NATIONAL TEXTILE UNIVERSITY

DEPARTMENT OF COMPUTER SCIENCE

SUBMITTED BY:

Eman Faisal	23-NTU-CS-1149

SECTION SE: 5th (A)

Operating System

SUBMITTED TO:

Sir Nasir Mahmood

SUBMISSION DATE:

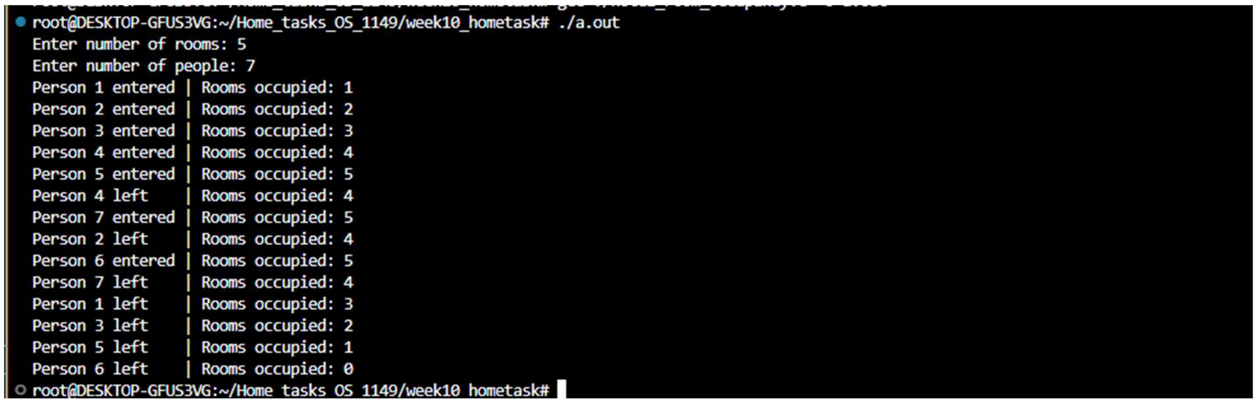
9th Dec 25

Home task week10

Task 1:

CODE:

```c
#include <stdio.h>
#include <pthread.h>
#include <semaphore.h>
#include <unistd.h>
#include <stdlib.h>
sem_t rooms;
int occupied = 0;
pthread_mutex_t lock;
void* rooms_place(void* arg) {
    int id = *(int*)arg;
    // Wait for a free room
    sem_wait(&rooms);
    pthread_mutex_lock(&lock);
    occupied++;
    printf("Person %d entered | Rooms occupied: %d\n", id, occupied);
    pthread_mutex_unlock(&lock);
    sleep(rand() % 3 + 1);
    pthread_mutex_lock(&lock);
    occupied--;
    printf("Person %d left    | Rooms occupied: %d\n", id, occupied);
    pthread_mutex_unlock(&lock);

    sem_post(&rooms);
    return NULL;
}
int main() {
    int N_ROOMS, N_PEOPLE;
    srand(time(NULL));
    printf("Enter number of rooms: ");
    scanf("%d", &N_ROOMS);
    printf("Enter number of people: ");
    scanf("%d", &N_PEOPLE);
    pthread_t t[N_PEOPLE];
    int ids[N_PEOPLE];
    // Initialize semaphore with user-defined rooms
    sem_init(&rooms, 0, N_ROOMS);
    pthread_mutex_init(&lock, NULL);
    // Create threads for each person
    for (int i = 0; i < N_PEOPLE; i++) {
        ids[i] = i + 1;
        pthread_create(&t[i], NULL, person, &ids[i]);
    }
    // Wait for all threads to finish
    for (int i = 0; i < N_PEOPLE; i++) {
        pthread_join(t[i], NULL);
    }
    sem_destroy(&rooms);
    pthread_mutex_destroy(&lock);
    return 0;
}
```



```
root@DESKTOP-GFUS3VG:~/Home_tasks_OS_1149/week10_hometask# ./a.out
Enter number of rooms: 5
Enter number of people: 7
Person 1 entered | Rooms occupied: 1
Person 2 entered | Rooms occupied: 2
Person 3 entered | Rooms occupied: 3
Person 4 entered | Rooms occupied: 4
Person 5 entered | Rooms occupied: 5
Person 4 left    | Rooms occupied: 4
Person 7 entered | Rooms occupied: 5
Person 2 left    | Rooms occupied: 4
Person 6 entered | Rooms occupied: 5
Person 7 left    | Rooms occupied: 4
Person 1 left    | Rooms occupied: 3
Person 3 left    | Rooms occupied: 2
Person 5 left    | Rooms occupied: 1
Person 6 left    | Rooms occupied: 0
root@DESKTOP-GFUS3VG:~/Home_tasks_OS_1149/week10_hometask#
```

