



# National Textile University

*Department of Computer Science*

**Subject:**

Operating system

**Submitted To:**

Sir Nasir

**Submitted By:**

Noor Ul Ain

**Registration No:**

23-NTU-CS-1221

**Lab No:**

10

**Semester:**

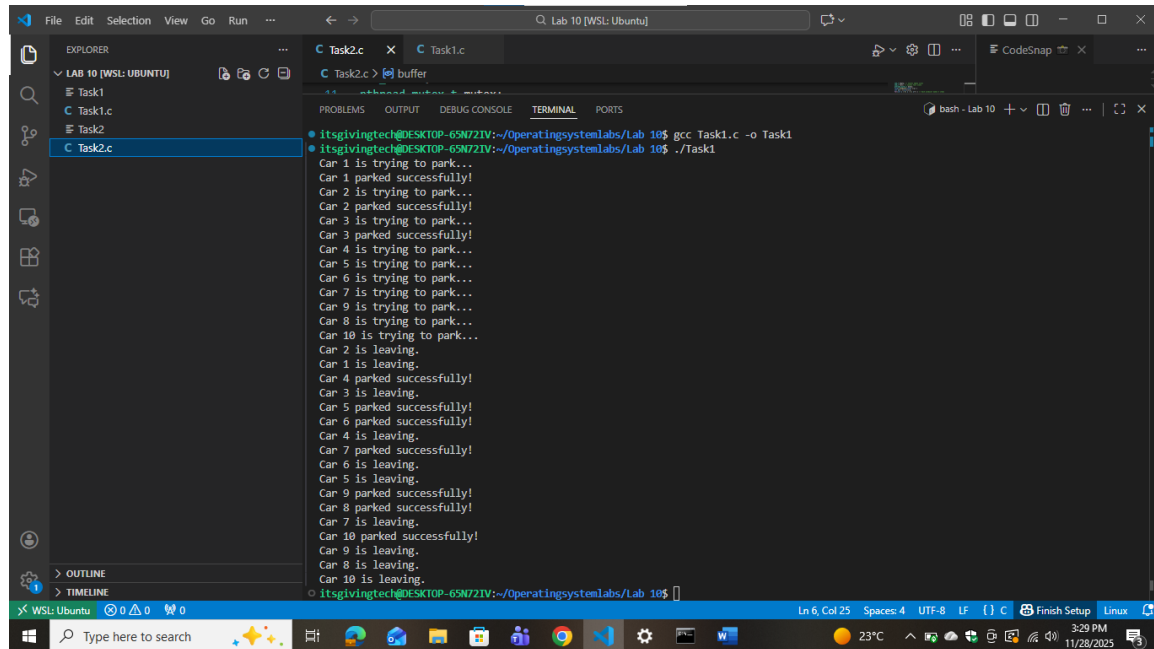
5

## Task 1:

### Code:

```
#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:



```
itsgivingtech@DESKTOP-6SN72IV:~/OperatingSystemLabs/Lab 10$ gcc Task1.c -o Task1
itsgivingtech@DESKTOP-6SN72IV:~/OperatingSystemLabs/Lab 10$ ./Task1
Car 1 is trying to park...
Car 1 parked successfully!
Car 2 is trying to park...
Car 2 parked successfully!
Car 3 is trying to park...
Car 3 parked successfully!
Car 4 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 10 is trying to park...
Car 2 is leaving.
Car 1 is leaving.
Car 4 parked successfully!
Car 3 is leaving.
Car 5 parked successfully!
Car 6 parked successfully!
Car 4 is leaving.
Car 7 parked successfully!
Car 6 is leaving.
Car 5 is leaving.
Car 9 parked successfully!
Car 8 parked successfully!
Car 7 is leaving.
Car 10 parked successfully!
Car 9 is leaving.
Car 8 is leaving.
Car 10 is leaving.
```

## Demonstration:

Here only one counting semaphore is used which is parking\_Spaces. It's value is initialized to 3 which means 3 cars can enter the parking lot before the semaphores make other cars to wait. When a car tries to enter the parking area it will go to `sem_Wait(parking_spaces)` to decrease the slot and if all the parking spaces are in use the car will wait.

- If a car gets space it will stay here for 2 seconds
- After that it will free the space and the semaphore value will increase through `sem_post`.

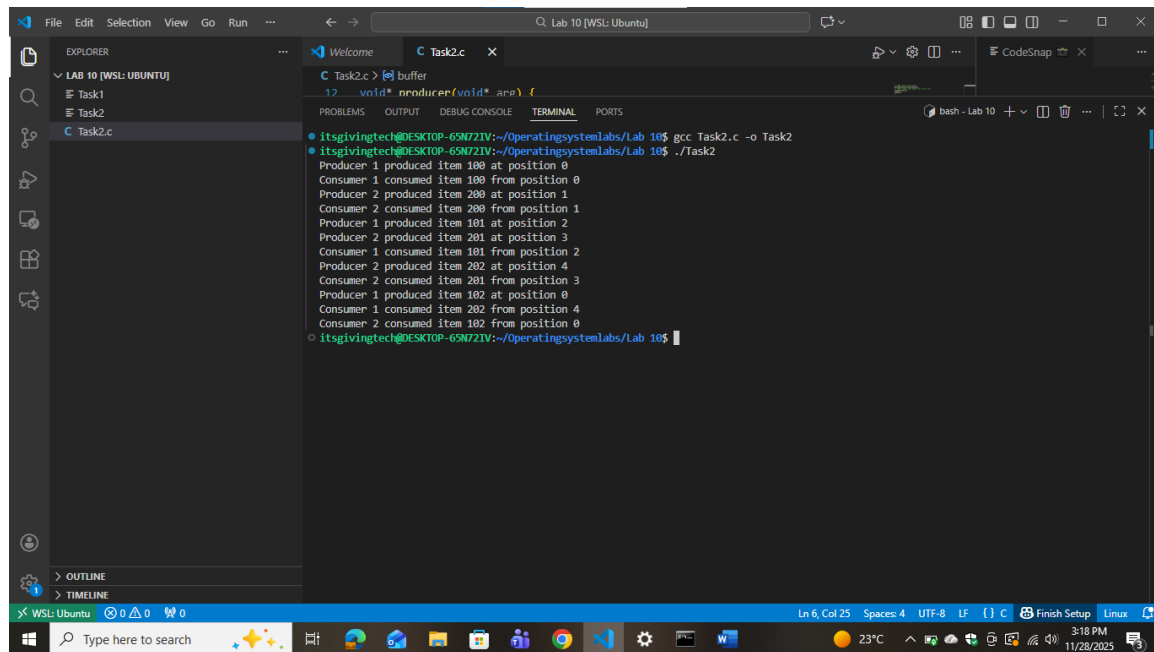
## Task 2:

```

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
34 void* consumer(void* arg) {
35     int id = *(int*)arg;
36     for(int i = 0; i < 3; i++) {
37         // TODO: Students complete this similar to producer
38         sem_wait(&full);
39         pthread_mutex_lock(&mutex);
40         int item = buffer[out];
41         printf("Consumer %d consumed item %d from position %d\n",
42             id, item, out);
43         out = (out + 1) % BUFFER_SIZE;
44         pthread_mutex_unlock(&mutex);
45         sem_post(&empty);
46         sleep(2); // Consumers are slower
47     }
48     return NULL;
49 }
50
51 int main() {
52     pthread_t prod[2], cons[2];
53     int ids[2] = {1, 2};
54     // Initialize semaphores
55     sem_init(&empty, 0, BUFFER_SIZE); // All slots empty initially
56     sem_init(&full, 0, 0); // No slots full initially
57     pthread_mutex_init(&mutex, NULL);
58     // Create producers and consumers
59     for(int i = 0; i < 2; i++) {
60         pthread_create(&prod[i], NULL, producer, &ids[i]);
61         pthread_create(&cons[i], NULL, consumer, &ids[i]);
62     }
63     // Wait for completion
64     for(int i = 0; i < 2; i++) {
65         pthread_join(prod[i], NULL);
66         pthread_join(cons[i], NULL);
67     }
68     // Cleanup
69     sem_destroy(&empty);
70     sem_destroy(&full);
71     pthread_mutex_destroy(&mutex);
72     return 0;
73 }

```

## Output:



```
Task2.c > | buffer
12 void* producer(void* arg) {
itsgivingtech@DESKTOP-65N72IV:~/OperatingSystemLabs/Lab 10$ gcc Task2.c -o Task2
itsgivingtech@DESKTOP-65N72IV:~/OperatingSystemLabs/Lab 10$ ./Task2
Producer 1 produced item 100 at position 0
Consumer 1 consumed item 100 from position 0
Producer 2 produced item 200 at position 1
Consumer 2 consumed item 200 from position 1
Producer 1 produced item 101 at position 2
Producer 2 produced item 201 at position 3
Consumer 1 consumed item 101 from position 2
Producer 2 produced item 202 at position 4
Consumer 2 consumed item 201 from position 3
Producer 1 produced item 102 at position 0
Consumer 1 consumed item 202 from position 4
Consumer 2 consumed item 102 from position 0
itsgivingtech@DESKTOP-65N72IV:~/OperatingSystemLabs/Lab 10$
```

## Demonstration:

Two semaphores are used here, one is “full” and one is “empty”. Empty is used by producers, it is initialized with 5 and if its value becomes zero which means the buffer is full. Full semaphore is used by the consumers, it is initialized with zero which means nothing is produced yet.

- **Producer function:**

In producer function, it goes to `sem_wait(&empty)` to decrease the empty slot, then the buffer will be locked and the producer will produce this item at the specific index, unlock the mutex and then go to the `sem_full(&full)`

- **Consumer function:**

In consumer function, it goes to `sem_wait(&full)` to decrease the full slot, then the buffer is locked and the consumer will consume the item which was produced by the specific producer, unlock the mutex and go to the `sem_post(&empty)`

- **In main function:**

- A loop will run two times for each producer and consumer.
- Each producer will produce 3 values so in total 2 producers will produce 6 values.
- Each consumer will consume 3 values so in total 2 consumers will consume 6 values.
- Item ids are 100, 101, 102, 200, 201, 202.

**Change in consumer value:**

- `for(int i = 0; i < 4; i++)`
- It means each consumer tries to consume 4 items so 2 consumers are trying to consume 8 items but the producers produce just 6 times which will make consumers to wait forever and causes a deadlock.

