



# 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:

```
● ● ●
1 #include <stdio.h>
2 #include <pthread.h>
3 #include <semaphore.h>
4 #include <unistd.h>
5 #include <stdlib.h>
6
7 sem_t rooms;           // Counting semaphore for available rooms
8 pthread_mutex_t mutex; // protect the occupied counter
9
10 int occupied = 0;      // no of occupied rooms
11
12 void* person(void* arg) {
13     int id = *(int*)arg;
14
15     printf("Person %d is trying to enter...\n", id);
16
17     // Wait for an available room (decrement semaphore)
18     sem_wait(&rooms);
19
20     // Lock the counter update
21     pthread_mutex_lock(&mutex);
22     occupied++;
23     printf("Person %d entered. Rooms occupied: %d\n", id, occupied);
24     pthread_mutex_unlock(&mutex);
25
26     // Stay for random 1-3 seconds
27     int stay = (rand() % 3) + 1;
28     sleep(stay);
29
30     // Leaving the room
31     pthread_mutex_lock(&mutex);
32     occupied--;
33     printf("Person %d left. Rooms occupied: %d\n", id, occupied);
34     pthread_mutex_unlock(&mutex);
35
36     // Free the room (increment semaphore)
37     sem_post(&rooms);
38
39     return NULL;
40 }
41
42 int main() {
43     int N = 3;           // Number of rooms
44     int totalPeople = 10; // Total threads
45     pthread_t people[totalPeople];
46     int ids[totalPeople];
47
48     srand(time(NULL)); // For random stay time
49
50     // Initialize semaphore with N rooms
51     sem_init(&rooms, 0, N);
52
53     // Initialize mutex
54     pthread_mutex_init(&mutex, NULL);
55
56     // Create threads (people)
57     for(int i = 0; i < totalPeople; i++) {
58         ids[i] = i + 1;
59         pthread_create(&people[i], NULL, person, &ids[i]);
60     }
61
62     // Wait for all people to complete
63     for(int i = 0; i < totalPeople; i++) {
64         pthread_join(people[i], NULL);
65     }
66
67     // Cleanup
68     sem_destroy(&rooms);
69     pthread_mutex_destroy(&mutex);
70
71     return 0;
72 }
```

Output:

```
kashmirkj@Kashmirpc:~/Operating_System/Home_Task_Week10$ ./Example1
Person 2 entered. Rooms occupied: 2
Person 3 is trying to enter...
Person 3 entered. Rooms occupied: 3
Person 4 is trying to enter...
Person 5 is trying to enter...
Person 6 is trying to enter...
Person 7 is trying to enter...
Person 8 is trying to enter...
Person 9 is trying to enter...
Person 10 is trying to enter...
Person 3 left. Rooms occupied: 2
Person 4 entered. Rooms occupied: 3
Person 4 left. Rooms occupied: 2
Person 5 entered. Rooms occupied: 3
Person 1 left. Rooms occupied: 2
Person 2 left. Rooms occupied: 1
Person 7 entered. Rooms occupied: 2
Person 6 entered. Rooms occupied: 3
Person 7 left. Rooms occupied: 2
Person 8 entered. Rooms occupied: 3
Person 5 left. Rooms occupied: 2
Person 9 entered. Rooms occupied: 3
Person 6 left. Rooms occupied: 2
Person 10 entered. Rooms occupied: 3
Person 10 left. Rooms occupied: 2
Person 8 left. Rooms occupied: 1
Person 9 left. Rooms occupied: 0
```

