CSci 5103
Operating Systems

Jon Weissman

Administrivia

Greetings

- Welcome to CSci 5103!
  - me: Jon Weissman, Professor CS
    - office hours M 9-11am, 4-225F KH
    - or when I am around
  - interests: distributed and parallel systems
  - cycling, hiking, XC-ski
  - TA: Bowen Wang
    - office hours TBD, 2-209 KH
- This is a grad-level OS course suitable for grad students and highly motivated senior undergrads

Who Gets In?

- 1 Effective TA – cap around 60
  - 62 enrolled in room, 9 in UNITE
- Will make final decision by next Thursday based on who shows up today; preference to CS grads, CS seniors, CS majors, ….
  - Class will be offered again in Spring 2020
- If you plan on dropping PLEASE let me know ASAP (as a courtesy to your classmates).

More Admin

- 5103 is hard work … but it will be fun work ☺

- Prereqs
  - undergraduate OS (4061 or equiv.)
  - soft prereq: Computer Org/Architecture (2021)

- Knowledge of C/C++, Unix, and debugging is key
  - get to know `gdb` or `ddd`
  - sorry can't use Java
    - easy to gen assembly/sys calls with C
    - believe me this is a bigger burden on us … but we think it is the right way to learn OS concepts

More Admin

- Website: `http://www.cse-labs.umn.edu/classes/Fall-2017/csci5103`
  - check it out – read announcements daily
  - start by looking at schedule, syllabus, dates

- Books
  - Operating Systems: Principles and Practice 2nd Edition, Recursive Books (Anderson and Dahlin)
  - More cutting edge than Tanenbaum, S&G: industry practice
  - On-line materials including research papers

More Admin

- Lectures + active exercises + class participation
  - coming to class is important
  - papers and more advanced topics this semester

More Admin

- Grades
  - 4 programming projects, 2 exams (mid + final), 4 written homeworks (exam prep)
- Late work – 1 proj, 10% penalty, 1 extra day
- Some/most projects will be groups; all get same score
- Regrading – within 2 week window

More Admin

- Working together
  - Team projects require a necessary collaboration. No barriers on this collaboration.
  - Homeworks are done individually!
  - Can discuss meaning of questions or issues, but should not share code, solutions.

Topics

- Course Introduction: History and Background (1)
- Kernel, Processes, API (1)
- Threads (1)
- Synchronization (2)
- Scheduling (1)
- Memory Management and Virtual Memory (3)
- File Systems and Storage, I/O (3)
- File System Reliability (1)
- Protection and Security (1)
- Wrapup (1)

What do I need for this course?

- Computer architecture
  - CPU, interrupts, I/O devices, protection
- C/**C++** and Unix comfort
  - Systems programming (e.g. 4061) is required
  - Experience with Unix debuggers is also helpful
- Willingness to work hard
  - Systems is hard work … but your hard work will be rewarded. “No Pain No Gain”

Course Materials for CSci 5103

- Operating Systems: Principles and Practice (OSPP)
  - source for most of the lecture content, but **not** all
  - may take a bit from Tanenbaum Modern Operating Systems

- Linux Device Drivers
  - see web-page

- There will also be some papers to read, they will be posted soon

Textbook

- Lazowska, U Washington: “The text is quite sophisticated. You won't get it all on the first pass. The right approach is to [read each chapter before class and] re-read each chapter once we've covered the corresponding material… more of it will make sense then. *Don't save this re-reading until right before the mid-term or final – keep up.”*

Am I up to it?

- If Chapter 1 has you worried, you may want to bail.
- Also, can you “grok” this code?

```
void  thread_create(thread_t *thread, void (*func)(int), int arg) {
    // Allocate TCB and stack
    TCB *tcb = new TCB();

    thread->tcb = tcb;
    tcb->stack_size = INITIAL_STACK_SIZE;
    tcb->stack = new Stack(INITIAL_STACK_SIZE);

    tcb->sp = tcb->stack + INITIAL_STACK_SIZE;
    tcb->pc = stub;

    // Create a stack frame by pushing stub's arguments and start address
    // onto the stack: func, arg
    *(tcb->sp) = arg;
    tcb->sp--;
    *(tcb->sp) = func;
    tcb->sp--;

    …
    (*func)(arg);               // Execute the function func()
    thread_exit(0);             // If func() does not call exit,  call it
}
```

