Chapter 2: Operating-System Structures

Chapter 2: Operating-System Structures

- Operating System Services
- User Operating System Interface
- System Calls
- Types of System Calls
- System Programs
- Operating System Structure

Objectives

- To describe the services an operating system provides to users, processes, and other systems
- To discuss the operation and types of system calls
- To discuss the various ways of structuring an operating system

Operating System Services

- Operating systems provide an environment for execution of programs and services to programs and users

- One set of operating-system services provides functions that are **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

Operating System Services (Cont.)

- One set of operating-system services provides functions that are **helpful to the user** (Cont.):
  - **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
    - Communications may be via shared memory or through message passing (packets of information in predefined formats moved between proceses by the OS)

Operating System Services (Cont.)

- One set of operating-system services provides functions that are **helpful to the user** (Cont.):
  - **Error detection** – OS needs to be constantly aware of possible errors
    - May occur in the CPU and memory hardware (e.g., power failure), in I/O devices (parity error on disk), in user program (attempt to access an illegal memory location)
    - For each type of error, OS should take the appropriate action to ensure correct and consistent computing
    - Debugging facilities can greatly enhance the user's and programmer's abilities to efficiently use the system

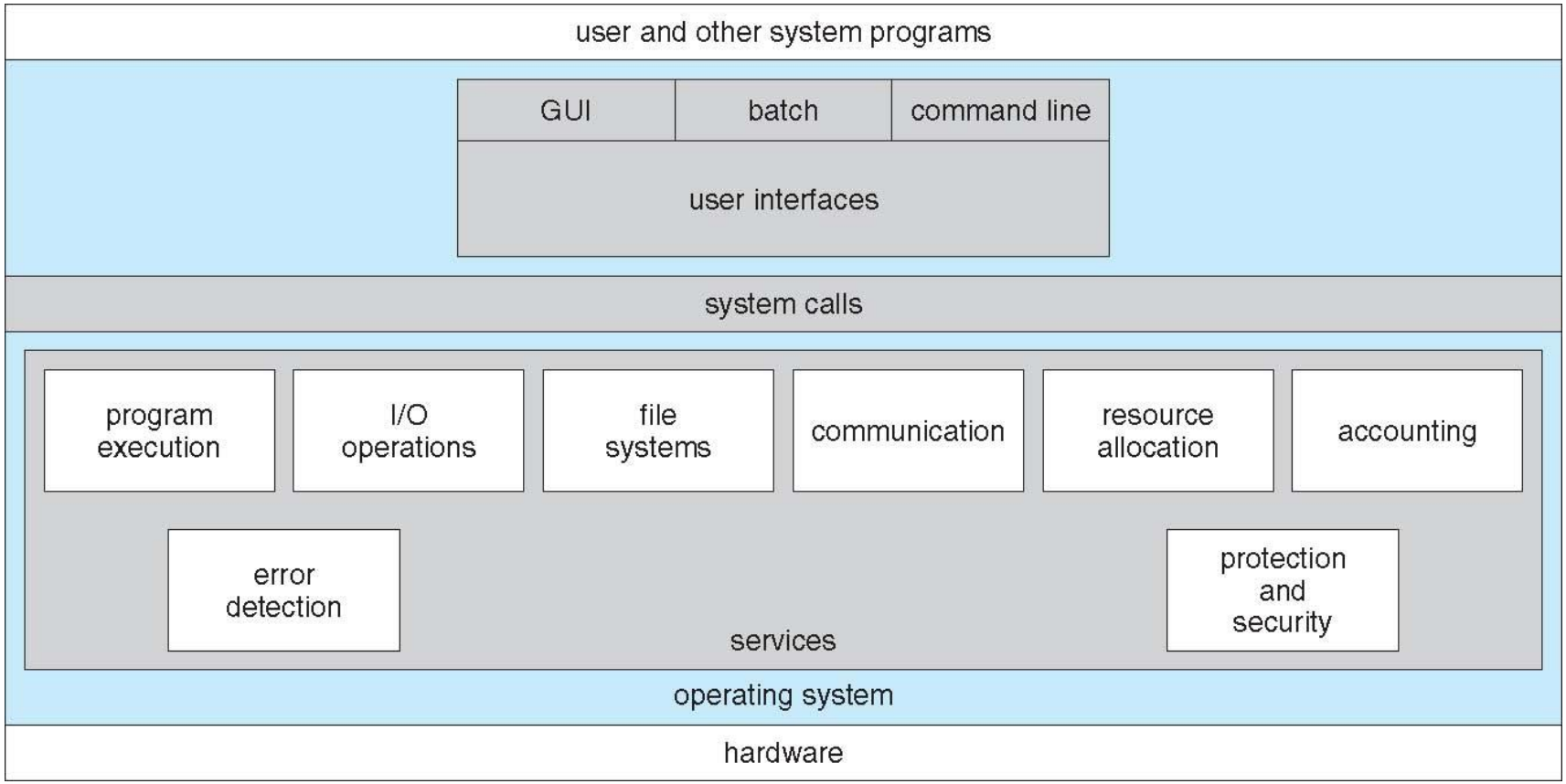
Operating System Services (Cont.)

