National Textile University

Department of Computer Science

Subject:

Operating System

Submitted to:

Sir Nasir Mehmood

Submitted by:

Akasha Fatima

Reg. number:

23-NTU-CS-FL-1132

Semester:

5th - A

Home Tasks 10

Task_01: Exercise 1 – Hotel Room Occupancy Problem

Scenario: A hotel has N Rooms.

Only N people can take rooms at a time; others must wait outside.

One person can only take one room, and one room can only be taken by one person.

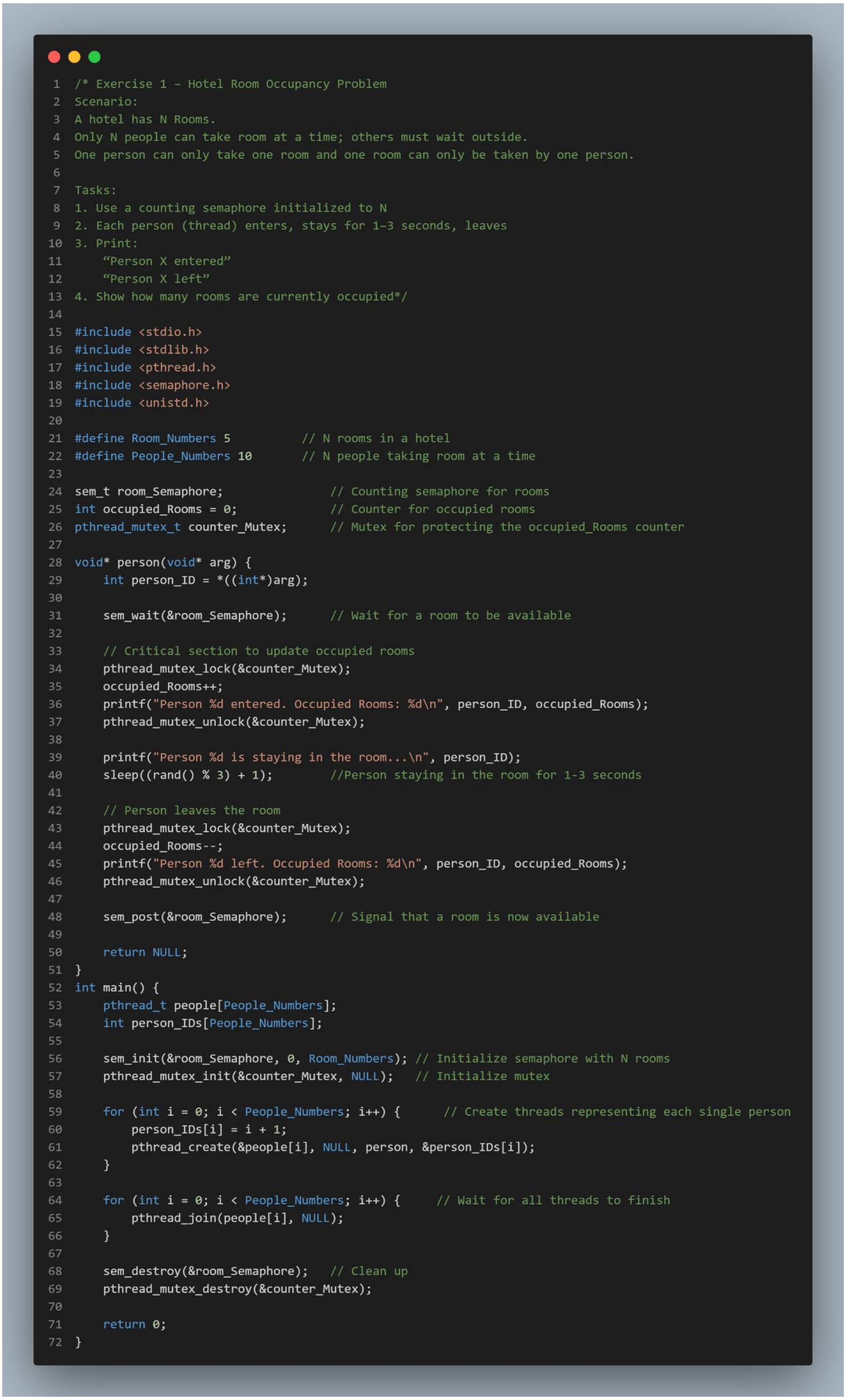
Tasks: 1. Use a counting semaphore initialized to N

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

3. Print: "Person X entered" "Person X left"

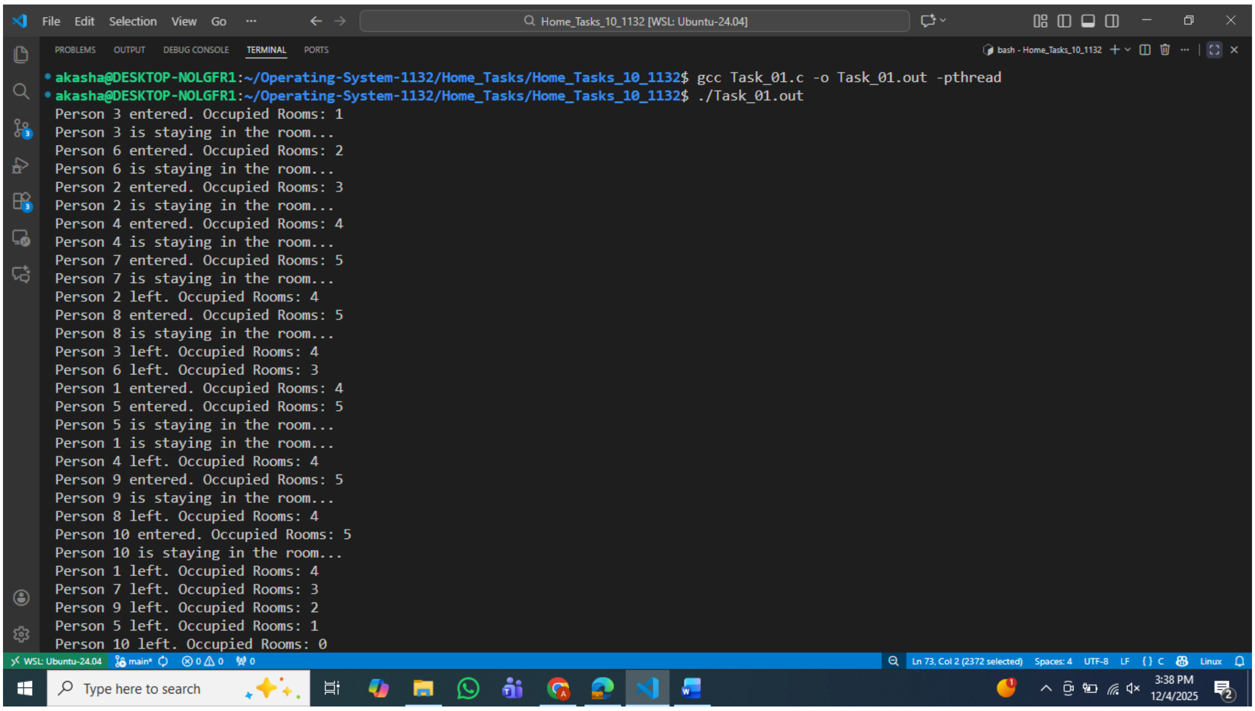
4. Show how many rooms are currently occupied

CODE:



```c
/* 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*/

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

#define Room_Numbers 5          // N rooms in a hotel
#define People_Numbers 10       // N people taking room at a time

sem_t room_Semaphore;               // Counting semaphore for rooms
int occupied_Rooms = 0;             // Counter for occupied rooms
pthread_mutex_t counter_Mutex;      // Mutex for protecting the occupied_Rooms counter

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

    sem_wait(&room_Semaphore);      // Wait for a room to be available

    // Critical section to update occupied rooms
    pthread_mutex_lock(&counter_Mutex);
    occupied_Rooms++;
    printf("Person %d entered. Occupied Rooms: %d\n", person_ID, occupied_Rooms);
    pthread_mutex_unlock(&counter_Mutex);

    printf("Person %d is staying in the room...\n", person_ID);
    sleep((rand() % 3) + 1);        //Person staying in the room for 1-3 seconds

    // Person leaves the room
    pthread_mutex_lock(&counter_Mutex);
    occupied_Rooms--;
    printf("Person %d left. Occupied Rooms: %d\n", person_ID, occupied_Rooms);
    pthread_mutex_unlock(&counter_Mutex);

    sem_post(&room_Semaphore);      // Signal that a room is now available

    return NULL;
}
int main() {
    pthread_t people[People_Numbers];
    int person_IDs[People_Numbers];

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

    for (int i = 0; i < People_Numbers; i++) {      // Create threads representing each single person
        person_IDs[i] = i + 1;
        pthread_create(&people[i], NULL, person, &person_IDs[i]);
    }

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

    sem_destroy(&room_Semaphore);   // Clean up
    pthread_mutex_destroy(&counter_Mutex);

    return 0;
}
```

