National Textile University

Department of Computer Science

Subject

Operating System

Submitted to:

Sir Nasir Mehmood

Submitted by:

Kashmir Jamshaid

Registration Number

23-NTU-CS-1167

Home Task

Week-10

Semester

5th

Q#1

Exercise 1 – Hotel Room Occupancy Problem

Scenario:

A hotel has **N Rooms**.
Only N people can take room at a time; others must wait outside.
One person can only take one room and one room can only be taken by one person.

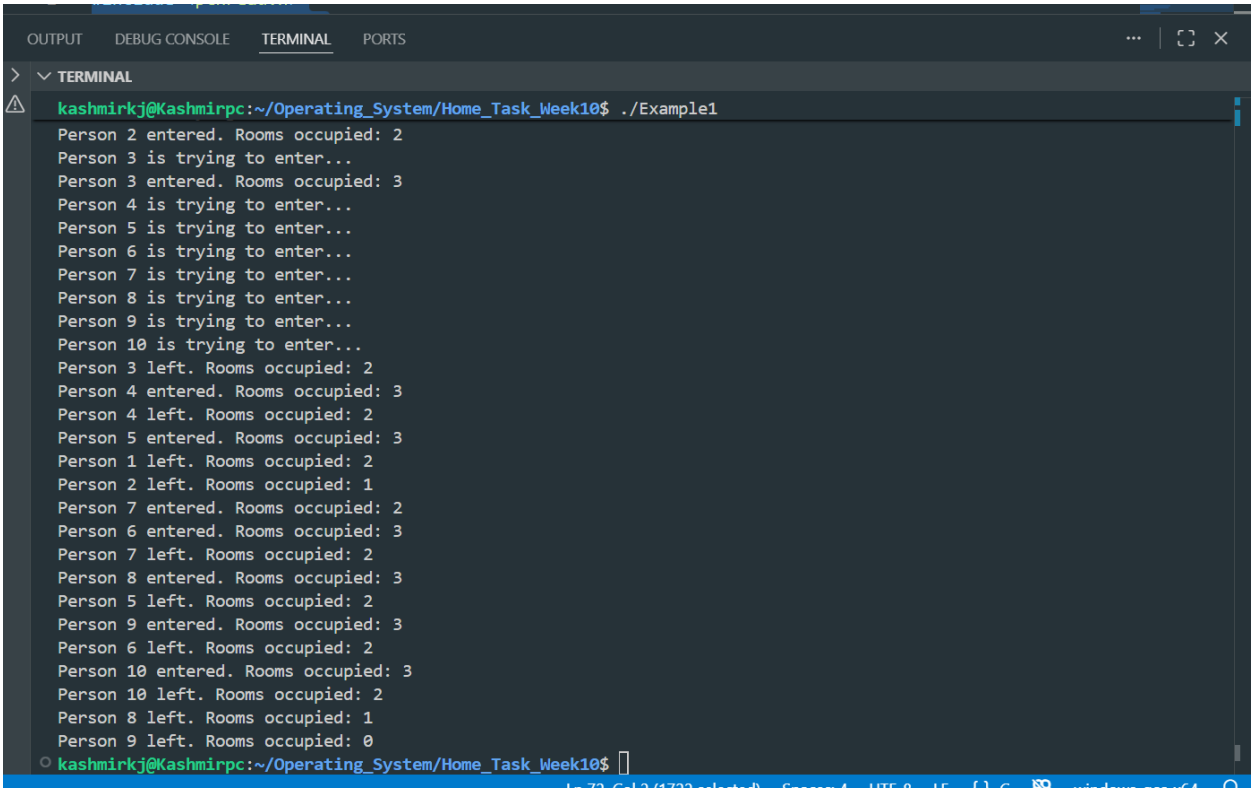
Tasks:

1. Use a **counting semaphore** initialized to `N`
2. Each person (thread) enters, stays for 1–3 seconds, leaves
3. Print:
   - "Person X entered"
   - "Person X left"
4. Show how many rooms are currently occupied