- Another set of OS functions exists for ensuring the efficient operation of the system itself via resource sharing
  - **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
    - E.g., CPU-scheduling routines that takes into account the speed of the CPU, the jobs to be executed, the number of registers available, and other factors
  - **Accounting -** To keep track of which users use how much and what kinds of computer resources

Operating System Services (Cont.)

- Another set of OS functions exists for ensuring the efficient operation of the system itself via resource sharing
  - **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

A View of Operating System Services



User Operating System Interface - CLI

CLI (command-line interface) or **command interpreter** allows direct command entry

- Sometimes implemented in kernel, sometimes by systems program
- Systems with multiple command interpreters to choose from – the interpreters are known as **shells**
- Primarily fetches a command from user and executes it
  - E.g., manipulate files: create, delete, list, print, copy, execute, etc.

User Operating System Interface - CLI

Commands can be implemented in two general ways

- **Built-in approach**, the command interpreter itself contains the code to execute the command
  - The number of commands that can be given determines the size of the command interpreter
  - *Advantage*: speed and overall simplicity
  - *Disadvantage*: new commands require rewriting the interpreter program which, after a number of modifications, may get complicated, messy, or too large

User Operating System Interface - CLI

Commands can be implemented in two general ways

- **Implement most commands through system programs**
  - E.g., UNIX command to delete a file

    rm file.txt

    - Search for a file called rm, load the file into memory, execute it with the parameter file.txt
  - *Advantage*: adding new features doesn't require shell modification
  - *Disadvantage*: reduced speed and the clumsiness of passing parameters from the interpreter to the system program

User Operating System Interface - GUI

- 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 in the 1970s
- 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)

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 are:
  - 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)

Note that the system-call names used throughout this text are generic

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 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
  - Simplest: pass the parameters in registers
    - In some cases, may be more parameters than registers
  - 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
  - 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

Types of System Calls: Six Major Categories

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

Types of System Calls (Cont.)

- Information maintenance
  - get time or date, set time or date
  - get system data, set system data
  - get and set process, file, or device attributes
- 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 (Cont.)

- Protection
  - Control access to resources
  - Get and set permissions
  - Allow and deny user access

System Programs

- System programs provide a convenient environment for program development and execution
- Can be thought of as bundles of useful system calls
  - They provide basic functionality to users so that users do not need to write their own programs to solve common problems
- Most users' view of the operation system is defined by system programs, not the actual system calls

System Programs

- System programs can be divided into:
  - File manipulation
  - Status information
  - File modification
  - Programming language support
  - Program loading and execution
  - Communications
  - Background services

System Programs

- **File management** - Create, delete, copy, rename, print, dump, list, and generally manipulate files and directories
- **Status information**
  - Some ask the system for info - date, time, amount of available memory, disk space, number of users
  - Others provide detailed performance, logging, and debugging information
  - Typically, these programs format and print the output to the terminal or other output devices
  - Some systems implement a **registry** - used to store and retrieve configuration information

System Programs (Cont.)

