CSc 422/522 — Homework 1, Part A

due Tuesday, February 1

The purposes of this assignment are to introduce you to SR and to the nature of multithreaded concurrent programs. It is not likely that you would *choose* to write a concurrent program to solve this problem. However, it is possible to do so and should be instructive.

This programming exercise is worth 20 points. Turn in commented listings of your four programs (paper copies only) *and* a brief description of the tests you ran and the output you observed. We are not going to use electronic turnin for this assignment.

Your programs should all be about one page in length. They will primarily be graded for correctness, but you should also use good programming style. See the attached information on Presentation Points, which is also on the class Web page. In addition, do not use excessive indentation; four spaces is plenty and I myself prefer two or three, as in the textbook.

You are to write four programs to solve the following simple problem: Given two input files, output a *perfect shuffle* of the two files. A perfect shuffle is an interleaving that contains the first line from input file 1, then the first line from input file 2, then the second line from input file 1, then the second line from input file 2, and so on. If one file is longer than the other, append the extra lines to the end of the output. If `shuffle` is the name of your executable file, then the program should be invoked with two command-line arguments as:

```
shuffle filename1 filename2
```

The program should write to standard output.

You are to implement *four* versions of a shuffle program:

- First write a sequential program.
- Then modify your program to use the "`co` inside `while`" style described in Section 2.2 of the textbook. The three independent activities are reading from `filename1`, reading from `filename2`, and writing to standard output. You will need to use what is called *double buffering* for each input file—i.e., read and write different buffers, then swap their roles.
- Next modify your sequential program to use the "`while` inside `co`" style. Use SR's `process` declaration and again use double buffering. Do *not* synchronize the actions of the processes. Just let them be independent and free running. What happens? Why?
- Finally, add synchronization code to your third program so that it is correct. Use counters and the busy-waiting style of synchronization described in Sections 2.2 and 2.5 of the text.

The SR implementation by default lets a process run until it blocks, then it executes another process, and so on. This makes execution pretty deterministic on a single processor. You can force SR to reschedule a process—and hence get a better simulation of true concurrency—in either of two ways. One is put calls of `nap(0)` in the bodies of loops. The second is to use the `-L` option with the SR linker (`srl`). In particular, if `shuffle` is the name of your main (only) resource, first compile your program, then execute "`srl -L 1 shuffle`". The executable now resides in `a.out`.