CCS3300 Operating System

Assoc. Prof. Ts. Dr. Nur Izura Udzir

1

1

Lecturer's profile

Assoc. Prof. Ts. Dr. Nur Izura Udzir

- Information Security Group
- Department of Computer Science
- Since 1998 – Lecturer at Faculty of Computer Science and Information Technology, UPM

Education:

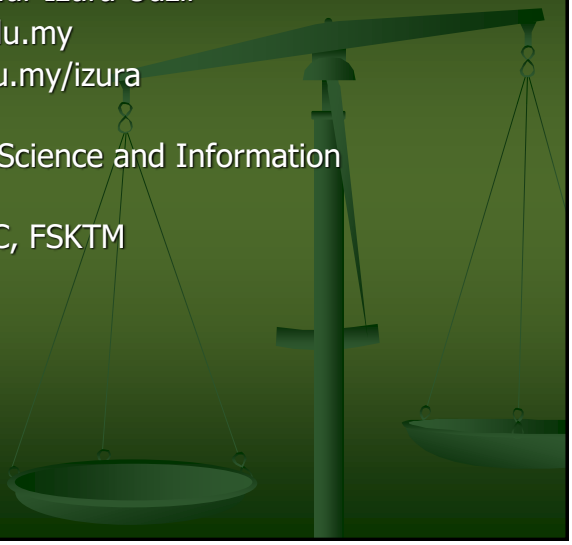
- Bac. Computer Sc. – UPM (1995)
- M.Sc (Computer Sc.) – UPM (1998)
- Ph.D (Computer Sc.) – York, UK (2006)

2

Lecturer's Contact Info

- Assoc. Prof. Ts. Dr. Nur Izura Udzir
- Email: izura@upm.edu.my
- Web: profile.upm.edu.my/izura

- Faculty of Computer Science and Information Technology, UPM
- Office: C1.07, Block C, FSKTM
- Tel: ~~03-97691708~~



3

Chapter 1: Introduction

4

4

Learning Objectives

At the end of the chapter, the students are able to:

- understand the major operating systems components
- understand basic computer system organization
- describe the services an operating system provides to users, processes, and other systems
- discuss the various ways of structuring an operating system
- explain how operating systems are installed and customized and how they boot

5

5

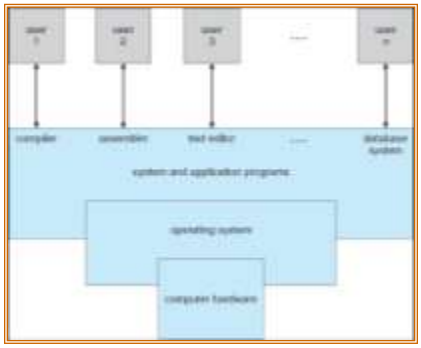
What is an Operating System?

- A program that acts as an intermediary between a user of a computer and the computer hardware.
- Operating system goals:
  - Execute user programs and make solving user problems easier.
  - Make the computer system convenient to use.
- Use the computer hardware in an efficient manner.

6

6

Computer System Structure



Computer system can be divided into four components

- **Hardware** – provides basic computing resources
  - CPU, memory, I/O devices
- **Operating system** - controls and coordinates use of hardware among various applications and users
- **Application programs** – define the ways in which the system resources are used to solve the computing problems of the users
  - Word processors, compilers, web browsers, database systems, video games
- **Users** - People, machines, other computers

7

7

Operating System Definition

- OS is a **resource allocator**
  - Manages all resources
  - Decides between conflicting requests for efficient and fair resource use
- OS is a **control program**
  - Controls execution of programs to prevent errors and improper use of the computer

8

8

Operating System Definition (Cont.)

- No universally accepted definition
- "Everything a vendor ships when you order an operating system" is good approximation
  - But varies wildly
- "The one program running at all times on the computer" is the **kernel.** Everything else is either a system program (ships with the operating system) or an application program

9

9

Computer Startup

10

10

Computer Startup

- 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
- **bootstrap program** is loaded at power-up or reboot
  - Typically stored in ROM or EEPROM , generally known as **firmware**
  - Initializes all aspects of system
  - Loads operating system kernel and starts execution

*EEPROM = electrically erasable programmable read-only memory

11

11

ROM BIOS Chip

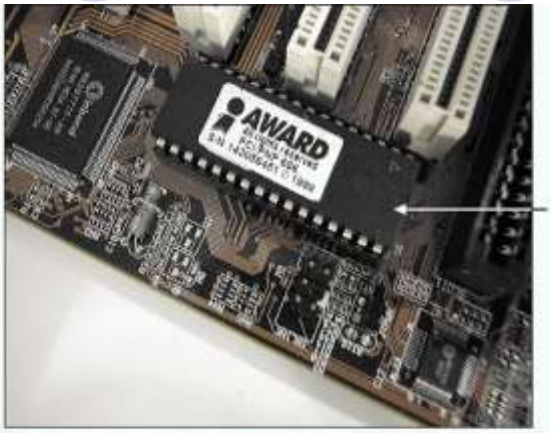


Figure 1-34 The ROM BIOS chip on the motherboard contains the programming to start up the PC as well as to perform many other fundamental tasks

BIOS = basic input/output system

12

Booting Up Your Computer

- Hard (cold) boot versus soft (warm) boot

  A cold boot is accomplished by powering up the computer from a shut down state. A warm boot is done when you need to restart while the computer is still powered but unresponsive, (for example, during a freeze up that isn't resolved with a force quit). You do this by holding down the Control and Command keys simultaneously then pressing the Power Up key (or the on/off key on a laptop).