Code:

```c
#include <stdio.h>
#include <pthread.h>
#include <semaphore.h>
#include <unistd.h>
#include <stdlib.h>

sem_t rooms;               // Counting semaphore for available rooms
pthread_mutex_t mutex;     // protect the occupied counter

int occupied = 0;          // no of occupied rooms

void* person(void* arg) {
    int id = *(int*)arg;

    printf("Person %d is trying to enter...\n", id);

    // Wait for an available room (decrement semaphore)
    sem_wait(&rooms);

    // Lock the counter update
    pthread_mutex_lock(&mutex);
    occupied++;
    printf("Person %d entered. Rooms occupied: %d\n", id, occupied);
    pthread_mutex_unlock(&mutex);

    // Stay for random 1-3 seconds
    int stay = (rand() % 3) + 1;
    sleep(stay);

    // Leaving the room
    pthread_mutex_lock(&mutex);
    occupied--;
    printf("Person %d left. Rooms occupied: %d\n", id, occupied);
    pthread_mutex_unlock(&mutex);

    // Free the room (increment semaphore)
    sem_post(&rooms);

    return NULL;
}

int main() {
    int N = 3;             // Number of rooms
    int totalPeople = 10; // Total threads
    pthread_t people[totalPeople];
    int ids[totalPeople];

    srand(time(NULL)); // For random stay time

    // Initialize semaphore with N rooms
    sem_init(&rooms, 0, N);

    // Initialize mutex
    pthread_mutex_init(&mutex, NULL);

    // Create threads (people)
    for(int i = 0; i < totalPeople; i++) {
        ids[i] = i + 1;
        pthread_create(&people[i], NULL, person, &ids[i]);
    }

    // Wait for all people to complete
    for(int i = 0; i < totalPeople; i++) {
        pthread_join(people[i], NULL);
    }

    // Cleanup
    sem_destroy(&rooms);
    pthread_mutex_destroy(&mutex);

    return 0;
}
```

Output:



Q#2

Exercise 2 – Download Manager Simulation

Scenario:

You have a download manager that can download **max 3 files at a time**.

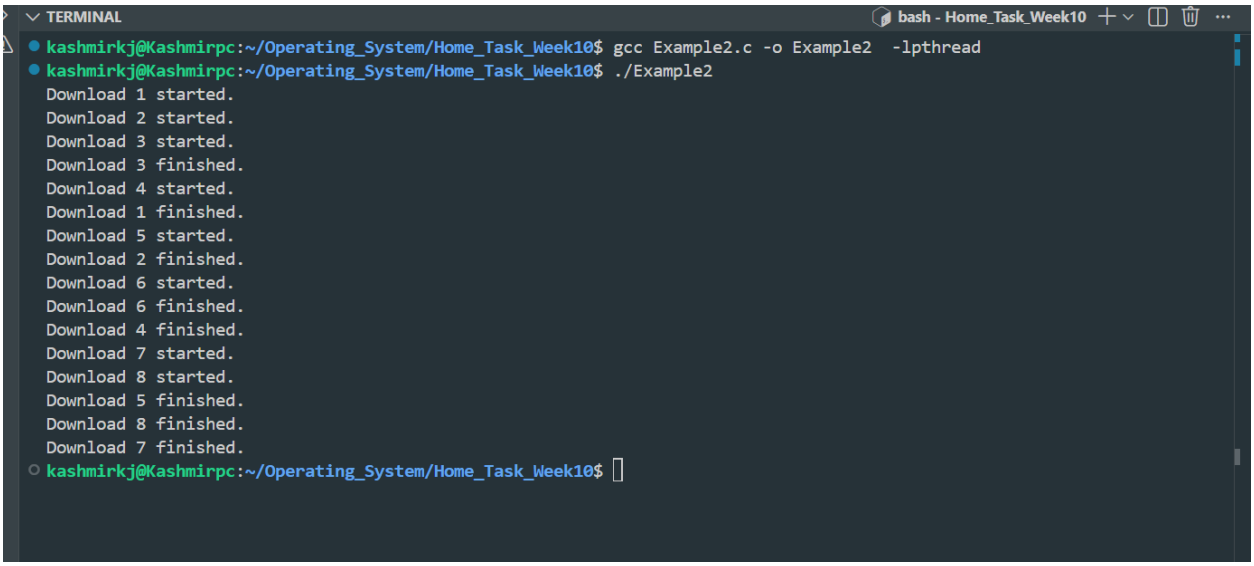
Tasks:

