CSE 451: Operating Systems
Autumn 2011

Module 0
Course Introduction

John Zahorjan
zahorjan@cs.washington.edu
534 Allen Center

© 2011 Gribble, Lazowska, Levy, Zahorjan

Today's agenda

- Mechanics
  - course overview
  - course staff
  - general structure
  - the text
  - policies
  - your to-do list
  -
- Review of key 351 material
  - processor operation
  - exceptions/interrupts
  - virtual memory
- What does an OS do?

© 2011 Gribble, Lazowska, Levy, Zahorjan

But first, a note from our laywers

- Some of the slides for this course are adapted from material supplied (and copyrighted) by Tom Doeppner (author of the course text)
- Some of the slides for this course are adapted from material supplied (and copyrighted) by various UW CSE 451 instructors

And now a note from me...

- I prefer the white board to slides
- You'll prefer my using the white board to my using slides
- I'll mostly be using the white board
- Lecture resources you'll have:
  - The course text
  - Lecture slides (that I mostly won't have shown)
  - Your notes and recollections of the lectures
  - My pictures of the white board just before erasing

Course Mechanics Overview

- Everything you need to know will be on the course web page:

http://www.cs.washington.edu/451/

Stuff You Can Find On The Course Web

- Course staff
  - John Zahorjan
  - James Athappilly
  - Owen Kim
- General Course Structure
  - Read the text prior to class
  - Class doesn't aim to repeat the text
  - Homework exercises...
  - Sections are likely to focus on projects
  - You're paying for interaction
    - Plus the degree...

The Textbook

- Tom Doeppner, *Operating Systems in Depth (2011)*
- This is a change from all prior offerings of 451...
  - This text is more firmly rooted in how things are actually done than previously available texts
  - It explains topics that are important to fully understanding what a running program actually is and can do that were ignored in past texts
  - It is written assuming a programming mindset that is further from CSE 1XX … CSE 3XX programming experiences than past texts

Policies and To-do List

- Collaboration vs. cheating
- Homework exercises: late policy
- Projects: late policy

- Please read the entire course web thoroughly
- Keep up with the reading in the text
  - Use mail, the course discussion board, office hours, etc. for questions
- Homework 0 (it might be fun!) is posted on the web **now**
  - due by **the start of class** next Monday
  - online turnin
- Project 0 is also on the web now
  - due a week from Friday, end of day

Course Pre-reqs

- This year: CSE 351 (CSE 378); CSE 332 (CSE 326)
- Next year: += CSE 333 (CSE 303)

- C programming experience
  - 20 of you have taken 303
  - 19 have taken 333
  - 15 have taken neither
  - (Some of you took 351 from me, some from "those other guys")

- Concurrent/parallel programming experience
  - I don't know how many of you have had 332 (vs. 326), but...
  - It's a second order effect anyway, based on what I can tell about the differences in material covered in 332 from quarter to quarter

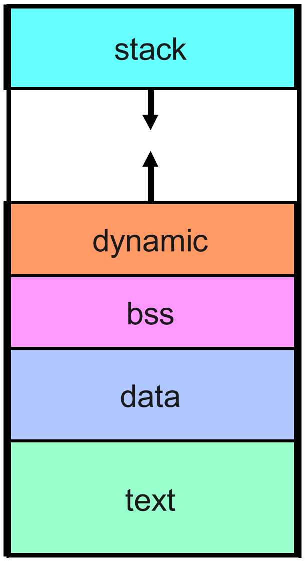
Essential Concepts From CSE 351

- The processor is a (deterministic) state machine
  - `add   r2, r3`
    - State after execution (in this case the contents of r3) is determined by the state before execution (contents of r2 and r3)
    - It doesn't matter how long it has been since the last instruction was executed

- Implication
  - An executing program can be stopped and restarted at any time, so long as the state it can see when resumed is the same when stopped as it would have been if it hadn't been stopped
    - the state isn't changed while the program isn't running, or
    - the state is saved and then restored

Any given running program can see only some of the full machine state

- It can see the state of the processor
  - Basically, the contents of registers
- It can see the state of the memory it owns
- To stop/resume an execution, it's sufficient to preserve register values and memory contents
  - Have to save everything because we're doing this at runtime



Preserving register contents

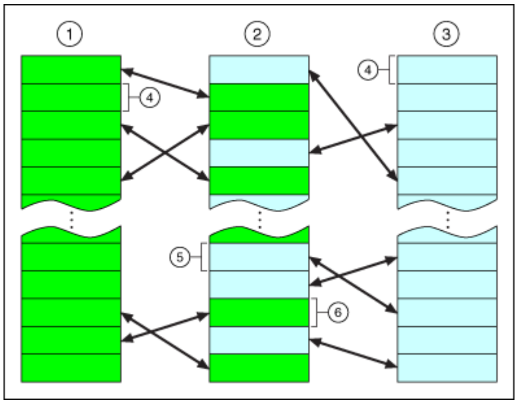
- Problem: whatever runs while the execution is stopped has to use the registers
  - Which means that thing changes the processor state
- No problem: just save the register contents before doing anything else, and restore them just before resuming the execution
- (This is pretty much exactly how procedure call works
  - Caller and callee together ensure that the caller can ignore the fact that the callee has used the processor by saving/restoring registers onto/from the stack)
- Just where are the registers saved?
  - Stay tuned to this course...

