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-hometasks

Semester:

5th

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.

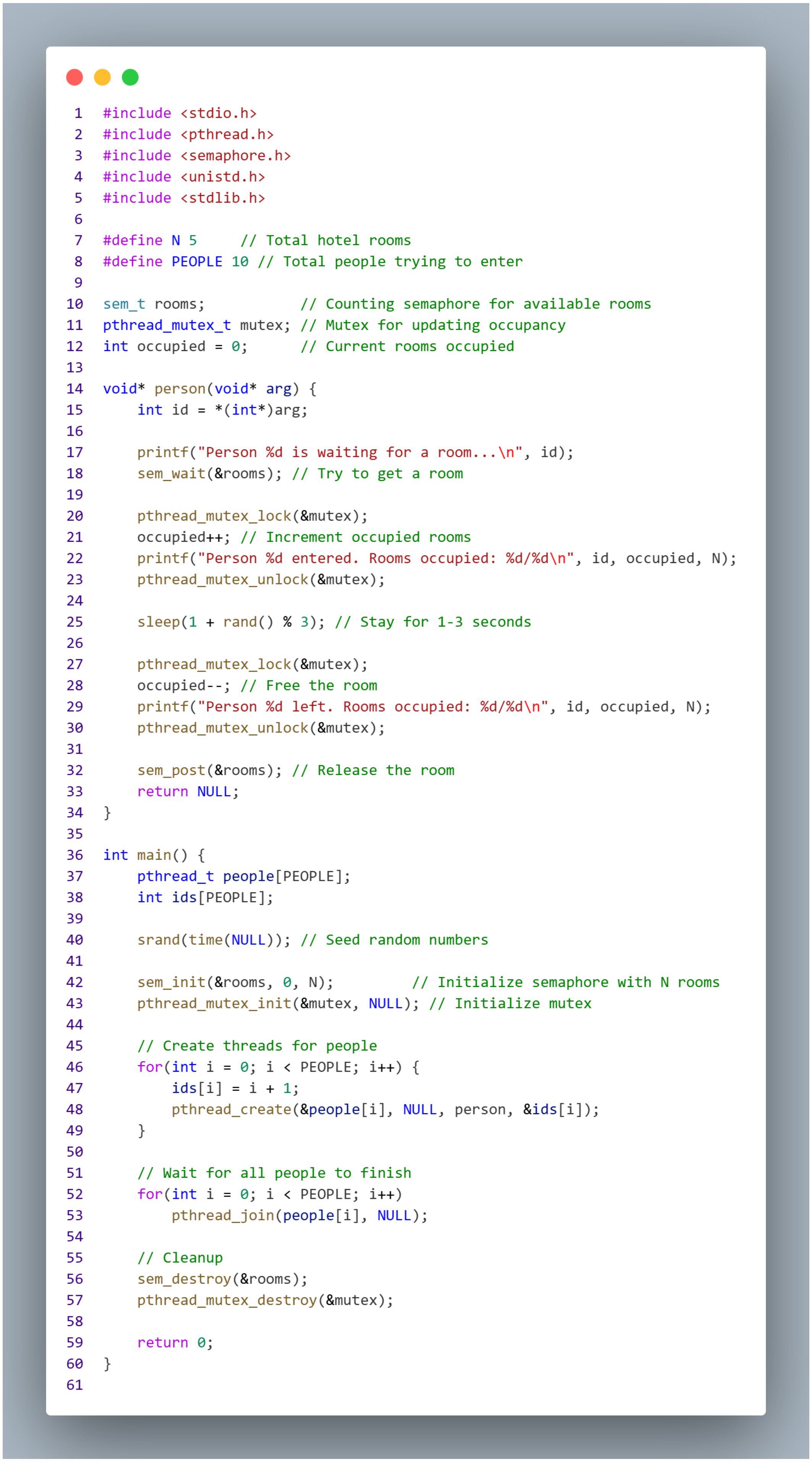
Tasks:

1. Use a counting semaphore initialized to N

2. Each person (thread) enters, stays for 1–3 seconds, leaves

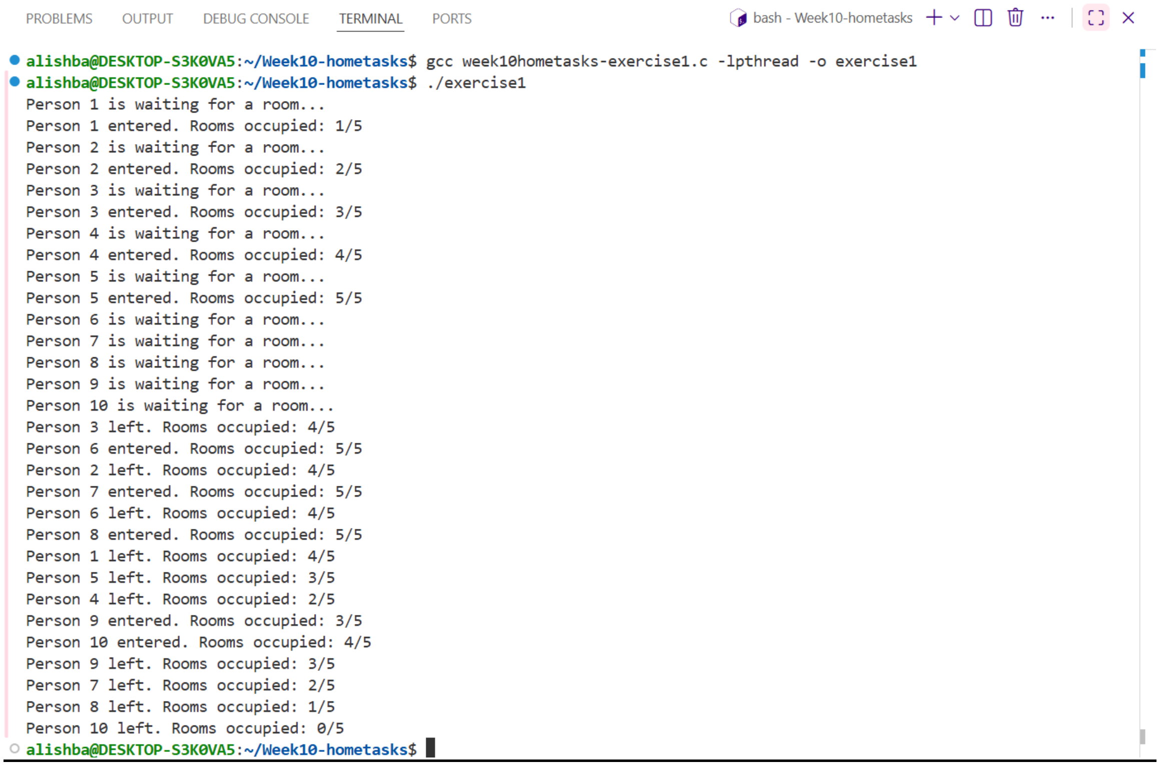
3. Print: “Person X entered” “Person X left”

4. Show how many rooms are currently occupied



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

#define N 5     // Total hotel rooms
#define PEOPLE 10 // Total people trying to enter

sem_t rooms;           // Counting semaphore for available rooms
pthread_mutex_t mutex; // Mutex for updating occupancy
int occupied = 0;      // Current rooms occupied