## Q#2

### Exercise 2 – Download Manager Simulation

#### Scenario:

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

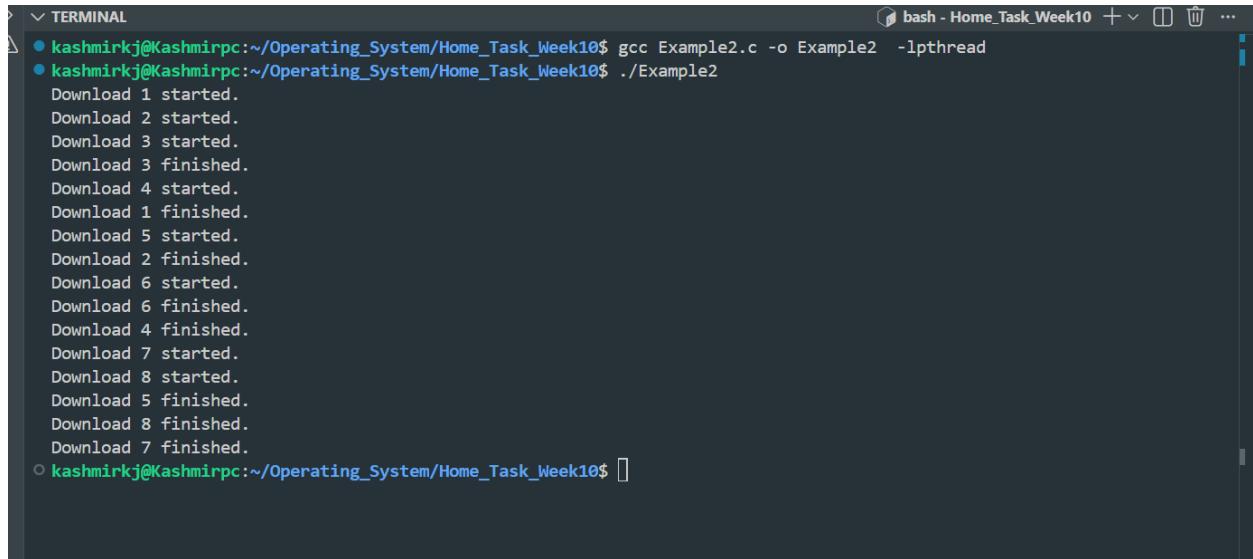
#### Tasks:

- 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:

```
 1 #include <stdio.h>
 2 #include <pthread.h>
 3 #include <semaphore.h>
 4 #include <unistd.h>
 5 #include <stdlib.h>
 6
 7 sem_t download_slots; // Semaphore to allow max 3 downloads at a time
 8
 9 void* download_file(void* arg) {
10     int id = *(int*)arg;
11
12     // Wait for an available download slot
13     sem_wait(&download_slots);
14
15     // Start download
16     printf("Download %d started.\n", id);
17
18     int time_needed = (rand() % 5) + 1; // Random 1-5 seconds
19     sleep(time_needed);
20
21     // End download
22     printf("Download %d finished.\n", id);
23
24     // Free the slot
25     sem_post(&download_slots);
26
27     return NULL;
28 }
29
30 int main() {
31     pthread_t downloads[8];
32     int ids[8];
33
34     srand(time(NULL));
35
36     // Initialize semaphore with 3 download slots
37     sem_init(&download_slots, 0, 3);
38
39     // Create 8 download threads
40     for(int i = 0; i < 8; i++) {
41         ids[i] = i + 1;
42         pthread_create(&downloads[i], NULL, download_file, &ids[i]);
43     }
44
45     // Wait for all downloads to complete
46     for(int i = 0; i < 8; i++) {
47         pthread_join(downloads[i], NULL);
48     }
49
50     // Clean up
51     sem_destroy(&download_slots);
52
53     return 0;
54 }
```

## Output:



```
TERMINAL
● kashmirkj@Kashmirpc:~/Operating_System/Home_Task_Week10$ gcc Example2.c -o Example2 -lpthread
● kashmirkj@Kashmirpc:~/Operating_System/Home_Task_Week10$ ./Example2
Download 1 started.
Download 2 started.
Download 3 started.
Download 3 finished.
Download 4 started.
Download 1 finished.
Download 5 started.
Download 2 finished.
Download 6 started.
Download 6 finished.
Download 4 finished.
Download 7 started.
Download 8 started.
Download 5 finished.
Download 8 finished.
Download 7 finished.
○ kashmirkj@Kashmirpc:~/Operating_System/Home_Task_Week10$
```

