rcount = 0;   // It is first writers priority. Once the writer has written something then // only any reader can read. void \*reader(void \*arg) {  int f;  f = ((int)arg);  sem\_wait(&mutex);  rcount = rcount + 1;  if(rcount==1)  sem\_wait(&writeblock);  sem\_post(&mutex);  printf("Data read by the reader%d is %d\n",f,data);  sleep(1);  sem\_wait(&mutex);  rcount = rcount - 1;  if(rcount==0)  sem\_post(&writeblock);  sem\_post(&mutex); }  void \*writer(void \*arg) {  int f;  f = ((int) arg);  sem\_wait(&writeblock);  data++;  printf("Data writen by the writer%d is %d\n",f,data);  sleep(1);  sem\_post(&writeblock); }  int main() {  int i,b;  pthread\_t rtid[5],wtid[5];  sem\_init(&mutex,0,1);  sem\_init(&writeblock,0,1);  for(i=0;i<=2;i++)  {  pthread\_create(&wtid[i],NULL,writer,(void \*)i);  pthread\_create(&rtid[i],NULL,reader,(void \*)i);  }  for(i=0;i<=2;i++)  {  pthread\_join(wtid[i],NULL);  pthread\_join(rtid[i],NULL);  }  return 0; } |

***Screenshots-***