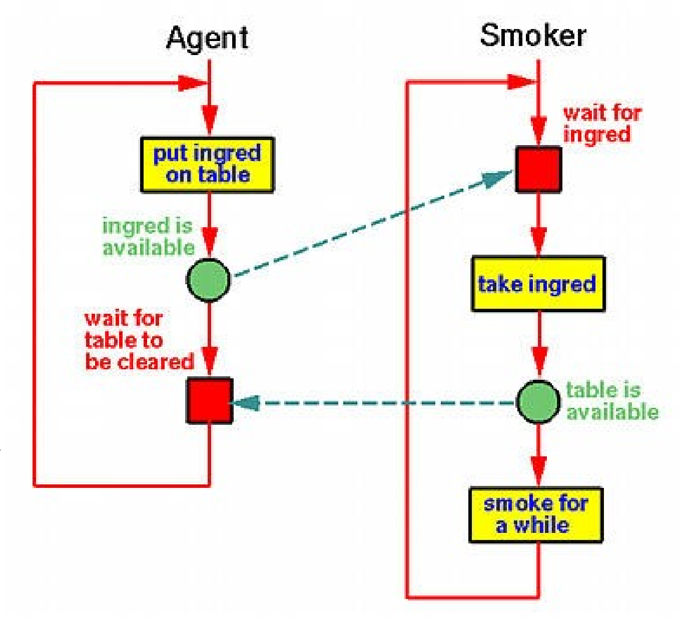
CS 370 – Project #3, Semaphores

Purpose: Become familiar with multi-threaded programming, semaphores, starvation, and deadlock concepts

Points: 125 (Project: 100, Analysis Question: 25 bonus points)

Introduction:

The Cigarette Smokers Problem[1] was first presented by Suhas Patil in 1971. There are three smokers and one agent. A cigarette is made of three ingredients: tobacco, paper and match. Each smoker has infinite supply of one ingredient. For example, the first one has infinite supply of tobacco, the second one has infinite supply of paper and the last one has infinite supply of matches. The agent has infinite supply of all ingredients. The agent repeatedly chooses two ingredients at random and puts them on the table, and the smoker who has the complementary ingredient can take the two ingredients and make a cigarette to smoke. For example, if the agent puts tobacco and paper on the table, the smoker who has infinite supply of matches can make and smoke a cigarette.



As such, there will be four threads in this problem: three smoker processes and an agent process. Each of the smoker processes, when possible, will make a cigarette and smoke it (which will take a random amount of time). The agent process places two of the three items on the table, and the smoker that has the third item makes the cigarette. You will need to synchronize the processes. There are many variations too this problem, so be sure to follow the provided guidance. We will be using the *pthread* semaphore's (not creating our own).

In this problem, the agent represents the operating system and the smokers represent users processes. The operating system should allocate the required resources to the processes and, at the same time, avoid deadlock.

Project:

Implement the Cigarette Smokers Problem in **C** (not C++).
See additional guidance on the following pages.

1 For more information, refer to: https://en.wikipedia.org/wiki/Cigarette_smokers_problem

Specifications
Global variables/constants (and no others)

> *smokeCount*
> semaphores for table, agent, and smokers[3];
> // *note*, logically, we will use: 0 → tobacco and paper
> // 1 → matches and tobacco
> // 2 → matches and paper

The ***main()*** function should perform the following actions:

- Get and validate smoke count
  - must be between 3 and 10 (inclusive)
- Initialize semaphores
  - table initialized to 1
  - all three smokers and agent initialized to 0
- Create one agent thread
- Create three smokers threads
- Wait for the appropriate threads to complete.

The ***agentThdFunc()*** function will perform the following actions:

```
P(table);
randNum = rand(0, 2);       // pick a random number
display appropriate message
V(table);
V(smoker[randNum])          // wake appropriate smoker
P(agent);                   // agent sleeps
```

The ***smokersThdFunc()*** function (one function, started multiple times) should perform the following actions. Below are the actions for one of the smokers (the one with tobacco). The others are similar.

```
Display starting message (see example)
P(smoker_tobacco);          // sleep right away
P(table);
// pick up match & paper
//  smoke (but don't inhale), so sleep random amount of time
usleep(rand()%1500000)
// display appropriate message in appropriate color
V(table);
V(agent);
```

In addition to the basic synchronization outlined, you will need to update the above algorithm to count the number of times each smoker has smoked, and terminate the thread when the smoke count has been reached. When each smoker completes the appropriate number of smokes, it should display a message (e.g., `"Smoker 1 dies of cancer.\n"`) and terminate that thread.
The following is an example of how to display a message in a color.

```
printf("\033[0;31mSmoker %d completed smoking\033[0m\n", 0);      // red
printf("\033[0;32mSmoker %d completed smoking\033[0m\n", 1);      // green
printf("\033[0;34mSmoker %d completed smoking\033[0m\n", 2);      // blue
```

Analysis Question:

Once the project is complete, answer the following question.

- Assume that after the first smoker "dies of cancer" (i.e., exits), assume that one (or more) of the other smokers (threads) decide to "quit smoking" and thus does not complete. Describe the impact of smokers not completing their execution on the agent and the program.
- Considering the analogy of agent representing the OS and smokers representing processes, when a process has not terminated (i.e., smoker decides to quit smoking), what is the state of the process? If the process continues to reside in the state forever, what is the impact on the system?

Submission

When complete, submit:

- A copy of the **C** (not C++) source file by class time.
- Copy of the program output (pdf format) from the script file (above)
- For 25 bonus points
  - Submit a write-up (PDF format) answering the analysis questions.
  - *Note*, overly long explanations will not be scored.

Submissions received after the due date/time will not be accepted.

Example Execution
The following is an example execution for reference.

```
ed-vm% ./smoker -s 5
Smoker 0 starts...
agent produced matches & tobacco
Smoker 1 starts...
Smoker 2 starts...
Smoker 1 completed smoking
agent produced tobacco & paper
Smoker 0 completed smoking
agent produced matches & paper
Smoker 2 completed smoking
agent produced matches & tobacco
Smoker 1 completed smoking
agent produced tobacco & paper
Smoker 0 completed smoking
agent produced matches & paper
Smoker 2 completed smoking
agent produced matches & paper
Smoker 2 completed smoking
agent produced matches & paper
Smoker 2 completed smoking
agent produced tobacco & paper
Smoker 0 completed smoking
agent produced matches & tobacco
Smoker 1 completed smoking
agent produced matches & paper
Smoker 2 completed smoking
Smoker 2 dies of cancer.
agent produced tobacco & paper
Smoker 0 completed smoking
agent produced matches & tobacco
Smoker 1 completed smoking
agent produced matches & tobacco
Smoker 1 completed smoking
Smoker 1 dies of cancer.
agent produced tobacco & paper
Smoker 0 completed smoking
Smoker 0 dies of cancer.
ed-vm%
```

Note, the order of message will vary between each execution. However, with a smoke count of 5, there will always be a “completed smoking” messages for each smoker.