**Resource Manager Approach Explanation**

The system that I have implemented revolves around the use of mutex locks so that processes looking to access resources can quickly check to see if that particular resource is in fact available at any given time. In short, each resource generated by the system for potential processes to use is assigned a mutex lock so that they can each individually be locked whenever a process is using that resource and then later unlocked when the process is finished with it. Originally, I attempted to use a counting semaphore to accomplish this same task but quickly realized that I had overcomplicated the given problem per usual, so I decided to shift gears to make sure that I was able to build an effective resource handler for SAC-SimOS.

**Resource Manager Approach Evaluation**

Although my Mutex Lock solution works effectively in the SAC-SimOS and ensures that no two processes are ever able to access a single resource at the same time, the reality is that it would very likely be inefficient at best in a more complicated system. This is because keeping track of every individual resource that a process could potentially access and keeping tabs on whether each of those resources are currently “locked” at all times would give a more realistic system a lot of unnecessary memory overhead that it simply shouldn’t have to deal with. It would also be wasteful from a time management perspective because in a real system an unlimited number of processes that only read from a resource should theoretically be allowed to access that particular resource without restraint. This unfortunately wouldn’t be the case with the system that I have implemented as every resource is automatically locked anytime that a single process uses it. In other words, my mutex lock approach is really good at ensuring the mutual exclusion of resources, particularly for ensuring that reading and writing processes are unable to access the same resources at the same time, but the reality is that in a more realistic system there tends to be a much greater volume of processes simply looking to read from a given resource as opposed to writing to one.