NATIONAL TEXTILE UNIVERSITY

DEPARTMENT OF COMPUTER SCIENCE

Subject

Operating System

SUBMITTED BY:

Fatima Waseem: 23-NTU-CS1155

SECTION SE: 5th (A)

SUBMITTED TO:

Sir Nasir

Lab 10 Home Task

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

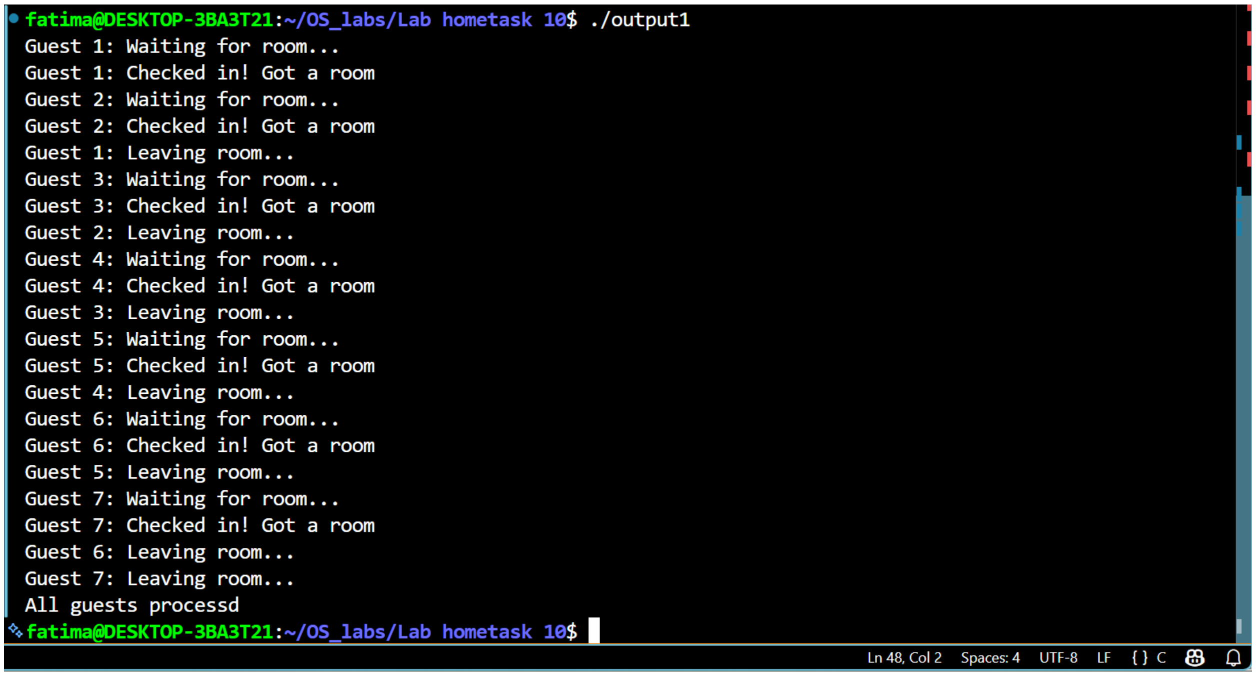
Code:

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

sem_t rooms;

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

    printf("Guest %d: Waiting for room...\n", id);

    sem_wait(&rooms);       // Try to take a room (wait if none available)
```

```
// Critical Section (guest gets a room)
    printf("Guest %d: Checked in! Got a room \n",id);
    sleep(2);       //staying in the room

    printf("Guest %d: Leaving room...\n", id);

    sem_post(&rooms);   // Release the room
    return NULL;
}
int main(){
    int N = 3;      //Number of rooms
    int total_guests = 7;   //Total people trying to get a room
    sem_init(&rooms, 0, N);      // Initialize rooms semaphore
with N rooms
    pthread_t  guests[total_guests];
    int ids[total_guests];

    for(int i = 0 ; i < total_guests; i++){
        ids[i] = i + 1;
        pthread_create(&guests[i], NULL,
guest_function,&ids[i]);
        sleep(1);
    }

    for(int i = 0 ; i < total_guests; i++){
        pthread_join(guests[i],NULL);
    }

    sem_destroy(&rooms);
    printf("All guests processd\n");

    return 0;
```

```
}
```

Output:



