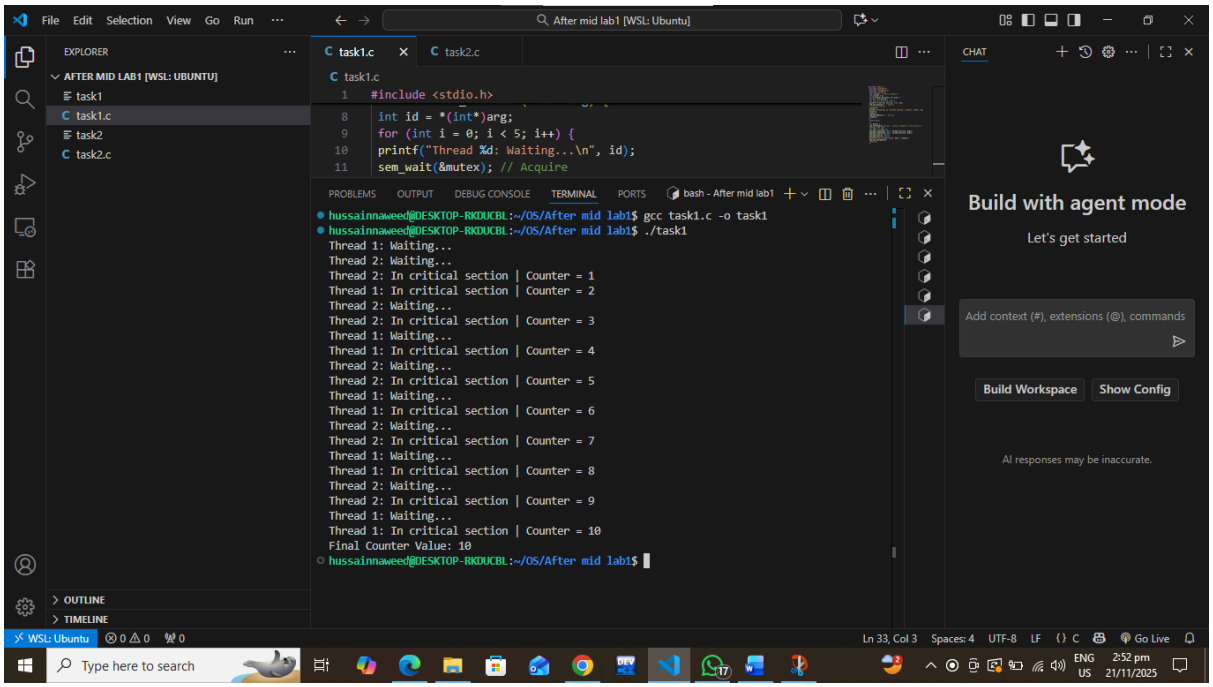
Task1

Code

```c
#include <stdio.h>
 #include <pthread.h>
 #include <semaphore.h>
 #include <unistd.h>
 sem_t mutex; // Binary semaphore
 int counter = 0;
 void* thread_function(void* arg) {
 int id = *(int*)arg;
 for (int i = 0; i < 5; i++) {
 printf("Thread %d: Waiting...\n", id);
 sem_wait(&mutex); // Acquire
// Critical section
 counter++;
 printf("Thread %d: In critical section | Counter = %d\n", id,
 counter);
 sleep(1);
 sem_post(&mutex); // Release
 sleep(1);
 }
 return NULL;
 }
 int main() {
 sem_init(&mutex, 0, 1); // Binary semaphore initialized to 1
 pthread_t t1, t2;
 int id1 = 1, id2 = 2;
 pthread_create(&t1, NULL, thread_function, &id1);
 pthread_create(&t2, NULL, thread_function, &id2);
 pthread_join(t1, NULL);
 pthread_join(t2, NULL);
 printf("Final Counter Value: %d\n", counter);
 sem_destroy(&mutex);
 return 0;
 }
```

