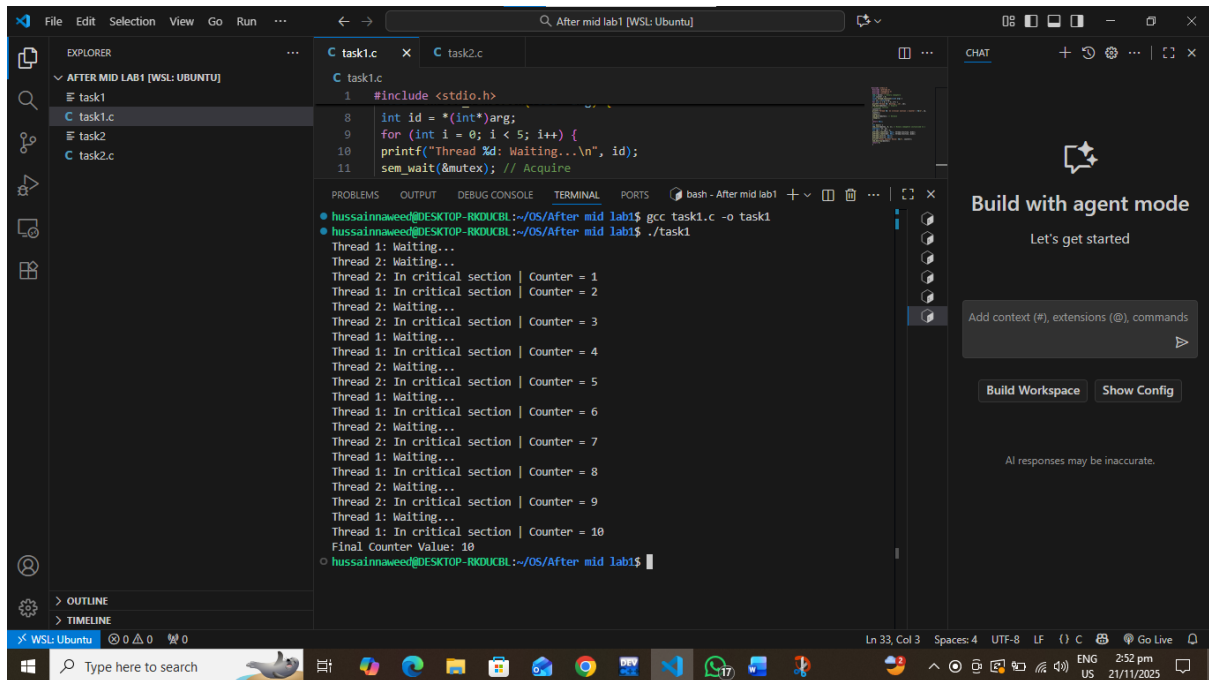


Task1

Code

```
1  #include <stdio.h>
2  #include <pthread.h>
3  #include <semaphore.h>
4  #include <unistd.h>
5  sem_t mutex; // Binary semaphore
6  int counter = 0;
7  void* thread_function(void* arg) {
8  int id = *(int*)arg;
9  for (int i = 0; i < 5; i++) {
10 printf("Thread %d: Waiting...\n", id);
11 sem_wait(&mutex); // Acquire
12 // Critical section
13 counter++;
14 printf("Thread %d: In critical section | Counter = %d\n", id,
15 counter);
16 sleep(1);
17 sem_post(&mutex); // Release
18 sleep(1);
19 }
20 return NULL;
21 }
22 int main() {
23 sem_init(&mutex, 0, 1); // Binary semaphore initialized to 1
24 pthread_t t1, t2;
25 int id1 = 1, id2 = 2;
26 pthread_create(&t1, NULL, thread_function, &id1);
27 pthread_create(&t2, NULL, thread_function, &id2);
28 pthread_join(t1, NULL);
29 pthread_join(t2, NULL);
30 printf("Final Counter Value: %d\n", counter);
31 sem_destroy(&mutex);
32 return 0;
33 }
```

Output



The screenshot shows the Visual Studio Code interface with a C program being executed. The Explorer panel on the left shows the file structure with 'task1.c' and 'task2.c'. The main editor displays the code for 'task1.c', which includes a semaphore and a loop that prints thread IDs. The Output panel at the bottom shows the execution results, including the compilation command, the execution command, and the output of the program. The output shows two threads, Thread 1 and Thread 2, alternating between waiting and being in a critical section, with a counter increasing from 1 to 10. The final output is 'Final Counter Value: 10'.

```
1 #include <stdio.h>
2
3 int main() {
4     sem_t mutex;
5     sem_init(&mutex, 0, 1);
6     int id = *(int*)arg;
7     for (int i = 0; i < 5; i++) {
8         printf("Thread %d: Waiting...\n", id);
9         sem_wait(&mutex); // Acquire
10    }
```

hussainnaweed@DESKTOP-RK0UCBL:~/OS/After mid lab1\$ gcc task1.c -o task1

hussainnaweed@DESKTOP-RK0UCBL:~/OS/After mid lab1\$./task1

Thread 1: Waiting...

