csci 297: Advanced Operating Systems

- Professor Gabriel Parmer (aka. Gabe)  
gparmer@gwu.edu, office hours 10am-12 Tuesday (Tentative. Does this work for you?)

- Class: Tompkins 205, Monday 6:10-8:40

Acknowledgements: Some slide material derived from Silberschatz, et al.
Today

1) Administrative information
   • Course requirements
   • Grading

2) What's an OS

3) Research OSes vs. Real-World Oses

4) Meet the Hardware
   • Background for reading this week
Administrative Info

• Paper-based class
  • We will read research papers
  • You will present them
  • We will discuss them

• Semester-long project
  • In-depth implementation study within OS
Grading

- Class participation
- Class presentations
- Final project

...that's it.
Class Participation

1) Class attendance

2) Contributions to the class discussion
   
   ...no zombies in class

   • questions/comments/stories/...
   
   • Always interrupt me

3) Paper summaries

   • 1-3 sentence summary of the purpose of the paper
   
   • Questions: you are not expected to know everything
   
   • What you liked, and what you identified as limitations
   
   • Due night before class (Sunday 11:59pm)
Class Presentations

• 1-2 presentations each
• Read a research paper, and do a 30 minute presentation on it
  • You will be stopped frequently
  • Goal: foster and encourage discussion
• A useful skill
• Read papers for next class on Schedule
Project

- Implementation experience with a real OS
  - Significant contribution
- C
  - You don't have to be an expert, but you have to know your way around (pointers, data-structs,...)
    - And know another language well
- Debugging is the hard part
Project II

- Do you have a systems-y project in mind?
  - Are you already doing work with a systems flavor?

- Double-dipping policy:
  - You can certainly do a project that you are using for another class, for your research, or for entertainment
  - You just have to OK it with me and talk about goals
Project III

• Most of you will *not* have a project in mind
  • Great!

• I have a number of them
  • Most are in the *Composite* OS, a GW-native OS
    – Can come to me with implementation difficulties
  • Linux
    – I can help to some degree
  • Other
    – More than welcome, but I can't guarantee I can help
Project IV

- Possibility for publication
  - But this will be a *lot* more work
  - Don't have to just get something working, but also
    - Evaluate it rigorously
    - Write a paper
Project Timetable

...webpage...
Course Materials

- Course papers and *Composite* virtual machine
  - Too large for blackboard (VM is 1.5G)
- ssh/sftp to <contact me for ip> (write this down)
  - User: <contact me>
  - Password: <contact me>
- VMWare player/workstation
  - I have licenses if needed (e.g. if you use OS X)
What is an Operating System!? 

- What does an OS do?
- What is the purpose of an OS?
Operating System as Abstraction

"The effective exploitation of his powers of abstraction must be regarded as one of the most vital activities of a competent programmer." - Edsger W. Dijkstra

- Allow users to translate intentions into actions
- Provide environment in which applications can execute
  - Each application believes it is sole user of HW
Computers as Distributed Systems

“Hardware: The parts of a computer system that can be kicked.”
- Jeff Pesis
OS as Hardware Manager

- Control a diverse set of hardware
  - Processors
  - Memory
  - Disks
  - Networking cards
  - Video cards
- Coordinates these hardware resources amongst user programs
Policy/Mechanism Separation in Hydra

- **Mechanism**: a utility or resource that can be used in a specific manner
- **Policy**: the algorithm or logic that determines how to use a mechanism
- A policy at one level might be a mechanism for a higher level of *abstraction*
  - Disk → blocks in mem → files → database → ...
  - Sequential exec → threads → scheduling → ...

---

# Policy/Mechanism Separation in Hydra

- **Mechanism**: a utility or resource that can be used in a specific manner
- **Policy**: the algorithm or logic that determines how to use a mechanism
- A policy at one level might be a mechanism for a higher level of *abstraction*
  - Disk → blocks in mem → files → database → ...
  - Sequential exec → threads → scheduling → ...

---
Policy/Mechanism Separation in Hydra

• OSes are concerned with building policy and mechanism
• Create a usable abstraction to achieve a system's goals
AN x64 PROCESSOR IS SCREAMING ALONG AT BILLIONS OF CYCLES PER SECOND TO RUN THE XNU KERNEL, WHICH IS FRANTICALLY WORKING THROUGH ALL THE POSIX-SPECIFIED ABSTRACTION TO CREATE THE DARWIN SYSTEM UNDERLYING OS X, WHICH IN TURN IS STRAINING ITSELF TO RUN FIREFOX AND ITS GECKO RENDERER, WHICH CREATES A FLASH OBJECT WHICH REnders dozens of video frames every second.

BECAUSE I WANTED TO SEE A CAT JUMP INTO A BOX AND FALL OVER.

I AM A GOD.
Fundamental OS Concepts

- Abstraction
- Concurrency
- Parallelism
- Resource management
- Protection
- Performance
  - Kernel doesn't *do* useful work, enables it
Course Topics

- System Structure
- Data Movement
- Accounting
- Concurrency
  - Threading models
- Parallelism
- Reliability
- Security

- Keep in mind any preferences you may have between topics
Research Papers

• We will be reading old and new papers...
  ...about systems...
  ...that noone uses.

