National Textile University

Department of Computer Science

Subject: Operating System

Submitted to: Nasir Mahmood

Submitted by: esha

Reg number:23-NTU-CS-1146

Lab : 06

Semester:5th

Lab manual 10

Task 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++) {
```

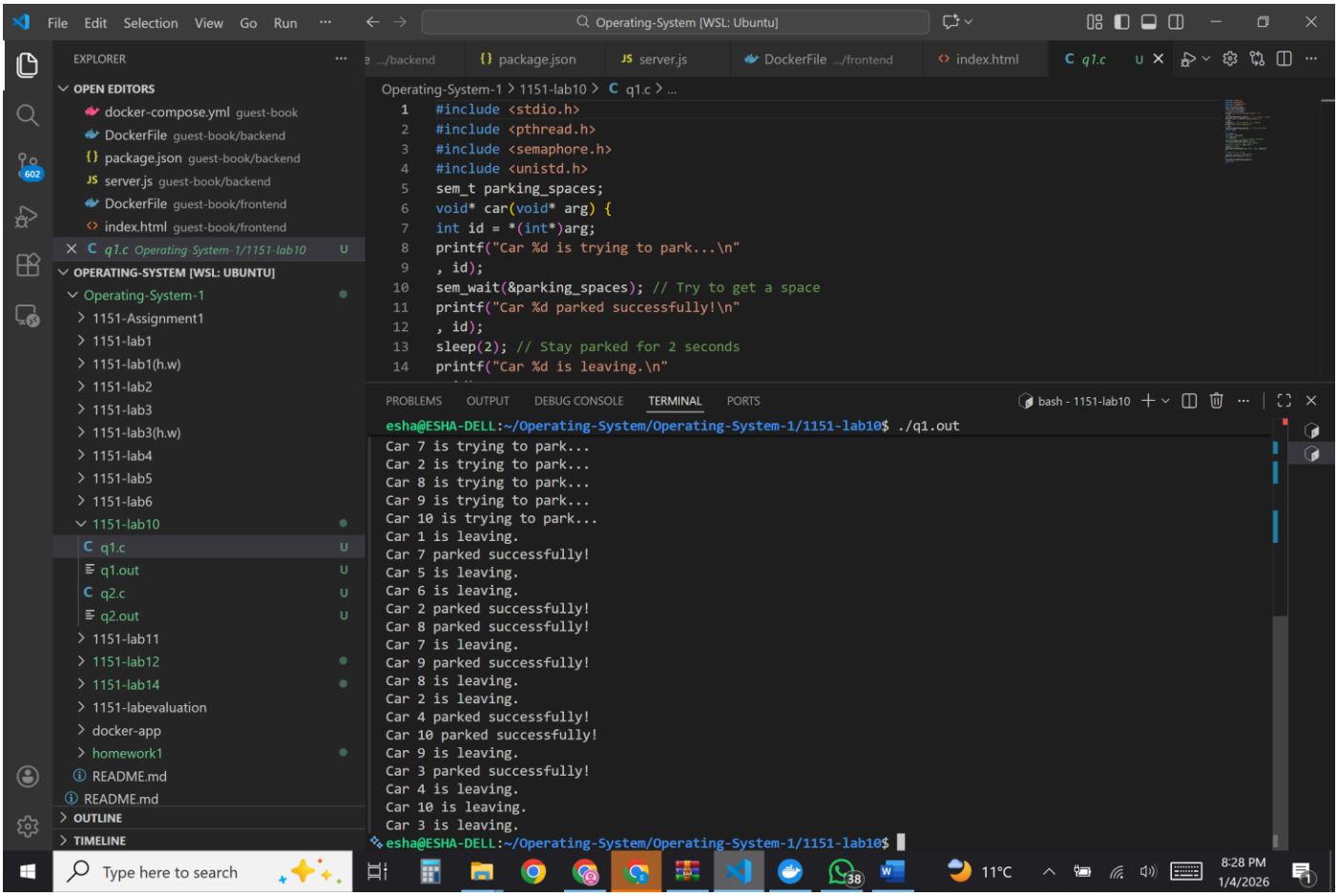