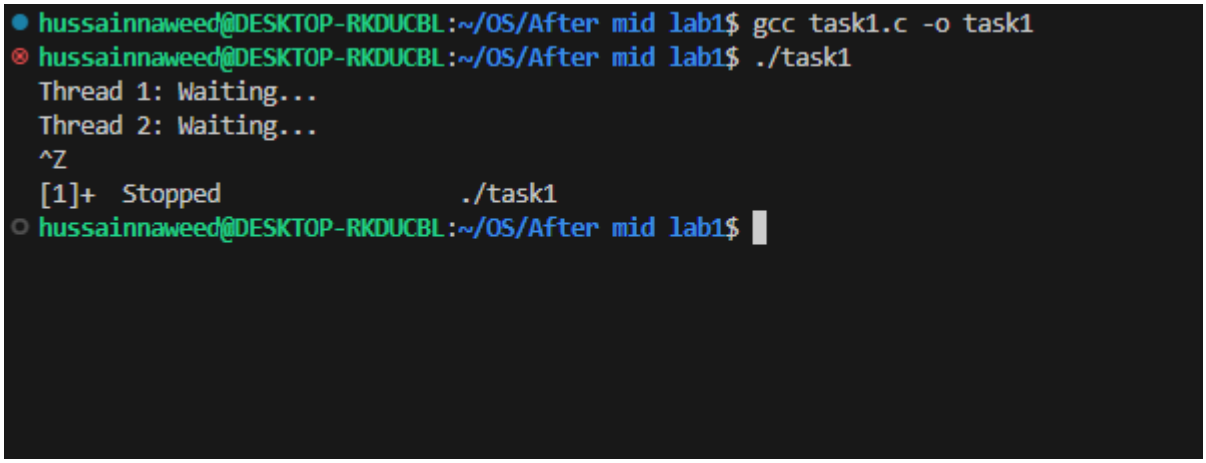
Output



Task2

```c
#include <stdio.h>
 #include <pthread.h>
 #include <semaphore.h>
 #include <unistd.h>
 sem_t mutex; // Binary semaphore
 int counter = 0;
 void* thread_function(void* arg) {
 int id = *(int*)arg;
 for (int i = 0; i < 5; i++) {
 printf("Thread %d: Waiting...\n", id);
 //sem_wait(&mutex); // Acquire
 Critical section
 counter++;
 printf("Thread %d: In critical section | Counter = %d\n", id,
 counter);
 sleep(1);
 sem_post(&mutex); // Release
 sleep(1);
 }
 return NULL;
 }
 int main() {
 sem_init(&mutex, 0,1 ); // Binary semaphore initialized to 1
 pthread_t t1, t2;
 int id1 = 1, id2 = 2;
 pthread_create(&t1, NULL, thread_function, &id1);
 pthread_create(&t2, NULL, thread_function, &id2);
 pthread_join(t1, NULL);
 pthread_join(t2, NULL);
 printf("Final Counter Value: %d\n", counter);
 sem_destroy(&mutex);
 return 0;
 }
```

