



**National Textile University Department of Computer Science Subject: Operating System**

---

**Submitted to: Nasir Mahmood**

---

**Submitted by: Ayema**

---

**Reg number:23-NTU-CS-1142**

---

**Lab : 10 (HW)**

---

**Semester:5th**

## Task 1: Hotel Room Occupancy Problem

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <pthread.h>
4  #include <semaphore.h>
5  #include <unistd.h> // for sleep
6  #include <time.h>
7
8  #define N 5 // Number of rooms
9  #define TOTAL_PEOPLE 10 // Total number of people trying to enter
10
11 sem_t rooms_semaphore; // Counting semaphore for rooms
12 int current_occupancy = 0; // Track occupancy
13 pthread_mutex_t occupancy_mutex; // Mutex to safely update occupancy
14
15 void* person_thread(void* arg) {
16     int person_id = *(int*)arg;
17     free(arg); // Free dynamically allocated memory
18
19     printf("Person %d is waiting to enter.\n", person_id);
20
21     // Wait for a room
22     sem_wait(&rooms_semaphore);
23
24     // Update occupancy
25     pthread_mutex_lock(&occupancy_mutex);
26     current_occupancy++;
27     printf("Person %d entered. Current occupancy: %d/%d\n", person_id, current_occupancy, N);
28     pthread_mutex_unlock(&occupancy_mutex);
29
30     // Stay for 1-3 seconds
31     int stay_time = (rand() % 3) + 1;
32     sleep(stay_time);
33
34     // Leave room
35     pthread_mutex_lock(&occupancy_mutex);
36     current_occupancy--;
37     printf("Person %d left. Current occupancy: %d/%d\n", person_id, current_occupancy, N);
38     pthread_mutex_unlock(&occupancy_mutex);
39
40     // Release the room
41     sem_post(&rooms_semaphore);
42
43     pthread_exit(NULL);
44 }
45
46 int main() {
47     srand(time(NULL)); // Seed random number generator
48
49     pthread_t threads[TOTAL_PEOPLE];
50     sem_init(&rooms_semaphore, 0, N); // Initialize semaphore with N rooms
51     pthread_mutex_init(&occupancy_mutex, NULL);
52
53     // Create threads for each person
54     for (int i = 0; i < TOTAL_PEOPLE; i++) {
55         int* id = malloc(sizeof(int));
56         *id = i + 1;
57         pthread_create(&threads[i], NULL, person_thread, id);
58     }
59
60     // Wait for all threads to finish
61     for (int i = 0; i < TOTAL_PEOPLE; i++) {
62         pthread_join(threads[i], NULL);
63     }
64
65     // Clean up
66     sem_destroy(&rooms_semaphore);
67     pthread_mutex_destroy(&occupancy_mutex);
68
69     printf("All people have finished their stay.\n");
70     return 0;
71 }
72
```

```
15 void* person_thread(void* arg) {
42
43     pthread_exit(NULL);
44 }
45
46 int main() {
    ...
}