Preserving memory contents

- No problem
  - Other applications in execution can't alter the contents of the suspended execution's memory
  - The OS can, but is careful not to
- What keeps other executions from modifying "my memory" while they are running?

Virtual Memory

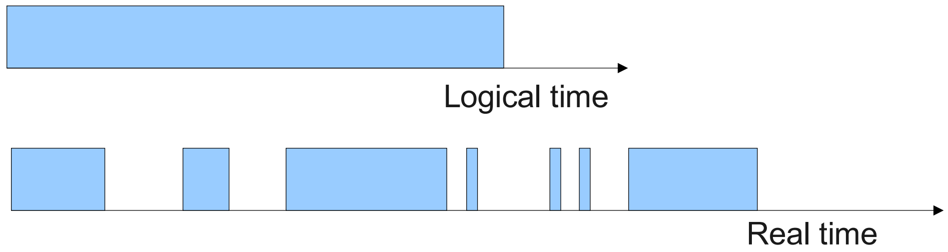
- Page tables map from virtual address to physical address
  - Each running program can modify only that part of real memory pointed at by its own page table
- OS makes sure that program A's page table doesn't point to any real memory also pointed to by program B's page table



1	Process A
2	Physical RAM
3	Process B
4	Page of virtual memory
5	Page of memory from Process B
6	Page of memory from Process A

Summary So Far

- Processor is a state machine, so...
- Each instruction gets deterministic results based solely on state when it is executed, so...
- Can suspend an running program (a sequence of instruction executions) between the completion of one instruction and the beginning of the next so long as the program's state is unaffected when the its execution resumes



Going a bit beyond CSE 351

- Q: Is the hardware providing the logical time abstraction to program executions, or is the operating system doing that?

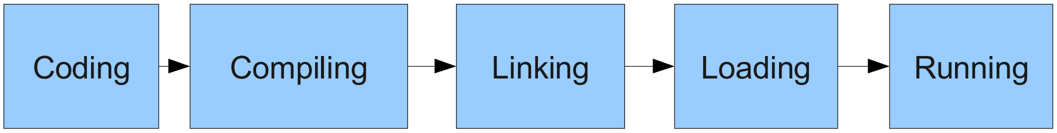
- A:Yes, and yes.
  - The hardware is designed with knowledge of what the OS can do, and the OS with the knowledge of what the hardware can do.
  - “The system” isn't either one in isolation, it's the two together.
  - In fact, “the system” is an even broader notion than that.

“The System”

- The title of this course is “Operating Systems,” but you should be careful not to try to restrict your thinking to just the OS.
- What is the point of the OS?
  - There are many, but one is to make it easy(er) to write and run programs that do something useful
- The OS doesn't do anything, except facilitate getting other things to run
  - The easier it is write programs, the more people who will do it, resulting in more programs being released, resulting in everyone being happier and richer, or both
- So, when you think of “the system” you need to consider everything that goes into creating and running a working program
- Some of that isn't part of the OS, but is intimately linked to it

© 2011 Gribble, Lazowska, Levy, Zahorjan

Programs: Time Dimension

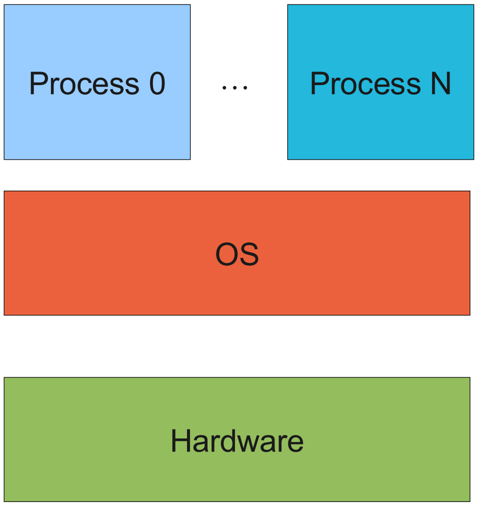


- Programmer codes (mostly) to an abstraction defined by the language
  - That abstraction is designed understanding what the linker, loader, and OS can do
- The linker supports what is needed by the languages, and what is required by the loader
  - Ditto for the loader
- The OS is involved only during run time. However, the entire pipeline is designed with an understanding of what the OS can do (even as far back as coding time), and the OS with some knowledge of what the “layers” above it are trying to do.
- “Stuff” needs to get done to write and execute programs. That stuff happens somewhere in this pipeline. Everything works together.
  - Is the OS the only runtime component provided by “the system”?

