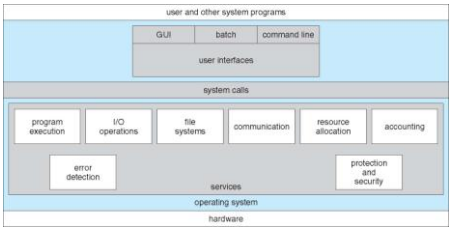
Operating System Structures

Operating Systems

A View of Operating System Services



Operating System Design and Implementation

- Internal structure of different Operating Systems can vary widely…
- Start by defining goals and specifications
  - Affected by choice of hardware, type of system
- **User** goals and **System** goals
  - User goals – operating system should be convenient to use, easy to learn, reliable, safe, and fast
  - System goals – operating system should be easy to design, implement, and maintain, as well as flexible, reliable, error-free, and efficient

Operating System Design and Implementation (Cont.)

- Important principle to separate
  Policy: ***What*** to do?
  Mechanism: ***How*** to do?
- 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 flexibility if policy decisions are to be changed later
- Specifying and designing OS is highly creative task. It is also an important **software engineering** **task**.

Implementation

- There are variations in time:
  - Early OSs were in assembly language
  - Later system programming languages like Algol and PL/1 are used together with assembly languages/
  - Now mostly C and C++ are used
- In reality a mix of languages is used
  - Lowest levels close to hardware is in assembly language, main body is coded in C
  - Systems programs are usually coded in C, C++.
  - Use of scripting languages like PERL, Python, shell scripts is possible

Operating System Structure

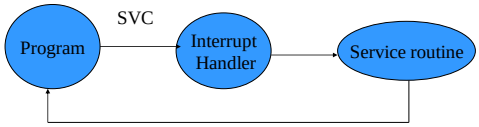
- General-purpose OS is a very large program.
- How to structure it is an important design decision. They may be structured differently

Possible Types of OS Structures

- Monolithic systems
- Hierarchy of layers
- Virtual machines
- Micro-kernel (client/server) model

Monolithic System

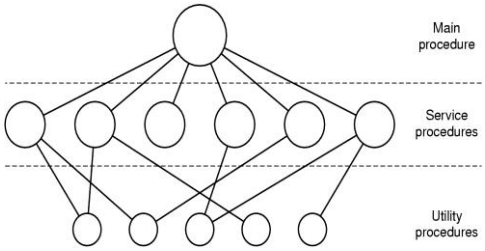
- OS has no structure. It is a collection of procedures with well defined calling interfaces to them…
- Appication - OS interface is implemented via supervisor calls (SVC). Return from SVC is the user program



Monolithic System (Cont.)

- OS code is one big object program (in machine language).
- It's source code may be logically divided into
  - Main body
  - System call service routines
  - Utility procedures which help service routines

A simple structuring model for a monolithic system



What is wrong with such a system?

- OS is one large program.
- Anytime you add a new device you must
  1. get a device driver for the device
  2. recompile the kernel with the new device driver
  3. reboot the machine so the new kernel will be used

Example Structures

Layered System

"THE" OS (a batch OS)

Layer	Function
5	The operator
4	User programs
3	Input/output management
2	Operator-process communication
1	Memory and drum management
0	Processor allocation and multiprogramming

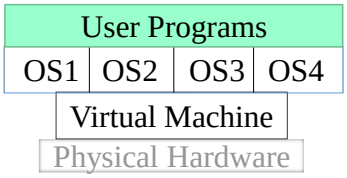
Layered System: bottom up

- Process switching, multi programming, CPU scheduling
- Memory and swap space (disk) management ("segment controller")
- Message interpretation, job control (JCL) functions
- I/O management (virtual peripherals)
- User programs
- Operator

Layered System (Cont.)

- Synchronisation between layers : Hardware and software (semaphores) interrupts
- Each layer provides some sort of a "virtual machine"

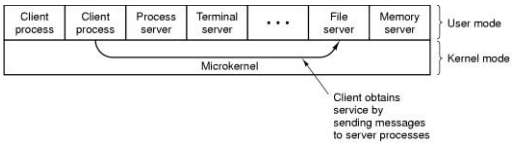
Virtual Machines



Different OSs can work on the same machine at the same time…

Micro-Kernel (Client/Server) Model

- The necessary functions of the OS form a core, known as ***Kernel*** *or* ***microkernel***.



