

## TASK 1:



```
1 #include <stdio.h>
2 #include <pthread.h>
3 #include <semaphore.h>
4 #include <unistd.h>
5 #define BUFFER_SIZE 5
6 int buffer[BUFFER_SIZE];
7 int in = 0; // Producer index
8 int out = 0; // Consumer index
9 sem_t empty; // Counts empty slots
10 sem_t full; // Counts full slots
11 pthread_mutex_t mutex;
12 void* producer(void* arg) {
13     int id = *(int*)arg;
14     for(int i = 0; i < 3; i++) { // Each producer makes 3 items
15         int item = id * 100 + i;
16         // TODO: Wait for empty slot
17         //sem_wait(&empty);
18         // TODO: Lock the buffer
19         pthread_mutex_lock(&mutex);
20         // Add item to buffer
21         buffer[in] = item;
22         printf("Producer %d produced item %d at position %d\n",
23             id, item, in);
24         in = (in + 1) % BUFFER_SIZE;
25         // TODO: Unlock the buffer
26         pthread_mutex_unlock(&mutex);
27         // TODO: Signal that buffer has a full slot
28         sem_post(&full);
29         sleep(1);
30     }
31     return NULL;
32 }
33 void* consumer(void* arg) {
34     int id = *(int*)arg;
35     for(int i = 0; i < 3; i++) {
36         // TODO: Students complete this similar to producer
37         sem_wait(&full);
38         pthread_mutex_lock(&mutex);
39         int item = buffer[out];
40         printf("Consumer %d consumed item %d from position %d\n",
41             id, item, out);
42         out = (out + 1) % BUFFER_SIZE;
43         pthread_mutex_unlock(&mutex);
44         sem_post(&empty);
45         sleep(2); // Consumers are slower
46     }
47     return NULL;
48 }
49 int main() {
50     pthread_t prod[2], cons[2];
51     int ids[2] = {1, 2};
52     // Initialize semaphores
53     sem_init(&empty, 0, BUFFER_SIZE); // All slots empty initially
54     sem_init(&full, 0, 0);
55     pthread_mutex_init(&mutex, NULL);
56     // No slots full initially
57     // Create producers and consumers
58     for(int i = 0; i < 2; i++) {
59         pthread_create(&prod[i], NULL, producer, &ids[i]);
60         pthread_create(&cons[i], NULL, consumer, &ids[i]);
61     }
62     // Wait for completion
63     for(int i = 0; i < 2; i++) {
64         pthread_join(prod[i], NULL);
65         pthread_join(cons[i], NULL);
66     }
67     // Cleanup
68     sem_destroy(&empty);
69     sem_destroy(&full);
70     pthread_mutex_destroy(&mutex);
71     return 0;
72 }
```

The screenshot shows a Windows desktop environment with a Visual Studio Code (VS Code) window open. The title bar of the VS Code window reads "Operating-system-1 [WSL: Ubuntu-24.04]". The code editor displays a C program named Task2.c, which simulates a parking garage with 10 slots. The program uses threads and mutexes to manage parking requests. The terminal tab shows the output of the program, indicating successful parking and leaving for each of the 10 cars. The Explorer sidebar shows other files in the workspace, including Task1.c, Task1.out, Task2.c, Task2.out, and README.md. The status bar at the bottom shows the weather as "23°C Sunny".

```
1142-lab10 > C Task1.c C Task2.c U ●
OPERATING-SYSTEM-1 [W...
> 1142-assignment1
> 1142-evaluation
> 1142-lab1
> 1142-lab1(HW)
> 1142-lab2
> 1142-lab3
> 1142-lab3(HW)
> 1142-lab4
> 1142-lab5
> 1142-lab6
> 1142-lab9
> 1142-lab10
C Task1.c U
E Task1.out U
C Task2.c U
E Task2.out U
① README.md

ayem@DESKTOP-1VWHCS6:~/1142-lab3/Operating-system-1/1142-lab3/operating-system/Operating-system/Operating-system-1/1142-lab10$ ./Task1.out
Car 1 is trying to park...
Car 1 parked successfully!
Car 2 is trying to park...
Car 2 parked successfully!
Car 3 is trying to park...
Car 3 parked successfully!
Car 4 is trying to park...
Car 4 parked successfully!
Car 5 is trying to park...
Car 5 parked successfully!
Car 6 is trying to park...
Car 6 parked successfully!
Car 7 is trying to park...
Car 7 parked successfully!
Car 8 is trying to park...
Car 8 parked successfully!
Car 9 is trying to park...
Car 9 parked successfully!
Car 10 is trying to park...
Car 10 parked successfully!
Car 3 is leaving.
Car 1 is leaving.
Car 2 is leaving.
Car 6 parked successfully!
Car 4 parked successfully!
Car 5 parked successfully!
Car 4 is leaving.
Car 5 is leaving.
Car 6 is leaving.
Car 7 parked successfully!
Car 9 parked successfully!
Car 8 parked successfully!
Car 7 is leaving.
Car 9 is leaving.
Car 10 parked successfully!
Car 8 is leaving.
Car 10 is leaving.
ayem@DESKTOP-1VWHCS6:~/1142-lab3/Operating-system-1/1142-lab3/operating-system/Operating-system/Operating-system-1/1142-lab10$
```