- **File modification**
  - Text editors to create and modify files
  - Special commands to search contents of files or perform transformations of the text
- **Programming-language support** - Compilers, assemblers, debuggers and interpreters sometimes provided
- **Program loading and execution**- Absolute loaders, relocatable loaders, linkage editors, and overlay-loaders, debugging systems for higher-level and machine language
- **Communications** - Provide the mechanism for creating virtual connections among processes, users, and computer systems
  - Allow users to send messages to one another's screens, browse web pages, send electronic-mail messages, log in remotely, transfer files from one machine to another

System Programs

- **Background Services**
  - Launch at boot time
    - Some for system startup, then terminate
    - Some from system boot to shutdown
  - Provide facilities like disk checking, process scheduling, error logging, printing
  - Constantly running system-program processes: known as **services**, **subsystems**, **daemons**

System Programs

- Along with system programs, most operating systems are supplied with **application programs**
  - Don’t pertain to system, e.g., Web browsers, word processors, …
  - Run by users
  - Not typically considered part of OS
  - Launched by command line, mouse click, finger poke

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

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

- System 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

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 since most services are running as user rather than kernel processes
- Detriments:
  - Performance overhead of user space to kernel space communication

Modules

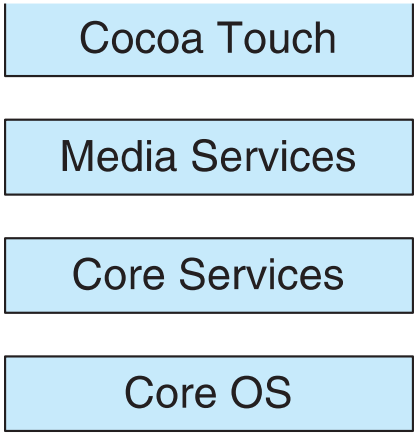
- Many modern operating systems implement **loadable kernel modules:** *the kernel has a set of core components and links in additional services via modules, either at boot time or during run time*
  - Uses object-oriented approach
  - Each core component is separate
  - Each talks to the others over known interfaces
  - Each is loadable as needed within the kernel
- Overall, similar to layers but with more flexible (because any module can call any other module)
  - Linux, Solaris, etc

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
  - Below is kernel consisting of Mach microkernel and BSD Unix parts, plus I/O kit and dynamically loadable modules (called **kernel extensions**)

iOS

- Apple mobile OS for ***iPhone***, ***iPad***
  - Structured on Mac OS X, added functionality
  - Does not run OS X applications natively
    - Also runs on different CPU architecture (ARM vs. Intel)
  - **Cocoa Touch** Objective-C API for developing apps
  - **Media services** layer for graphics, audio, video
  - **Core services** provides cloud computing, databases
  - Core operating system, based on Mac OS X kernel



Android

- Developed by Open Handset Alliance (mostly Google)
  - Open Source
- Similar stack to IOS
- Based on Linux kernel but modified
  - Provides process, memory, device-driver management
  - Adds power management
- Runtime environment includes core set of libraries and Dalvik virtual machine
  - Apps developed in Java plus Android API
    - Java class files compiled to Java bytecode then translated to executable than runs in Dalvik VM
- Libraries include frameworks for web browser (webkit), database (SQLite), multimedia, smaller libc

Summary

- Operating systems provide a number of services
  - At the lowest level, system calls allow a running program to make requests from the operating system directly
  - At a higher level, the command interpreter or shell provides a mechanism for a user to issue a request without writing a program
- System calls can be classified into several categories: program control, file manipulation, device manipulation, information maintenance, communications, and protection
- Structures: layer, microkernel, dynamically loaded modules

Exercises

- 1. What is the purpose of the command interpreter? Why is it usually separate from the kernel?

- It reads commands from the user or from a file of commands and executes them, usually by turning them into one or more system calls.

- It is usually not part of the kernel because the command interpreter is subject to changes.

Exercises

- 2. List the major activities of an operating system in regard to file management.

- The creation and deletion of files
- The creation and deletion of directories
- The support of primitives for manipulating files and directories
- The mapping of files onto secondary storage
- The backup of files on stable (nonvolatile) storage media

Exercises

- 3. What are the advantages of using loadable kernel modules?

- It is difficult to predict what features an operating system will need when it is being designed. The advantage of using loadable kernel modules is that functionality can be added to and removed from the kernel while it is running. There is no need to either recompile or reboot the kernel.

End of Chapter 2