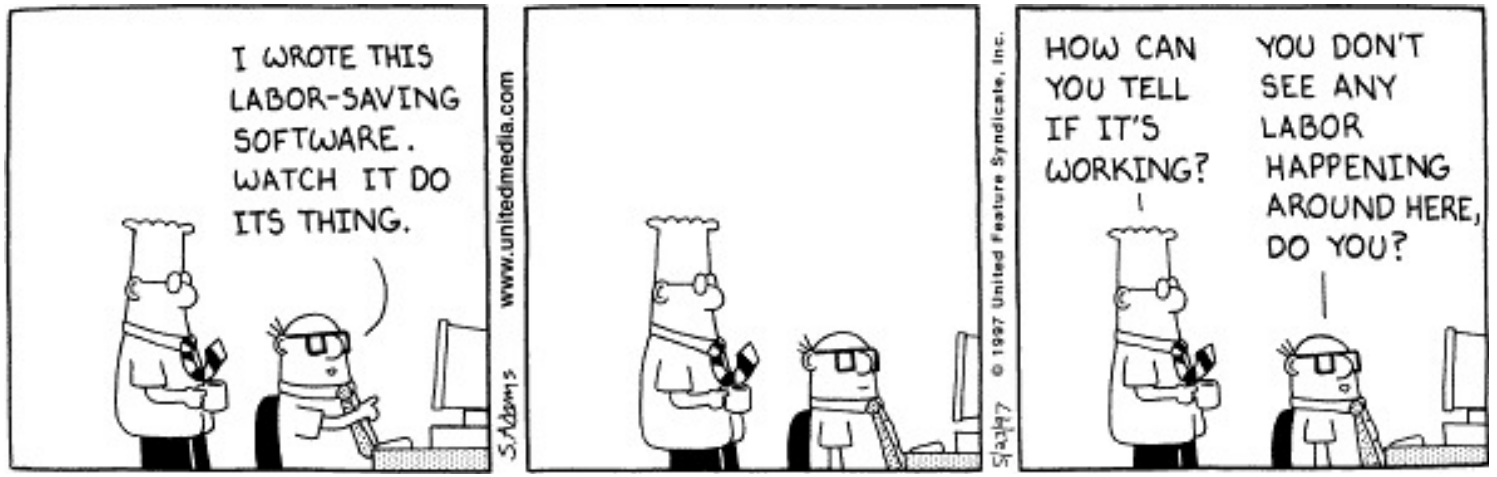
Part 1

Week 6: Processes

CS211: Programming and Operating Systems

Thursday, 18 March 2021



Usual reminders...

	Mon	Tue	Wed	Thu	Fri
09:00					
10:00					
11:00					
12:00					
13:00				***Recorded***	
14:00					
15:00	**LAB**		***Recorded***		
16:00	**LAB**				

1. This week, we have just one recorded classes: Thursday at 13:00.
2. The lab next Monday (22 March) will be a continuation of the one that started this week.
3. An introduction to the lab wa srecorded, and is now available.

Feedback on Feedback

- Thank you to the 8 of you that completed the feedback form circulated by Noelle Gannon.
- On average, it took 6 minutes, 33 seconds to complete.
- Mostly very positive.
- Several people are “unsure” or “disagree somewhat” with the statement that “The feedback I have received is helping me to improve my learning”. Which is fair! (Will do better!).
- The “live-but-recorded” lectures seem to be popular (which I was unsure of, since the quality is not very high).
- Some good suggestions for improvement:
  - ***“Exam style questions with some worked solutions near the end of the module”***. [Response: will post last year’s exam, and solutions.]
  - ***“... it would be great if the lecturer did more examples step by step”. “ more time explaining and repeating the basics and the syntax at the beginning would help”. “Would be really helpful is if we could see the program being written and run during the lecture!”*** [Response: very helpful – will try to do this]

Feedback on Feedback

- ***"Getting worked examples of the assignments after they've been submitted would be really helpful too"***. [Response: Yes! Will do this once I get the assignments graded].
- ***"I think it would be better to have the homework posted a few days before the live lab session"***. [Response: Good suggestion, and I hope the new approach of having an assignment running over two weeks will help. But I can't promise more, since I'm already stretched getting it posted the night before the 3pm lab.]

This week in CS211:

1. Part 1: The Process
   - Process API
   - Process State
2. Part 2: Process Creation
   - Example 1: `fork()`
   - Example 2: `02Fork2.c`
   - Example 3: `getppid()`
3. Exercises

This week of "Programming and Operating Systems", we segue from **Programming** to **Operating Systems**, starting with the concept of a **process** (OS). But we will write C programs that manipulate processes (Programming + OS).

CS211
Week 6: Processes

Start of ...

PART 1: The Process

Part 1: The Process

As we now move towards the “Operating System” part of the course, the need to learn some classical OS Theory. The presentation given here is quite standard, and you should find equivalent descriptions in any OS text-book.

Material from this point one relates to Chapters 4 and 5 of Operating Systems: Three Easy Pieces by Remzi H. Arpaci-Dusseau and Andrea C. Arpaci-Dusseau:

Processes: `http://pages.cs.wisc.edu/~remzi/OSTEP/cpu-intro.pdf`

Process API: `http://pages.cs.wisc.edu/~remzi/OSTEP/cpu-api.pdf`

"A Process... is a running programme" (OSTEP, p25)

Most OS will give the impression that many programmes are running at one time. The user/programmer does not know or care of the CPU is currently busy: the OS gives them the impression that it is available for their (exclusive) use.

This is made possible by abstracting the concept of a running program as a **process**.

"*The OS creates this illusion by virtualizing the CPU*": we will study, later, how this scheduling achieved. For now, we will take it that we need the concept of the **process** to do this.

Every process consists of:

- the ***Process Text*** - the code of the program
- the ***program counter*** – the address of the next instruction to be executed.
- the ***process stack*** (temporary data, e.g., local variables, return addresses, etc)
- the ***data section*** – global variables.

A process is ***not*** (just) a program: if two users run the same program at the same time they create different processes.
A program is a ***passive*** entity, whereas a process is ***active/dynamic***.

Often, the terms *process* and *job* can be used interchangeably.

Here is a minimal set of operations that an OS must be able to apply to a process.

Create a new process, e.g., when you click on an icon.

Destroy (or terminate) a process,

Wait that is **pause** the process until some other event occurs.

Suspend and resume: like wait, but invoked more explicitly.

Status report: information about a process, such as how long it has run for, how much memory has been allocated to it, etc.

The **state** of a process is defined (in part) by the current activity of that process:

new: The process is being created

running: Instructions are being executed

blocked: (also called "waiting"). The process is waiting for some event to occur

ready: The process is waiting to be assigned to a processor

terminated: The process has finished execution.

Here is a diagram of the process life-cycle, featuring

new • **running** • **blocked** • **ready** • **terminated**