Person 4 is waiting to enter.
Person 5 is waiting to enter.
Person 1 entered. Current occupancy: 1/5
Person 2 entered. Current occupancy: 2/5
Person 3 entered. Current occupancy: 3/5
Person 6 is waiting to enter.
Person 5 entered. Current occupancy: 4/5
Person 4 entered. Current occupancy: 5/5
Person 8 is waiting to enter.
Person 7 is waiting to enter.
Person 9 is waiting to enter.
Person 10 is waiting to enter.
Person 1 left. Current occupancy: 4/5
Person 6 entered. Current occupancy: 5/5
Person 2 left. Current occupancy: 4/5
Person 3 left. Current occupancy: 3/5
Person 8 entered. Current occupancy: 4/5
Person 7 entered. Current occupancy: 5/5
Person 4 left. Current occupancy: 4/5
Person 9 entered. Current occupancy: 5/5
Person 8 left. Current occupancy: 4/5
Person 6 left. Current occupancy: 3/5
Person 7 left. Current occupancy: 2/5
Person 10 entered. Current occupancy: 3/5
Person 5 left. Current occupancy: 2/5
Person 9 left. Current occupancy: 1/5
Person 10 left. Current occupancy: 0/5
All people have finished their stay.
```

## Task 2: Download Manager Simulation

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <pthread.h>
4  #include <semaphore.h>
5  #include <unistd.h> // for sleep
6  #include <time.h>
7
8  #define MAX_CONCURRENT_DOWNLOADS 3
9  #define TOTAL_DOWNLOADS 8
10
11  sem_t download_semaphore;    // Counting semaphore
12  pthread_mutex_t print_mutex; // Mutex to safely print messages
13
14  void* download_thread(void* arg) {
15      int file_id = *(int*)arg;
16      free(arg); // free allocated memory
17
18      // Wait for a free slot to download
19      sem_wait(&download_semaphore);
20
21      // Print start message
22      pthread_mutex_lock(&print_mutex);
23      printf("Download %d started.\n", file_id);
24      pthread_mutex_unlock(&print_mutex);
25
26      // Simulate download (1-5 seconds)
27      int download_time = (rand() % 5) + 1;
28      sleep(download_time);
29
30      // Print end message
31      pthread_mutex_lock(&print_mutex);
32      printf("Download %d finished. (Took %d seconds)\n", file_id, download_time);
33      pthread_mutex_unlock(&print_mutex);
34
35      // Release semaphore slot
36      sem_post(&download_semaphore);
37
38      pthread_exit(NULL);
39  }
40
41  int main() {
42      srand(time(NULL)); // Seed for random times
43
44      pthread_t threads[TOTAL_DOWNLOADS];
45
46      // Initialize semaphore with 3 slots
47      sem_init(&download_semaphore, 0, MAX_CONCURRENT_DOWNLOADS);
48      pthread_mutex_init(&print_mutex, NULL);
49
50      // Create download threads
51      for (int i = 0; i < TOTAL_DOWNLOADS; i++) {
52          int* id = malloc(sizeof(int));
53          *id = i + 1;
54          pthread_create(&threads[i], NULL, download_thread, id);
55      }
56
57      // Wait for all threads to finish
58      for (int i = 0; i < TOTAL_DOWNLOADS; i++) {
59          pthread_join(threads[i], NULL);
60      }
61
62      // Clean up
63      sem_destroy(&download_semaphore);
64      pthread_mutex_destroy(&print_mutex);
65
66      printf("All downloads completed.\n");
67      return 0;
68  }
69

```

The screenshot displays the Visual Studio Code interface with a C program being edited and executed. The Explorer panel on the left shows the project structure for 'OPERATING-SYSTEM-1 [WSL: UBUNTU-24.04]', including files like '1142-lab10(HW)', 'task1.c', 'task2.c', and 'task2.out'. The main editor shows the code for 'task2.c', which defines a 'download\_thread' function and a 'main' function that uses 'srand' for randomization. The TERMINAL panel at the bottom shows the execution of 'task1.out' and 'task2.out'. The output of 'task1.out' shows occupancy counts for 10 people. The output of 'task2.out' shows the execution of 8 download threads with their respective start and finish times.

```
14 void* download_thread(void* arg) {
39 }
40
41 int main() {
42     srand(time(NULL)); // Seed for random times
43     ...
44 }

ayema@DESKTOP-1VHKS6:~/1142-lab3/Operating-system-1/1142-lab3/Operating-system/Operating-system-1/1142-lab10(HW)$ ./task1.o
ut
Person 6 left. Current occupancy: 3/5
Person 7 left. Current occupancy: 2/5
Person 10 entered. Current occupancy: 3/5
Person 5 left. Current occupancy: 2/5
Person 9 left. Current occupancy: 1/5
Person 10 left. Current occupancy: 0/5
All people have finished their stay.
ayema@DESKTOP-1VHKS6:~/1142-lab3/Operating-system-1/1142-lab3/Operating-system/Operating-system-1/1142-lab10(HW)$ gcc task2.c -o task2.out -lpthread
ayema@DESKTOP-1VHKS6:~/1142-lab3/Operating-system-1/1142-lab3/Operating-system/Operating-system-1/1142-lab10(HW)$ ./task2.o
ut
Download 1 started.
Download 2 started.
Download 3 started.
Download 3 finished. (Took 4 seconds)
Download 2 finished. (Took 4 seconds)
Download 4 started.
Download 5 started.
Download 1 finished. (Took 5 seconds)
Download 6 started.
Download 5 finished. (Took 1 seconds)
Download 8 started.
Download 4 finished. (Took 2 seconds)
Download 7 started.
Download 6 finished. (Took 2 seconds)
Download 7 finished. (Took 1 seconds)
Download 8 finished. (Took 5 seconds)
All downloads completed.
ayema@DESKTOP-1VHKS6:~/1142-lab3/Operating-system-1/1142-lab3/Operating-system/Operating-system-1/1142-lab10(HW)$
```

