Michael Weaver

CST – 221

John Zupan

1/28/2018

GitHub Link: <https://github.com/battousairurik/CST-221>

**Monitors vs Semaphores**

Description of Scenario

With this scenario both threads are utilizing a shared resource, in this case a string, but ultimately any shared resource can be substituted. Synchronization is necessary because each thread seeks to utilize the resource, setting its value, and then in this case printing the value to the terminal. If synchronization was not present, then the value of the shared resource could be changed before a process exits its critical region.

Comparison for scenario

For this scenario both Monitor and Semaphore act exactly the same. Both lock out the other thread while the initial process finishes then sends a signal to unlock and allow the other thread access to the shared resource. Because we are dealing with the C programming language, the actual Monitor class does not exist and we are forced to use a Mutex to represent. This is a huge con, given that the materials we are supposed to be using do not even exist.

Recommendation for scenario

For this scenario it is simpler to use a Semaphore, both in implementation and code consistency. Ultimately both execute and simpile the same, so the main reason for Semaphore use is its ease. Instead of working with the locking mechanism, you simply call the wait and signal functions.

Program Execution Results

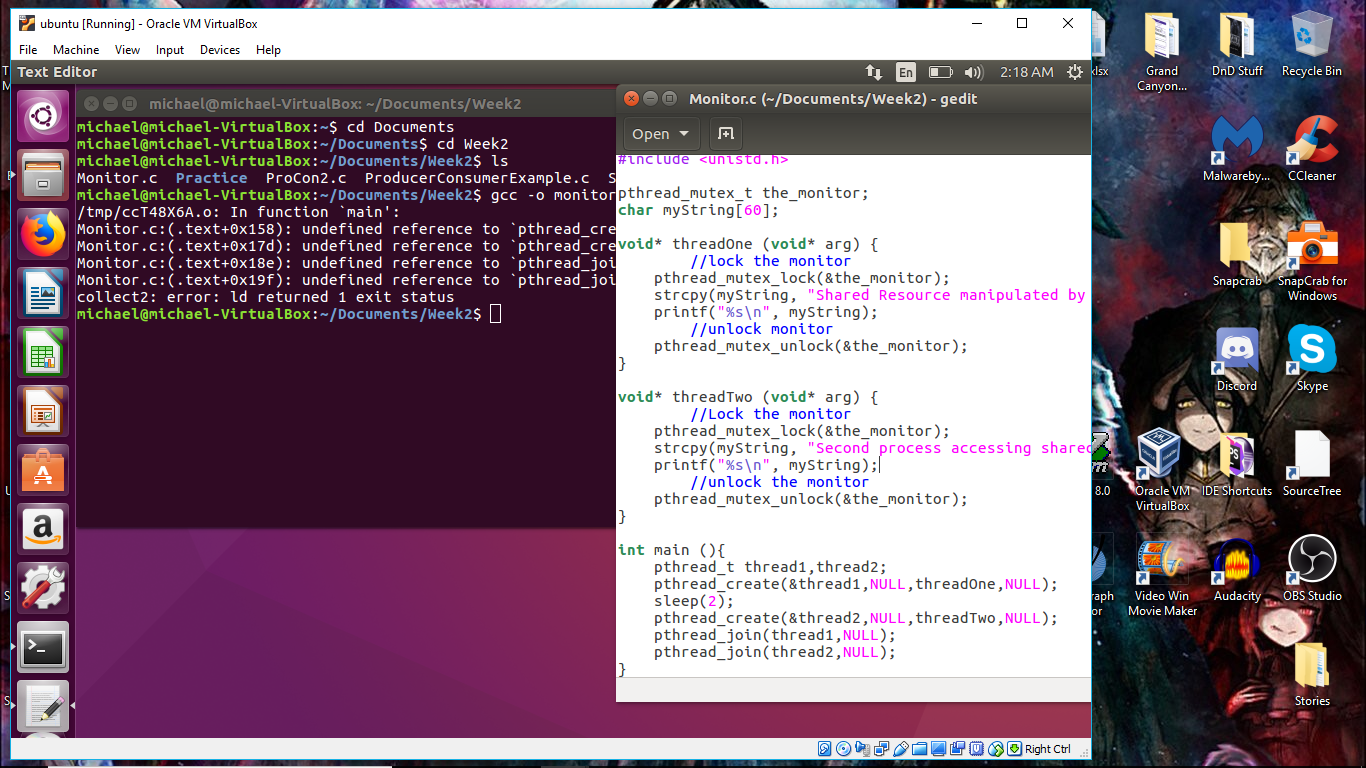
Incomplete at this time. Neither program will compile, so no execution can be accomplished.

