Operating System Structures

Operating Systems

Yu-Ting Wu

1

Outline

- Operating system services
- System calls and APIs
- Operating system structure
- Operating system debugging

2

Operating System Services

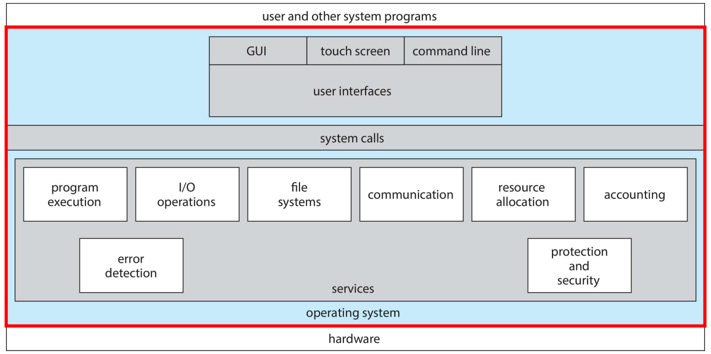
3

Operating System Services

- User interface
- Program execution
- I/O operations
- File-system manipulations
- Communication
- Error detection
- Resource allocation
- Accounting
- Protection and security

4

Operating System Services (cont.)



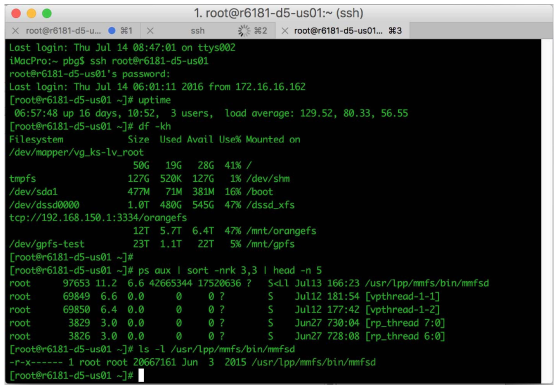
5

User Interface

- **Command line interface (CLI)**
  - Fetch a command from user and execute it
  - Shell (command-line interpreter)
    - Ex: CSHELL, BASH
    - Allow to some modification based on user behavior and preference
- **Graphic user interface (GUI)**
  - Usually with mouse, keyboard, and monitor
  - Icons are used to represent files, directories, programs, etc.
  - Usually built on CLI
- Most systems have both CLI and GUI

6

Command Line Interface



Bourne Shell (default shell of UNIX ver. 7)

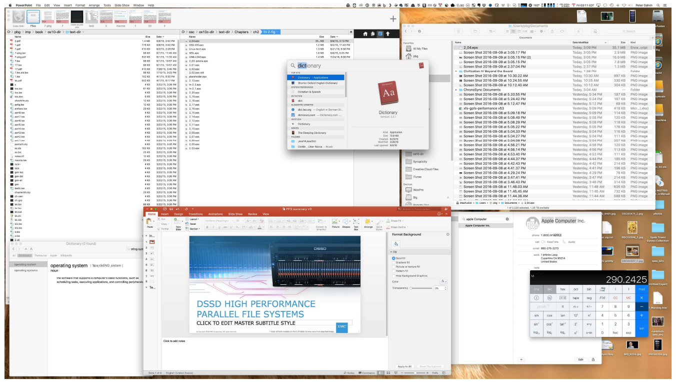
7

Command Line Interface (cont.)

- **Two approaches for the command interpreter**
  - **Contain the codes for executing commands**
    - Pros: fast
    - Cons: file size / painful revision
  - **Implement commands as system program**
    - Search execution files on the fly
    - Pros: easy to upgrade / keep the interpreter small
    - Cons: slow
    - Additional issues
      - Parameters passing
      - Inconsistent interpretation of parameters
- Most OS use a hybrid approach: keep a small subset of core functions in interpreter and use exec. for the others

8

Graphic User Interface



Mac OS X GUI

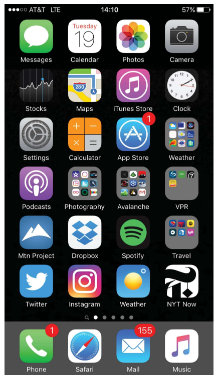
Graphic User Interface (cont.)