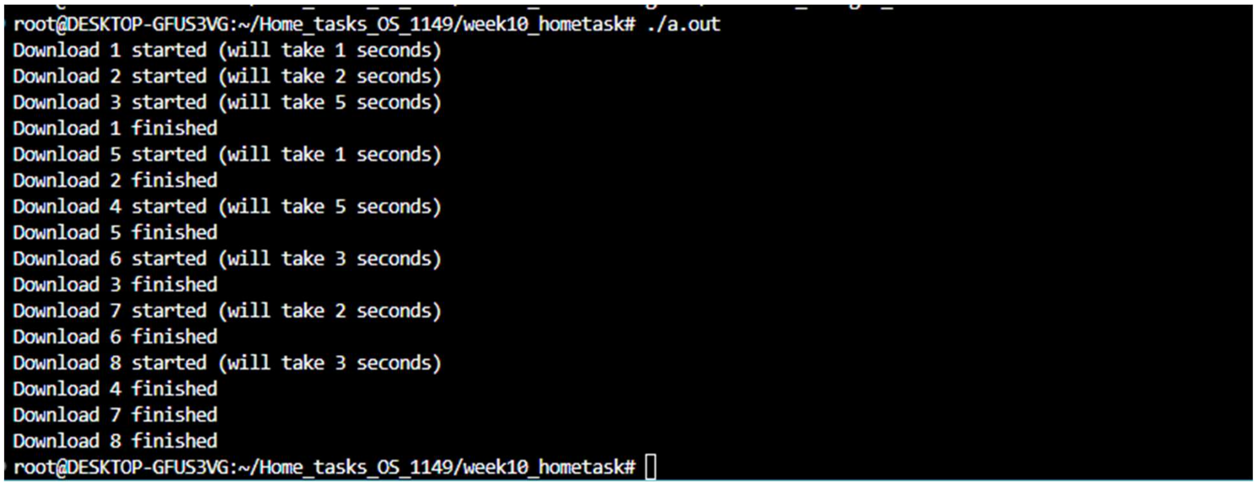
TASK 2:

CODE:

```
#include <stdio.h>

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

sem_t slots;

void* downloadFile(void* arg) {
    int id = *(int*)arg;
    int time_needed = rand() % 5 + 1;
    sem_wait(&slots);
```

```c
    printf("Download %d started (will take %d seconds)\n", id, time_needed);
    sleep(time_needed);

    printf("Download %d finished!!\n", id);
    sem_post(&slots);
    return NULL;
}

int main() {
    srand(time(NULL));

    sem_init(&slots, 0, 3); // Only 3 downloads at a time

    pthread_t threads[8];
    int ids[8];

    for (int i = 0; i < 8; i++) {
        ids[i] = i + 1;
        pthread_create(&threads[i], NULL, downloadFile, &ids[i]);
    }
    for (int i = 0; i < 8; i++) {
        pthread_join(threads[i], NULL);
    }
    sem_destroy(&slots);
    return 0;
}
```

OUTPUT:

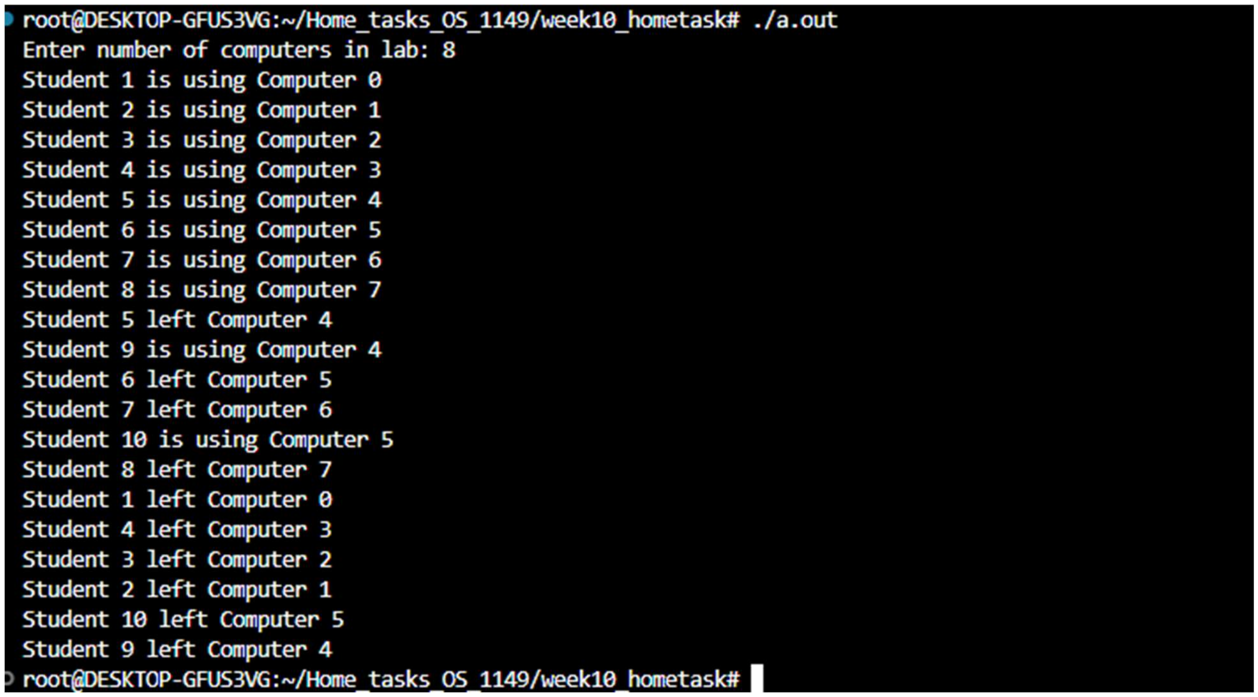


```
root@DESKTOP-GFUS3VG:~/Home_tasks_OS_1149/week10_hometask# ./a.out
Download 1 started (will take 1 seconds)
Download 2 started (will take 2 seconds)
Download 3 started (will take 5 seconds)
Download 1 finished
Download 5 started (will take 1 seconds)
Download 2 finished
Download 4 started (will take 5 seconds)
Download 5 finished
Download 6 started (will take 3 seconds)
Download 3 finished
Download 7 started (will take 2 seconds)
Download 6 finished
Download 8 started (will take 3 seconds)
Download 4 finished
Download 7 finished
Download 8 finished
root@DESKTOP-GFUS3VG:~/Home_tasks_OS_1149/week10_hometask#
```

TASK 3:

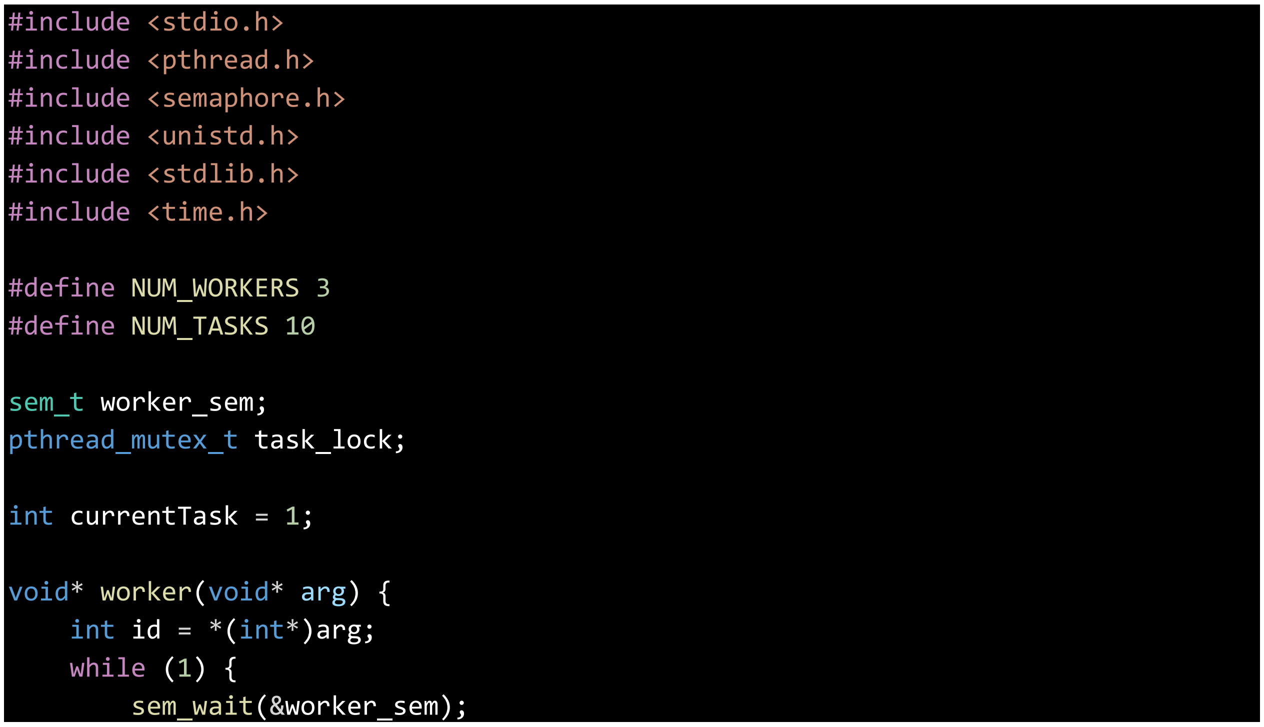
```c
#include <stdio.h>
#include <pthread.h>
#include <semaphore.h>
#include <unistd.h>
#include <stdlib.h>
#include <time.h>
#define NUM_STUDENTS 10   // total students
sem_t computers;          // counting semaphore for available computers
pthread_mutex_t lock;     // mutex for shared array
int *computerUsedBy;      // array: computerUsedBy[i] = student ID OR -1
int K; // number of computers
void* student(void* arg) {
    int id = *(int*)arg;
    // wait for a computer to be free
    sem_wait(&computers);
    pthread_mutex_lock(&lock);
    int assigned = -1;

    // find a free computer in the shared array
    for (int i = 0; i < K; i++) {
        if (computerUsedBy[i] == -1) {
            computerUsedBy[i] = id;  // assign this computer
            assigned = i;
            printf("Student %d is using Computer %d\n", id, i);
            break;
        }
    }

    pthread_mutex_unlock(&lock);

    // student uses the computer for random time (1-4 sec)
    sleep(rand() % 4 + 1);

    // student leaves
    pthread_mutex_lock(&lock);
    computerUsedBy[assigned] = -1;
    printf("Student %d left Computer %d\n", id, assigned);
    pthread_mutex_unlock(&lock);

    // free one computer
    sem_post(&computers);
    return NULL;
}

int main() {
    srand(time(NULL));

    printf("Enter number of computers in lab: ");
    scanf("%d", &K);

    computerUsedBy = malloc(K * sizeof(int));

    // mark all computers as free
    for (int i = 0; i < K; i++) {
        computerUsedBy[i] = -1;
    }

    // initialize semaphore & mutex
    sem_init(&computers, 0, K);
    pthread_mutex_init(&lock, NULL);

    pthread_t threads[NUM_STUDENTS];
    int ids[NUM_STUDENTS];
    // create student threads
    for (int i = 0; i < NUM_STUDENTS; i++) {
        ids[i] = i + 1;
        pthread_create(&threads[i], NULL, student, &ids[i]);
    }
    // wait for all to finish
    for (int i = 0; i < NUM_STUDENTS; i++) {
        pthread_join(threads[i], NULL);
    }
    // cleanup
    sem_destroy(&computers);
    pthread_mutex_destroy(&lock);
    free(computerUsedBy);

    return 0;
}
```

Output:



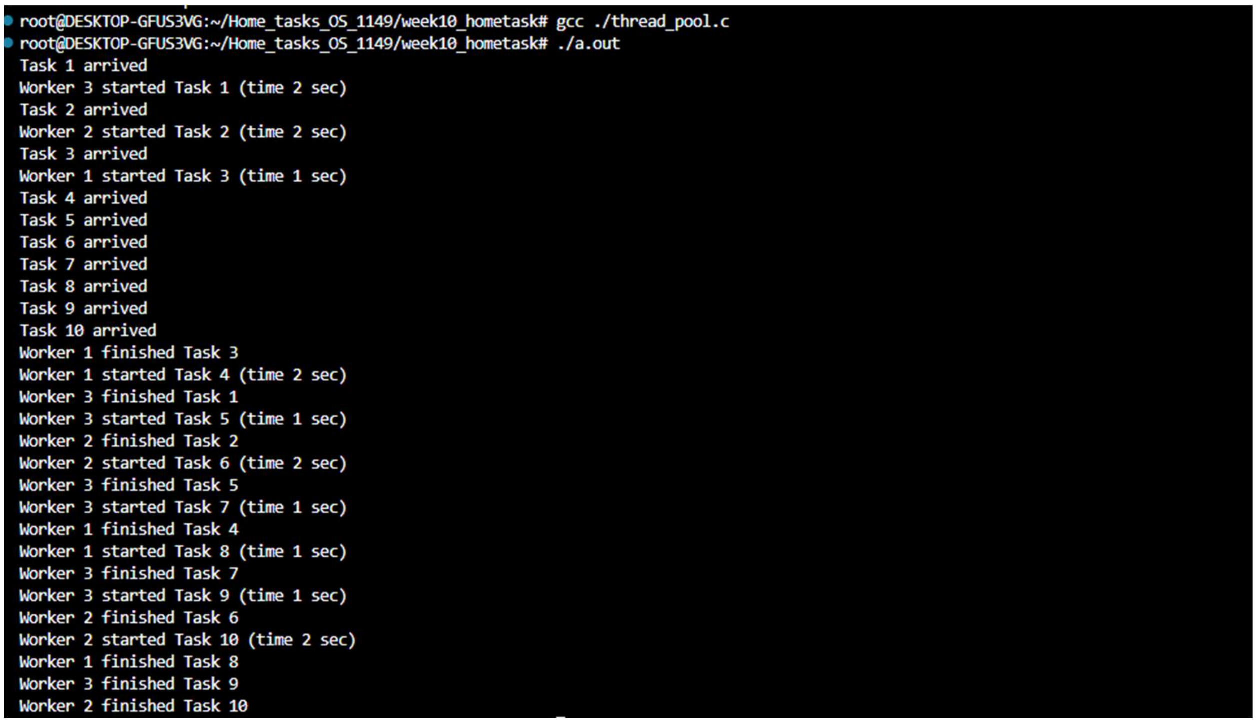
```
root@DESKTOP-GFUS3VG:~/Home_tasks_OS_1149/week10_hometask# ./a.out
Enter number of computers in lab: 8
Student 1 is using Computer 0
Student 2 is using Computer 1
Student 3 is using Computer 2
Student 4 is using Computer 3
Student 5 is using Computer 4
Student 6 is using Computer 5
Student 7 is using Computer 6
Student 8 is using Computer 7
Student 5 left Computer 4
Student 9 is using Computer 4
Student 6 left Computer 5
Student 7 left Computer 6
Student 10 is using Computer 5
Student 8 left Computer 7
Student 1 left Computer 0
Student 4 left Computer 3
Student 3 left Computer 2
Student 2 left Computer 1
Student 10 left Computer 5
Student 9 left Computer 4
root@DESKTOP-GFUS3VG:~/Home_tasks_OS_1149/week10_hometask#
```