Or this?

```
#define DO_SYSCALL syscall(SYS_getpid)

unsigned int timediff(struct timeval before,
                      struct timeval after) {
  unsigned int diff;

  diff = after.tv_sec - before.tv_sec;
  diff *= 1000000;
  diff += (after.tv_usec - before.tv_usec);


  return diff;
}
```

4061 vs. 5103

- Small overlap in OS concepts
- We'll explore concepts in greater depth
  - 4061: locks, condition variables
  - 5103: how are these implemented, used today

- Focus is on the inside-view of the OS
  - How are things implemented INSIDE the OS
  - 4061: how can I manipulate processes?
  - 5103: how are processes implemented inside the kernel?
    - What kinds of architectural support is needed?

OS as case study

- Book promotes idea that OS is great way to learn about many system concepts useful even if you never ever look at OS source code!
  - abstraction
  - policy vs. mechanism
  - …

Programming Projects

- Reflect the 5103 orientation
- Systems-programming is the focus of 4061 – how does one use OS facilities from the outside
- Our projects generally reflect inside perspective
  - projects will help shed light on how the OS works internally, often this is a "grey-box" approach
  - some kernel level experimentation

Questions?

CSci 5103
Operating Systems

Jon Weissman

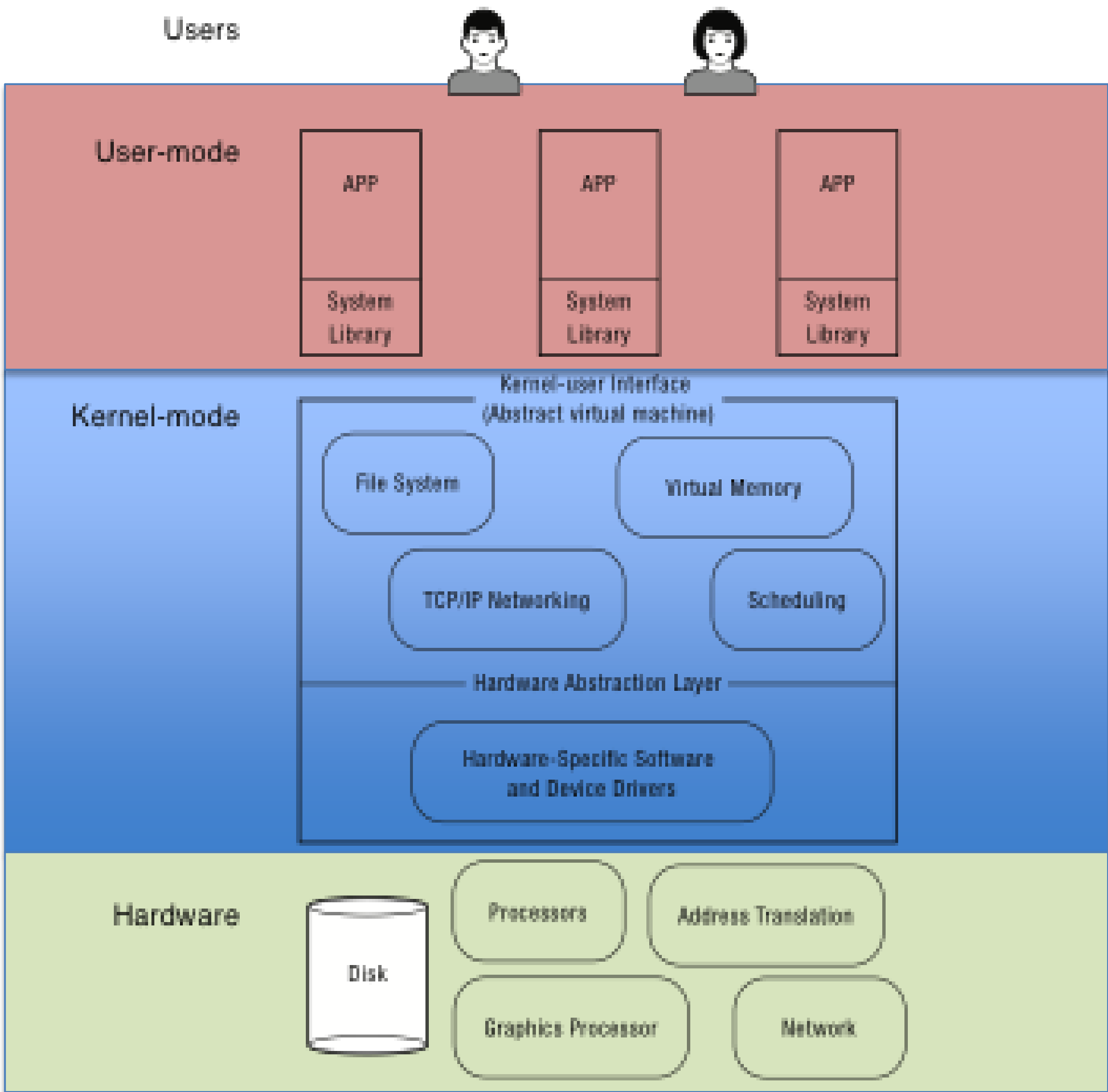
Introduction
Chapter 1, 2 OSPP

Main Points (for today)

