CS39002: Operating Systems Lab
Spring 2015
Assignment 5 (Semaphore)
Due on: March 16, 2015 (EOD)

Assignment 5a: The lion and jackal problem.

Problem Definition:

There is a national forest where lives two species: the lion and the jackal. In addition there is a single ranger who provides meat in *three* specific 'meat pit' in the forest for both the lion and jackal to feed on. The ranger, therefore, acts as the *producer* and the lions and jackals act as the *consumer.* Use semaphores to synchronise these three sets of processes: the lion, the jackal and the ranger. In the following description, the term 'signal' not necessarily mean 'signal handling', rather points to the wakeup and sleep events based on semaphore.

There exist three meat pits in the jungle of capacity 50 units. In addition, there are *nL* lions, and *nJ* jackals (take as input). Both the lion and jackal visit any one meat pit randomly and eat unit capacity of meat per visit. However, if the lion (or jackal) finds that the meat pit is empty or a jackal (or lion) is already feeding, it checks the next meat pit. If all the meat pits are unavailable (either empty or a different species is feeding on it), the animals block. Each lion and jackal iterates a fixed number of times before terminating (take as input). From a given meat pile more than one lions or jackals can eat simultaneously, but the same species must be eating in one meat pile at a time.
The ranger piles meat of amount 10 in a random meat pit if it finds it empty/having space and no animal is eating.

The algorithm of the action of each of the three processes is now presented below

The lion:

1. Each lion process sleeps and eats alternately.
2. When trying to eat, it generates a random number between 1, 2 and 3. Say the number is n
3. The lion process tries to eat from meat pit n.
4. If the available food of meat pit n is >0 and there is no jackal, it starts eating from it (i.e. it gains control over the semaphore of n). Note that there may be other lion processes eating from pit n. Once a lion visits a pit, reduce the food amount of the pit n by 1. After eating, lion falls asleep.
5. If there is no meat in the pit n (food amount is now 0), or atleast one jackal process is in control of the semaphore of the meat pit (that is, jackals are eating from the meat pit), the lion tries to eat from meat pit (n+1)%4. Same happens if the ranger is in control of the pit n.
6. The lion checks all the meat pits in a daisy chain method. (If it had generated random number 2, it checks 2, 3 then 1). If it can gain access to any one of these pits (by point 4), it reduces the pit's current content by 1.
7. If all the pits are unavailable, it blocks in the wait queue of the last meat pit it checks. (If lion had started from 2, it blocks at wait queue of 1, following the sequence 2, 3, 1)

8. After a lion has finished eating, it checks to see if there are any more lions feeding in the meat pit. If there is atleast one lion or if the capacity of the pit is zero, the process quits. Else, it sends a signal to the wait queue of all the meat pits such that the jackals waiting can come to this meat pit to eat (same applies to any sleeping ranger). The signal sent to the meat pits should therefore contain a flag to indicate which meat pit is free such that the blocked jackals can go to that pit.

The jackal:

The jackal works in the exactly similar manner as the lion. Read the lion's algorithm stated above for the jackal by replacing the term lion with jackal and vice versa.

The Ranger:

1. The ranger generates a random number between 1, 2 and 3. Say the number is n.
2. It then tries to put food of amount 10 units in meat pit *n*.
3. If either lions or jackals are eating from pit n, or capacity of pit n is full, it tries pit (n+1)%4. Note that if a pit's capacity is >40, the ranger will not put meat into it.
4. The ranger tries all the meat pits, if none are free or empty or having of units <=40, the ranger sleeps on the last checked pile.
5. If a ranger puts food in pit n, it sends signal to wait queue of all three meat pits. Note that in this case, both lions and jackals may be in the wait queue. Design your program such that if a lion had requested control over that pit before the jackal, then they get control over it and vice versa.

Assignment 5b: The Sleeping barber problem

This classical semaphore problem takes place in a barber shop. The shop has one barber, one barber chair, and n chairs for waiting customers, if any, to sit on. If there are no customers present, the barber sits down in the barber chair and falls asleep, as illustrated in the Figure. When a customer arrives, he has to wake up the sleeping barber. If additional customers arrive while the barber is cutting a customer's hair, they either sit down (if there are empty chairs) or leave the shop (if all chairs are full). The problem is to program the barber and the customers without getting into race conditions.

Implementation:

You can use three semaphores:

a) customer, which counts waiting customers (excluding the customer in the barber chair, who is not waiting),
b) barber, the number of barbers (0 or 1) who are idle, waiting for customers, and
c) mutex, which is used for mutual exclusion.

In the solution, a customer entering the shop has to count the number of waiting customers. If it is less than the number of chairs (take as input), he stays; otherwise, he leaves. When the barber opens the shop in the morning, he executes the procedure barber, causing him to block on the semaphore customers because it is initially 0. The barber then goes to sleep. He stays asleep until the first customer shows up.

When a customer arrives, Barber serves the *customer*, starting by acquiring *mutex* to enter a critical region. If another customer enters shortly thereafter, he will not be able to do anything until the first one

has released *mutex*. The customer then checks to see if the number of waiting customers is less than the number of chairs. If not, he releases *mutex* and leaves without a haircut.

If there is an available chair, the customer increments an integer variable, *waiting*. Then he does an Up on the semaphore customers, thus waking up the barber. When the customer releases mutex, the barber grabs it, does some housekeeping, and begins the haircut.
When the haircut is over, the customer exits the procedure and leaves the shop. Nobody is allowed to cut more than once.