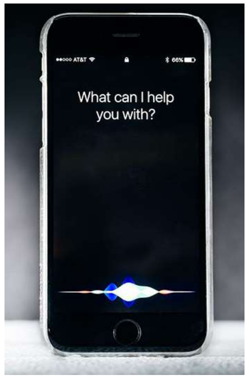
- Components
  - Screen
  - Icons
  - Folders
  - Pointers
  - etc.

- History
  - Xerox PARC research facilities (1970's)
  - Mouse (1968)
  - Mac OS (1980's)
  - Windows 1.0 ~ 11

Other Interfaces

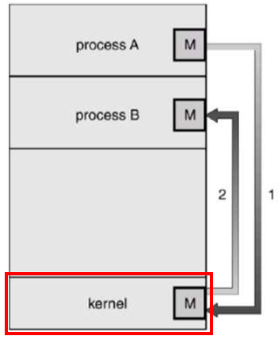
- Batch
- Touch-screen
- Voice control



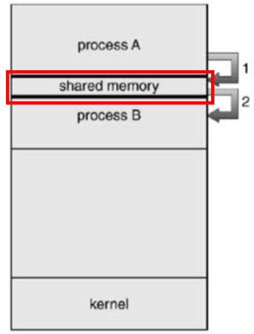


Communication Models

- Using either **Message Passing** or **Shared Memory**



message passing



shared memory

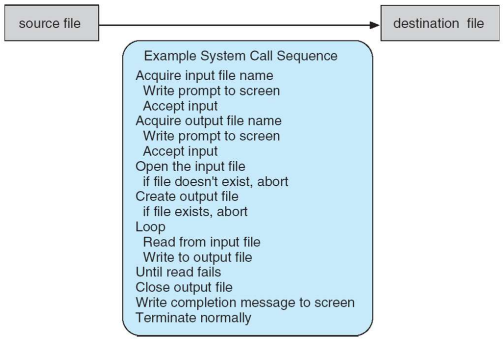
System Calls and APIs

System Calls

- **Programming interface to the services provided by the OS**
- Mostly accessed by programs via a high-level **Application Programming Interface (API)** rather than direct system call use

System Calls (cont.)

- Example: a sequence of system calls for copying a file



System Calls (cont.)

- Request OS services
  - **Process control**
    - End (normal exit) or abort (abnormal)
    - Load and execute
    - Create and terminate
    - Get or set attributes of process
    - Wait for a specific amount of time or an event
    - Memory dumping, profiling, tracing, allocate, and free
  - **File management**
    - Create and delete
    - Open and close
    - Read, write, and reposition
    - Get or set attributes
    - Operations for directories

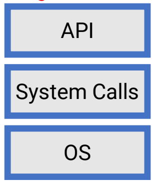
System Calls (cont.)

- Request OS services (cont.)
  - **Device management**
    - Request or release
    - Logically attach or detach devices
  - **Information maintenance**
    - Get or set time or date
    - Get or set system data (e.g., maximum memory for a process)
  - **Communications**
    - Send and receive messages
    - Message passing or shared memory
  - **Protection**

Application Programming Interface (API)

- An **encapsulation** of system calls for user programs
- Provide **portability**
- Usually implemented by high-level languages
  - Ex: C library, Java
- Could involve zero or multiple system calls
  - Ex: abs(): zero
  - Ex: fopen(): multiple
  - Ex: malloc(), free() ➔ brk()

e.g., Win32 API

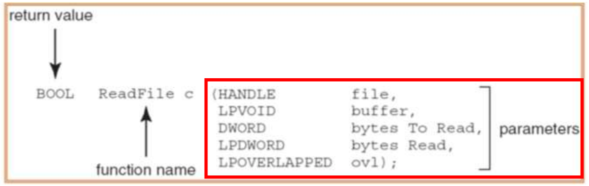


API (cont.)

