EC 440 – Introduction to Operating Systems

Orran Krieger (BU)
Larry Woodman (Red Hat)

Exam Structure & Rules

- The exam will be written open book.
- Available over a 24 hour period, with a fixed time to complete it on gradescope.
- It will focus on basic concepts and your understanding of the course material up through this lecture.
- There may be some code examples or basic questions on the test.
- What you learned in projects will also be covered.
- Pay attention to where we said in class book is wrong.
- We care about concepts, not memorization

Topics

- Introduction, role OS, key abstractions:
  - L1 & 3/Ch1/Proj 1
- Key OS services:
  - Processes, Threads, Scheduling & Deadlock:
    - L5-L7, L9-11, Ch2/6, Proj 2, 3
  - Memory Management:
    - L13-L17 + L18 (Proj 4), Ch3
  - File systems & I/O:
    - L19-L23, Ch4, Ch5.1-5.6, Ch10.6, Proj 5
- Important capabilities:
  - Virtualization & Cloud: L24, Ch7
  - Security L25, Ch9.1-9.3, 9.7

Other lectures

- L2 - Programming
- L8 - Calling code conventions
- L12 - Kernel hacking
- L23 - VFS

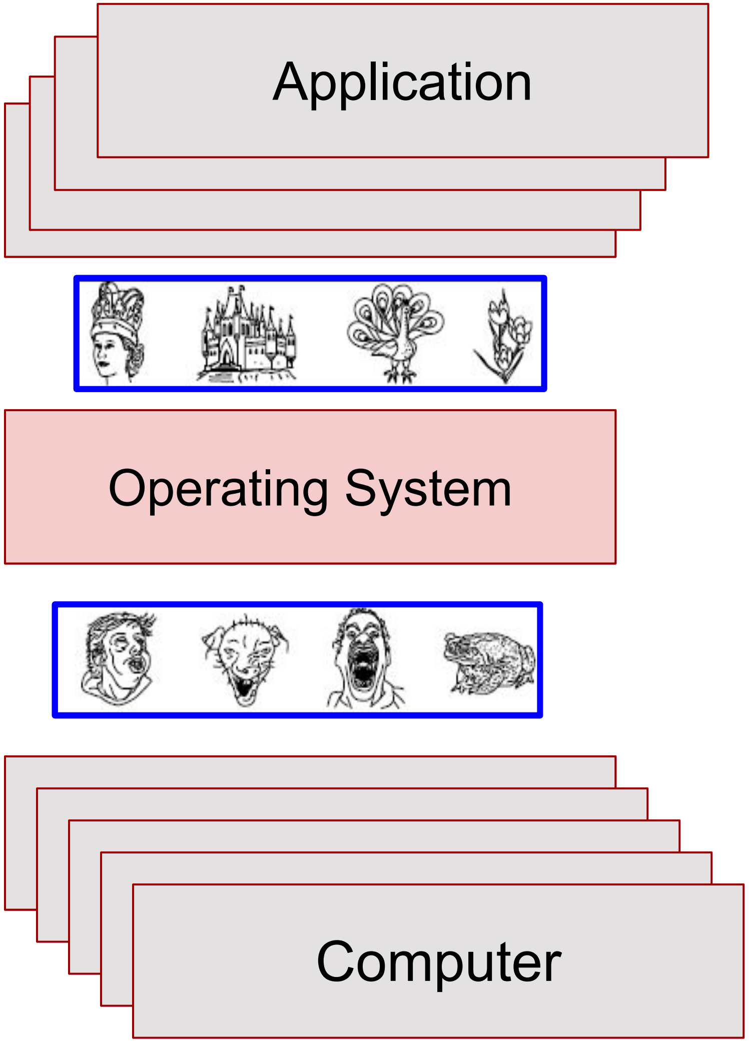
Topics

- Introduction, role OS, key abstractions:
  - L1 & 3/Ch1/Proj 1
- Key OS services:
  - Processes, Threads, Scheduling & Deadlock:
    - L5-L7, L9-11, Ch2/6, Proj 2, 3
  - Memory Management:
    - L13-L17 + L18 (Proj 4), Ch3
  - File systems & I/O:
    - L19-L23, Ch4, Ch5.1-5.6, Ch10.6, Proj 5
- Important capabilities:
  - Virtualization & Cloud: L24, Ch7
  - Security L25, Ch9.1-9.3, 9.7

Lecture 1 - Introduction

- Role of the OS
  - Provide clean abstractions over ugly hardware
  - Space/time multiplex hardware
- History and why OS are still cool
- The clean abstractions that the OS provides
  - Process - (really threads) virtual computer
  - Virtual memory - abstraction of physical memory
  - File system - abstraction over storage
  - Virtual I/O - abstraction over devices
- Why OSes are the coolest area to work