```
fatima@DESKTOP-3BA3T21:~/OS_labs/Lab hometask 10$ ./output1
Guest 1: Waiting for room...
Guest 1: Checked in! Got a room
Guest 2: Waiting for room...
Guest 2: Checked in! Got a room
Guest 1: Leaving room...
Guest 3: Waiting for room...
Guest 3: Checked in! Got a room
Guest 2: Leaving room...
Guest 4: Waiting for room...
Guest 4: Checked in! Got a room
Guest 3: Leaving room...
Guest 5: Waiting for room...
Guest 5: Checked in! Got a room
Guest 4: Leaving room...
Guest 6: Waiting for room...
Guest 6: Checked in! Got a room
Guest 5: Leaving room...
Guest 7: Waiting for room...
Guest 7: Checked in! Got a room
Guest 6: Leaving room...
Guest 7: Leaving room...
All guests processd
fatima@DESKTOP-3BA3T21:~/OS_labs/Lab hometask 10$
```

Exercise 2 – Download Manager Simulation

Scenario:

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

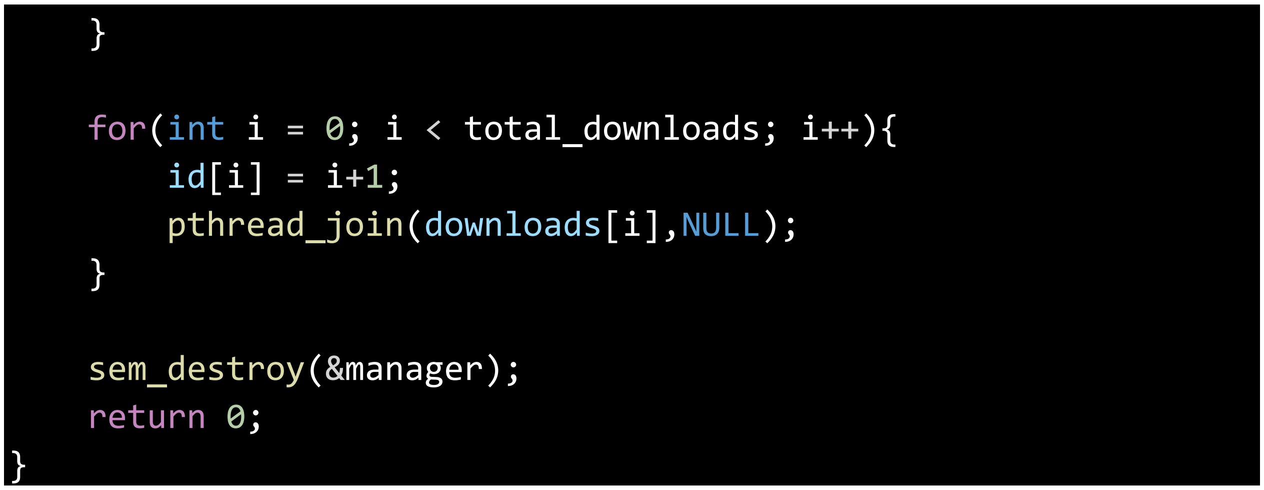
Tasks:

- 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:

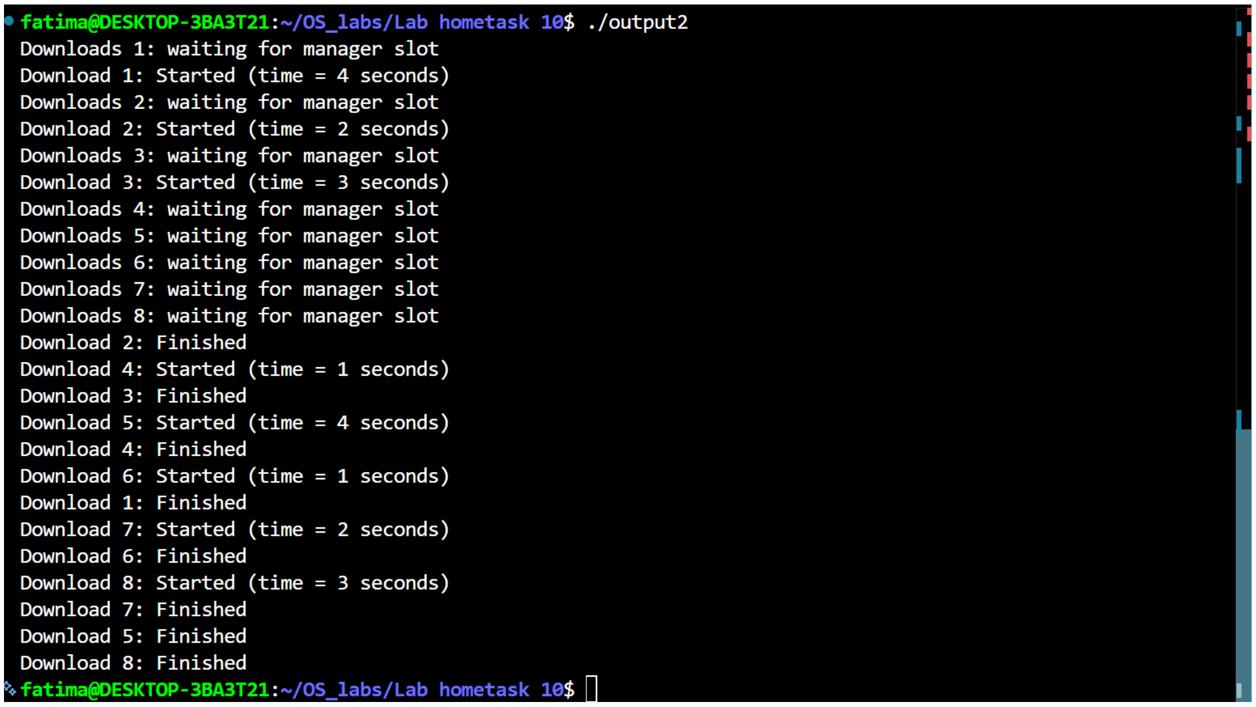
```
#include <pthread.h>
#include <unistd.h>
```

