Computer System Structure

Reading:

Silberschatz

chapter 3

Additional Reading:

Stallings

chapter 2

Outline

- OS Services
- User Interfaces
- System Call
- OS Design
- OS Implementation
- System Structure
  - MSDOS
  - UNIX
  - OS/2
  - Win NT
- Virtual Machines
- System Installation

Aspects of OS

- OS from several vantage points
  - *Services that system provides*
  - *Available interfaces to users & programmers*
  - *Components and interconnections*

OS Services

- Program execution – System capability to load a program into memory and to run it
- I/O operations – User programs cannot execute I/O operations directly, OS provides means to perform I/O
- File-system manipulation – Program capability to read, write, create, and delete files
- Communications – Exchange of information between processes running on same/different computers, *shared memory* or *message passing*
- Error detection – Ensure correct computing by detecting errors in the CPU and memory hardware, in I/O devices, or in user programs

Additional OS Services

Additional functions to ensure efficient system operations

- Resource allocation – allocating resources to multiple users/jobs running at the same time
- Accounting – keep track of and record which users use how much and what kinds of resources
- Protection – ensuring that all access to system resources is controlled

User OS Interface

- Two approaches
  - Command-line interface (command interpreter)
  - Graphical user interface (GUI)
- Command Interpreter
  - Main function – Get and execute the next command
    - MSDOS and UNIX shell
  - Multiple command interpreter in UNIX and Linux
    - *Bourne shell*, *C shell*, *Bourne-Again shell*, *Korn shell*, etc.
  - Two approaches to implement the commend execution
    - CI itself contains the code to interpret the command
    - Implement most commands through system program – UNIX *e.g.* `rm file.txt`, new commands by adding new files

User OS Interface

- GUI
  - Mouse-based window and menu system
  - User friendly

  - UNIX systems – dominated by CLI traditionally
  - Various GUI interfaces in commercial version of UNIX
    - *Common Desktop Environment* (*CDE*), *X-Windows*, *etc*.
  - Significant GUI developments from open source projects
    - *K Desktop environment* (*KDE*), *GNOME desktop*
    - Many are available under open-source license
    - Linux and various UNIX systems
  - Command-line or GUI ?
    - Powerful shell interface (many UNIX users)
    - Windows user friendly GUI (many window users)

System Calls

- Enter OS and perform a privileged operation

  Interface to the services made available by OS

- Key Points
  - Difference between *procedure call* and *system call*
  - Generally available as routines written in C and C++
  - Some low-level tasks may need to be written using assembly language
  - How system calls are used?
    - Example – read data from file and write to another
    - Read file name (I/O system calls), check error, message on console, etc.
    - Several system calls to perform simple operation
  - Application Programming Interface (API)
    - API – set of function available to an application programmer
  - Most common APIs
    - Win32 API for Windows system
    - POSIX API for most versions of UNIX, Linux and Mac OS X
    - Java API for designing programs for JVM
  - Behind the scene? Actual system calls are invoked – portability
  - Most details of OS interface are hidden by API, managed by run-time support library

System Call groups

System calls can be grouped into five major categories

- Process Control
  - fork(), exec(), wait(), abort()
- File manipulation
  - chmod(), link(), stst(), creat()
- Device manipulation
  - open(), close(), ioctl(), select()
- Information maintenance
  - time(), act(), gettimeofday()
- Communications
  - socket(), accept(), send(), recv()

OS design and Implementation

- Design Goals
  - Type of hardware and type of system
  - User Goals
    - *Convenient, easy to learn/use, reliable, safe and fast*
  - System Goals
    - *Easy to design, implement/maintain, flexible, reliable, error-free and efficient*

    (vague requirements! has several interpretations)
- Implementation
  - Traditionally written in assembly language
    - Now mostly written in high-level languages such as C or C++
    - Linux and Windows XP - mostly in C, small section of assembly code device drivers
  - Advantages of implementing in HLL
    - *Compact, fast and easier to understand/debug*
    - Easier to port to some other hardware
  - Disadvantages of implementing in HLL
    - Reduced speed and increased storage requirements

