What is an Operating System?

- An operating system is
  - a government: legislates/enforces proper use of system resources
  - a resource allocator
  - a control program: prevents errors and improper use

No OS (1950s)

- Bare machine, single user
  - a button which executed a bootstrap loader
  - programming in octal, no assembler or compiler
  - input via paper tape, or punched cards
- Operator
  - user
  - machine operator
- Main perceived problems
  - human actions were slow
  - inefficient use of expensive hardware
Batch Systems (early 1960s)

- Reduce set-up time by batching jobs with similar requirements
- One user at a time
- I/O devices
  - punched cards, magnetic tape, text on paper
- Operation
  - load jobs (punched cards) onto magnetic tape
  - process jobs on tape serially
  - output to tape
  - print output tape
- Main perceived problem
  - turn-around time: up to two days
  - CPU often underutilized
    - most of the time spent reading and writing from tape

Innovation 1: Resident Monitor

- Automatic job sequencing
  - use of control cards
- Job Control Language
  - commands
    - mount this tape
    - compile
    - run
- OSes begin to be important
  - IBM: Fortran Monitor System
- Main perceived problems:
  - turn-around time
  - inefficient use of expensive hardware
  - CPU is still mostly idle

Innovation 2: Spooling

- Use of disks to buffer input/output to tapes
  - disks are random-access I/O devices
- Overlapped I/O and computation
  - one job’s I/O can be overlapped with another’s computation
  - need for interrupts
- Independent I/O controllers
  - CPU: starts i/o operation; continues computation
  - Controller: does I/O; interrupts CPU
- Initially off-line spooling, later on-line

Multiprogramming Systems (1960s)

- Many programs simultaneously in memory
  - objective: to keep CPU busy
  - OS switches between user processes
- How to ensure that these programs do not interfere with each other?
- Hardware innovations
  - memory protection
  - privileged instructions
Time Sharing and Interactive Systems

- Originally proposed by Strachey ~1960
  - programs could interact with user
- Programs
  - could wait for I/O for an arbitrary time
    - CPU switched to another job (CPU Scheduling)
  - however, resident jobs took up valuable memory
    - needed to be swapped out to disk
    - virtual memory
- OS research in 60s
  - CTSS, MULTICS at MIT
  - Atlas (spooling, demand paging) at Manchester U

OS Requirements (late 1960s)

- Multiprogramming
  - memory allocation & protection
  - I/O operations were responsibility of OS
- Interactive systems
  - scheduling issues
  - swapping, or virtual memory
- Users wanted permanent files
  - hierarchical directory systems
- Identification of core components of an OS

OSes in the 1960s

- Increased in size and complexity
- Were not well understood
  - IBM: OS/360
  - CDC: Sipros, Chippewa, NOS
- Much money was spent and wasted
- WHY ???

UNIX (early 1970s)

- Originally developed at Bell Labs for the PDP-7
  - Ken Thompson
  - Dennis Ritchie
- Smaller & simpler
  - process spawn and control
    - each command creates a new process
  - simple inter-process communication
  - command interpreter not built in: runs as a user process
  - files were streams of bytes
  - hierarchical file system
- Advantages
  - written in a high-level language
  - distributed in source form
  - powerful OS primitives on an inexpensive platform
Personal Computers (1980s)

- Originally
  - single user
  - simplified OSs
    - no memory protection
    - MS-DOS
- Now run sophisticated OSes
  - Windows NT, Linux

Windowing Systems (1980s)

- Originally based on work at Xerox Parc
- Popularized by the Macintosh
- Characterized by
  - graphical interface
  - mouse control

Networks of Workstations (1990s)

- High-speed network connections
- Local & world-wide
- Client-server systems
  - file systems
  - remote windowing systems
- Support a variety of node OSes
  - Unix, Windows NT

The Future

- Distributed systems
  - network is invisible
- Micro-kernel and extensible OSes
  - support multiple OS flavors
    - e.g., Mach, Amoeba, Windows NT
  - allow insertion of application-specific functionality
- Embedded devices and network computers
  - computer runs a very thin OS (Java Virtual Machine)

- Unfortunately, we will not have time to talk about these in this course
  - but, opportunities in my research group
    http://www.cs.nyu.edu/pdsg
Summary: History of OSes

Outline

* Announcements
  - Text book should be available in stores today
  - Accounts on courses1 and courses2
  - Questions?

