National Textile University

Department of Computer Science

Subject:

Operating System

Submitted To:

Sir Nasir Mehmood

Submitted By:

Hafsa Tayyab

Registration No:

23-NTU-CS-1163

Lab No:

10

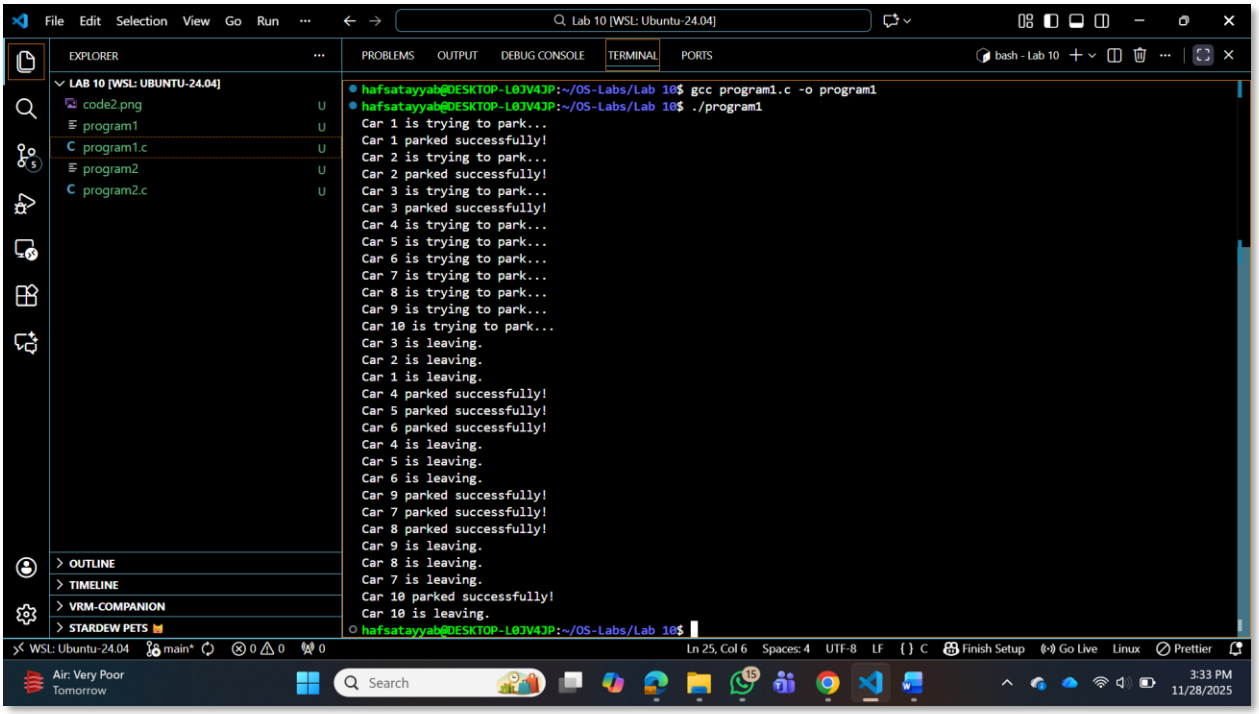
Semester:

5th

Counting Semaphores

Example 1: Parking Lot Simulation

```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;
}
```



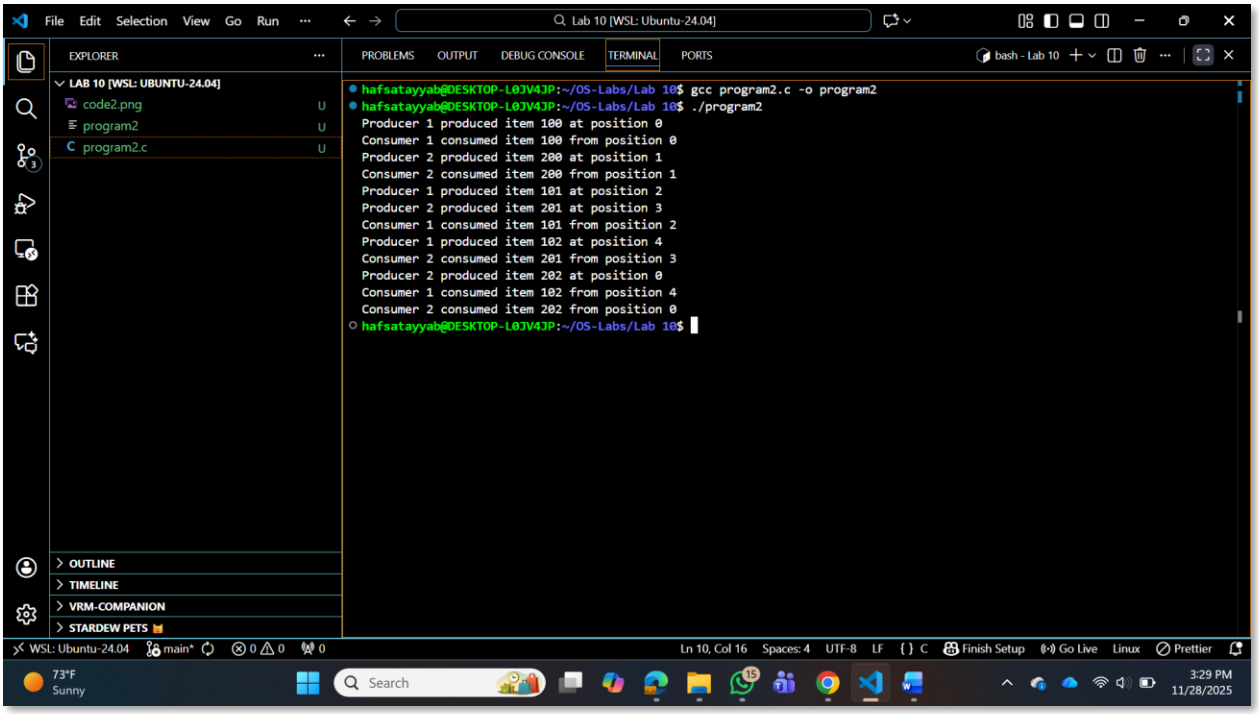
Remarks:

It is a basic demonstration of counting semaphore example where a parking_spaces semaphore is tracking the number of available parking spaces. Only 3 threads can occupy the space at the same time, thus other 7 cars wait for the post function execution. Once a release function runs, one car comes out from the waiting situation and go ton occupy the space.

- Sem_wait(&parking_spaces) = occupy space.
- Sem_post(&parking_spaces) = free space.

Example 2: Producer-Consumer Problem

```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;
}
```



Remarks:

In this program, to demonstrate counting semaphore example,

- 2 producer threads and 2 consumer threads,
- Two semaphores, one to count empty slots and second to count full slots are used.
- Producer threads create 6 items, 3 by each thread.
- Same with the consumer one.
- Items ids: 100, 101, 102, 200, 201, 202.
- Producer runs a wait function to claim an empty slot first and then consumer consumes that spot.

Deadlock Problem:

The problem starts when producer produces 6 items but consumer function's for loop is creted to run 4 times thus ready to consume 8 items.
The Producer function runs a post function for the full semaphore to indicate it to consume a slot. But in this case producer will signal the consumer to 6 spots only, thus the consumer will remain waiting for signal (a post function). The program will go to a deadlock situation where all functions are waiting for other functions to complete.