TASK 4:



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

#define NUM_WORKERS 3
#define NUM_TASKS 10

sem_t worker_sem;
pthread_mutex_t task_lock;

int currentTask = 1;

void* worker(void* arg) {
    int id = *(int*)arg;
    while (1) {
        sem_wait(&worker_sem);
        pthread_mutex_lock(&task_lock);
        if (currentTask > NUM_TASKS) {
            pthread_mutex_unlock(&task_lock);
            sem_post(&worker_sem);
            break;
        }
        int taskID = currentTask;
        currentTask++;
        pthread_mutex_unlock(&task_lock);
        int time_needed = rand() % 2 + 1;
        printf("Worker %d started Task %d (time %d sec)\n",
               id, taskID, time_needed);

        sleep(time_needed);
        printf("Worker %d finished Task %d\n", id, taskID);
    }
    return NULL;
}

int main() {
    srand(time(NULL));

    pthread_t workers[NUM_WORKERS];
    int ids[NUM_WORKERS];

    sem_init(&worker_sem, 0, 0);
    pthread_mutex_init(&task_lock, NULL);

    for (int i = 0; i < NUM_WORKERS; i++) {
        ids[i] = i + 1;
        pthread_create(&workers[i], NULL, worker, &ids[i]);
    }

    for (int i = 1; i <= NUM_TASKS; i++) {
        printf("Task %d arrived\n", i);
        sem_post(&worker_sem);
        usleep(100000);
    }

    for (int i = 0; i < NUM_WORKERS; i++) {
        sem_post(&worker_sem);
    }
    for (int i = 0; i < NUM_WORKERS; i++) {
        pthread_join(workers[i], NULL);
    }
    sem_destroy(&worker_sem);
    pthread_mutex_destroy(&task_lock);

