



# 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:

```
● ● ●
1 // Problem Statement:
2 // Simulate a parking lot with N parking spaces. Multiple cars (threads) try to park. If the lot is
3 // full, cars must wait until a space becomes available
4 #include <stdio.h>
5 #include <pthread.h>
6 #include <semaphore.h>
7 #include <unistd.h>
8 sem_t parking_spaces;
9 void* car(void* arg) {
10 int id = *(int*)arg;
11 printf("Car %d is trying to park...\n", id);
12 sem_wait(&parking_spaces); // Try to get a space
13 printf("Car %d parked successfully!\n", id);
14 sleep(2); // Stay parked for 2 seconds
15 printf("Car %d is leaving.\n", id);
16 sem_post(&parking_spaces); // Free the space
17 return NULL;
18 }
19 int main() {
20 pthread_t cars[10];
21 int ids[10];
22 // Initialize: 3 parking spaces available
23 sem_init(&parking_spaces, 0, 3);
24 // Create 10 cars (more than spaces!)
25 for(int i = 0; i < 10; i++) {
26 ids[i] = i + 1;
27 pthread_create(&cars[i], NULL, car, &ids[i]);
28 }
29 // Wait for all cars
30 for(int i = 0; i < 10; i++) {
31 pthread_join(cars[i], NULL);
32 }
33 sem_destroy(&parking_spaces);
34 return 0;
35 }
```

### Output:

```
TERMINAL
⚠ kashmirkj@Kashmirpc:~/Operating_System/Lab_08$ ./Task1
Car 3 parked successfully!
Car 2 is trying to park...
Car 5 is trying to park...
Car 6 is trying to park...
Car 7 is trying to park...
Car 8 is trying to park...
Car 1 parked successfully!
Car 10 is trying to park...
Car 9 is trying to park...
Car 3 is leaving.
Car 1 is leaving.
Car 2 parked successfully!
Car 5 parked successfully!
Car 4 is leaving.
Car 6 parked successfully!
Car 2 is leaving.
Car 7 parked successfully!
Car 5 is leaving.
Car 8 parked successfully!
Car 6 is leaving.
Car 10 parked successfully!
Car 7 is leaving.
Car 8 is leaving.
Car 9 parked successfully!
Car 10 is leaving.
Car 9 is leaving.
○ kashmirkj@Kashmirpc:~/Operating_System/Lab_08$
```

## Task2:

Input:

```
● ● ●
1 #include <stdio.h>
2 #include <pthread.h>
3 #include <semaphore.h>
4 #include <unistd.h>
5 #define BUFFER_SIZE 5
6 int buffer[BUFFER_SIZE];
7 int in = 0; // Producer index
8 int out = 0; // Consumer index
9 sem_t empty; // Counts empty slots
10 sem_t full; // Counts full slots
11 pthread_mutex_t mutex;
12 void* producer(void* arg) {
13     int id = *(int*)arg;
14     for(int i = 0; i < 3; i++) { // Each producer makes 3 items
15         int item = id * 100 + i;
16         // TODO: Wait for empty slot
17         sem_wait(&empty);
18         // TODO: Lock the buffer
19         pthread_mutex_lock(&mutex);
20         // Add item to buffer
21         buffer[in] = item;
22         printf("Producer %d produced item %d at position %d\n",
23                id, item, in);
24         in = (in + 1) % BUFFER_SIZE;
25         // TODO: Unlock the buffer
26         pthread_mutex_unlock(&mutex);
27         // TODO: Signal that buffer has a full slot
28         sem_post(&full);
29         sleep(1);
30     }
31     return NULL;
32 }
33 void* consumer(void* arg) {
34     int id = *(int*)arg;
35     for(int i = 0; i < 3; i++) {
36         // TODO: Students complete this similar to producer
37         sem_wait(&full);
38         pthread_mutex_lock(&mutex);
39         int item = buffer[out];
40         printf("Consumer %d consumed item %d from position %d\n",
41                id, item, out);
42         out = (out + 1) % BUFFER_SIZE;
43         pthread_mutex_unlock(&mutex);
44         sem_post(&empty);
45         sleep(2); // Consumers are slower
46     }
47     return NULL;
48 }
49 int main() {
50     pthread_t prod[2], cons[2];
51     int ids[2] = {1, 2};
52     // Initialize semaphores
53     sem_init(&empty, 0, BUFFER_SIZE); // ALL slots empty initially
54     sem_init(&full, 0, 0);
55     pthread_mutex_init(&mutex, NULL);
56     // No slots full initially
57     // Create producers and consumers
58     for(int i = 0; i < 2; i++) {
59         pthread_create(&prod[i], NULL, producer, &ids[i]);
60         pthread_create(&cons[i], NULL, consumer, &ids[i]);
61     }
62     // Wait for completion
63     for(int i = 0; i < 2; i++) {
64         pthread_join(prod[i], NULL);
65         pthread_join(cons[i], NULL);
66     }
67     // Cleanup
68     sem_destroy(&empty);
69     sem_destroy(&full);
70     pthread_mutex_destroy(&mutex);
71     return 0;
72 }
```

## Output:

```
TERMINAL
● kashmirkj@Kashmirpc:~/Operating_System/Lab_08$ gcc Task2.c -o Task2 -lpthread
● kashmirkj@Kashmirpc:~/Operating_System/Lab_08$ ./Task2
Producer 1 produced item 100 at position 0
Producer 2 produced item 200 at position 1
Consumer 1 consumed item 100 from position 0
Consumer 2 consumed item 200 from position 1
Producer 1 produced item 101 at position 2
Producer 2 produced item 201 at position 3
Producer 2 produced item 202 at position 4
Consumer 2 consumed item 101 from position 2
Producer 1 produced item 102 at position 0
Consumer 1 consumed item 201 from position 3
Consumer 2 consumed item 202 from position 4
Consumer 1 consumed item 102 from position 0
○ kashmirkj@Kashmirpc:~/Operating_System/Lab_08$
```

Ln 72, Col 2 Spaces: 4 UTF-8 LF { } C windows-gcc-x64

## 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:

- $1 \times 100 + 1 = 101$
- $1 \times 100 + 2 = 102$
- $1 \times 100 + 3 = 103$

When id = 2:

Items:

- $2 \times 100 + 1 = 201$
- $2 \times 100 + 2 = 202$
- $2 \times 100 + 3 = 203$

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)  
mutex\_lock(&mutex)  
#critical section  
mutex\_unlock(&mutex)  
post (&full)

// Decrement empty  
// lock mutex  
// unlock section  
// Increment full.

### Consumer :

Wait (&full)  
mutex\_lock(&mutex)  
#critical section  
mutex\_unlock(&mutex)  
Post (&empty).

// Decrement full  
// lock section below  
// unlock section  
// Increment empty.

