Introduction

CS 4410
Operating Systems

[R. Agarwal, L. Alvisi, A. Bracy, M. George, F. B. Schneider, E. G. Sirer, R. van Renesse]
What an OS does

- OS is an intermediary between programs and hardware.
- OS creates an environment to execute programs in a convenient and efficient manner:
  - allocates resources (CPU and storage)
  - controls programs
    - cooperation (sharing and synchronization)
    - isolation (protection and resource management)
Ways to view an OS

• **Services** it provides to programs

• **Components** implementing those services
  – internal design and implementation
    • Real hardware is ugly.
  – interactions
Why Study OS?

Learn solutions to problems arising in all systems:

- Resource sharing (scheduling)
- Cooperation (concurrent programming: communication, synchronization)
- System structure (abstractions, interfaces)
How designing an OS differs from designing a program.

• **Measure of success**: OS concerned with extensibility, security, reliability, …

• **External interface**: OS more complicated and subject to change. Eg I/O devices.

• **Internal structure**: OS must pick boundaries between functions:
  - RISC vs CISC
  - C-time vs run-time (cf Java)
  - Heap with garbage collection vs stack allocation

• **Structuring techniques**: OS employs
  - modules, layers, client-server, event-handler, transaction systems vs Programs (I)
Systems vs Programs (II)

How designing an OS differs from designing a program.

- OS must bridge mismatched performance characteristics.
  - Registers vs RAM vs Disk
  - Desktop processor vs Cloud
End to End Argument: Trade-offs

Copy file F from cpu 1 to cpu 2 despite possible transients. Copy proceeds one block b1, b2, b3, … at a time. Packets size ≠ block size.
Copy file F from cpu 1 to cpu 2 despite possible transients. Copy proceeds one block b1, b2, b3, … at a time. Packets size ≠ block size.

Soln 1: Checksum F, send all blocks, send checksum.
End to End Argument: Trade-offs

Copy file F from cpu 1 to cpu 2 despite possible transients. Copy proceeds one block b1, b2, b3, … at a time. Packets size ≠ block size.

Soln 1: Checksum F, send all blocks, send checksum.

Soln 2: Checksum sent with each packet on network.
Copy file F from cpu 1 to cpu 2 despite possible transients. Copy proceeds one block b1, b2, b3, … at a time. Packets size ≠ block size.

Soln 1: Checksum F, send all blocks, send checksum.

Soln 2: Checksum sent with each packet on network.

Soln 3: Checksum successive file prefixes. So sends…
• B1 chk(b1), b2 chk(b1 b2), b3 chk(b1, b2, b3)
What makes systems complex?

**Emergent properties**: Evident only when components are combined.

Example: Millennium Bridge (London)
What makes systems complex?

**Propagation of Effects**: When small changes have disproportionate effects.

Examples:

- Power failures in power grid
- Change auto tire size from 13” to 15”
- Boeing 737 max 8 design
  - 4th generation of 737: new engines, same exit doors.
What makes systems complex?

**Incommensurate Scaling**: Different parts follow different scaling rules.

Examples:

- Height limits on skyscrapers
- Size limits on railroad trains and cargo ships
  - Horizon distance is linear in size of object
  - Stopping distance is proportional to object volume.
How to Manage Complexity

• **Modularity**: Good modularity minimizes connections between components.

• **Abstraction**: Separate interface from internals; separate specification from implementation

• **Hierarchy**: Constrains interactions so easier to understand.
  • L levels with N components / level
  • \((N \times L)^2\) vs \(L \times N^2\) interactions
OS has many roles

Referee
• Manages shared resources: CPU, memory, disks, networks, displays, cameras, etc.

Illusionist
• Look! Infinite memory! Your own private processor!

Glue
• Offers set of common services (e.g., UI routines)
• Separates apps from I/O devices
OS as Referee

Resource allocation
• Multiple concurrent tasks, how does OS decide who gets how much?

Isolation
• A faulty app should not disrupt other apps or OS
• OS must export less than full power of underlying hardware

Communication/Coordination
• Apps need to coordinate and share state
OS as Illusionist (1)

Virtualization: Resources seem present but aren’t.
- processor, memory, screen space, disk, network
- the entire computer:
  - fooling the illusionist itself!
  - ease of debugging, portability, isolation

Operating System (VMM)
OS as Illusionist (2)

**Abstraction**: Enables new assumptions for clients

- **Atomic operations**
  - HW guarantees atomicity at word level
    - what happens during concurrent updates to complex data structures?
    - what if computer crashes during a block write?
  - At the hardware level, packets are lost...

- Reliable communication channels
OS as Glue

Simplify app design and facilitate sharing due to:
  • send/receive of byte streams
  • read/write files
  • pass messages
  • share memory
  • UI

Decouples HW and app development
Issues in OS Design

• **Structure**: how is the OS organized?
• **Concurrency**: how are parallel activities created and controlled?
• **Sharing**: how are resources shared?
• **Naming**: how are resources named by users?
• **Protection**: how are distrusting parties protected from each other?
• **Security**: how to authenticate, authorize, and ensure privacy?
• **Performance**: how to make it fast?
Issues in OS Design

- **Reliability**: how do we deal with failures??
- **Portability**: how to write once, run anywhere?
- **Extensibility**: how do we add new features?
- **Communication**: how do we exchange information?
- **Scale**: what happens as demands increase?
- **Persistence**: how do we make information outlast the processes that created it?
- **Accounting**: who pays the bill and how do we control resource usage?