Micro-Kernel

- Necessary functions:
  - memory management
  - basic CPU management
  - inter-process communication (messages)
  - I/O support
- Other functionality provided by user level processes, whicj are still components of OS.

Characteristics of Micro_kernel

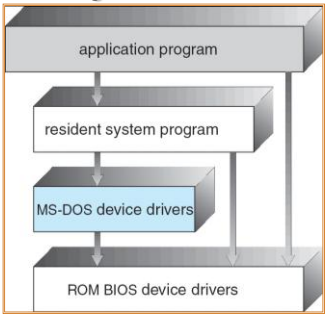
- Communication between os components and user level subsystems makes use of message passing
- Easy to replace server processes
- Easier to write and port OS, as an important portion of it is outside the kernel
- Design is more suitable for distributed systems, as different components (file system, memory management, etc.) are fairly independent
- Higher performance for lower level, but possible lower performance for higher levels

Few OS Examples: very short introduction

MS-DOS: Simplest OS example

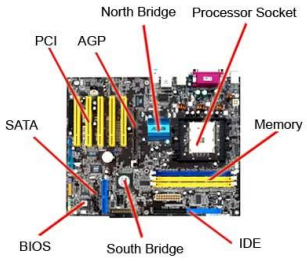
- MS-DOS (MicroSoft Disc Operating System) – written to provide the most functionality in the least space
  - Not divided into well defined modules
    - Although MS-DOS seems to have some structure, its interfaces and levels of functionality are not well separated

MS-DOS Layer Structure



BIOS example: A chip or a set of chips the motherboard

A typical motherboard



Functions of a PC BIOS

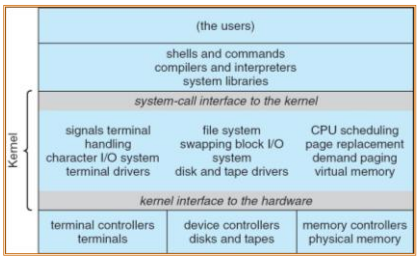
- POST - Test the computer hardware and make sure no errors exist before loading the operating system.
- Bootstrap Loader - Locate the operating system
- BIOS drivers - Low level drivers that give the computer basic operational control over your computer's hardware.
- BIOS or CMOS Setup - Configuration program, to configure hardware settings, such as computer passwords, time, and date.

UNIX structuring

The UNIX OS consists of two separable parts

- Systems programs, in the form of libraries
- The kernel
  - Everything below the system-call interface and above the physical hardware
  - Provides core file system, CPU scheduling, memory management, and other main operating-system functions

UNIX System Structure



Microkernel OS Structure

- Moves as much from the kernel into "*user*" space
- Communication between user modules is done 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 kernel space communication

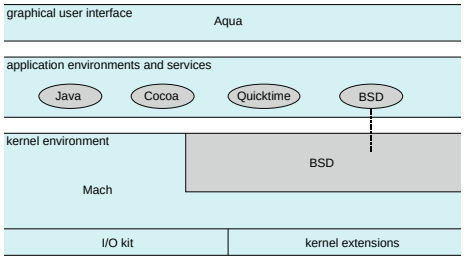
Modules

- Most modern operating systems implement 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 flexibility

Hybrid Systems

- Most modern operating systems actually do not follow just one pure model
  - Hybrid combines multiple approaches to address performance, security, usability needs. Kernel is often monolithic.
- Apple Mac OSs are hybrid.
- Below is kernel consisting of Mach microkernel and BSD Unix parts, plus I/O kit and dynamically loadable modules (called **kernel extensions**)

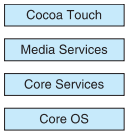
Mac OS X Structure



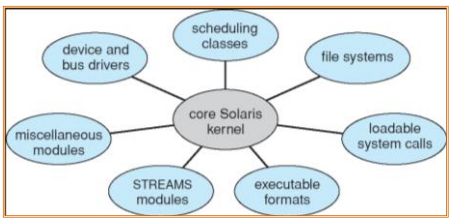
Below is kernel consisting of Mach microkernel and BSD Unix parts, plus I/O kit and dynamically loadable modules (called kernel extensions)

