

## Assignment 2

**Study the example program in the attached file. This program creates a new thread. Both the main thread and the new thread then increment the variable counter infinitely. After incrementing the value of counter, each thread prints a message showing the value of the variable. One of the threads exits when it finds that the counter value has exceeded 25. Run the program and explain the output. Why do the print statements stop appearing after a certain point in the program? Explain.**

`pthread_mutex_lock ()` and `pthread_mutex_unlock ()` are normally used to protect data structures. That is, you make sure that only one thread at a time can access a certain data structure by locking and unlocking it.

**Working of Mutexes:** If main thread tries to lock the mutex while child thread has the same mutex locked, main thread goes to sleep. As soon as child thread releases the mutex (via a `pthread_mutex_unlock ()` call), main thread will be able to lock the mutex.

-We cannot predict them to take the lock alternately because child thread can unlock and again lock before even main thread tries to lock the mutex.

### **Explanation of Program :**

Let us assume that main thread has the lock first

--sleeps for a sec

--increases the counter value

--unlocks the mutex.

--Prints the value of the counter as "Main counter is \_\_\_\_".

In this mean time, if child thread tries to take the lock, it will go to sleep.

--tries to take the lock again.

If it succeeds it will repeat all the steps.

If child succeeds in taking the lock this time,

--sleeps for a sec

--Checks for the counter value if it is greater than 25, it exits without unlocking the mutex ().

Else increases the counter value

--unlocks the mutex (assuming the if condition is not true).

In this mean time, if main thread tries to take the lock, it will go to sleep.

--tries to take the lock again.

If it succeeds it will repeat all the steps

### **Explanation of Output:**

The output for this program is highly unpredictable and depends which thread has the lock for and how many times.

Case 1) If main thread gets the lock all the time, it will print indefinitely since it doesn't have a restriction.

Case 2) If child thread gets the lock 25 times back to back, it will print all the numbers till 26 and exits.

Case 3) If counter value is 26 and from then main thread has the lock, then we will be able to print number above 26.

### Print statements stop appearing :

```
harry@harry-HP-Pavilion-15-Notebook-PC:~/shell_test1$ ./assignment_mt
Main: counter=1
Child1: counter=2
Child1: counter=3
Child1: counter=4
Child1: counter=5
Child1: counter=6
Child1: counter=7
Child1: counter=8
Child1: counter=9
Child1: counter=10
Child1: counter=11
Child1: counter=12
Child1: counter=13
Child1: counter=14
Child1: counter=15
Child1: counter=16
Child1: counter=17
Child1: counter=18
Child1: counter=19
Child1: counter=20
Child1: counter=21
Child1: counter=22
Main: counter=23
Child1: counter=24
Main: counter=25
Child1: counter=26
Main: counter=27
^C
harry@harry-HP-Pavilion-15-Notebook-PC:~/shell_test1$
```

