



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

```
● ● ●

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

The screenshot shows a Windows desktop environment with the Visual Studio Code (VS Code) application open. The title bar reads "Lab 10 [WSL: Ubuntu-24.04]". The main area displays the terminal output of a C program. The code uses semaphores to manage a limited number of parking spaces. The terminal output shows 10 cars attempting to park, with only 3 successful entries and 7 others waiting. Once a car leaves, another is freed up.

```
hafsatayyab@DESKTOP-L0JV4JP:~/OS-Labs/Lab 10$ gcc program1.c -o program1
hafsatayyab@DESKTOP-L0JV4JP:~/OS-Labs/Lab 10$ ./program1
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 9 is trying to park...
Car 10 is trying to park...
Car 3 is leaving.
Car 2 is leaving.
Car 1 is leaving.
Car 4 parked successfully!
Car 5 parked successfully!
Car 6 parked successfully!
Car 4 is leaving.
Car 5 is leaving.
Car 6 is leaving.
Car 9 parked successfully!
Car 7 parked successfully!
Car 8 parked successfully!
Car 9 is leaving.
Car 8 is leaving.
Car 7 is leaving.
Car 10 parked successfully!
Car 10 is leaving.
hafsatayyab@DESKTOP-L0JV4JP:~/OS-Labs/Lab 10$
```

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 to occupy the space.

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

Example 2: Producer-Consumer Problem

```
● ○ ●
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", id, item, in);
23         in = (in + 1) % BUFFER_SIZE;
24         // TODO: Unlock the buffer
25         pthread_mutex_unlock(&mutex);
26         // TODO: Signal that buffer has a full slot
27         sem_post(&full);
28         sleep(1);
29     }
30     return NULL;
31 }
32 void* consumer(void* arg) {
33     int id = *(int*)arg;
34     for(int i = 0; i < 3; i++) {
35         // TODO: Students complete this similar to producer
36         sem_wait(&full);
37         pthread_mutex_lock(&mutex);
38         int item = buffer[out];
39         printf("Consumer %d consumed item %d from position %d\n",
40             id, item, out);
41         out = (out + 1) % BUFFER_SIZE;
42         pthread_mutex_unlock(&mutex);
43         sem_post(&empty);
44         sleep(2); // Consumers are slower
45     }
46     return NULL;
47 }
48 int main() {
49     pthread_t prod[2], cons[2];
50     int ids[2] = {1, 2};
51     // Initialize semaphores
52     sem_init(&empty, 0, BUFFER_SIZE); // All slots empty initially
53     sem_init(&full, 0, 0);
54     pthread_mutex_init(&mutex, NULL);
55     // No slots full initially
56     // Create producers and consumers
57     for(int i = 0; i < 2; i++) {
58         pthread_create(&prod[i], NULL, producer, &ids[i]);
59         pthread_create(&cons[i], NULL, consumer, &ids[i]);
60     }
61     // Wait for completion
62     for(int i = 0; i < 2; i++) {
63         pthread_join(prod[i], NULL);
64         pthread_join(cons[i], NULL);
65     }
66     // Cleanup
67     sem_destroy(&empty);
68     sem_destroy(&full);
69     pthread_mutex_destroy(&mutex);
70     return 0;
71 }
```

```

● hafsatayyab@DESKTOP-L0JV4JP:~/OS-Labs/Lab 10$ gcc program2.c -o program2
● hafsatayyab@DESKTOP-L0JV4JP:~/OS-Labs/Lab 10$ ./program2
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 1 produced item 102 at position 4
Consumer 2 consumed item 201 from position 3
Producer 2 produced item 202 at position 0
Consumer 1 consumed item 102 from position 4
Consumer 2 consumed item 202 from position 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 created 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.