• If the proposed systems aren't being used, why do we care
  • Competitive environment: that which is best will prevail, right?
The Rise of “Worse is Better”

- Richard Gabriel – lisp researcher
- 1991
- Lisp vs Unix/C
evalquote[fn;x] = apply[fn;x;NIL]

where

apply[fn;x;a] =
    [atom[fn] ← [eq[fn;CAR] → caar[x];
        eq[fn;CDR] → cadr[x];
        eq[fn;CONS] → cons[car[x];cadr[x]];]
    eq[fn;ATOM] → atom[car[x]];]
    eq[fn;EQ] → eq[car[x];cadr[x]];]
    T → apply[eval[fn;a];x;a];]

    eq[car[fn];LAMBDA] → eval[caddr[fn];pairlis[cadr[fn];x;a]];]
    eq[car[fn];LABEL] → apply[caddr[fn];x;cons[cons[cadr[fn];
        caddr[fn]];a]];]

eval[e;a] = [atom[e] → cdr[assoc[e;a]];]

    atom[car[e]] ←
        [eq[car[e];QUOTE] → cadr[e];
        eq[car[e];COND] → evcon[cdr[e];a];
        T → apply[car[e];evlis[cdr[e];a];a];]
        T → apply[car[e];evlis[cdr[e];a];a];]

pairlis and assoc have been previously defined.

    evcon[c;a] = [eval[caar[c];a] → eval[cadr[c];a];
        T → evcon[cadr[c];a]]

and

evlis[m;a] = [null[m] → NIL;
        T → cons[eval[caar[m];a];evlis[cadr[m];a]]]
Intertia vs. “The right thing”

- Normal OS class: how systems we use work

- This class:
  - Will include some of how current systems work
  - Focus on non-typical design decisions to explore the possibilities of OSes
Intertia vs. “The right thing” II

- Why read these papers?
  - Understanding different design's trade-offs makes us better understand the systems we do use
  - The computing environment changes
    - Hardware – multicore
    - Culture/economics – power/cloud
    - Requires “out of the box” thinking
Blank Slate

• Hardware as a blank slate
  • Where do we go from there?

• Each paper in this class approaches this question differently

• Next slides: what does this blank slate look like?
Basic Hardware: Mechanism

- CPU – sequential execution
- Memory – large array of *physical* memory
- Devices –
  - Receive instructions from CPU...
  - Over a distributed system...
  - To interact with the outside world
Complicating the CPU

- Sequential execution:
  - Stream of instructions are executed
  - Manipulate *registers* and memory
  - Use *stack* for storage/bookkeeping
- Wish to execute multiple applications
  → multiple sequential streams of instructions
- Switch between these *threads*: dispatching mechanism

```c
struct thread *current, *next;
switch_regs(current, next)

switch_regs:
  mov %a, current->regs.a
  ...
  mov %sp, current->regs.sp
  mov post_switch, current->regs.ip
  mov next->regs.a, %a
  ...
  mov next->regs.sp, %sp
  jmp next->regs.ip
post_switch:
  ret
```

Do threads complicate the system as a whole?
Complicating Memory

- Want multiple applications
- Protection – reliability and security
  - Segregate different system parts from each other: Virtual Mem
  - Memory accesses in virtual address space
- Virtual memory mechanism provided by hardware
  - Paging/segmentation/etc...
Complicating Memory II

- System complications:
  - Page-table maintenance
  - Overhead of switching between address spaces:
    \[
    \text{mov pgtbl}_\text{addr}, \%\text{cr3} = 300-800 \text{ cycles on P4}
    \]
  - Hydra – “Given that user-level policy programs must execute in their own protection domains, and that domain switching is costly..., it is impractical to invoke such programs each time a policy decision is required.

    Thus we compromise. We give this compromise a name: the principle of policy/mechanism separation.”
Complicating Devices

• General operations (type of data/device differs):
  • CPU → Device: transfer data @ address $x$ to the device
  • CPU → Device: when you have data ready (?), transfer it to address $y$ in memory
  • Direct Memory Access (DMA)
• How does the CPU know when the device has placed new data at $y$?
Complicating Devices II

- Devices can raise *interrupts* on CPU
  - Halt current stream of instructions
    - Save some register state such as instruction ptr (where?)
  - Begin execution of an interrupt service routine (ISR)
    - Understands how to communicate with device
- Interrupts can happen at any time
  - Except when they are disabled: cli … sti

How do interrupts complicate the system?
Done?

- With these hardware-provided mechanisms
  - Do we have the necessary building blocks for complex systems?
    - With multiple applications

- What else do we need from hardware? Why?
Separation of Privileges

- Can all applications
  - Switch page tables?
  - Switch between any threads?
  - Send commands to devices?
  - Disable interrupts?

- What keeps them from doing this?
Separation of Privileges II

- Dual-mode execution
  - User-mode
    - Applications execute in user-mode in protection domains
    - Cannot execute sensitive instructions
    - Cannot access kernel memory (memory marked with mode)
  - Kernel-mode
    - *Trusted* code
    - Can execute sensitive instructions (cli, sti, mov cr3, …)
    - Creates and manages protection domains
    - The *kernel*!
System Calls

User-level syscall function:

```assembly
mov syscall_num, %eax
/* save normal regs */
push %ebp
mov %esp, %ebp
mov $1f, %ecx
sysenter
```

1:

```assembly
mov %eax, 0
pop %ebp
/* restore normal regs */
```

Kernel system call handler:

```assembly
pushl %ebp /* user-sp */
pushl %ecx /* user-ip */
call *cos_syscall_tbl(,%eax,4)
popl %edx /* user-ip */
popl %ecx /* and sp */
sysexit
```

- Sysenter instruction changes the mode from user → kernel
- Sysexit does opposite
System Structure

- Defines
  - how different *parts* of the system (or *subsystems*) interact
  - the separation of mechanism/policy throughout the system
Bias

• As you read papers, please, choose sides
  • What do you like about specific approaches?
  • And what are the limitations?
  • And always: what is new about the paper (contributions)

• Which systems present the best trade-offs?
Volunteers

- Two system structure papers 2 weeks from today...

- Everyone: Email me with
  - Your interests
  - If you have any projects you're already working on
  - Which topics/titles (on the schedule) are most interesting to you