Output:



```
akasha@DESKTOP-NOLGFR1:~/Operating-System-1132/Home_Tasks/Home_Tasks_10_1132$ gcc Task_01.c -o Task_01.out -pthread
akasha@DESKTOP-NOLGFR1:~/Operating-System-1132/Home_Tasks/Home_Tasks_10_1132$ ./Task_01.out
Person 3 entered. Occupied Rooms: 1
Person 3 is staying in the room...
Person 6 entered. Occupied Rooms: 2
Person 6 is staying in the room...
Person 2 entered. Occupied Rooms: 3
Person 2 is staying in the room...
Person 4 entered. Occupied Rooms: 4
Person 4 is staying in the room...
Person 7 entered. Occupied Rooms: 5
Person 7 is staying in the room...
Person 2 left. Occupied Rooms: 4
Person 8 entered. Occupied Rooms: 5
Person 8 is staying in the room...
Person 3 left. Occupied Rooms: 4
Person 6 left. Occupied Rooms: 3
Person 1 entered. Occupied Rooms: 4
Person 5 entered. Occupied Rooms: 5
Person 5 is staying in the room...
Person 1 is staying in the room...
Person 4 left. Occupied Rooms: 4
Person 9 entered. Occupied Rooms: 5
Person 9 is staying in the room...
Person 8 left. Occupied Rooms: 4
Person 10 entered. Occupied Rooms: 5
Person 10 is staying in the room...
Person 1 left. Occupied Rooms: 4
Person 7 left. Occupied Rooms: 3
Person 9 left. Occupied Rooms: 2
Person 5 left. Occupied Rooms: 1
Person 10 left. Occupied Rooms: 0
```

Task_02: Exercise 2 – Download Manager Simulation

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

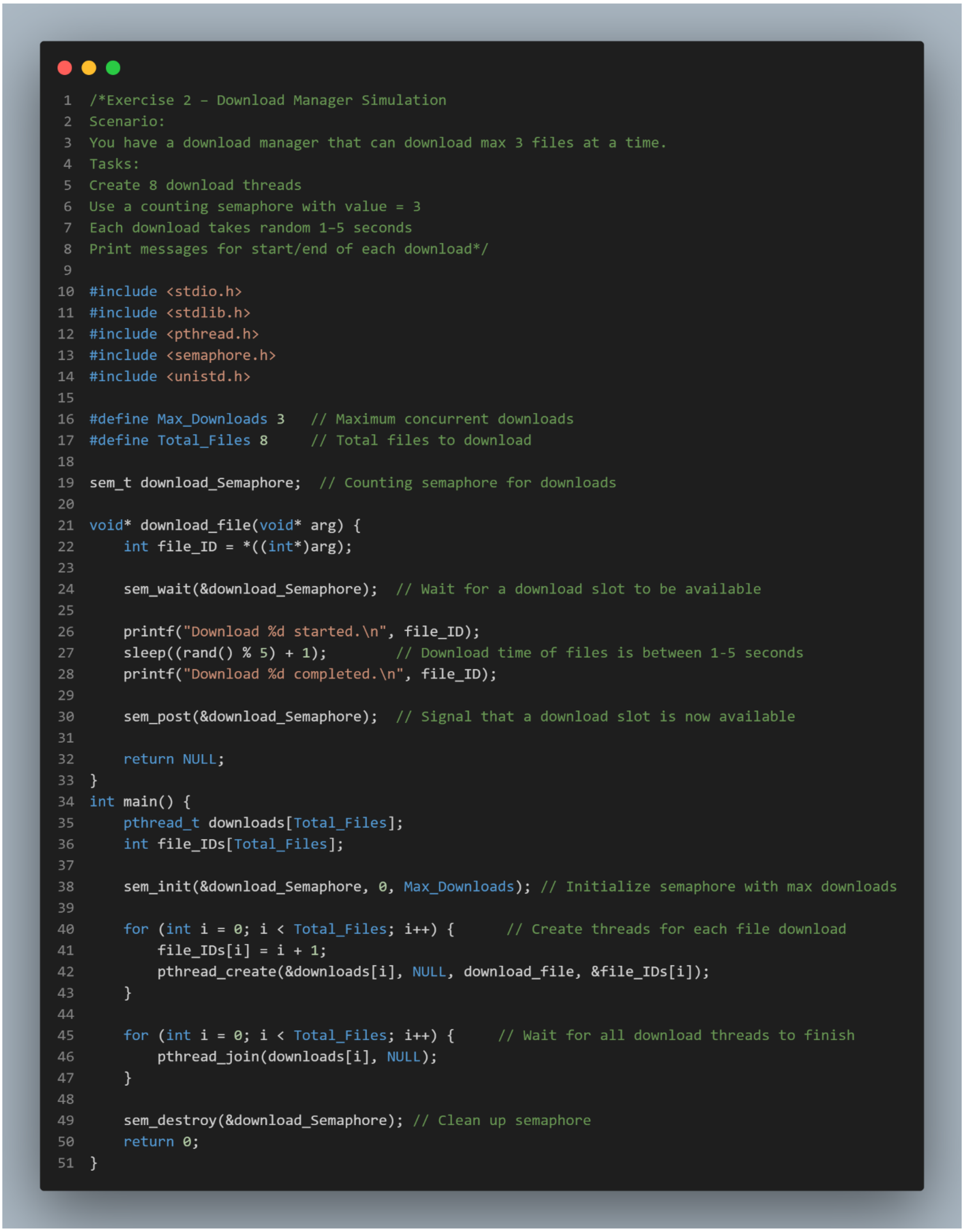
Tasks: Create 8 download threads

Use a counting semaphore with value = 3

Each download takes random 1–5 seconds

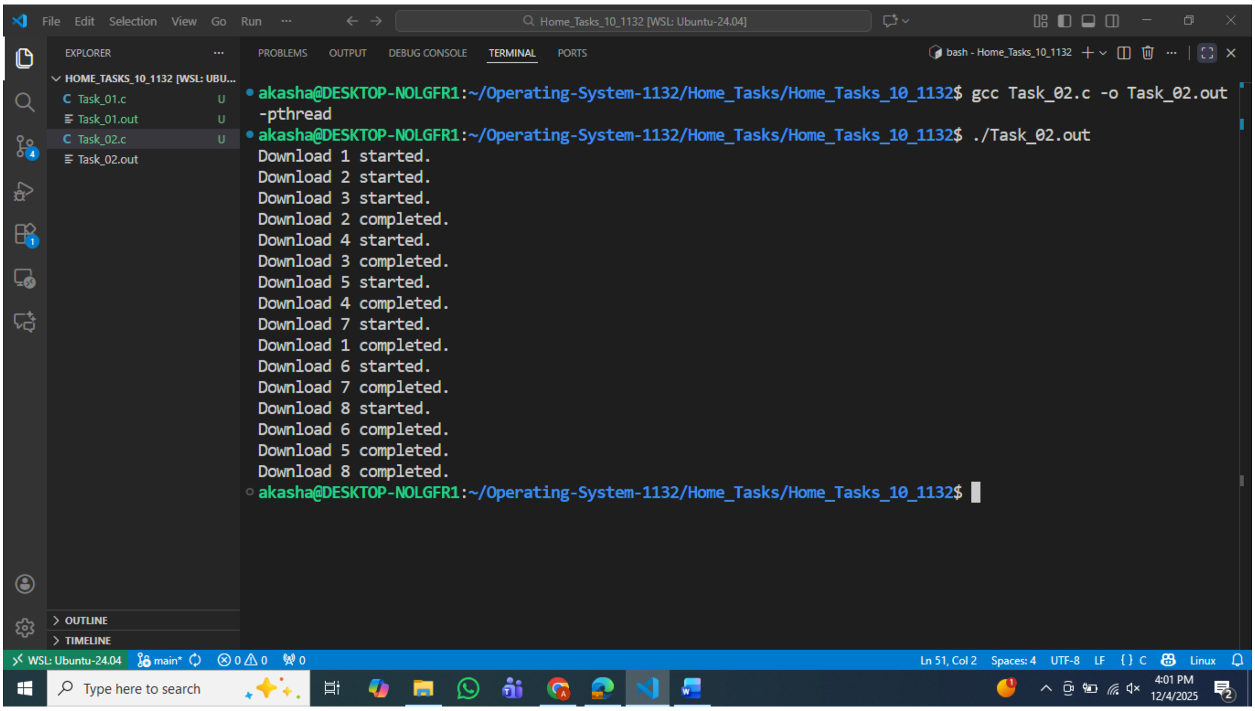
Print messages for start/end of each download