    return 0;
}
```

Output:

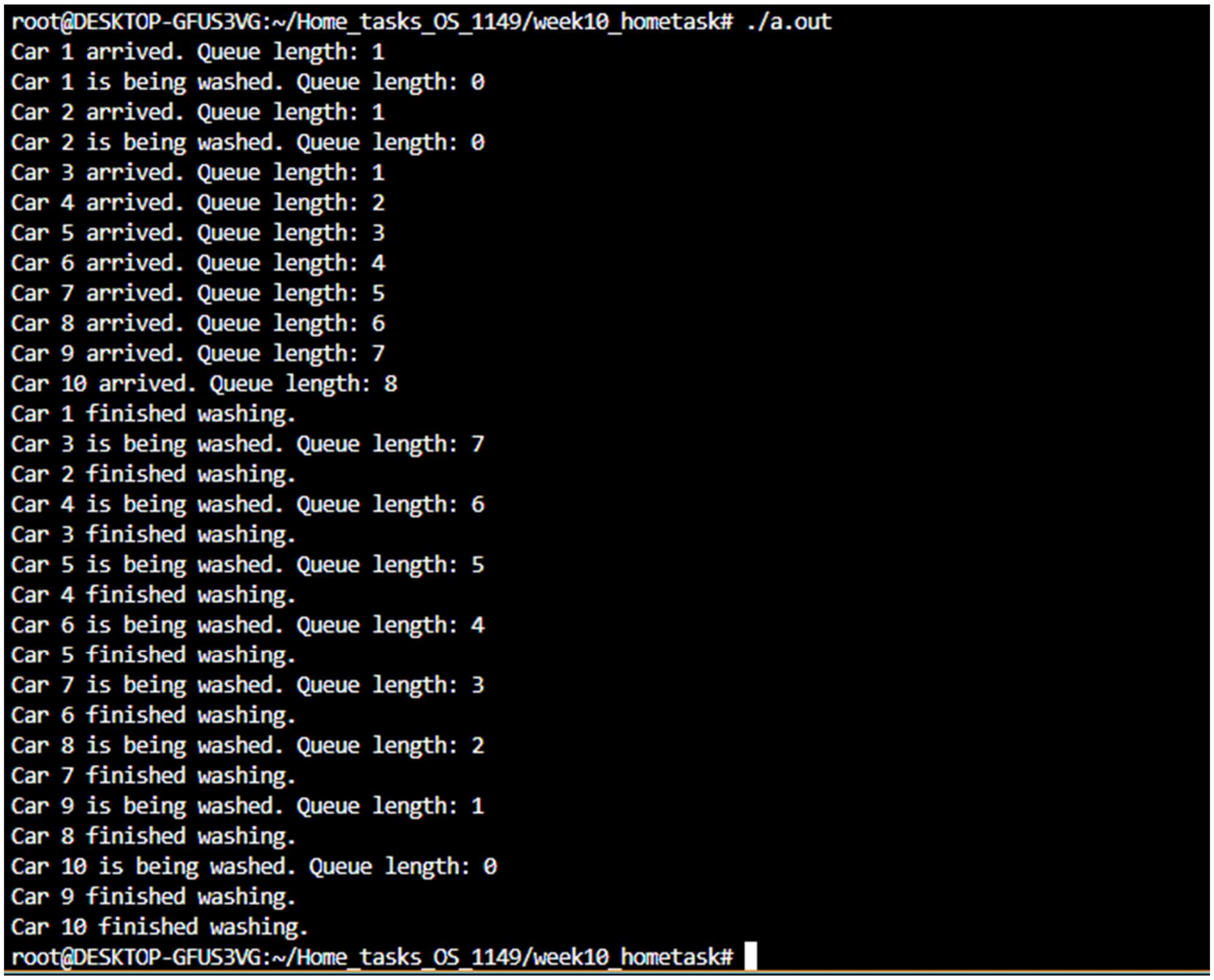


```
root@DESKTOP-GFUS3VG:~/Home_tasks_OS_1149/week10_hometask# gcc ./thread_pool.c
root@DESKTOP-GFUS3VG:~/Home_tasks_OS_1149/week10_hometask# ./a.out
Task 1 arrived
Worker 3 started Task 1 (time 2 sec)
Task 2 arrived
Worker 2 started Task 2 (time 2 sec)
Task 3 arrived
Worker 1 started Task 3 (time 1 sec)
Task 4 arrived
Task 5 arrived
Task 6 arrived
Task 7 arrived
Task 8 arrived
Task 9 arrived
Task 10 arrived
Worker 1 finished Task 3
Worker 1 started Task 4 (time 2 sec)
Worker 3 finished Task 1
Worker 3 started Task 5 (time 1 sec)
Worker 2 finished Task 2
Worker 2 started Task 6 (time 2 sec)
Worker 3 finished Task 5
Worker 3 started Task 7 (time 1 sec)
Worker 1 finished Task 4
Worker 1 started Task 8 (time 1 sec)
Worker 3 finished Task 7
Worker 3 started Task 9 (time 1 sec)
Worker 2 finished Task 6
Worker 2 started Task 10 (time 2 sec)
Worker 1 finished Task 8
Worker 3 finished Task 9
Worker 2 finished Task 10
```

TASK 5:

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

#define NUM_CARS 10   // total cars arriving

sem_t stations;       // semaphore for 2 wash stations
pthread_mutex_t qlock;
```

```c
int queueLength = 0;  // cars waiting in queue

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

    // Car arrives → joins queue
    pthread_mutex_lock(&qlock);
    queueLength++;
    printf("Car %d arrived. Queue length: %d\n", id, queueLength);
    pthread_mutex_unlock(&qlock);

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

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

    sleep(3); // washing takes 3 seconds

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

    sem_post(&stations); // free station

    return NULL;
}
int main() {
    srand(time(NULL));
    pthread_t cars[NUM_CARS];
    int ids[NUM_CARS];
    sem_init(&stations, 0, 2);     // 2 washing stations
    pthread_mutex_init(&qlock, NULL);
    // create car threads
    for (int i = 0; i < NUM_CARS; i++) {
        ids[i] = i + 1;
        pthread_create(&cars[i], NULL, carWash, &ids[i]);

        usleep(300000); // small delay so arrivals look realistic
    }
    // wait for all cars
    for (int i = 0; i < NUM_CARS; i++) {
        pthread_join(cars[i], NULL);
    }
    sem_destroy(&stations);
    pthread_mutex_destroy(&qlock);

    return 0;
}
```



