National Textile University

Department of Computer Science

Subject

Operating System

Submitted to:

Sir Nasir Mehmood

Submitted by:

Kashmir Jamshaid

Registration Number

23-NTU-CS-1167

Lab No.

08

Semester

5th

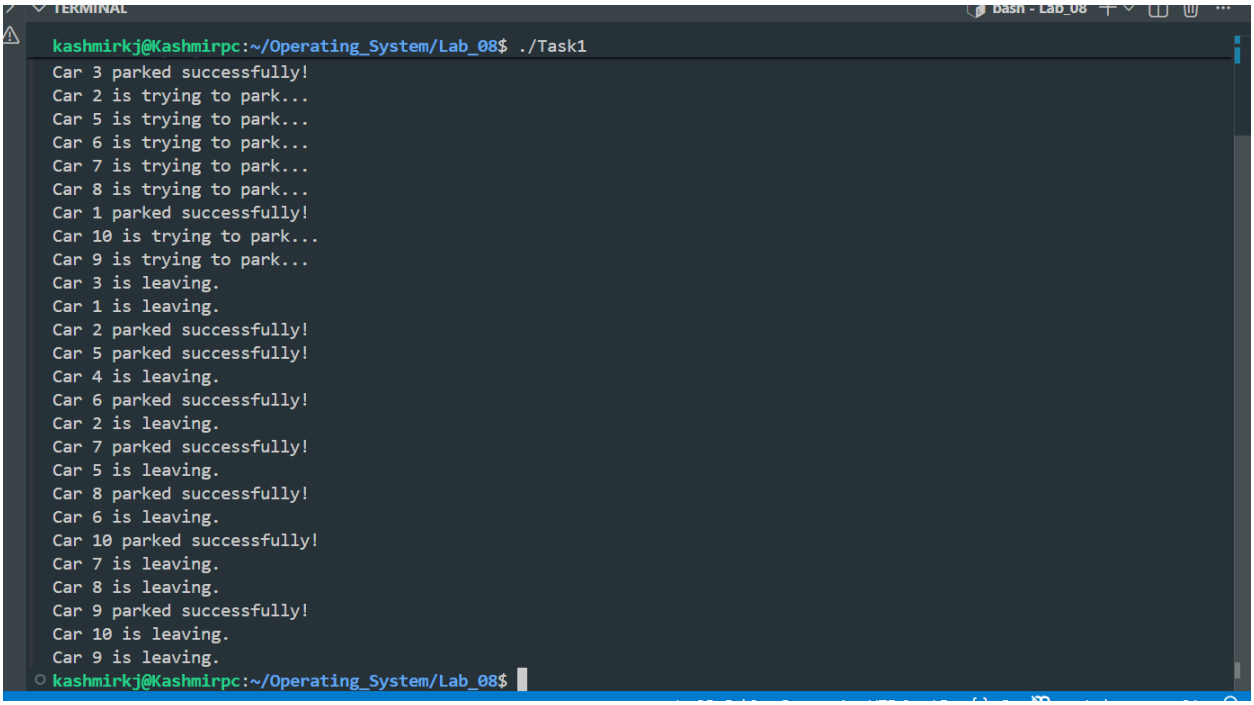
Task1:

Code:

Input:

```c
//  Problem Statement:
//  Simulate a parking lot with N parking spaces. Multiple cars (threads) try to park. If the lot is
//  full, cars must wait until a space becomes available
 #include <stdio.h>
 #include <pthread.h>
 #include <semaphore.h>
 #include <unistd.h>
 sem_t parking_spaces;
 void* car(void* arg) {
 int id = *(int*)arg;
 printf("Car %d is trying to park...\n", id);
 sem_wait(&parking_spaces); // Try to get a space
 printf("Car %d parked successfully!\n", id);
 sleep(2); // Stay parked for 2 seconds
 printf("Car %d is leaving.\n", id);
 sem_post(&parking_spaces); // Free the space
 return NULL;
 }
 int main() {
 pthread_t cars[10];
 int ids[10];
 // Initialize: 3 parking spaces available
 sem_init(&parking_spaces, 0, 3);
 // Create 10 cars (more than spaces!)
 for(int i = 0; i < 10; i++) {
 ids[i] = i + 1;
 pthread_create(&cars[i], NULL, car, &ids[i]);
 }
 // Wait for all cars
 for(int i = 0; i < 10; i++) {
 pthread_join(cars[i], NULL);
 }
 sem_destroy(&parking_spaces);
 return 0;
}
```

