National Textile University

Department of Computer Science

Subject: Operating System

Submitted to: Sir Nasir

Submitted by: Amina

Reg. number: 23-NTU-CS-1136

Lab no.: lab10

Semester:5th

Example 1: Parking Lot Simulation

Problem Statement:
Simulate a parking lot with N parking spaces. Multiple cars (threads) try to park. If the lot is full, cars must wait until a space becomes available.

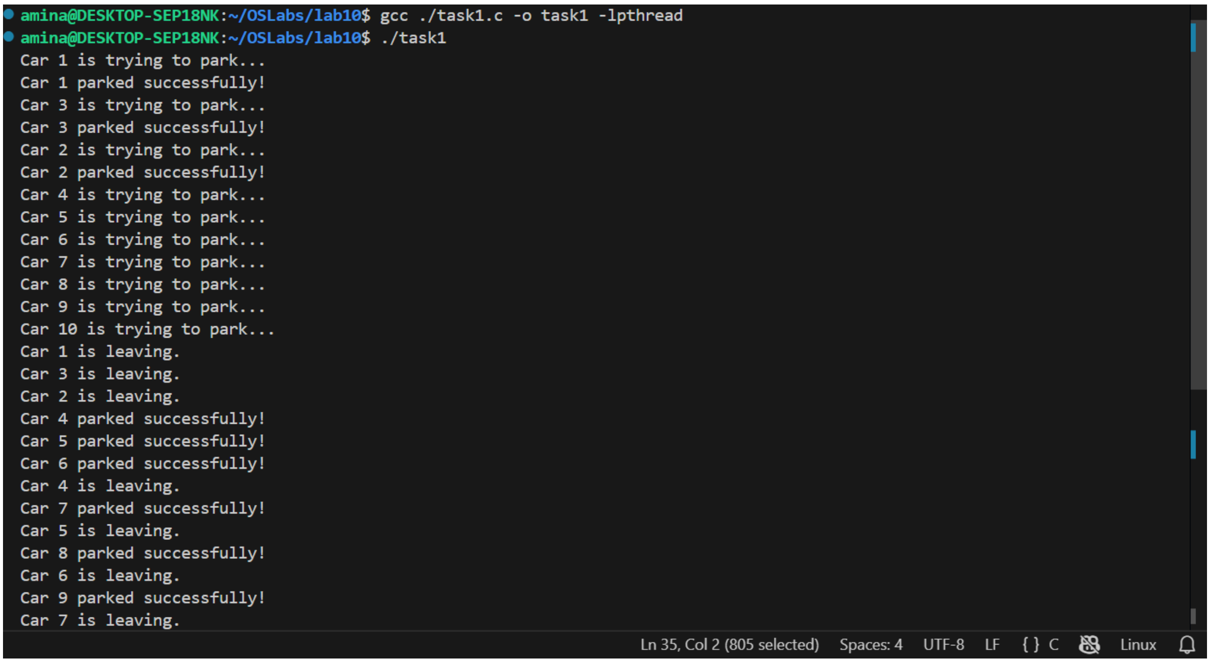
Code:

```
#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;
}
```



```
amina@DESKTOP-SEP18NK:~/OSLabs/lab10$ gcc ./task1.c -o task1 -lpthread
amina@DESKTOP-SEP18NK:~/OSLabs/lab10$ ./task1
Car 1 is trying to park...
Car 1 parked successfully!
Car 3 is trying to park...
Car 3 parked successfully!
Car 2 is trying to park...
Car 2 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 1 is leaving.
Car 3 is leaving.
Car 2 is leaving.
Car 4 parked successfully!
Car 5 parked successfully!
Car 6 parked successfully!
Car 4 is leaving.
Car 7 parked successfully!
Car 5 is leaving.
Car 8 parked successfully!
Car 6 is leaving.
Car 9 parked successfully!
Car 7 is leaving.
```