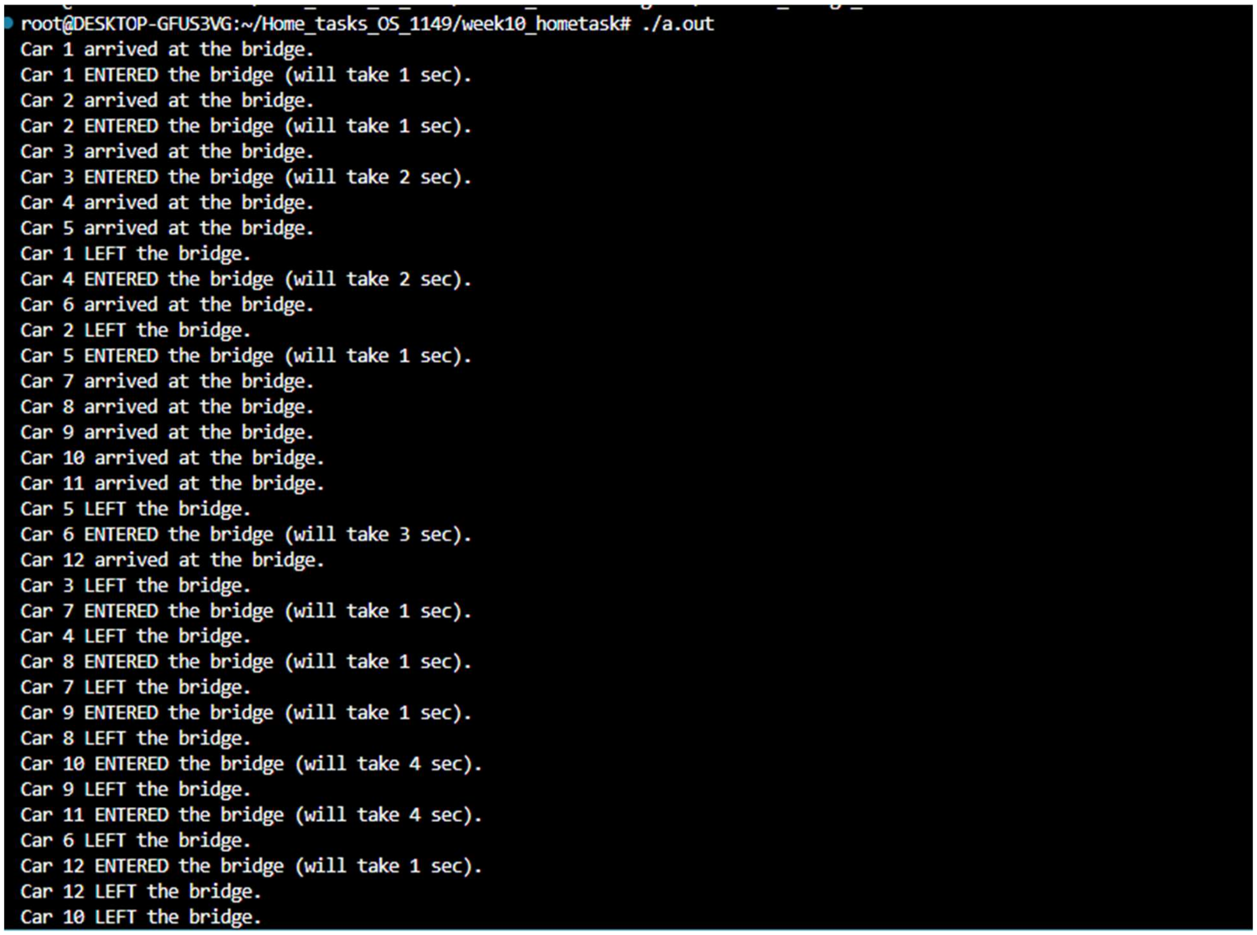
```
root@DESKTOP-GFUS3VG:~/Home_tasks_OS_1149/week10_hometask# ./a.out
Car 1 arrived. Queue length: 1
Car 1 is being washed. Queue length: 0
Car 2 arrived. Queue length: 1
Car 2 is being washed. Queue length: 0
Car 3 arrived. Queue length: 1
Car 4 arrived. Queue length: 2
Car 5 arrived. Queue length: 3
Car 6 arrived. Queue length: 4
Car 7 arrived. Queue length: 5
Car 8 arrived. Queue length: 6
Car 9 arrived. Queue length: 7
Car 10 arrived. Queue length: 8
Car 1 finished washing.
Car 3 is being washed. Queue length: 7
Car 2 finished washing.
Car 4 is being washed. Queue length: 6
Car 3 finished washing.
Car 5 is being washed. Queue length: 5
Car 4 finished washing.
Car 6 is being washed. Queue length: 4
Car 5 finished washing.
Car 7 is being washed. Queue length: 3
Car 6 finished washing.
Car 8 is being washed. Queue length: 2
Car 7 finished washing.
Car 9 is being washed. Queue length: 1
Car 8 finished washing.
Car 10 is being washed. Queue length: 0
Car 9 finished washing.
Car 10 finished washing.
root@DESKTOP-GFUS3VG:~/Home_tasks_OS_1149/week10_hometask#
```