* History of operating systems
* Computer-system structures
  - I/O structures
  - storage structures and hierarchy
  - memory, secondary storage, tape
  - hardware support for protection
* Operating-system structures
  - the functional view (what it does)
  - the components view (designer’s view)
  - the services view (user/programmers’ view)
  - the structure view (implementers’ view)

[Silberschatz/Galvin: Chapters 1-3]

The Hardware of a Modern Computer System

Computer-System Structures (1): Input/Output

* Device controllers
  - special-purpose processors
  - local buffer storage
  - controllers contain registers
    - control (write-only)
    - data (read-write)
    - status (read-only)

* How do the CPU and the device controllers communicate?
  - instructions
    - read/write I/O addresses
      - e.g., video memory
    - registers in I/O controllers addressed as memory
  - interrupts
    - device controllers can interrupt the CPU
Interrupt Handling

- Interrupts are “asynchronous requests for service”
  - signal on a wire connecting the devices

- When an interrupt occurs, the CPU
  - preserves the present CPU state
    - this includes its registers and program counter
  - forces execution of code at an interrupt address
    - this may be dependent on the source of the interrupt
    - typically, table-driven: a table stores addresses of interrupt handlers
      - indexed by the interrupt number (ISR)
  - interrupt handlers
    - perform the requested service
    - selective processing of other interrupts
      - e.g., only higher-priority interrupts may be handled
  - resumes the interrupted program

- Most modern OSes are interrupt-driven

Interrupt Handling (contd.)

Interrupts vs. Traps

- Interrupts
  - asynchronous
  - triggered by devices outside the CPU

- Traps
  - synchronous
  - triggered by special instructions in user program

- Other than the above, handling of interrupts and traps is identical
- Traps are the hardware mechanism for implementing system calls

I/O Operation

- Two approaches: Synchronous and Asynchronous

  ![Diagram of I/O Operation]

  - Problem with the above schemes: CPU handles all I/O
    - it can spend all its time doing interrupt processing
      - disk I/O, network I/O, video I/O
Solution: Direct Memory Access (DMA)

- The main idea: add a special device to “intervene” between the device controller and the system’s memory

- Operation
  - the CPU tells this DMA controller
    - the “chunk” size to be transferred
      - e.g., 128 - 4096 bytes (sectors) for disks
    - the starting address in memory where this chunk ought to be stored
  - the DMA controller
    - accesses the secondary device via its controller
    - transfers the chunk from the device to system memory (and vice-versa)

- Benefit: Interrupts are now less frequent
  - at the level of chunks of data: only to indicate completion
  - hence, CPU can do a lot of work between interrupts

Memory-mapped I/O

- Traditionally, the CPU could directly access only main memory
  - all other devices are handled via controllers
  - accessed using special I/O instructions

- In most recent systems
  - controller’s registers are mapped into RAM space
    - results in uniform treatment of the I/O devices
      - can be handled via memory management procedures
      - all addressing is to RAM space
      - DMA access, interrupt handling, polling, …
  - controller’s buffers are mapped into RAM space
    - makes sense if the I/O is to a device that is particularly fast
      - e.g., a CRT screen where each pixel is an addressable location in RAM

Computer-System Structures (2): Storage

- Primary storage: Main memory (volatile)
  - accessed directly using load/store instructions
    - 1 cycle (registers), 2-5 cycles (cache), 20-50 cycles (RAM)
    - before: only one outstanding memory operation, CPU waits for completion
    - now: several outstanding operations

- Secondary storage: Disks (non-volatile)
  - accessed using a disk controller
  - supports random access but with non-uniform cost

- Tertiary storage: Tapes (non-volatile)
  - typically used only for backup
  - very inefficient support for random access