pthread_join(cars[i], NULL);

}

sem_destroy(&parking_spaces);

return 0;

}



Task 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

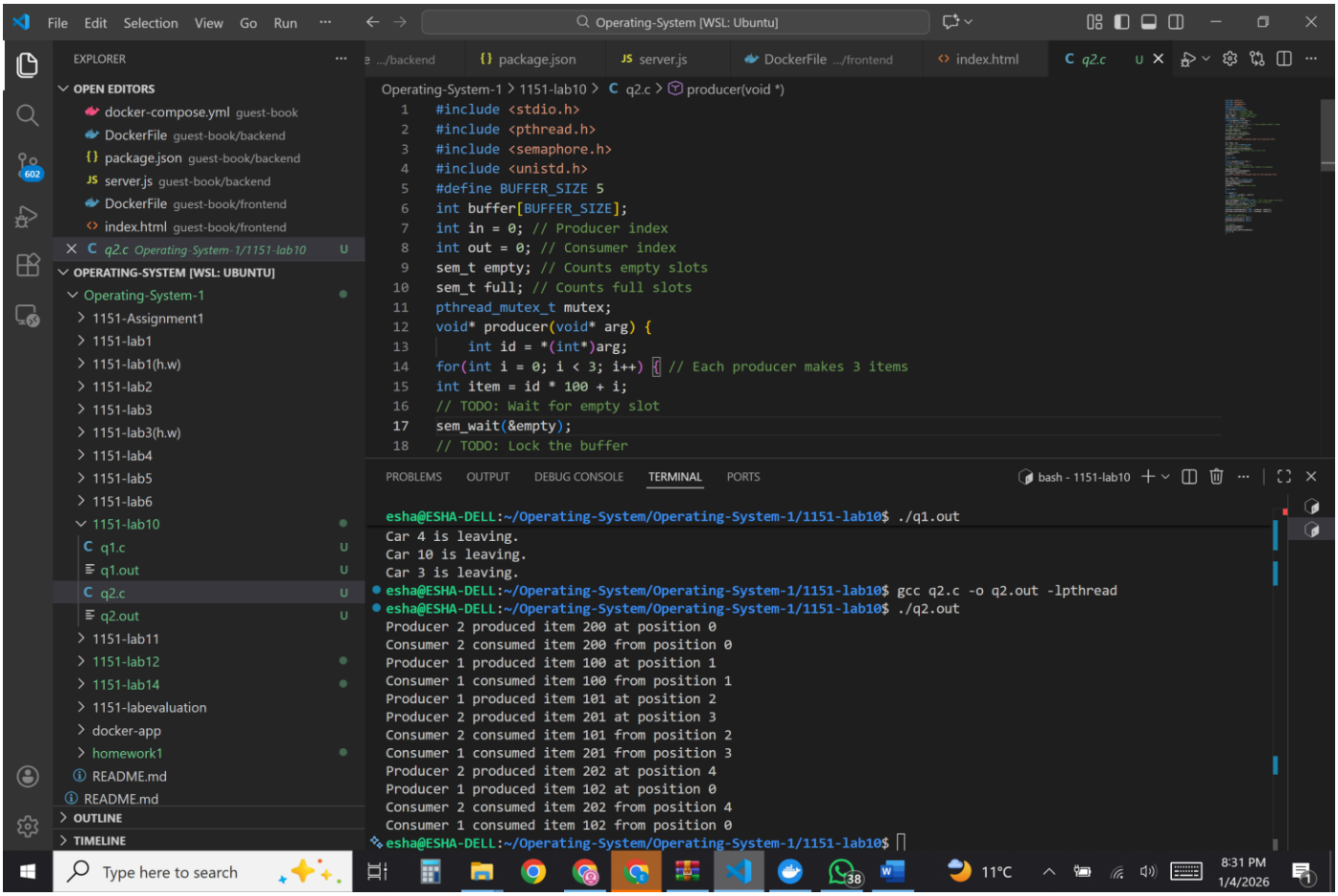
sem_t empty; // Counts empty slots

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); // 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;
}
```