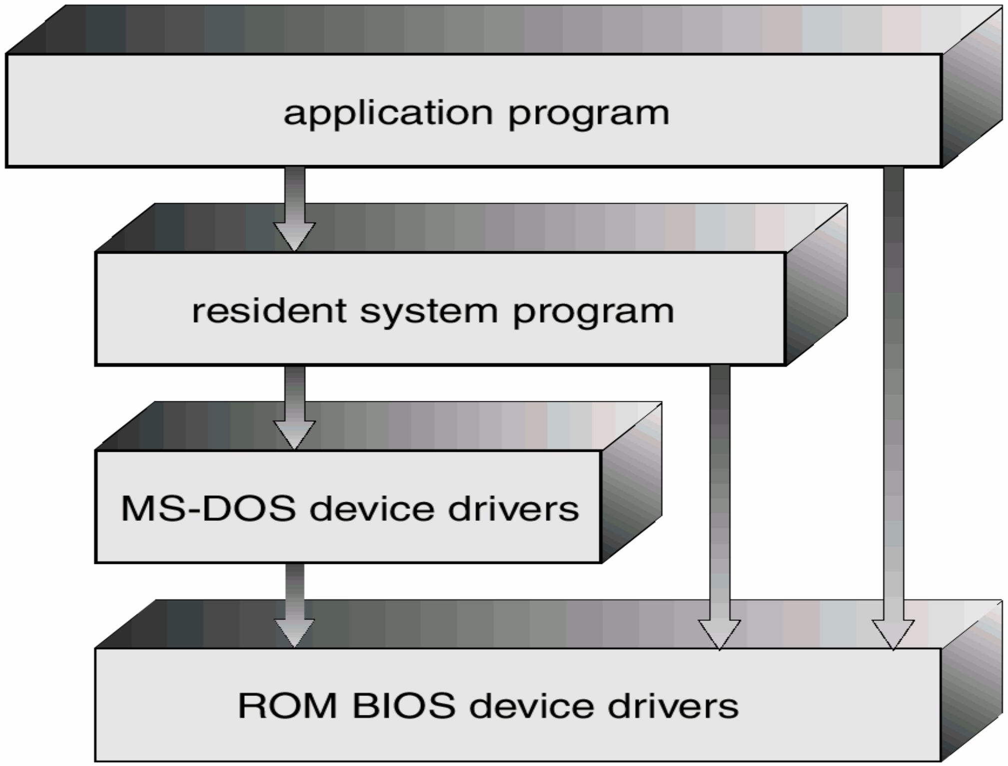
System Structure

- Possible ways to structure an operating system
  - Simple, single-user
    - *MSDOS, MacOS, Windows*
  - Monolithic, multi-user
    - *UNIX, Multics, OS/360*
  - Hybrid
    - *Win NT*
  - Virtual Machine
    - *IBM VM/370*
  - Client/Server (microkernel)
    - *Chorus/Mix*

Structure of MSDOS

- MSDOS – written to provide the most functionality in the least space
  - Not divided into modules
  - Interfaces and levels of functionality are not well separated (*e.g*. application programs access I/O)
  - Written for Intel 8088, No dual mode and no hardware protection