Example 2: Producer-Consumer Problem

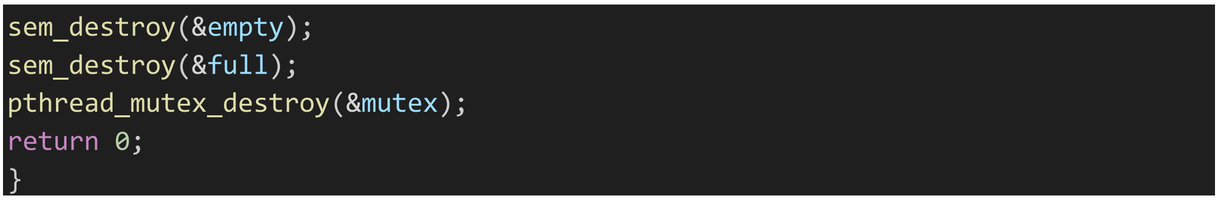
Scenario: A factory assembly line

- Producers make items and place them on a conveyor belt (buffer)
- Consumers take items from the belt
- The belt has limited space (say, 10 items max)

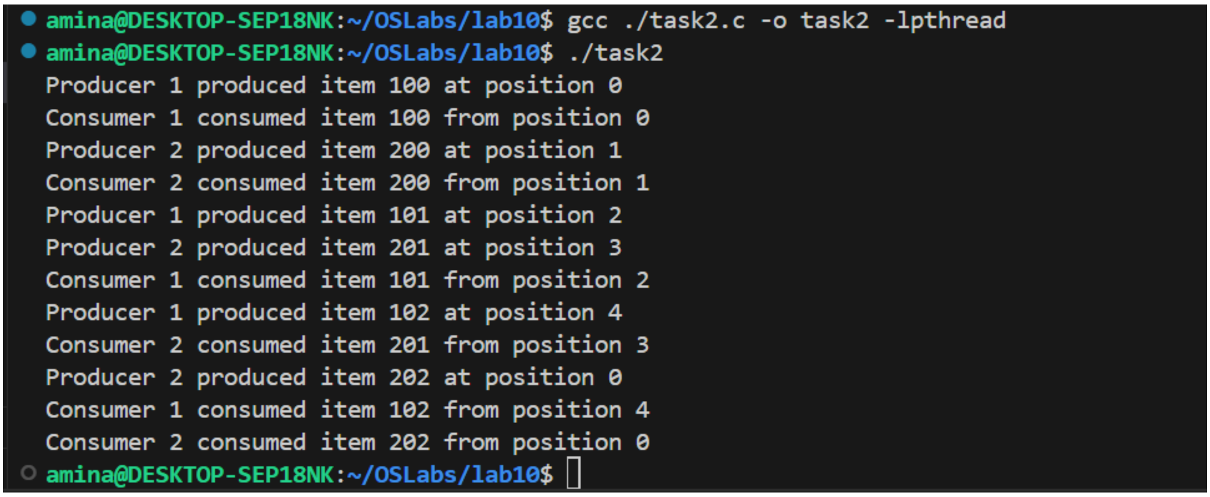
Code:

```
#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); // 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;
}
```



```
amina@DESKTOP-SEP18NK:~/OSLabs/lab10$ gcc ./task2.c -o task2 -lpthread
amina@DESKTOP-SEP18NK:~/OSLabs/lab10$ ./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 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
amina@DESKTOP-SEP18NK:~/OSLabs/lab10$
```

bufferSize=5

Producer:

1. sem-wait(&empty)   //producer will produce of buffer is empty
2. p_thread_mutex_lock(&mutex)   // lock the mutex
3. add value
4. unlock mutex
5. sem_post(&full)   //increment in full size by 1

Consumer:

1. sem-wait(&full)   //consumer will consume of buffer is full
2. p_thread_mutex_lock(&mutex)   // lock the mutex
3. use slot
4. unlock mutex
5. sem_post(&empty)   //increment in empty size by 1

Main:

- We have 2 producer, 2 consumer
- Initially empty=5 and full =0
- Total threads is 4

Question1:

```
int item = id * 100 + i;
```

- producer has id 1, 2
- total 6 items will produce
- each has 100, 101, 102, 200, 201, 202
- 4 threads will produce

Question2:

If consumer size is bigger than producer?

- Deadlock condition will form
- Consumer will keep waiting and has nothing to consume