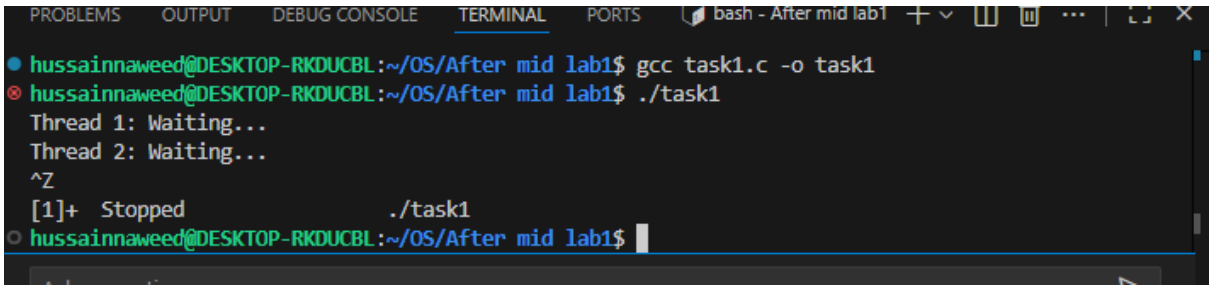
Output


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

```c
#include <stdio.h>
 #include <pthread.h>
 #include <semaphore.h>
 #include <unistd.h>
 sem_t mutex; // Binary semaphore
 int counter = 0;
 void* thread_function(void* arg) {
 int id = *(int*)arg;
 for (int i = 0; i < 5; i++) {
 printf("Thread %d: Waiting...\n", id);
 sem_wait(&mutex); // Acquire
// Critical section
 counter++;
 printf("Thread %d: In critical section | Counter = %d\n", id,
 counter);
 sleep(1);
 //sem_post(&mutex); // Release
 sleep(1);
 }
 return NULL;
 }
 int main() {
 sem_init(&mutex, 1,0 ); // Binary semaphore initialized to 1
 pthread_t t1, t2;
 int id1 = 1, id2 = 2;
 pthread_create(&t1, NULL, thread_function, &id1);
 pthread_create(&t2, NULL, thread_function, &id2);
 pthread_join(t1, NULL);
 pthread_join(t2, NULL);
 printf("Final Counter Value: %d\n", counter);
 sem_destroy(&mutex);
 return 0;
 }
```

