O.S. Lab 2: POSIX Threads and Semaphores

Finally, we consider an interesting and challenging synchronization problem, the cigarette smokers' problem originally proposed by Patil[1]. His claim was that the problem could not be solved with semaphores, which is true if you make some pretty artificial assumptions. Our version is solvable, just not trivially so.

Suppose that a cigarette requires three ingredients: tobacco, a rolling paper and a match. There are three chain smokers, each of whom has only one of these ingredients with infinite supply. There is an agent who has an infinite supply of all three ingredients. To make a cigarette, the smoker who has tobacco must have the other two ingredients, a paper and a match. (You can figure out what happens with the smokers who start with papers and matches, respectively.) The agent and smokers share a table. The agent randomly generates two ingredients and notifies the smoker who needs these two ingredients. Once they are taken from the table, the agent supplies another two. On the other hand, each smoker waits for the agent's notification. Once notified, the smoker picks up the ingredients, makes a cigarette, smokes for a while, and goes back to the table waiting for his next ingredients. The problem is to come up with an algorithm for the smokers using semaphores as synchronization primitives.

Now, all jokes aside, this is an actual problem related to computer science. The agent represents an operating system that allocates resources, while the smokers represent applications that need resources. The problem is to make sure that if resources are available that would allow one or more applications to proceed, those applications should be awakened. Conversely, we want to avoid waking an application if it cannot proceed.

The one of Patil's restrictions that we will retain is that you are not allowed to modify the agent code. (Indeed, if the agent represents an o.s. it makes sense to assume that you don't want to modify it every time a new application comes along.) The agent uses these binary semaphores and actually consists of three concurrent threads (one of which is given at right). Each waits on `agentSem`. Each time it's signaled, one agent thread wakes up and provides ingredients by signaling two other semaphores.

```
Semaphore agentSem = 1
          tobacco = 0
          paper = 0
          match = 0
```

```
while (true) {
   P(agentSem)
   V(tobacco)
   V(paper)
}
```

(One thing you will need to change when you implement this is to make the three agent threads sleep for a random period of time – up to 200 milliseconds – before beginning to wait on `agentSem`. This will hopefully mix things up and make this more interesting.)

[1] S.S. Patil. *Limitations and capabilities of Dijkstra's semaphore primitives for co-ordination among processes*. Proj. MAC, Computational Structures Group Memo 57, February 1971.

```
Boolean isTobacco = false
        isPaper = false
        isMatch = false
Semaphore tobaccoSem = 0
          paperSem = 0
          matchSem = 0
          mutex = 1
```

This becomes a hard problem because you can show that the natural solution leads to deadlock. To get around this, Parnas[2] proposed the use of three additional "pusher" threads that respond to the signals from the agents, keep track of the available ingredients and signal the appropriate smoker. We first need three Boolean variables to indicate whether or not an ingredient is on the table, three new semaphores to signal the smokers, and a semaphore for preserving mutual exclusion.

```
while (true) {
   P(tobacco)
   P(mutex)
   if (isPaper) {
      isPaper = false
      V(matchSem)
   }
   else if (isMatch) {
      isMatch = false
      V(paperSem)
   }
   else isTobacco = true
   V(mutex)
}
```

Pseudo code for one of the pushers (the one who wakes up when there's tobacco on the table) appears at left. If this pusher finds paper, it knows that the paper pusher has already run, so it can signal the smoker with matches. Similarly, if it finds matches, the smoker with rolling papers is signaled. Finally, if this is the first pusher to run, it cannot signal any smoker, so sets `isTobacco`.

The other pushers are similar. Since they do all the work, the smoker code is trivial. Pseudo code for the smoker with tobacco appears at right below; the others are similar. As above simulate the making and smoking of the cigarette by having the thread sleep for a short period of time (up to 50 milliseconds for both the making and smoking).

```
while (true) {
   P(tobaccoSem)
       Make a cigarette
   V(agentSem)
       Smoke the cigarette
}
```

Parnas' solution was more convoluted than this, and his variations on this problem more complicated, but what's presented here is enough for our purposes. Your mission, should you choose to accept it, is to create a simulation of this problem using C/C++ and POSIX threads. Create three agent threads, three pushers and six smokers (two holding tobacco, two papers, two matches). Each smoker finishes three cigarettes before exiting. (They said something about being hungry as they left.) As such, rather than loop forever, each agent loops six times, and each pusher twelve times. (Think about it.) Remember to join ***all*** threads before your program terminates. When finished submit a copy of your source code to Springboard so that we can compile and execute your program[3].

Last updated 8.5.2019 by T. O'Neil. Previous revisions 10.23.2009.

[2] D.L. Parnas. *On a solution to the Cigarette smoker's problem (without conditional statements).* Communications of the ACM **1975**, *18* (*3*), 181-183.

[3] We are required to mention that neither the University of Akron nor any of its affiliated parties endorse, encourage or condone the use of tobacco or any other controlled substance. The Surgeon General of the United States of America has determined that cigarette smoking is dangerous to your health, causing lung cancer, heart disease, emphysema and pregnancy complications (fetal injury, premature birth and low birth weight). Just say no kids.