Albert Chang

Csc 33200 – S

**Semaphores vs. Pthreads**

Both semaphore and pthread based solutions are sufficient to solving the cigarette problem. With the pthread approach, a mutex lock would be created so that only one process can be active at a time by locking before the execution of a process and unlocking once the process completes its task. This way processes won’t accidentally pull from the same resource concurrently and return conflicting results. In a sense, the lock is what ultimately controls the flow of actions between the processes since only the process assigned with the lock can act. This method is efficient when handling multiple processes that require the same limited resources in order to complete its task. For lab 7 however, we assume that resources are unlimited so the issue of limitations are removed. However, only one smoker (process) can come up and smoke at a time so we still apply a mutex lock to allow a single smoker to perform the action at a time.

The semaphore approach differs in a sense that the program would keep track of the total amount of resources currently available and allocated it to each individual process with its own separate mutex lock. Any remaining processes that need certain resources not currently available will have to wait until it’s available again before executing. A counter would be placed on each semaphore as well where if a semaphore is requesting access to readily available resources to execute, the semaphore count decrements by 1. When they are completed and no longer need the resources, the semaphore will return the resources and increment by 1 signaling the next process in queue can proceed. With regards to the cigarette problem, the need to keep a counter on the resources is unnecessary because we assume there is an infinite supply. However, only one process can occupy the smoking action at once so that variable will be considered the resource that the other two semaphores would need to wait for.

Both approaches have its benefits and drawbacks when it comes to implementation. For pthreads, if the resource is extremely limited among concurrent processes, this would be an efficient method to properly conduct a flow of actions amongst them while avoid conflicts. However, the drawback is that it may be slow because due to the lock, only one process can access the readily available resources at a time. As for the semaphore approach, it’s beneficial in a sense that most or all the resources would be actively used at once which definitely improves efficiency and reduces wait times. However, the drawback would be that the implementations are more extensive because we take into account all the resources available and allocate accordingly depending on what processes require to execute their tasks.