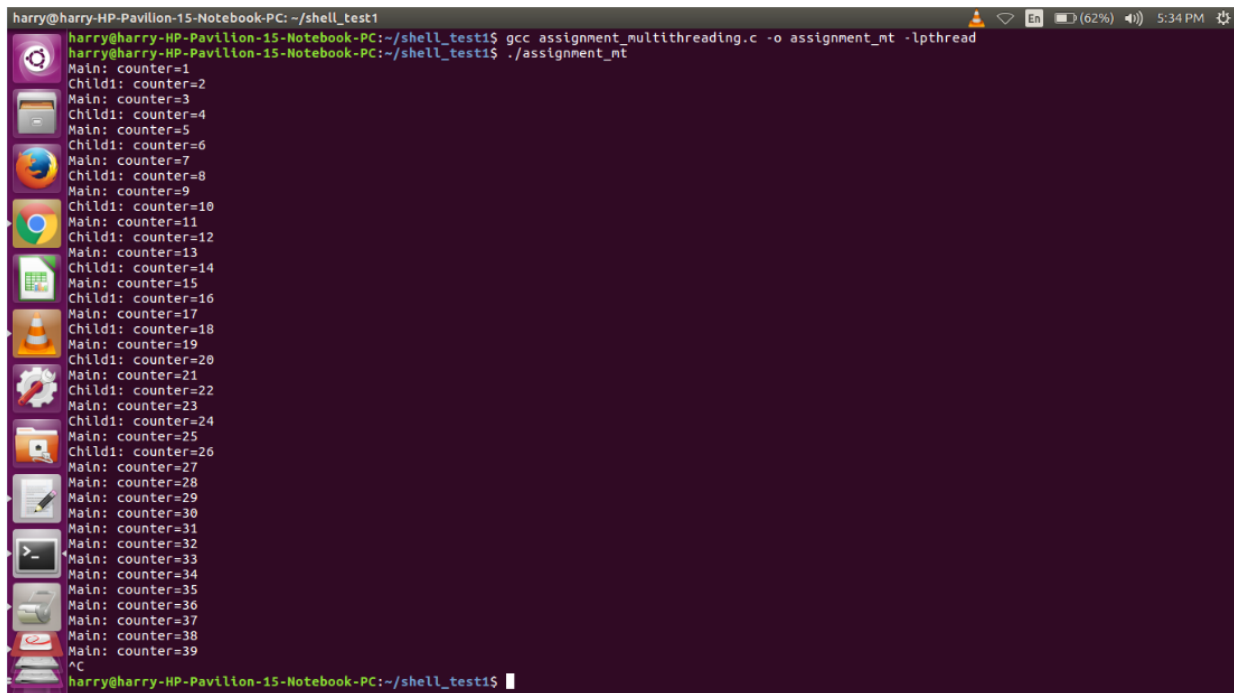
In the screenshot, Print statements stop appearing after “Main : counter =27” since lock has gone to child thread. In the child thread if condition fails and exits without unlocking the mutex. Because of this main thread won't get the lock for the mutex.

### 2. Modify the program and write a correct version that fixes the problem that you just discovered. Explain how you fixed the program.

```
int counter;

void *child1(void *arg)
{
    while(1){
        pthread_mutex_lock(&mutex_1);
        sleep(1);
        if(counter > 25){
            pthread_mutex_unlock(&mutex_1); //UNLOCKING THE MUTEX
            pthread_exit(NULL); //EXIT AFTER UNLOCK
        }
        else
            counter++;
        pthread_mutex_unlock(&mutex_1);
        printf("Child1: counter=%d\n", counter);
    }
}
```

Output after the unlock line is put inside the if condition.



```
harry@harry-HP-Pavilion-15-Notebook-PC: ~/shell_test1
harry@harry-HP-Pavilion-15-Notebook-PC:~/shell_test1$ gcc assignment_multithreading.c -o assignment_mt -lpthread
harry@harry-HP-Pavilion-15-Notebook-PC:~/shell_test1$ ./assignment_mt
Main: counter=1
Child1: counter=2
Main: counter=3
Child1: counter=4
Main: counter=5
Child1: counter=6
Main: counter=7
Child1: counter=8
Main: counter=9
Child1: counter=10
Main: counter=11
Child1: counter=12
Main: counter=13
Child1: counter=14
Main: counter=15
Child1: counter=16
Main: counter=17
Child1: counter=18
Main: counter=19
Child1: counter=20
Main: counter=21
Child1: counter=22
Main: counter=23
Child1: counter=24
Main: counter=25
Child1: counter=26
Main: counter=27
Main: counter=28
Main: counter=29
Main: counter=30
Main: counter=31
Main: counter=32
Main: counter=33
Main: counter=34
Main: counter=35
Main: counter=36
Main: counter=37
Main: counter=38
Main: counter=39
^C
harry@harry-HP-Pavilion-15-Notebook-PC:~/shell_test1$
```

When the counter goes over 25 and the child thread is run, the code enters the “if” block and unlocks the mutex before exiting the thread. So when the other thread wants to access the counter variable, it is free to do so. Since we are killing the child thread the Parent thread Locks the mutex, increments the counter and unlocks the mutex. The parent thread continues infinitely till it is interrupted.