CODE:



```c
/*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*/

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

#define Max_Downloads 3   // Maximum concurrent downloads
#define Total_Files 8     // Total files to download

sem_t download_Semaphore;  // Counting semaphore for downloads

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

    sem_wait(&download_Semaphore);  // Wait for a download slot to be available

    printf("Download %d started.\n", file_ID);
    sleep((rand() % 5) + 1);        // Download time of files is between 1-5 seconds
    printf("Download %d completed.\n", file_ID);

    sem_post(&download_Semaphore);  // Signal that a download slot is now available

    return NULL;
}
int main() {
    pthread_t downloads[Total_Files];
    int file_IDs[Total_Files];

    sem_init(&download_Semaphore, 0, Max_Downloads); // Initialize semaphore with max downloads

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

    for (int i = 0; i < Total_Files; i++) {     // Wait for all download threads to finish
        pthread_join(downloads[i], NULL);
    }

    sem_destroy(&download_Semaphore); // Clean up semaphore
    return 0;
}
```

Output:


File Edit Selection View Go Run
Home_Tasks_10_1132 [WSL: Ubuntu-24.04]
EXPLORER
HOME_TASKS_10_1132 [WSL: UBU...
Task_01.c
Task_01.out
Task_02.c
Task_02.out
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
bash - Home_Tasks_10_1132
OUTLINE
TIMELINE
WSL: Ubuntu-24.04
main*
Ln 51, Col 2 Spaces: 4 UTF-8 LF { } C Linux
Type here to search
4:01 PM
12/4/2025

```
akasha@DESKTOP-NOLGFR1:~/Operating-System-1132/Home_Tasks/Home_Tasks_10_1132$ gcc Task_02.c -o Task_02.out
-pthread
akasha@DESKTOP-NOLGFR1:~/Operating-System-1132/Home_Tasks/Home_Tasks_10_1132$ ./Task_02.out
Download 1 started.
Download 2 started.
Download 3 started.
Download 2 completed.
Download 4 started.
Download 3 completed.
Download 5 started.
Download 4 completed.
Download 7 started.
Download 1 completed.
Download 6 started.
Download 7 completed.
Download 8 started.
Download 6 completed.
Download 5 completed.
Download 8 completed.
akasha@DESKTOP-NOLGFR1:~/Operating-System-1132/Home_Tasks/Home_Tasks_10_1132$
```

Task_03: Exercise 3 – Library Computer Lab Access

Scenario:

A university lab has K computers.

Students must wait until a computer becomes free.

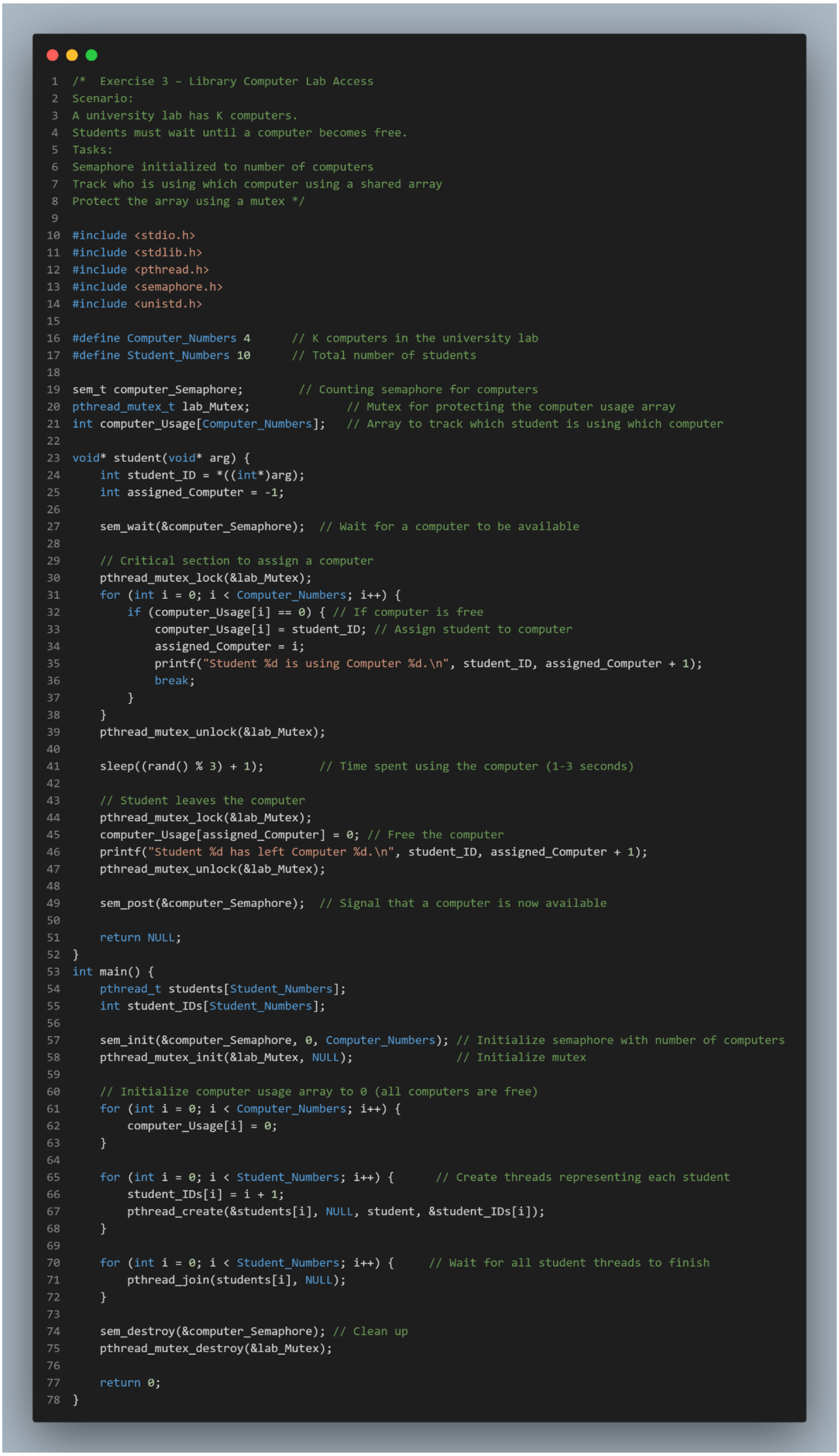
Tasks:

Semaphore initialized to number of computers

Track who is using which computer using a shared array

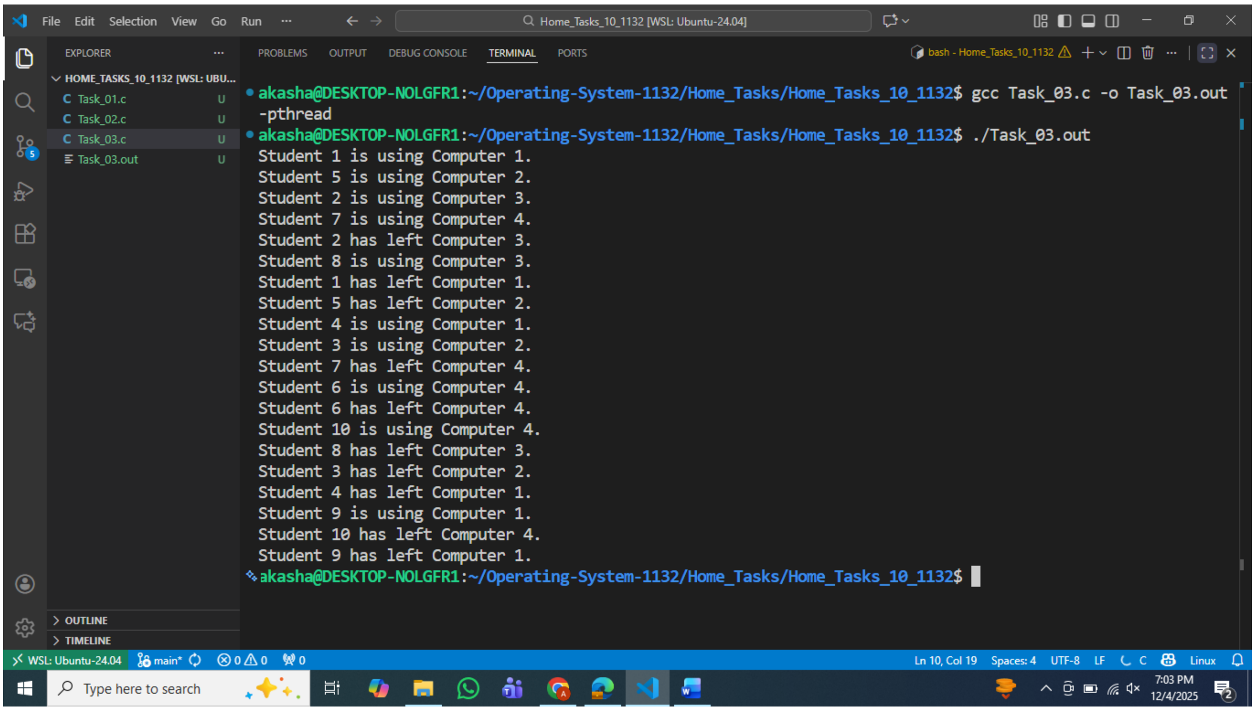
Protect the array using a mutex

CODE:



```c
/*  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 */

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

#define Computer_Numbers 4      // K computers in the university lab
#define Student_Numbers 10      // Total number of students

sem_t computer_Semaphore;        // Counting semaphore for computers
pthread_mutex_t lab_Mutex;              // Mutex for protecting the computer usage array
int computer_Usage[Computer_Numbers];   // Array to track which student is using which computer

void* student(void* arg) {
    int student_ID = *((int*)arg);
    int assigned_Computer = -1;

    sem_wait(&computer_Semaphore);  // Wait for a computer to be available

    // Critical section to assign a computer
    pthread_mutex_lock(&lab_Mutex);
    for (int i = 0; i < Computer_Numbers; i++) {
        if (computer_Usage[i] == 0) { // If computer is free
            computer_Usage[i] = student_ID; // Assign student to computer
            assigned_Computer = i;
            printf("Student %d is using Computer %d.\n", student_ID, assigned_Computer + 1);
            break;
        }
    }
    pthread_mutex_unlock(&lab_Mutex);

    sleep((rand() % 3) + 1);        // Time spent using the computer (1-3 seconds)

    // Student leaves the computer
    pthread_mutex_lock(&lab_Mutex);
    computer_Usage[assigned_Computer] = 0; // Free the computer
    printf("Student %d has left Computer %d.\n", student_ID, assigned_Computer + 1);
    pthread_mutex_unlock(&lab_Mutex);

    sem_post(&computer_Semaphore);  // Signal that a computer is now available

    return NULL;
}
int main() {
    pthread_t students[Student_Numbers];
    int student_IDs[Student_Numbers];

    sem_init(&computer_Semaphore, 0, Computer_Numbers); // Initialize semaphore with number of computers
    pthread_mutex_init(&lab_Mutex, NULL);               // Initialize mutex

    // Initialize computer usage array to 0 (all computers are free)
    for (int i = 0; i < Computer_Numbers; i++) {
        computer_Usage[i] = 0;
    }

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

    for (int i = 0; i < Student_Numbers; i++) {     // Wait for all student threads to finish
        pthread_join(students[i], NULL);
    }

    sem_destroy(&computer_Semaphore); // Clean up
    pthread_mutex_destroy(&lab_Mutex);

    return 0;
}
```

Output:



```
akasha@DESKTOP-NOLGFR1:~/Operating-System-1132/Home_Tasks/Home_Tasks_10_1132$ gcc Task_03.c -o Task_03.out
-pthread
akasha@DESKTOP-NOLGFR1:~/Operating-System-1132/Home_Tasks/Home_Tasks_10_1132$ ./Task_03.out
Student 1 is using Computer 1.
Student 5 is using Computer 2.
Student 2 is using Computer 3.
Student 7 is using Computer 4.
Student 2 has left Computer 3.
Student 8 is using Computer 3.
Student 1 has left Computer 1.
Student 5 has left Computer 2.
Student 4 is using Computer 1.
Student 3 is using Computer 2.
Student 7 has left Computer 4.
Student 6 is using Computer 4.
Student 6 has left Computer 4.
Student 10 is using Computer 4.
Student 8 has left Computer 3.
Student 3 has left Computer 2.
Student 4 has left Computer 1.
Student 9 is using Computer 1.
Student 10 has left Computer 4.
Student 9 has left Computer 1.
akasha@DESKTOP-NOLGFR1:~/Operating-System-1132/Home_Tasks/Home_Tasks_10_1132$
```

Task_04: Exercise 4 – Thread Pool / Worker Pool Simulation

Scenario:

A server has fixed number of worker threads.

More tasks arrive than workers available.

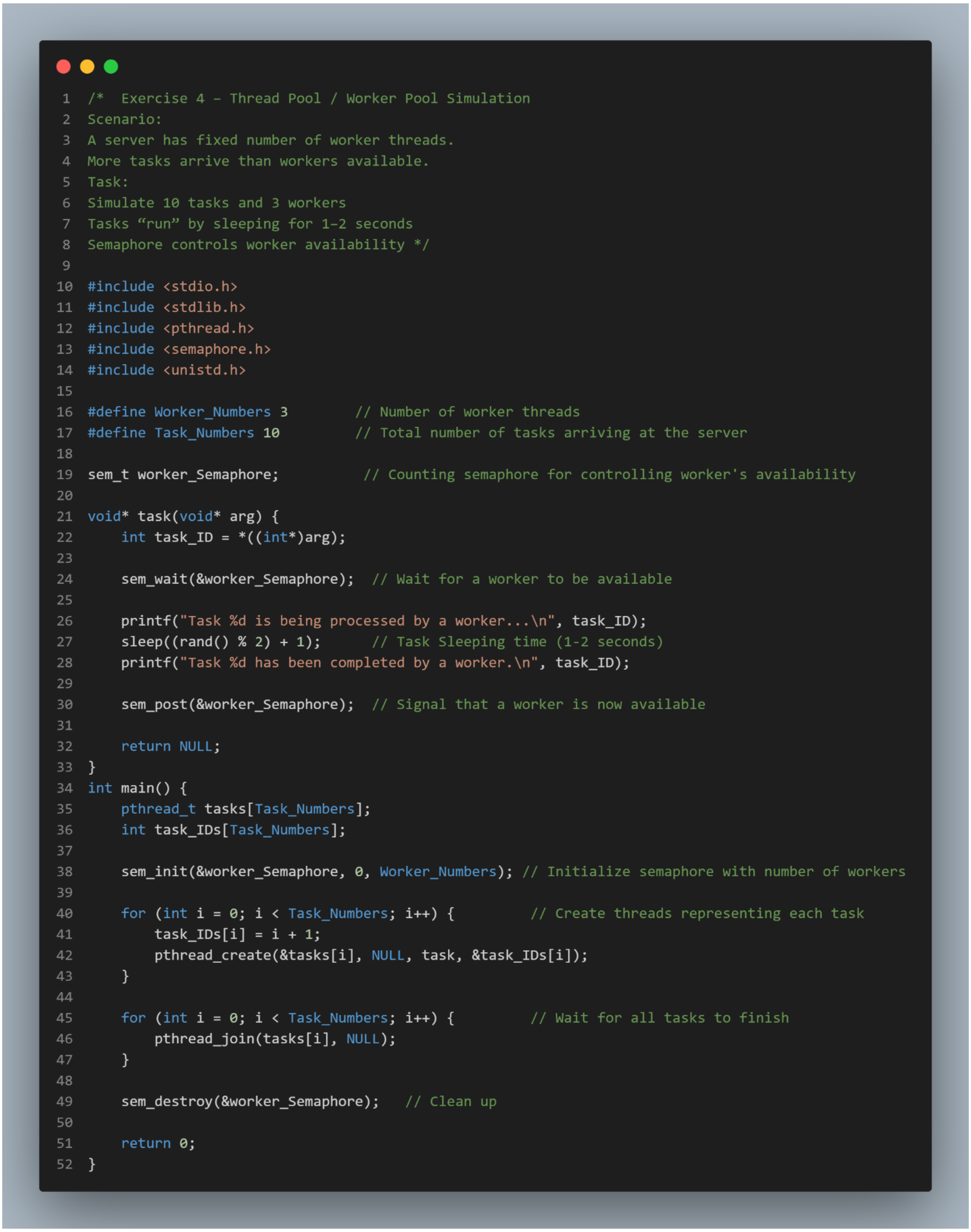
Task:

Simulate 10 tasks and 3 workers

Tasks “run” by sleeping for 1–2 seconds

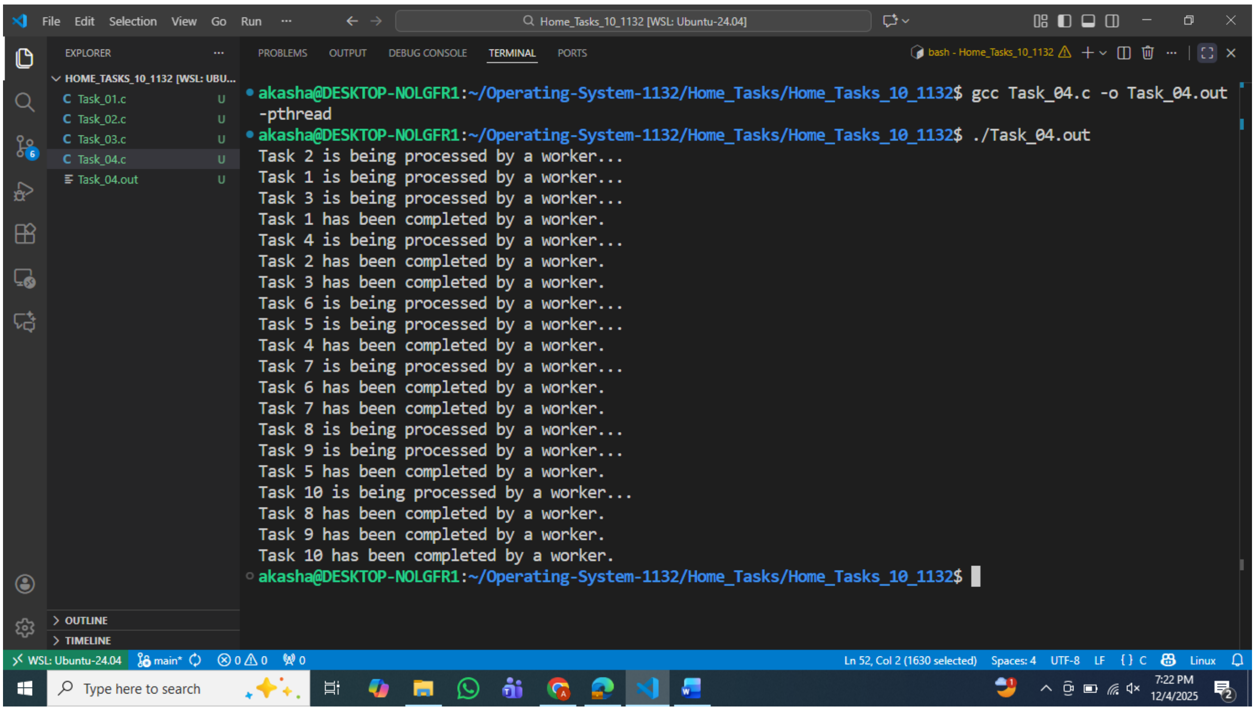
Semaphore controls worker availability

CODE:



```c
/*  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 */

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

#define Worker_Numbers 3        // Number of worker threads
#define Task_Numbers 10         // Total number of tasks arriving at the server

sem_t worker_Semaphore;          // Counting semaphore for controlling worker's availability

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

    sem_wait(&worker_Semaphore);  // Wait for a worker to be available

    printf("Task %d is being processed by a worker...\n", task_ID);
    sleep((rand() % 2) + 1);      // Task Sleeping time (1-2 seconds)
    printf("Task %d has been completed by a worker.\n", task_ID);

    sem_post(&worker_Semaphore);  // Signal that a worker is now available

    return NULL;
}
int main() {
    pthread_t tasks[Task_Numbers];
    int task_IDs[Task_Numbers];

    sem_init(&worker_Semaphore, 0, Worker_Numbers); // Initialize semaphore with number of workers

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

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

    sem_destroy(&worker_Semaphore);   // Clean up

    return 0;
}
```

Output:



File Edit Selection View Go Run ...
Home_Tasks_10_1132 [WSL: Ubuntu-24.04]
EXPLORER
HOME_TASKS_10_1132 [WSL: UBU...
Task_01.c
Task_02.c
Task_03.c
Task_04.c
Task_04.out
OUTLINE
TIMELINE
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
bash - Home_Tasks_10_1132
WSL: Ubuntu-24.04
main*
Ln 52, Col 2 (1630 selected) Spaces: 4 UTF-8 LF { } C Linux
Type here to search
7:22 PM
12/4/2025

```
akasha@DESKTOP-NOLGFR1:~/Operating-System-1132/Home_Tasks/Home_Tasks_10_1132$ gcc Task_04.c -o Task_04.out
-pthread
akasha@DESKTOP-NOLGFR1:~/Operating-System-1132/Home_Tasks/Home_Tasks_10_1132$ ./Task_04.out
Task 2 is being processed by a worker...
Task 1 is being processed by a worker...
Task 3 is being processed by a worker...
Task 1 has been completed by a worker.
Task 4 is being processed by a worker...
Task 2 has been completed by a worker.
Task 3 has been completed by a worker.
Task 6 is being processed by a worker...
Task 5 is being processed by a worker...
Task 4 has been completed by a worker.
Task 7 is being processed by a worker...
Task 6 has been completed by a worker.
Task 7 has been completed by a worker.
Task 8 is being processed by a worker...
Task 9 is being processed by a worker...
Task 5 has been completed by a worker.
Task 10 is being processed by a worker...
Task 8 has been completed by a worker.
Task 9 has been completed by a worker.
Task 10 has been completed by a worker.
akasha@DESKTOP-NOLGFR1:~/Operating-System-1132/Home_Tasks/Home_Tasks_10_1132$
```

Task_05: Exercise 5 – Car Wash Station

Scenario:

Car wash has two washing stations.