Thread 2: Waiting...

Thread 2: In critical section | Counter = 1

Thread 1: In critical section | Counter = 2

Thread 2: Waiting...

Thread 2: In critical section | Counter = 3

Thread 1: Waiting...

Thread 1: In critical section | Counter = 4

Thread 2: Waiting...

Thread 2: In critical section | Counter = 5

Thread 1: Waiting...

Thread 1: In critical section | Counter = 6

Thread 2: Waiting...

Thread 2: In critical section | Counter = 7

Thread 1: Waiting...

Thread 1: In critical section | Counter = 8

Thread 2: Waiting...

Thread 2: In critical section | Counter = 9

Thread 1: Waiting...

Thread 1: In critical section | Counter = 10

Final Counter Value: 10

hussainnaweed@DESKTOP-RK0UCBL:~/OS/After mid lab1\$

Task2



```
1  #include <stdio.h>
2  #include <pthread.h>
3  #include <semaphore.h>
4  #include <unistd.h>
5  sem_t mutex; // Binary semaphore
6  int counter = 0;
7  void* thread_function(void* arg) {
8  int id = *(int*)arg;
9  for (int i = 0; i < 5; i++) {
10 printf("Thread %d: Waiting...\n", id);
11 //sem_wait(&mutex); // Acquire
12 Critical section
13 counter++;
14 printf("Thread %d: In critical section | Counter = %d\n", id,
15 counter);
16 sleep(1);
17 sem_post(&mutex); // Release
18 sleep(1);
19 }
20 return NULL;
21 }
22 int main() {
23 sem_init(&mutex, 0, 1 ); // Binary semaphore initialized to 1
24 pthread_t t1, t2;
25 int id1 = 1, id2 = 2;
26 pthread_create(&t1, NULL, thread_function, &id1);
27 pthread_create(&t2, NULL, thread_function, &id2);
28 pthread_join(t1, NULL);
29 pthread_join(t2, NULL);
30 printf("Final Counter Value: %d\n", counter);
31 sem_destroy(&mutex);
32 return 0;
33 }
```

Output

```
● hussainnaweed@DESKTOP-RKDUCBL:~/OS/After mid lab1$ gcc task1.c -o task1
⊗ hussainnaweed@DESKTOP-RKDUCBL:~/OS/After mid lab1$ ./task1
Thread 1: Waiting...
Thread 2: Waiting...
^Z
[1]+  Stopped                  ./task1
○ hussainnaweed@DESKTOP-RKDUCBL:~/OS/After mid lab1$
```

Task3

```

1  #include <stdio.h>
2  #include <pthread.h>
3  #include <semaphore.h>
4  #include <unistd.h>
5  sem_t mutex; // Binary semaphore
6  int counter = 0;
7  void* thread_function(void* arg) {
8  int id = *(int*)arg;
9  for (int i = 0; i < 5; i++) {
10 printf("Thread %d: Waiting...\n", id);
11 sem_wait(&mutex); // Acquire
12 // Critical section
13 counter++;
14 printf("Thread %d: In critical section | Counter = %d\n", id,
15 counter);
16 sleep(1);
17 //sem_post(&mutex); // Release
18 sleep(1);
19 }
20 return NULL;
21 }
22 int main() {
23 sem_init(&mutex, 1, 0); // Binary semaphore initialized to 1
24 pthread_t t1, t2;
25 int id1 = 1, id2 = 2;
26 pthread_create(&t1, NULL, thread_function, &id1);
27 pthread_create(&t2, NULL, thread_function, &id2);
28 pthread_join(t1, NULL);
29 pthread_join(t2, NULL);
30 printf("Final Counter Value: %d\n", counter);
31 sem_destroy(&mutex);
32 return 0;
33 }

```

Output

```

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  bash - After mid lab1
● hussainnaweed@DESKTOP-RKDUCBL:~/OS/After mid lab1$ gcc task1.c -o task1
● hussainnaweed@DESKTOP-RKDUCBL:~/OS/After mid lab1$ ./task1
Thread 1: Waiting...
Thread 2: Waiting...
^Z
[1]+  Stopped                  ./task1
● hussainnaweed@DESKTOP-RKDUCBL:~/OS/After mid lab1$