Output:

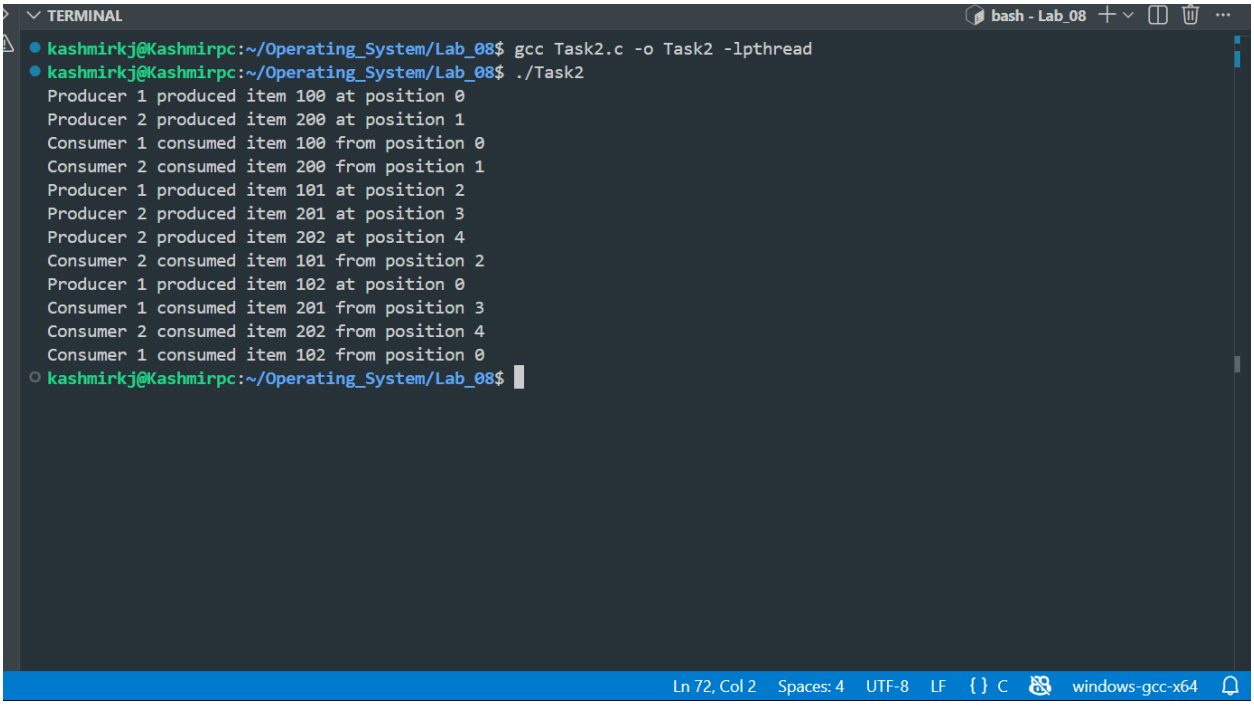


Task2:

