CS414 SP 2007 Assignment 2

Due Feb. 19 at 11:59pm Submit your assignment using CMS

1. Explain what goes wrong in the following variation of Peterson's algorithm:

```
Process P_i:
         do {
               flag[i] = TRUE;
               turn = i;
               while (flag[j] && turn == j);

               <critical section>

               flag[i] = FALSE;

               <remainder section>

         } while (TRUE);
```

2. Three processes are involved in printing a file (pictured below). Process A reads the file data from the disk to Buffer 1, Process B copies the data from Buffer 1 to Buffer 2, finally Process C takes the data from Buffer 2 and print it.

```
 Process A          -------------          Process B                ------------        Process C
--------------------->|  Buffer 1 |---------------------------------->|  Buffer 2 |----------------------------->
Read from File       -------------            copy                  -------------          print
```

Assume all three processes operate on one (file) record at a time, both buffers' capacity are one record. Write a program to coordinate the three processes using semaphores.

3. Some monkeys are trying to cross a ravine. A single rope traverses the ravine, and monkeys can cross hand-over-hand. Up to five monkeys can be hanging on the rope at any one time. if there are more than five, then the rope will break and they will all die. Also, if eastward-moving monkeys encounter westward moving monkeys, all will fall off and die.

Each monkey operates in a separate thread, which executes the code below:

```
typedef enum {EAST, WEST} Destination ;

void monkey(int id, Destination dest) {
        WaitUntilSafeToCross(dest);
        CrossRavine(id, dest);
        DoneWithCrossing(dest);
}
```

Variable id holds a unique number identifying that monkey. CrossRavine(int monkeyid, Destination d) is a synchronous call provided to you, and it returns when the calling monkey has safely reached its destination. **Use semaphores to implement WaitUntilSafeToCross(Destination d) and DoneWithCrossing(Destination d).**

(a) For Part (a) You implementation should ensure that :

i) At most 5 monkeys simultaneously execute CrossRavine().
ii) All monkeys executing in CrossRavine() are heading in the same direction.
iii) No monkey should wait unnecessarily. (This way, you can maximize the throughput)

(b) Instead of maximizing the throughput, your solution for Part (b) should ensure there is no starvation (the first two conditions in part (a) still apply). That is, no monkey should wait at the ravine forever. You can assume that semaphore's wait queue is FIFO queue.

(c) Now instead of monkeys, we have students crossing the ravine. Because students are much smarter than monkeys, assume that students heading different directions **CAN** be on the rope at same time. Write a semaphore implementation that ensures the following requirements.

i) At most M eastward-moving students simultaneously execute CrossRavine()
ii) At most N westward-moving students simultaneously execute CrossRavine()
iii) At most K students simultaneously execute CrossRavine()
iv) you solution should maximize the throughput under above conditions.

.4. A commonly used group synchronization mechanism is called **barrier**. Some computations are divided into phases and have the rule that no thread may proceed into the next phase until all threads are ready to proceed to the next phase. The behavior may be achieved by letting each thread execute **barrier.done** at the end of each phase. Suppose there are N threads in the computation, a call to barrier.done blocks until all of the N threads have called barrier.done. Then all thread proceed. There could be multiple phases requiring synchronization Here is implementation of barrier.done.

```
Semaphore mutex = 1;
semaphore barrier = 0;
int count = 0;

void barrier.done() {
      wait(mutex);
      count++;
      if (count < N ) {
            signal(mutex);
            wait(barrier);
      }
      else {
            signal(mutex);
            count = 0;
            for (int i = 1; i < N; i++)
                  signal(barrier);
      }
}
```

Explain why above solution might not work. (Hint: your solution should not depend on the implementation of the semaphore's wait queue)

5.
(a) Suppose we replace the wait() and signal() operations of monitors with a single construct await(B), where B is a general Boolean expression that causes the process executing it to wait until B become true. Explain why, in general, this construct cannot be implemented efficiently.

(b) The textbook gives a monitor solution to the dining-philosopher problem (on page 214, Figure 6.19). One problem with this solution is that it is possible for a philosopher to starve to death. Illustrate why?