Role of the OS



Multiplexing

- creating an illusion of multiple (logical) resources from a single (physical) one

Allocation

- keep track of who has the right to use what

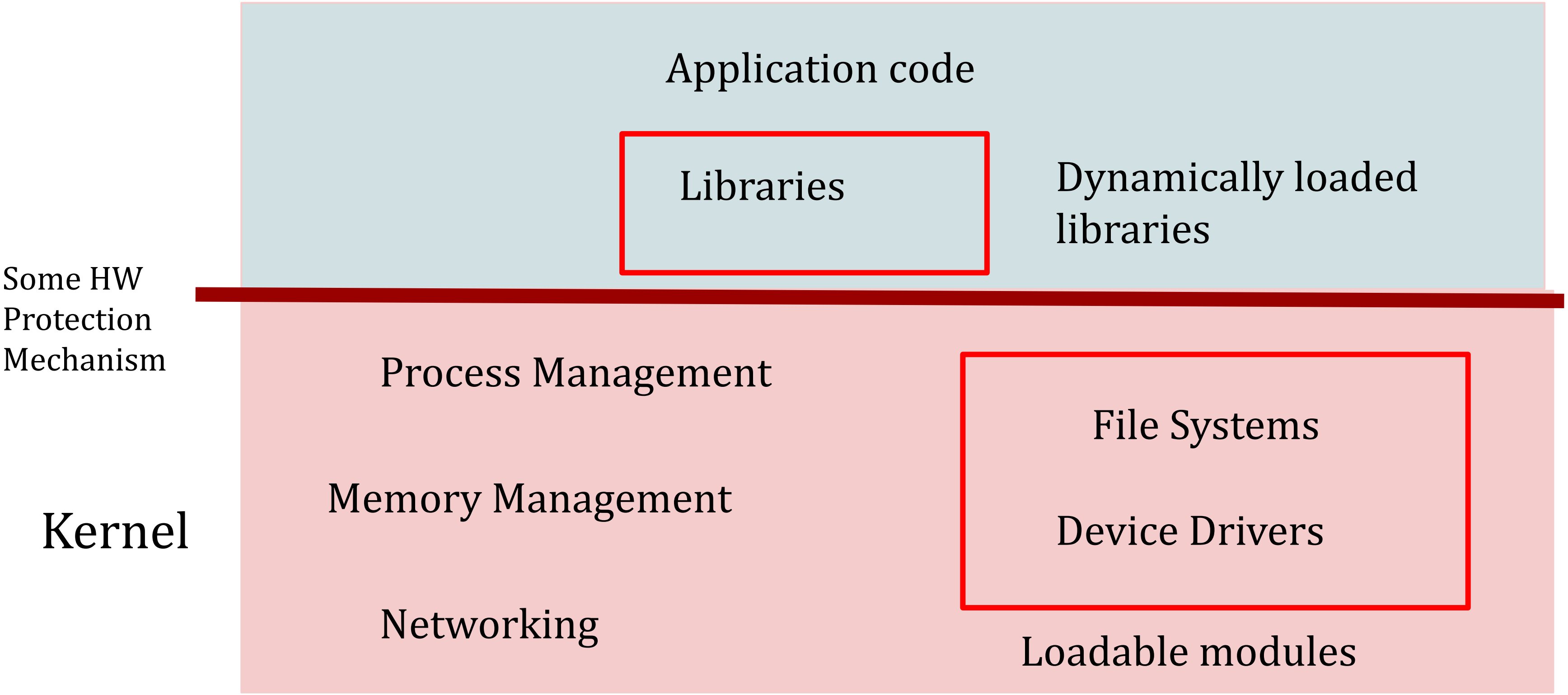
Transformation

- creating a new resource (logical) from existing resource (physical)

primarily for "ease of use"

