Introduction

ICS332
Operating Systems

Henri Casanova (henric@hawaii.edu)

Course Goal

- At this point in your life you:
  - Have used at least one OS
  - Know which OS runs on your computer
  - Know that with the OS you couldn't use your computer
- Yet, for most of you, the OS is pretty mysterious
- Say your art major friend asks: "What really happens when I double click on an icon to run an application on my computer?"
- Could you give a decent answer besides: "Amazingly, it all works?"
- As a Computer Scientist it's ok to not know much about how a car/fridge/airplane works, but it's not ok to be clueless about how an OS works

Motivation to Study OSes?

- After all, very few of you will develop an actual OS
- **But**, most of you will work on complex systems that couple together many components (not an ICS211 assignment)
- **Obvious Motivation:** These systems all use the OS heavily
  - Important to know how to the use the OS as a programmer
  - Important to know that the OS can and cannot do
  - Important to know what happens under the cover to understand bug, security, performance, etc.
- **Meta Motivation (you have to trust me on this one):** Knowing OS principles makes you a better software architect and developer
  - OS concepts are massively re-usable in your own projects
    - Asking oneself "how does the OS do this?" Is always useful
    - The need for OS knowledge arises regularly for almost every developer throughout their career

OS in the News

- If you follow the news, general or tech-oriented, you know there are quite a few OS-related item each month
  - Some about new “exciting” features targeted at consumers, often very vague on details
  - Some about problems/bugs, typically targeted at computer professionals
- After taking this course you should be able to understand these, to at least the OS side to them
  - Often, understanding to computer architecture is also needed, as vulnerabilities/attacks are typically at the software/hardware interface
- Let’s see two famous examples…

The New York Times

The software patches could slow the performance of affected machines by 20 to 30 percent, said Andres Freund, an independent software developer who has tested the new Linux code. The researchers who discovered the flaws voiced similar concerns

What's a kernel?

PCWorld FROM IDG

The kernel inside a chip is basically an invisible process that facilitates the way apps and functions work on your computer. It has complete control over your operating system. Your PC needs to switch between user mode and kernel mode thousands of times a day, making sure instructions and data flow seamlessly and instantaneously. Here's how The Register puts it: "Think

The Register® Biting the hand that feeds IT

Think of the kernel as God sitting on a cloud, looking down on Earth. It's there, and no normal being can see it, yet they can pray to it.

These KPTI patches move the kernel into a completely separate address space, so it's not just invisible to a running process, it's not even there at all. Really, this shouldn't be needed, but clearly there is a flaw in Intel's silicon that allows kernel access protections to be bypassed in some way.

THE VERGE

THURSDAY, JANUARY 4, 2018 | CHIPOCALYPSE NOW

FLAW IS RELATED TO KERNEL MEMORY ACCESS

The exact bug is related to the way that regular apps and programs can discover the contents of protect kernel memory areas. Kernels in operating systems have complete control over the entire system, and connect applications to the processor, memory, and other hardware inside a computer. There appears to be a flaw in Intel's processors that lets attackers bypass kernel access protections so that regular apps can read the contents of kernel memory. To protect against this, Linux programmers have been separating the kernel's memory away from user processes in what's being called "Kernel Page Table Isolation."

January '18 "Spectre, Meltdown"

The New York Times

The so
perf
perc
soft
cod
voi

What's a kernel? PCWorld FROM IDG

The kernel inside a chip is basically an invisible process that facilitates the way apps and functions work on your computer. It has complete control over

d

ink

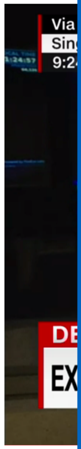
t's

ess

at

s

way.



You'll also be able to spot glaring errors/confusions in newspaper articles (there is a big one on this slide)

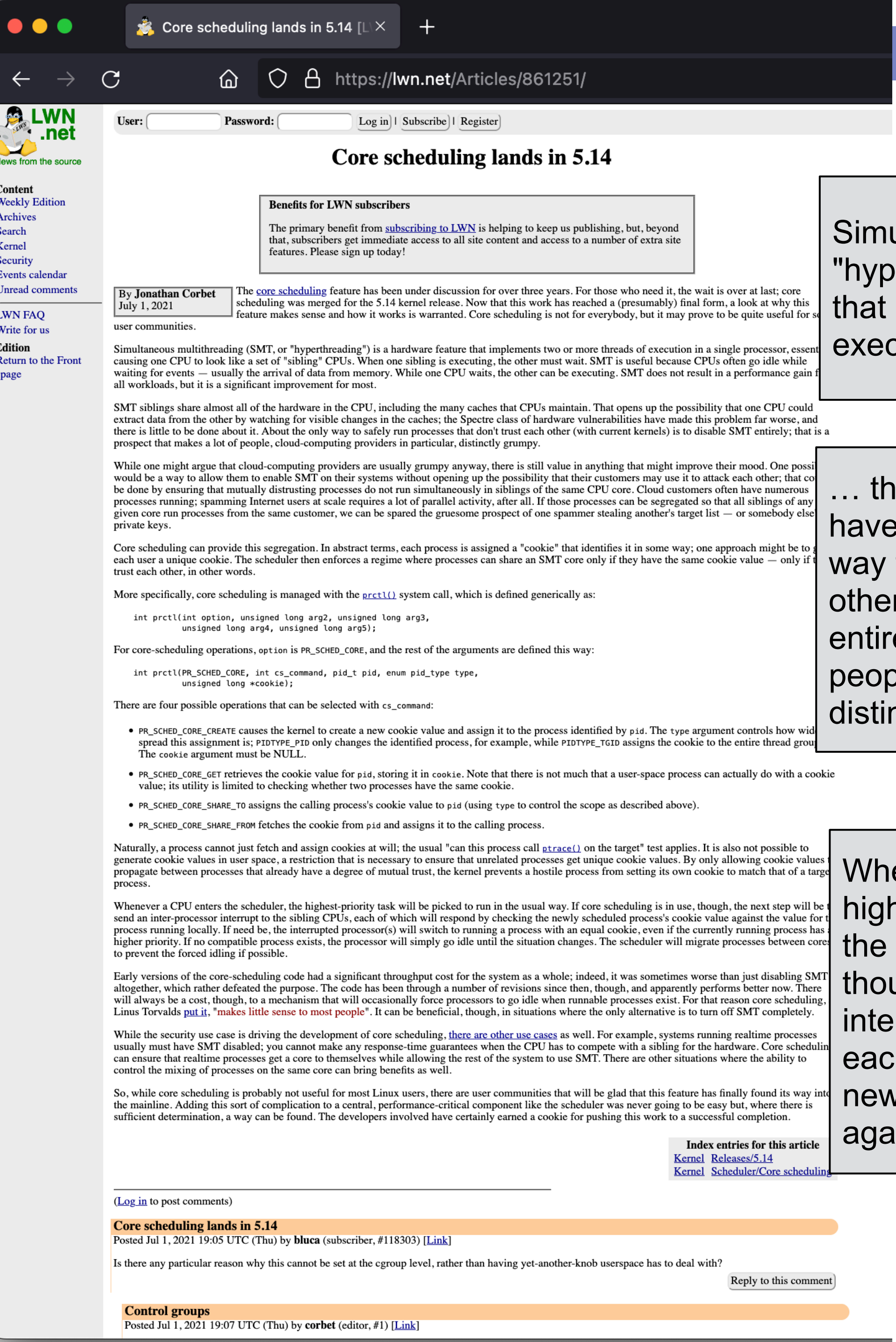
We'll show this slide again to point them out…

ets

s of

Ja
“Spectre,
Meltdown”

kernel memory. To protect against this, Linux programmers have been separating the kernel's memory away from user processes in what's being called “Kernel Page Table Isolation.”


