Winter 2025

EECS 482: Introduction to Operating Systems

Lecture 25: Review

Prof. Ryan Huang

Administration

Final exam

- April 28th (next Monday) 7pm - 9pm

Scope

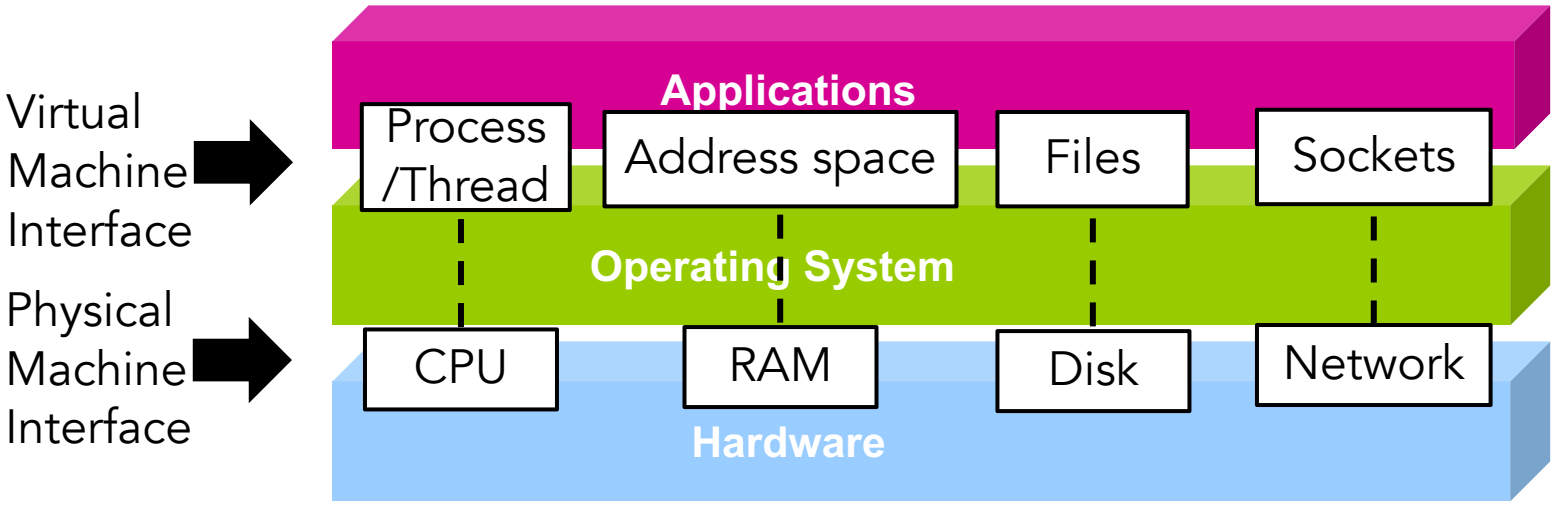
- Lecture materials
- Content in labs, projects

In-person, closed-books, on-paper

- Cannot use notes, slides, books, electronic devices

Piazza post @659 on preparing for the exam

~~Operating~~ system roles



Abstraction

Management of shared resources

~~Operating~~ system goals

Convenience

- Abstraction; removal of resource limits

Performance

- Throughput, latency, deadlines

Fairness

- Variance, guarantees

Reliability

- Correctness in the presence of failure

Security

- Correctness in the presence of attacks

~~Operating~~ system techniques

- Abstraction
- Concurrency
- Atomicity
- Indirection
- Caching
- Prediction
- Naming and translation
- Scheduling/ordering
- Deferring (or anticipating) work
- Resource trading

The Big Ideas

Software abstractions

- A "better" version of hardware reality

Three "flavors" of abstraction

- Limited resources appear more numerous
- Distinct resources appear unified
- Failures appear not to happen

Limited Resources

Examples

- One CPU → many threads
- Small physical memory → larger virtual memory
- One network interface → many interfaces

Techniques

- Time-sliced multiplexing: CPU, Memory
- Higher-level mechanisms: RMW -> Mesa Monitors
- Logical division: IP Address -> IP, Port pairs

Distinct Resources

Examples

- Multiple disks → one file system
- Distributed machines → one computation
- Replicated objects → one object

Techniques

- Stitching entities together in a larger structure
- Active messaging between distinct parts
- Concurrency control

