Operating System Structures

- Parts of an operating system
- Services that an operating system provides
- Calls to the operating system
- Operating system programs
- Operating system structure
  - » Layering
  - » Levels of abstraction
- Virtual machines
  - » General VMs
  - » DLX virtual machine
- System design and implementation

Operating System Components

- Process management
- Memory management
  - » Main memory
  - » Secondary storage (disk) management
- I/O system management
- File management
- Protection
  - » Users from one another
  - » Operating system from everyone
- Networking
- Command interpreter (user interface)

Managing Processes

- What is a process?
  - » Program in execution
  - » More generally, a single stream of instructions
- What does a process need?
  - » CPU time
  - » Memory, files & disk space
  - » I/O devices
- What is the operating system responsible for?
  - » Creating and deleting processes
  - » Starting and stopping processes
  - » Providing mechanisms to allow
    - – Processes to synchronize with one another
    - – Processes to communicate with one another

Managing Main Memory

- What is main memory?
  - » Large array of bytes (or words)
  - » Volatile storage: loses contents if the system crashes or fails
- What is the OS responsible for?
  - » Allocating pieces of memory to the processes that want to use it
  - » Keeping track of who is using which part of memory
  - » Deciding which processes to run based on available memory

Disk Management

- What is disk?
  - » Larger storage than main memory
  - » Non-volatile: contents survive system failures and crashes (though disk crashes can lose data…)
  - » Repository for most programs and data (accessed through file system)
  - » Other types of storage possible, but not common for active use (flash memory, tape)
- What's the OS responsible for?
  - » Allocating storage to those who request it
  - » Managing the available free space
  - » Scheduling the operation of the disk

I/O System Management

- What is the I/O (input/output) system?
  - » Other devices such as video, tape drives, mouse & keyboard
  - » Typically not used as active storage
- What's the OS responsibility
  - » Maintaining buffers for devices to use for transfers
  - » Providing a standard interface to different kinds of devices via device drivers
  - » Providing drivers for many kinds of hardware devices

File Management

- What's a file?
  - » Information grouped together by its creator
  - » Examples
    - – Program
    - – Documents (spreadsheet, paper, etc.)
    - – Data
- What's the OS responsible for?
  - » Creating and deleting files
  - » Allowing users to find files they've previously stored
    - – Creating and deleting directories
    - – Looking up files
  - » Getting file contents to and from memory
  - » Backing up files for safety reasons

Protection

- What is protection?
  - » Controlling accesses to resources by
    - – Users
    - – Processes
  - » Maintaining integrity of resources
- What is OS responsibility?
  - » Distinguish between allowed and unauthorized usage
    - – Allow different types to be specified
    - – Keep track of who's allowed to change the rules
  - » Enforce protection rules
    - – Prevent unauthorized accesses
    - – Possibly prevent users from finding out what they can't access...

Network Management

- What's a network?
  - » Communication system tying computers together
  - » Mechanism for allowing multiple computers to act together (distributed system)
- What's the OS responsibility?
  - » Decide who's allowed to use the network
  - » Figure out who gets messages, and how to deliver them
  - » Take CMSC 481 to find out more...

Command Interpreter / User Interface

- What is it?
  - » Method for users to ask the computer to do something
  - » Mechanism for processes (programs) to make requests
- What are the OS responsibilities?
  - » Interpret control statements that request previously mentioned functions
    - – Process & resource management
    - – Protection
    - – File system & I/O device access
  - » Provide an easy-to-use (hopefully) interface to the OS
    - – Command line interpreter (shell)
    - – Graphical user interface (GUI)

OS-Provided Services

- Execute a program
  - » Load into memory
  - » Run it
- Perform I/O operations on behalf of users & processes
- File system operations (read/write/create/delete, etc.)
- Communications
  - » Between processes on the same computer
  - » Between a process on this computer and one on another
  - » Uses either *shared memory* or *message passing*
- Detect errors
  - » Report (and perhaps work around) errors in CPU, I/O devices, and memory
  - » Contain errors in user processes (and OS!)

Internal OS "Services"

- Additional services not directly requested by users
  - » Necessary to allow OS to function properly
  - » Often largely invisible to users
- Resource allocation
  - » Divide available resources between processes
  - » Ensure that resources aren't overallocated
- Accounting
  - » Track resource usage by users & processes
    - – Billing
    - – Gather usage statistics
- Protection
  - » Control access to system resources
  - » Prevent unauthorized resource usage

System Calls

- Provide an interface between a process and the OS
  - » Implemented as assembly language instructions
  - » Usually "hidden" in a standard library of code
  - » Most languages (C, Fortran, Pascal, etc.) allow system calls to be made directly
- Allow programs to pass information to the OS
  - » Pass by value
    - – Put parameters in CPU registers
    - – Push parameters onto the stack
  - » Store parameters in memory (table or otherwise), and pass pointers to the parameters (pass by reference)
  - » Operating system knows where to find information, and reads it after the system call gives it control
  - » Information is returned on stack or in register

OS Programs

- Not part of the operating system kernel (central part of OS)
- Provide services that can be done by user-level processes
  - » File & directory manipulation (`cat`, `ls`)
  - » Operating system status information (`ps, top`)
  - » Programming language support (`gcc, f77, perl`)
  - » Program loading and execution (`ld`)
  - » Communications (`ssh, ping`)
  - » Protection manipulation (`chmod, chgrp`)
  - » Interface (`X, tcsh`)
- Users normally interact with programs, not system calls
  - » Friendlier interface
  - » More error checking: protect users from themselves