Core scheduling lands in 5.14

https://lwn.net/Articles/861251/

LWN .net
lews from the source

Content
Weekly Edition
Archives
Search
Kernel
Security
Events calendar
Unread comments

LWN FAQ
Write for us

Edition
Return to the Front page

User: Password: Log in | Subscribe | Register

Core scheduling lands in 5.14

Benefits for LWN subscribers

The primary benefit from subscribing to LWN is helping to keep us publishing, but, beyond that, subscribers get immediate access to all site content and access to a number of extra site features. Please sign up today!

By **Jonathan Corbet**
July 1, 2021

The core scheduling feature has been under discussion for over three years. For those who need it, the wait is over at last; core scheduling was merged for the 5.14 kernel release. Now that this work has reached a (presumably) final form, a look at why this feature makes sense and how it works is warranted. Core scheduling is not for everybody, but it may prove to be quite useful for s user communities.

Simultaneous multithreading (SMT, or "hyperthreading") is a hardware feature that implements two or more threads of execution in a single processor, essent causing one CPU to look like a set of "sibling" CPUs. When one sibling is executing, the other must wait. SMT is useful because CPUs often go idle while waiting for events — usually the arrival of data from memory. While one CPU waits, the other can be executing. SMT does not result in a performance gain f all workloads, but it is a significant improvement for most.

SMT siblings share almost all of the hardware in the CPU, including the many caches that CPUs maintain. That opens up the possibility that one CPU could extract data from the other by watching for visible changes in the caches; the Spectre class of hardware vulnerabilities have made this problem far worse, and there is little to be done about it. About the only way to safely run processes that don't trust each other (with current kernels) is to disable SMT entirely; that is a prospect that makes a lot of people, cloud-computing providers in particular, distinctly grumpy.

While one might argue that cloud-computing providers are usually grumpy anyway, there is still value in anything that might improve their mood. One possi would be a way to allow them to enable SMT on their systems without opening up the possibility that their customers may use it to attack each other; that co be done by ensuring that mutually distrusting processes do not run simultaneously in siblings of the same CPU core. Cloud customers often have numerous processes running; spamming Internet users at scale requires a lot of parallel activity, after all. If those processes can be segregated so that all siblings of any given core run processes from the same customer, we can be spared the gruesome prospect of one spammer stealing another's target list — or somebody else private keys.

Core scheduling can provide this segregation. In abstract terms, each process is assigned a "cookie" that identifies it in some way; one approach might be to each user a unique cookie. The scheduler then enforces a regime where processes can share an SMT core only if they have the same cookie value — only if t trust each other, in other words.

More specifically, core scheduling is managed with the `prctl()` system call, which is defined generically as:

```
    int prctl(int option, unsigned long arg2, unsigned long arg3,
              unsigned long arg4, unsigned long arg5);
```

For core-scheduling operations, `option` is `PR_SCHED_CORE`, and the rest of the arguments are defined this way:

```
    int prctl(PR_SCHED_CORE, int cs_command, pid_t pid, enum pid_type type,
              unsigned long *cookie);
```

There are four possible operations that can be selected with `cs_command`:

- `PR_SCHED_CORE_CREATE` causes the kernel to create a new cookie value and assign it to the process identified by `pid`. The `type` argument controls how wid spread this assignment is; `PIDTYPE_PID` only changes the identified process, for example, while `PIDTYPE_TGID` assigns the cookie to the entire thread grou The `cookie` argument must be NULL.
- `PR_SCHED_CORE_GET` retrieves the cookie value for `pid`, storing it in `cookie`. Note that there is not much that a user-space process can actually do with a cookie value; its utility is limited to checking whether two processes have the same cookie.
- `PR_SCHED_CORE_SHARE_TO` assigns the calling process's cookie value to `pid` (using `type` to control the scope as described above).
- `PR_SCHED_CORE_SHARE_FROM` fetches the cookie from `pid` and assigns it to the calling process.

Naturally, a process cannot just fetch and assign cookies at will; the usual "can this process call `ptrace()` on the target" test applies. It is also not possible to generate cookie values in user space, a restriction that is necessary to ensure that unrelated processes get unique cookie values. By only allowing cookie values propagate between processes that already have a degree of mutual trust, the kernel prevents a hostile process from setting its own cookie to match that of a targe process.

Whenever a CPU enters the scheduler, the highest-priority task will be picked to run in the usual way. If core scheduling is in use, though, the next step will be send an inter-processor interrupt to the sibling CPUs, each of which will respond by checking the newly scheduled process's cookie value against the value for t process running locally. If need be, the interrupted processor(s) will switch to running a process with an equal cookie, even if the currently running process has higher priority. If no compatible process exists, the processor will simply go idle until the situation changes. The scheduler will migrate processes between cores to prevent the forced idling if possible.

Early versions of the core-scheduling code had a significant throughput cost for the system as a whole; indeed, it was sometimes worse than just disabling SMT altogether, which rather defeated the purpose. The code has been through a number of revisions since then, though, and apparently performs better now. There will always be a cost, though, to a mechanism that will occasionally force processors to go idle when runnable processes exist. For that reason core scheduling, Linus Torvalds put it, "makes little sense to most people". It can be beneficial, though, in situations where the only alternative is to turn off SMT completely.

While the security use case is driving the development of core scheduling, there are other use cases as well. For example, systems running realtime processes usually must have SMT disabled; you cannot make any response-time guarantees when the CPU has to compete with a sibling for the hardware. Core schedulin can ensure that realtime processes get a core to themselves while allowing the rest of the system to use SMT. There are other situations where the ability to control the mixing of processes on the same core can bring benefits as well.

So, while core scheduling is probably not useful for most Linux users, there are user communities that will be glad that this feature has finally found its way int the mainline. Adding this sort of complication to a central, performance-critical component like the scheduler was never going to be easy but, where there is sufficient determination, a way can be found. The developers involved have certainly earned a cookie for pushing this work to a successful completion.

