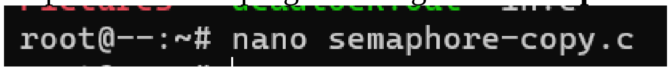
Ramzy Izza Wardhana
21/472698/PA/20322
IUP CS B

Assignment 9 – Computer System and Networking

Create a program that has two threads, threadA and threadB that are trying to access and modify two shared variables, varA and varB. In this case, both threads need to have access to both variables to finish their execution, similar to the deadlock case. Use the semaphore approach to ensure that deadlock does not happen. Take some screenshots and answer the following questions:

Answer:

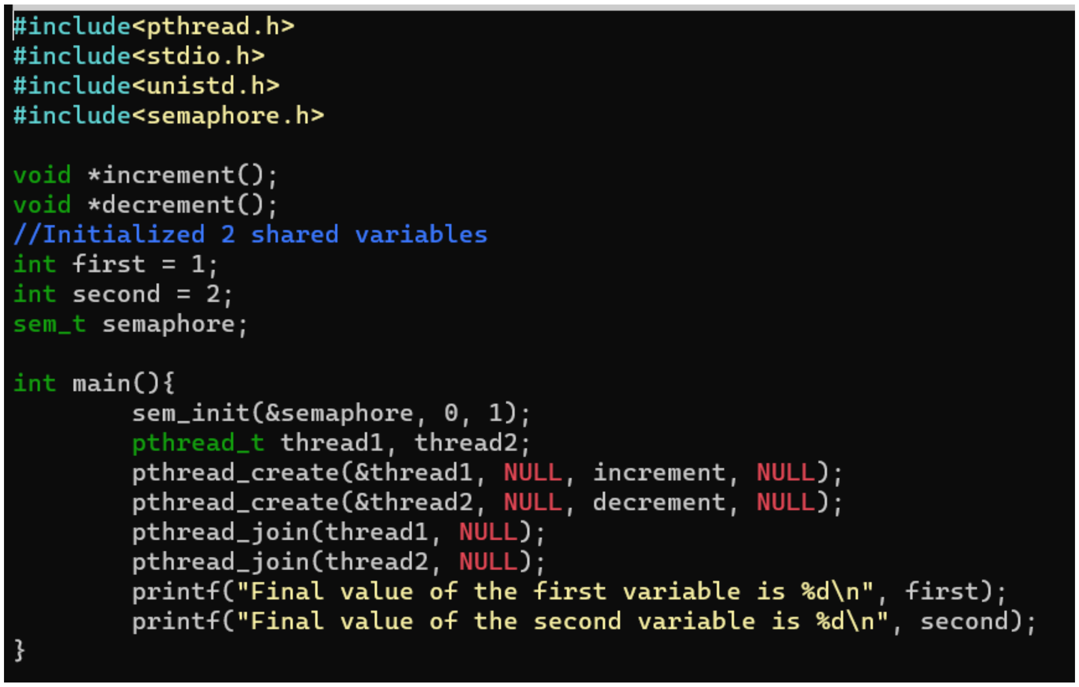
Step 1: Create the program using **nano semaphore-copy.c**



```
root@--:~# nano semaphore-copy.c
```