### Task 3: Library Computer Lab Access

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <pthread.h>
4  #include <semaphore.h>
5  #include <unistd.h>
6
7  #define NUM_COMPUTERS 4
8  #define NUM_STUDENTS 8
9
10 sem_t computers_sem;
11 pthread_mutex_t lab_mutex;
12 int computer_in_use[NUM_COMPUTERS] = {0}; // 0=free, 1=occupied
13
14 void* student_thread(void* arg) {
15     int student_id = *(int*)arg;
16     free(arg);
17
18     printf("Student %d is waiting for a computer.\n", student_id);
19     sem_wait(&computers_sem); // Wait for available computer
20
21     pthread_mutex_lock(&lab_mutex);
22     int assigned = -1;
23     for (int i = 0; i < NUM_COMPUTERS; i++) {
24         if (!computer_in_use[i]) {
25             computer_in_use[i] = 1;
26             assigned = i;
27             break;
28         }
29     }
30     printf("Student %d is using computer %d\n", student_id, assigned);
31     pthread_mutex_unlock(&lab_mutex);
32
33     sleep(rand() % 3 + 1); // Using computer 1-3 seconds
34
35     pthread_mutex_lock(&lab_mutex);
36     computer_in_use[assigned] = 0;
37     printf("Student %d left computer %d\n", student_id, assigned);
38     pthread_mutex_unlock(&lab_mutex);
39
40     sem_post(&computers_sem); // Free the computer
41     pthread_exit(NULL);
42 }
43
44 int main() {
45     srand(time(NULL));
46     pthread_t students[NUM_STUDENTS];
47
48     sem_init(&computers_sem, 0, NUM_COMPUTERS);
49     pthread_mutex_init(&lab_mutex, NULL);
50
51     for (int i = 0; i < NUM_STUDENTS; i++) {
52         int* id = malloc(sizeof(int));
53         *id = i + 1;
54         pthread_create(&students[i], NULL, student_thread, id);
55     }
56
57     for (int i = 0; i < NUM_STUDENTS; i++) {
58         pthread_join(students[i], NULL);
59     }
60
61     sem_destroy(&computers_sem);
62     pthread_mutex_destroy(&lab_mutex);
63
64     printf("All students finished using the lab.\n");
65     return 0;
66 }
67

```