- Three most common APIs
  - **Win32 API**
    - For Microsoft Windows
    - https://en.wikipedia.org/wiki/Windows_API
    - https://docs.microsoft.com/zh-tw/windows/win32/apiindex/windows-api-list?redirectedfrom=MSDN
  - **POSIX API**
    - POSIX stands for **P**ortable **O**perating **S**ystem **I**nterface for Uni**x**
    - Used by Unix, Linux, and Max OS X
    - https://en.wikipedia.org/wiki/POSIX
  - **Java**
    - For Java virtual machine (JVM)

API (cont.)

- Example: ReadFile() in Win32 API

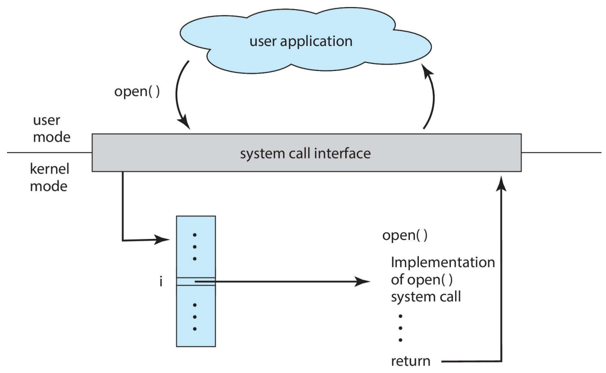


  - Parameters
    - HANDLE file: the file to be read
    - LPVOID buffer: a buffer where the data will be read into
    - DWORD bytesToRead: number of bytes to be read into the buffer
    - LPDWORD bytesRead: number of bytes read during the last read
    - LPOVERLAPPED ovl: indicates if overlapped I/O is being used

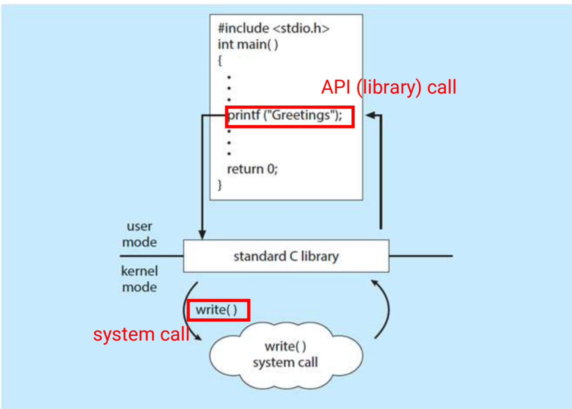
Why Do We Need API?

- **Simplicity**
  - API is designed for programmers and applications
- **Portability**
  - API is a unified defined interface
- **Efficiency**
  - Not all functions require OS services or involve kernel

System Call and API



System Call and API

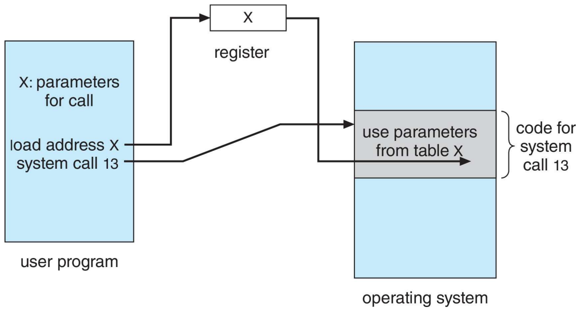


Passing Parameters

- Three general approaches for passing parameters between a program and the OS
- **Using register**
- **Store in a table in memory (Linux)**
  - The address of the table is passed by register
- **Push parameters onto the stack by the program**
  - And pop off by the OS

Passing Parameters (cont.)