An OS multiplexes, allocates, and transforms HW resources

Lecture 3&4 - OS structure and interfaces



Application code

Libraries

Dynamically loaded libraries

Some HW Protection Mechanism

Kernel

Process Management

Memory Management

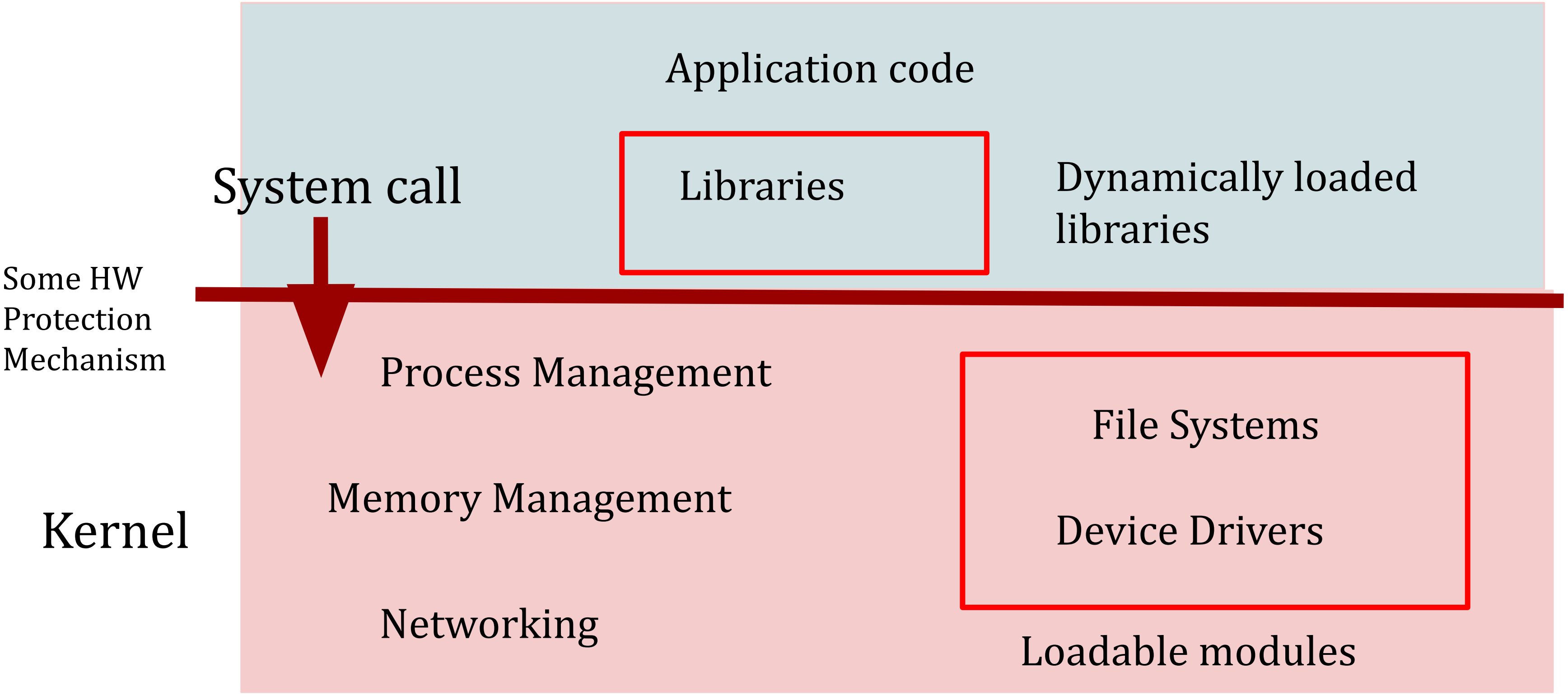
Networking

File Systems

Device Drivers

Loadable modules

Lecture 3&4 - OS structure and interfaces



Application code

System call

Libraries

Dynamically loaded libraries

Some HW Protection Mechanism

Process Management

Memory Management

Networking

Kernel

File Systems

Device Drivers

Loadable modules

Lecture 3&4 - OS structure and interfaces

- Abstractions:
  - Process and related system calls
  - Signals and related system calls
  - Memory layout and related system calls
  - Files and related system calls
  - pipes, dup and how to exec two programs that can talk to each other
- Example shell

Project 1: Shell

- A basic OS abstractions/interfaces (i.e. system calls)
  - process, files, signals
- Writing complex program and test driven development
- Idea of fork or posix spawn
  - which was easier?

Questions

- What is the role of the OS?
- Time versus space multiplexing
- What are some of the key abstractions the OS provides and how do they differ from HW
- What is the difference between static and dynamic analysis
- What is user mode versus kernel mode?
- What is a system call?
- What are some of the key system calls for x?

Topics

- Introduction, role OS, key abstractions:
  - L1 & 3/Ch1/Proj 1
- Key OS services:
  - Processes, Threads, Scheduling & Deadlock:
    - L5-L7, L9-11, Ch2/6, Proj 2, 3
  - Memory Management:
    - L13-L17 + L18 (Proj 4), Ch3
  - File systems & I/O:
    - L19-L23, Ch4, Ch5.1-5.6, Ch10.6, Proj 5
- Important capabilities:
  - Virtualization & Cloud: L24, Ch7
  - Security L25, Ch9.1-9.3, 9.7

Lecture 5 - Processes and Threads

- Process concept/virtual computer
  - Process, PCB in OS, states….
- Thread and difference from a process
  - Process/thread states
  - What gets shared
- Thread implementation:
  - user space vs kernel space