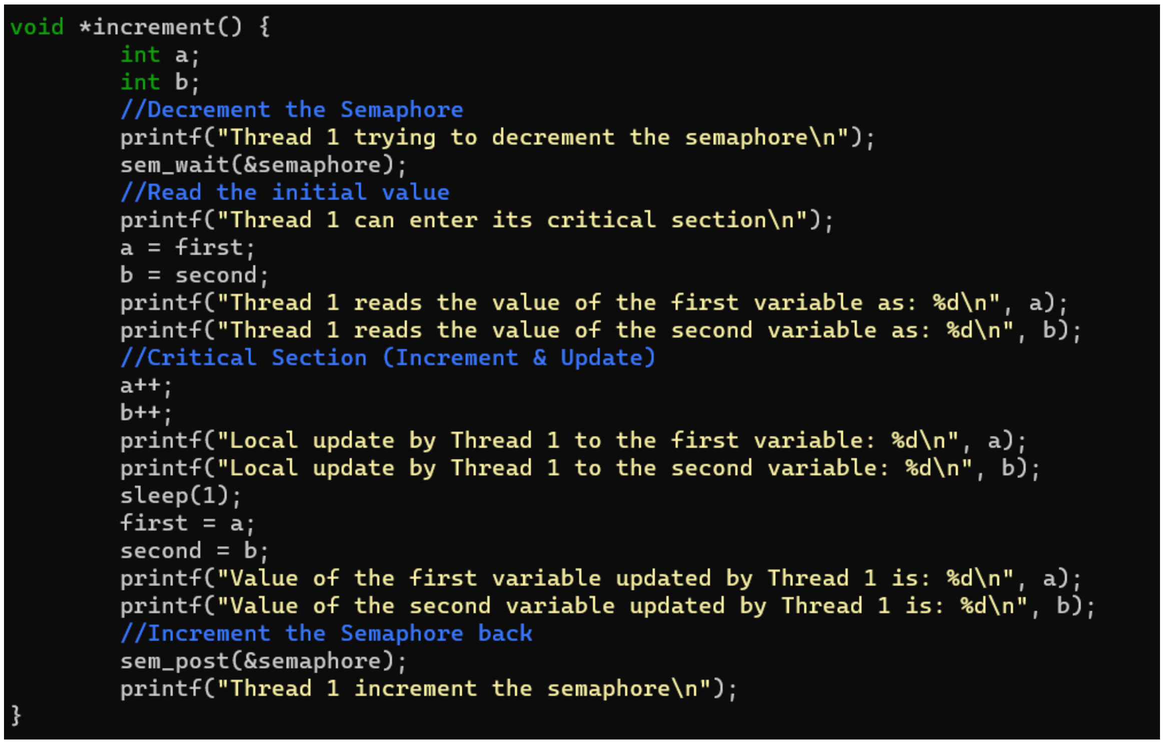
Step 2: Write the program

Main Function:



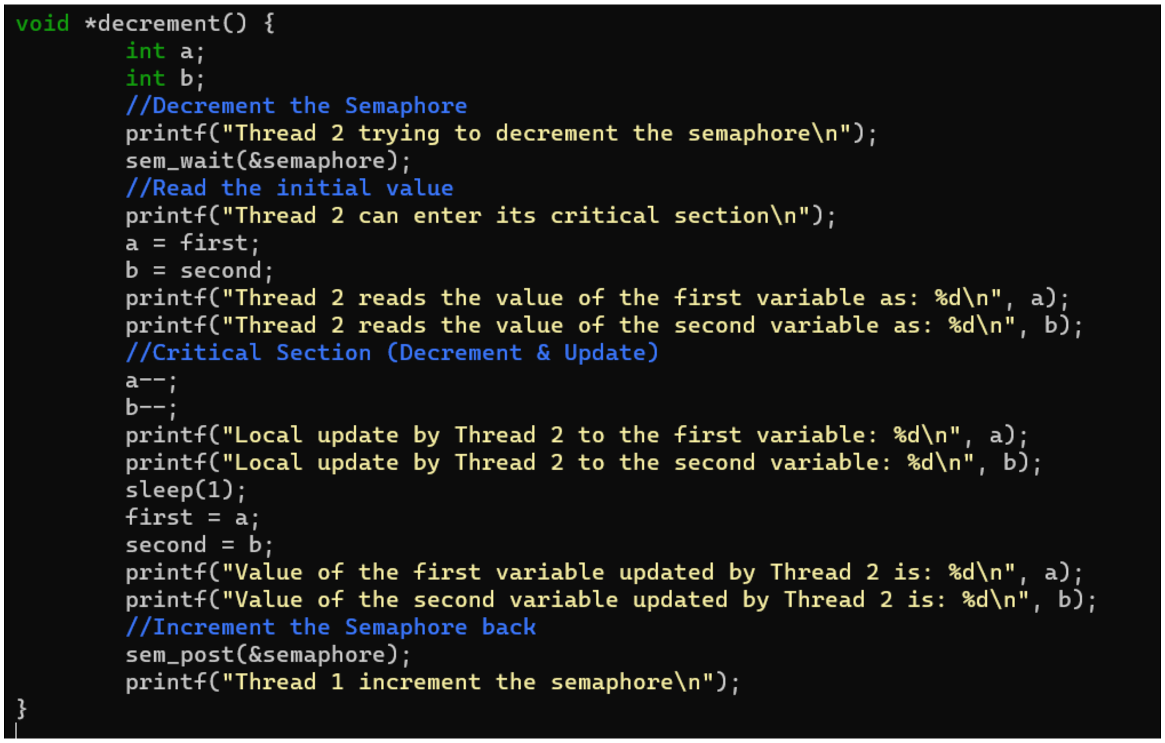
```c
#include<pthread.h>
#include<stdio.h>
#include<unistd.h>
#include<semaphore.h>

void *increment();
void *decrement();
//Initialized 2 shared variables
int first = 1;
int second = 2;
sem_t semaphore;

int main(){
        sem_init(&semaphore, 0, 1);
        pthread_t thread1, thread2;
        pthread_create(&thread1, NULL, increment, NULL);
        pthread_create(&thread2, NULL, decrement, NULL);
        pthread_join(thread1, NULL);
        pthread_join(thread2, NULL);
        printf("Final value of the first variable is %d\n", first);
        printf("Final value of the second variable is %d\n", second);
}
```

