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:

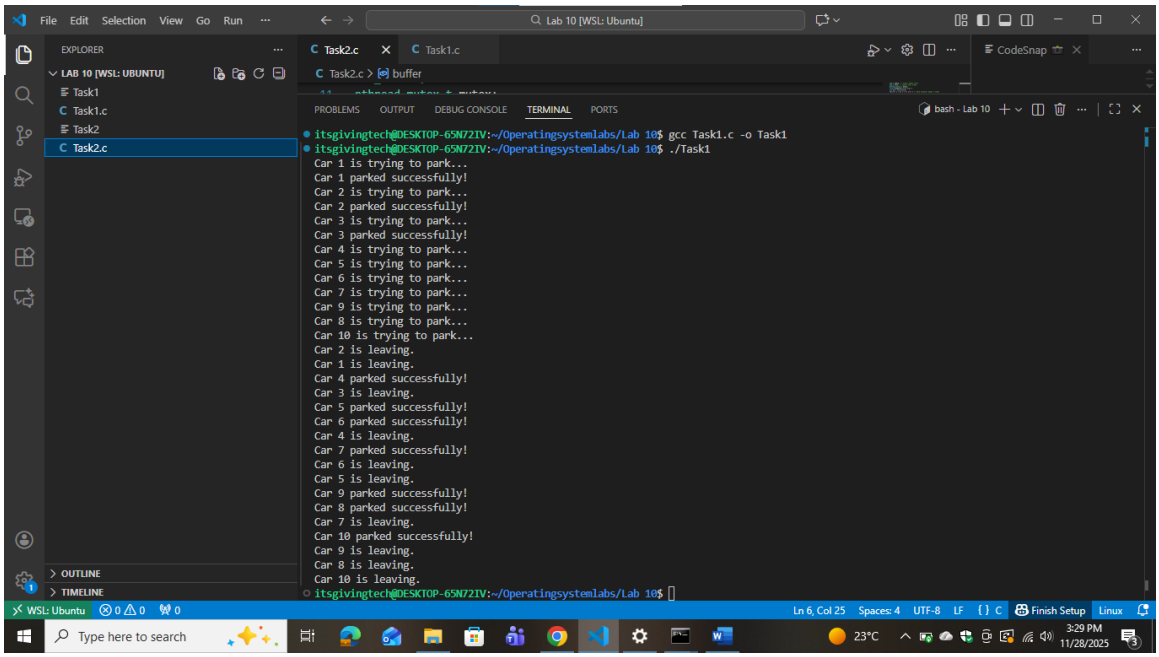
5

Task 1:

Code:

```c
#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:



Demonstration:

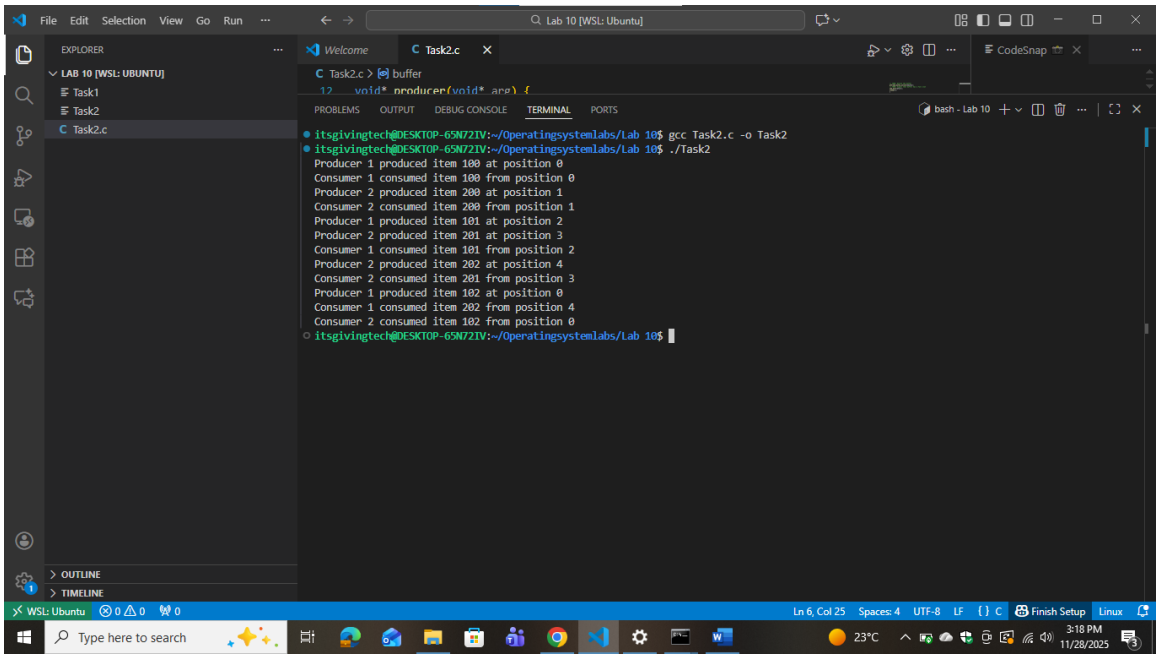
Here only one counting semaphore is used which is parking_Spaces. It's value is initialized to 3 which means 3 cars can enter the parking lot before the semaphores make other cars to wait. When a car tries to enter the parking area it will go to sem_Wait(parking_spaces) to decrease the slot and if all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

```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); // No slots full initially
pthread_mutex_init(&mutex, NULL);
// 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:



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 it 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.