Operating System Structure

- "Jumble" approach
  - » Little structure
  - » Smallest code size
- Modular approach
  - » Code grouped into modules
  - » All modules run at the same "level" and can access the same things
- Layered approaches
  - » Modules either layered or at same level
  - » Modules can only access the structures they need

 3-15

OS Structure : "Jumble" Approach

- Simple approach
  - » Most functionality in least space
  - » Difficult to add more functionality later
  - » Bugs in one module can crash the whole system
- Example : MS-DOS
  - » Not divided into modules
  - » Interfaces and levels of functionality not well-separated
  - » Difficult to add new functionality
    - – Changes made all over the system
    - – Bugs in additions could crash the entire system
  - » Advantage: small memory footprint

 3-16

OS Structure : Modular Approach

- Still relatively simple
  - » No need for advanced hardware
  - » Relatively fast (low overhead)
- Example : BSD 4.x UNIX
  - » Two basic levels of structure
    - – Systems programs
    - – Operating system kernel
  - » Kernel further broken down into modules
    - – Individual modules perform specific functions
    - – Device drivers easily added (well-defined interface)
  - » All modules can access all data structures
    - – Interaction between modules may be easier to program
    - – Bugs in one module may affect other modules

OS Structure: Layered Approach

- Break operating system into modules
- Allow each module to access only those structures it needs to perform its job
  - » Fewer bugs due to unusual interactions: more stability?
  - » Better overall protection
  - » May be a bit slower
- Example: Mach (microkernel)
  - » Each module (memory management, process management, etc.) can only access its own structures
  - » Communications between modules via simple interface
  - » Bug in one module may not crash entire machine
  - » Flaky device drivers can be dealt with
  - » Potentially easier to add code to operating system

OS Structure: Layered Approach

- OS divided into layers, each one using functions only of lower layers
- Example: THE operating system
  - » Layer 5: user programs
  - » Layer 4: I/O device buffering
  - » Layer 3: operator/console device driver
  - » Layer 2: memory management
  - » Layer 1: CPU scheduling
  - » Layer 0: hardware

Virtual Machines

- Provide an interface to the user identical to the underlying bare hardware
  - » Each process has access to all hardware features
    - – Its own CPU & memory
    - – Its own I/O devices
  - » Virtual machine interprets requests that might be “dangerous” and changes them to keep processes separate
  - » CPU scheduling gives a process the illusion of its own CPU
  - » Virtual I/O devices created by interleaving accesses from processes
- Programs can be written that use the raw hardware
- Virtual machine need not be the same as the actual machine on which it’s running

Why Use a Virtual Machine?

- Protect system resources
  - » Each VM is isolated from all other VMs
  - » Individual VM can run simple (or no) operating system
  - » However, no direct sharing of resources?
- Provide a platform for OS (and architecture) research & development
  - » Use “raw” hardware to develop new operating systems without crashing current system
  - » Simulate architecture changes without actually building them
- Unfortunately, VMs are difficult to implement because they must provide an exact duplicate of the underlying machine

Goals for OS Design

- Goals for the user experience: OS should be:
  - » Reliable
  - » Easy to use
    - – Graphical interface?
    - – Simple-to-understand commands
  - » Fast
  - » Safe
- Goals for the OS designer: OS should be:
  - » Well-designed
    - – Easier to implement & maintain
    - – Error-free design (reliable, few bugs)
  - » Flexible: able to add new pieces without a total redesign
  - » Efficient: use the hardware well without wasting resources

Distinguishing Mechanism from Policy

- OS designers must separate *mechanism* from *policy*
- Mechanisms
  - » Tell the system *how* to accomplish something
  - » May be changed without changing policies
    - – New mechanisms may be more efficient
    - – New hardware may require new mechanisms
- Policies
  - » Tell the system *what* to do
  - » Changes in policy need not result in new mechanisms
    - – Existing mechanisms used in different ways
    - – Single operating systems can support multiple policies

How are Systems Implemented?

- Assembly language
  - » May be faster
  - » Allows access to specific features of hardware
- High-level languages
  - » Code is easier and quicker to write
  - » Code is more compact
  - » Code is easier to understand, and thus debug
  - » Code can be *ported* (moved to other hardware) by simply recompiling
- Some assembly language is necessary for an OS
  - » Low-level details of manipulating hardware
  - » Context switches
  - » Goal: minimize assembly language code

Porting Operating Systems

- Operating systems can run on multiple platforms
- To allow this, use modular code and change only what's necessary
  - » Code to do context switch and other low-level hardware operations
  - » Device drivers for particular kinds of devices
- Recompile everything else for the new system
  - » Compiler can produce code optimized for a particular model of CPU
  - » Compiler can produce code that will run on a wide range of models
- Trade off efficiency and ease of porting

Creating an OS for a New Machine

- Sometimes necessary to create an OS from scratch
  - » Brand-new architecture (PowerPC, Intel Merced)
  - » Brand-new OS design (BeOS)
- Make it easier by:
  - » Creating development tools that run on existing OS
  - » Recycling code for existing OS
  - » Running on a virtual machine with debugging tools available
    - – OS runs slower
    - – Development can begin before hardware is available