Index entries for this article	
Kernel	Releases/5.14
Kernel	Scheduler/Core scheduling

(Log in to post comments)

Core scheduling lands in 5.14
Posted Jul 1, 2021 19:05 UTC (Thu) by **bluca** (subscriber, #118303) [Link]

Is there any particular reason why this cannot be set at the cgroup level, rather than having yet-another-knob userspace has to deal with?

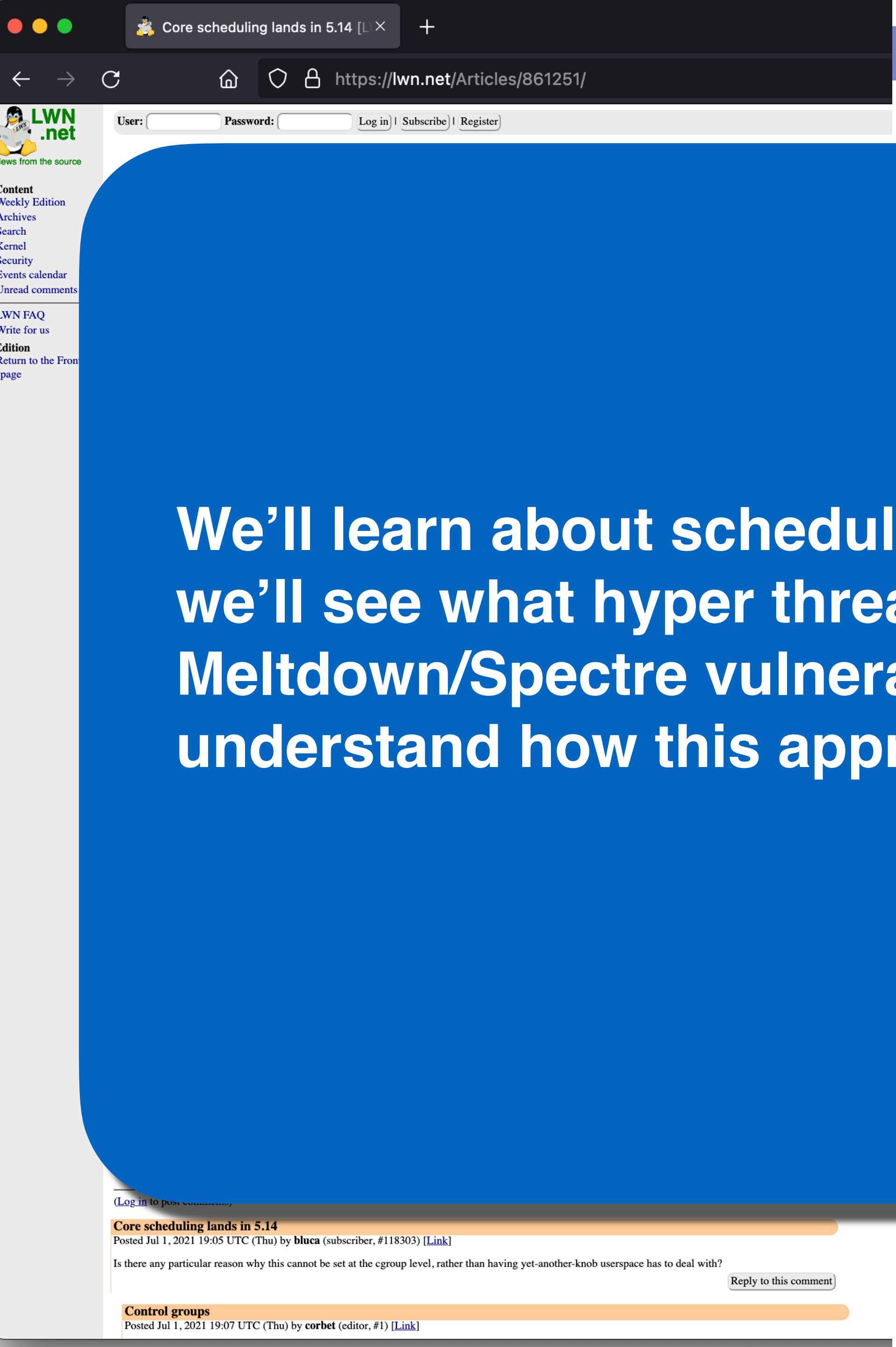
Reply to this comment

Control groups
Posted Jul 1, 2021 19:07 UTC (Thu) by **corbet** (editor, #1) [Link]

Simultaneous multithreading (SMT, or "hyperthreading") is a hardware feature that implements two or more threads of execution in a single processor

… the **Spectre** class of hardware vulnerabilities have made this problem far worse. About the only way to safely run processes that don't trust each other (with current kernels) is to disable SMT entirely; that is a prospect that makes a lot of people, cloud-computing providers in particular, distinctly grumpy.

Whenever a CPU enters the scheduler, the highest-priority task will be picked to run in the usual way. If core scheduling is in use, though, the next step will be to send an inter-processor interrupt to the sibling CPUs, each of which will respond by checking the newly scheduled process's cookie value against the value for the process running


Core scheduling lands in 5.14 [L

https://lwn.net/Articles/861251/

User: Password: Log in | Subscribe | Register

LWN .net
ews from the source

ontent
Weekly Edition
Archives
Search
Kernel
Security
Events calendar
Unread comments

WN FAQ
Write for us

dition
Return to the Fron
page

(Log in to post comments)

Core scheduling lands in 5.14
Posted Jul 1, 2021 19:05 UTC (Thu) by **bluca** (subscriber, #118303) [Link]

Is there any particular reason why this cannot be set at the cgroup level, rather than having yet-another-knob userspace has to deal with?

Reply to this comment

Control groups
Posted Jul 1, 2021 19:07 UTC (Thu) by **corbet** (editor, #1) [Link]

We’ll learn about scheduling as done by the kernel, we’ll see what hyper threading is, and given the Meltdown/Spectre vulnerabilities, we’ll easily understand how this approach works

Disclaimer (1)

- Wouldn’t it be great for us to develop an OS during the semester
- Sadly, it’s not feasible:
  - Too hard, time-consuming, programming-heavy for most students at this stage in their education, especially students who may struggle with C
  - Or at least given the expected number of hours a student should put into an undergraduate course...
- As a result, undergraduate OS courses are often less “hands-on” than what some students expect
  - Typically a few of you come into this thinking we’re going to do awesome hacks in the Linux kernel….and will be disappointed
- If your dream is to get your hands dirty with the OS kernel you can:
  - Do an internship at a company that does low level / OS-related work, do an OS-related Google Summer of code, ..
  - Take the OS graduate course
  - Do it on your own (each semester there is at least one student who does this and there are tons of on-line resources!)