- Operating system definition
- OS challenges briefly
  - Reliability, security, responsiveness, portability, …
- OS history
  - How we got here and where we are going?

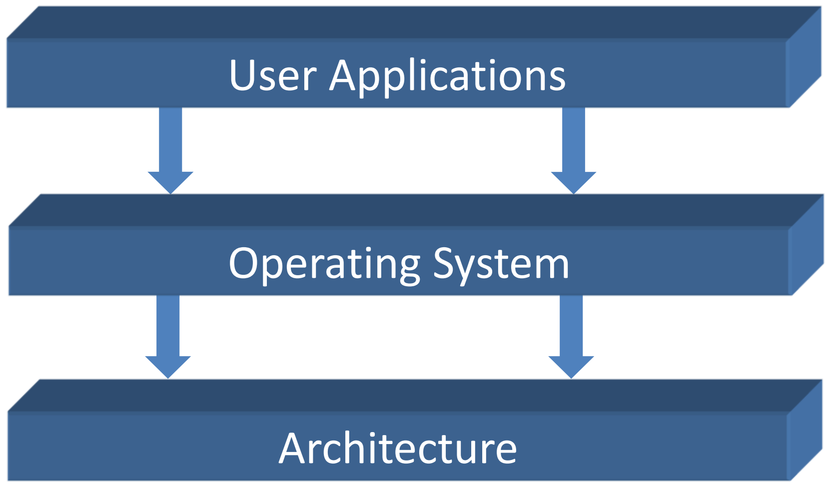
What is an operating system?

- Software to manage a computer's resources for its users and applications
- Two key interfaces



Operating Systems: Two Interfaces

- The operating system (OS) is the interface between user applications and the hardware.



- An OS implements a *virtual machine* that is easier to program than the raw hardware
  – Example?

Operating System Roles: OS Design Pattern

- Referee
  - Resource allocation among users, applications
  - Isolation of different users, applications from each other
  - Communication between users, applications
- Illusionist
  - Each application appears to have the entire machine to itself
  - Infinite number of processors, (near) infinite amount of memory, reliable storage, reliable network transport
- Glue
  - Common services for apps: libraries, terminals, drivers, cut-and-paste, …

Example: File Systems

- Referee
  - Prevent users from accessing each other's files without permission
  - Sharing disk space across the file system
- Illusionist
  - Files can grow (nearly) arbitrarily large
  - Files persist even when the machine crashes in the middle of a save
- Glue
  - named directories, stdio library (e.g. printf)

More?

