東南大學
SOUTHEAST UNIVERSITY

OPERATING SYSTEM CONCEPTS

Chapter 2. Operating-System Structures

A/Prof. Kai Dong

Warm-up

Some History

- 1st Gen, **vacuum tube** (1945-1955) — mainframe: Libraries
  - "OS": a set of libraries of commonly-used functions.
  - One program ran at a time, as controlled by a human operator.
    - Batch system with a human operator.
- 2nd Gen, **transistor** (1955-1965) — beyond Libraries: + Protection
  - Now suppose a bus replacing the operator.
  - What if an application read from / write to anywhere on disk?
    - System call + Dual mode
- 3rd Gen, **integrated circuit** (1965-1980) — minicomputer: + Long-term Scheduling
  - Multiprogramming
    - Batch system with an operating system
- 4th Gen, **large-scale integration** (1980-now) — personal computer (PC): + Short-term Scheduling
  - Timesharing

Objectives

- To describe the services an operating system provides to users, processes, and other systems.
- To discuss the various ways of structuring an operating system.
- To explain how operating systems are installed and customized and how they boot.

Operating System Services

*Functions Helpful to the **User***

- **User interface** — Almost all operating systems have a user interface (UI).
  - Varies between Command-Line (CLI), Graphics User Interface (GUI), Batch.
- **Program execution** — The system must be able to load a program into memory and to run that program, end execution, either normally or abnormally (indicating error).
- **I/O operations** — A running program may require I/O, which may involve a file or an I/O device.
- **File-system manipulation** — The file system is of particular interest. Programs need to read and write files and directories, create and delete them, search them, list file Information, permission management.
- **Communications** — Processes may exchange information, on the same computer or between computers over a network.
- **Error detection** — OS needs to be constantly aware of possible errors.

Operating System Services

*Functions Ensuring Efficient Operation of the **System***

- **Resource allocation** — When multiple users or multiple jobs running concurrently, resources must be allocated to each of them.
  - Many types of resources — CPU cycles, main memory, file storage, I/O devices.
- **Accounting** — To keep track of which users use how much and what kinds of computer resources.
- **Protection** and **security** — The owners of information stored in a multiuser or networked computer system may want to control use of that information, concurrent processes should not interfere with each other.
  - **Protection** involves ensuring that all access to system resources is controlled.
  - **Security** of the system from outsiders requires user authentication, extends to defending external I/O devices from invalid access attempts.

User Operating System Interface

Command Line Interfaces

- Command line interface (CLI) or command interpreter allows direct command entry.
  - Sometimes implemented in kernel, sometimes by systems program.
  - Sometimes multiple flavors implemented — shells.
  - Primarily fetches a command from user and executes it.
  - Sometimes commands built-in, sometimes just names of programs.
    - If the latter, adding new features doesn’t require shell modification.

User Operating System Interface

Graphical User Interfaces

- User-friendly desktop metaphor interface.
  - Usually mouse, keyboard, and monitor.
  - Icons represent files, programs, actions, etc.
  - Various mouse buttons over objects in the interface cause various actions (provide information, options, execute function, open directory (known as a folder).
  - Invented at Xerox PARC.
- Many systems now include both CLI and GUI interfaces.
  - Microsoft Windows is GUI with CLI “command” shell.
  - Apple Mac OS X is “Aqua” GUI interface with UNIX kernel underneath and shells available.
  - Unix and Linux have CLI with optional GUI interfaces (CDE, KDE, GNOME).

User Operating System Interface

Touchscreen Interfaces

- Touchscreen devices require new interfaces.
  - Mouse not possible or not desired.
  - Actions and selection based on gestures.
  - Virtual keyboard for text entry.
- Voice commands.

System Calls

- Programming interface to the services provided by the OS.
- Typically written in a high-level language (C or C++).
- Mostly accessed by programs via a high-level Application Programming Interface (API) rather than direct system call use.
- Three most common APIs
  - Win32 API for Windows
  - POSIX API for POSIX-based systems (including virtually all versions of UNIX, Linux, and Mac OS X)
  - Java API for the Java virtual machine (JVM)
- Why do users use procedure calls / APIs rather than system calls?
  - Program portability.
  - System calls are more detailed and difficult to work with.

System Calls

System Call Implementation

- Typically, a **number** associated with each system call.
  - **System-call interface** maintains a table indexed according to these numbers.
- The system call interface invokes the intended system call in OS kernel and returns status of the system call and any return values.
- The caller need know nothing about how the system call is implemented.
  - Just needs to obey API and understand what OS will do as a result call.
  - Most details of OS interface hidden from programmer by API.
    - Managed by run-time support library (set of functions built into libraries included with compiler).

System Calls

System Call Parameter Passing

- Often, more information is required than simply identity of desired system call.
  - Exact type and amount of information vary according to OS and call.
- Three general methods used to pass parameters to the OS.
  1. Simplest: pass the parameters in registers.
     » In some cases, may be more parameters than registers.
  2. Parameters stored in a block, or table, in memory, and address of block passed as a parameter in a register.
     » This approach taken by Linux and Solaris.
  3. Parameters placed, or pushed, onto the stack by the program and popped off the stack by the operating system.
  - Block and stack methods do not limit the number or length of parameters being passed.