Disclaimer (2)

- I am not an "OS geek"
- If you've read the source code of some OS, if you have worked on an OS (internship), if you have any useful knowledge, you are very welcome to share with the class
- I'm always happy to have course contents evolve dynamically based on students suggested topics within some reasonable bounds
- This said, the class is more on general principles than specific implementations (since those change often, as we will see)
  - You can learn a lot about OSes without necessarily spending hours looking at OS code

Teaching OS is not easy

- A significant part of the material is of the “here is how it works and why it’s a good idea” kind
  - Precisely because it’s not feasible to have the full-fledged hands-on experience at the undergraduate level
  - Don’t fear, there will be plenty of programming / hands-on activities in this course
- Although I try to make the course as interactive as possible, there is just a limit to what any instructor can do for some of this material at the undergraduate level
- Bottom line:
  - The course is fun when students are engaged and ask questions
  - The course is dull when students are silent

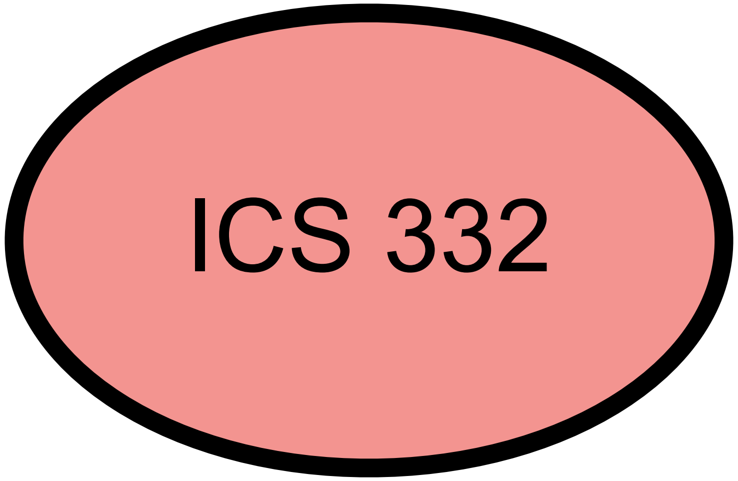
What we will learn (1)

- Roles of an operating system
- Fundamental principles of operating system design and kernel implementation
- Key features of operating systems of practical importance
  - The course content is not specific to a particular OS
  - Many OSes do things in similar way, but they also have key differences
  - We will often reference Unix derivatives (Mac OS, Linux, iOS, Android, ...) and Windows
  - We will mention ”historical” OSes whenever relevant
  - We will not study special-purpose OSes (e.g., real-time, network operating systems, ...)

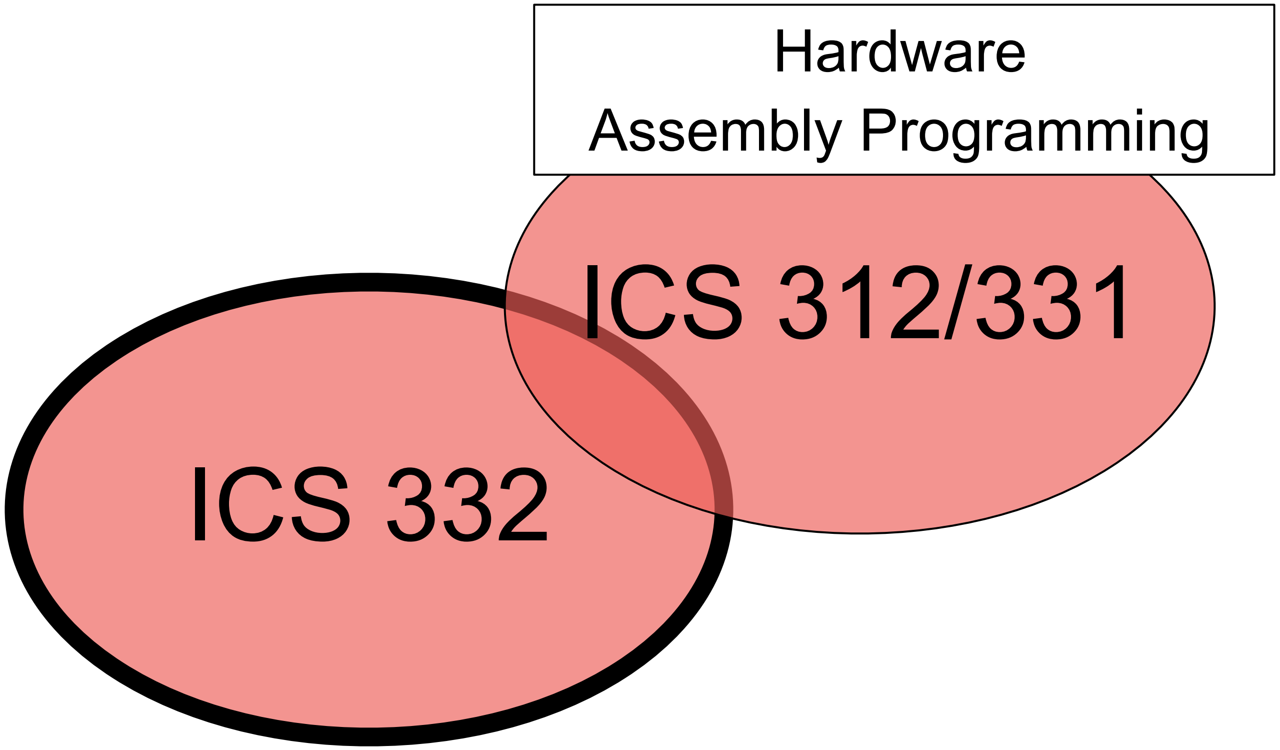
What we will learn (2)

- Fundamental components and principles of modern operating systems:
  - Processes and Threads Management Scheduling
  - Synchronization (barely scratching the surface here)
  - Memory and Virtual Memory Management
  - Storage and, if time, File Systems
- We will have several programming assignments are are more about “using” the OS than about “implementing” the OS
  - i.e., what most of you will need most in your profession

