Class 4: Once Upon a Process

Action Items

Assignments due: **Problem Set 1** is due on Tuesday, September 10 (11:59pm). Note: given the instructor's belief that sensible countries should have national holidays when they qualify for world cups and that the University should observe national holidays, though, in the event that the US qualifies for World Cup 2014 before 11:59pm tonight, the assignment will be accepted without penalty until 11:59pm tomorrow. Before assuming you can safely rely on this, though, you should carefully consider the US team's injury and yellow card situation, and the likelihood that you'll have to deal with a grumpy instructor if you are asking for an extension under the scenario where the team fails to win tonight.

Problem Set 2 will be posted by Thursday and due on 24 September. For PS2, you are **required to work with a partner** (you can choose on your own), so try to find a partner you want to work with now. To help find a partner, you can use the Piazza teammates forum.

Processes

What is a process?

Some terminology (not really important, but you'll hear people use them):

- **Multiprogramming** - program gets to run until it gets stuck, then supervisor takes over and selects another program to run.
- **Non-pre-emptive multi-tasking** (sometimes called *co-operative multi-tasking* - program gets to run until it voluntarily gives control back to the supervisor. (Sometimes used to mean the programs are all scheduled statically to recieve a particular processing time slice.)
- Normal (pre-emptive) **multi-tasking** - program gets to run until (approximately) supervisor decides its someone else's turn.

(For historical reasons, the terms and hyphenating-conventions are confusing, using "program", "task", and "process" to essentially interchangably, except that the historical term *multiprogramming* is different from *multitasking*.)

What are some consequences of the difference between *non pre-emptive multi-tasking* and *pre-emptive multi-tasking*?

How often should the kernel timer interrupt (a.k.a., "supervisor's alarm clock") go off?

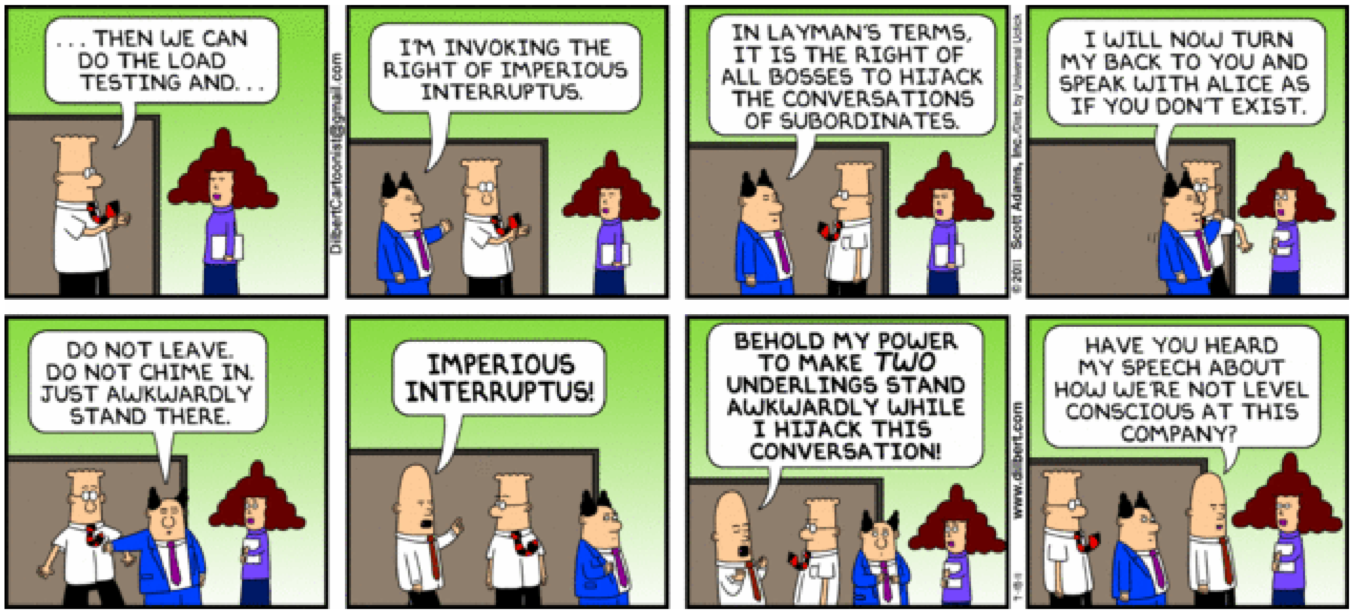
. . . THEN WE CAN DO THE LOAD TESTING AND. . .
I'M INVOKING THE RIGHT OF IMPERIOUS INTERRUPTUS.
IN LAYMAN'S TERMS, IT IS THE RIGHT OF ALL BOSSES TO HIJACK THE CONVERSATIONS OF SUBORDINATES.
I WILL NOW TURN MY BACK TO YOU AND SPEAK WITH ALICE AS IF YOU DON'T EXIST.
DO NOT LEAVE. DO NOT CHIME IN. JUST AWKWARDLY STAND THERE.
IMPERIOUS INTERRUPTUS!
BEHOLD MY POWER TO MAKE TWO UNDERLINGS STAND AWKWARDLY WHILE I HIJACK THIS CONVERSATION!
HAVE YOU HEARD MY SPEECH ABOUT HOW WE'RE NOT LEVEL CONSCIOUS AT THIS COMPANY?
DilbertCartoonist@gmail.com
© 2011 Scott Adams, Inc./Dist. by Universal Uclick
www.dilbert.com

Figure 1: Imperious Interruptus

What programs should be able to change the kernel timer interrupt frequency?

Who prefers the kernel timer interrupt interval to be shorter? Who prefers it to be longer?

Links

I'm not certain this is completely reliable, but here's what I used to determine the kernel timer frequency for my MacBook Air: How To determine Linux Kernel Timer Interrupt Frequency

```
:::shell
$ gcc timer.c ; ./a.out
kernel timer interrupt frequency is approx. 4016 Hz or higher
```

It is worth a **midterm exemption** (and Rust sticker!) for the first person (or team of two people) who writes an accurate kernel timer in Rust.