Lecture 6 - Scheduling

- Time multiplexing the CPU to different processes and threads
  - Context switching & its cost
  - Kinds of Jobs: CPU bound vs I/O bound
  - When to (re)schedule: must (e.g. process exits) vs might (e.g. clock interrupt)
  - Pre-emption
- Goals: fairness, policy, balance (i.e., keeping busy)
  - Types of systems: batch, interactive, real-time
- Some basic scheduling algorithms
  - First-come, first served
  - Shortest Job First
  - Shortest Remaining time next
  - Round Robin
  - Priority
  - Lottery
- Thread scheduling - user versus kernel, sharing quantum or not

Lecture 6 - Scheduling

- Time multiplexing the CPU to different processes and threads
  - Context switching & its cost
  - Kinds of Jobs: CPU bound vs I/O bound
  - When to (re)schedule: must (e.g. process exits) vs might (e.g. clock interrupt)
  - Pre-emption
- Goals: fairness, policy, balance (i.e., keeping busy)
  - Types of systems: batch, interactive, real-time
- Some basic scheduling algorithms
  - First-come, first served - Batch
  - Shortest Job First - Batch
  - Shortest Remaining time next - Batch
  - Round Robin - Interactive
  - Priority - Interactive/Real time
  - Lottery - Interactive/Real time
- Thread scheduling - user versus kernel, sharing quantum or not

Project 2: Threads

- Scheduling
  - Pre-emptive vs non-pre-emptive
  - You implemented round-robin. What else is there?
- Thread contexts
  - Understand their purpose
  - Know what goes in them (not specific to your implementation, but in general)

Lecture 7&9 - Synchronization

- Control access for shared memory, deal with race conditions
- Issues:
  - Critical regions & mutual exclusion
  - HW mechanisms for acquiring lock
  - Busy waiting vs sleep & wakeup
- Semaphores, Mutex, Monitors, RW locks…
- Scalability and synchronization:
  - set&test atomic instructions
  - ticketed spinlocks
  - MCS locks
  - think about cache, false sharing, NUMA locality...
  - lockless & RCU

Lecture 10 - Deadlocks

- Stuck "*each process in the set is waiting for an event that only another process in the set can cause*"
- Four conditions for deadlock:
  - mutual exclusion, hold and wait, non-preemptable, circular dependency
- Strategies for dealing with deadlocks
  - Ignore, detect & recover, avoid, prevent
  - Avoid: Safe/unsafe states - do we give resource
    - banker's algorithm
  - Prevention: invalidate one of the conditions
    - e.g., two phase locking
- Livelock and starvation

Lecture 11 - Synchronization and Deadlock in the real world

- Ostrich approach for processes
- Locking hierarchy for kernel locks
  - Try locks for when you need to violate
- Other deadlock conditions:
  - single thread - invoke something that might come back - recursive locks
  - deadlock with interrupt context/thread

Project 3: Thread Synchronization

- Synchronization primitives
  - You implemented mutex and barrier
  - Be sure to know other common techniques and how they relate.
- Blocked threads

Questions

- What is a process?, thread??, the difference???
- What are the tradeoffs between user levels and kernel level implementation of threads?
- What is scheduling? context switching??
- What are some of the scheduling algorithms?
- What is synchronization?
- What are the basic synchronization primitives
- What is a deadlock? What is livelock?
- How are deadlocks eliminated/prevented?
- What is RCU, how does it differ from reader/writer locks?
- What resources can be involved in deadlock?
- Point out deadlock in code