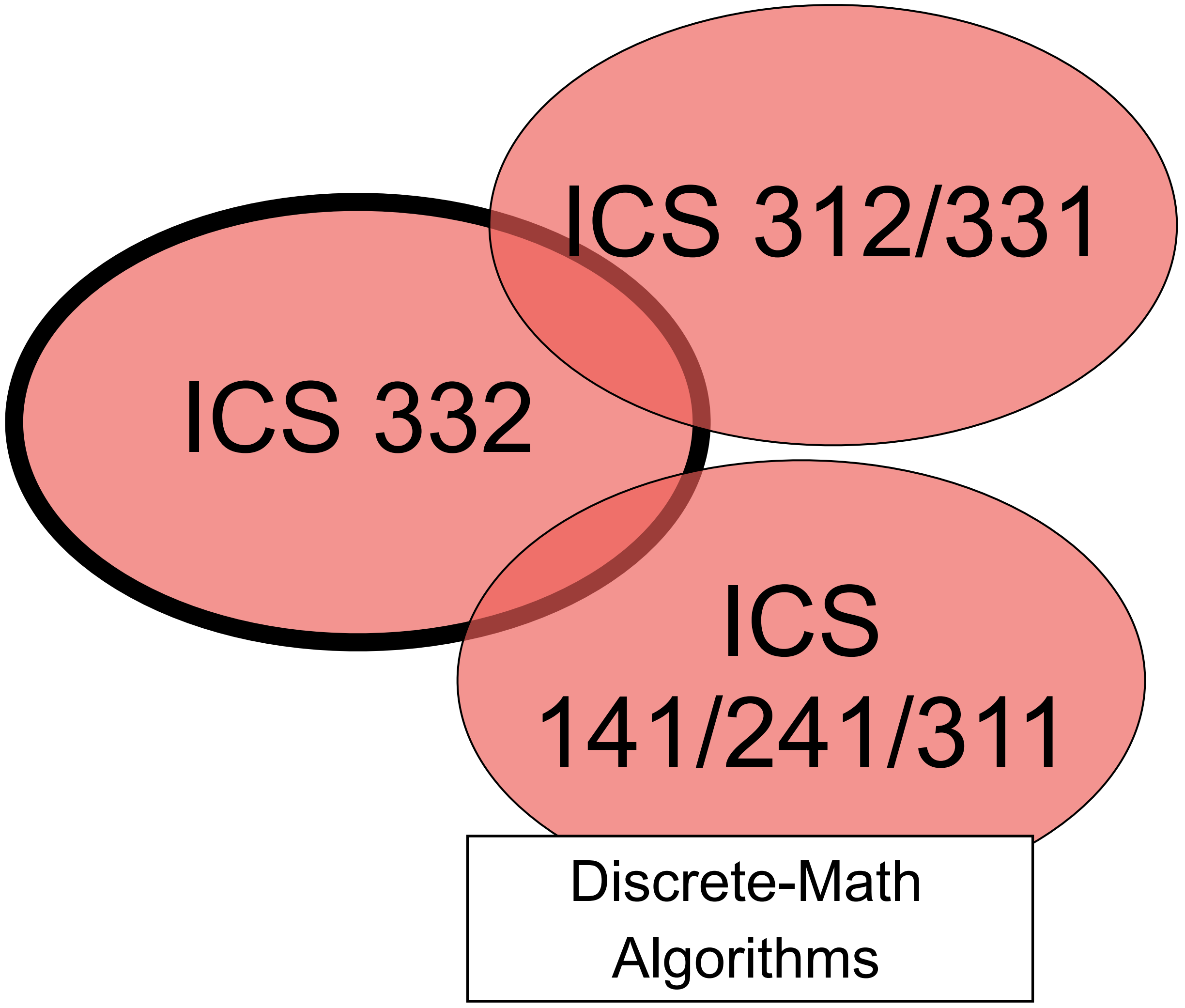
ICS332 and the ICS Curriculum



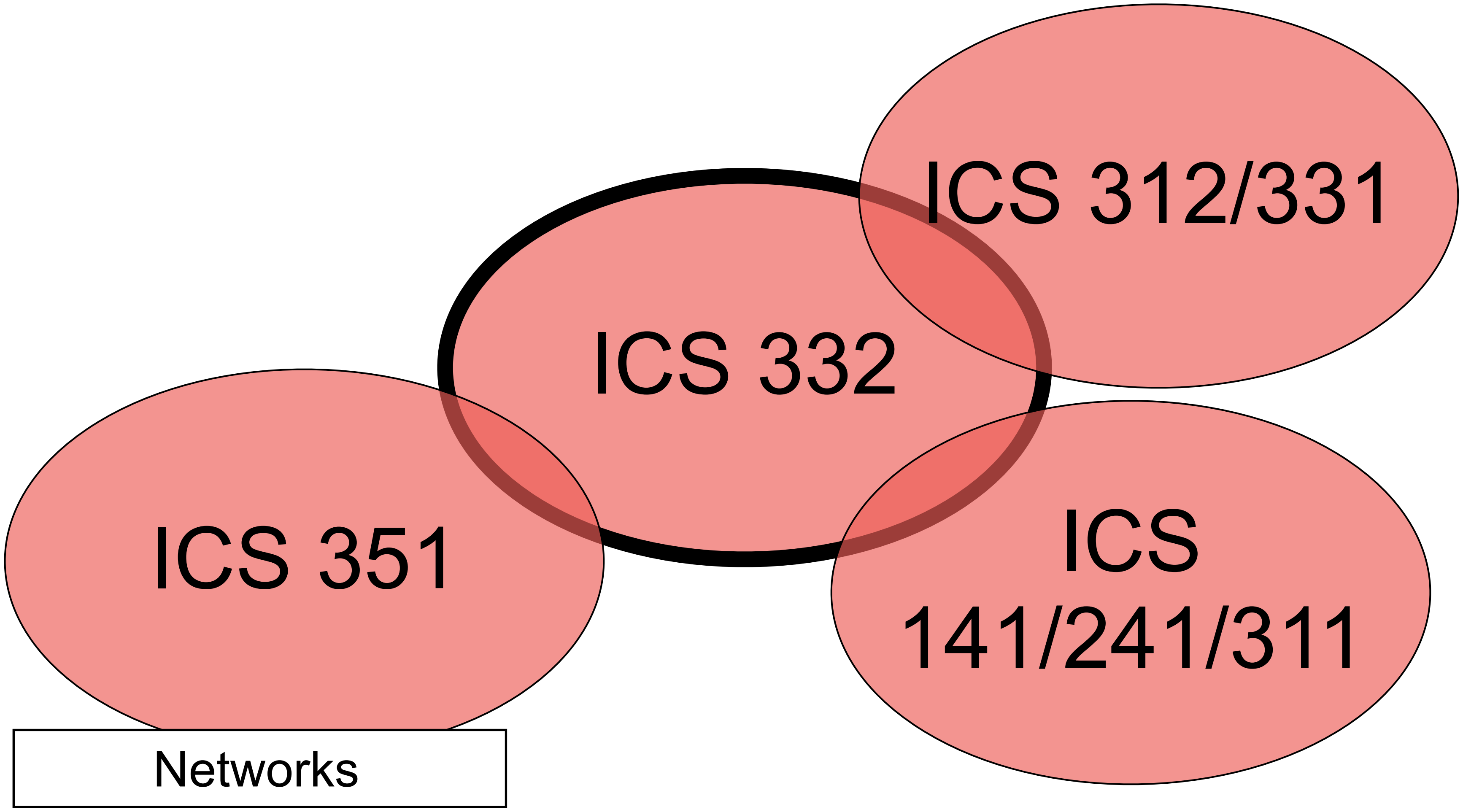
ICS332 and the ICS Curriculum



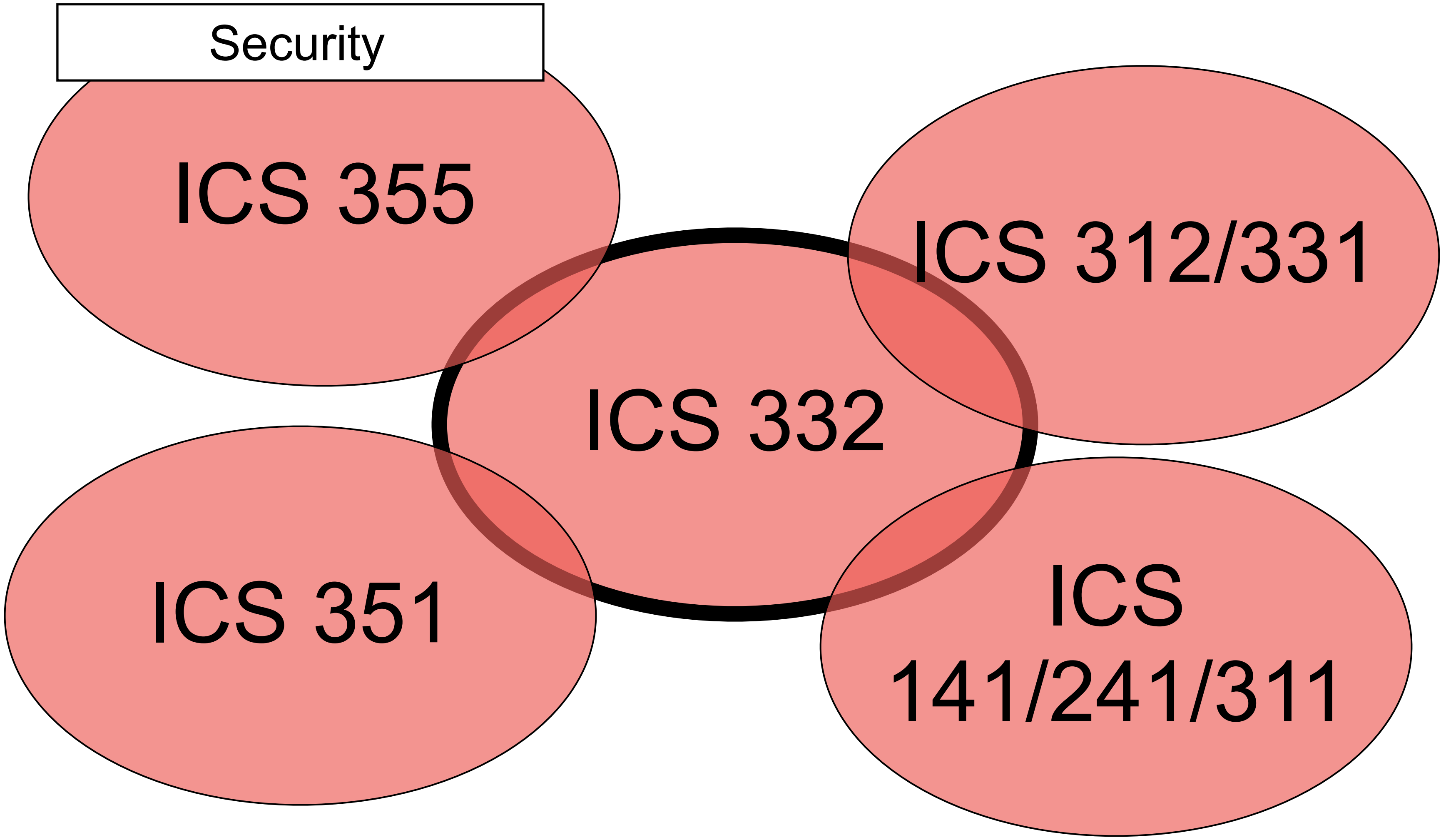
ICS332 and the ICS Curriculum



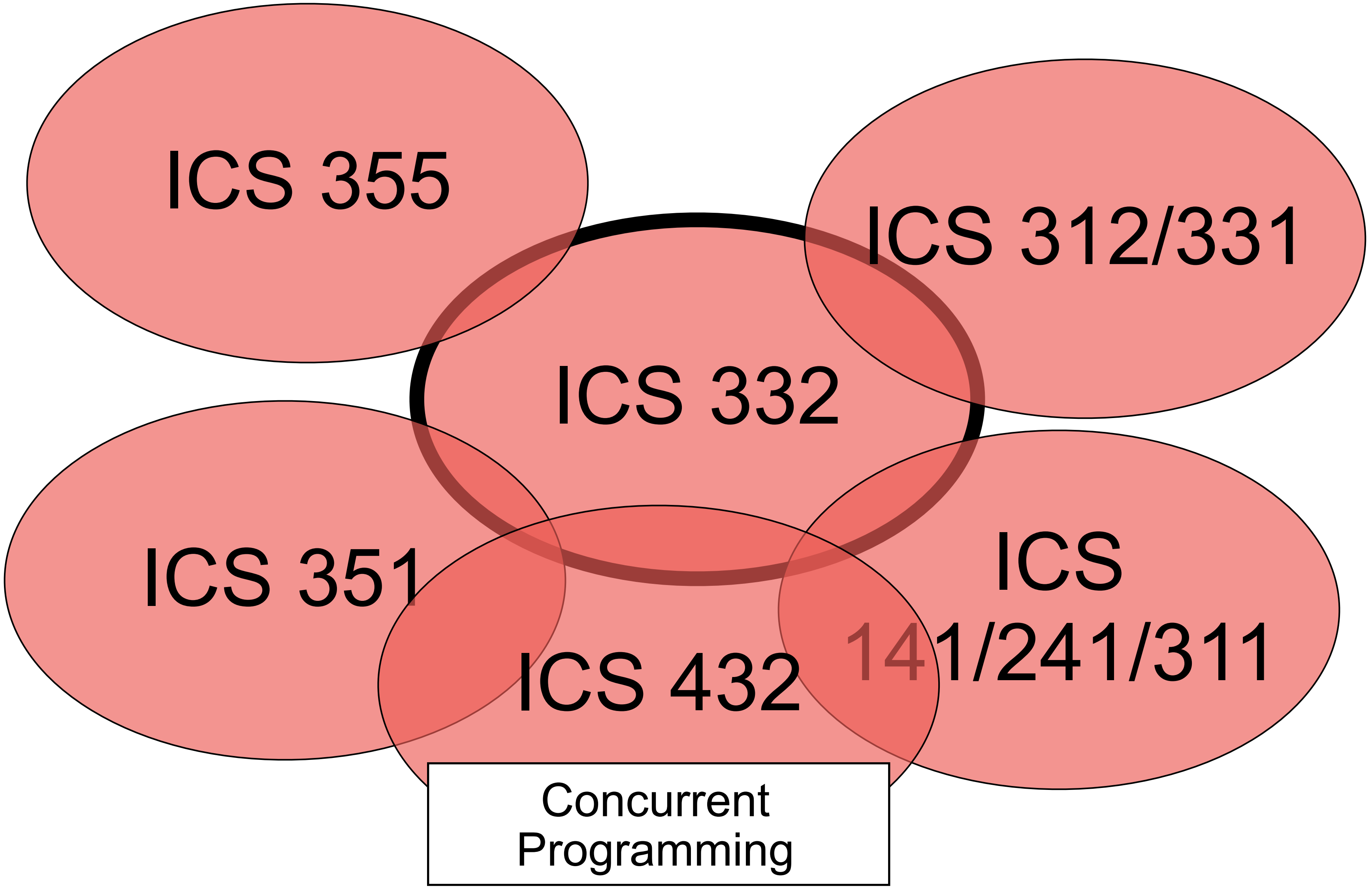
ICS332 and the ICS Curriculum



ICS332 and the ICS Curriculum



ICS332 and the ICS Curriculum



Course Website

- Located at:
  - https://www.chadmorita.com/ics332s24
- Organized as Modules
  - All lecture notes as PDF files
  - Pointers to useful on-line material
  - All assignments
  - A link to the Syllabus
    - Which we're going over now in these slides
- Let's look at the Web site...

Textbook

- Operating Systems: Three Easy Pieces (a.k.a. OSTEP) 1.00 by Arpaci-Dusseau, R. H. and Arpaci-Dusseau, A. C
    - Freely available!
- Lectures are tightly connected to particular chapters therein
- There will be reading assignments from this textbook, as indicated on the lecture notes
    - Up to you whether you prefer to read them before or after our lectures....
- Some exam questions and assignments will be directly from or inspired by the textbook
- There are several classic texts for Operating Systems (shown in the "syllabus" page on the course Web site)

Course Content

- In spite of my best efforts it happens that the course Web site could have small problems (typos, missing link, etc.)

- Anytime you see anything strange/broken on the Web site, please let me know right away!

Grading

- Two exams (35%)
  - One midterm exam
  - One final exam
- Quizzes (15%)
  - Roughly every week
- Homework assignments (50%)
- Read the syllabus’ statement about “academic dishonesty”

Quizzes