Failures

Examples

- Atomicity in persistent storage
- Reliability in network transmissions
- RAID in persistent storage

Techniques

- Ordered writes and/or transactions
- Retransmit lost or corrupted network messages
- Replication with built-in redundancy

The Big Ideas

Five tools of software systems

- Indirection: pointer-to-thing rather than thing
- Caching: storage hierarchy: speed vs. size/cost
- Hashing: probabilistically represent w/summaries
- Replication: copies for performance/availability
- ~~Encryption: make something hard to know~~

Indirection and Hashing

Indirection is ubiquitous

- Large page tables -> multi-level
- Large inodes -> multi-level
- File system structures
- Page Table Base Register
- Thread Control Block (stack pointer)

Hashing:

- Checksum of a network packet

Caching and Replication

Caching: one authoritative copy:

- Local file system (uses physical memory)
- AFS: file system operations to local copy
- One "authoritative" copy at a time
- P3: authoritative copy moves between disk/memory

Replication: multiple authoritative copies

- RAID (mirroring and RAID-5, but not striping)
- Consistency, Availability, Partition-tolerance
- AFS: a CP system, read-replicas
- Coda: an AP system, read-write replicas

The Big Ideas

Respect Invariants and Dependencies

Build in Layers

Exploit Access Patterns

Amortize Overhead

Avoid Performance Extremes

Go Beyond Testing

Separate Concerns

Invariants and Dependencies

THE #2 THING I want you to take from 482

Invariants: rules about state

- Data structure: when is it safe to expose updates?
- Interface: how do you reason about correctness?

Dependencies: state connecting behavior

- Explicit: use a read of some state in a later write
- Implicit: value of a read influences program flow

Build in Layers

Synchronization

- Mesa Monitors out of atomic read-modify-write
- Reader-writer locks out of Mesa Monitors

Storage

- Generic block-device abstracts spinning/solid-state
- File system abstracts generic block device
- Transactions simplify multi-step updates

Build in Layers

Network

- Physical/Media Access: local connectivity
- Internet: remote connectivity
- Transport: better abstractions
- Application: application semantics

Something goes in a lower layer if and only if

- It makes the higher layer easier to build
- There is a performance advantage for doing so

End-to-End Arguments in System Design

Exploit Access Patterns

Memory and file access patterns:

- Temporal locality
- Spatial locality

Additional file access patterns:

- Sequential more common than random
- Reads more common than writes
- Writes are often short-lived
- Write sharing is rare

Exploit Access Patterns

Examples

- Writes short-lived: defer in hopes of avoiding
- Temporal locality: LRU and Clock
- Spatial locality: disk layout policy
- Infrequent write sharing: optimistic concurrency

Patterns in a given workload might be different

- Databases look "random" but ordered by key
- Human preferences are heavy-tailed

Amortize Overhead

Overhead vs. productive work

- Disk positioning vs. data transfer
- Trap to the kernel vs. work done in the system call

Keep overhead proportional to productive work

- Add overhead only when necessary
- But take advantage of room when you can

Amortize Overhead

Memory translation vs. file system syscalls

- Translation for memory accesses is expensive
- But CPU time to do FS mapping (translation) is cheap compared to the disk I/O time

Disk operations

- Re-ordering operations minimizes overhead
- Pre-fetching can read data very cheaply

Network operations

- Work added to necessary round trips also cheap

Avoid Performance Extremes

Performance can fall off a cliff

- Working set just under vs. just over memory size
- Response time at the expense of throughput
- Recovery cascades (revalidating a big cache)

Response time matters, but only up to a point

- Human response time ~100ms
- Network response time ~10ms

Go Beyond Testing

Testing is helpful, but not enough

- Cannot generate all interesting interleavings
- Tends to find only the easy bugs

Instead, must also reason about correctness

- Specifications and invariants are critical!

Going slow when writing is often helpful

Separate Concerns

Functional abstraction:

- Specification vs. implementation

Exceptions:

- Common case vs. error handling

Invariants:

- Local inspection implies global correctness

The cardinal rule of computer science: #1 THING

I hope you have…

Learned OS concepts and techniques

Become better at designing, writing, testing, and debugging hard systems

Become more adept at C++

Enjoyed the class

And Finally….

And Finally….

THANK YOU

for joining me in 482

Best wishes on the final and your career!