- Create 8 download threads
- Use a counting semaphore with value = 3
- Each download takes random 1–5 seconds
- Print messages for start/end of each download

Code:

```c
#include <stdio.h>
#include <pthread.h>
#include <semaphore.h>
#include <unistd.h>
#include <stdlib.h>

sem_t download_slots;   // Semaphore to allow max 3 downloads at a time

void* download_file(void* arg) {
    int id = *(int*)arg;


    // Wait for an available download slot
    sem_wait(&download_slots);

    // Start download
    printf("Download %d started.\n", id);

    int time_needed = (rand() % 5) + 1;  // Random 1-5 seconds
    sleep(time_needed);

    // End download
    printf("Download %d finished.\n", id);

    // Free the slot
    sem_post(&download_slots);

    return NULL;
}

int main() {
    pthread_t downloads[8];
    int ids[8];

    srand(time(NULL));

    // Initialize semaphore with 3 download slots
    sem_init(&download_slots, 0, 3);

    // Create 8 download threads
    for(int i = 0; i < 8; i++) {
        ids[i] = i + 1;
        pthread_create(&downloads[i], NULL, download_file, &ids[i]);
    }

    // Wait for all downloads to complete
    for(int i = 0; i < 8; i++) {
        pthread_join(downloads[i], NULL);
    }

    // Clean up
    sem_destroy(&download_slots);

    return 0;
}
```

Output:



Question #3

Exercise 3 – Library Computer Lab Access

Scenario:

A university lab has **K computers**.
Students must wait until a computer becomes free.

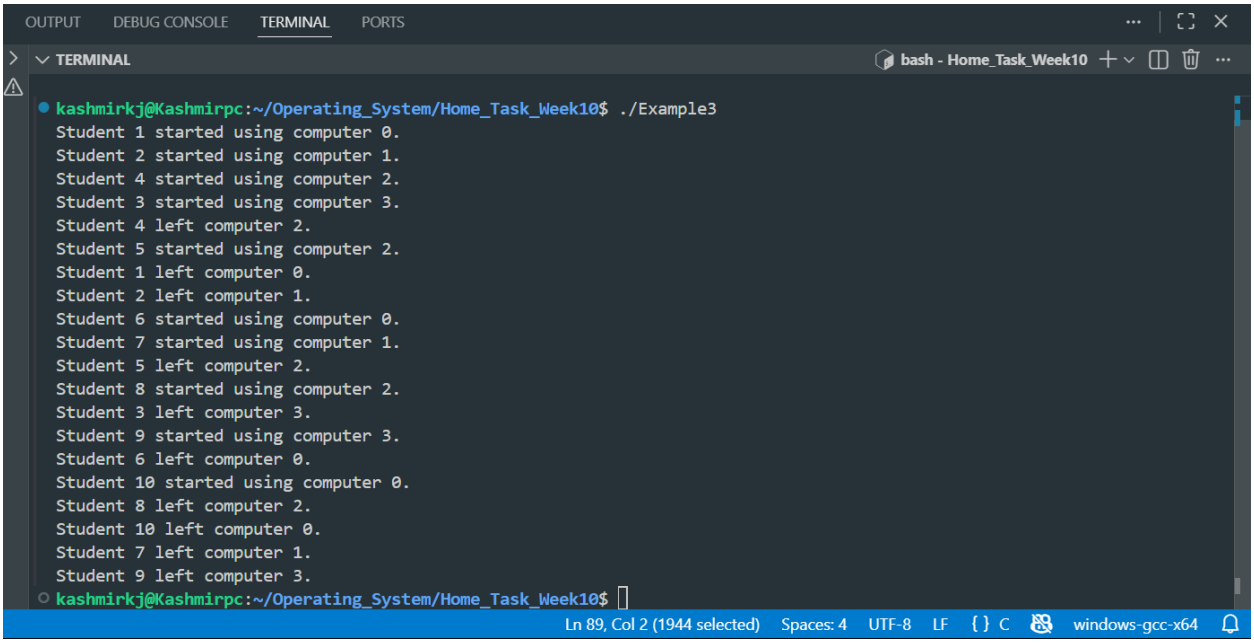
Tasks:

- Semaphore initialized to number of computers
- Track who is using which computer using a shared array
- Protect the array using a mutex

Code:

```c
#include <stdio.h>
#include <pthread.h>
#include <semaphore.h>
#include <unistd.h>
#include <stdlib.h>

#define K 4       // Number of computers in the lab
#define STUDENTS 10

sem_t lab_sem;                // Semaphore for available computers
pthread_mutex_t mutex;        // Protects shared array

int computers[K];             // -1 = free, else student ID

void* student(void* arg) {
    int id = *(int*)arg;


    // Wait for a free computer
    sem_wait(&lab_sem);

    // Find and assign the free computer
    pthread_mutex_lock(&mutex);

    int assigned_computer = -1;
    for (int i = 0; i < K; i++) {
        if (computers[i] == -1) {
            computers[i] = id;
            assigned_computer = i;
            break;
        }
    }

    printf("Student %d started using computer %d.\n", id, assigned_computer);

    pthread_mutex_unlock(&mutex);

    // Use computer for random 1-3 seconds
    int use_time = (rand() % 3) + 1;
    sleep(use_time);

    // Student leaves the computer
    pthread_mutex_lock(&mutex);

    computers[assigned_computer] = -1;
    printf("Student %d left computer %d.\n", id, assigned_computer);

    pthread_mutex_unlock(&mutex);

    // Free a computer
    sem_post(&lab_sem);

    return NULL;
}

int main() {
    pthread_t threads[STUDENTS];
    int ids[STUDENTS];

    srand(time(NULL));

    // Initialize computers array to -1 (all free)
    for (int i = 0; i < K; i++) {
        computers[i] = -1;
    }

    // Initialize semaphore
    sem_init(&lab_sem, 0, K);

    // Initialize mutex
    pthread_mutex_init(&mutex, NULL);

    // Create student threads
    for (int i = 0; i < STUDENTS; i++) {
        ids[i] = i + 1;
        pthread_create(&threads[i], NULL, student, &ids[i]);
    }

    // Join all students
    for (int i = 0; i < STUDENTS; i++) {
        pthread_join(threads[i], NULL);
    }

    // Cleanup
    sem_destroy(&lab_sem);
    pthread_mutex_destroy(&mutex);

    return 0;
}
```

Output:



Q#4

Exercise 4 – Thread Pool / Worker Pool Simulation

Scenario:

A server has **fixed number of worker threads**.
More tasks arrive than workers available.