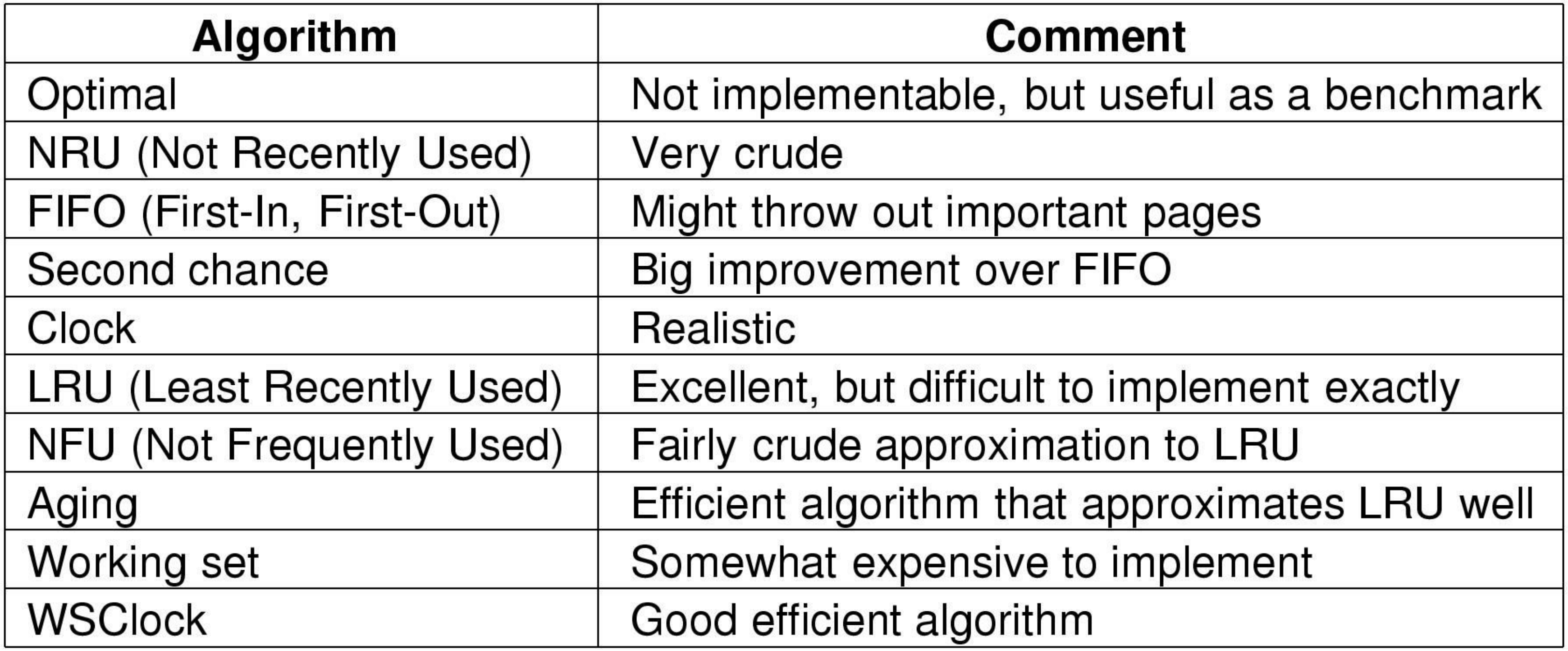
Topics

- Introduction, role OS, key abstractions:
  - L1 & 3/Ch1/Proj 1
- Key OS services:
  - Processes, Threads, Scheduling & Deadlock:
    - L5-L7, L9-11, Ch2/6, Proj 2, 3
  - Memory Management:
    - L13-L17 + L18 (Proj 4), Ch3
  - File systems & I/O:
    - L19-L23, Ch4, Ch5.1-5.6, Ch10.6, Proj 5
- Important capabilities:
  - Virtualization & Cloud: L24, Ch7
  - Security L25, Ch9.1-9.3, 9.7

Lecture 13, 14, 15, 16 & 17 - Memory Management

- Physical (ugly) versus virtual - address space
- Segmentation
- Swapping, Compaction
- MMU, TLB, page table organization:
  - multi-level - overhead...
  - inverted,
  - tagged,
  - software controlled
  - bits protection
- file system caching & anonymous memory

Page reclamation



Algorithm	Comment
Optimal	Not implementable, but useful as a benchmark
NRU (Not Recently Used)	Very crude
FIFO (First-In, First-Out)	Might throw out important pages
Second chance	Big improvement over FIFO
Clock	Realistic
LRU (Least Recently Used)	Excellent, but difficult to implement exactly
NFU (Not Frequently Used)	Fairly crude approximation to LRU
Aging	Efficient algorithm that approximates LRU well
Working set	Somewhat expensive to implement
WSClock	Good efficient algorithm

Lecture 13, 14, 15, 16 & 17 - Memory Management

- Page size tradeoffs
- x86 page table organization, superpages
- Page Fault details
- Shared memory
- Types of faults:
  - Anonymous memory & faults
  - Mapped file memory & faults
  - Copy On Write(COW) faults
- Linux: everything is real
  - address space organization
  - page cache
  - buddy allocator for de-fragmenting
  - compaction
  - core data structures

Project 4: CoW TLS

- TLS
- CoW
  - Design concerns related to p*erformance and data consistency*
- Segmentation Faults
- Page management

Questions

- What is segmentation versus paging?
- What is the purpose of MMU/TLB?
- What is most of the memory used for in a real system?
- What is the tradeoff between having larger and smaller page sizes?
- How does a super page work with a multi-level page table?
- What is good/bad about inverted pages tables, single level page table, multi-level page table?
- What is fragmentation and how does page size effect it?