```
#include <semaphore.h>
#include <stdio.h>
#include <stdlib.h>

sem_t manager;
void* download_funtion(void* arg){
    int id = *(int*)arg;        //Cast first, dereference
second

    printf("Downloads %d: waiting for manager slot\n", id);
    sem_wait(&manager);

    int time = (rand() % 5) + 1;
    printf("Download %d: Started (time = %d seconds)\n",id
,time);
    sleep(time);
    printf("Download %d: Finished\n",id);

    sem_post(&manager);     //release slot;
    return NULL;

}

int main(){
    int N = 3;   //maximum simultaneous downloads
    int total_downloads = 8;
    sem_init(&manager, 0, N); //Initialize semaphore with
number of n capacity
    pthread_t downloads[total_downloads];
    int id[total_downloads];

    for(int i = 0; i < total_downloads; i++){
        id[i] = i+1;
        pthread_create(&downloads[i], NULL, download_funtion,
&id[i]);
```



```
    }

    for(int i = 0; i < total_downloads; i++){
        id[i] = i+1;
        pthread_join(downloads[i],NULL);
    }

    sem_destroy(&manager);
    return 0;
}
```

Output:



```
fatima@DESKTOP-3BA3T21:~/OS_labs/Lab hometask 10$ ./output2
Downloads 1: waiting for manager slot
Download 1: Started (time = 4 seconds)
Downloads 2: waiting for manager slot
Download 2: Started (time = 2 seconds)
Downloads 3: waiting for manager slot
Download 3: Started (time = 3 seconds)
Downloads 4: waiting for manager slot
Downloads 5: waiting for manager slot
Downloads 6: waiting for manager slot
Downloads 7: waiting for manager slot
Downloads 8: waiting for manager slot
Download 2: Finished
Download 4: Started (time = 1 seconds)
Download 3: Finished
Download 5: Started (time = 4 seconds)
Download 4: Finished
Download 6: Started (time = 1 seconds)
Download 1: Finished
Download 7: Started (time = 2 seconds)
Download 6: Finished
Download 8: Started (time = 3 seconds)
Download 7: Finished
Download 5: Finished
Download 8: Finished
fatima@DESKTOP-3BA3T21:~/OS_labs/Lab hometask 10$
```

Exercise 3 – Library Computer Lab Access