© 2011 Gribble, Lazowska, Levy, Zahorjan

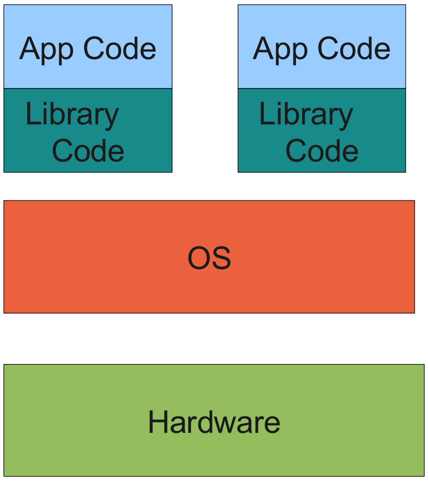
Programs: Space Dimension

- Here is the usual, layered view of executing software:

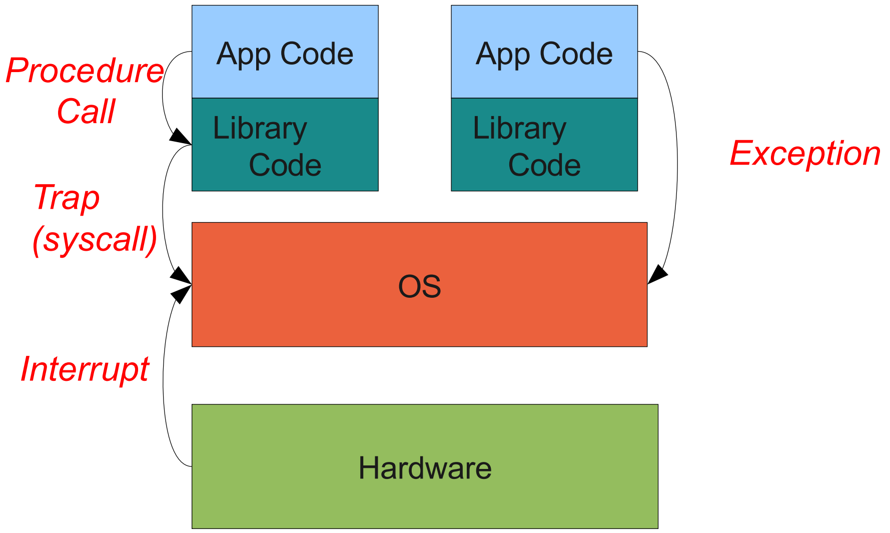


Control Flow and Communication

- There are many components, implemented by many different people
- How does your main() method communicate a request that some code run to:
  - a method foo() that you wrote?
  - a library method bar()?
  - the disk read method in the OS?
- In each case state is saved at call time and restored at return, because that makes life a lot simpler when you're writing the caller
- 1. The details differ, but all of these look pretty much the same
- 2. Control flow is one way to effect communication (i.e., send a request)

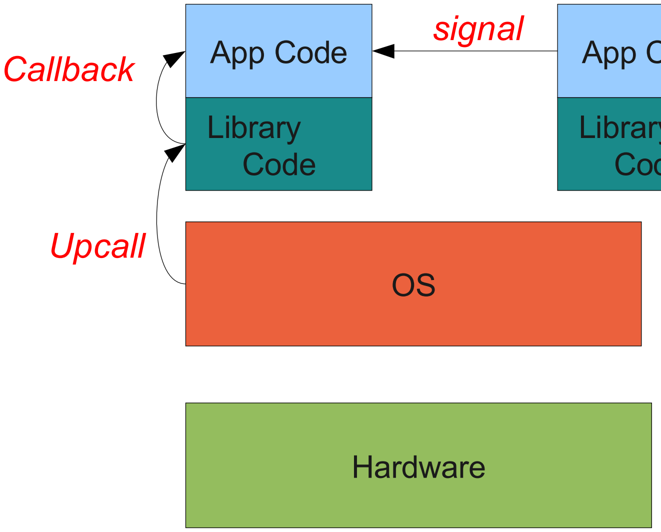


Communication You Know About



© 2011 Gribble, Lazowska, Levy, Zahorjan

Other Communication Paths



· Callee *registers* the address of a method to be run when the caller has something to say

· Caller causes PC to be set to that address (possibly along with some arguments)

© 2011 Gribble, Lazowska, Levy, Zahorjan

Asynchronous vs. Synchronous Communication

- (Almost) All the techniques so far have been *asynchronous* wrt the callee's execution
  - A control flow change "just happens"
  - That can be handy, because the client doesn't have to write any code to indicate when the communication can take place

- There are other approaches, though:
  - blocking: invoke some method with semantics "don't return until a communication has arrived"
    - `line = System.console().readline();`
  - polling: client writes code to ask if any communication has arrived, and executes that code from time to time

Linux vs. Windows

- This course concentrates on fundamental concepts that are incorporated in all operating systems, one way or another
- For specificity, we'll stick mostly to Linux
- The text also uses Linux as the primary concrete example, but discusses Windows a bit as well

- One way the two differ is in how process-to-process communication occurs:
  - Linux: signals (asynchronous control flow transfer)
  - Windows: blocking calls (client reads from message queue)
- Conceptually, the distinction is small. As a practical matter, somewhat subtle side-effects crop up as a result of the decision made

Based on Your Experience....

- What does the operating system do for you?

- What constraints are there on how the operating system provides those things?

© 2011 Gribble, Lazowska, Levy, Zahorjan