 ***DEPARTMENT OF COMPUTER ENGINEERING***

Experiment No.

|  |  |
| --- | --- |
| Semester | S.E. Semester IV – Computer Engineering |
| Subject | Operating System |
| Subject Professor In-charge | Snehal Andhare |
| Assisting Teachers | Ms. Rasika Ransing |
| Laboratory | M310B – Computer Engineering Laboratory |

|  |  |  |
| --- | --- | --- |
| Student Name | Chinmay Tiwari | |
| Roll Number | 18102A0066 | |
| Grade and Subject Teacher’s Signature |  |  |

|  |  |  |
| --- | --- | --- |
| Experiment Number |  | |
| Experiment Title | PBL 2 | |
| Resources / Apparatus Required | Hardware:  Computer | Software:  Compiler |
| Objectives  (Skill Set / Knowledge Tested / Imparted) | Implementation of Producer Consumer Problem using Semaphores and p\_threads in C | |
| Theory: | Producer–consumer problem (also known as the bounded-buffer problem) is a classic example of a multi-process synchronization problem. The problem describes two processes, the producer and the consumer, who share a common, fixed-size buffer used as a queue. The producer's job is to generate data, put it into the buffer, and start again. At the same time, the consumer is consuming the data (i.e., removing it from the buffer), one piece at a time. The problem is to make sure that the producer won't try to add data into the buffer if it's full and that the consumer won't try to remove data from an empty buffer.  The solution for the producer is to either go to sleep or discard data if the buffer is full. The next time the consumer removes an item from the buffer, it notifies the producer, who starts to fill the buffer again. In the same way, the consumer can go to sleep if it finds the buffer empty. The next time the producer puts data into the buffer, it wakes up the sleeping consumer. The solution can be reached by means of inter-process communication, typically using semaphores. An inadequate solution could result in a deadlock where both processes are waiting to be awakened. | |
| Code: | #include <stdio.h>  #include <pthread.h>  #include <semaphore.h>  #include <stdlib.h>  #include <signal.h>  #include <fcntl.h>  #define MAX\_THREADS 5  #define BUFFER\_SIZE 10  sem\_t \*empty, \*full, \*mutex;  int buffer[BUFFER\_SIZE];  int in = 0, out = 0;  static volatile int keepRunning = 1;  void intHandler(int dummy) {  keepRunning = 0;  }  void \*producer(void \* id\_ptr) {  int ID = \*((int \*) id\_ptr);  static int nextProduced = 0;  while (keepRunning) {  (void) sem\_wait(empty);  (void) sem\_wait(mutex);  if (buffer[in] != -1) {  fprintf(stderr, "Synchronization Error: Producer %d Just overwrote %d from Slot %d\n", ID, buffer[in], in);  exit(1);  }  nextProduced++;  buffer[in] = nextProduced;  printf("Producer %d. Put %d in slot %d\n", ID, nextProduced, in);  in = (in + 1) % BUFFER\_SIZE;  printf("incremented in!\n");  (void) sem\_post(mutex);  (void) sem\_post(full);  }  return NULL;  }  void \*consumer (void \*id\_ptr) {  int ID = \*((int \*) id\_ptr);  static int nextConsumed = 0;  while (keepRunning) {  (void) sem\_wait(full);  (void) sem\_wait(mutex);  nextConsumed = buffer[out];  if (nextConsumed == -1) {  fprintf(stderr, "Synch Error: Consumer %d Just Read from empty slot %d\n", ID, out);  exit(1);  }  printf("Consumer %d Just consumed item %d from slot %d\n", ID, nextConsumed, out);  buffer[out] = -1;  out = (out + 1) % BUFFER\_SIZE;  printf("incremented out!\n");  (void) sem\_post(mutex);  (void) sem\_post(empty);  }  return NULL;  }  int main() {  int ID[MAX\_THREADS];  pthread\_t TID[MAX\_THREADS];  empty = sem\_open("/empty", O\_CREAT, 0644, 10);  full = sem\_open("/full", O\_CREAT, 0644, 0);  mutex = sem\_open("/mutex", O\_CREAT, 0644, 1);  signal(SIGINT, intHandler);  for (int i = 0; i < MAX\_THREADS; i++) {  ID[i] = i;  }  for (int i = 0; i < BUFFER\_SIZE; i++) {  buffer[i] = -1;  }  pthread\_create(&TID[0], NULL, producer, (void \*) &ID[0]);  printf("Producer ID = %d created!\n", 0);  pthread\_create(&TID[1], NULL, consumer, (void \*) &ID[1]);  printf("Consumer ID = %d created!\n", 1);  pthread\_create(&TID[2], NULL, producer, (void \*) &ID[2]);  printf("Producer ID = %d created!\n", 2);  pthread\_create(&TID[3], NULL, consumer, (void \*) &ID[3]);  printf("Consumer ID = %d created!\n", 3);  for (int i = 0; i < 4; i++) {  pthread\_join(TID[i], NULL);  }  (void) sem\_unlink("/empty");  (void) sem\_unlink("/full");  (void) sem\_unlink("/mutex");  return 0;  } | |
| Output: |  | |