



National Textile University

Department of Computer Science

Subject:

Operating Systems

Submitted To:

Sir Nasir Mehmood

Submitted By:

Alishba Riasat

Registration No:

23-NTU-CS-1135

Lab No:

10

Semester:

5th

Example 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:

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS bash - Lab_10 + v [ ] [ ] ... [ ] [ ] x

● alishba@DESKTOP-S3K0VA5:~/Lab_10$ gcc Lab_10_Example1.c -o Example1 -lpthread
● alishba@DESKTOP-S3K0VA5:~/Lab_10$ ./Example1
Car 1 is trying to park...
Car 2 is trying to park...
Car 1 parked successfully!
Car 3 is trying to park...
Car 3 parked successfully!
Car 4 is trying to park...
Car 2 parked successfully!
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 1 is leaving.
Car 4 parked successfully!
Car 3 is leaving.
Car 2 is leaving.
Car 6 parked successfully!
Car 5 parked successfully!
Car 4 is leaving.
Car 5 is leaving.
Car 7 parked successfully!
Car 6 is leaving.
Car 8 parked successfully!
Car 9 parked successfully!
Car 7 is leaving.
Car 8 is leaving.
Car 9 is leaving.
Car 10 parked successfully!
Car 10 is leaving.
● alishba@DESKTOP-S3K0VA5:~/Lab_10$ [ ]
```

Demonstration:

In this parking lot simulation program, we have 3 parking spaces initialized as `sem_init(&parking_spaces, 0, 3)`. 10 cars are trying to park. These threads are created as `pthread_t cars[10]`. To avoid the racing condition for this program, we have a car function, in which we have used `sem_wait(&parking_spaces)` and `sem_post(&parking_spaces)`.

`sem_wait(&parking_spaces)` ensures that:

- If a parking space is available, the cards park and decrement the parking spaces.
- If no space is available and parking spaces ==0 then cars wait until the space is free.

`sem_post(&parking_spaces)` ensures that:

The car leaves and frees up the space and increments parking spaces.

As a result, 3 cars park while other 7 wait until the parking spaces are free.

Example 2:

Code:

```
#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;
}

```

Output:

```

bash - Lab_10
● alishba@DESKTOP-S3K0VA5:~/Lab_10$ gcc Example2.c -o Example2 -lpthread
● alishba@DESKTOP-S3K0VA5:~/Lab_10$ ./Example2
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
Consumer 2 consumed item 201 from position 3
Producer 1 produced item 102 at position 4
Producer 2 produced item 202 at position 0
Consumer 1 consumed item 102 from position 4
Consumer 2 consumed item 202 from position 0
○ alishba@DESKTOP-S3K0VA5:~/Lab_10$

```

Ln 59, Col 1 (136 selected) Spaces: 4 UTF-8 LF {} C

Demonstration:

- In this producer consumer problem, we have a buffer whose size is specified to 5.
- We have 2 semaphores initialized as empty and full.
- Empty semaphore counts empty slots in the buffer. Initially all slots are empty and size of empty is same as buffer size.
- We have 2 producer threads and 2 consumer threads. One producer thread produces three items as 100, 101 and 102 and other as 200, 201 and 202. 2 consumer threads run three times to consume total of 6 items.
- In producer function, `sem_wait(&empty)` decrements the empty slots, lock the mutex and go into critical section to produce the item. After that mutex is unlocked and `sem_post(&full)` increments the full slots which indicates that some items are produced by producer threads.
- In consumer function, `sem_wait(&full)` decrements the full slots, lock the mutex, go into critical section and consumes the items. After that mutex is unlocked and `sem_post(&empty)` increments the empty slots which indicates that some items are consumed by consumer threads.
- In this loop `for(int i = 0; i < 3; i++)` in consumer function, if $i < 4$, each consumer will try to consume 4 items, total of 8 items, but only 6 items are being produced. The last 2 consumers wait and block forever which is a deadlock condition.