Scenario:

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

Tasks:

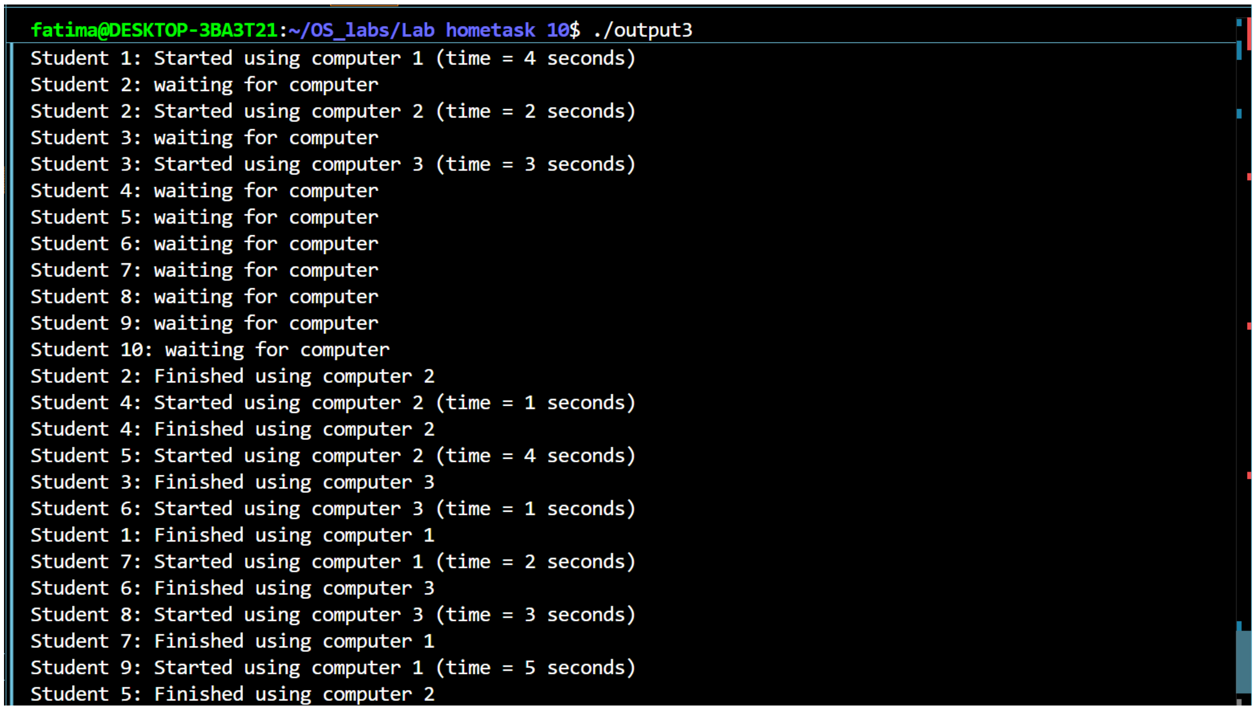
- Semaphore initialized to number of computers
- Track who is using which computer using a shared array
- Protect the array using a mutex

Code:

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


#define K 3
#define total_students 10

sem_t computer_lab;


pthread_mutex_t mutex;

int comp[K]; // Array to track computer usage

void* computerusage_function(void* arg){
    int id = *(int*)arg;
    printf("Student %d: waiting for computer\n", id);
    sem_wait(&computer_lab);

    // Find a free computer

    pthread_mutex_lock(&mutex);

    int computer_id = -1;
    for(int i = 0; i < K; i++){
        if(comp[i] == 0){
            comp[i] = 1; // Mark as occupied
            computer_id = i;
            break;
        }
    }


    pthread_mutex_unlock(&mutex);

    if (computer_id != -1) {
        int time = (rand() % 5) + 1;
        printf("Student %d: Started using computer %d (time =
%d seconds)\n", id, computer_id + 1, time);
        sleep(time);
        printf("Student %d: Finished using computer %d\n", id,
computer_id + 1);

        // Release the computer
        pthread_mutex_lock(&mutex);
        comp[computer_id] = 0; // Mark as free
        pthread_mutex_unlock(&mutex);
    }

    sem_post(&computer_lab);     // Release slot
    return NULL;
```

```
}

int main(){
    pthread_t students[total_students];
    int id[total_students];

    for(int i = 0; i < K; i++){
        comp[i] = 0; // 0 means free, 1 means occupied
    }

    sem_init(&computer_lab, 0 , K);


    pthread_mutex_init(&mutex, NULL);

    for(int i = 0; i < total_students; i++){
        id[i] = i + 1;
        pthread_create(&students[i], NULL,
computerusage_function, &id[i]);
    }

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

        pthread_join(students[i], NULL);
    }


    pthread_mutex_destroy(&mutex);
    sem_destroy(&computer_lab);

    return 0;
}
```

