1. When a process executes the acquire() operation on a semaphore, the integer value associated with the semaphore is decremented. If the resulting value is greater than zero, the process continues immediately. If the resulting value is zero, the calling process is put to sleep in a queue associated with the semaphore. Conversely, when a process executes the release() operation on a semaphore, if its value is greater than zero, the value is incremented and the process continues immediately. If the semaphore's value is zero, then there must be a process waiting in the semaphore's queue. The effect of the release(), in this case, in addition to incrementing the semaphore's value, is to unblock one of the waiting processes. Once a semaphore operation starts, no other process can access the semaphore until the operation has completed. We have discussed two kinds of semaphores: binary semaphores, which have a minimum value of 0 and a maximum value of 1, and counting semaphores, which can range over an unrestricted domain of integer values. Assuming you have at your disposal only binary semaphores and ordinary machine instructions, show how counting semaphores can be implemented. Be sure to list all data structures you will need in your implementation together with the initial values that allow your solution to work correctly.

2. The Sleeping Barber Problem. A barber shop has one barber, one barber chair, and n chairs on which any waiting customers sit. When there are no customers present, the barber sits down on his barber chair and sleeps. When a customer arrives, he has to wake up the barber. If additional customers arrive while the barber is busy cutting someone's hair, they either sit down (if there are empty chairs) or go away (if the chairs are all full).

Using semaphores, design a solution to the Sleeping Barber Problem that does not allow race conditions. You should start by thinking about what are the individual goals you need to achieve in this solution. This should lead you to figure out the number and the types of semaphores and additional variables you’re solution will use. Be sure to list all data structures you will need in your implementation together with the initial values that allow your solution to work correctly. Your solution should consist of two kinds of processes that use these semaphores and variables: one barber process and several customer processes.

3. The Cigarette-Smokers Problem. Consider a system with three smoker processes and one agent process. Each smoker continuously rolls a cigarette and then smokes it. In order to be able to roll and smoke a cigarette, a smoker needs tobacco, paper, and matches. One of the smoker processes has paper, another has tobacco, and the third has matches. The agent has an infinite supply of all three materials. The agent places two of the ingredients on the table. The smoker who has the remaining ingredient then makes and smokes a cigarette, signaling the agent on completion. The agent then puts out another two of the three ingredients (chosen at random), and the cycle repeats.

Using semaphores, design a solution to the Cigarette-Smokers Problem that does not allow race conditions. You should start by thinking about what are the individual goals you need to achieve in this solution. This should lead you to figure out the number and the types of semaphores and additional variables you’re solution will use. Be sure to list all data structures you will need in your implementation together with the initial values that allow your solution to work correctly. Your solution should consist of two kinds of processes that use these semaphores and variables: one agent process and three smoker processes.