CS347M Take-home Quiz 3

Prasann Viswanathan

IIT Bombay

Spring Semester 2021-22

Cigarette Smokers Problem

The cigarette smokers problem is one of the classical problems. It was posed by Suhas Patil, who claimed it could be solved by semaphores. In this presentation I attempt to explain and implement the problem in code.

Problem statement

1. Four threads are involved: an agent and three smokers. The smokers loop forever, first waiting for ingredients, then making and smoking cigarettes.
2. We assume that the agent has an infinite supply of all three distinct ingredients, and each smoker has an infinite supply of one of the ingredients.
3. The agent repeatedly chooses two different ingredients at random and makes them available to the smokers. Depending on which ingredients are chosen, the smoker with the complementary ingredient should pick up both resources and proceed.

Restrictions

To model a resource-management problem of operating systems in real situations, the following constraints are applied to the agent:

1. The agent is only allowed to communicate by signaling the availability of a resource using a condition variable or semaphore.
2. The agent is not permitted to disclose resource availability in any other way; i.e., smokers cannot ask the agent what is available.
3. The agent is not permitted to know anything about the resource needs of smokers; i.e., the agent cannot wakeup a smoker directly.
4. Each time the agent makes two resources available, it must wait on a condition variable for a smoker to smoke before it can make any additional resources available.

Crux of the question

When the agent makes two items available, **every smoker** thread can use at least one of them, but **only one** can use both. For example, if the agent makes paper and matches available, both the paper and the matches smokers want one of these, but neither will be able to smoke because neither has tobacco. But, if either of them does wake up and consume a resource, that will prevent the tobacco thread from begin able to smoke and thus also prevent the agent from waking up to deliver additional resources. If this happens, the system is deadlocked. This is the idea to give wrong implementations as well.

Concept of implementation

There are different approaches to implement the strategy mentioned above. I have implemented a solution inspired by resources available online, due to the time constraint.
Each listener is only responsible for a certain type of ingredient event. I use a global **sum** variable to record the events that listeners have received so far. Since we want each pair of ingredient values to give a unique sum value, the ingredient values are initialized to 1, 2 and 4. Consequently, the characteristic values which should trigger an action are 3, 5 and 6 and no others.

Condition variables - Working

We use condition variables m_p, p_t and t_m to show which two ingredients are provided by the agent.
The agent itself has four condition variables, for the three ingredients and one to signal smokers.
The agent code is standard, irrespective of the type of implementation used.
We signal the correct smoker based on the available ingredients.
The smokers listen on the channel for the agent and wait till their needed ingredients arrive.
On signal the correct smoker runs.

Code Snippets CV (1)

```
// To create the agent, constructor
struct Agent* createAgent() {
  struct Agent* agent = malloc (sizeof (struct Agent));
  agent->mutex   = PTHREAD_MUTEX_INITIALIZER;
  agent->paper   = PTHREAD_COND_INITIALIZER;
  agent->match   = PTHREAD_COND_INITIALIZER;
  agent->tobacco = PTHREAD_COND_INITIALIZER;
  agent->smoke   = PTHREAD_COND_INITIALIZER;
  return agent;
}
```

Code Snippets CV (2)

```
// Listens to agent for available ingredient news
void* tobacco_listener(void* av){
  struct Agent* a = av;
  // We need to lock to carry this part out to avoid sleep and no wakeup case
  pthread_mutex_lock(a->mutex);
  while(1){
    pthread_cond_wait(a->tobacco, a->mutex);
    sum=sum+TOBACCO;
    call_smoker(sum);
  }
  pthread_mutex_unlock(a->mutex);
}
```

Code Snippets CV (3)

```
// Signals to correct smoker
void* tobacco_smoker (void* av){
  struct Agent* a = av;
  pthread_mutex_lock(a->mutex);
  while(1){
    pthread_cond_wait(m_p, a->mutex);
    printf("Tobacco smoker is smoking.\n");
    pthread_cond_signal(a->smoke);
    smoke_count [TOBACCO]++;
  }
  pthread_mutex_unlock(a->mutex);
}
```

Semaphores - Working

Here I have referred the solution provided in The Little Book of Semaphores.
The agent uses four semaphores, agentSem, tobSem, matSem and papSem. The first is initialized to 1 the ingredients are initialized to 0.
Three concurrent threads run to simulate the agent, all of them wait on the agent thread.
Next the issue arises that if one semaphore is updated, it is not necessary that the other ingredient on the table is also needed by a selected smoker. We need both ingredients to be the ones the smoker doesn't have.
For this we need three pusher codes to check if both the ingredients are indeed available

```
// Pusher for any time Tobacco is on the table of ingredients offered
pthread_cond_wait(a->tobacco, a->mutex);
pthread_mutex_lock(a->mutex);
  if(isPaper){
    isPaper=false; //reset the variable because we found a match
    pthread_cond_signal(a->match);
  }
  else if(isMatch){
    isMatch=false;
    pthread_cond_signal(a->paper);
  }
  else{
    isTobacco=true;
  }
pthread_mutex_unlock(a->mutex);
```

Wrong Implementation

Wrong implementation technique for both condition variable and semaphore implementation involves the same idea.
Take for example a thread simulating the smoker with tobacco. He is waiting on both paper and matches. If each of the smokers had their logic as:

```
papSem.wait()
matSem.wait()
agentSem.signal()
```

Then the following deadlock could arise:
Suppose tobacco and paper is on the table. The matches smoker is waiting on both tobacco and paper, so seeing tobacco it gets unblocked. However, it is also likely that the tobacco smoker was waiting on paper and gets unblocked as well. This causes the first smoker to block on paper even though paper is available, and results in a deadlock.

Thank you[1]

[1] Sorry for the brevity of the presentation, I started the assignment too late to do it satisfactorily