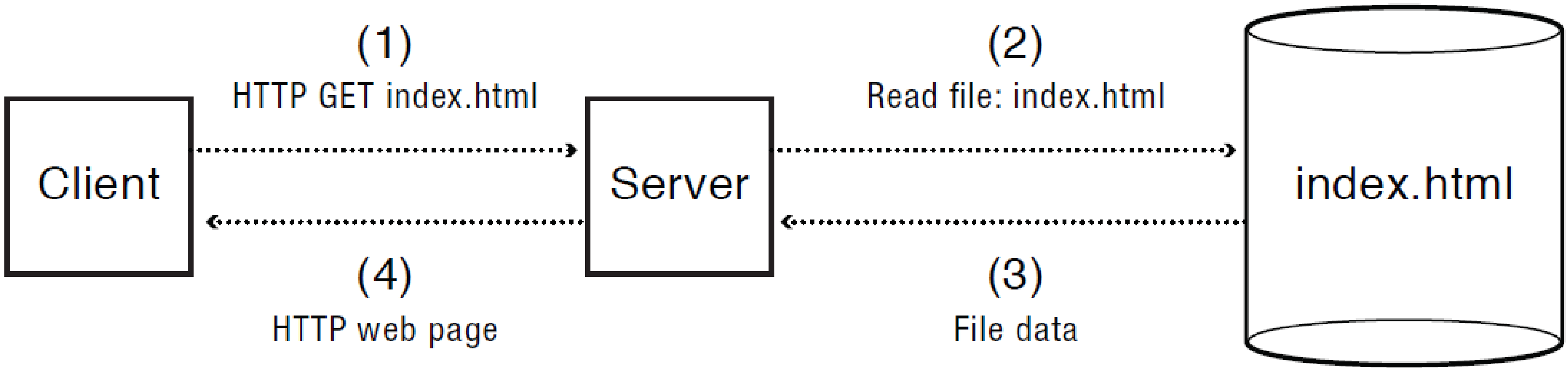
- Other examples from OS?

Not easy: many policy choices

- How should an operating system allocate processing time between competing uses?
  - Give the CPU to the first to arrive?
  - To the one that needs the least resources to complete? To the one that needs the most resources?

- Many choices as referee, illusionist, even glue represent trade-offs. No clear-cut best.

OS Design Pattern: web service



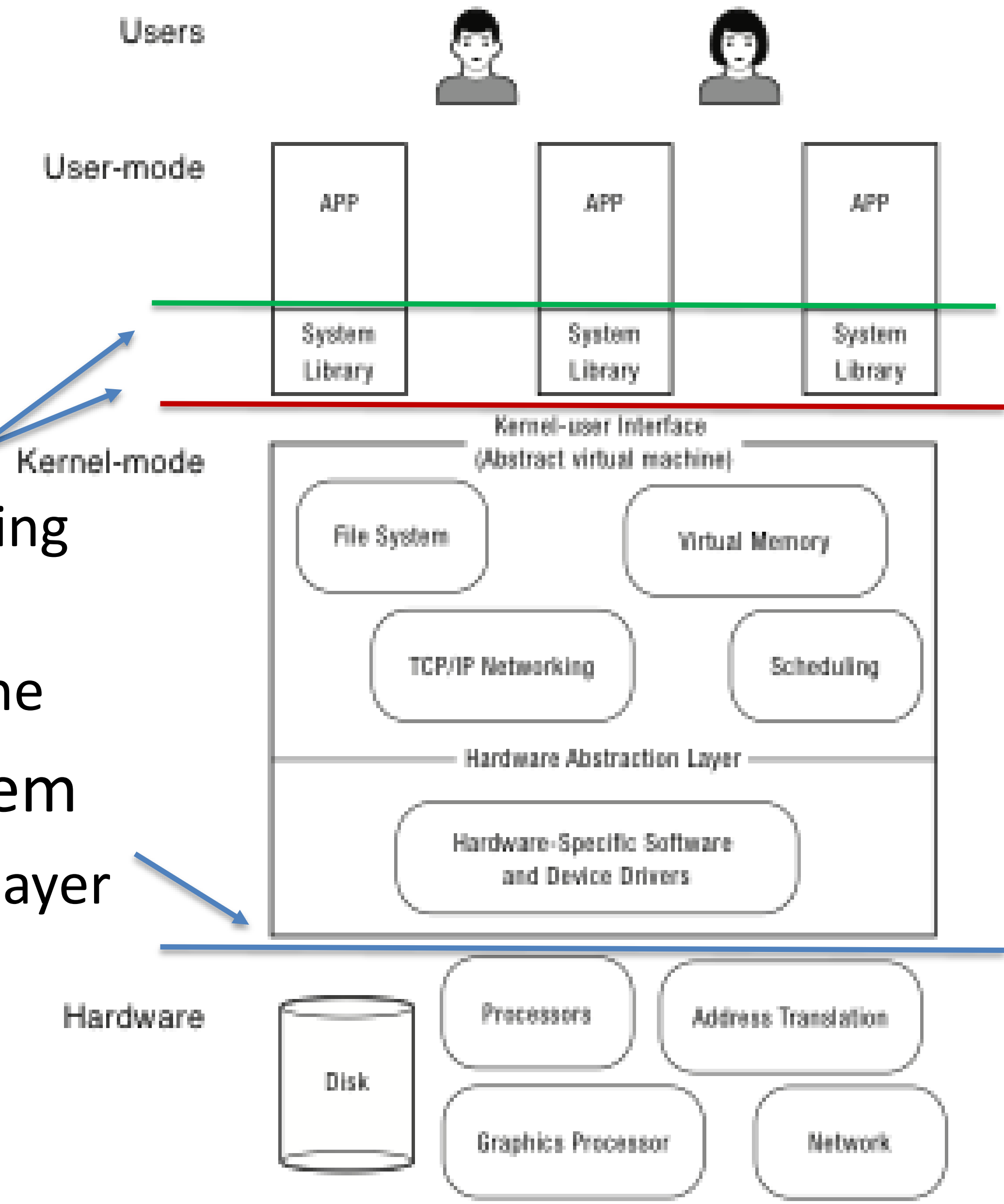
- How does the server manage many simultaneous client requests?
- R on client side?
- How do we make it seem that all web pages are local? (I)
- How do we enable Web programming, client-server connectivity, etc. (G)
- Book has other nice examples!