```
14 void* student_thread(void* arg) {
15     printf("Student %d left computer %d\n", student_id, assigned);
16     pthread_mutex_unlock(&lab_mutex);
17
18     sem_post(&computers_sem); // Free the computer
19     pthread_exit(NULL);
20 }
```

ayema@DESKTOP-1VHCS6:~/1142-lab3/Operating-system-1/1142-lab3/Operating-system-1/1142-lab10(HW)\$ cd "1142-lab10(HW)"

ayema@DESKTOP-1VHCS6:~/1142-lab3/Operating-system-1/1142-lab3/Operating-system-1/1142-lab10(HW)\$ gcc task3.c -o task3.out -lpthread

ayema@DESKTOP-1VHCS6:~/1142-lab3/Operating-system-1/1142-lab3/Operating-system-1/1142-lab10(HW)\$ ./task3.out

Student 2 is waiting for a computer.

Student 2 is using computer 0

Student 1 is waiting for a computer.

Student 1 is using computer 1

Student 3 is waiting for a computer.

Student 3 is using computer 2

Student 5 is waiting for a computer.

Student 5 is using computer 0

Student 6 is waiting for a computer.

Student 4 is waiting for a computer.

Student 7 is waiting for a computer.

Student 8 is waiting for a computer.

Student 2 left computer 0

Student 6 is using computer 0

Student 3 left computer 2

Student 4 is using computer 2

Student 6 left computer 0

Student 7 is using computer 0

Student 4 left computer 2

Student 8 is using computer 2

Student 1 left computer 1

Student 5 left computer 0

Student 7 left computer 0

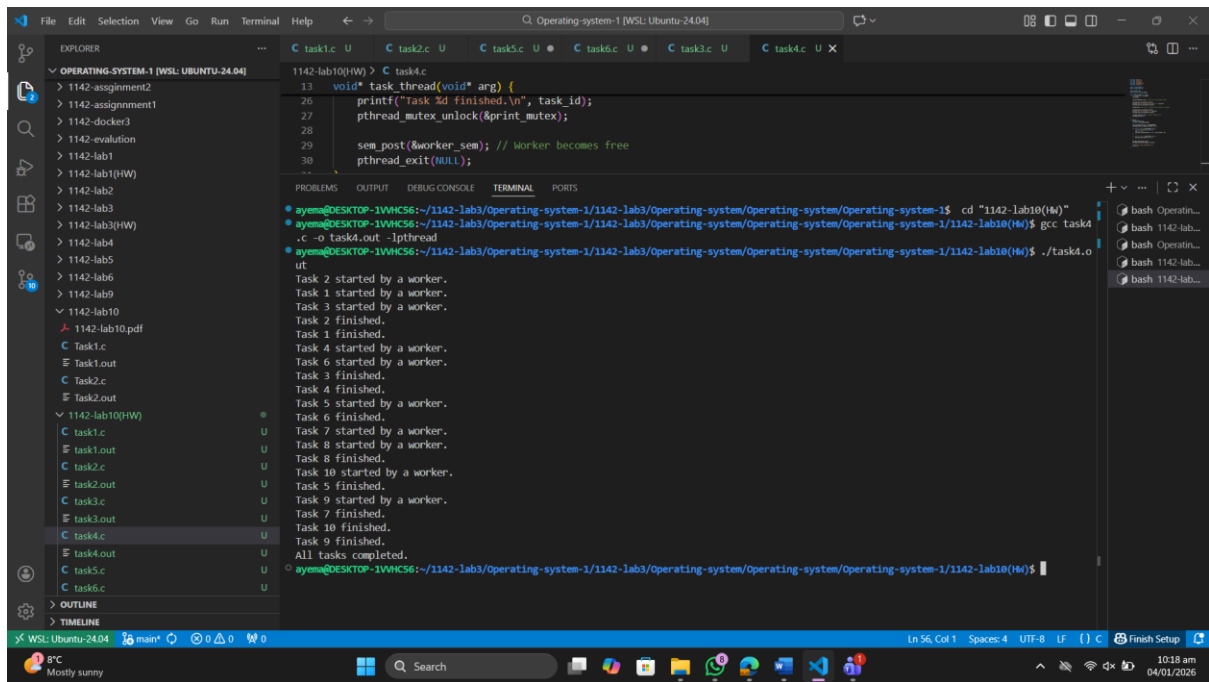
Student 8 left computer 2

All students Finished using the lab.

## Task 4: Thread Pool / Worker Pool Simulation



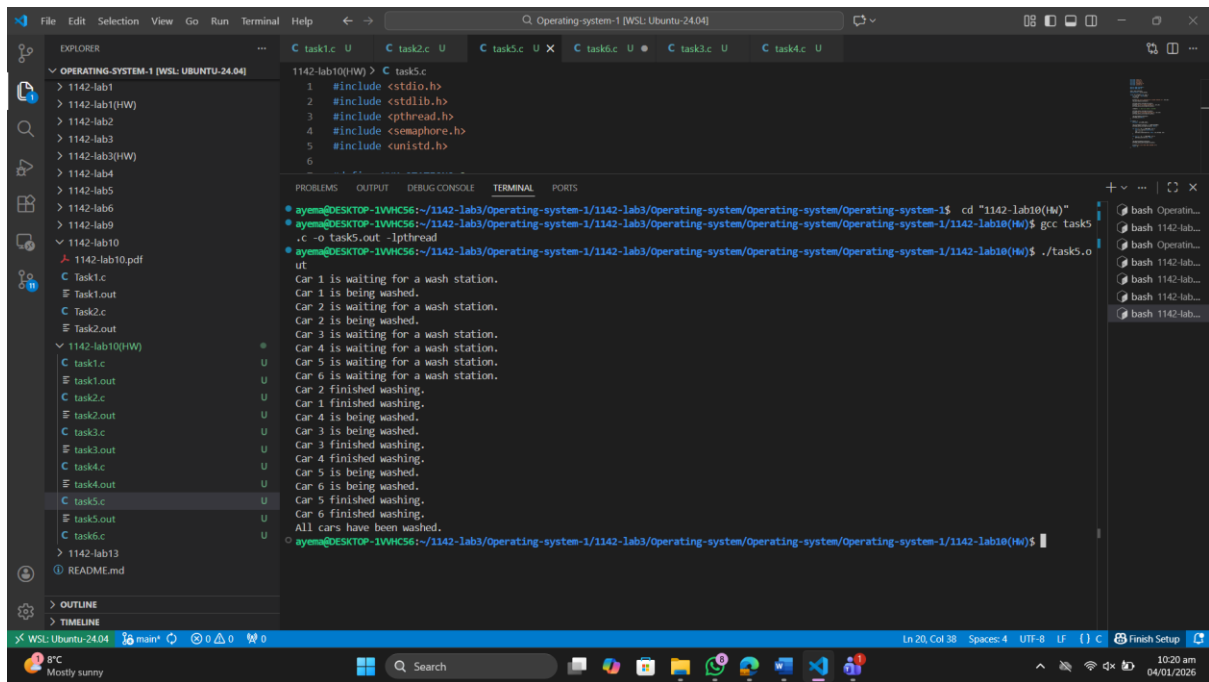
```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <pthread.h>
4  #include <semaphore.h>
5  #include <unistd.h>
6
7  #define NUM_WORKERS 3
8  #define NUM_TASKS 10
9
10 sem_t worker_sem;
11 pthread_mutex_t print_mutex;
12
13 void* task_thread(void* arg) {
14     int task_id = *(int*)arg;
15     free(arg);
16
17     sem_wait(&worker_sem); // Wait for an available worker
18
19     pthread_mutex_lock(&print_mutex);
20     printf("Task %d started by a worker.\n", task_id);
21     pthread_mutex_unlock(&print_mutex);
22
23     sleep(rand() % 2 + 1); // Simulate task running 1-2 seconds
24
25     pthread_mutex_lock(&print_mutex);
26     printf("Task %d finished.\n", task_id);
27     pthread_mutex_unlock(&print_mutex);
28
29     sem_post(&worker_sem); // Worker becomes free
30     pthread_exit(NULL);
31 }
32
33 int main() {
34     srand(time(NULL));
35     pthread_t tasks[NUM_TASKS];
36
37     sem_init(&worker_sem, 0, NUM_WORKERS);
38     pthread_mutex_init(&print_mutex, NULL);
39
40     for (int i = 0; i < NUM_TASKS; i++) {
41         int* id = malloc(sizeof(int));
42         *id = i + 1;
43         pthread_create(&tasks[i], NULL, task_thread, id);
44     }
45
46     for (int i = 0; i < NUM_TASKS; i++) {
47         pthread_join(tasks[i], NULL);
48     }
49
50     sem_destroy(&worker_sem);
51     pthread_mutex_destroy(&print_mutex);
52
53     printf("All tasks completed.\n");
54     return 0;
55 }
56
```



## Task 5: Car Wash Station