- Roughly one quiz a week
- Quizzes will be unannounced

Homework Assignments (1)

- All assignments must be turned in electronically using Laulima by 11:55PM HST on the day the assignment is due
  - **Scanned hand-written assignments not allowed**
- Late Assignments
  - 10% penalty for up to 24 hours of lateness
  - A grade of *zero* for more than 24 hours of lateness
  - e.g., if the due date is 3/10, an assignment turned in at 1AM on 3/11 will be penalized by 10%, and given a zero if turned in at 5PM on 3/12

Homework Assignments (2)

- If Laulima is down, just e-mail us (me and the TA) your submission immediately (don’t send an e-mail that says “Laulima is down what should I do?” Which we’ll only see the day after)
- After submitting double-check what you submitted!
  - “Oops, I submitted an empty file... here is what I really meant to submit yesterday” will not be accepted

Homework Assignments (3)

- All assignments are individual (not group) assignments
- Some assignments will be pencil-and-paper that require no programming
  - But may require the use of a Linux box to observe things
- Some assignments are programming assignments
  - Write code and/or report on code execution
- Pencil-and-paper and programming assignments can overlap in time

Homework Assignments (4)

- Instructor/TA will not answer assignment-related e-mails on the day the assignment is due!

Programming Assignments (1)

- Java 8 or later, some C, some scripting
- Some programming assignments can indifferently be implemented on Windows, Mac OS, or Linux, others will require Linux or Mac OS, and yet others will require Linux.
