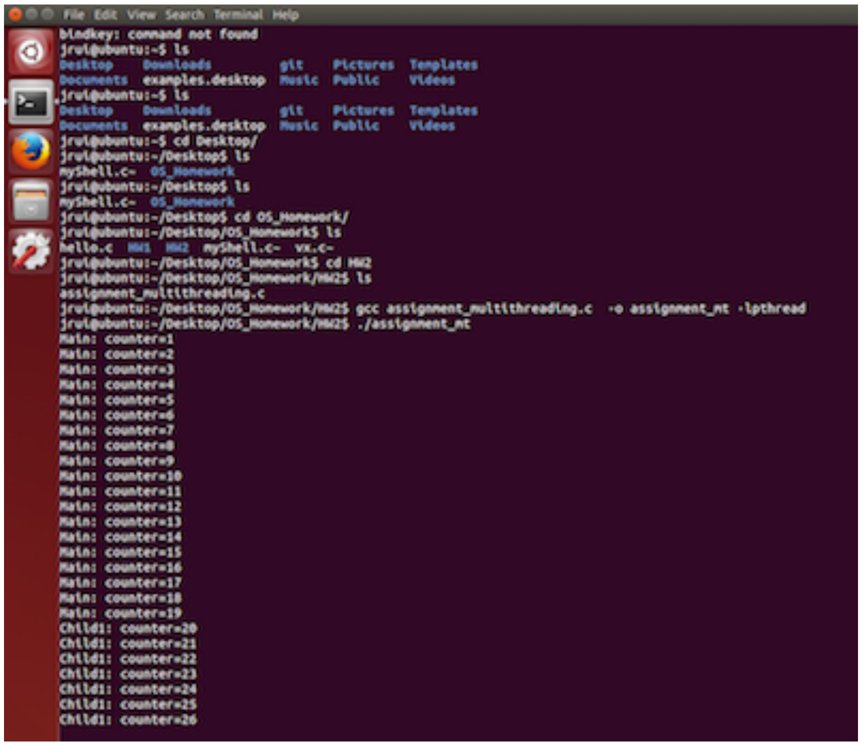
Multithreading - Programming Assignment Report

Name: Jinglong Li
NYU ID: N13903873

1. Compile and run the program on a Linux system. Find out what results it produces and explain the results.

(1) Run Screenshot:



```
bindkey: command not found
jrui@ubuntu:~$ ls
Desktop    Downloads         git    Pictures  Templates
Documents  examples.desktop  Music  Public    Videos
jrui@ubuntu:~$ ls
Desktop    Downloads         git    Pictures  Templates
Documents  examples.desktop  Music  Public    Videos
jrui@ubuntu:~$ cd Desktop/
jrui@ubuntu:~/Desktop$ ls
myShell.c~  OS_Homework
jrui@ubuntu:~/Desktop$ ls
myShell.c~  OS_Homework
jrui@ubuntu:~/Desktop$ cd OS_Homework/
jrui@ubuntu:~/Desktop/OS_Homework$ ls
hello.c  HW1  HW2  myShell.c~  vx.c~
jrui@ubuntu:~/Desktop/OS_Homework$ cd HW2
jrui@ubuntu:~/Desktop/OS_Homework/HW2$ ls
assignment_multithreading.c
jrui@ubuntu:~/Desktop/OS_Homework/HW2$ gcc assignment_multithreading.c  -o assignment_mt -lpthread
jrui@ubuntu:~/Desktop/OS_Homework/HW2$ ./assignment_mt
Main: counter=1
Main: counter=2
Main: counter=3
Main: counter=4
Main: counter=5
Main: counter=6
Main: counter=7
Main: counter=8
Main: counter=9
Main: counter=10
Main: counter=11
Main: counter=12
Main: counter=13
Main: counter=14
Main: counter=15
Main: counter=16
Main: counter=17
Main: counter=18
Main: counter=19
Child1: counter=20
Child1: counter=21
Child1: counter=22
Child1: counter=23
Child1: counter=24
Child1: counter=25
Child1: counter=26
```

(2) The results it produces:

First, what is printed in terminal is from `Main: counter=1` to `Main: counter=19`.
Then, what is printed in terminal is from `Child1: counter=20` to `Child1: counter=26`.
Finally, deadlock happens and the program stuck here.

(3) Explain the results:

First, `Main thread` controls the CPU and run its `while` loop until `counter = 18`, which print `Main: counter = 19` in terminal(because in the while loop, `print` operation is after `counter++`). Then `Main thread` is suspended and `child1 thread` get the CPU control and get into its while loop. It increase the counter as `Main thread` does and print the counter result until `counter = 26`.
Then the deadlock happens because the `if` sentence make the `child1 thread` exit before the mutex is unlocked. So after `child1 thread` exit, the mutex is still locked and the `Main thread` keep waiting the mutex to be unlocked, which is impossible since the `child1 thread` has exited and the code that make mutex unlocked will never be excuted.

2. Why do the print statements stop appearing after a certain point in the program ? Explain.

If we run this code, print statements stop appearing after a certain point because deadlock happened. The deadlock happens because when the `counter` is larger than 25 and if the `child1 thread` is running, the code makes the `child1 thread` exit before the mutex is unlocked. So after `child1 thread` exit, the mutex is still locked and will lock foever. Meanwhile the `Main thread` keep waiting the mutex to be unlocked, which is impossible since the `child1 thread` has exited and the code that make mutex unlocked will never be excuted.

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

(1) Explain how you fixed the program:

I move the `if` block in the `child1 thread` to the position after the mutex is unlocked. And I move all the `counter++` and `print` operation into the block between `lock` and `unlock` to protect them.
So, if the counter is larger than 25, the `child1 thread` will increase the counter and print 1 time, then unlock the mutex and exit. **The deadlock won't happen and then `Main thread` will be excuted.**

(2) Fixed Version:

```c
#include <stdio.h>
#include <pthread.h>
#include <unistd.h>
pthread_mutex_t mutex_1;

int counter;

void *child1(void *arg)
{
  while(1){
    pthread_mutex_lock(&mutex_1); // lock
    sleep(1);
    counter++;
    printf("Child1: counter=%d\n", counter);
    pthread_mutex_unlock(&mutex_1); // unlock
    if (counter > 25) { // after mutex unlocked, judge the counter
      pthread_exit(NULL);
    }
  }
}


int main(void)
{
  pthread_t tid1;
  counter = 0;

  pthread_mutex_init(&mutex_1,NULL);
  pthread_create(&tid1,NULL,child1,NULL);
  do {
    pthread_mutex_lock(&mutex_1);
    sleep(1);
    counter++;
    pthread_mutex_unlock(&mutex_1);
    printf("Main: counter=%d\n", counter);
  } while(1);
  pthread_exit(0);
}
```