Analysis

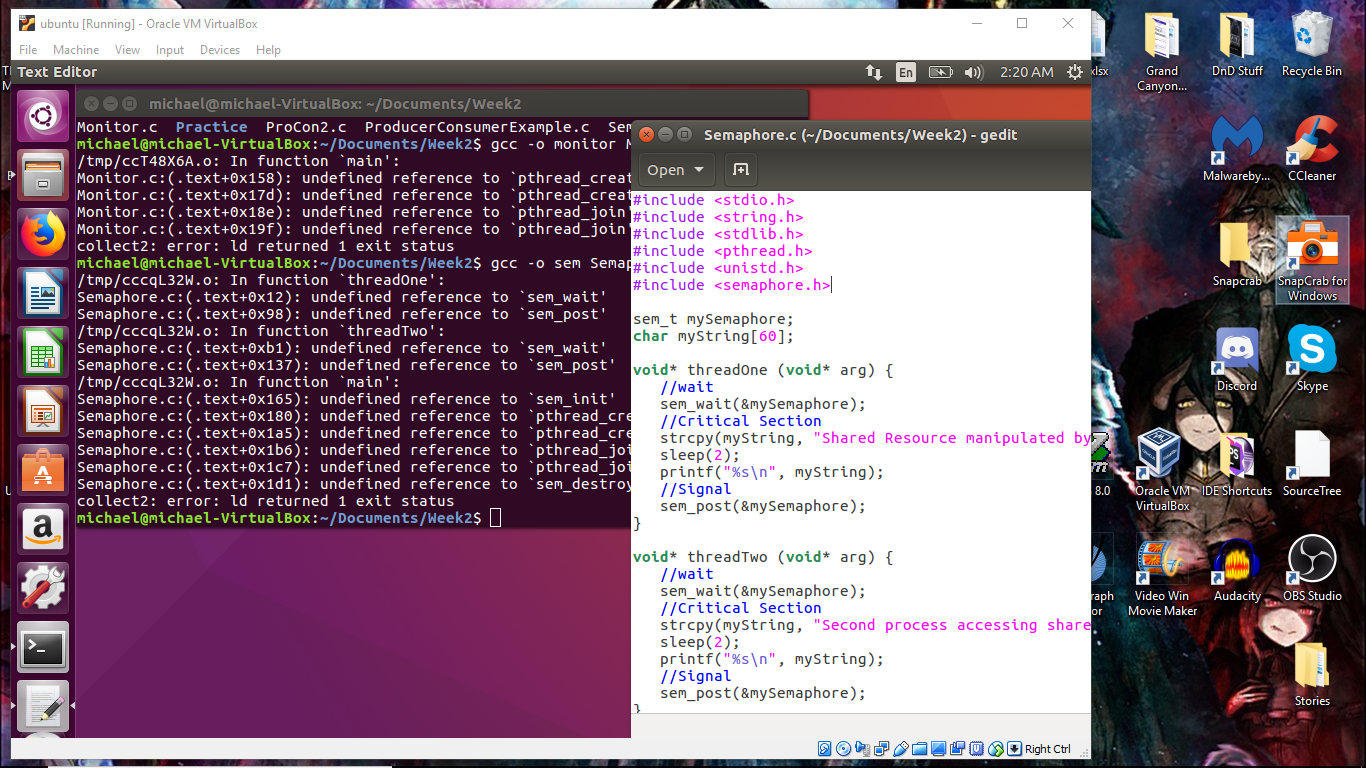
In theory, Monitors are more efficient than Semaphores because of their locking mechanism compared to the semaphores wait and signal. Both are nearly identical to the other in implementation given that they both use pthreads, though in this case we are not actually using a Monitor but a Mutex to simulate the best the language can.

Screenshots

Monitor



Semaphore



Afterthoughts

I would really appreciate feedback on how to solve these issues. The fact that neither class will compile causes a huge issue for me. To my knowledge there should be nothing preventing both classes form being implemented, though both display undefined reference issues causing the application to exit immediately. I have no idea how to solve this problem and therefor cannot make any additional progress at this point in time.

References

Techdifferences.com. (2017). *Difference between semaphore and monitor in OS.* Retrieved from <https://techdifferences.com/difference-between-semaphore-and-monitor-in-os.html>

GeeksforGeeks. (n.d.). *How to use POSIX semaphores in C language*. Retrieved from <https://www.geeksforgeeks.org/use-posix-semaphores-c/>

Tausen. (2013). No Title. Retrieved from <https://gist.github.com/tausen/4261887>

Arora, Himanshu. (2012). *How to Use C Mutex Lock Examples for Linux Thread Synchronization*. Retrieved from <https://www.thegeekstuff.com/2012/05/c-mutex-examples/>