Task 3: Block condition

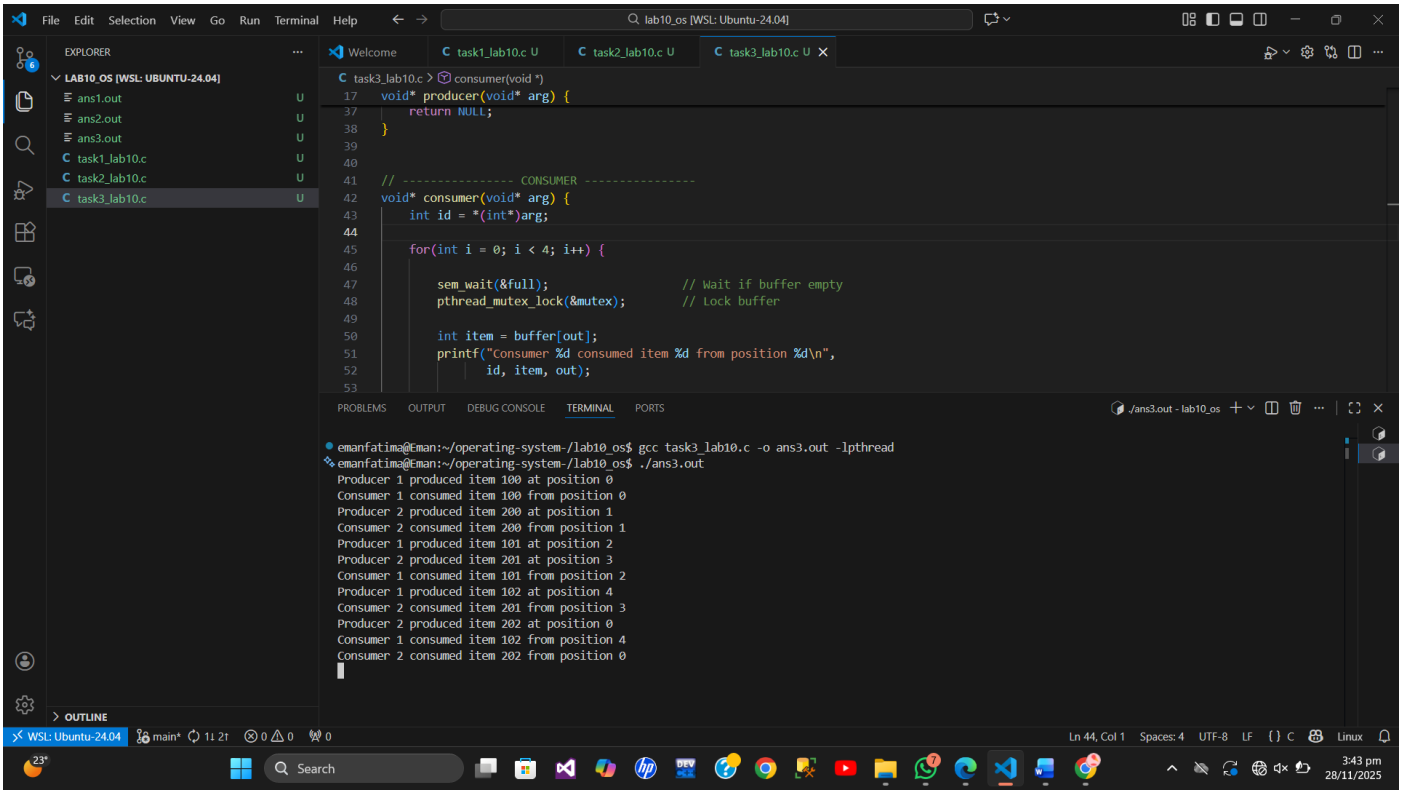
Code :

```
#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;

// ---------- PRODUCER ----------
```

```
void* producer(void* arg) {
    int id = *(int*)arg;

    for(int i = 0; i < 3; i++) {  // Each producer produces 3 items
        int item = id * 100 + i;

        sem_wait(&empty);                // Wait if buffer is full
        pthread_mutex_lock(&mutex);      // Lock shared buffer

        buffer[in] = item;
        printf("Producer %d produced item %d at position %d\n",
               id, item, in);

        in = (in + 1) % BUFFER_SIZE;

        pthread_mutex_unlock(&mutex);    // Unlock buffer
       sem_post(&full);                  // Signal item added

        sleep(1);
    }
    return NULL;
}

// ---------------- CONSUMER ----------------
void* consumer(void* arg) {
    int id = *(int*)arg;

    for(int i = 0; i < 4; i++) {

        sem_wait(&full);                 // Wait if buffer empty
        pthread_mutex_lock(&mutex);      // Lock buffer

        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);      // Unlock buffer
        sem_post(&empty);                  // Signal empty slot

        sleep(2);  // Consumers are slower
    }
    return NULL;
}

// ---------------- MAIN ----------------
int main() {
    pthread_t prod[2], cons[2];
    int ids[2] = {1, 2};

    // Initialize semaphores
    sem_init(&empty, 0, BUFFER_SIZE);  // All slots empty
    sem_init(&full, 0, 0);             // No slots full

    pthread_mutex_init(&mutex, NULL);

    // Create producer & consumer threads
    for(int i = 0; i < 2; i++) {
        pthread_create(&prod[i], NULL, producer, &ids[i]);
        pthread_create(&cons[i], NULL, consumer, &ids[i]);
    }

    // Join threads
    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 it will be in block deadlock situation**
- **Because the values are only 6 but consumer is consuming 8 vaalues so it will go in block situation increaseing value of consumer less values produce.**

Q A:

- **Total values produce: 6 in task 1**
- **Threads : 4 (read and write)**
- **Semaphore : 2 (b empty and full**