CSC332 – Operating Systems

Task 6 – Cigarette Smokers Problem

**Part 1**: In the first part of the assignment, we were to synchronize the agent and smoker processes using semaphores. To begin the implementation, we use the pseudocode provided in the link. Accordingly, there were 5 semaphores that were created: tobacco, paper, match, agent, and lock used to represent the smoker and agent processes. To synchronize these processes, we essentially use the agent process to manipulate the 3 smoker processes (i.e. their semaphores) to determine which 2 of the three ingredients are placed on the table and then send the signal to le the smoker know when they can smoke. In addition, we include the mutex lock we created to ensure that there are no race conditions. Furthermore, to implement the 3 smoker processes we use fork() as well as the pseudocode provided since they are analogous. Lastly, there were slight modifications made where instead of having the agent infinitely supply two of three ingredients, we set a finite number of times in which the agent can supply, that is 10 times. Once done, the semaphores are to be killed (had some slight issues where when trying to kill the semaphores, it went on an infinite loop so had to comment out. Ideally, it should work).

**Part 2**: In the second part of the assignment, instead of using semaphores to synchronize the agent and smoker processes, we use pthreads. Similar to the semaphore approach, we use mutex pthreads instead. In addition, there were no need to create child processes using fork(), rather we created two functions: one to represent the agent process and the other to represent the smoker process (i.e. one function for all 3 smoker processes). Using the provided information in the link, we are able to create a smoker function that takes in an argument to determine which process to call, for example if we have an argument of 1 then we call the smoker with match process. In addition, all functions have almost exact same code as the semaphore approach with the only difference being instead of using semaphore lock and unlock, we use pthreads version. Just like before, the agent process will determine which of the smoker process will be called.

|  |  |  |
| --- | --- | --- |
| Comparison | Merits | Demerits |
| Semaphore | Allows multiple threads/processes to enter the critical section  Can stop the processes using the parent process and wait | Typically slower than pthreads as there are more processes to check  Need to be careful in placing the Ps and Vs to ensure no racing conditions occur |
| Pthread | More efficient and faster than the semaphore approach as there is no need to fork() a child process.  Can share functions/threads with each other | Shares global variables so in the case of a medication of one of it can lead to errors  Only one thread can enter the critical section |