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

    printf("Person %d is waiting for a room...\n", id);
    sem_wait(&rooms); // Try to get a room

    pthread_mutex_lock(&mutex);
    occupied++; // Increment occupied rooms
    printf("Person %d entered. Rooms occupied: %d/%d\n", id, occupied, N);
    pthread_mutex_unlock(&mutex);

    sleep(1 + rand() % 3); // Stay for 1-3 seconds

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

    sem_post(&rooms); // Release the room
    return NULL;
}

int main() {
    pthread_t people[PEOPLE];
    int ids[PEOPLE];

    srand(time(NULL)); // Seed random numbers

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

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

    // Wait for all people to finish
    for(int i = 0; i < PEOPLE; i++)
        pthread_join(people[i], NULL);

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

    return 0;
}

```

Output:



```
alishba@DESKTOP-S3K0VA5:~/Week10-hometasks$ gcc week10hometasks-exercise1.c -lpthread -o exercise1
alishba@DESKTOP-S3K0VA5:~/Week10-hometasks$ ./exercise1
Person 1 is waiting for a room...
Person 1 entered. Rooms occupied: 1/5
Person 2 is waiting for a room...
Person 2 entered. Rooms occupied: 2/5
Person 3 is waiting for a room...
Person 3 entered. Rooms occupied: 3/5
Person 4 is waiting for a room...
Person 4 entered. Rooms occupied: 4/5
Person 5 is waiting for a room...
Person 5 entered. Rooms occupied: 5/5
Person 6 is waiting for a room...
Person 7 is waiting for a room...
Person 8 is waiting for a room...
Person 9 is waiting for a room...
Person 10 is waiting for a room...
Person 3 left. Rooms occupied: 4/5
Person 6 entered. Rooms occupied: 5/5
Person 2 left. Rooms occupied: 4/5
Person 7 entered. Rooms occupied: 5/5
Person 6 left. Rooms occupied: 4/5
Person 8 entered. Rooms occupied: 5/5
Person 1 left. Rooms occupied: 4/5
Person 5 left. Rooms occupied: 3/5
Person 4 left. Rooms occupied: 2/5
Person 9 entered. Rooms occupied: 3/5
Person 10 entered. Rooms occupied: 4/5
Person 9 left. Rooms occupied: 3/5
Person 7 left. Rooms occupied: 2/5
Person 8 left. Rooms occupied: 1/5
Person 10 left. Rooms occupied: 0/5
alishba@DESKTOP-S3K0VA5:~/Week10-hometasks$ 
```

Exercise 2 – Download Manager Simulation

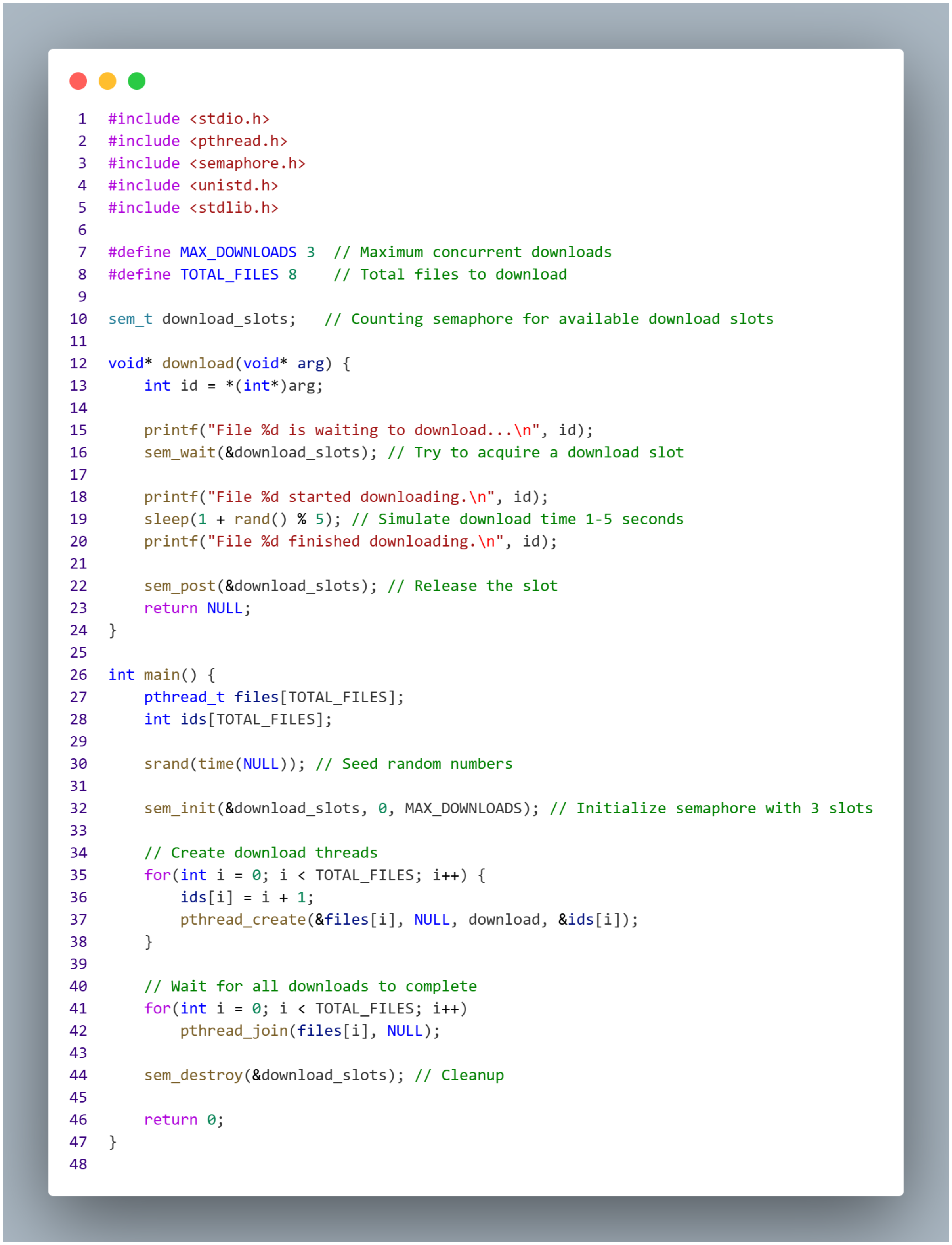
Scenario:

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

Tasks:

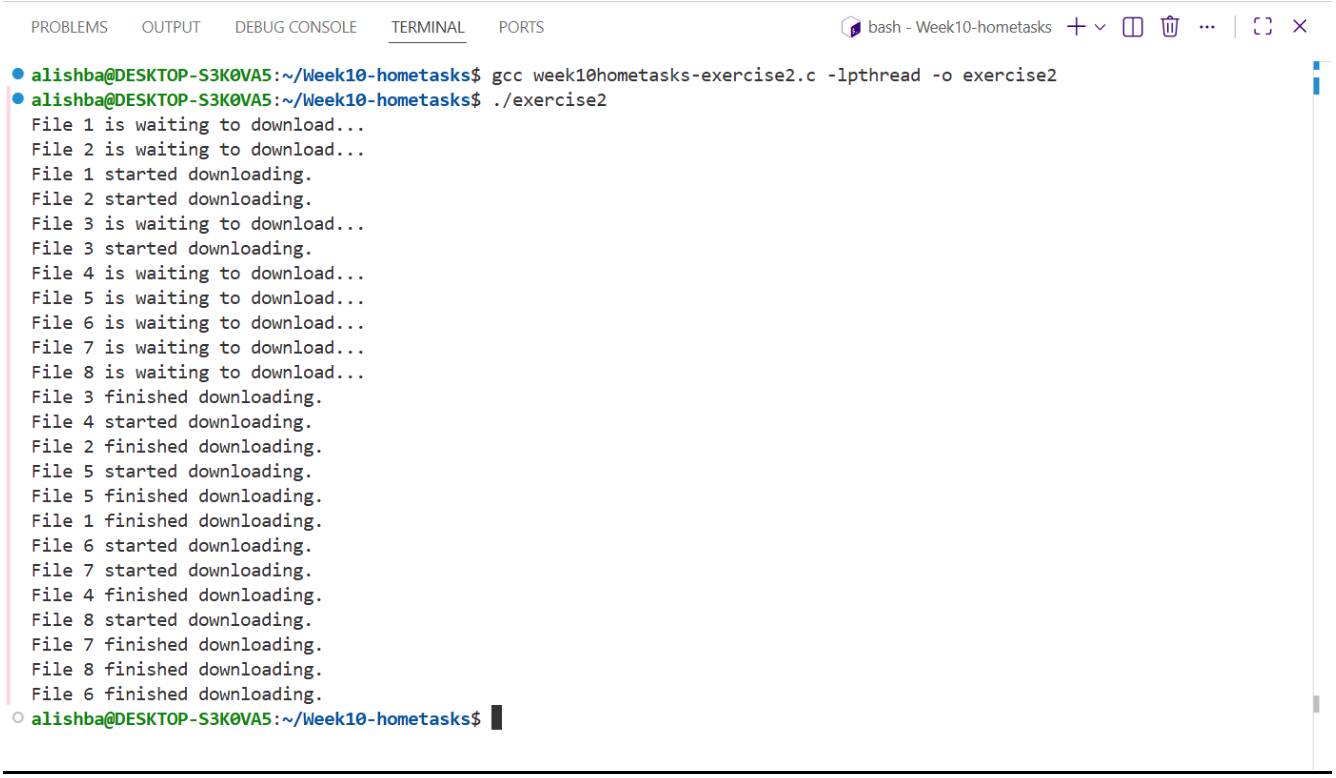
- Create 8 download threads
- Use a counting semaphore with value = 3
- Each download takes random 1–5 seconds
- Print messages for start/end of each download

Code:



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

#define MAX_DOWNLOADS 3  // Maximum concurrent downloads
#define TOTAL_FILES 8    // Total files to download

sem_t download_slots;   // Counting semaphore for available download slots

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

    printf("File %d is waiting to download...\n", id);
    sem_wait(&download_slots); // Try to acquire a download slot

    printf("File %d started downloading.\n", id);
    sleep(1 + rand() % 5); // Simulate download time 1-5 seconds
    printf("File %d finished downloading.\n", id);

    sem_post(&download_slots); // Release the slot
    return NULL;
}

int main() {
    pthread_t files[TOTAL_FILES];
    int ids[TOTAL_FILES];

    srand(time(NULL)); // Seed random numbers

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

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

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

    sem_destroy(&download_slots); // Cleanup

    return 0;
}