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



Solaris(UNIX) Modular Approach



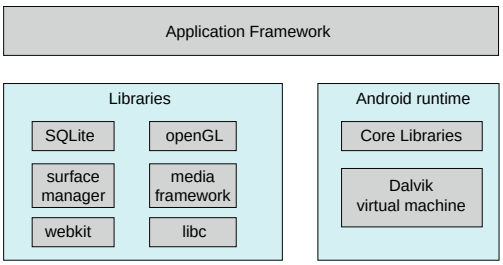
Android

- Developed by Open Handset Alliance (mostly Google)
  - Open Source
- Similar to IOS
- Based on Linux kernel with modification
  - Provides process, memory, device-driver management
  - Adds power management

Android (cont.)

- 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
- Dalvik has been replaced by ART(AndroidRunTime) starting from android 5.

Android Architecture1



Virtual Machines

- It hides hardware.
- A virtual machine provides a machine interface *based on* the underlying bare hardware.
- A operating system on VM can create the illusion of multiple processes, each executing on its own processor with its own (virtual) memory.

Virtual Machines (Cont.)

- The resources of the physical computer are shared to create the virtual machines
  - CPU scheduling can create the appearance that users have their own processor
  - The file system can provide virtual readers and virtual line printers
  - A normal user time-sharing terminal serves as the virtual machine operator's console

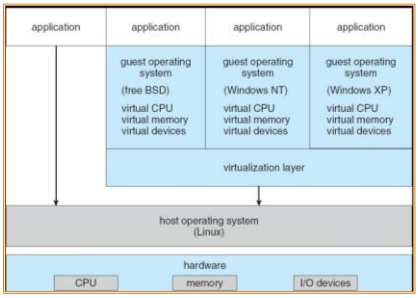
Virtual Machines (Cont.)

- The virtual-machine concept provides complete protection of system resources since each virtual machine can be isolated from all other virtual machines.
- A virtual-machine system is a perfect vehicle for operating-systems research and development.

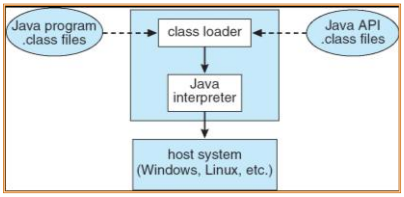
Virtual Machines (Cont.)

- System development is done on the virtual machine, instead of on a physical machine and so does not disrupt normal system operation.
- The virtual machine concept is difficult to implement due to the effort required to provide an *exact* duplicate to the underlying machine hardware.

VMware Architecture



The Java Virtual Machine

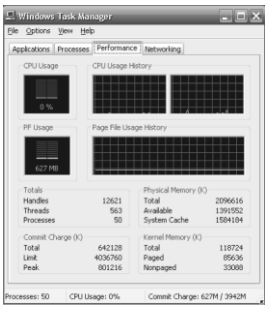


Operating-System Debugging

- **Debugging** is finding and fixing errors, or **bugs**
- OSs generate **log files** containing error information
- Failure of an application can generate **core dump** file capturing memory of the process
- Operating system failure can generate **crash dump** file containing kernel memory
- Beyond crashes, performance tuning can optimize system performance
  - Sometimes using ***trace listings*** of activities, recorded for analysis
  - **Profiling** is periodic sampling of instruction pointer to look for statistical trends

Performance Tuning

- Improve performance by removing bottlenecks
- OS must provide means of computing and displaying measures of system behavior



Operating System Generation

- Operating systems are designed to run on any machine; the system must be configured for each specific computer site.
- SYSGEN program obtains information concerning the specific configuration of the hardware system.
- *Booting* – starting a computer by loading the kernel.
- *Bootstrap program* – code stored in ROM that is able to locate the kernel, load it into memory, and start its execution.

System Boot

- Operating system must be made available to hardware so hardware can start it
  - Small piece of code – **bootstrap loader**, locates the kernel, loads it into memory, and starts it
  - Sometimes two-step process where **boot block** at fixed location loads bootstrap loader
  - When power initialized on system, execution starts at a fixed memory location
    - Firmware used to hold initial boot code