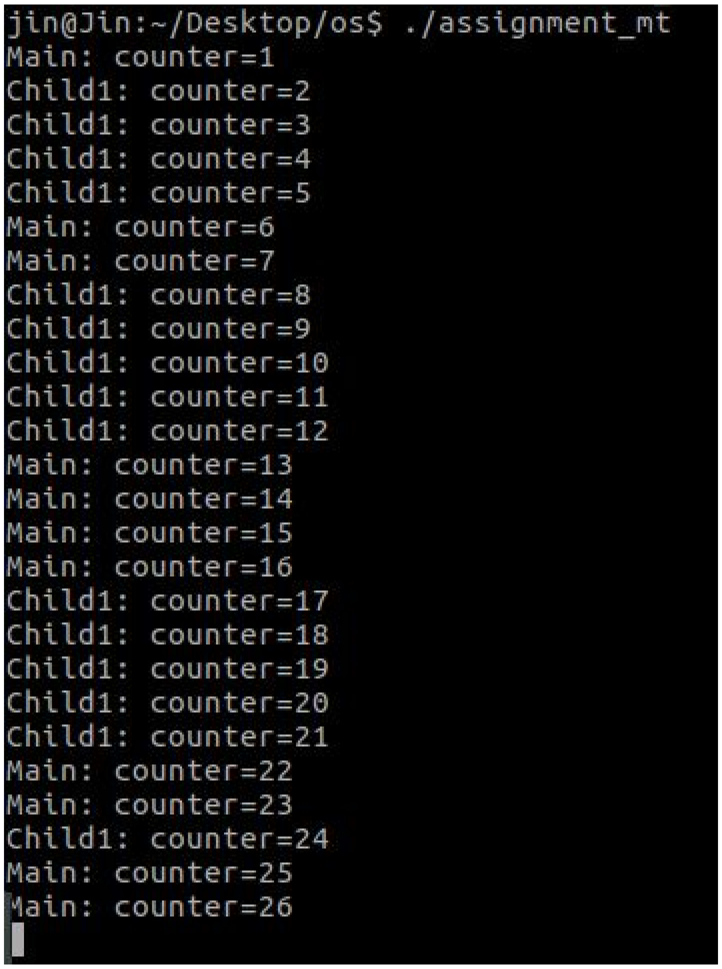
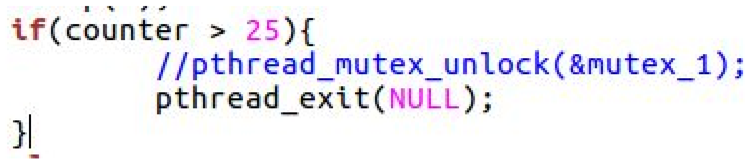
This is the output. There are 2 threads and they will compete with each other to seize the cpu. And the code add lock to the global variable counter to ensure only 1 thread can modify it. That’s why sometimes child thread prints and sometimes main thread prints.



```
jin@Jin:~/Desktop/os$ ./assignment_mt
Main: counter=1
Child1: counter=2
Child1: counter=3
Child1: counter=4
Child1: counter=5
Main: counter=6
Main: counter=7
Child1: counter=8
Child1: counter=9
Child1: counter=10
Child1: counter=11
Child1: counter=12
Main: counter=13
Main: counter=14
Main: counter=15
Main: counter=16
Child1: counter=17
Child1: counter=18
Child1: counter=19
Child1: counter=20
Child1: counter=21
Main: counter=22
Main: counter=23
Child1: counter=24
Main: counter=25
Main: counter=26
```

But, when main thread print out 26, it seems no statement after that point. From child thread code, we can find that if counter is larger than 25, the child thread will exit. But, at this time, there is still a lock on global variable counter. It means that the child thread add a lock and then exit. So, no one else can access to the global variable counter. That’s why main thread is waiting here.



```
if(counter > 25){
        //pthread_mutex_unlock(&mutex_1);
        pthread_exit(NULL);
}
```

As for how to solve this problem, just add unlock code “pthread_mutex_unlock(&mutex_1);” before the child thread exits. So the main thread has access to this global variable again.