- **Store in a table in memory (Linux)**
  - The address of the table is passed by register



System Structure

Overview of OS Structure

- Simple OS architecture
- Layer OS architecture
- Microkernel OS
- Modular OS architecture
- Hybrid systems
- Virtual machine

Design of an OS

- Start the design by defining **goals** and **specifications**
- **User goals**
  - Easy to use and learn
  - Reliable
  - Safe
  - Fast (interactive)
- **System goals**
  - Easy to design and implement
  - Easy to maintain
  - Reliable
  - Error-free
  - Efficient

Policy and Mechanism

- **User goals: what needs to be done?**
  - Example: time sharing after every 100 seconds
- **Mechanism: how to do something**
  - Example: timer

- The separation of policy from mechanism is important
  - Allow maximum flexibility if policy decisions are to be changed later

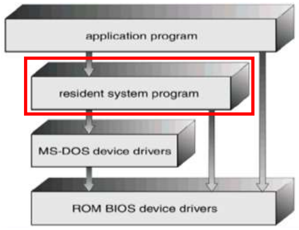
Implementation

- Much variation
  - Early Oses are implemented by assembly language
  - Now high level languages, such as C, C++
- Actually usually a **mix of languages**
  - Lowest levels in assembly
  - Main body in C
  - System programs in C or C++
  - Scripting languages using PERL, Python, shell scripts

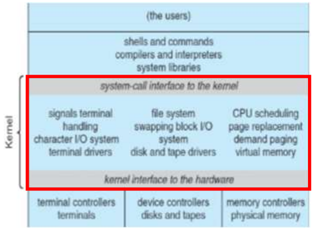
- More high-level language, easier to **port** to other hardware

Simple OS Architecture

- Only one or two levels
- Drawbacks
  - Unsafe
  - Difficult to enhance



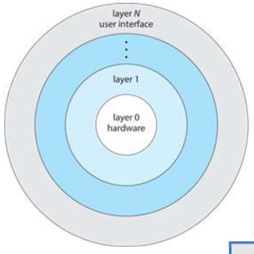
MS-DOS

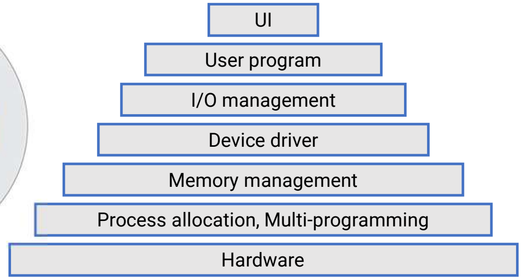


UNIX

Layered OS Architecture

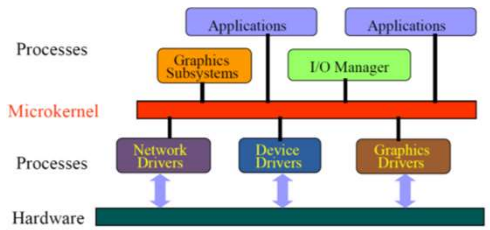
- Lower levels are independent of upper levels
- Pros: easier debugging and maintenance
- Cons: less efficient and difficult to define layers





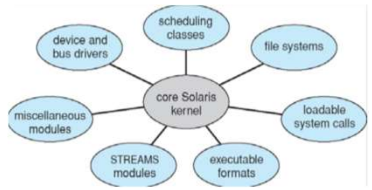
Microkernel OS

- **Kernel should be as small as possible**
  - Move most parts of the original kernels into user space
- Communication is provided by **message passing**
- Easier for extending and porting
- Slow



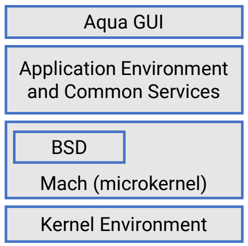
Modular OS Architecture

- **Employed by most modern OS**
  - Object-oriented approach
  - Each core component is separate
  - Each module talks to the others over known interfaces
  - Each module is loadable as needed **within the kernel**
- Similar to layers but with more flexibility
- Ex: Solaris