OS Challenges

- Reliability
  - Does the system do what it was designed to do?
- Availability
  - What portion of the time is the system working?
  - Mean Time To Failure (MTTF), Mean Time to Repair
- Security
  - Can the system be compromised by an attacker?
- Privacy
  - Data is accessible only to authorized users

OS Challenges

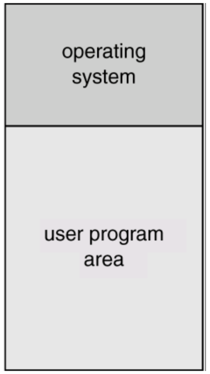
- Portability
  - For programs:
    - Application programming interface (API)
    - Abstract virtual machine
  - For the operating system
    - Hardware abstraction layer



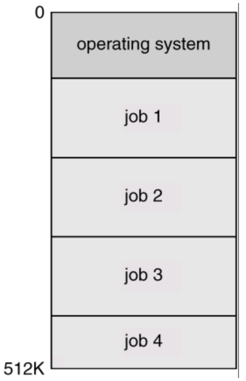
OS Challenges

- Performance
  - Latency/response time
    - How long does an operation take to complete?
  - Throughput
    - How many operations can be done per unit of time?
  - Overhead
    - How much extra work is done by the OS?
  - Fairness
    - How equal is the performance received by different users?
  - Predictability
    - How consistent is the performance over time?

Early Operating Systems: Computers Very Expensive

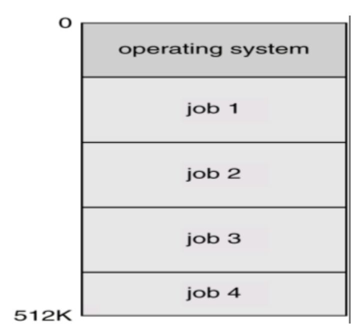


- One application at a time
  - Had complete use of hardware
  - OS was runtime library
  - Users would stand in line to use the computer
- Batch systems: multiprogramming
  - Keep CPU busy by having a queue of jobs
  - OS would load next job while current one runs
  - Users would submit jobs, and wait, and wait
  - What new OS facilities are needed?



Interactive: People Expensive

- Multiple users on computer at same time
  - Interactive performance: try to complete everyone’s tasks quickly: good response
  - As computers became cheaper, more important to optimize for user time, not computer time



Today's Operating Systems: Computers Cheap

- Smart phones
- Embedded systems
- Laptops
- Tablets
- Virtual machines
- Data center servers
- Different resources?
  - power

Tomorrow’s Operating Systems

- Giant-scale data centers
- Increasing numbers of processors per computer
- Increasing numbers of computers per user
- Very large scale storage
- Going the other way …
- Internet of things
  - New concerns?
  - Privacy!, Reliability!!