Assignment 2: Monitors and Semaphores

Austin Johns

1 March 2020

William Hurst

CST-315

**Need for Synchronization in an Operating System**

In this project, the main goal is to recognize the importance for a synchronization system. A synchronization system in an operating system with multiple threads or tasks being completed is absolutely crucial. Without any synchronization of the processes in an operating system, there may be issues with data consistency, process deadlocks, and race conditions. A race condition is when two or more operations are executed simultaneously. Without monitors or semaphores these conditions will have no schedule and processes will take place out of the order they need to take place in. This is important because if there is a critical section of an execution that needs to take place before any other execution, and it doesn’t, the execution will fail. For example, if a program installer is running without any synchronization and there is no designated installation folder, the process will fail, and the program will not install. Monitors and semaphores allow for mutual exclusion, a rule that only allows one process to be completed in a critical section at any given time. Another need in an operating system is mutex locks. In a semaphore, a process can be locked until the other process is completed. Once the other process is completed, the locked process is unlocked and free to execute.

**Justifications for the use of Monitors and Semaphores**

In our case, a monitor is used to be accessed from multiple threads. A monitor provides mutual exclusion, meaning that only one thread can perform an action at a time. If one process is running, a monitor will ensure that another process will wait until the other process is finished before running.

A semaphore is a bit different than a monitor. A semaphore is considered a lower-level object because it is just a counter. When a semaphore’s count is positive, a process acquires the semaphore and the semaphore’s count is decremented. When a thread is done performing an action, the thread releases the semaphore and increments its count. This count is useful because as soon as the count reaches a negative number, it forces any other processes to wait until the semaphore is incremented.