```

Output:



```
alishba@DESKTOP-S3K0VA5:~/Week10-hometasks$ gcc week10hometasks-exercise2.c -lpthread -o exercise2
alishba@DESKTOP-S3K0VA5:~/Week10-hometasks$ ./exercise2
File 1 is waiting to download...
File 2 is waiting to download...
File 1 started downloading.
File 2 started downloading.
File 3 is waiting to download...
File 3 started downloading.
File 4 is waiting to download...
File 5 is waiting to download...
File 6 is waiting to download...
File 7 is waiting to download...
File 8 is waiting to download...
File 3 finished downloading.
File 4 started downloading.
File 2 finished downloading.
File 5 started downloading.
File 5 finished downloading.
File 1 finished downloading.
File 6 started downloading.
File 7 started downloading.
File 4 finished downloading.
File 8 started downloading.
File 7 finished downloading.
File 8 finished downloading.
File 6 finished downloading.
alishba@DESKTOP-S3K0VA5:~/Week10-hometasks$ 
```

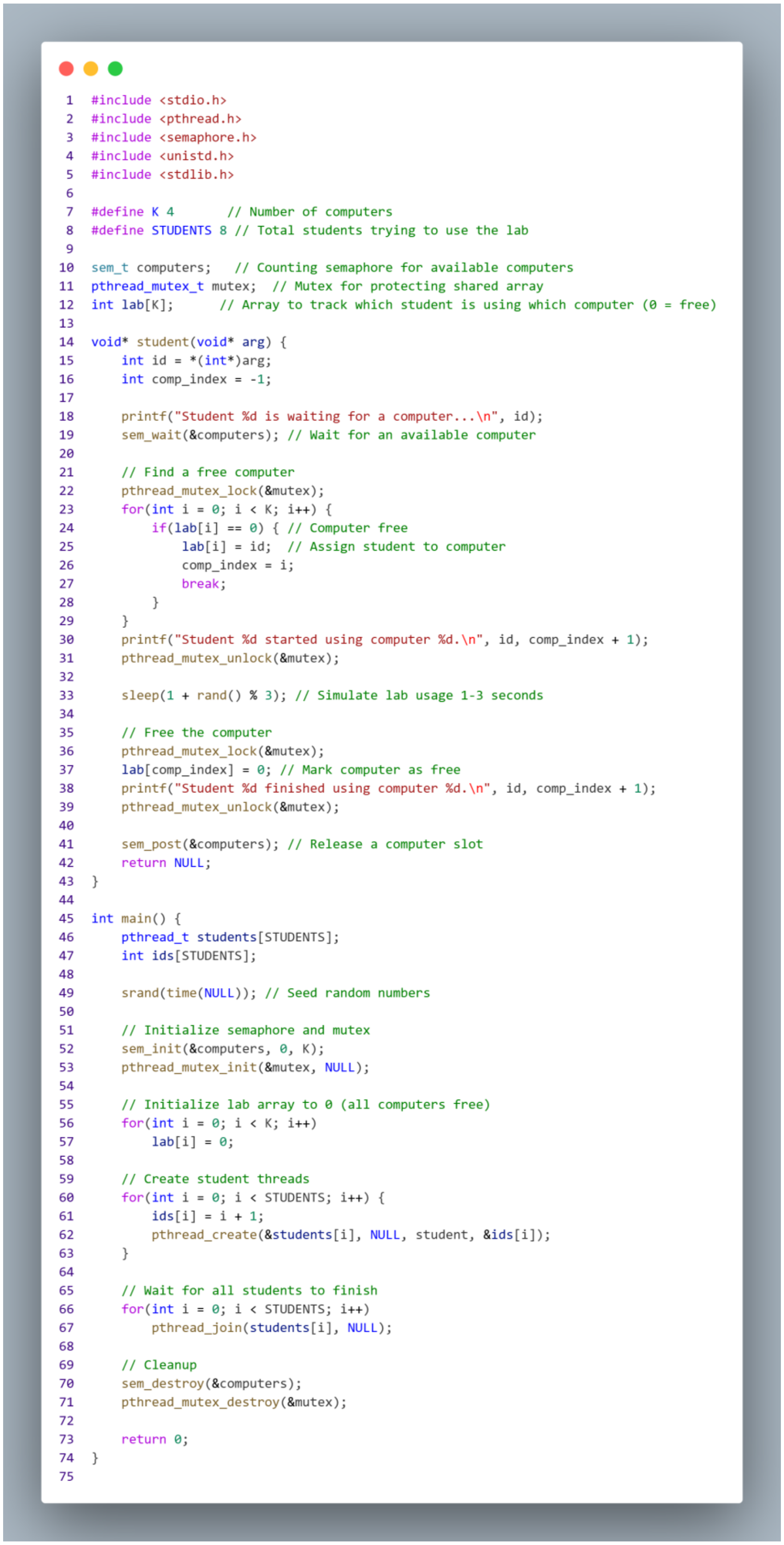
Exercise 3 – Library Computer Lab Access