Intput:

```c
#include <stdio.h>
#include <pthread.h>
#include <semaphore.h>
#include <unistd.h>
#define BUFFER_SIZE 5
int buffer[BUFFER_SIZE];
int in = 0; // Producer index
int out = 0; // Consumer index
sem_t empty; // Counts empty slots
sem_t full; // Counts full slots
pthread_mutex_t mutex;
void* producer(void* arg) {
int id = *(int*)arg;
for(int i = 0; i < 3; i++) { // Each producer makes 3 items
int item = id * 100 + i;
// TODO: Wait for empty slot
sem_wait(&empty);
// TODO: Lock the buffer
pthread_mutex_lock(&mutex);
// Add item to buffer
buffer[in] = item;
printf("Producer %d produced item %d at position %d\n",
id, item, in);
in = (in + 1) % BUFFER_SIZE;
// TODO: Unlock the buffer
pthread_mutex_unlock(&mutex);
// TODO: Signal that buffer has a full slot
sem_post(&full);
sleep(1);
}
return NULL;
}
void* consumer(void* arg) {
int id = *(int*)arg;
for(int i = 0; i < 3; i++) {
// TODO: Students complete this similar to producer
sem_wait(&full);
pthread_mutex_lock(&mutex);
int item = buffer[out];
printf("Consumer %d consumed item %d from position %d\n",
id, item, out);
out = (out + 1) % BUFFER_SIZE;
pthread_mutex_unlock(&mutex);
sem_post(&empty);
sleep(2); // Consumers are slower
}
return NULL;
}
int main() {
pthread_t prod[2], cons[2];
int ids[2] = {1, 2};
// Initialize semaphores
sem_init(&empty, 0, BUFFER_SIZE); // All slots empty initially
sem_init(&full, 0, 0);
pthread_mutex_init(&mutex, NULL);
// No slots full initially
// Create producers and consumers
for(int i = 0; i < 2; i++) {
pthread_create(&prod[i], NULL, producer, &ids[i]);
pthread_create(&cons[i], NULL, consumer, &ids[i]);
}
// Wait for completion
for(int i = 0; i < 2; i++) {
pthread_join(prod[i], NULL);
pthread_join(cons[i], NULL);
}
// Cleanup
sem_destroy(&empty);
sem_destroy(&full);
pthread_mutex_destroy(&mutex);
return 0;
}
```

Output:



Description:

- **The Buffer size initialized i.e. 5 and declared globally.**
- **The program has a shared buffer where producers put items means produce items and consumers take them out means consumes the items.**
- **Two counters, empty and full keep checking/ Tracking of how many spaces are empty or filled .**
- **A lock makes sure only one thread touches the buffer at a time.**
- **Producers wait if the buffer is full, then put an item inside.**
- **Consumers wait if the buffer is empty, then take an item out.**
- **Two producers and two consumers work at the same time without crashing into each other.**

Questions That Ask By Sir :

item = id * 100 + i

Each producer generates 3 items: for i = 1, 2, 3.

When id = 1

Items:

- **1 × 100 + 1 = 101**
- **1 × 100 + 2 = 102**
- **1 × 100 + 3 = 103**

When id = 2:

Items:

- **2 × 100 + 1 = 201**
- **2 × 100 + 2 = 202**
- **2 × 100 + 3 = 203**

Total items produced = 6 items

Values = 101, 102, 103, 201, 202, 203

Buffer and Blocking Explanation

- **There are 5 buffers.**
- **Producers attempt to produce 6 items.**
- **The first 5 items fill the buffer.**

- **When the 6th item is produced, the producer blocks/waits because the buffer is full.**

Consumer Situation

- **Consumers expect 8 items,**

but producers generate only 6 items.

- **Since two items never get produced,**

and producers are blocked waiting for buffer space that never frees, this results in a deadlock.

Producer :

```
wait (&empty)              // Decrement Empty
mutex_lock(&mutex)         // lock mutex
// critical section
mutex_unlock(&mutex)       // unlock section
post (&full)               // Increment full.
```

Consumer :

```
wait (&full)               // Decrement full
mutex_lock(&mutex)         // lock section below
// critical section
mutex_unlock(&mutex)       // unlock section
post (&empty).             // Increment empty.
```