Increment Function:



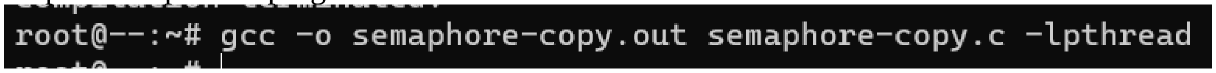
```
void *increment() {
        int a;
        int b;
        //Decrement the Semaphore
        printf("Thread 1 trying to decrement the semaphore\n");
        sem_wait(&semaphore);
        //Read the initial value
        printf("Thread 1 can enter its critical section\n");
        a = first;
        b = second;
        printf("Thread 1 reads the value of the first variable as: %d\n", a);
        printf("Thread 1 reads the value of the second variable as: %d\n", b);
        //Critical Section (Increment & Update)
        a++;
        b++;
        printf("Local update by Thread 1 to the first variable: %d\n", a);
        printf("Local update by Thread 1 to the second variable: %d\n", b);
        sleep(1);
        first = a;
        second = b;
        printf("Value of the first variable updated by Thread 1 is: %d\n", a);
        printf("Value of the second variable updated by Thread 1 is: %d\n", b);
        //Increment the Semaphore back
        sem_post(&semaphore);
        printf("Thread 1 increment the semaphore\n");
}
```

Decrement Function:



```
void *decrement() {
        int a;
        int b;
        //Decrement the Semaphore
        printf("Thread 2 trying to decrement the semaphore\n");
        sem_wait(&semaphore);
        //Read the initial value
        printf("Thread 2 can enter its critical section\n");
        a = first;
        b = second;
        printf("Thread 2 reads the value of the first variable as: %d\n", a);
        printf("Thread 2 reads the value of the second variable as: %d\n", b);
        //Critical Section (Decrement & Update)
        a--;
        b--;
        printf("Local update by Thread 2 to the first variable: %d\n", a);
        printf("Local update by Thread 2 to the second variable: %d\n", b);
        sleep(1);
        first = a;
        second = b;
        printf("Value of the first variable updated by Thread 2 is: %d\n", a);
        printf("Value of the second variable updated by Thread 2 is: %d\n", b);
        //Increment the Semaphore back
        sem_post(&semaphore);
        printf("Thread 1 increment the semaphore\n");
}
```