Topics

- Introduction, role OS, key abstractions:
  - L1 & 3/Ch1/Proj 1
- Key OS services:
  - Processes, Threads, Scheduling & Deadlock:
    - L5-L7, L9-11, Ch2/6, Proj 2, 3
  - Memory Management:
    - L13-L17 + L18 (Proj 4), Ch3
  - File systems & I/O:
    - L19-L23, Ch4, Ch5.1-5.6, Ch10.6, Proj 5
- Important capabilities:
  - Virtualization & Cloud: L24, Ch7
  - Security L25, Ch9.1-9.3, 9.7

Lecture 19 & 20 - File Systems

- The physical (ugly) abstraction versus the virtual abstraction of file system
  - files
  - directories
- Organization meta-data: free info, blocks used, directory
- Tradeoff block size, organization free list
- File system consistency: fsck, ordered writes, soft update, logging
- Log-structured FS
- Practical example in ext2/3:
  - block groups, superblocks, inode & directory structure, journaling, extents

Lecture 21 & 22 - Devices & IO

- Device drivers role in the OS, types of them, interfaces, size of device portion of the OS.
- I/O Hardware:
  - Device controller may be a complex system: huge attack surface
  - Registers can be modified by: Port I/O, or memory mapped
  - DMA for transferring large buffers
  - Interrupts: controller, how CPU responds, saving state

Lecture 21 & 22 - Devices & IO

- I/O SW in kernel
  - Device drivers
  - What happens with programmed/interrupt driven, DMA
  - How interrupt handlers work.
  - Design considerations on Device Drivers
  - Driver APIs
  - Buffering
- Disks & SSDs:
  - Organization: tracks, cylinders, sectors, zones
  - skew to handle seek latency for sequential access and interleaving to handle sequential access within track
  - Disk arm scheduling: FIFO, SSF, Elevator
  - RAID: 5,6 common today
  - SSD pros and cons
- Clocks/keyboards/terminals

Lecture 23 - VFS layer

- It’s the VFS layer that lets multiple file systems run concurrently
- How file systems are implemented. Key issues:
  - How files/inodes refer to used blocks
  - How directories refer to inodes
  - How do you keep track of free space on the disk
  - How do you keep the meta-data consistent

Project 5: Filesystem

- inodes
  - Direct and indirect blocks
- Directory entries
- File descriptors
- Tracking used blocks

Questions

- Why do we care about file system consistency? Why is this problem specific to file systems?
- What is the major challenge introduced by a log structured file system that other file systems don't have?
- What is the tradeoff between different ways of organizing disk blocks?
- What is the tradeoff between different disk block sizes?
- What is a block group in EXT2?
- Why EXT3/4 succeed over other alternatives?
- If you need bytes 5-6 from file1.txt, how would you find that byte on disk in an inode filesystem (which data structures do you need?)
- What are the tradeoffs between levels of block indirection for inodes?

Questions

- What are 2 basic device types?
- What is the difference between Port IO & memory mapped IO?
- With memory mapped I/O, does caching help with performance?
- What is the difference between cycle stealing and burst mode for DMA?
- What is the difference between precise and imprecise interrupts?
- Why would we use interrupt driven and or DMA, rather than just synchronously polling the device?
- How much work do we normally do in interrupt handler?
- Why are drivers written to be re-entrant?
- Why are most drivers loadable modules?
- Why would we use Elevator over something fair like FIFO? What's the problem with SSF?
- How do SSDs compare to HDDs ? cost, performance, lifespan
- How do the data structures relate?

Topics

- Introduction, role OS, key abstractions:
  - L1 & 3/Ch1/Proj 1
- Key OS services:
  - Processes, Threads, Scheduling & Deadlock:
    - L5-L7, L9-11, Ch2/6, Proj 2, 3
  - Memory Management:
    - L13-L17 + L18 (Proj 4), Ch3
  - File systems & I/O:
    - L19-L23, Ch4, Ch5.1-5.6, Ch10.6, Proj 5
- Important capabilities:
  - Virtualization & Cloud: L24, Ch7
  - Security L25, Ch9.1-9.3, 9.7

Lecture 24 - Virtualization & Cloud

- Virtualization & Cloud Computing
- What virtualization is and what are the cool things you can do with it.
- Virtualization goes back to the 1960s; unfortunately, x86 is non virtualizable
- Basic idea:
  - Run user programs directly, simulate guest OS.
  - Simulate devices behind memory mapped I/O
  - Shadow page tables - guest updates, OS maps guest physical to real physical
- Original optimization techniques:
  - Binary translation of kernel
  - Paravirtualized devices
- Add new HW:
  - new privilege mode for hypervisor
  - extra level of page table - guest physical to real physical
  - virtual functions/SRIOV
- Live migration
- What is cloud computing and why it is possible today: IaaS, PaaS, SaaS
- Motivation for using Cloud:
  - Price (for some), Elasticity, Services
- Transformation of applications
- Some of the concerns about cloud computing
- Impact on operating systems

Questions

- What are the cool things people have done with virtualization?
- Why is this hard?
- What is paravirtualization?
- How simulate, and how make fast before/after HW support: processor, I/O devices, page mappings
- How do you move VMs between computers?
- What is 99% tail latency and why important?
- What are some of the reasons why Cloud computing is transforming computing?

