CS 134: Operating Systems
Definitions, Abstractions, Taxonomies, Early History
Overview

What Is an OS?

History

Hardware
Class Exercise: Devise three separate definitions. Discuss.
It's A Programmer's Toolkit

Provide useful functionality to programs:

- Prevent duplicated work
- Promote reuse
What Is an OS?

It's a Control Program

Provide the rules for the how the machine will operate:

- Control the operation of the I/O devices
- Ensure smooth running of the machine
What Is an OS?

It’s an Abstraction Layer

Make the machine “nicer”, easier to program, higher level...

- Hide some of the idiosyncrasies of the machine
- Provide functionality the underlying machine doesn’t have

User

Application

Operating System

Hardware
It’s an Abstraction Layer

Make the machine “nicer”, easier to program, higher level...

- Hide some of the idiosyncrasies of the machine
- Provide functionality the underlying machine doesn’t have
OS provides an *environment*
This environment can be seen as a “new machine”...

Hardware
  + Core OS
  + OS Libraries
  + OS Utilities
  + Application

—Physical machine
—Virtual machine
—Virtual machine
—Virtual machine
What Is an OS?

It’s a Protection Layer

Make the machine more robust—less scope for a bug to have devastating consequences

- OS does everything programs can’t be trusted to do
- OS makes programs play nice with others
OS provides the mechanisms to enforce various policies
OS provides the mechanisms to enforce various policies

Class Exercise: Examples?
The operating system manages physical resources:

- Processor
- Memory
- Storage devices
- Network devices
- etc...
The operating system manages virtual resources:

- Processes
- Files
- Users
- Network connections
- Windows
  etc...
Many operating systems are sold by commercial companies

- Market vs. technical considerations
- The operating system is what comes in the box marked “operating system”
What are the (user-level) abstractions we’d expect to find in a modern OS?
What Is an OS?

Fundamental Abstractions

Include...

- System calls
- Processes
  - Threads
  - Address spaces
- Files
  - Files
  - Directories
  - Filesystems
- Events
  - Asynchronous
  - Synchronous
- IPC Mechanisms
  - Semaphores
  - Mutexes
  - Condition Variables
- Communications channels
  - Pipelines
  - Network connections
- Users
- (Remote) Hosts
What are the “resources” that an operating system manages?
What Is an OS?

It’s a Resource Manager

The operating system manages physical resources:

- Processor
- Memory
- Storage devices
- Network devices

etc...
The operating system manages virtual resources:

- Processes
- Files
- Users
- Network connections
- Windows

etc...
Different computer systems ask different things from their OS
Different computer systems ask different things from their OS

Class Exercise: Give some dimensions across which computer systems vary
Different computer systems ask different things from their OS:

- Special-purpose ↔ General-purpose
- Single-user ↔ Multi-user
- Non–Resource-sharing ↔ Resource sharing
- Single processor ↔ Multiprocessor
- Stand alone ↔ Networked
- Centralized ↔ Distributed
- Batch ↔ Interactive
- Deadline-free ↔ Real-time
- Insecure ↔ Secure
- Symmetric ↔ Asymmetric
- Simple ↔ Complex
- Small ↔ Large
- Inexpensive ↔ Expensive
- etc.
1950s—large complex machines
- Operated directly from a console
- Used interactively by a single user
- Ran one program at a time (uniprogramming)
- Read data from paper tape, punched cards, or toggle switches

OS? Maybe a library containing code to work the I/O devices was useful.
Simple Batch Systems

Provide better use of resources:

- Users access computer indirectly
- Programs and input (jobs) taken from a batch queue
- Computer has a human operator to feed it jobs

Need to:

- Manage the jobs:
- Protect the next program from the previous program
SPOOLing Batch Systems

Provide better use of resources—buffer input and output
- Read-ahead input from disk/tape
- Write-behind output to disk/tape

Class Exercise: Why does buffering improve performance? Does buffering always improve performance? (What assumptions are we making?)
SPOOLing Batch Systems

Provide better use of resources—buffer input and output

- Read-ahead input from disk/tape
- Write-behind output to disk/tape

Class Exercise: Why does buffering improve performance?
Does buffering *always* improve performance?
(What assumptions are we making?)
Multiprogrammed Batch Systems

Provide better use of resources—multiplex the processor:

- Run multiple independent programs at once
- Switch to another program when running program waits for I/O

More work for OS. More complex management of

- I/O
- Memory
- Processor
Time-Sharing Systems

Provide better environment for users—multiplex the processor between users:

▶ Run multiple independent programs at once
▶ Switch between users rapidly
  ▶ Illusion of having the machine’s full attention

Yet more complexity for OS:
As new, “smaller” hardware appears, it tends to repeat this evolution

- Mini computers
- Personal computers
- PDAs
- Embedded systems
  - Cell phones
  - MP3 Players
  - Cameras, etc.
Need to perform computation!

- Memory contains program instructions and program data
- Processor registers maintain processor state. Registers include:
  - General purpose (address & data) registers
  - Instruction pointer (aka program counter)
  - Stack pointer(s)
  - Control and status registers
Computer Hardware—I/O Devices

Need to communicate with the world!

- I/O devices and CPU execute concurrently
- Devices have hardware controllers
  - Handles devices of a particular device type
  - Some level of autonomy
  - Local buffer
- I/O is from the device to local buffer of controller
Programmed I/O

After I/O starts, control returns to user program only on I/O completion

- CPU waits until I/O completes.
- At most one I/O request is outstanding at a time
  - No simultaneous I/O processing
Polling == Querying the I/O device
Separate I/O into two parts:
  ▶ Initiation
  ▶ Polling
Advantages?
Separate I/O into two parts:
- Initiation
- Asynchronous notification
User-level code almost always uses “programmed I/O” (e.g. read and write on a file)

Why?
CPU needs another feature...

Start → Fetch Instruction → Execute Instruction

- No
- Interrupts Enabled? No → Interrupt? No → Jump to Handler
- Yes → Interrupt? No → Save State
- Yes → Interrupt? Yes → Jump to Handler
Handling an Interrupt

What needs to happen:

- Save state
  - All registers
  - Switch stacks?
- Find out what interrupt was...  
  - Polling
  - Vectored interrupts
Types of Interrupts

Various types

▶ Software exception (also called a trap)
▶ Timer
▶ I/O
▶ Hardware failure

A modern operating system is *interrupt driven*
We’ve covered interrupts, but hardware has other cool features, including:

- Caches
- Memory management
- Protection

We’ll come back to hardware as we address these topics.