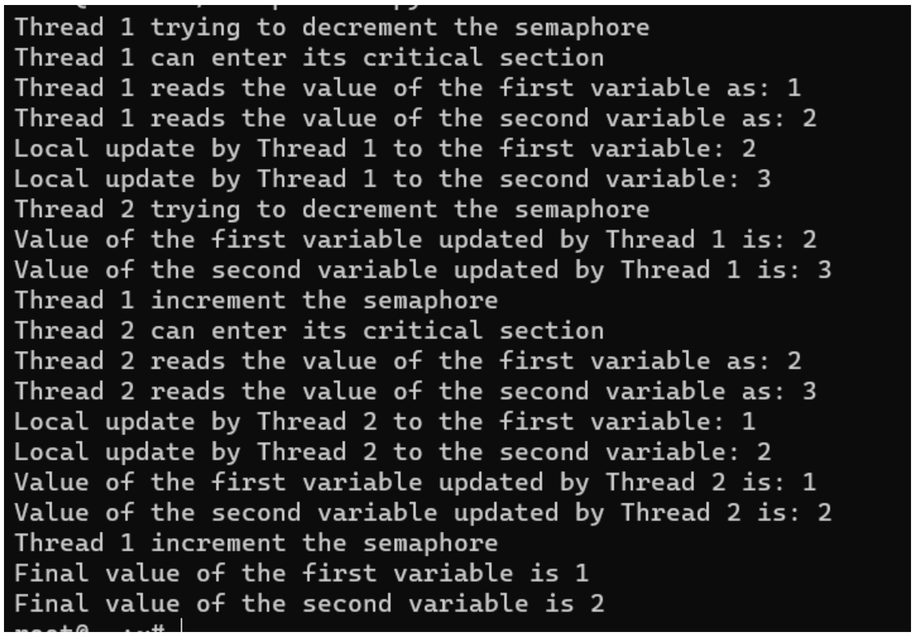
Step 3: Compile the program



```
root@--:~# gcc -o semaphore-copy.out semaphore-copy.c -lpthread
```

Step 4: Run the program

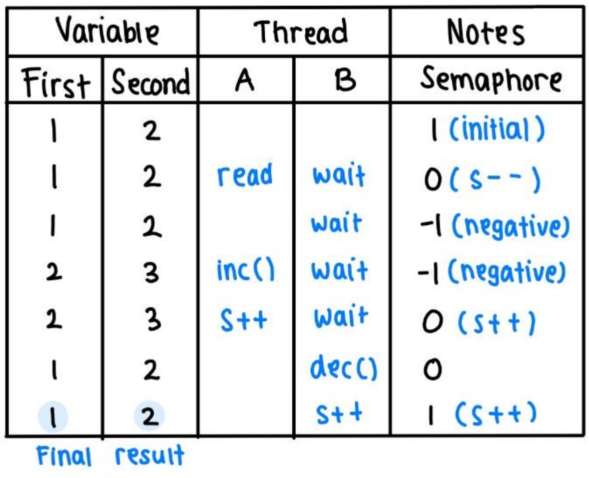
Result



```
Thread 1 trying to decrement the semaphore
Thread 1 can enter its critical section
Thread 1 reads the value of the first variable as: 1
Thread 1 reads the value of the second variable as: 2
Local update by Thread 1 to the first variable: 2
Local update by Thread 1 to the second variable: 3
Thread 2 trying to decrement the semaphore
Value of the first variable updated by Thread 1 is: 2
Value of the second variable updated by Thread 1 is: 3
Thread 1 increment the semaphore
Thread 2 can enter its critical section
Thread 2 reads the value of the first variable as: 2
Thread 2 reads the value of the second variable as: 3
Local update by Thread 2 to the first variable: 1
Local update by Thread 2 to the second variable: 2
Value of the first variable updated by Thread 2 is: 1
Value of the second variable updated by Thread 2 is: 2
Thread 1 increment the semaphore
Final value of the first variable is 1
Final value of the second variable is 2
```

1. **Which thread completed first?**
   Answer: From the result above, Thread 1 is the first thread that completed, followed by Thread 2 as the last. This happened to avoid deadlock in which Thread 1 will decrement semaphore and must complete all processing of first and second variable. When the process currently happening, Thread 2 must wait. Right after all processes finished on Thread 1, it will increment back the Semaphore and will satisfy the condition of > 0. In this case, Thread 2 will then proceed to its critical section right after Thread 1 and decrement the semaphore with similar concept above.

2. **Illustrate how each thread modifies the semaphore and continues/waiting for their turn.**



Variable		Thread		Notes
First	Second	A	B	Semaphore
1	2			1 (initial)
1	2	read	wait	0 (s--)
1	2		wait	-1 (negative)
2	3	inc()	wait	-1 (negative)
2	3	S++	wait	0 (s++)
1	2		dec()	0
1	2		s++	1 (s++)

Final result

From the illustration above, notice that the initial value of semaphore is set to 1. On the next phase, Thread will decrement it until non-negative (-1). At this state the process of incrementing and decrementing will be start in order (Thread 1 -> Thread 2) until the semaphore is incremented back to 1.