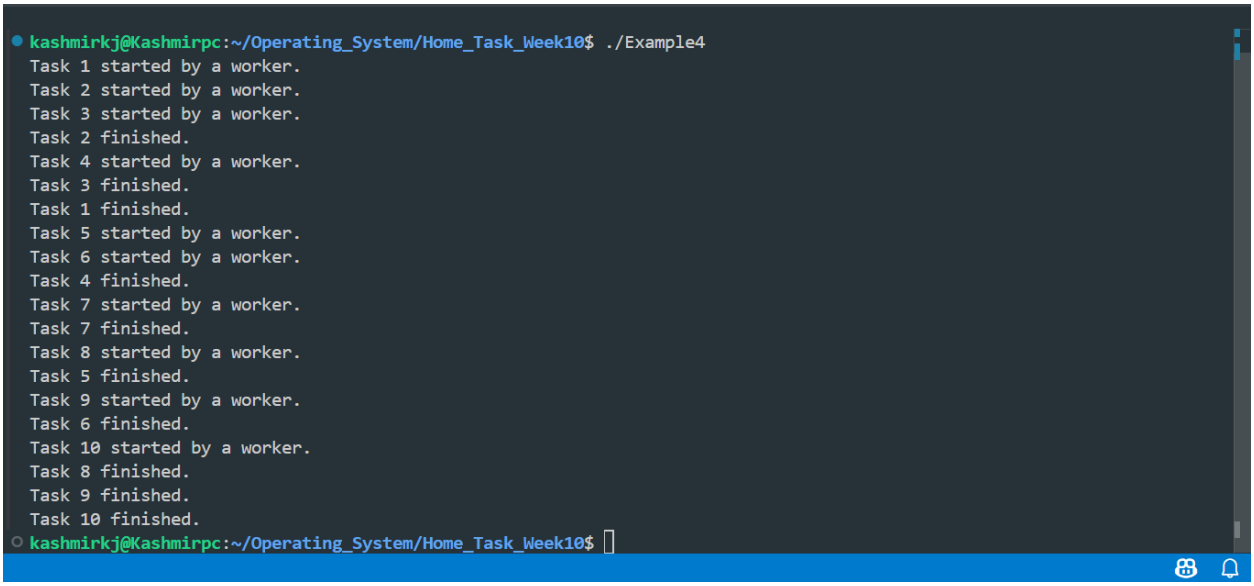
Task:

- Simulate 10 tasks and 3 workers
- Tasks “run” by sleeping for 1–2 seconds
- Semaphore controls worker availability

Code:

```c
#include <stdio.h>
#include <pthread.h>
#include <semaphore.h>
#include <unistd.h>
#include <stdlib.h>

#define WORKERS 3
#define TASKS 10

sem_t worker_sem;           // Controls worker availability

pthread_mutex_t mutex;      // For printing / tracking safely

void* task(void* arg) {
    int id = *(int*)arg;


    // Wait until a worker is free
    sem_wait(&worker_sem);

    pthread_mutex_lock(&mutex);
    printf("Task %d started by a worker.\n", id);
    pthread_mutex_unlock(&mutex);

    // Simulate task work
    int t = (rand() % 2) + 1;
    sleep(t);

    pthread_mutex_lock(&mutex);
    printf("Task %d finished.\n", id);
    pthread_mutex_unlock(&mutex);

    // Release worker
    sem_post(&worker_sem);

    return NULL;
}

int main() {
    pthread_t tasks[TASKS];
    int ids[TASKS];

    srand(time(NULL));

    // Initialize semaphore with 3 workers available
    sem_init(&worker_sem, 0, WORKERS);

    // Initialize mutex
    pthread_mutex_init(&mutex, NULL);

    // Create task threads
    for (int i = 0; i < TASKS; i++) {
        ids[i] = i + 1;
        pthread_create(&tasks[i], NULL, task, &ids[i]);
    }

    // Wait for all tasks to finish
    for (int i = 0; i < TASKS; i++) {
        pthread_join(tasks[i], NULL);
    }

    // Cleanup
    sem_destroy(&worker_sem);
    pthread_mutex_destroy(&mutex);

    return 0;
}
```

Output:



Q#5

Exercise 5 – Car Wash Station

Scenario:

Car wash has **two washing stations**.