Lecture 25 - Security

- Threat modeling
- System understanding
- Threat categorization
- Trusted computing base
- Protection domains
- Unix protection model
  - users & groups
  - filesystem modes
  - Access control lists
- Code modifications and hacks
- Address space randomization

Questions?

- What is a thread and what are we trying to protect?
- What are the privilege boundaries?
- What do we mean by **STRIDE**?
  - **S**poofing, **T**ampering, **R**epudiation, **I**nformation Disclosure, **D**enial of Service, **E**levation of Privilege
- What is a Trusted Computing Base?
- What is a Protection Domain
- What is the Unix/Linux protection model
  - users & groups
  - filesystem modes
  - Extended attributes & Access control lists
- How do we control/limit code modifications and hacks
- Why do we use Address Space Randomization
- What are side channel attacks?

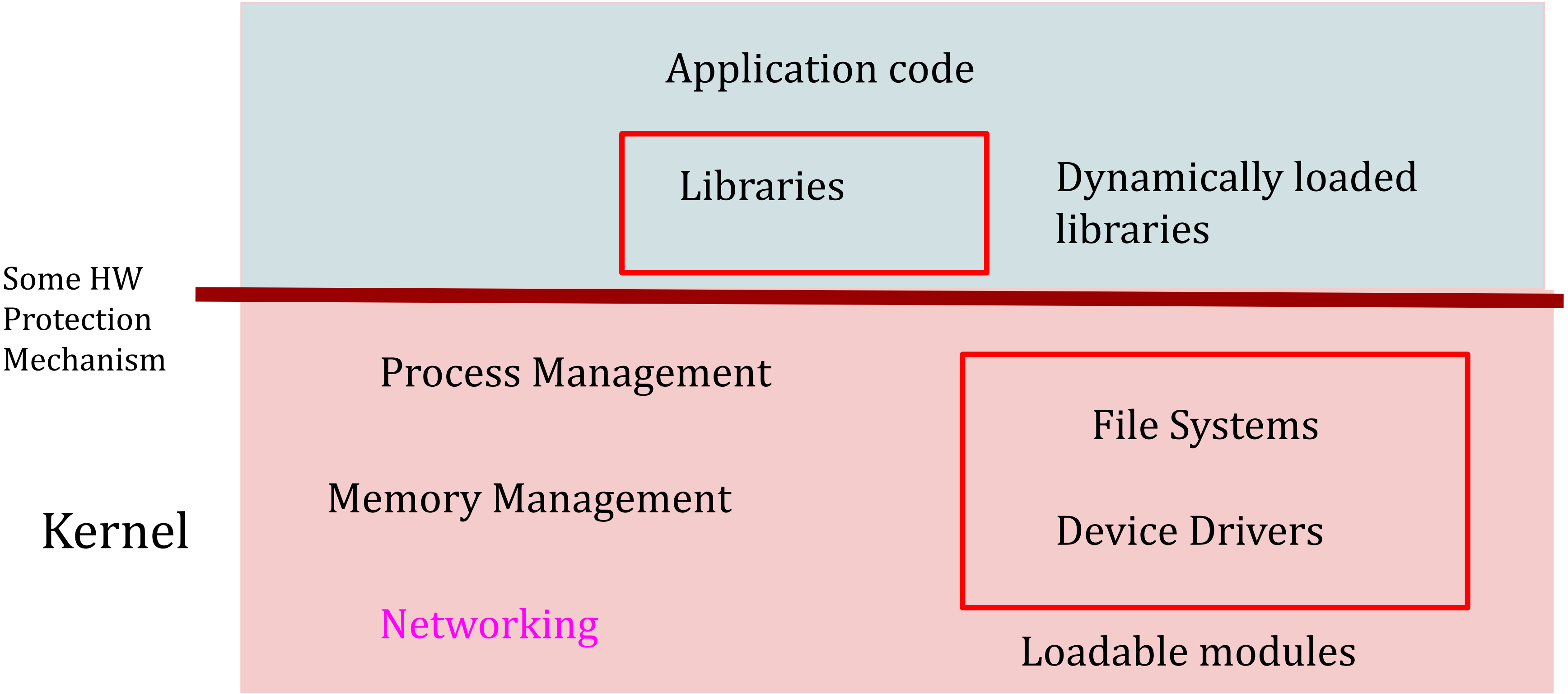
Other lectures

- L2 - revision control, testing, dynamic/static analysis, makefiles
- L8 - calling code conventions and HW2
- L12 - kernel hacking
  - separate compilation, example tests and deadlock, how linker works...
  - building linux kernel
- L23 - looking through code simple Linux file system and the VFS layers that interact with it

Other lecture questions

- What does a sanitizer do?
- Why do you divide program into multiple files?
- Develop a unit test to do X
- How does a linker resolve undefined symbols?