```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <pthread.h>
4  #include <semaphore.h>
5  #include <unistd.h>
6
7  #define NUM_STATIONS 2
8  #define NUM_CARS 6
9
10 sem_t wash_stations;
11 pthread_mutex_t print_mutex;
12
13 void* car_thread(void* arg) {
14     int car_id = *(int*)arg;
15     free(arg);
16
17     printf("Car %d is waiting for a wash station.\n", car_id);
18     sem_wait(&wash_stations);
19
20     pthread_mutex_lock(&print_mutex);
21     printf("Car %d is being washed.\n", car_id);
22     pthread_mutex_unlock(&print_mutex);
23
24     sleep(3); // Each car takes 3 seconds
25
26     pthread_mutex_lock(&print_mutex);
27     printf("Car %d finished washing.\n", car_id);
28     pthread_mutex_unlock(&print_mutex);
29
30     sem_post(&wash_stations);
31     pthread_exit(NULL);
32 }
33
34 int main() {
35     pthread_t cars[NUM_CARS];
36
37     sem_init(&wash_stations, 0, NUM_STATIONS);
38     pthread_mutex_init(&print_mutex, NULL);
39
40     for (int i = 0; i < NUM_CARS; i++) {
41         int* id = malloc(sizeof(int));
42         *id = i + 1;
43         pthread_create(&cars[i], NULL, car_thread, id);
44     }
45
46     for (int i = 0; i < NUM_CARS; i++) {
47         pthread_join(cars[i], NULL);
48     }
49
50     sem_destroy(&wash_stations);
51     pthread_mutex_destroy(&print_mutex);
52
53     printf("All cars have been washed.\n");
54     return 0;
55 }
56
```



## Task 6: Traffic Bridge Control (Single-Lane Bridge)