Structure of MSDOS

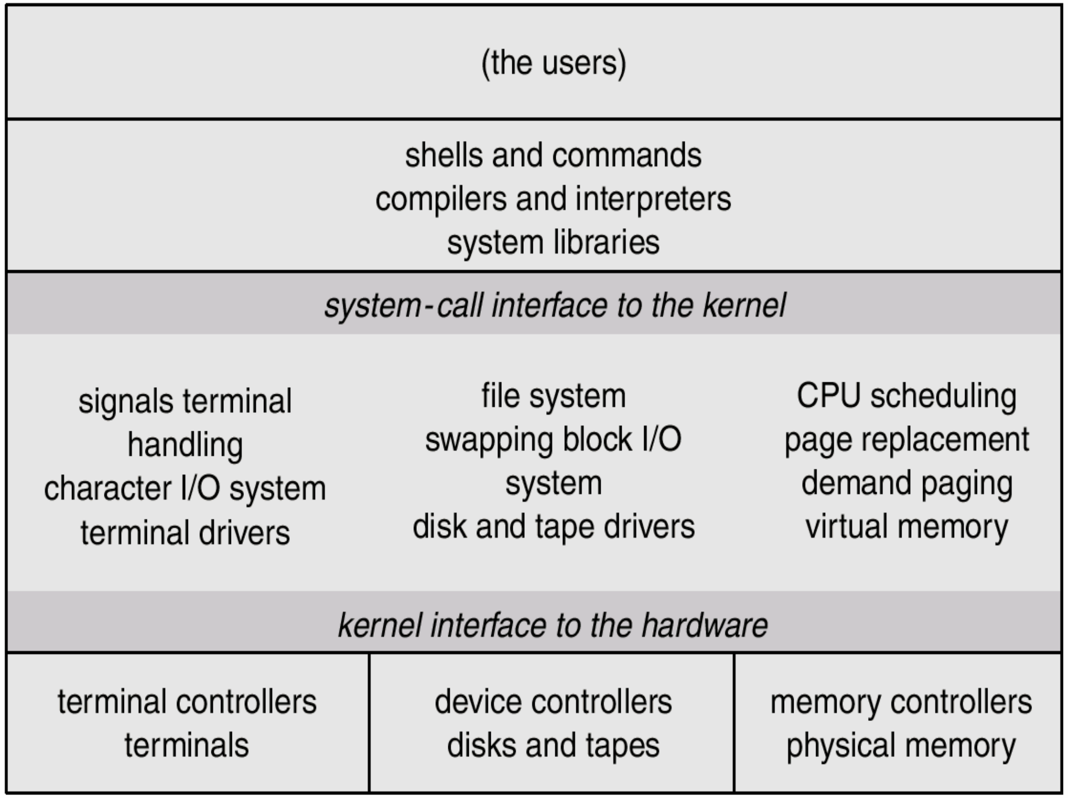


MSDOS Layer Structure

UNIX system Structure

- The original UNIX OS 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 file system, CPU scheduling, memory management, and other OS functions through system calls
    - Enormous amount of functionality into one level