TASK-6:

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

```c
#define NUM_CARS 12
sem_t bridge;
pthread_mutex_t print_lock;
void* car(void* arg) {
    int id = *(int*)arg;
    int time_needed = (rand() % 4) + 1; // 1-4 seconds
    pthread_mutex_lock(&print_lock);
    printf("Car %d arrived at the bridge.\n", id);
    pthread_mutex_unlock(&print_lock);
    sem_wait(&bridge);
    pthread_mutex_lock(&print_lock);
    printf("Car %d ENTERED the bridge (will take %d sec).\n", id, time_needed);
    pthread_mutex_unlock(&print_lock);
    sleep(time_needed);
    pthread_mutex_lock(&print_lock);
    printf("Car %d LEFT the bridge.\n", id);
    pthread_mutex_unlock(&print_lock);
    sem_post(&bridge);
    return NULL;
}

int main() {
    srand(time(NULL));

    pthread_t cars[NUM_CARS];
    int ids[NUM_CARS];

    sem_init(&bridge, 0, 3);
    pthread_mutex_init(&print_lock, NULL);

    for (int i = 0; i < NUM_CARS; i++) {
        ids[i] = i + 1;
        pthread_create(&cars[i], NULL, car, &ids[i]);

        usleep(200000);
    }

    for (int i = 0; i < NUM_CARS; i++) {
        pthread_join(cars[i], NULL);
    }
    sem_destroy(&bridge);
    pthread_mutex_destroy(&print_lock);
    return 0;
}
```

Output:



```
root@DESKTOP-GFUS3VG:~/Home_tasks_OS_1149/week10_hometask# ./a.out
Car 1 arrived at the bridge.
Car 1 ENTERED the bridge (will take 1 sec).
Car 2 arrived at the bridge.
Car 2 ENTERED the bridge (will take 1 sec).
Car 3 arrived at the bridge.
Car 3 ENTERED the bridge (will take 2 sec).
Car 4 arrived at the bridge.
Car 5 arrived at the bridge.
Car 1 LEFT the bridge.
Car 4 ENTERED the bridge (will take 2 sec).
Car 6 arrived at the bridge.
Car 2 LEFT the bridge.
Car 5 ENTERED the bridge (will take 1 sec).
Car 7 arrived at the bridge.
Car 8 arrived at the bridge.
Car 9 arrived at the bridge.
Car 10 arrived at the bridge.
Car 11 arrived at the bridge.
Car 5 LEFT the bridge.
Car 6 ENTERED the bridge (will take 3 sec).
Car 12 arrived at the bridge.
Car 3 LEFT the bridge.
Car 7 ENTERED the bridge (will take 1 sec).
Car 4 LEFT the bridge.
Car 8 ENTERED the bridge (will take 1 sec).
Car 7 LEFT the bridge.
Car 9 ENTERED the bridge (will take 1 sec).
Car 8 LEFT the bridge.
Car 10 ENTERED the bridge (will take 4 sec).
Car 9 LEFT the bridge.
Car 11 ENTERED the bridge (will take 4 sec).
Car 6 LEFT the bridge.
Car 12 ENTERED the bridge (will take 1 sec).
Car 12 LEFT the bridge.
Car 10 LEFT the bridge.
```