Scenario:

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

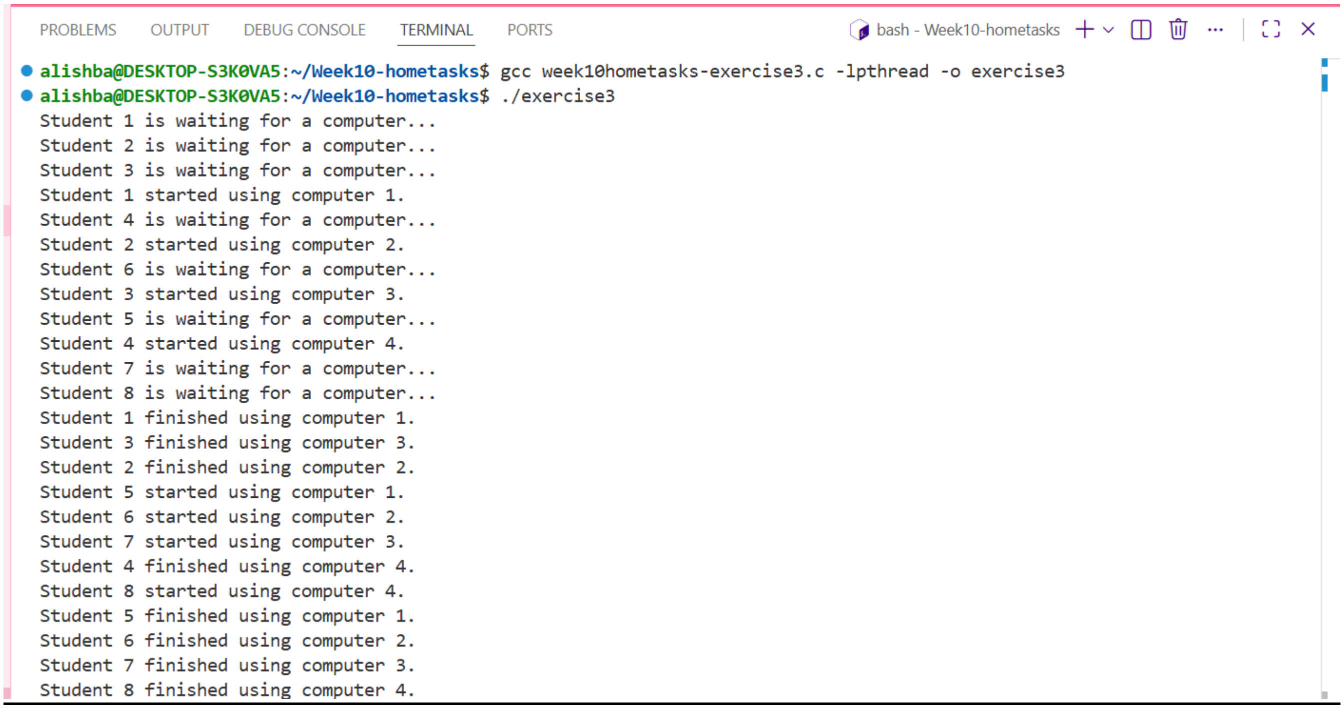
Tasks:

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



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

#define K 4       // Number of computers
#define STUDENTS 8 // Total students trying to use the lab

sem_t computers;   // Counting semaphore for available computers
pthread_mutex_t mutex;  // Mutex for protecting shared array
int lab[K];      // Array to track which student is using which computer (0 = free)

void* student(void* arg) {
    int id = *(int*)arg;
    int comp_index = -1;

    printf("Student %d is waiting for a computer...\n", id);
    sem_wait(&computers); // Wait for an available computer

    // Find a free computer
    pthread_mutex_lock(&mutex);
    for(int i = 0; i < K; i++) {
        if(lab[i] == 0) { // Computer free
            lab[i] = id;  // Assign student to computer
            comp_index = i;
            break;
        }
    }
    printf("Student %d started using computer %d.\n", id, comp_index + 1);
    pthread_mutex_unlock(&mutex);

    sleep(1 + rand() % 3); // Simulate lab usage 1-3 seconds

    // Free the computer
    pthread_mutex_lock(&mutex);
    lab[comp_index] = 0; // Mark computer as free
    printf("Student %d finished using computer %d.\n", id, comp_index + 1);
    pthread_mutex_unlock(&mutex);

    sem_post(&computers); // Release a computer slot
    return NULL;
}

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

    srand(time(NULL)); // Seed random numbers

    // Initialize semaphore and mutex
    sem_init(&computers, 0, K);
    pthread_mutex_init(&mutex, NULL);

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

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

    // Wait for all students to finish
    for(int i = 0; i < STUDENTS; i++)
        pthread_join(students[i], NULL);

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

    return 0;
}
```

Output:



```
● alishba@DESKTOP-S3K0VA5:~/Week10-hometasks$ gcc week10hometasks-exercise3.c -lpthread -o exercise3
● alishba@DESKTOP-S3K0VA5:~/Week10-hometasks$ ./exercise3
  Student 1 is waiting for a computer...
  Student 2 is waiting for a computer...
  Student 3 is waiting for a computer...
  Student 1 started using computer 1.
  Student 4 is waiting for a computer...
  Student 2 started using computer 2.
  Student 6 is waiting for a computer...
  Student 3 started using computer 3.
  Student 5 is waiting for a computer...
  Student 4 started using computer 4.
  Student 7 is waiting for a computer...
  Student 8 is waiting for a computer...
  Student 1 finished using computer 1.
  Student 3 finished using computer 3.
  Student 2 finished using computer 2.
  Student 5 started using computer 1.
  Student 6 started using computer 2.
  Student 7 started using computer 3.
  Student 4 finished using computer 4.
  Student 8 started using computer 4.
  Student 5 finished using computer 1.
  Student 6 finished using computer 2.
  Student 7 finished using computer 3.
  Student 8 finished using computer 4.
```

Exercise 4 – Thread Pool / Worker Pool Simulation

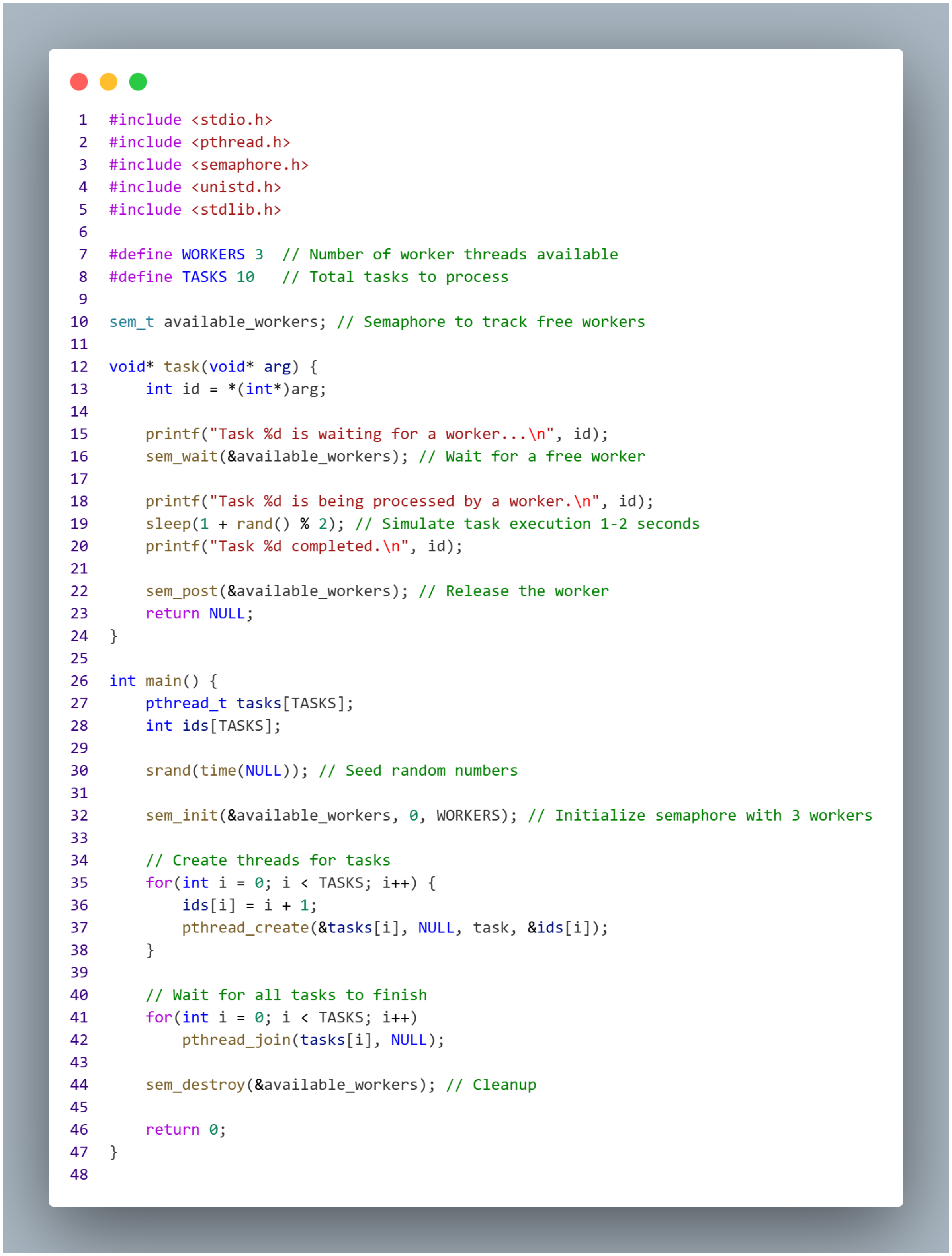
Scenario:

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

Tasks:

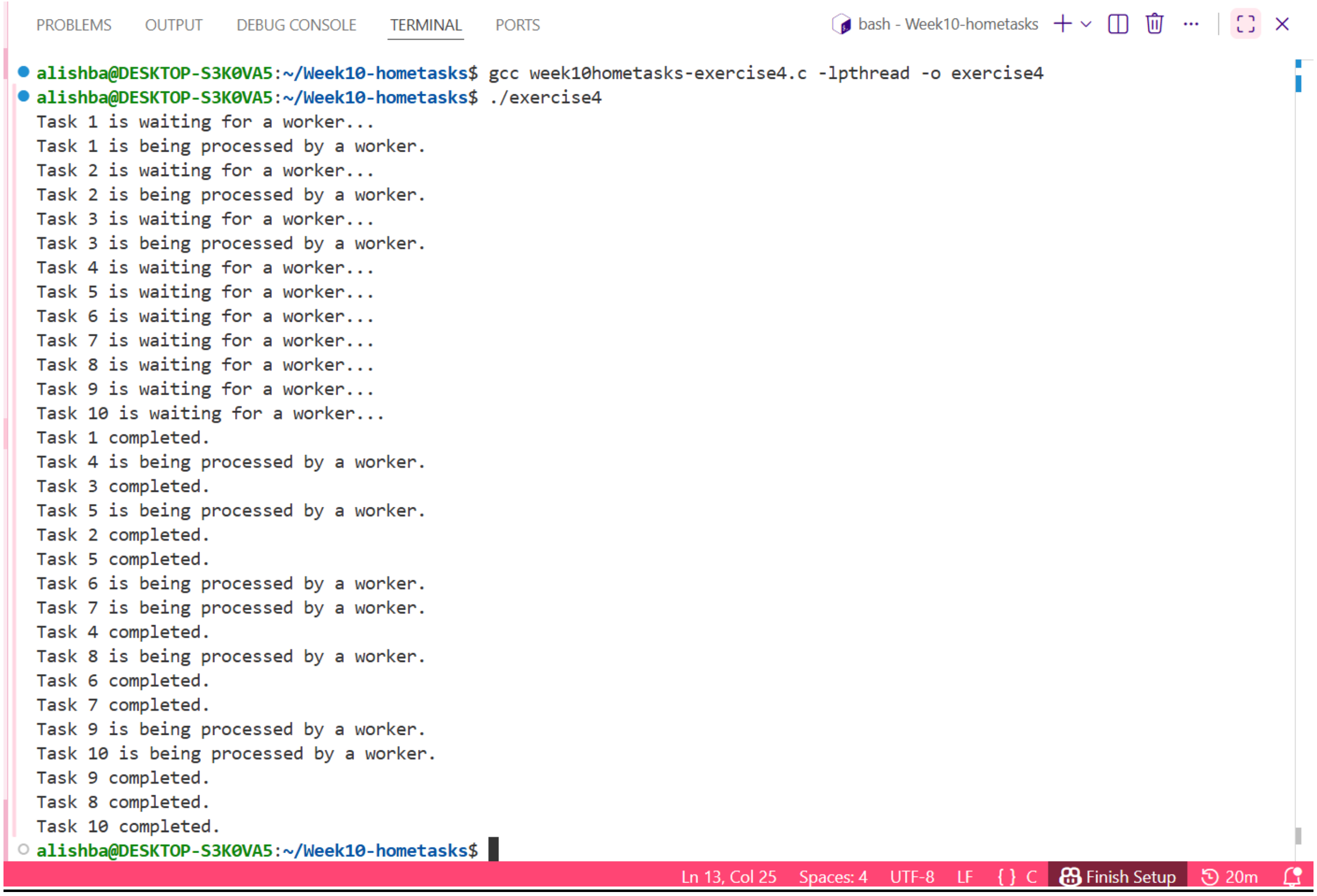
- Simulate 10 tasks and 3 workers
- Tasks “run” by sleeping for 1–2 seconds
- Semaphore controls worker availability

Code:



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

#define WORKERS 3  // Number of worker threads available
#define TASKS 10   // Total tasks to process

sem_t available_workers; // Semaphore to track free workers

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

    printf("Task %d is waiting for a worker...\n", id);
    sem_wait(&available_workers); // Wait for a free worker

    printf("Task %d is being processed by a worker.\n", id);
    sleep(1 + rand() % 2); // Simulate task execution 1-2 seconds
    printf("Task %d completed.\n", id);

    sem_post(&available_workers); // Release the worker
    return NULL;
}

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

    srand(time(NULL)); // Seed random numbers

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

    // Create threads for tasks
    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);

    sem_destroy(&available_workers); // Cleanup

    return 0;
}

