Project-1

Multithreaded Programming and Synchronization

Submitted to

Dr. Liting Hu

Professor

School of Computing & Information Sciences

SCIS, FIU

By

Md. Abdullah Al Mamun

Student Id# 6144422

School of Computing & Information Sciences

For assessment as part of

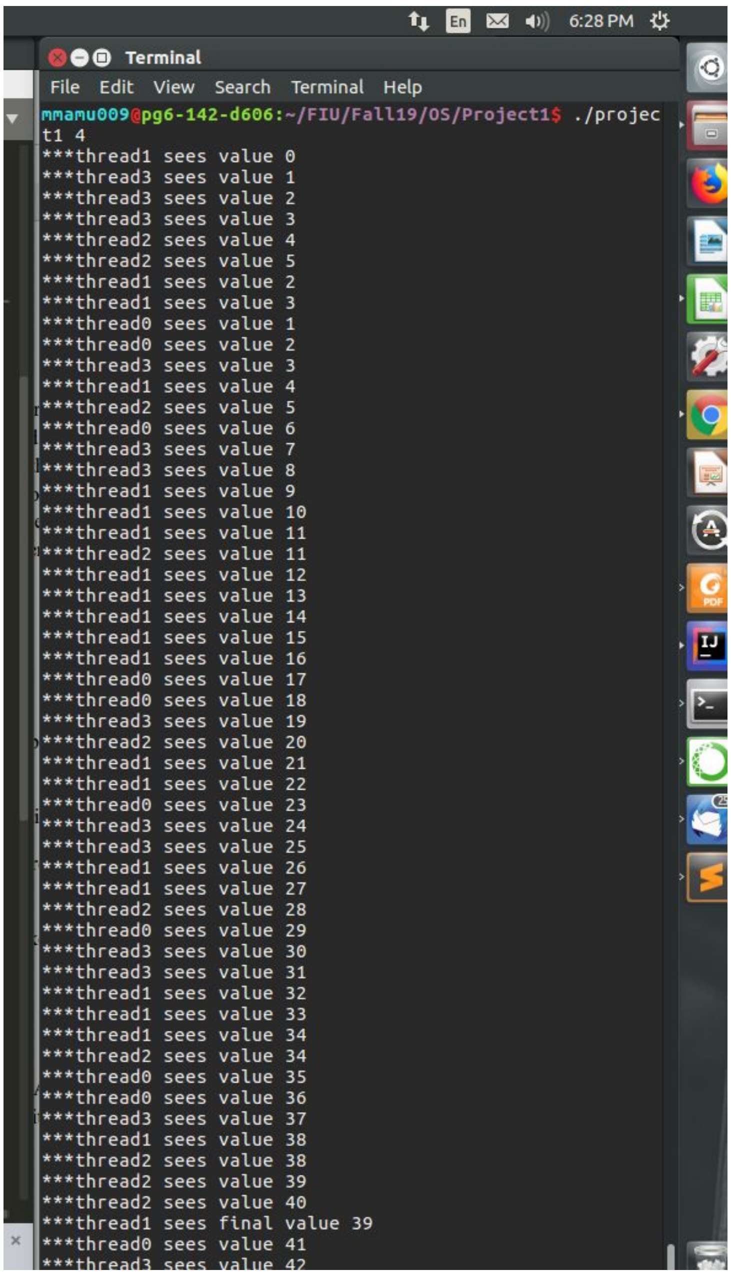
COP 5614 – Operating Systems Principles

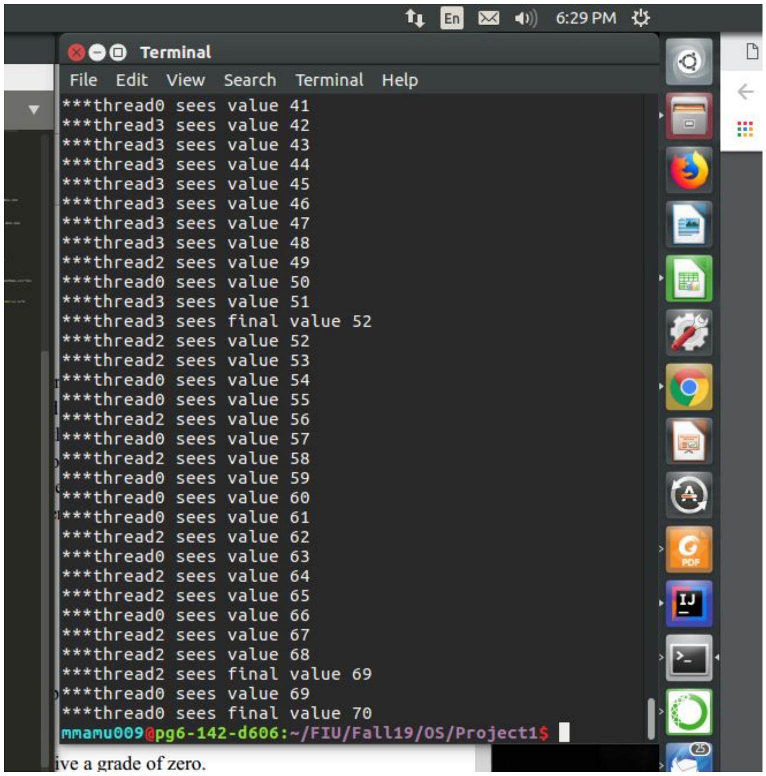
Fall 2019

Output Discussion

Step1: Output without Pthreads synchronization

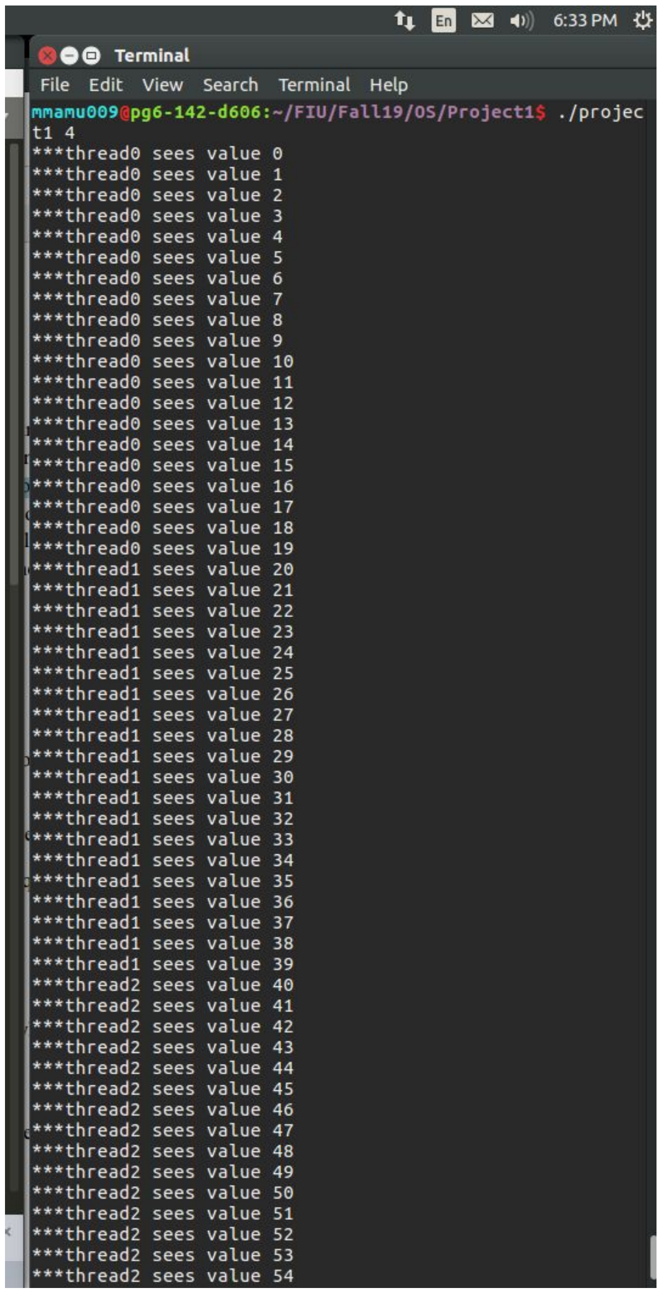
In step1, no mechanism has been used for synchronization, thus, there is no idea about the final see. For example, in the case of 4 threads, the final see is different for one thread to another. Thread1 see final value 39 whereas thread3 see final value 52. Even if I repeatedly run the program multiple times, value of the final see always changed but never synchronized. Similar behavior has been observed for different thread sizes e.g. 2, 3, 4 up to 200. However, the program with a single thread observed as synchronization which is obvious. The output of a 4 thread run is depicted in the figure below.