## Question #3

### **Exercise 3 – Library Computer Lab Access**

#### **Scenario:**

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

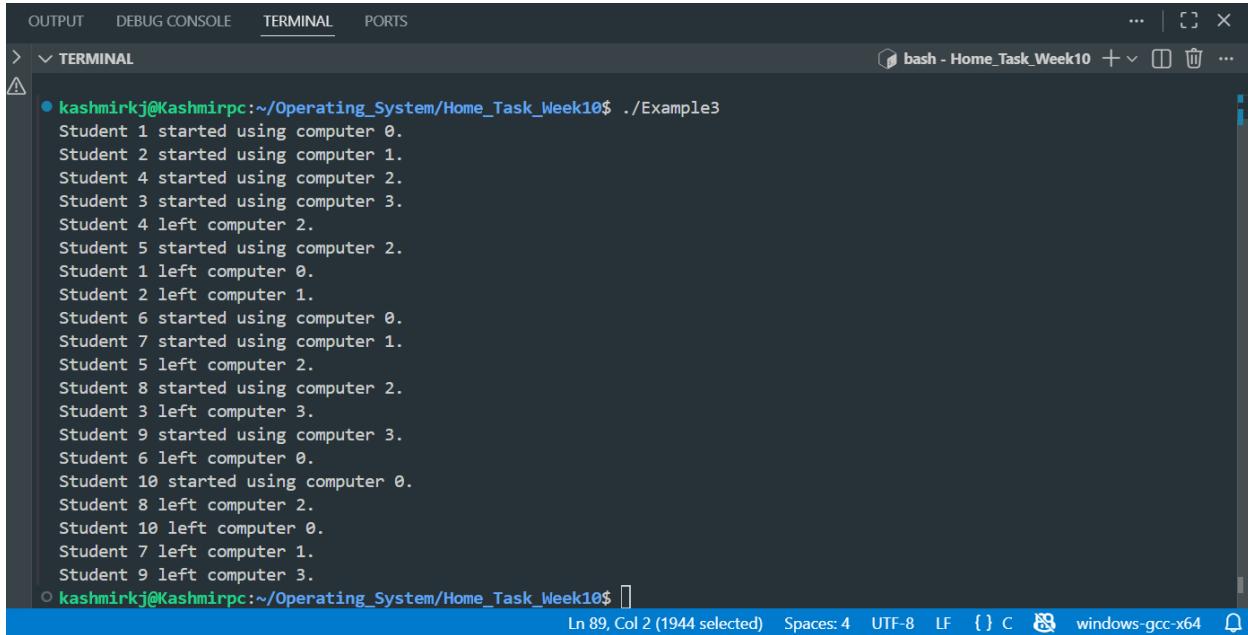
#### **Tasks:**

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

#### **Code:**

```
1 #include <stdio.h>
2 #include <pthread.h>
3 #include <semaphore.h>
4 #include <unistd.h>
5 #include <stdlib.h>
6
7 #define K 4           // Number of computers in the Lab
8 #define STUDENTS 10
9
10 sem_t lab_sem;           // Semaphore for available computers
11 pthread_mutex_t mutex;   // Protects shared array
12
13 int computers[K];        // -1 = free, else student ID
14
15 void* student(void* arg) {
16     int id = *(int*)arg;
17
18     // Wait for a free computer
19     sem_wait(&lab_sem);
20
21     // Find and assign the free computer
22     pthread_mutex_lock(&mutex);
23
24     int assigned_computer = -1;
25     for (int i = 0; i < K; i++) {
26         if (computers[i] == -1) {
27             computers[i] = id;
28             assigned_computer = i;
29             break;
30         }
31     }
32 }
33
34 printf("Student %d started using computer %d.\n", id, assigned_computer);
35
36 pthread_mutex_unlock(&mutex);
37
38 // Use computer for random 1-3 seconds
39 int use_time = (rand() % 3) + 1;
40 sleep(use_time);
41
42 // Student Leaves the computer
43 pthread_mutex_lock(&mutex);
44
45 computers[assigned_computer] = -1;
46 printf("Student %d left computer %d.\n", id, assigned_computer);
47
48 pthread_mutex_unlock(&mutex);
49
50 // Free a computer
51 sem_post(&lab_sem);
52
53 return NULL;
54 }
55
56 int main() {
57     pthread_t threads[STUDENTS];
58     int ids[STUDENTS];
59
60     srand(time(NULL));
61
62     // Initialize computers array to -1 (all free)
63     for (int i = 0; i < K; i++) {
64         computers[i] = -1;
65     }
66
67     // Initialize semaphore
68     sem_init(&lab_sem, 0, K);
69
70     // Initialize mutex
71     pthread_mutex_init(&mutex, NULL);
72
73     // Create student threads
74     for (int i = 0; i < STUDENTS; i++) {
75         ids[i] = i + 1;
76         pthread_create(&threads[i], NULL, student, &ids[i]);
77     }
78
79     // Join all students
80     for (int i = 0; i < STUDENTS; i++) {
81         pthread_join(threads[i], NULL);
82     }
83
84     // Cleanup
85     sem_destroy(&lab_sem);
86     pthread_mutex_destroy(&mutex);
87
88     return 0;
89 }
```