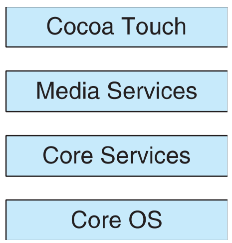
Hybrid: Mac OS

- Combine **layer** and **microkernel** design
  - Aqua graphical user interface
  - Applications environments and common services
  - BSD
    - Command line interface, networking, file systems, POSIX APIs
  - Mach
    - Memory management
    - Remote procedure calls
    - Inter-process communication
  - Kernel environment
    - I/O kit for device drivers
    - Dynamic loadable modules



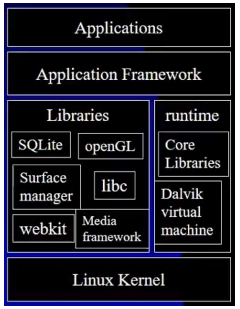
Hybrid: iOS

- Structured on Mac OS, added functionalities
  - Cocoa Touch
    - Objective-C API for developing apps
  - Media services
    - Layer for graphics, audio, video
  - Core services
    - Cloud computing ,database
  - Core OS
    - Based on Mac OS X kernel



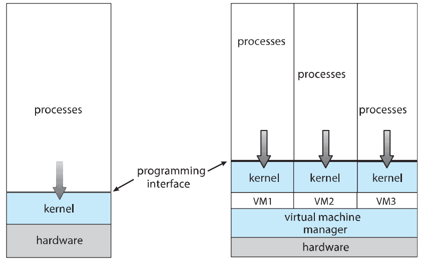
Hybrid: Android

- Developed by Handset Alliance (mostly Google)
  - Open source
- Based on Linux kernel (modified)
  - Add power management
- Runtime environment
  - Core set libraries
  - Dalvik VM



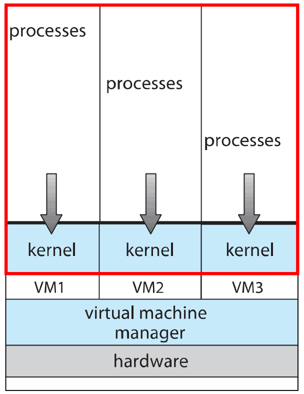
Virtual Machine

- **Layered** approach
- Provide an interface that is identical to the underlying bare hardware
  - Each process is provided with a (virtual) copy of the underlying computer



Virtual Machine (cont.)

- **Challenges**
  - Privileged instructions



Virtual Machine (cont.)

- **Advantages**
  - Provide complete protection of system resources
  - Provide an approach to solve system compatibility problems
  - Provide a vehicle for OS research and development
  - Provide a mean for increasing **resource utilization** in cloud computing

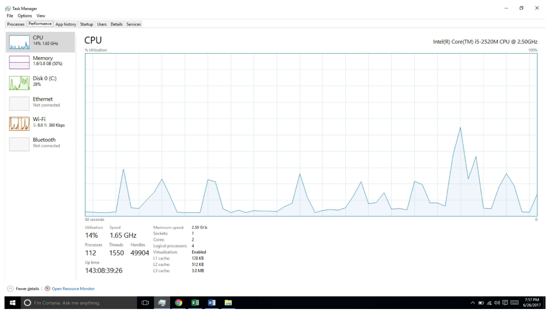
Operating System Debugging

Operating System Debugging

- **Debugging**
  - An activity in finding and fixing errors or bugs (including performance problems) that exist in hardware or software
- Terminologies
  - **Performance tuning**
    - A procedure that seeks to improve performance by removing bottleneck
  - **Core dump**
    - A capture of the memory of a process or OS
  - **Crash**
    - A kernel failure

Operating System Debugging (cont.)

- **Performance tuning**
  - OS must provide means of computing and displaying measures of system behavior



Objectives Review

- Identify services provided by an operating system
- Illustrate how system calls are used to provide operating system services
- Compare and contrast monolithic, layered, microkernel, modular, and hybrid strategies for designing operating systems