Tasks:

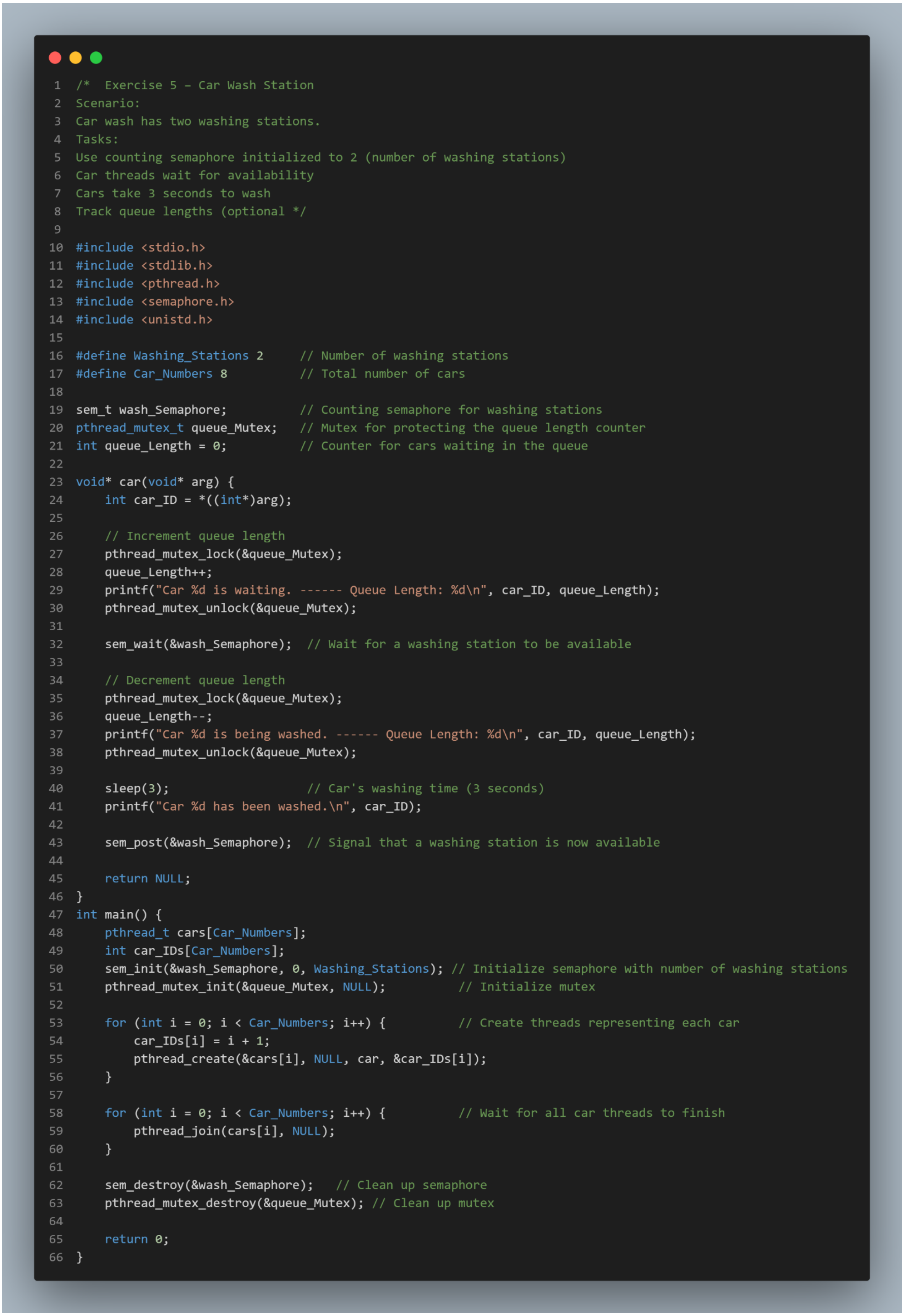
Use counting semaphore initialized to 2 (number of washing stations)

Car threads wait for availability

Cars take 3 seconds to wash

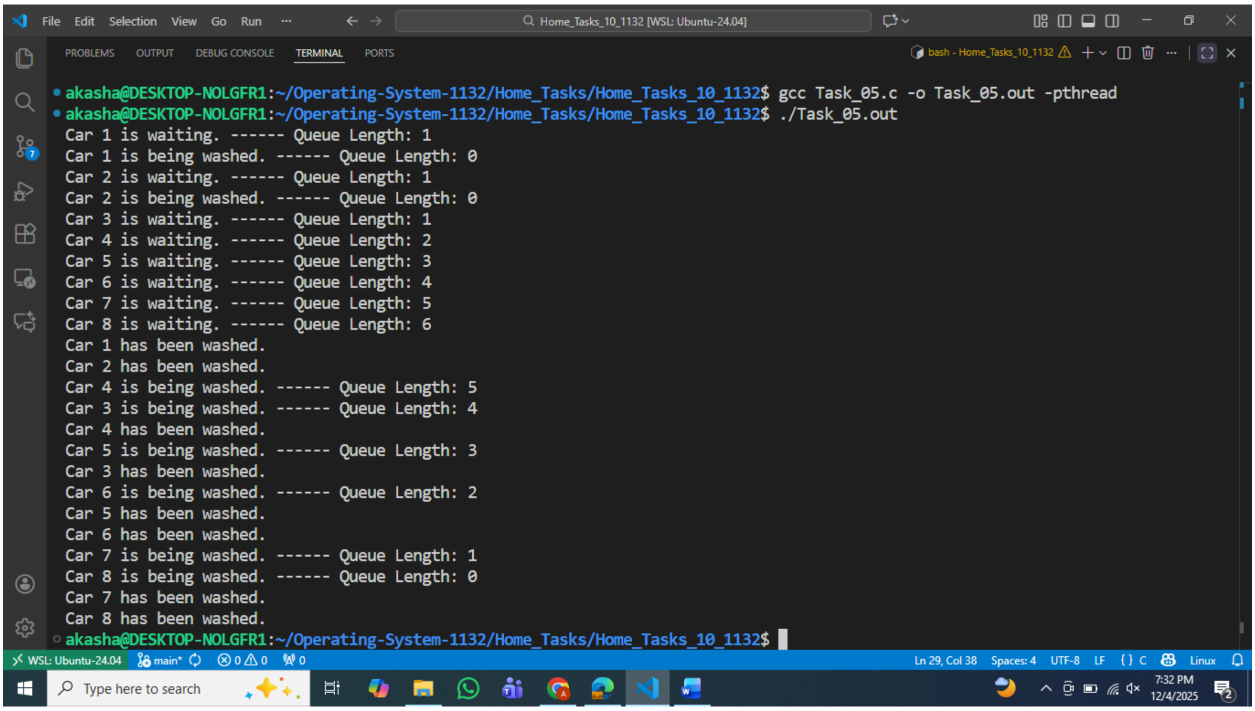
Track queue lengths (optional

CODE:



```c
/*  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 */

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

#define Washing_Stations 2     // Number of washing stations
#define Car_Numbers 8          // Total number of cars

sem_t wash_Semaphore;          // Counting semaphore for washing stations
pthread_mutex_t queue_Mutex;   // Mutex for protecting the queue length counter
int queue_Length = 0;          // Counter for cars waiting in the queue

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

    // Increment queue length
    pthread_mutex_lock(&queue_Mutex);
    queue_Length++;
    printf("Car %d is waiting. ------ Queue Length: %d\n", car_ID, queue_Length);
    pthread_mutex_unlock(&queue_Mutex);

    sem_wait(&wash_Semaphore);  // Wait for a washing station to be available

    // Decrement queue length
    pthread_mutex_lock(&queue_Mutex);
    queue_Length--;
    printf("Car %d is being washed. ------ Queue Length: %d\n", car_ID, queue_Length);
    pthread_mutex_unlock(&queue_Mutex);

    sleep(3);                   // Car's washing time (3 seconds)
    printf("Car %d has been washed.\n", car_ID);

    sem_post(&wash_Semaphore);  // Signal that a washing station is now available

    return NULL;
}
int main() {
    pthread_t cars[Car_Numbers];
    int car_IDs[Car_Numbers];
    sem_init(&wash_Semaphore, 0, Washing_Stations); // Initialize semaphore with number of washing stations
    pthread_mutex_init(&queue_Mutex, NULL);          // Initialize mutex

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

    for (int i = 0; i < Car_Numbers; i++) {          // Wait for all car threads to finish
        pthread_join(cars[i], NULL);
    }

    sem_destroy(&wash_Semaphore);   // Clean up semaphore
    pthread_mutex_destroy(&queue_Mutex); // Clean up mutex

    return 0;
}
```

Output:


File Edit Selection View Go Run ···
Home_Tasks_10_1132 [WSL: Ubuntu-24.04]
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
bash - Home_Tasks_10_1132
WSL: Ubuntu-24.04
main*
Ln 29, Col 38 Spaces: 4 UTF-8 LF { } C Linux
Type here to search
7:32 PM
12/4/2025

```
akasha@DESKTOP-NOLGFR1:~/Operating-System-1132/Home_Tasks/Home_Tasks_10_1132$ gcc Task_05.c -o Task_05.out -pthread
akasha@DESKTOP-NOLGFR1:~/Operating-System-1132/Home_Tasks/Home_Tasks_10_1132$ ./Task_05.out
Car 1 is waiting. ------ Queue Length: 1
Car 1 is being washed. ------ Queue Length: 0
Car 2 is waiting. ------ Queue Length: 1
Car 2 is being washed. ------ Queue Length: 0
Car 3 is waiting. ------ Queue Length: 1
Car 4 is waiting. ------ Queue Length: 2
Car 5 is waiting. ------ Queue Length: 3
Car 6 is waiting. ------ Queue Length: 4
Car 7 is waiting. ------ Queue Length: 5
Car 8 is waiting. ------ Queue Length: 6
Car 1 has been washed.
Car 2 has been washed.
Car 4 is being washed. ------ Queue Length: 5
Car 3 is being washed. ------ Queue Length: 4
Car 4 has been washed.
Car 5 is being washed. ------ Queue Length: 3
Car 3 has been washed.
Car 6 is being washed. ------ Queue Length: 2
Car 5 has been washed.
Car 6 has been washed.
Car 7 is being washed. ------ Queue Length: 1
Car 8 is being washed. ------ Queue Length: 0
Car 7 has been washed.
Car 8 has been washed.
akasha@DESKTOP-NOLGFR1:~/Operating-System-1132/Home_Tasks/Home_Tasks_10_1132$
```

Task_06: Exercise 6 – Traffic Bridge Control (Single-Lane Bridge)

Scenario:

Only 3 cars are allowed on the bridge at once.

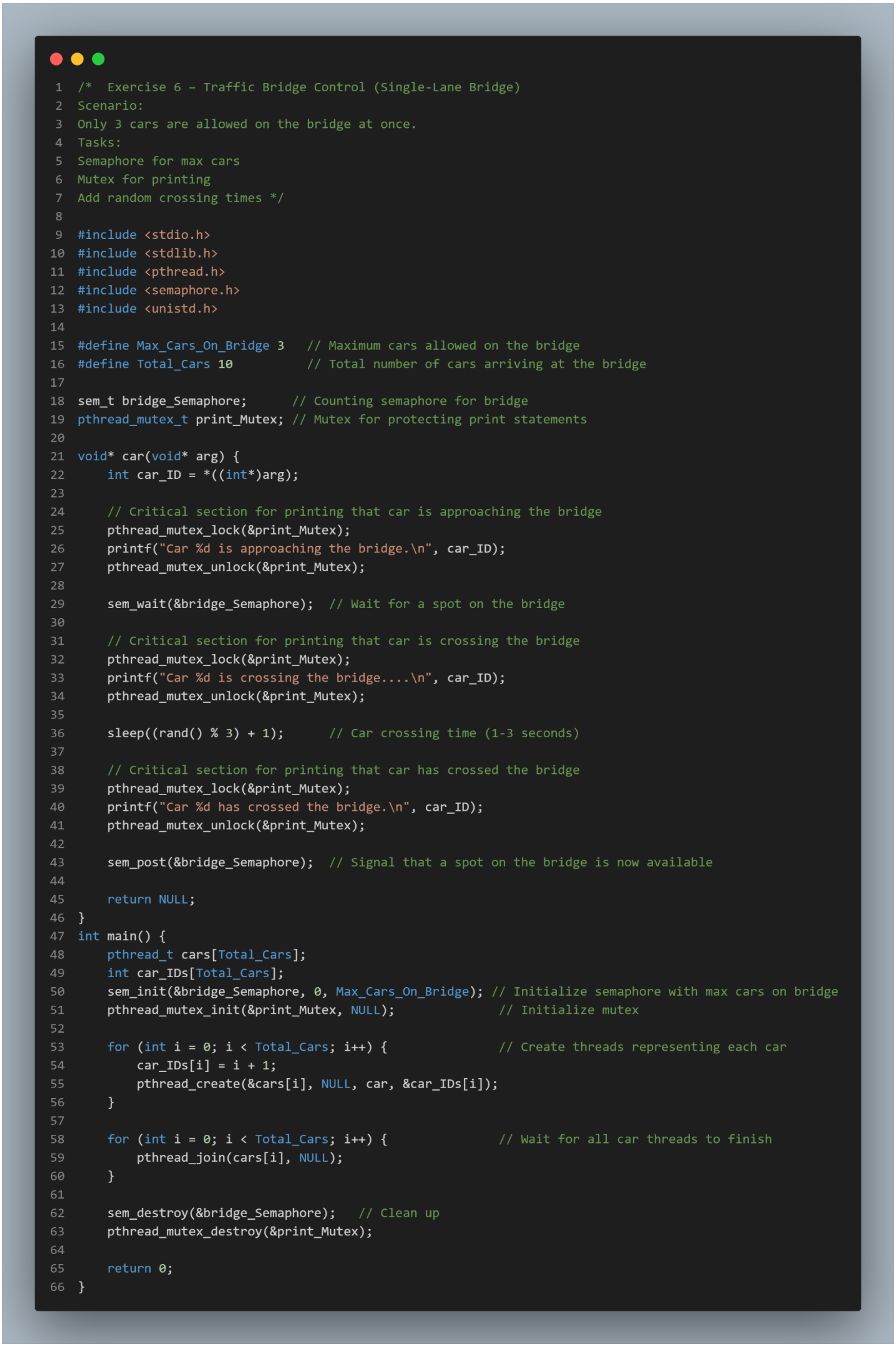
Tasks:

Semaphore for max cars

Mutex for printing

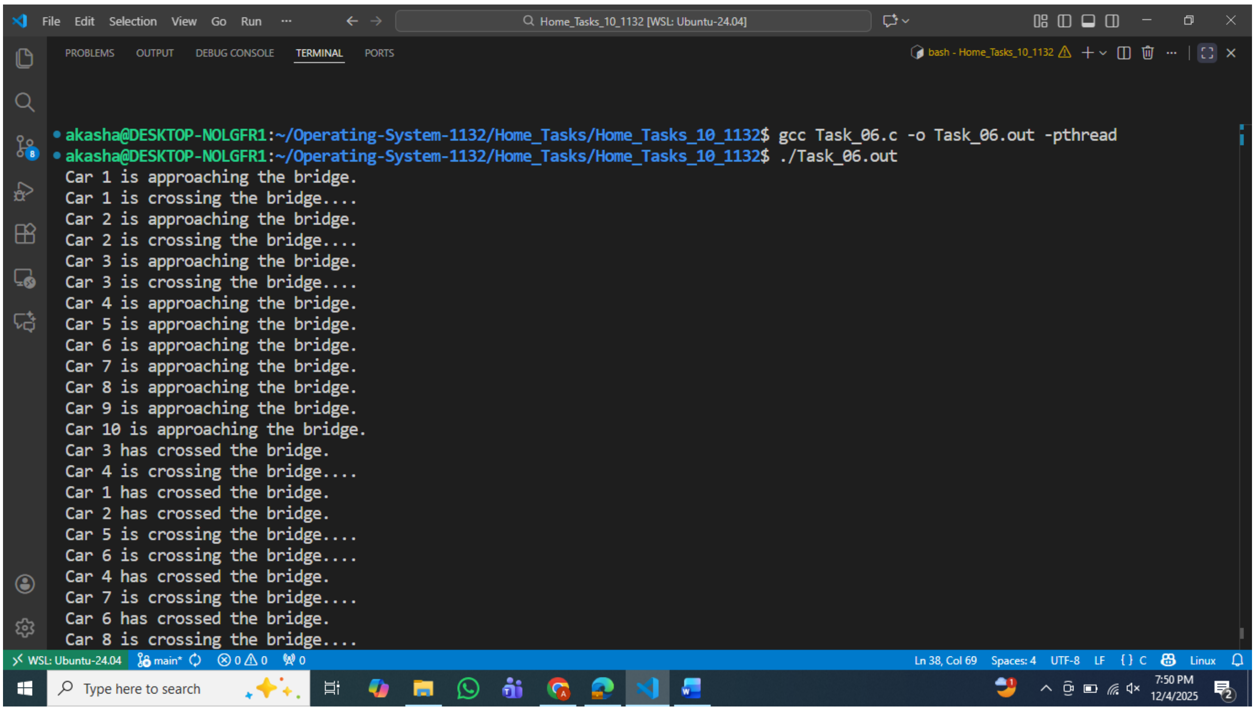
Add random crossing times

CODE:



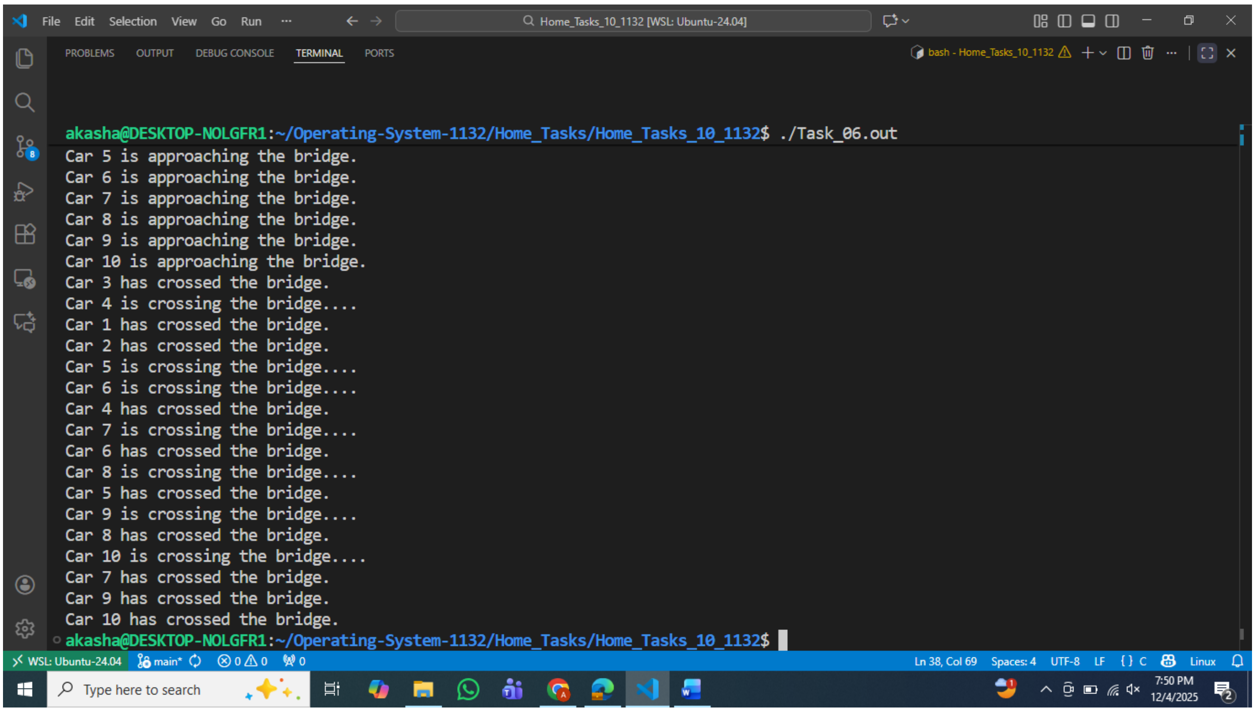
```c
/*  Exercise 6 – Traffic Bridge Control (Single-Lane Bridge)
Scenario:
Only 3 cars are allowed on the bridge at once.
Tasks:
Semaphore for max cars
Mutex for printing
Add random crossing times */

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

#define Max_Cars_On_Bridge 3   // Maximum cars allowed on the bridge
#define Total_Cars 10          // Total number of cars arriving at the bridge

sem_t bridge_Semaphore;      // Counting semaphore for bridge
pthread_mutex_t print_Mutex; // Mutex for protecting print statements

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

    // Critical section for printing that car is approaching the bridge
    pthread_mutex_lock(&print_Mutex);
    printf("Car %d is approaching the bridge.\n", car_ID);
    pthread_mutex_unlock(&print_Mutex);

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

    // Critical section for printing that car is crossing the bridge
    pthread_mutex_lock(&print_Mutex);
    printf("Car %d is crossing the bridge....\n", car_ID);
    pthread_mutex_unlock(&print_Mutex);

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

    // Critical section for printing that car has crossed the bridge
    pthread_mutex_lock(&print_Mutex);
    printf("Car %d has crossed the bridge.\n", car_ID);
    pthread_mutex_unlock(&print_Mutex);

    sem_post(&bridge_Semaphore);  // Signal that a spot on the bridge is now available

    return NULL;
}
int main() {
    pthread_t cars[Total_Cars];
    int car_IDs[Total_Cars];
    sem_init(&bridge_Semaphore, 0, Max_Cars_On_Bridge); // Initialize semaphore with max cars on bridge
    pthread_mutex_init(&print_Mutex, NULL);              // Initialize mutex

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

    for (int i = 0; i < Total_Cars; i++) {               // Wait for all car threads to finish
        pthread_join(cars[i], NULL);
    }

    sem_destroy(&bridge_Semaphore);   // Clean up
    pthread_mutex_destroy(&print_Mutex);

    return 0;
}
```

Output:



File Edit Selection View Go Run ...
Home_Tasks_10_1132 [WSL: Ubuntu-24.04]
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
bash - Home_Tasks_10_1132
WSL: Ubuntu-24.04 main* 0 0 0
Ln 38, Col 69 Spaces: 4 UTF-8 LF { } C Linux
Type here to search
7:50 PM
12/4/2025

```
akasha@DESKTOP-NOLGFR1:~/Operating-System-1132/Home_Tasks/Home_Tasks_10_1132$ gcc Task_06.c -o Task_06.out -pthread
akasha@DESKTOP-NOLGFR1:~/Operating-System-1132/Home_Tasks/Home_Tasks_10_1132$ ./Task_06.out
Car 1 is approaching the bridge.
Car 1 is crossing the bridge....
Car 2 is approaching the bridge.
Car 2 is crossing the bridge....
Car 3 is approaching the bridge.
Car 3 is crossing the bridge....
Car 4 is approaching the bridge.
Car 5 is approaching the bridge.
Car 6 is approaching the bridge.
Car 7 is approaching the bridge.
Car 8 is approaching the bridge.
Car 9 is approaching the bridge.
Car 10 is approaching the bridge.
Car 3 has crossed the bridge.
Car 4 is crossing the bridge....
Car 1 has crossed the bridge.
Car 2 has crossed the bridge.
Car 5 is crossing the bridge....
Car 6 is crossing the bridge....
Car 4 has crossed the bridge.
Car 7 is crossing the bridge....
Car 6 has crossed the bridge.
Car 8 is crossing the bridge....
```



File Edit Selection View Go Run ...
Home_Tasks_10_1132 [WSL: Ubuntu-24.04]
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
bash - Home_Tasks_10_1132
WSL: Ubuntu-24.04 main* 0 0 0
Ln 38, Col 69 Spaces: 4 UTF-8 LF { } C Linux
Type here to search
7:50 PM
12/4/2025

```
akasha@DESKTOP-NOLGFR1:~/Operating-System-1132/Home_Tasks/Home_Tasks_10_1132$ ./Task_06.out
Car 5 is approaching the bridge.
Car 6 is approaching the bridge.
Car 7 is approaching the bridge.
Car 8 is approaching the bridge.
Car 9 is approaching the bridge.
Car 10 is approaching the bridge.
Car 3 has crossed the bridge.
Car 4 is crossing the bridge....
Car 1 has crossed the bridge.
Car 2 has crossed the bridge.
Car 5 is crossing the bridge....
Car 6 is crossing the bridge....
Car 4 has crossed the bridge.
Car 7 is crossing the bridge....
Car 6 has crossed the bridge.
Car 8 is crossing the bridge....
Car 5 has crossed the bridge.
Car 9 is crossing the bridge....
Car 8 has crossed the bridge.
Car 10 is crossing the bridge....
Car 7 has crossed the bridge.
Car 9 has crossed the bridge.
Car 10 has crossed the bridge.
akasha@DESKTOP-NOLGFR1:~/Operating-System-1132/Home_Tasks/Home_Tasks_10_1132$
```