## Output:



The screenshot shows a terminal window with the following output:

```
kashmirkj@Kashmirpc:~/Operating_System/Home_Task_Week10$ ./Example3
Student 1 started using computer 0.
Student 2 started using computer 1.
Student 4 started using computer 2.
Student 3 started using computer 3.
Student 4 left computer 2.
Student 5 started using computer 2.
Student 1 left computer 0.
Student 2 left computer 1.
Student 6 started using computer 0.
Student 7 started using computer 1.
Student 5 left computer 2.
Student 8 started using computer 2.
Student 3 left computer 3.
Student 9 started using computer 3.
Student 6 left computer 0.
Student 10 started using computer 0.
Student 8 left computer 2.
Student 10 left computer 0.
Student 7 left computer 1.
Student 9 left computer 3.
kashmirkj@Kashmirpc:~/Operating_System/Home_Task_Week10$
```

The terminal window has tabs for OUTPUT, DEBUG CONSOLE, TERMINAL (which is selected), and PORTS. The status bar at the bottom showsLn 89, Col 2 (1944 selected) Spaces: 4 UTF-8 LF { } C windows-gcc-x64

## 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:

```
● ○ ●
1 #include <stdio.h>
2 #include <pthread.h>
3 #include <semaphore.h>
4 #include <unistd.h>
5 #include <stdlib.h>
6
7 #define WORKERS 3
8 #define TASKS 10
9
10 sem_t worker_sem;           // Controls worker availability
11
12 pthread_mutex_t mutex;     // For printing / tracking safely
13
14 void* task(void* arg) {
15     int id = *(int*)arg;
16
17     // Wait until a worker is free
18     sem_wait(&worker_sem);
19
20     pthread_mutex_lock(&mutex);
21     printf("Task %d started by a worker.\n", id);
22     pthread_mutex_unlock(&mutex);
23
24     // Simulate task work
25     int t = (rand() % 2) + 1;
26     sleep(t);
27
28     pthread_mutex_lock(&mutex);
29     printf("Task %d finished.\n", id);
30     pthread_mutex_unlock(&mutex);
31
32     // Release worker
33     sem_post(&worker_sem);
34
35     return NULL;
36 }
37
38 int main() {
39     pthread_t tasks[TASKS];
40     int ids[TASKS];
41
42     srand(time(NULL));
43
44     // Initialize semaphore with 3 workers available
45     sem_init(&worker_sem, 0, WORKERS);
46
47     // Initialize mutex
48     pthread_mutex_init(&mutex, NULL);
49
50     // Create task threads
51     for (int i = 0; i < TASKS; i++) {
52         ids[i] = i + 1;
53         pthread_create(&tasks[i], NULL, task, &ids[i]);
54     }
55
56     // Wait for all tasks to finish
57     for (int i = 0; i < TASKS; i++) {
58         pthread_join(tasks[i], NULL);
59     }
60
61     // Cleanup
62     sem_destroy(&worker_sem);
63     pthread_mutex_destroy(&mutex);
64
65     return 0;
66 }
```

## Output:

```
● kashmirkj@Kashmirpc:~/Operating_System/Home_Task_Week10$ ./Example4
Task 1 started by a worker.
Task 2 started by a worker.
Task 3 started by a worker.
Task 2 finished.
Task 4 started by a worker.
Task 3 finished.
Task 1 finished.
Task 5 started by a worker.
Task 6 started by a worker.
Task 4 finished.
Task 7 started by a worker.
Task 7 finished.
Task 8 started by a worker.
Task 5 finished.
Task 9 started by a worker.
Task 6 finished.
Task 10 started by a worker.
Task 8 finished.
Task 9 finished.
Task 10 finished.
○ kashmirkj@Kashmirpc:~/Operating_System/Home_Task_Week10$
```

## Q#5

### Exercise 5 – Car Wash Station

#### Scenario:

Car wash has **two washing stations**.