UNIX System Structure



UNIX System Structure

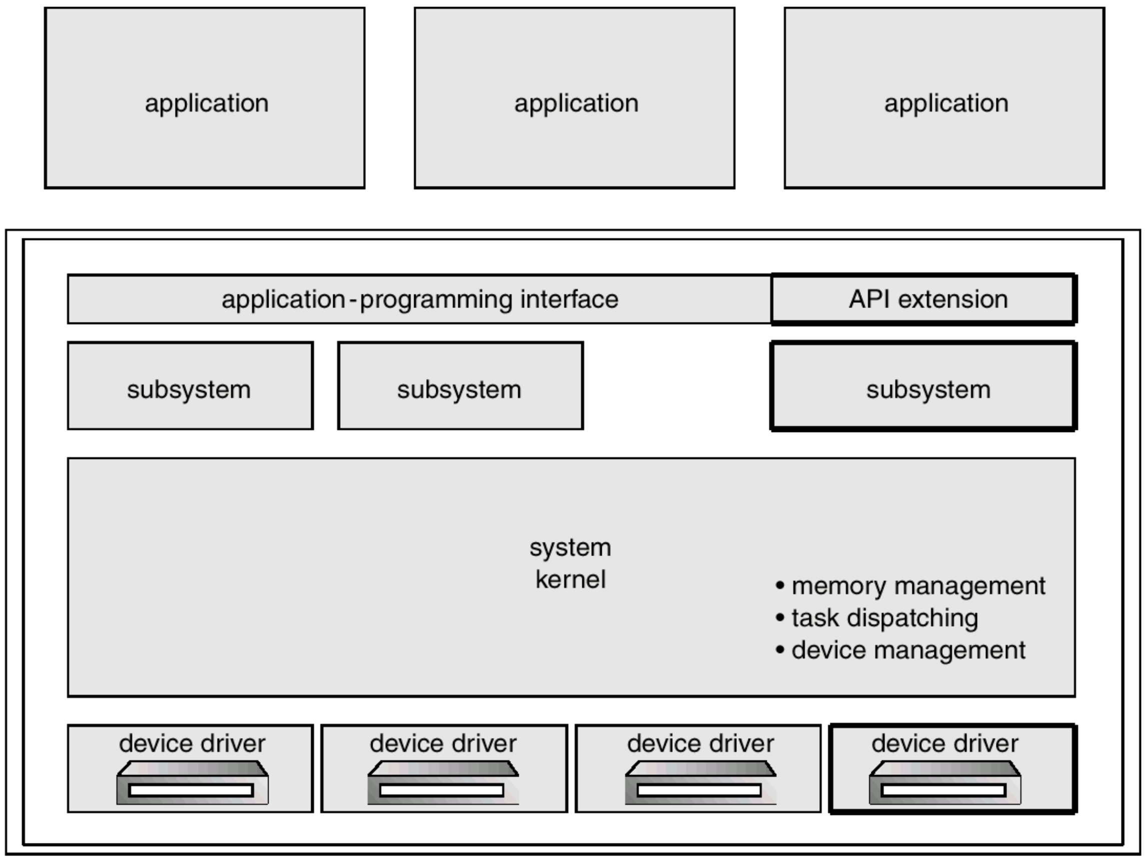
Layered Approach

- The modularization of System – Layered Approach
- The OS is divided into a number of layers (levels) The bottom layer (layer 0), is the hardware; the highest (layer N) is the user interface
- Layers are selected such that each uses functions and services of only lower-level layers
- Problem – more overhead, less efficient

- OS/2 descendent of MSDOS – Multitasking and dual mode operations
  - Advantage – direct user access to low-level facilities is prohibited

- Example – Windows NT
  - First release highly layered – low performance Vs Windows 95
  - Windows NT 4.0 – Moved layers from user space to kernel space

Layered Approach

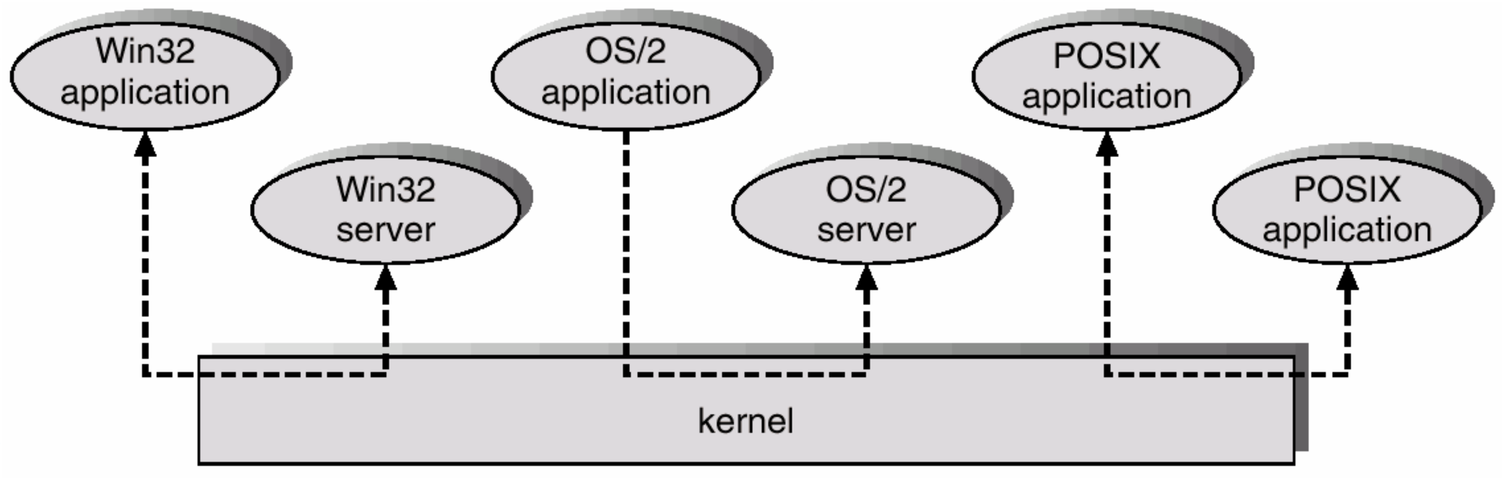


OS/2 Layer Structure

Microkernel System Structure

- Removing all nonessential components from kernel, implementing as user-level programs
- Moves as much from the *kernel* into *user* space
  - Resulting smaller kernel - *Microkernel*
  - Minimal process and memory management +
  - Communication facility using message passing
- Benefits
  - Easier to extend a microkernel
  - Easier to port OS to new architectures
  - More reliable and secure