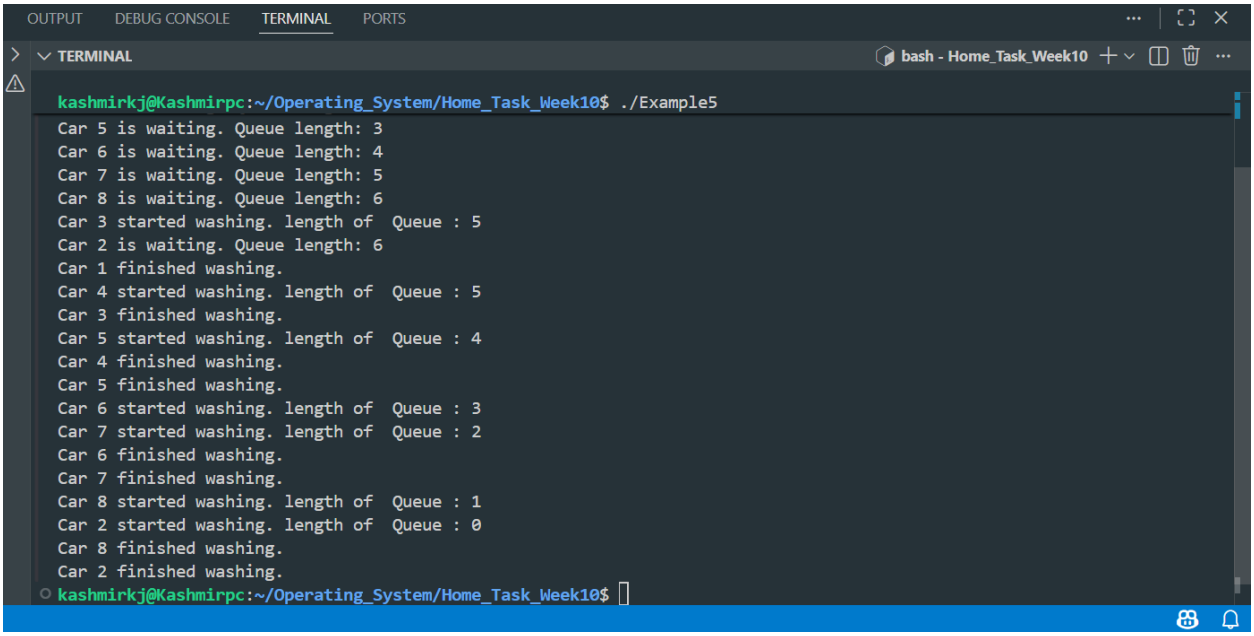
Tasks:

- Use counting semaphore initialized to 2 (number of washing stations)
- Car threads wait for availability
- Cars take 3 seconds to wash
- Track queue lengths (optional)

Code:

```c
#include <stdio.h>
#include <pthread.h>
#include <semaphore.h>
#include <unistd.h>

#define STATIONS 2
#define CARS 8

sem_t wash_sem;               // Controls washing station availability
pthread_mutex_t mutex;        // Protects queue counter

int waiting = 0;              // queue length tracker

void* car(void* arg) {
    int id = *(int*)arg;


    // Car enters waiting queue
    pthread_mutex_lock(&mutex);
    waiting++;
    printf("Car %d is waiting. Queue length: %d\n", id, waiting);
    pthread_mutex_unlock(&mutex);

    // Wait for washing station
    sem_wait(&wash_sem);

    // Car gets a washing station → remove from queue
    pthread_mutex_lock(&mutex);
    waiting--;
    printf("Car %d started washing. length of  Queue : %d\n", id, waiting);
    pthread_mutex_unlock(&mutex);

    // Washing for 3 seconds
    sleep(3);

    printf("Car %d finished washing.\n", id);

    // Free the washing station
    sem_post(&wash_sem);

    return NULL;
}

int main() {
    pthread_t cars[CARS];
    int ids[CARS];

    // Initialize semaphore with 2 stations
    sem_init(&wash_sem, 0, STATIONS);

    // Initialize mutex
    pthread_mutex_init(&mutex, NULL);

    // Create car threads
    for (int i = 0; i < CARS; i++) {
        ids[i] = i + 1;
        pthread_create(&cars[i], NULL, car, &ids[i]);
    }

    // Wait for all cars
    for (int i = 0; i < CARS; i++) {
        pthread_join(cars[i], NULL);
    }

    // Cleanup
    sem_destroy(&wash_sem);
    pthread_mutex_destroy(&mutex);

    return 0;
}
```

Output:



Q#6

Exercise 6 – Traffic Bridge Control (Single-Lane Bridge)

Scenario:

Only **3 cars** are allowed on the bridge at once.