- Organized as a hierarchy
  - small amount of faster, more expensive storage closer to the CPU
  - larger amounts of slower, less expensive storage further away

Storage Hierarchy

- Rationale
  - keep CPU busy: lots of fast memory
  - keep system cost down

- How does it work
  - caching: upon access, move datum or instruction and its neighbors into higher levels of the hierarchy
  - replacement when a level fills up

- Why does it work
  - Real programs demonstrate locality
    - e.g.: rows and columns of a matrix
    - e.g.: sequential instructions
  - once a datum or instruction is used, things “near” them are likely to be used “soon”
Computer-System Structures (3): Protection

- Goal: Prevent user processes from accidently/maliciously damaging
  - the OS structures
  - parts of other process’s memory space
  - other user’s I/O devices

- Mechanisms address different ways in which protection breaks down
  1. dual-mode operation
     - user process taking over part of the OS and using this to overwrite other processes or even modify the OS itself (as in MS-DOS)
  2. privileged instructions
     - user process intervening in I/O of another process via control of the I/O handlers and indirectly causing damage
  3. memory protection
     - user process directly accessing another user process’ storage
  4. CPU protection via timers
     - hanging the OS -- e.g., via an infinite loop

Protection Mechanisms (1):
Dual-mode Operation and Privileged Instructions

- Dual-mode operation
  - supervisor and user modes
  - system starts off in supervisor mode and reenters it for interrupt processing
  - operating system gains control in supervisor mode
- Privileged instructions
  - restrict use of certain instructions to supervisor mode
    - I/O, including interrupt control
    - may be done by memory mapping
  - affect memory mapping
  - affect CPU mode (user/supervisor)
  - hardware support crucial for performance and for atomicity

Protection Mechanisms (2): Memory Protection

- Basic method: Memory is divided into segments
  - logical addresses are mapped to physical addresses
    - provides sharing, etc.
  - hardware support for address mapping
  - on a memory protection violation
    - user process traps to (interrupts) the OS

Protection Mechanisms (3): Timers

- OS code can enforce policies only if it gets a chance to run
- Timers maintain a count of elapsed (system) clock ticks
  - when timer expires, the CPU is interrupted
- Used for
  - interrupting hung processes
  - context switching in time-shared systems
- Access to timers is (usually) privileged
  - WHY?
Outline

* Announcements
  - Text book should be available in stores today
  - Accounts on courses 1 and courses 2
  - Questions?

* History of operating systems
* Computer-system structures
  - I/O structures
  - storage structures and hierarchy
    - memory, secondary storage, tape
  - hardware support for protection
* Operating-system structures
  - the functional view (what it does)
  - the components view (designer's view)
  - the services view (user/programmers' view)
  - the structure view (implementers' view)

[Silberschatz/Galvin: Chapters 1-3]

OS Views (1): Functional View (Directly Motivated)

* Program execution and handling:
  - starting programs, managing their execution, and communicating results
* I/O operations:
  - mechanisms for initiating and managing I/O
* File-system management:
  - creating, maintaining and manipulating files
* Communication:
  - between processes of the same user
    - such as sending the results of an input request to a user program,
  - between different users
* Exception detection and handling:
  - protection related issues
  - safety in the case of power failures via backups
  - detecting undesirable states such as printers out of paper

OS Views (2): Components View

* Resource allocation:
  - includes processor and I/O scheduling, memory management
* Accounting:
  - to track user's usage of resources for billing and/or statistical reasons
* Protection:
  - maintaining integrity of users' data
  - integrity checks to keep out unauthorized users
  - maintaining logs of incorrect attempts

Course organization follows this view
Components View: Processes

- A process is
  - the dynamic entity defined by the execution of a program
  - typically, program + run-time control information
    - control information includes memory maps, program counter values etc.
    - may also have context information
  - OS attempts to treat all processes uniformly

- Processes may play different roles
  - user processes
  - OS (system) processes

- A single process can spawn other processes
  - a computation typically requires many processes
    - shell, one or more user processes, one or more system processes

