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, 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 difficult to use directly
Why Study OS?

Learn solutions to problems arising in all systems:

- Resource sharing (scheduling)
- Cooperation (concurrent programming: communication, synchronization)
- System structure (abstractions, interfaces)
Systems vs Programs (I)

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. E.g. I/O devices

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

How designing an OS differs from designing a program

*OS must bridge mismatched performance characteristics*

- Registers vs RAM vs Disk
- Phone vs Laptop vs Server
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”
  - kills suspension
- Boeing 737 max 8 design
  - 4\(^{th}\) generation of 737
  - larger engines, mounted further forward and higher
  - pushes up nose of jet
  - compensated by sensors and software…
What makes systems complex?

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

Examples:

- Height limits on skyscrapers
- Size limits on cargo ships
  - Horizon distance is linear in size of object
  - Stopping distance is proportional to object volume
- Giant in Jack and the Beanstalk
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
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
- 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 share state
Virtualization: Resources seem present but aren’t
- processor, memory, screen space, disk, network
- the entire computer (*virtual machine*):
  - fooling the illusionist itself!
  - ease of debugging, portability, isolation
Abstraction: Enables new assumptions for clients

• Atomic operations
  • HW provides atomicity at word level
    – what happens during concurrent updates to complex data structures?
    – what if computer crashes during a file write?

• Reliable communication channels
  • At the hardware level, packets are lost…
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?