Output:



```
fatima@DESKTOP-3BA3T21:~/OS_labs/Lab hometask 10$ ./output3
Student 1: Started using computer 1 (time = 4 seconds)
Student 2: waiting for computer
Student 2: Started using computer 2 (time = 2 seconds)
Student 3: waiting for computer
Student 3: Started using computer 3 (time = 3 seconds)
Student 4: waiting for computer
Student 5: waiting for computer
Student 6: waiting for computer
Student 7: waiting for computer
Student 8: waiting for computer
Student 9: waiting for computer
Student 10: waiting for computer
Student 2: Finished using computer 2
Student 4: Started using computer 2 (time = 1 seconds)
Student 4: Finished using computer 2
Student 5: Started using computer 2 (time = 4 seconds)
Student 3: Finished using computer 3
Student 6: Started using computer 3 (time = 1 seconds)
Student 1: Finished using computer 1
Student 7: Started using computer 1 (time = 2 seconds)
Student 6: Finished using computer 3
Student 8: Started using computer 3 (time = 3 seconds)
Student 7: Finished using computer 1
Student 9: Started using computer 1 (time = 5 seconds)
Student 5: Finished using computer 2
```

Exercise 4 – Thread Pool / Worker Pool Simulation

Scenario:

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

Task:

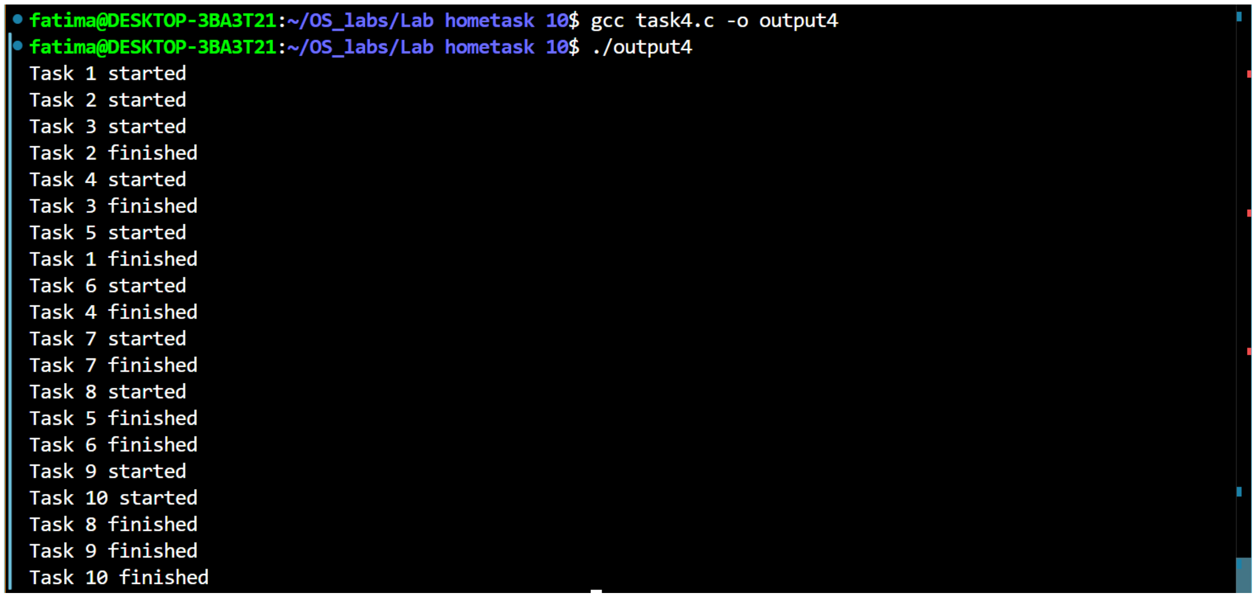
- Simulate 10 tasks and 3 workers
- Tasks "run" by sleeping for 1–2 seconds
- Semaphore controls worker availability

Code:

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

```c
#define TASKS 10
#define WORKERS 3

sem_t worker_sem;

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

    sem_wait(&worker_sem);   // wait for free worker

    printf("Task %d started\n", id);
    sleep(rand() % 2 + 1);   // task runs for 1–2 seconds
    printf("Task %d finished\n", id);

    sem_post(&worker_sem);   // worker becomes free
    return NULL;
}

int main() {
    pthread_t t[TASKS];
    int id[TASKS];

    sem_init(&worker_sem, 0, WORKERS);   // 3 workers

    for (int i = 0; i < TASKS; i++) {
        id[i] = i + 1;
        pthread_create(&t[i], NULL, task, &id[i]);
    }

    for (int i = 0; i < TASKS; i++)
        pthread_join(t[i], NULL);

    sem_destroy(&worker_sem);
    return 0;
}
```

Output:



```
fatima@DESKTOP-3BA3T21:~/OS_labs/Lab hometask 10$ gcc task4.c -o output4
fatima@DESKTOP-3BA3T21:~/OS_labs/Lab hometask 10$ ./output4
Task 1 started
Task 2 started
Task 3 started
Task 2 finished
Task 4 started
Task 3 finished
Task 5 started
Task 1 finished
Task 6 started
Task 4 finished
Task 7 started
Task 7 finished
Task 8 started
Task 5 finished
Task 6 finished
Task 9 started
Task 10 started
Task 8 finished
Task 9 finished
Task 10 finished
```

Exercise 5 – Car Wash Station

Scenario:

Car wash has **two washing stations**.

Tasks:

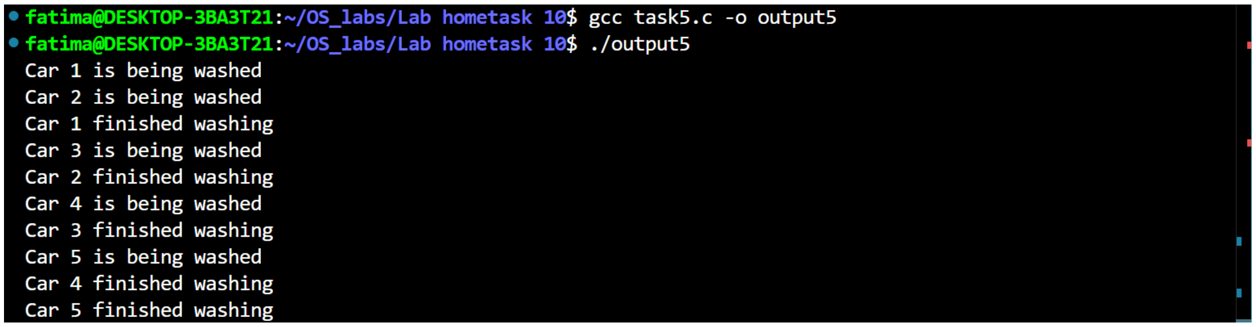
- Use counting semaphore initialized to 2 (number of washing stations)
- Car threads wait for availability
- Cars take 3 seconds to wash
- Track queue lengths (optional)

Code:

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

#define CARS 5
```

```c
sem_t wash_station;

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

    sem_wait(&wash_station);
    printf("Car %d is being washed\n", id);

    sleep(3);

    printf("Car %d finished washing\n", id);
    sem_post(&wash_station);

    return NULL;
}

int main() {
    pthread_t t[CARS];
    int id[CARS];

    sem_init(&wash_station, 0, 2); //COUNTING semaphore

    for (int i = 0; i < CARS; i++) {
        id[i] = i + 1;
        pthread_create(&t[i], NULL, car, &id[i]);
    }

    for (int i = 0; i < CARS; i++)
        pthread_join(t[i], NULL);

    sem_destroy(&wash_station);
    return 0;
}
```

Output:



```
fatima@DESKTOP-3BA3T21:~/OS_labs/Lab hometask 10$ gcc task5.c -o output5
fatima@DESKTOP-3BA3T21:~/OS_labs/Lab hometask 10$ ./output5
Car 1 is being washed
Car 2 is being washed
Car 1 finished washing
Car 3 is being washed
Car 2 finished washing
Car 4 is being washed
Car 3 finished washing
Car 5 is being washed
Car 4 finished washing
Car 5 finished washing
```

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

Scenario:

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

Tasks:

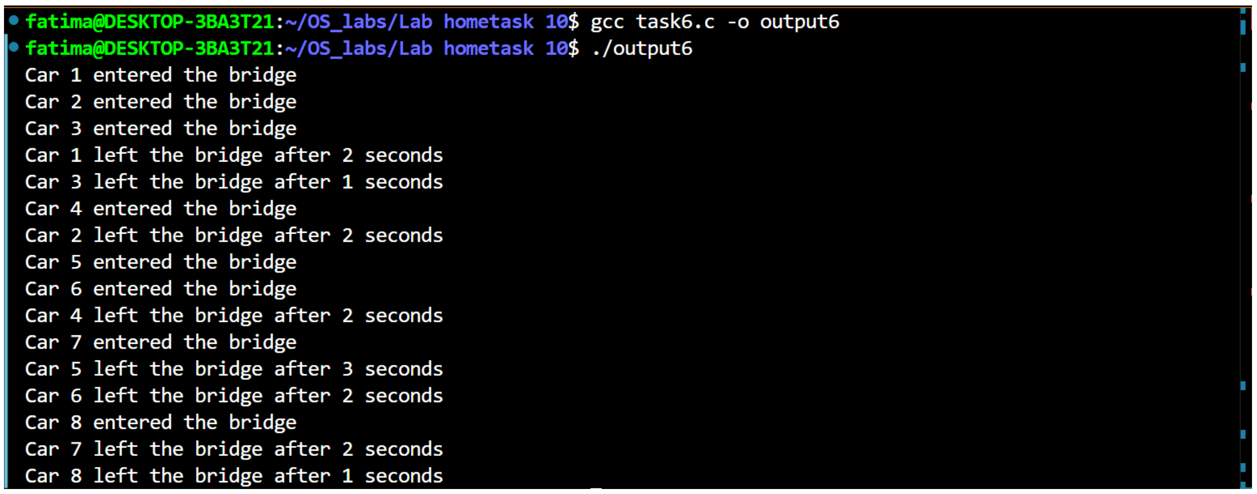
- Semaphore for max cars
- Mutex for printing
- Add random crossing times

Code:

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

#define CARS 8
```

```c
#define MAX_ON_BRIDGE 3

sem_t bridge;                // counting semaphore
pthread_mutex_t print_lock; // mutex for printing

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

    sem_wait(&bridge);       //wait for space on bridge

    pthread_mutex_lock(&print_lock);
    printf("Car %d entered the bridge\n", id);
    pthread_mutex_unlock(&print_lock);

    int time = rand() % 3 + 1;   //random 1–3 seconds
    sleep(time);

    pthread_mutex_lock(&print_lock);
    printf("Car %d left the bridge after %d seconds\n", id,
time);
    pthread_mutex_unlock(&print_lock);

    sem_post(&bridge);       //leave bridge
    return NULL;
}

int main() {
    pthread_t t[CARS];
    int id[CARS];

    sem_init(&bridge, 0, MAX_ON_BRIDGE);    //allow 3 cars
    pthread_mutex_init(&print_lock, NULL);

    for (int i = 0; i < CARS; i++) {
        id[i] = i + 1;
        pthread_create(&t[i], NULL, car, &id[i]);
        sleep(1);   // cars arrive one by one
    }

    for (int i = 0; i < CARS; i++)
        pthread_join(t[i], NULL);

    sem_destroy(&bridge);
    pthread_mutex_destroy(&print_lock);

    return 0;
}
```

Output:



```
fatima@DESKTOP-3BA3T21:~/OS_labs/Lab hometask 10$ gcc task6.c -o output6
fatima@DESKTOP-3BA3T21:~/OS_labs/Lab hometask 10$ ./output6
Car 1 entered the bridge
Car 2 entered the bridge
Car 3 entered the bridge
Car 1 left the bridge after 2 seconds
Car 3 left the bridge after 1 seconds
Car 4 entered the bridge
Car 2 left the bridge after 2 seconds
Car 5 entered the bridge
Car 6 entered the bridge
Car 4 left the bridge after 2 seconds
Car 7 entered the bridge
Car 5 left the bridge after 3 seconds
Car 6 left the bridge after 2 seconds
Car 8 entered the bridge
Car 7 left the bridge after 2 seconds
Car 8 left the bridge after 1 seconds
```