- You can use your own machine (or a machine in a lab) for the assignments, using whatever editor or IDE you want
- BUT you must test/run your code on a Linux (Virtual) machine
  - Because that’s what we’ll use to testing/grading
  - One source of errors is Windows (“\”, ”;”) vs. Linux (“/”, ”:”)

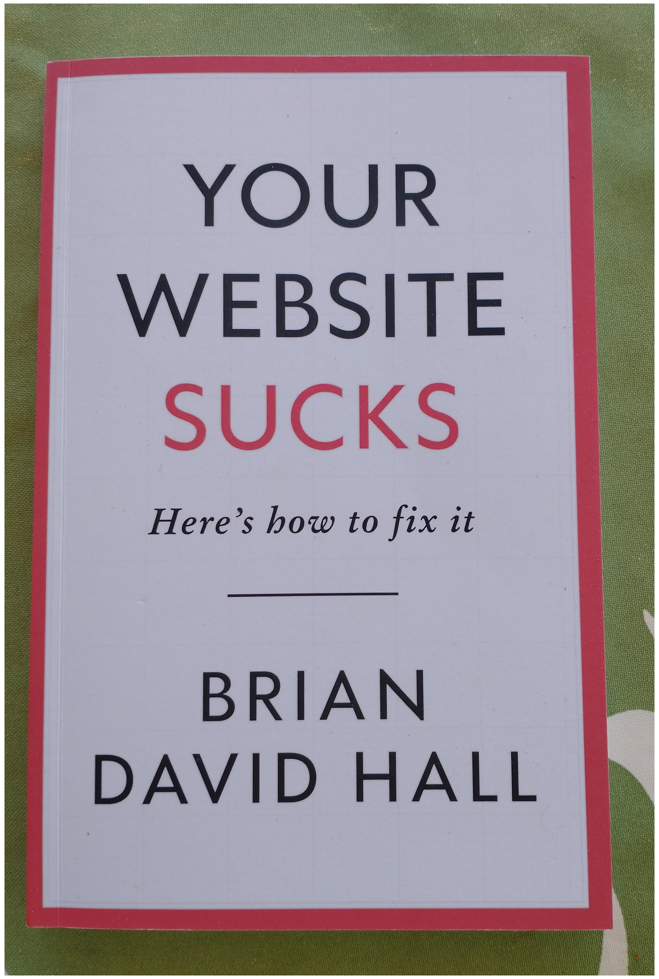
Programming Assignments (2)

- Each programming assignment has specifications and examples
  - Command-line arguments, file names, etc.
  - If command-line arguments are not correct, then your programs have to exit gracefully
  - If you find specifications unclear, let us know right away
- Non conforming with these specifications makes us less tolerant (and sometimes non-tolerant) in terms of grading
  - We will not go into your code to fix it;
  - You will lose points for not following the specifications
- Following specifications and writing robust code will be a HUGE part of your professional lives

The Linux CLI (Shell)

- Knowledge of the UNIX/Linux CLI (Shell) is needed in this course
- Show of hand: who feels somewhat familiar with the Unix/Linux CLI?
- A huge potential side-benefit of taking this course is to become decent/good with the command-line
    - About 25% of students passing the course tell me that they are forever grateful that they forced themselves to learn/use the Shell
    - This is also something we hear from alumni who, once in a job, “discover” that 99% of back-end systems are Linux based and using the Shell is a daily activity
- This module (Getting Started) contains some pointers. Let’s look at them now...
    - Many of you really want to look at this material to prepare for upcoming programming assignments
- In case you’re not convinced …..