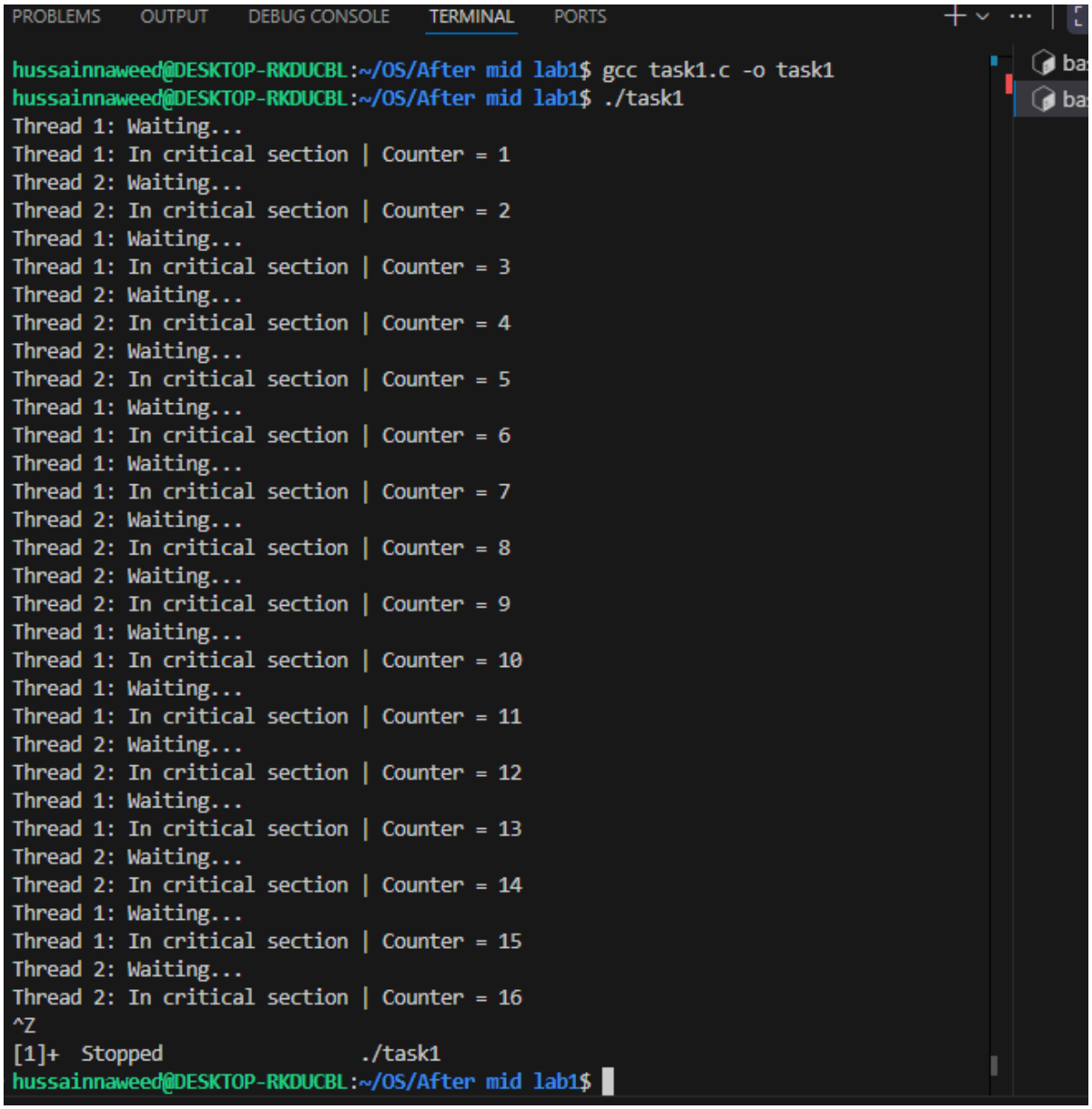
Output



Task4

```c
#include <stdio.h>
 #include <pthread.h>
 #include <semaphore.h>
 #include <unistd.h>
 sem_t mutex; // Binary semaphore
 int counter = 0;
 void* thread_function(void* arg) {
 int id = *(int*)arg;
 for (int i = 0; i < 500; i++) {
 printf("Thread %d: Waiting...\n", id);
 //sem_wait(&mutex); // Acquire
// Critical section
 counter++;
 printf("Thread %d: In critical section | Counter = %d\n", id,
 counter);
 sleep(1);
 sem_post(&mutex); // Release
 sleep(1);
 }
 return NULL;
 }
 int main() {
 sem_init(&mutex, 1,0 ); // Binary semaphore initialized to 1
 pthread_t t1, t2;
 int id1 = 1, id2 = 2;
 pthread_create(&t1, NULL, thread_function, &id1);
 pthread_create(&t2, NULL, thread_function, &id2);
 pthread_join(t1, NULL);
 pthread_join(t2, NULL);
 printf("Final Counter Value: %d\n", counter);
 sem_destroy(&mutex);
 return 0;
 }
```

Output


PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
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$

```c
#include <stdio.h>
 #include <pthread.h>
 #include <semaphore.h>
 #include <unistd.h>
 sem_t mutex; // Binary semaphore
 int counter = 0;
 void* thread_function(void* arg) {
 int id = *(int*)arg;
 for (int i = 0; i < 5; i++) {
 printf("Thread %d: Waiting...\n", id);
 //sem_wait(&mutex); // Acquire
 Critical section
 counter++;
 printf("Thread %d: In critical section | Counter = %d\n", id,
 counter);
 sleep(1);
 sem_post(&mutex); // Release
 sleep(1);
 }
 return NULL;
 }|
 void* thread_function1(void* arg) {
 int id = *(int*)arg;
 for (int i = 0; i < 5; i++) {
 printf("Thread %d: Waiting...\n", id);
 //sem_wait(&mutex); // Acquire
 Critical section
 counter--;
 printf("Thread %d: In critical section | Counter = %d\n", id,
 counter);
 sleep(1);
 sem_post(&mutex); // Release
 sleep(1);
 }
 return NULL;
 }
 int main() {
 sem_init(&mutex, 1,0 ); // Binary semaphore initialized to 1
 pthread_t t1, t2;
 int id1 = 1, id2 = 2;
 pthread_create(&t1, NULL, thread_function, &id1);
 pthread_create(&t2, NULL, thread_function1, &id2);
 pthread_join(t1, NULL);
 pthread_join(t2, NULL);
 printf("Final Counter Value: %d\n", counter);
 sem_destroy(&mutex);
 return 0;
 }
```

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