```

Output:



```
● alishba@DESKTOP-S3K0VA5:~/Week10-hometasks$ gcc week10hometasks-exercise4.c -lpthread -o exercise4
● alishba@DESKTOP-S3K0VA5:~/Week10-hometasks$ ./exercise4
  Task 1 is waiting for a worker...
  Task 1 is being processed by a worker.
  Task 2 is waiting for a worker...
  Task 2 is being processed by a worker.
  Task 3 is waiting for a worker...
  Task 3 is being processed by a worker.
  Task 4 is waiting for a worker...
  Task 5 is waiting for a worker...
  Task 6 is waiting for a worker...
  Task 7 is waiting for a worker...
  Task 8 is waiting for a worker...
  Task 9 is waiting for a worker...
  Task 10 is waiting for a worker...
  Task 1 completed.
  Task 4 is being processed by a worker.
  Task 3 completed.
  Task 5 is being processed by a worker.
  Task 2 completed.
  Task 5 completed.
  Task 6 is being processed by a worker.
  Task 7 is being processed by a worker.
  Task 4 completed.
  Task 8 is being processed by a worker.
  Task 6 completed.
  Task 7 completed.
  Task 9 is being processed by a worker.
  Task 10 is being processed by a worker.
  Task 9 completed.
  Task 8 completed.
  Task 10 completed.
○ alishba@DESKTOP-S3K0VA5:~/Week10-hometasks$ 
```

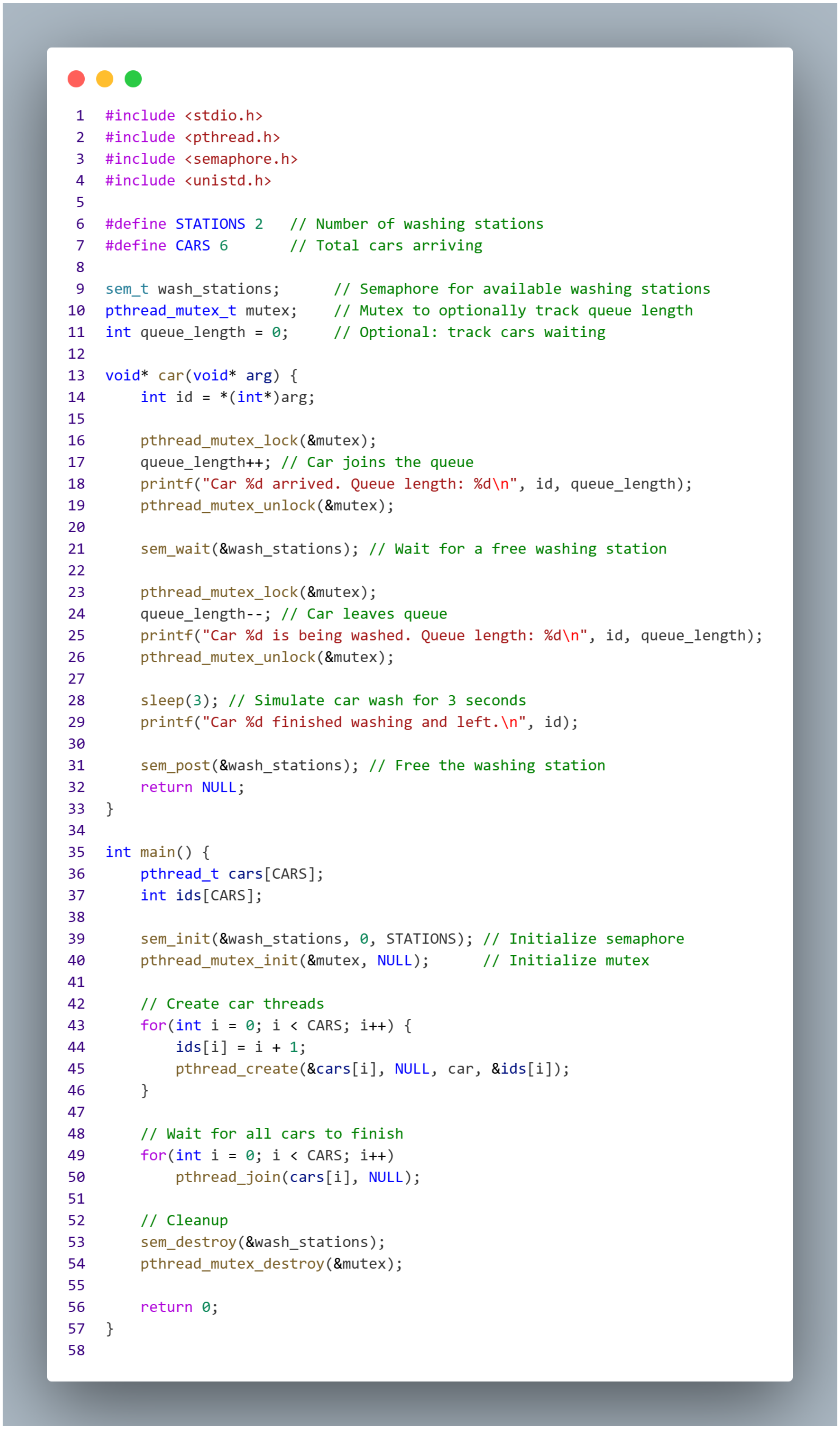
Exercise 5 - Car Wash Station

Scenario:

Car wash has two washing stations.