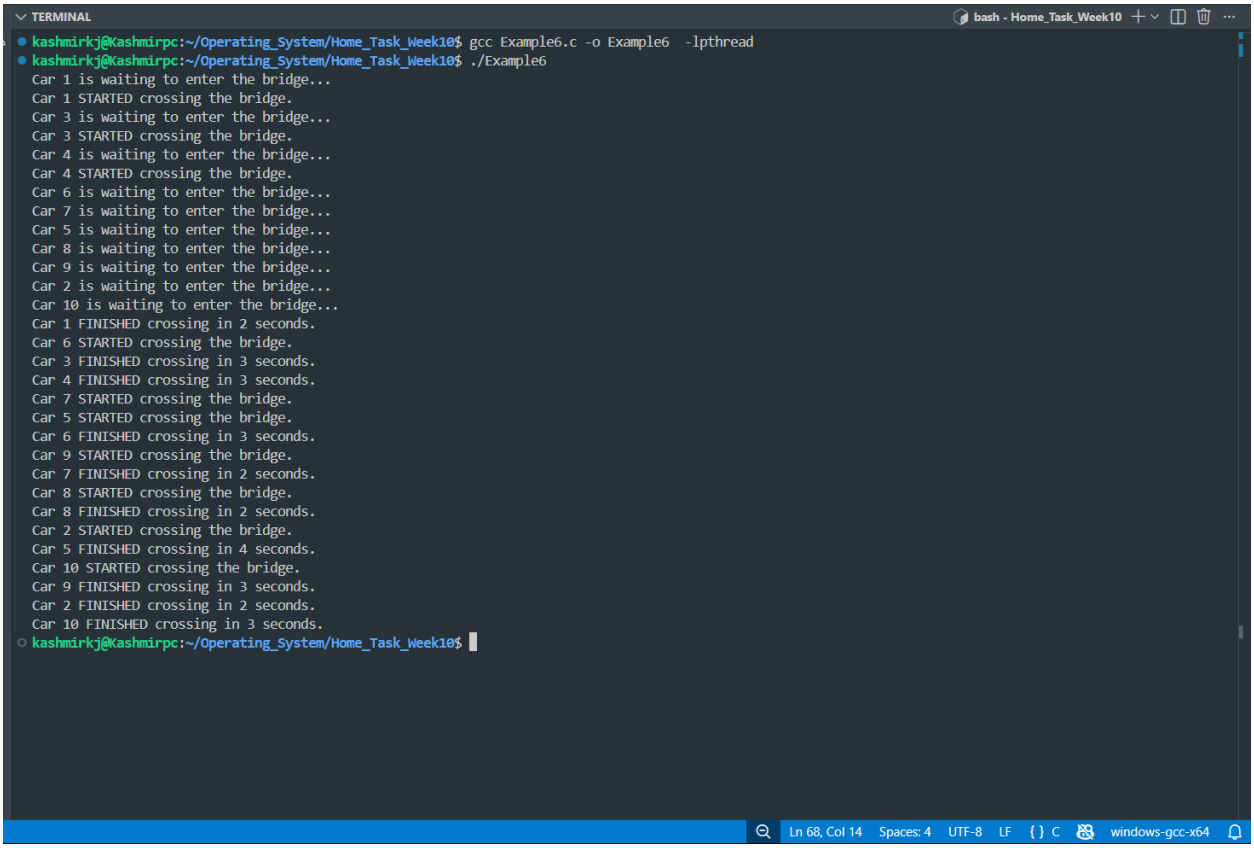
Tasks:

- Semaphore for max cars
- Mutex for printing
- Add random crossing times

Code:

```c
#include <stdio.h>
#include <pthread.h>
#include <semaphore.h>
#include <unistd.h>
#include <stdlib.h>

#define MAX_ON_BRIDGE 3
#define TOTAL_CARS 10

sem_t bridge_sem;            // Controls number of cars on bridge
pthread_mutex_t print_mutex; // For clean printing

void* car(void* arg) {
    int id = *(int*)arg;

    pthread_mutex_lock(&print_mutex);
    printf("Car %d is waiting to enter the bridge...\n", id);
    pthread_mutex_unlock(&print_mutex);

    // Wait for permission to enter bridge
    sem_wait(&bridge_sem);

    pthread_mutex_lock(&print_mutex);
    printf("Car %d STARTED crossing the bridge.\n", id);
    pthread_mutex_unlock(&print_mutex);

    // Random crossing time (1-4 seconds)
    int t = (rand() % 4) + 1;
    sleep(t);

    pthread_mutex_lock(&print_mutex);
    printf("Car %d FINISHED crossing in %d seconds.\n", id, t);
    pthread_mutex_unlock(&print_mutex);

    // Free space on the bridge
    sem_post(&bridge_sem);

    return NULL;
}

int main() {
    pthread_t cars[TOTAL_CARS];
    int ids[TOTAL_CARS];

    srand(time(NULL));

    // Semaphore allows 3 cars at a time
    sem_init(&bridge_sem, 0, MAX_ON_BRIDGE);

    // Initialize mutex
    pthread_mutex_init(&print_mutex, NULL);

    // Create car threads
    for (int i = 0; i < TOTAL_CARS; i++) {
        ids[i] = i + 1;
        pthread_create(&cars[i], NULL, car, &ids[i]);
    }

    // Wait for all cars
    for (int i = 0; i < TOTAL_CARS; i++) {
        pthread_join(cars[i], NULL);
    }

    // Cleanup
    sem_destroy(&bridge_sem);
    pthread_mutex_destroy(&print_mutex);

    return 0;
}
```

Output:


TERMINAL
bash - Home_Task_Week10
kashmirkj@Kashmirpc:~/Operating_System/Home_Task_Week10$ gcc Example6.c -o Example6 -lpthread
kashmirkj@Kashmirpc:~/Operating_System/Home_Task_Week10$ ./Example6
Car 1 is waiting to enter the bridge...
Car 1 STARTED crossing the bridge.
Car 3 is waiting to enter the bridge...
Car 3 STARTED crossing the bridge.
Car 4 is waiting to enter the bridge...
Car 4 STARTED crossing the bridge.
Car 6 is waiting to enter the bridge...
Car 7 is waiting to enter the bridge...
Car 5 is waiting to enter the bridge...
Car 8 is waiting to enter the bridge...
Car 9 is waiting to enter the bridge...
Car 2 is waiting to enter the bridge...
Car 10 is waiting to enter the bridge...
Car 1 FINISHED crossing in 2 seconds.
Car 6 STARTED crossing the bridge.
Car 3 FINISHED crossing in 3 seconds.
Car 4 FINISHED crossing in 3 seconds.
Car 7 STARTED crossing the bridge.
Car 5 STARTED crossing the bridge.
Car 6 FINISHED crossing in 3 seconds.
Car 9 STARTED crossing the bridge.
Car 7 FINISHED crossing in 2 seconds.
Car 8 STARTED crossing the bridge.
Car 8 FINISHED crossing in 2 seconds.
Car 2 STARTED crossing the bridge.
Car 5 FINISHED crossing in 4 seconds.
Car 10 STARTED crossing the bridge.
Car 9 FINISHED crossing in 3 seconds.
Car 2 FINISHED crossing in 2 seconds.
Car 10 FINISHED crossing in 3 seconds.
kashmirkj@Kashmirpc:~/Operating_System/Home_Task_Week10$
Ln 68, Col 14
Spaces: 4
UTF-8
LF
{ } C
windows-gcc-x64