```

Task4

```
1  #include <stdio.h>
2  #include <pthread.h>
3  #include <semaphore.h>
4  #include <unistd.h>
5  sem_t mutex; // Binary semaphore
6  int counter = 0;
7  void* thread_function(void* arg) {
8  int id = *(int*)arg;
9  for (int i = 0; i < 500; i++) {
10 printf("Thread %d: Waiting...\n", id);
11 //sem_wait(&mutex); // Acquire
12 // Critical section
13 counter++;
14 printf("Thread %d: In critical section | Counter = %d\n", id,
15 counter);
16 sleep(1);
17 sem_post(&mutex); // Release
18 sleep(1);
19 }
20 return NULL;
21 }
22 int main() {
23 sem_init(&mutex, 1, 0 ); // Binary semaphore initialized to 1
24 pthread_t t1, t2;
25 int id1 = 1, id2 = 2;
26 pthread_create(&t1, NULL, thread_function, &id1);
27 pthread_create(&t2, NULL, thread_function, &id2);
28 pthread_join(t1, NULL);
29 pthread_join(t2, NULL);
30 printf("Final Counter Value: %d\n", counter);
31 sem_destroy(&mutex);
32 return 0;
33 }
```

Output

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  + v ... | [
hussainnaweed@DESKTOP-RKDUCBL:~/OS/After mid lab1$ gcc task1.c -o task1
hussainnaweed@DESKTOP-RKDUCBL:~/OS/After mid lab1$ ./task1
Thread 1: Waiting...
Thread 1: In critical section | Counter = 1
Thread 2: Waiting...
Thread 2: In critical section | Counter = 2
Thread 1: Waiting...
Thread 1: In critical section | Counter = 3
Thread 2: Waiting...
Thread 2: In critical section | Counter = 4
Thread 2: Waiting...
Thread 2: In critical section | Counter = 5
Thread 1: Waiting...
Thread 1: In critical section | Counter = 6
Thread 1: Waiting...
Thread 1: In critical section | Counter = 7
Thread 2: Waiting...
Thread 2: In critical section | Counter = 8
Thread 2: Waiting...
Thread 2: In critical section | Counter = 9
Thread 1: Waiting...
Thread 1: In critical section | Counter = 10
Thread 1: Waiting...
Thread 1: In critical section | Counter = 11
Thread 2: Waiting...
Thread 2: In critical section | Counter = 12
Thread 1: Waiting...
Thread 1: In critical section | Counter = 13
Thread 2: Waiting...
Thread 2: In critical section | Counter = 14
Thread 1: Waiting...
Thread 1: In critical section | Counter = 15
Thread 2: Waiting...
Thread 2: In critical section | Counter = 16
^Z
[1]+  Stopped                  ./task1
hussainnaweed@DESKTOP-RKDUCBL:~/OS/After mid lab1$
```




```
1  #include <stdio.h>
2  #include <pthread.h>
3  #include <semaphore.h>
4  #include <unistd.h>
5  sem_t mutex; // Binary semaphore
6  int counter = 0;
7  void* thread_function(void* arg) {
8  int id = *(int*)arg;
9  for (int i = 0; i < 5; i++) {
10 printf("Thread %d: Waiting...\n", id);
11 //sem_wait(&mutex); // Acquire
12 Critical section
13 counter++;
14 printf("Thread %d: In critical section | Counter = %d\n", id,
15 counter);
16 sleep(1);
17 sem_post(&mutex); // Release
18 sleep(1);
19 }
20 return NULL;
21 }|
22 void* thread_function1(void* arg) {
23 int id = *(int*)arg;
24 for (int i = 0; i < 5; i++) {
25 printf("Thread %d: Waiting...\n", id);
26 //sem_wait(&mutex); // Acquire
27 Critical section
28 counter--;
29 printf("Thread %d: In critical section | Counter = %d\n", id,
30 counter);
31 sleep(1);
32 sem_post(&mutex); // Release
33 sleep(1);
34 }
35 return NULL;
36 }
37 int main() {
38 sem_init(&mutex, 1, 0 ); // Binary semaphore initialized to 1
39 pthread_t t1, t2;
40 int id1 = 1, id2 = 2;
41 pthread_create(&t1, NULL, thread_function, &id1);
42 pthread_create(&t2, NULL, thread_function1, &id2);
43 pthread_join(t1, NULL);
44 pthread_join(t2, NULL);
45 printf("Final Counter Value: %d\n", counter);
46 sem_destroy(&mutex);
47 return 0;
48 }
```

Task6

Mutex	Semaphore
Only one thread can access the resource	Multiple threads can access depending on the count
Works like a lock	Works like a counter/signal
Has only two states : locked/unlocked	Has a value that increases or decreases
Used for mutual exclusion	Used for controlling access to limited resources
Only the owner can release it	Anyone can signal (increase) it