Windows NT Client-Server Structure

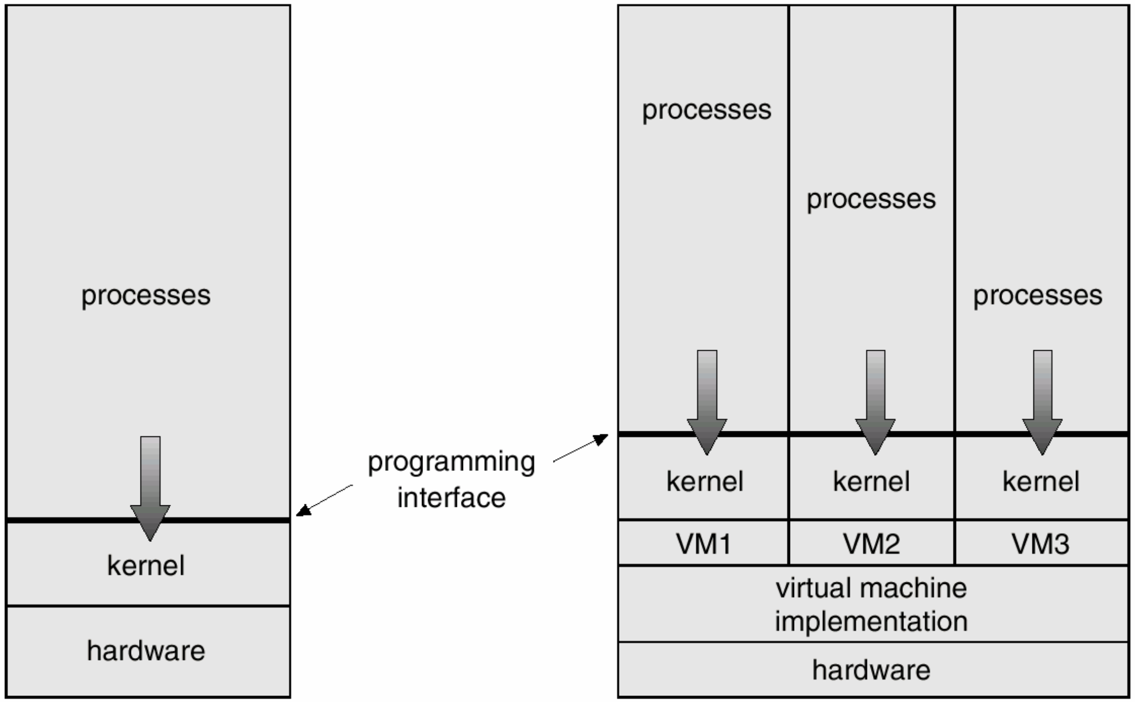


Hybrid Structure of Windows NT

Virtual Machines

- A *virtual machine* is logical conclusion of the layered approach
  - Hardware and OS kernel are treated as hardware
  - The OS creates illusion of multiple process, each executing on its own processor with its own memory
- The resources of physical computer are shared to create the virtual machines
  - CPU scheduling can create the appearance that users have their own processor
  - Virtual Memory techniques create illusion of processors own memory
  - Spooling and a file system can provide virtual card readers and virtual line printers
  - A normal user time-sharing terminal serves as the virtual machine operator's console

Virtual Machines



Non-virtual Machine

Virtual Machine

Virtual Machines

- Complete protection of system resources - Each virtual machine is isolated another (but isolation prevents direct sharing of resources)
- System development on virtual machine, instead of on a physical machine, does not disrupt normal system operation
- The virtual machine concept is difficult to implement - Efforts required to provide an *exact* duplicate to the underlying machine

Java Virtual Machine

- Compiled Java programs are platform-neutral bytecodes executed by Java Virtual Machine (JVM)
- JVM consists of

  - class loader

  - class verifier

  - runtime interpreter
- Just-In-Time (JIT) compilers increase performance

System Generation (installation)

- OS are designed to run on any of a class of machines. Information required for configuring for each specific computer
  - What CPU type is used? Options?
  - Number of CPUs?
  - How much memory is available?
  - What devices are available?
  - OS parameters (max # users, buffer size, max # devices, *etc*.)
  - OS features
    - Networking
    - Other file systems
    - Servers

System Generation (installation)

- How does the hardware know where the kernel is? or how to load the kernel?
  - 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

Protection Level in Intel Processor

- Hardware level → Intel provides 4 protection levels
- **Level 0** → Most Protected (for use of kernel)
- **Level 1** → Intended for non-kernel parts of OS
- **Level 2** → Offered for device drivers (Most needy of protection from user applications
- **Level 3** → Least protected and intended for use by user applications
- Each data in memory is also tagged
  - Program running at certain level can only access data that is in same level or higher level
  - Most OS (Linux, UNIX, Windows) → Only 2 of 4 levels