## Task 2:

```
1  #include <stdio.h>
2  #include <pthread.h>
3  #include <semaphore.h>
4  #include <unistd.h>
5  #define BUFFER_SIZE 5
6  int buffer[BUFFER_SIZE];
7  int in = 0; // Producer index
8  int out = 0; // Consumer index
9  sem_t empty; // Counts empty slots
10 sem_t full; // Counts full slots
11 pthread_mutex_t mutex;
12 void* producer(void* arg) {
13     int id = *(int*)arg;
14     for(int i = 0; i < 3; i++) { // Each producer makes 3 items
15         int item = id * 100 + i;
16         // TODO: Wait for empty slot
17         //sem_wait(&empty);
18         // TODO: Lock the buffer
19         pthread_mutex_lock(&mutex);
20         // Add item to buffer
21         buffer[in] = item;
22         printf("Producer %d produced item %d at position %d\n",
23             id, item, in);
24         in = (in + 1) % BUFFER_SIZE;
25         // TODO: Unlock the buffer
26         pthread_mutex_unlock(&mutex);
27         // TODO: Signal that buffer has a full slot
28         sem_post(&full);
29         sleep(1);
30     }
31     return NULL;
32 }
33 void* consumer(void* arg) {
34     int id = *(int*)arg;
35     for(int i = 0; i < 3; i++) {
36         // TODO: Students complete this similar to producer
37         sem_wait(&full);
38         pthread_mutex_lock(&mutex);
39         int item = buffer[out];
40         printf("Consumer %d consumed item %d from position %d\n",
41             id, item, out);
42         out = (out + 1) % BUFFER_SIZE;
43         pthread_mutex_unlock(&mutex);
44         sem_post(&empty);
45         sleep(2); // Consumers are slower
46     }
47     return NULL;
48 }
49 int main() {
50     pthread_t prod[2], cons[2];
51     int ids[2] = {1, 2};
52     // Initialize semaphores
53     sem_init(&empty, 0, BUFFER_SIZE); // All slots empty initially
54     sem_init(&full, 0, 0);
55     pthread_mutex_init(&mutex, NULL);
56     // No slots full initially
57     // Create producers and consumers
58     for(int i = 0; i < 2; i++) {
59         pthread_create(&prod[i], NULL, producer, &ids[i]);
60         pthread_create(&cons[i], NULL, consumer, &ids[i]);
61     }
62     // Wait for completion
63     for(int i = 0; i < 2; i++) {
64         pthread_join(prod[i], NULL);
65         pthread_join(cons[i], NULL);
66     }
67     // Cleanup
68     sem_destroy(&empty);
69     sem_destroy(&full);
70     pthread_mutex_destroy(&mutex);
71     return 0;
72 }
```

```

OPERATING-SYSTEM-1 [W...]
  > 1142-assignment1
  > 1142-evaluation
  > 1142-lab1
  > 1142-lab1(HW)
  > 1142-lab2
  > 1142-lab3
  > 1142-lab4
  > 1142-lab5
  > 1142-lab6
  > 1142-lab9
  > 1142-lab10
    C Task1.c
    E Task1.out
    C Task2.c
    E Task2.out
  README.md

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

ayem@DESKTOP-IVNC56:~/1142-lab3/Operating-system-1/1142-lab3/Operating-system/Operating-system-1$ cd 1142-lab10
ayem@DESKTOP-IVNC56:~/1142-lab3/Operating-system-1/1142-lab3/Operating-system/Operating-system-1/1142-lab10$ gcc Task2.c -o Task2.out -lpthread
ayem@DESKTOP-IVNC56:~/1142-lab3/Operating-system-1/1142-lab3/Operating-system/Operating-system-1/1142-lab10$ ./Task2.out
Producer 1 produced item 100 at position 0
Producer 2 produced item 200 at position 1
Consumer 1 consumed item 100 from position 0
Consumer 2 consumed item 200 from position 1
Producer 4 produced item 101 at position 2
Producer 2 produced item 201 at position 3
Consumer 2 consumed item 101 from position 2
Consumer 1 consumed item 201 from position 3
Producer 1 produced item 102 at position 4
Producer 2 produced item 202 at position 0
Consumer 2 consumed item 102 from position 4
Consumer 1 consumed item 202 from position 0
ayem@DESKTOP-IVNC56:~/1142-lab3/Operating-system-1/1142-lab3/Operating-system/Operating-system-1/1142-lab10$ gcc Task2.c -o Task2.out -lpthread
ayem@DESKTOP-IVNC56:~/1142-lab3/Operating-system-1/1142-lab3/Operating-system/Operating-system-1/1142-lab10$ ./Task2.out
Producer 1 produced item 100 at position 0
Producer 2 produced item 200 at position 1
Consumer 1 consumed item 100 from position 0
Consumer 2 consumed item 200 from position 1
Producer 1 produced item 102 at position 2
Producer 2 produced item 202 at position 0
Consumer 1 consumed item 102 from position 3
Consumer 2 consumed item 202 from position 0
ayem@DESKTOP-IVNC56:~/1142-lab3/Operating-system-1/1142-lab3/Operating-system/Operating-system-1/1142-lab10$ 

```

WSL: Ubuntu-24.04

23°C Sunny

Ln 1, Col 4 Spaces: 4 UTF-8 LF ⚡ C Finish Setup

3:07 pm 28/11/2025

## Demonstrations:

**As 3 items in code so total no is 6.**

**Total thread is 3**

**Mutex is 1.**

**Semaphore 2**

**If Item 4 items it produce total 8. Buffer size 5 Hain toh first 5 produce ho gai baki 3 blocked ho jai gai. Fir baad meh consumer consume karay ga aur fir Producer produce karkai sub consume or produce ho gai.**