Overview



Application code

Libraries

Dynamically loaded libraries

Some HW Protection Mechanism

Process Management

Memory Management

Networking

File Systems

Device Drivers

Loadable modules

Kernel

What do we hope you got out of this...

- The role of operating system, its key services, some of the theory.
- How these capabilities are realized in modern OS - Linux
- Experience writing complicated system software and best practices
- The role of Open Source in platforms...

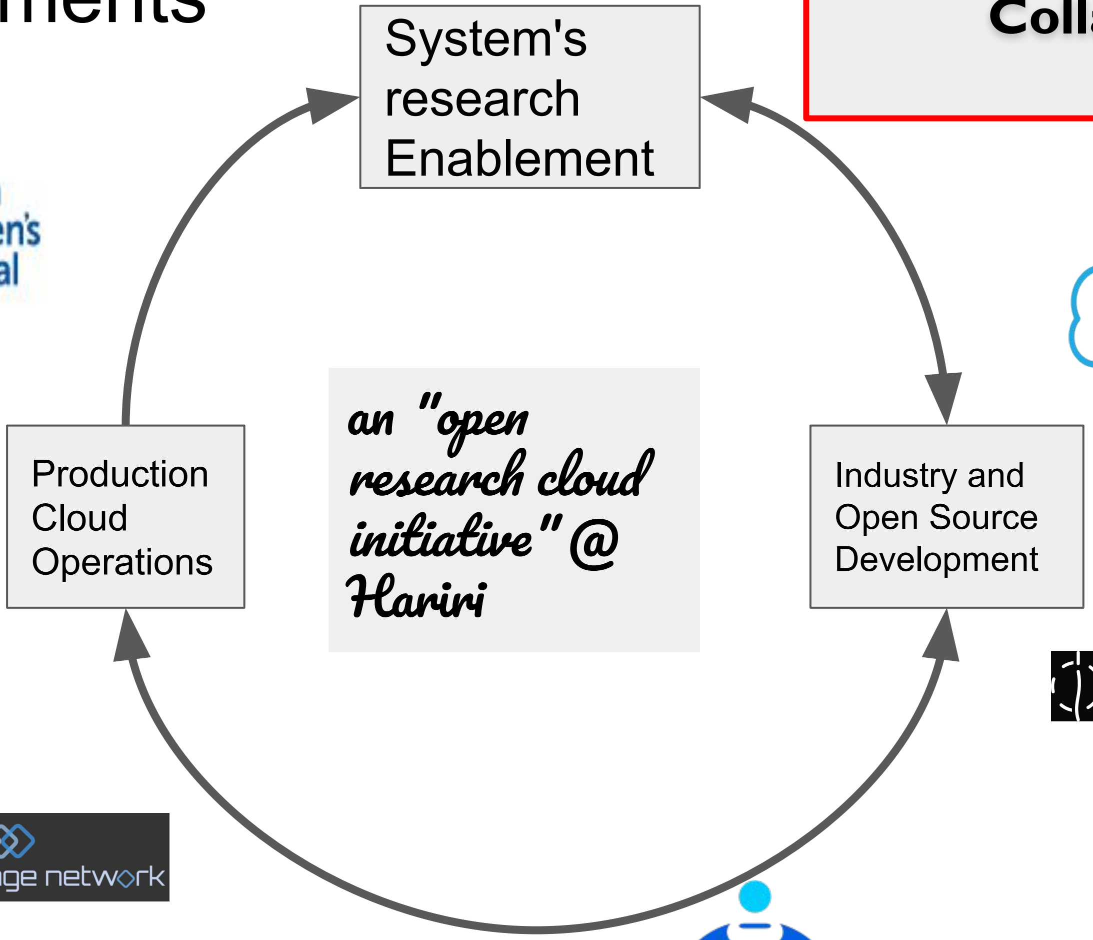
- Open source is not enough anymore...

Background

Recent Announcements

Open Cloud Testbed
Exploring Next-Generation Cloud Platforms

Red Hat | BOSTON UNIVERSITY
Collaboratory


System's research Enablement
Industry and Open Source Development
Production Cloud Operations
an "open research cloud initiative"@ Hariri

Boston Children's Hospital

MOC

NERC

NESE

OPERATE FIRST

open storage network

OpenInfra Labs

Some of the cool OS projects Collaboratory

- Function as a service
- Linux as a unikernel
- OS policy services
- Fuzzing for security
- Memory BW control
- Elastic Secure Infrastructure
- Partitioning hypervisor
- OS for FPGA

Opportunities

- Internships in operations, operating systems, cloud software, research
- Student ambassador program
- Path to research

That’s all folks!