System Calls

Motivating System Call

- What should a user process do to perform a privileged operation?
  - **System Calls**
- How to execute a system call?
  - **Trap**, **trap-handler** and **return-from-trap**
- In a trap, which code to run inside the OS?
  - **Trap table** –> **trap-handler**.
- When return-from-trap, how to restore context?
  - Kernel stack for each process.

System Calls

What Happens When the Computer System is Booting?

OS @boot (kernel mode)	**Hardware**
initialize *trap table*	
	remember addresses of ...
	system call handler
	timer handler
	illegal instruction handler
start interrupt timer	
	start timer; interrupt after *X* ms

System Calls

What Happens When the Computer System is Running?

OS @run (kernel mode)	**Hardware**	**Program (user mode)**
to start process *A*:		
return-from-trap (into *A*)		
	move to *user mode*	
		process *A* runs:
		fetch instruction
		execute instruction
		...
		call *system-call*(), via *api*
		trap into OS
	system call interface	
	move to *kernel mode*	
	jump to *system call handler*	
handle the *trap* (*system call*)		
return-from-trap (into *A*)		
		...

Types of System Calls

Process Control

- create process, terminate process
- end, abort
- load, execute
- get process attributes, set process attributes
- wait for time
- wait event, signal event
- allocate and free memory
- dump memory if error
- debugger for determining bugs, single step execution
- locks for managing access to shared data between processes

Types of System Calls

File Management

- create file, delete file
- open, close file
- read, write, reposition
- get and set file attributes

Types of System Calls

Device Management

- request device, release device
- read, write, reposition
- get device attributes, set device attributes
- logically attach or detach devices

Types of System Calls

Information Maintenance

- get time or date, set time or date
- get system data, set system data
- get and set process, file, or device attributes

Types of System Calls

Communications

- create, delete communication connection
- send, receive messages if message passing model to host name or process name
  - from client to server
- Shared-memory model create and gain access to memory regions
- transfer status information
- attach and detach remote devices

Types of System Calls

Protection

- control access to resources
- get and set permissions
- allow and deny user access

Operating System Design and Implementation

Policy & Mechanism

- Important principle to separate
  - **Policy**: What will be done?
  - **Mechanism**: How to do it?
- Mechanisms determine how to do something, policies decide what will be done.
- The separation of policy from mechanism is a very important principle, it allows maximum flexibility if policy decisions are to be changed later (example — timer).
- Specifying and designing an OS is highly creative task of software engineering.

Operating System Structure

- General-purpose OS is very large program
- Various ways to structure ones
  - Simple structure — MS-DOS
  - More complex — UNIX
  - Layered — an abstraction
  - Microkernel — Mach
  - Modules — Linux
  - Hybrid

Operating System Structure

Simple Structure — MS-DOS

- MS-DOS — written to provide the most functionality in the least space.
  - Not divided into modules.
  - Although MS-DOS has some structure, its interfaces and levels of functionality are not well separated.

Operating System Structure

Non Simple Structure — UNIX

- UNIX — limited by hardware functionality, the original UNIX operating system had limited structuring.
- The UNIX OS consists of two separable parts.
  - Systems programs
  - The kernel
    - Consists of everything below the system-call interface and above the physical hardware.
    - Provides the file system, CPU scheduling, memory management, and other operating-system functions; a large number of functions for one level.

Operating System Structure

Layered Approach

- The operating system is divided into a number of layers (levels), each built on top of lower layers. The bottom layer (layer 0), is the hardware; the highest (layer N) is the user interface.
- With modularity, layers are selected such that each uses functions (operations) and services of only lower-level layers

Operating System Structure

Microkernel System Structure

- Moves as much from the kernel into user space.
- Mach example of microkernel.
  - Mac OS X kernel (Darwin) partly based on Mach.
- Communication takes place between user modules using message passing.
- Benefits:
  - Easier to extend a microkernel.
  - Easier to port the operating system to new architectures.
  - More reliable (less code is running in kernel mode).
  - More secure.
- Detriments:
  - Performance overhead of user space to kernel space communication.

Operating System Structure

Modules

- Many modern operating systems implement loadable kernel modules.
  - Uses object-oriented approach.
  - Each core component is separate.
  - Each talks to the others over known interfaces.
  - Each is loadable as needed within the kernel.
- Overall, similar to layers but with more flexible.
  - Linux, Solaris, etc

Operating System Structure

Hybrid Systems

- Most modern operating systems are actually not one pure model.
  - Hybrid combines multiple approaches to address performance, security, usability needs.
  - Linux and Solaris kernels in kernel address space, so monolithic, plus modular for dynamic loading of functionality.
  - Windows mostly monolithic, plus microkernel for different subsystem personalities.
  - Apple Mac OS X hybrid, layered, Aqua UI plus Cocoa programming environment.

Operating System Structure

Recent Trends

- Library Operating System

  —— Madhavapeddy, Anil, et al. "Unikernels: Library operating systems for the cloud." ACM SIGARCH Computer Architecture News 41.1 (2013): 461-472.

- Exokernel

  —— Engler, Dawson R., M. Frans Kaashoek, and James O'Toole Jr. "Exokernel: An operating system architecture for application-level resource management." ACM SIGOPS Operating Systems Review 29.5 (1995): 251-266.

- Multikernel

  —— Baumann, Andrew, et al. "The multikernel: a new OS architecture for scalable multicore systems." Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles. 2009.

- Splitkernel

  —— Shan, Yizhou, et al. "Legoos: A disseminated, distributed OS for hardware resource disaggregation." 13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18). 2018.