**CS350 Semester I, 2015-16**

**Homework 3**
**Due Date: October 29, 2015**

1 . Non-preemptive multitasking : extend the oz kernel language in assignment 2 with the statement

`thread` ⟨`s`⟩ `end`

which works as follows. You should extend the semantic stack in Assignment 2 to a semantic multistack. Each stack in the multistack will correspond to a thread in the code.

If the top statement in the currently active semantic stack is `(thread` ⟨`s`⟩ `end,E)`, then create a new semantic stack and push `(`⟨`s`⟩`,E)` onto it.

There will be no thread scheduler. The execution continues with the previously active stack until it becomes empty.

Once a stack completes (or suspends), you should pick another non-empty stack and continue execution until it completes and so on. If there is no non-empty stack, or if every stack is suspended, then the execution completes. [25 points]

(All other questions carry 10 points each.)

2 . Write a program with two threads - the first thread produces a stream of random bits using `OS.rand` or `OS.randLimits` and stores it in a stream, say `Xs`. The second should produce a stream `Ys` whose $n^{\text{th}}$ element should be the average of the first $n$ bits of `Xs`. `Ys` should be written as a self-referential stream.

3 . Write a lazy version of append which can concatenate two finite lists.

4 . Write a lazy version of QuickSort to sort a finite list of integers. You might want to write a lazy version of Filter.

5 . Write a version of the Hamming Problem which uses threads instead of (lazy) streams.

6 . In the message-passing model, implement a non-deterministic conditional block

```
NSelect [X1#S1 X2#S2 ...  Xn#Sn true#Sn+1]
```

which acts as follows: if any of X1, ..., Xn is true, then non-deterministically select one, say Xi, and execute the corresponding statement Si. (Automatically, if all of them are false, Sn+1 will be executed). However, you must also ensure the following: if all of X1, ..., Xn are unbound, then the execution should suspend until at least one of them gets bound (to true or false).

7 . Write a procedure

```
Barrier [P1 P2 ...  Pn]
```

which takes a list of procedures, and executes each procedure in a thread by itself. The main thread should wait until all the procedures have finished, and only then exit.