6:29 PM
Terminal
File Edit View Search Terminal Help
***thread0 sees value 41
***thread3 sees value 42
***thread3 sees value 43
***thread3 sees value 44
***thread3 sees value 45
***thread3 sees value 46
***thread3 sees value 47
***thread3 sees value 48
***thread2 sees value 49
***thread0 sees value 50
***thread3 sees value 51
***thread3 sees final value 52
***thread2 sees value 52
***thread2 sees value 53
***thread0 sees value 54
***thread0 sees value 55
***thread2 sees value 56
***thread0 sees value 57
***thread2 sees value 58
***thread0 sees value 59
***thread0 sees value 60
***thread0 sees value 61
***thread2 sees value 62
***thread0 sees value 63
***thread2 sees value 64
***thread2 sees value 65
***thread0 sees value 66
***thread2 sees value 67
***thread2 sees value 68
***thread2 sees final value 69
***thread0 sees value 69
***thread0 sees final value 70
mmamu009@pg6-142-d606:~/FIU/Fall19/OS/Project1$
ive a grade of zero.

Step2: Output with Pthreads synchronization

In step2, a locking mechanism has been introduced for synchronization, thus, the final see for all threads is always same. For example, in the case of 4 threads, the final see is 80 for all threads. More preciously, Thread1 see final value 80 and thread3 see final value 80 too. Even if I repeatedly run the program multiple times, value of the final see always same which is a clear indication of synchronization. Similar behavior has been observed for different thread sizes e.g. 2, 3, 4 up to 200. Similarly, the program with a single thread observed as synchronization which is obvious. The output of a 4 thread run is depicted in the figure below.



```
***thread2 sees value 51
***thread2 sees value 52
***thread2 sees value 53
***thread2 sees value 54
***thread2 sees value 55
***thread2 sees value 56
***thread2 sees value 57
***thread2 sees value 58
***thread2 sees value 59
***thread3 sees value 60
***thread3 sees value 61
***thread3 sees value 62
***thread3 sees value 63
***thread3 sees value 64
***thread3 sees value 65
***thread3 sees value 66
***thread3 sees value 67
***thread3 sees value 68
***thread3 sees value 69
***thread3 sees value 70
***thread3 sees value 71
***thread3 sees value 72
***thread3 sees value 73
***thread3 sees value 74
***thread3 sees value 75
***thread3 sees value 76
***thread3 sees value 77
***thread3 sees value 78
***thread3 sees value 79
***thread2 sees final value 80
***thread1 sees final value 80
***thread0 sees final value 80
***thread3 sees final value 80
mmamu009@pg6-142-d606:~/FIU/Fall19/OS/Project1$
```

Reasons for the difference

In step1

- The shared variable was unlocked, thus any thread can access/execute the critical section aka shared variable before first thread finishes.
- Thread barrier has not used, thus none of the threads are waiting until previous threads finish.

Therefore, the final sees are not the same for all the threads that result in unsynchronized output because no synchronization technique is applied in step1. The figure of Pthreads with synchronization shows the inconsistency between outputs.

In step2

- The shared variable was locked using mutex and release after used, thus no thread can access the shared variable until the previous thread completed its use.
- Thread barrier has used, thus all threads are waiting to go together that results in the same output (final sees) for all the threads.

Therefore, the final sees are the same for all the threads and output is well synchronized because one of the synchronization techniques e.g. mutex is used in step2. The figure of Pthreads without synchronization shows the consistency between threads.

Program Design

To overcome the race condition, I have used a popular synchronization technique called **Mutexes**. It protects the critical region from unexpected use. I have organized my program as follows.

- I have initialized, created, and joined multiple threads using **Pthread**[1] in the ***main*** function.
- **Mutex**, a locking mechanism is initialized in the main function.
- I set the mutex before using the shared variable and release it after using it in the ***SimpleThread*** function where the shared variable is increasing.
- I have also used **Pthread** barrier before assigning the final value to the shared variable at the end of ***SimpleThread*** function to allow all the threads to wait until the finish.
- Finally, ***pthread_mutex_destroy*** has been used at the end of the main function to destroy used mutexes.
- Moreover, I have checked the number of command-line parameters at the beginning of the main function. Also, I make sure the user can only pass a positive number as an argument which is the number of threads.

Conclusion

I have developed my code in locking fashion using Mutex to ensures synchronization among the threads while working on the shared variable.

References

[1] Pthreads Primer: http://pages.cs.wisc.edu/~travitch/pthreads_primer.html