```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <pthread.h>
4  #include <semaphore.h>
5  #include <unistd.h>
6
7  #define MAX_CARS_ON_BRIDGE 3
8  #define TOTAL_CARS 8
9
10 sem_t bridge_sem;
11 pthread_mutex_t print_mutex;
12
13 void* car_thread(void* arg) {
14     int car_id = *(int*)arg;
15     free(arg);
16
17     printf("Car %d is waiting to enter the bridge.\n", car_id);
18     sem_wait(&bridge_sem);
19
20     pthread_mutex_lock(&print_mutex);
21     printf("Car %d is on the bridge.\n", car_id);
22     pthread_mutex_unlock(&print_mutex);
23
24     sleep(rand() % 3 + 1); // Random crossing time 1-3 sec
25
26     pthread_mutex_lock(&print_mutex);
27     printf("Car %d has left the bridge.\n", car_id);
28     pthread_mutex_unlock(&print_mutex);
29
30     sem_post(&bridge_sem);
31     pthread_exit(NULL);
32 }
33
34 int main() {
35     srand(time(NULL));
36     pthread_t cars[TOTAL_CARS];
37
38     sem_init(&bridge_sem, 0, MAX_CARS_ON_BRIDGE);
39     pthread_mutex_init(&print_mutex, NULL);
40
41     for (int i = 0; i < TOTAL_CARS; i++) {
42         int* id = malloc(sizeof(int));
43         *id = i + 1;
44         pthread_create(&cars[i], NULL, car_thread, id);
45     }
46
47     for (int i = 0; i < TOTAL_CARS; i++) {
48         pthread_join(cars[i], NULL);
49     }
50
51     sem_destroy(&bridge_sem);
52     pthread_mutex_destroy(&print_mutex);
53
54     printf("All cars have crossed the bridge.\n");
55     return 0;
56 }
57
```

