Cigarette Smoker's Problem

Problem

There are four processes in this problem: three smoker processes and an agent process. Each of the smoker processes will make a cigarette and smoke it. To make a cigarette requires tobacco, paper, and matches. Each smoker process has one of the three items. I.e., one process has tobacco, another has paper, and a third has matches. The agent has an infinite supply of all three. The agent places two of the three items on the table, and the smoker that has the third item makes the cigarette. Synchronize the processes.

Solution

This seems like a fairly easy solution. The three smoker processes will make a cigarette and smoke it. If they can't make a cigarette, then they will go to sleep. The agent process will place two items on the table, and wake up the appropriate smoker, and then go to sleep. All semaphores except `lock` are initialized to 0. `lock` is initialized to 1, and is a mutex variable.

Here's the code for the agent process.

```
 1   do forever {
 2     wait( lock );
 3     randNum = rand( 1, 3 ); // Pick a random number from 1-3
 4     if ( randNum == 1 ) {
 5        // Put tobacco on table
 6        // Put paper on table
 7        signal( smoker_match );  // Wake up smoker with match
 8      } else if ( randNum == 2 ) {
 9        // Put tobacco on table
10        // Put match on table
11        signal( smoker_paper );  // Wake up smoker with paper
12      } else {
13        // Put match on table
14        // Put paper on table
15        signal( smoker_tobacco ); } // Wake up smoker with tobacco
16      signal( lock );
17      wait( agent );  //  Agent sleeps
18     }  // end forever loop
```

I will give code to one of the smokers. The others are analogous.

```
 1   do forever {
 2      wait( smoker_tobacco );  // Sleep right away
 3      wait( lock );
 4      // Pick up match
 5      // Pick up paper
 6      signal( agent );
 7      signal( lock );
 8      // Smoke (but don't inhale).
 9   }
```

The smoker immediately sleeps. When the agent puts the two items on the table, then the agent will wake up the appropriate smoker. The smoker will then grab the items, and wake the agent. While the smoker is smoking, the agent can place two items on the table, and wake a different smoker (if the items placed aren't the same). The agent sleeps immediately after placing the items out. This is something like the producer-consumer problem except the producer can only produce 1 item (although a choice of 3 kinds of items) at a time.

Cigarette smokers problem

From Wikipedia, the free encyclopedia
Jump to: navigation, search

The **cigarette smokers problem** is a concurrency problem, originally described in 1971 by S. S. Patil.

Contents

[hide]

[edit] Problem description

Assume a cigarette requires three ingredients to smoke:

1. Tobacco
2. Paper
3. A match

Assume there are also three chain smokers around a table, each of whom has an infinite supply of *one* of the three ingredients — one smoker has an infinite supply of tobacco, another has an infinite supply of paper, and the third has an infinite supply of matches.

Assume there is also a non-smoking arbiter. The arbiter enables the smokers to make their cigarettes by selecting two of the smokers at random (nondeterministically), taking one item out of each of their supplies, and placing the items on the table. He then notifies the third smoker that he has done this. The third smoker removes the two items from the table and uses them (along with his own supply) to make a cigarette, which he smokes for a while. Meanwhile, the arbiter, seeing the table empty, again chooses two smokers at random and places their items on the table. This process continues forever.

The smokers do not hoard items from the table; a smoker only begins to roll a new cigarette once he is finished smoking the last one. If the arbiter places tobacco and paper on the table while the match man is smoking, the tobacco and paper will remain untouched on the table until the match man is finished with his cigarette and collects them.

The problem is to simulate all four roles as software programs, using only a certain set of synchronization primitives. In Patil's original formulation, the only synchronization primitive allowed was the semaphore, and none of the four programs were allowed to contain conditional jumps — only the conditional waits implied by the semaphores' `wait` operations.

[edit] Argument

Patil's argument was that Edsger Dijkstra's semaphore primitives were limited. He used the cigarette smokers problem to illustrate this point by saying that it cannot be solved with semaphores. However, Patil placed heavy constraints on his argument:

1. The agent code is not modifiable.
2. The solution is not allowed to use conditional statements or an array of semaphores.

With these two constraints, a solution to the cigarette smokers problem is impossible.

The first restriction makes sense, as Downey says in The Little Book of Semaphores, because if the agent represents an operating system, it would be intractable to modify it every time a new application came along. However, as David Parnas points out, the second restriction makes almost any nontrivial problem impossible to solve:

> It is important, however, that such an investigation [of Dijkstra primitives] not investigate the power of these primitives under artificial restrictions. By *artificial* we mean restrictions which cannot be justified by practical considerations. In this author's opinion, restrictions prohibiting either conditionals or semaphore arrays are artificial. On the other hand, prohibition of "busy waiting" is quite realistic.

[edit] Solution

If we remove the second of Patil's constraints, the cigarette smokers problem becomes solvable using binary semaphores, or mutexes. Let us define an array of binary semaphores *A*, one for each smoker; and a binary semaphore for the table, *T*. Initialize the smokers' semaphores to zero and the table's semaphore to 1. Then the arbiter's code is

```
while true {
    wait(T);
    choose smokers i and j nondeterministically, making the third
smoker k
    signal(A[k]);
}
```

and the code for smoker *i* is

```
while true {
    wait(A[i]);
    make a cigarette
    signal(T);
```

```
    smoke the cigarette
}
```

[edit] References

- *Modern Operating Systems (2nd Edition)*, by Andrew S. Tanenbaum (ISBN 0-13-031358-0)
- *The Little Book of Semaphores*, by Allen B. Downey, http://greenteapress.com/semaphores
- *On a solution to the cigarette smokers' problem without conditional statements*, by David Parnas, Communications of the ACM, 18:181-183, March 1975
- *Limitations and capabilities of Dijkstra's semaphore primitives for coordination among processes*, by Suhas Patil, technical report, MIT, 1971