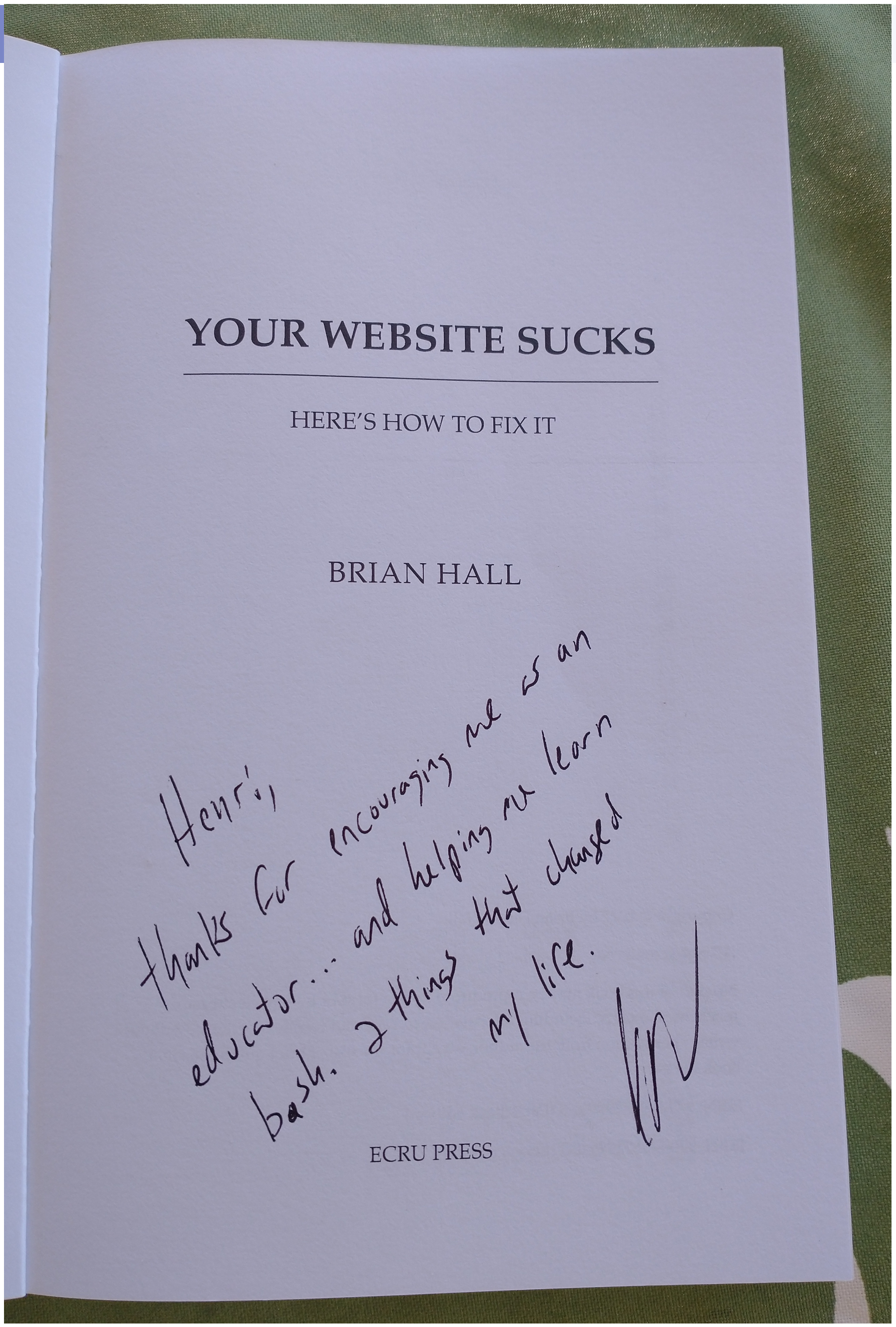
ICS Alumnus Brian Hall



YOUR WEBSITE SUCKS

Here's how to fix it

BRIAN DAVID HALL



YOUR WEBSITE SUCKS

HERE'S HOW TO FIX IT

BRIAN HALL

Henri,
thanks for encouraging me as an educator... and helping me learn bash. 2 things that changed my life.
[illegible]

ECRU PRESS

Brian Hall <bdh@briandavidhall.com>
Email him to request access to his Udemy course on the CLI (for free!)

On the Importance of Terminology

- As we learn about Operating Systems we will encounter a lot of terminology
- Recognizing and using the correct terminology is part of what we learn in this course
- Knowing the terminology is very important (e.g., in a job interview, in a professional context)
- So, in class, whenever a student asks a question (which is highly encouraged!), I might rephrase the question using proper terminology
  - This is not to be annoying or belittling, it's to make sure we all come out of this course speaking the language of Operating Systems (and of Computer Science)
- Of course, terminology will be part of the quizzes/exams

How to not do well in this course?

- **Don’t come to class (“the slides are nice”)**
  - We do a LOT of stuff in class, including live coding, and I give a lot of explanations, examples
- **Start assignments late (“I work better under pressure”)**
  - OSes are difficult topic
  - Starting late seems to be a growing trend, and it’s a problem
  - Read the assignment early to subconsciously start thinking about it
- **Don’t turn in assignments**
  - Every semester some students do not turn in assignments and then seem surprised to fail (not sure what that’s about)
  - Just count your points to know where you’re at!
- **Don’t come to office hours (“The instructor is scary because he shows ‘how to no do well in this course?’ slides”)**
  - After you struggle for a while on something, drop by
  - But don’t expect to “camp” in the office hours for the solutions to be given out
  - Instructor and TA office hours are an amazing service provided to you, and yet, they go mostly unused

How to not do well in this course?

- **Cheat**
  - Almost every semester students are caught cheating
    - Cheating is bad for many reasons, including hurting the reputation of ICS graduates!
    - This is part of the reason for 50% of the course's points being exams
  - If you are caught cheating or enabling cheating:
    - zero on the assignment/exam
    - overall grade lowered by a step (i.e., a "B" becomes a "C")
    - reported to UH's Office of Judicial Affairs (as required)
- **Expect that "what can I do for extra credit?" will be met with a positive response… it won't**
- **Don't study for the quizzes**
  - "It's only 7% of the grade"
  - But studying for quizzes is a HUGE help to prepare for exams
  - When I don't do quizzes, the average grade drops!

Questions?

- Any questions on the syllabus?
- Any questions on the course in general?
- Do the “participation verification” thing on Laulima!

Conclusion

- Let’s look at (ungraded) Homework #0…