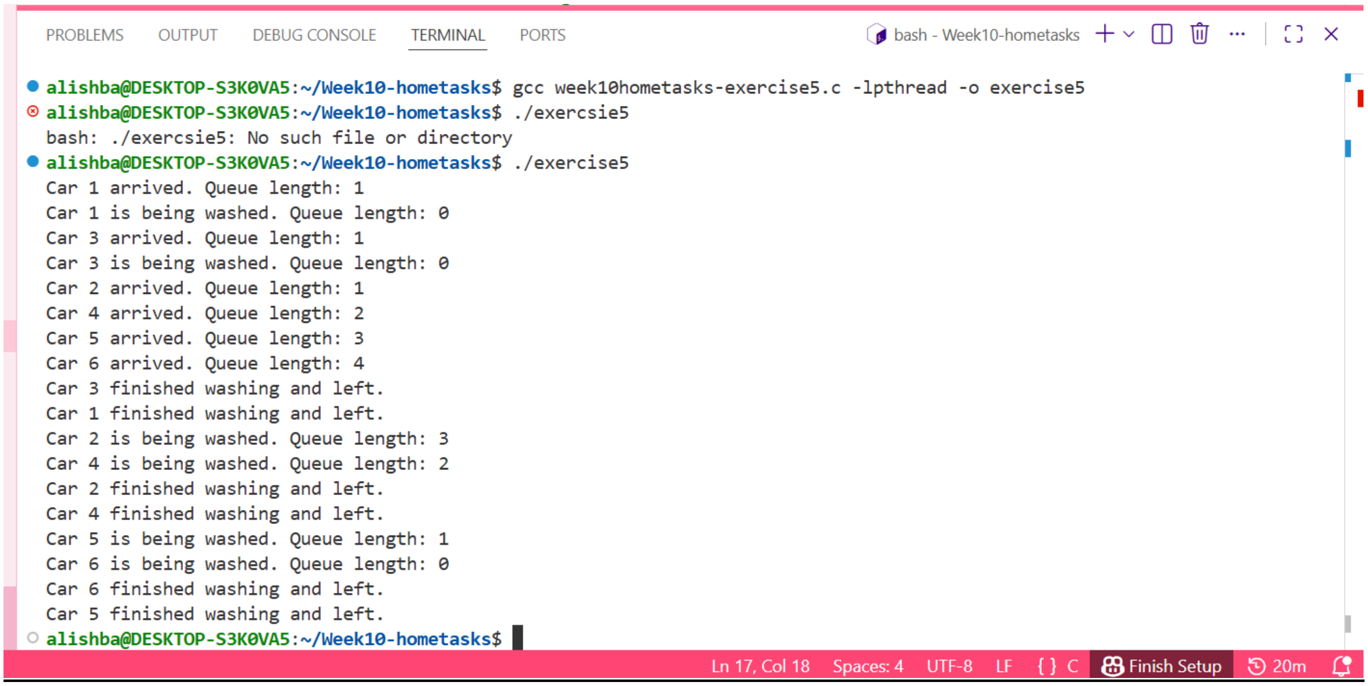
Tasks:

- 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)



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

#define STATIONS 2   // Number of washing stations
#define CARS 6       // Total cars arriving

sem_t wash_stations;      // Semaphore for available washing stations
pthread_mutex_t mutex;    // Mutex to optionally track queue length
int queue_length = 0;     // Optional: track cars waiting

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

    pthread_mutex_lock(&mutex);
    queue_length++; // Car joins the queue
    printf("Car %d arrived. Queue length: %d\n", id, queue_length);
    pthread_mutex_unlock(&mutex);

    sem_wait(&wash_stations); // Wait for a free washing station

    pthread_mutex_lock(&mutex);
    queue_length--; // Car leaves queue
    printf("Car %d is being washed. Queue length: %d\n", id, queue_length);
    pthread_mutex_unlock(&mutex);

    sleep(3); // Simulate car wash for 3 seconds
    printf("Car %d finished washing and left.\n", id);

    sem_post(&wash_stations); // Free the washing station
    return NULL;
}

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

    sem_init(&wash_stations, 0, STATIONS); // Initialize semaphore
    pthread_mutex_init(&mutex, NULL);      // Initialize mutex

    // 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 to finish
    for(int i = 0; i < CARS; i++)
        pthread_join(cars[i], NULL);

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

    return 0;
}
```

Output:



```
alishba@DESKTOP-S3K0VA5:~/Week10-hometasks$ gcc week10hometasks-exercise5.c -lpthread -o exercise5
alishba@DESKTOP-S3K0VA5:~/Week10-hometasks$ ./exercsie5
bash: ./exercsie5: No such file or directory
alishba@DESKTOP-S3K0VA5:~/Week10-hometasks$ ./exercise5
Car 1 arrived. Queue length: 1
Car 1 is being washed. Queue length: 0
Car 3 arrived. Queue length: 1
Car 3 is being washed. Queue length: 0
Car 2 arrived. Queue length: 1
Car 4 arrived. Queue length: 2
Car 5 arrived. Queue length: 3
Car 6 arrived. Queue length: 4
Car 3 finished washing and left.
Car 1 finished washing and left.
Car 2 is being washed. Queue length: 3
Car 4 is being washed. Queue length: 2
Car 2 finished washing and left.
Car 4 finished washing and left.
Car 5 is being washed. Queue length: 1
Car 6 is being washed. Queue length: 0
Car 6 finished washing and left.
Car 5 finished washing and left.
alishba@DESKTOP-S3K0VA5:~/Week10-hometasks$ 
```

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

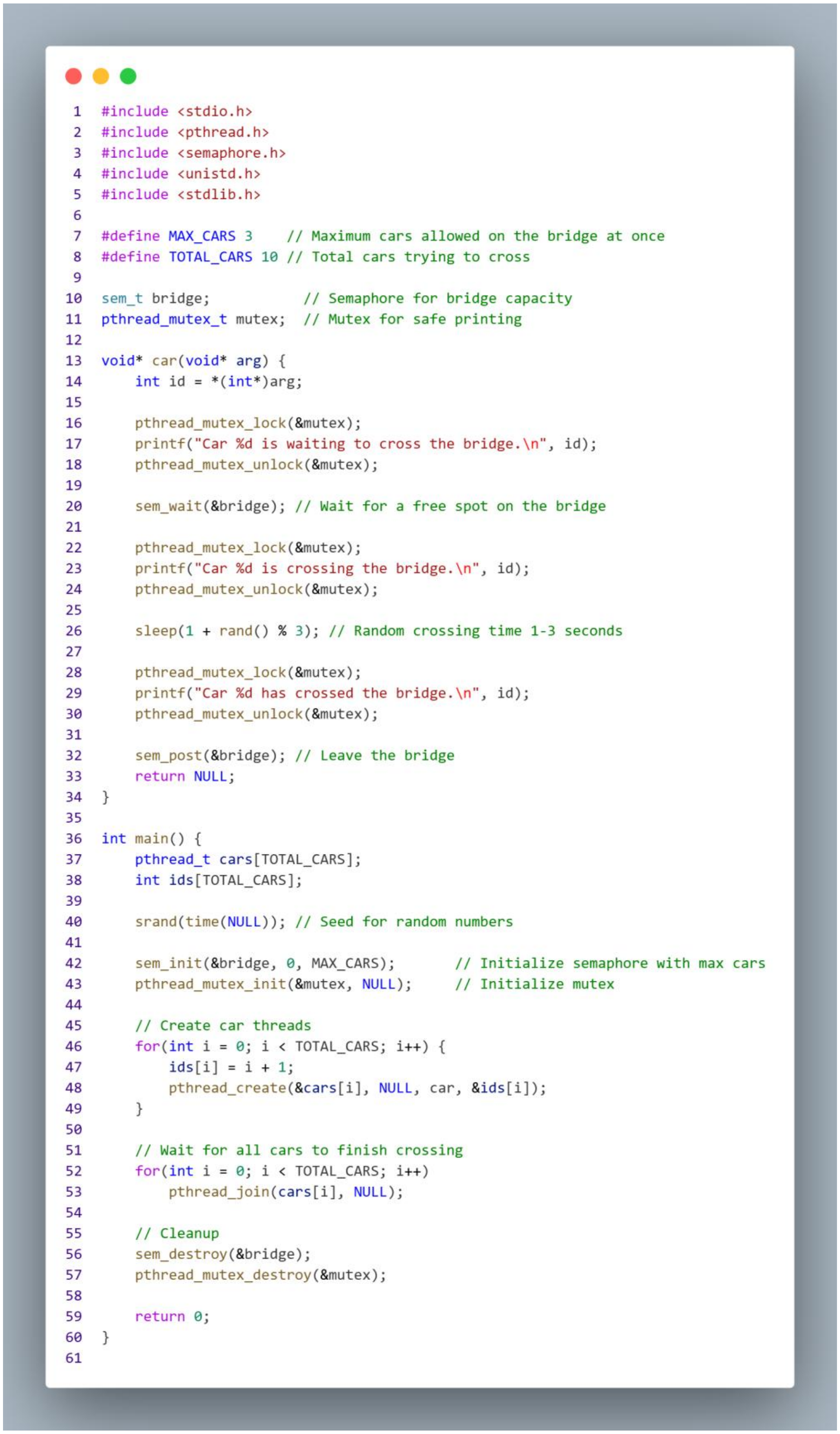
Scenario:

Only 3 cars are allowed on the bridge at once.

Tasks:

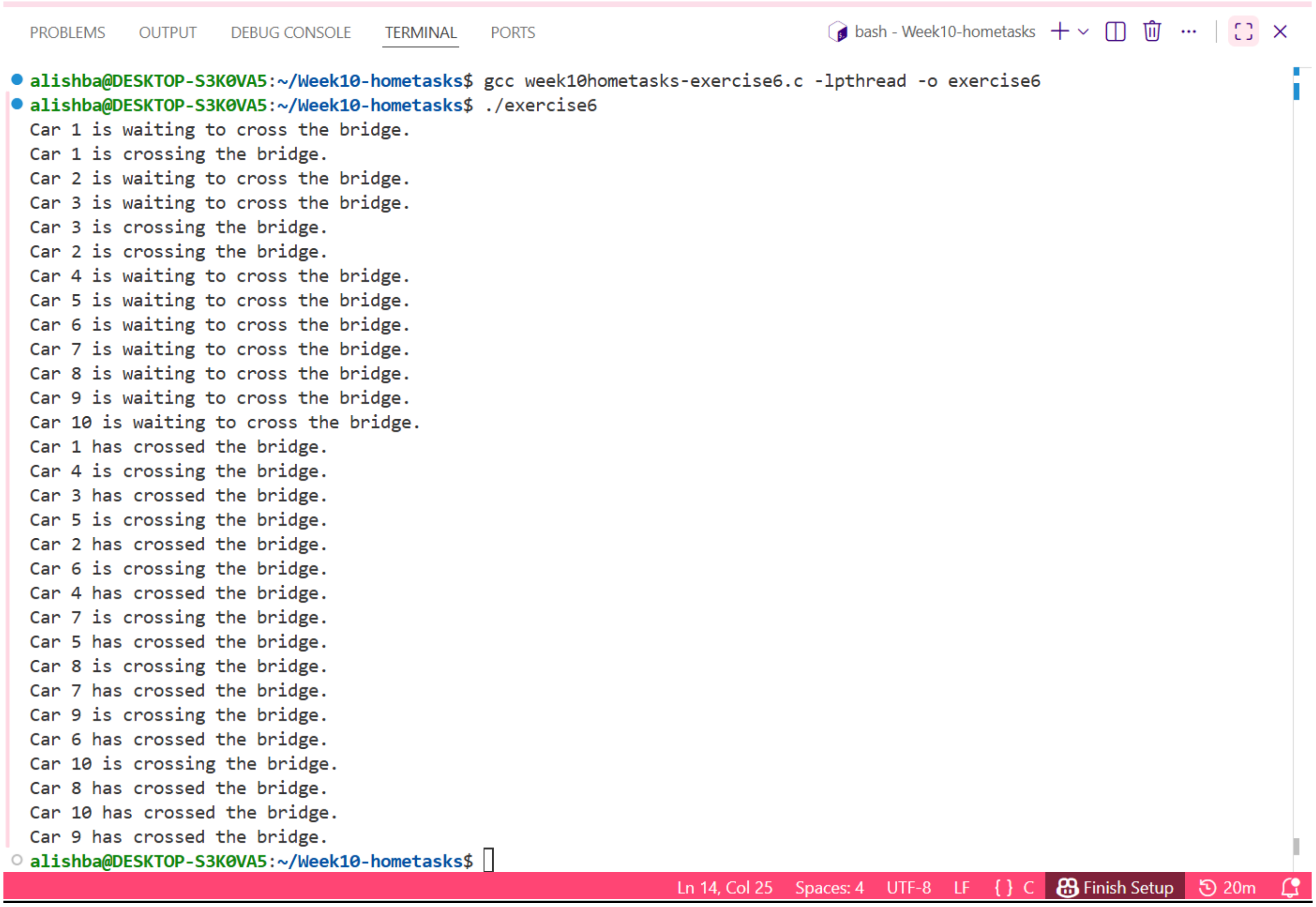
- Semaphore for max cars
- Mutex for printing
- Add random crossing times

Code:



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

#define MAX_CARS 3    // Maximum cars allowed on the bridge at once
#define TOTAL_CARS 10 // Total cars trying to cross

sem_t bridge;           // Semaphore for bridge capacity
pthread_mutex_t mutex;  // Mutex for safe printing

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

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

    sem_wait(&bridge); // Wait for a free spot on the bridge

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

    sleep(1 + rand() % 3); // Random crossing time 1-3 seconds

    pthread_mutex_lock(&mutex);
    printf("Car %d has crossed the bridge.\n", id);
    pthread_mutex_unlock(&mutex);

    sem_post(&bridge); // Leave the bridge
    return NULL;
}

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

    srand(time(NULL)); // Seed for random numbers

    sem_init(&bridge, 0, MAX_CARS);       // Initialize semaphore with max cars
    pthread_mutex_init(&mutex, NULL);     // Initialize mutex

    // 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 to finish crossing
    for(int i = 0; i < TOTAL_CARS; i++)
        pthread_join(cars[i], NULL);

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

    return 0;
}
```

Output:



```
alishba@DESKTOP-S3K0VA5:~/Week10-hometasks$ gcc week10hometasks-exercise6.c -lpthread -o exercise6
alishba@DESKTOP-S3K0VA5:~/Week10-hometasks$ ./exercise6
Car 1 is waiting to cross the bridge.
Car 1 is crossing the bridge.
Car 2 is waiting to cross the bridge.
Car 3 is waiting to cross the bridge.
Car 3 is crossing the bridge.
Car 2 is crossing the bridge.
Car 4 is waiting to cross the bridge.
Car 5 is waiting to cross the bridge.
Car 6 is waiting to cross the bridge.
Car 7 is waiting to cross the bridge.
Car 8 is waiting to cross the bridge.
Car 9 is waiting to cross the bridge.
Car 10 is waiting to cross the bridge.
Car 1 has crossed the bridge.
Car 4 is crossing the bridge.
Car 3 has crossed the bridge.
Car 5 is crossing the bridge.
Car 2 has crossed the bridge.
Car 6 is crossing the bridge.
Car 4 has crossed the bridge.
Car 7 is crossing the bridge.
Car 5 has crossed the bridge.
Car 8 is crossing the bridge.
Car 7 has crossed the bridge.
Car 9 is crossing the bridge.
Car 6 has crossed the bridge.
Car 10 is crossing the bridge.
Car 8 has crossed the bridge.
Car 10 has crossed the bridge.
Car 9 has crossed the bridge.
alishba@DESKTOP-S3K0VA5:~/Week10-hometasks$
```