Components View: Process Management

- Operations
  - creation and termination
  - suspension and resumption
    - due to interrupts, context switches ...
  - synchronizing processes
    - making sure that a process that is waiting on an I/O waits till it is completed and does not wait forever, i.e., wakes up soon after the I/O process terminates
    - managing concurrent access to shared resources
  - communication
    - between two processes enabling them to cooperate
    - deadlock detection and avoidance

- Detailed discussion in Lectures 3-11

Components View: Storage Management

- Managing main memory
  - allocating main memory to active processes
    - maintaining a map of allocated vs. free memory
  - deallocating currently used memory to make room for new processes

- Managing secondary storage
  - managing the free sectors/tracks on disks
  - allocating this storage to programs
  - scheduling access requests to the disk

- Detailed discussion in Lectures 12-18

Components View: I/O Management

- Devices
  - device drivers
  - accepting an I/O request and invoking the appropriate device driver

- Files
  - non-volatile representations of user/system programs and data
  - file systems
    - support logical organization of data that the user might want to see
    - map data onto the physical storage devices and orchestrate their access/update
  - operations
    - creation, manipulation, and deletion of files and directories
    - moving files from primary to secondary storage while maintaining structure
    - interaction with the memory manager
    - backup and protection

- Detailed discussion in Lectures 15-21
OS Views (3): Services View

- User programs interact with a command interpreter

- Different levels of specification for the services view
  - system calls
    - the interfaces through which processes invoke specific OS functions
  - system programs
    - may use capabilities not available via system calls
    - the interfaces at these levels can be standardized
      - e.g. POSIX, network protocols, ...

Services View: Command Interpreters

- Interface between the user and the operating system
  - can be a part of the kernel or a separate process (shell)
  - striking differences between OSes
    - range from GUIs to (cryptic) control card interpreters
  - typical commands
    - process creation and (implicitly) destruction
    - I/O handling and file system manipulation
    - communication: interact with remote devices
    - protection management: changing file/directory access control, etc.

- Three varieties of shells:
  - the interpreter contains the code for the requested command (e.g., delete)
  - the interpreter calls a system routine to handle the request
  - the interpreter spawns new process(es) to handle the request
    - process lookup through some general procedure
    - most extensible

Services View: System Calls

- Interface between a process and the operating system
  - arguments typically passed in registers, a memory block, or on the stack

- Process control
  - load, execute, end, abort, create/terminate
  - get and set process attributes (priorities, allowable execution times)
  - wait for time/event, signal event, allocate memory

- File manipulation
  - create/delete, open/close, read/write and reposition
  - get/set file attributes (protection parameters, locations in directories ...)

- Device manipulation
  - same set of calls as above (devices are treated as special files) + attach/detach

- Information maintenance
  - get/set system data (time, memory/CPU usage), process and device attributes

- Communications
  - create/delete links, send/receive messages
  - transfer status information
  - modes can be message passing or shared memory

Services View: System Programs

- Each system call is usually supported by a system program

- Additionally, we also have standard applications
  - command interpreter
  - programming language support:
    - compilers, assemblers, debuggers
  - loaders and linkers:
    - for enabling the execution of the programs that start new processes and connect different ones
  - application programs:
    - text processing software, graphics support, spreadsheets, games
### OS Views (4): Structure View

- How to structure OS functionality
  - layering
  - virtual machines
- Designing and implementing an OS
- Read Sections 3.5-3.9, Silberschatz and Galvin
- Look at Nachos source code
  - Thomas Narten’s roadmap

### Lecture Summary

- History of operating systems
- Computer-system structures
  - I/O structures
  - storage structures
  - support for protection
- Operating-system structures
  - functional view: what does an OS do?
  - components view: what does an OS contain?
  - services view: how do user programs interact with an OS?
  - structure view: how should OSes be built/designed?

### Next Lecture

- Processes and Threads
  - process concept
  - process scheduling
  - operation on processes
  - cooperating processes
  - threads

**Reading**
- Silberschatz/Galvin: Sections 4.1-4.5