

Problem Set #3.**Due: February 17 before class.**

Solve the following synchronization problems by providing pseudo-code. Your pseudo-code should be clear and state the meaning of all variables and their initial values. Use semaphores and, where necessary, condition variables in your solutions. Consulting the *Little Book of Semaphores* (and, of course, the textbook and class notes) is permitted, but no other sources (such as colleagues, the Web, etc). All sources of inspiration, if any, should be accurately cited.

1. There is a deep canyon somewhere in Kruger National Park, South Africa, and a single rope that spans the canyon. Baboons can cross the canyon by swinging hand-over-hand on the rope, but if two baboons going in opposite directions meet in the middle, they will fight and drop to their deaths. Furthermore, the rope is only strong enough to hold 5 baboons. If there are more baboons on the rope at the same time, it will break. Assuming that we can teach the baboons to use semaphores, we would like to design a synchronization scheme with the following properties:
 - Once a baboon has begun to cross, it is guaranteed to get to the other side without running into a baboon going the other way.
 - There are never more than 5 baboons on the rope.
 - A continuing stream of baboons crossing in one direction should not bar baboons going the other way indefinitely (no starvation).
2. Three professors have gone to the faculty club to eat candy. Each piece of candy costs 36 cents, that can only be paid with a quarter, a dime, and a penny (the faculty club does not give change and does not take credit cards). The first professor has a pocket full of pennies, the second a supply of quarters, and the third a supply of dimes. A wealthy alum walks up to the bar and lays down at random two of the three coins needed for a piece of candy. The professor with the third coin takes the money and buys the candy. The cycle then repeats. Show how to synchronize the professors and the alum.
3. A group of students studying for the OS exam can study only while eating pizza. Each student executes the following loop: *while (true) { pick up a piece of pizza; study while eating the pizza}*. If a student finds that the pizza is gone, the student goes to sleep until another pizza arrives. The first student to discover that the group is out of pizza phones Five Star Pizza to order another pizza before going to sleep. Each pizza has *S* slices.

Write code to synchronize the student threads and the pizza delivery thread. Your solution should avoid deadlock and phone Five Star Pizza (i.e., wake up the delivery thread) exactly once each time a pizza is exhausted. No slice of pizza may be consumed by more than one student.

4. The Deli at the Publix serves customers FIFO in the following way. Each customer willing to buy deli takes a single "ticket" with a number from a "sequencer" on the

counter. The ticket numbers dispensed by the sequencer are guaranteed to be unique and sequentially increasing. When a deli assistant is ready to serve the next customer, it posts on a digital board the next highest unserved number previously dispensed by the sequencer. Each customer waits until the board shows the number on its ticket. Each deli assistant waits until the customer with the ticket it called places an order.

Show how to synchronize the deli assistant and customer threads.

5. Once class is over, professor Johnson likes to sleep if there are no students waiting at his door to ask questions. Write the procedures to synchronize threads representing one professor and an arbitrary number of students.

Professor Johnson checks to see if a student is waiting outside the office by calling a procedure *IdleProf()*. *IdleProf* sleeps if there are no students waiting, otherwise it signals one student to enter the office, and returns. A student with a question to ask calls *ArrivingStudent()*, which joins the queue of students waiting outside the office for a signal from the professor; if no students are waiting, then the student wakes up the sleeping professor. The professor and exactly one student will return from their respective functions “at the same time” to discuss a topic of mutual interest. After the discussion is over, the student goes back to studying and the professor calls *IdleProf* again.

- a) Implement *IdleProf* and *ArrivingStudent* using mutexes and condition variables. You may assume that mutexes and condition variables are fair (e.g., FIFO).
 - b) Implement *IdleProf* and *ArrivingStudent* using semaphores. You may assume that semaphores implement a FIFO policy for their queues.
6. Show that mutexes and counting semaphores are equivalent.