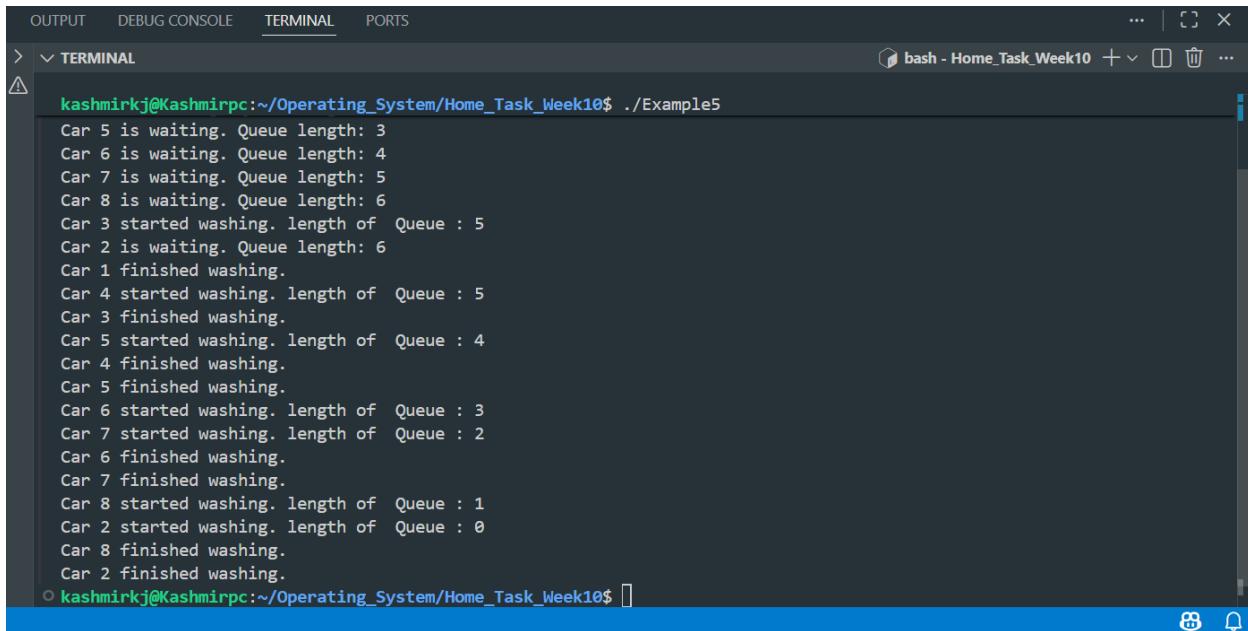
#### Tasks:

- 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:

```
● ● ●
1 #include <stdio.h>
2 #include <pthread.h>
3 #include <semaphore.h>
4 #include <unistd.h>
5
6 #define STATIONS 2
7 #define CARS 8
8
9 sem_t wash_sem;           // Controls washing station availability
10 pthread_mutex_t mutex;    // Protects queue counter
11
12 int waiting = 0;          // queue Length tracker
13
14 void* car(void* arg) {
15     int id = *(int*)arg;
16
17     // Car enters waiting queue
18     pthread_mutex_lock(&mutex);
19     waiting++;
20     printf("Car %d is waiting. Queue length: %d\n", id, waiting);
21     pthread_mutex_unlock(&mutex);
22
23     // Wait for washing station
24     sem_wait(&wash_sem);
25
26     // Car gets a washing station → remove from queue
27     pthread_mutex_lock(&mutex);
28     waiting--;
29     printf("Car %d started washing. length of Queue : %d\n", id, waiting);
30     pthread_mutex_unlock(&mutex);
31
32     // Washing for 3 seconds
33     sleep(3);
34
35     printf("Car %d finished washing.\n", id);
36
37     // Free the washing station
38     sem_post(&wash_sem);
39
40     return NULL;
41 }
42
43 int main() {
44     pthread_t cars[CARS];
45     int ids[CARS];
46
47     // Initialize semaphore with 2 stations
48     sem_init(&wash_sem, 0, STATIONS);
49
50     // Initialize mutex
51     pthread_mutex_init(&mutex, NULL);
52
53     // Create car threads
54     for (int i = 0; i < CARS; i++) {
55         ids[i] = i + 1;
56         pthread_create(&cars[i], NULL, car, &ids[i]);
57     }
58
59     // Wait for all cars
60     for (int i = 0; i < CARS; i++) {
61         pthread_join(cars[i], NULL);
62     }
63
64     // Cleanup
65     sem_destroy(&wash_sem);
66     pthread_mutex_destroy(&mutex);
67
68     return 0;
69 }
70 }
```

## Output:



```
kashmirkj@Kashmirpc:~/Operating_System/Home_Task_Week10$ ./Example5
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 3 started washing. length of Queue : 5
Car 2 is waiting. Queue length: 6
Car 1 finished washing.
Car 4 started washing. length of Queue : 5
Car 3 finished washing.
Car 5 started washing. length of Queue : 4
Car 4 finished washing.
Car 5 finished washing.
Car 6 started washing. length of Queue : 3
Car 7 started washing. length of Queue : 2
Car 6 finished washing.
Car 7 finished washing.
Car 8 started washing. length of Queue : 1
Car 2 started washing. length of Queue : 0
Car 8 finished washing.
Car 2 finished washing.
o kashmirkj@Kashmirpc:~/Operating_System/Home_Task_Week10$
```

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:

```
● ● ●
1 #include <stdio.h>
2 #include <pthread.h>
3 #include <semaphore.h>
4 #include <unistd.h>
5 #include <stdlib.h>
6
7 #define MAX_ON_BRIDGE 3
8 #define TOTAL_CARS 10
9
10 sem_t bridge_sem; // Controls number of cars on bridge
11 pthread_mutex_t print_mutex; // For clean printing
12
13 void* car(void* arg) {
14     int id = *(int*)arg;
15
16     pthread_mutex_lock(&print_mutex);
17     printf("Car %d is waiting to enter the bridge...\n", id);
18     pthread_mutex_unlock(&print_mutex);
19
20     // Wait for permission to enter bridge
21     sem_wait(&bridge_sem);
22
23     pthread_mutex_lock(&print_mutex);
24     printf("Car %d STARTED crossing the bridge.\n", id);
25     pthread_mutex_unlock(&print_mutex);
26
27     // Random crossing time (1-4 seconds)
28     int t = (rand() % 4) + 1;
29     sleep(t);
30
31     pthread_mutex_lock(&print_mutex);
32     printf("Car %d FINISHED crossing in %d seconds.\n", id, t);
33     pthread_mutex_unlock(&print_mutex);
34
35     // Free space on the bridge
36     sem_post(&bridge_sem);
37
38     return NULL;
39 }
40
41 int main() {
42     pthread_t cars[TOTAL_CARS];
43     int ids[TOTAL_CARS];
44
45     srand(time(NULL));
46
47     // Semaphore allows 3 cars at a time
48     sem_init(&bridge_sem, 0, MAX_ON_BRIDGE);
49
50     // Initialize mutex
51     pthread_mutex_init(&print_mutex, NULL);
52
53     // Create car threads
54     for (int i = 0; i < TOTAL_CARS; i++) {
55         ids[i] = i + 1;
56         pthread_create(&cars[i], NULL, car, &ids[i]);
57     }
58
59     // Wait for all cars
60     for (int i = 0; i < TOTAL_CARS; i++) {
61         pthread_join(cars[i], NULL);
62     }
63
64     // Cleanup
65     sem_destroy(&bridge_sem);
66     pthread_mutex_destroy(&print_mutex);
67
68     return 0;
69 }
```

## Output:

```
✓ TERMINAL
• 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$
```