- Startup BIOS is in control when boot process begins
  - Turns control over to the OS

Steps in the Boot Process

1. Startup BIOS runs power-on self test (POST) and assigns resources
2. ROM BIOS startup program searches for and loads an OS
3. OS configures the system and completes its own loading
4. Application software is loaded and executed

Boot Step 1: POST

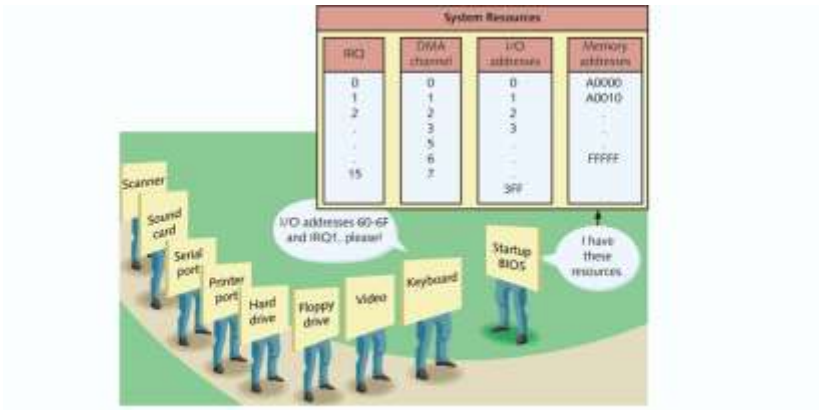


Figure 3-1 Boot Step 1: ROM BIOS startup program surveys hardware resources and needs and assigns system resources to satisfy those needs

How the BIOS Finds and Loads the OS

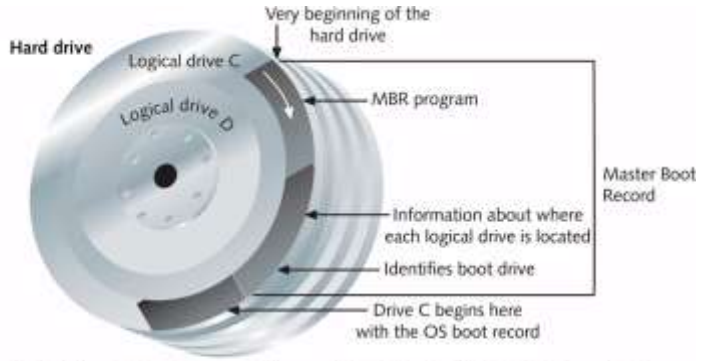


Figure 3-2 A hard drive might contain more than one logical drive; the Master Boot Record at the beginning of the drive contains information about the location of each logical drive, indicates which drive is the boot drive, and holds the master boot program that begins the process of loading an operating system

How the BIOS Finds and Loads the OS (continued)

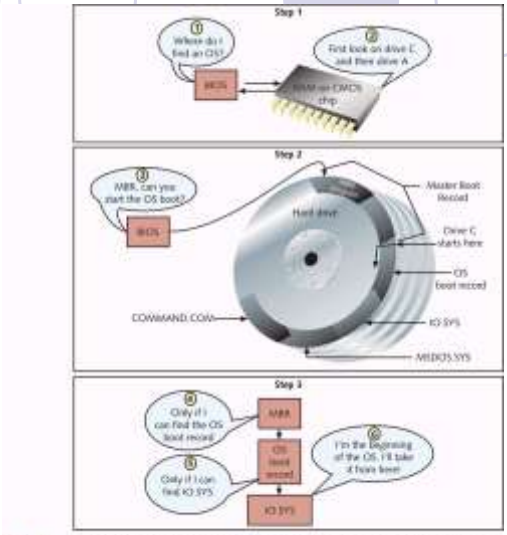
- BIOS executes MBR program
  - Turns to partition table to find OS boot record
- Program in OS boot record attempts to find a boot loader program for OS
  - Ntldr (Windows NT/2000/XP)
  - Io.sys (Windows 9x)

How the BIOS Finds and Loads the OS (continued)

A **Master Boot Record (MBR)**, or **partition sector**, is the 512-byte boot sector that is the first sector of a partitioned data storage device such as a hard disk. (The boot sector of a non-partitioned device is a Volume Boot Record, which is also the term used to describe the first sector of an individual partition on a partitioned device)

It is sometimes used for bootstrapping operating systems, sometimes used for holding a disc's partition table, and sometimes used for uniquely identifying individual disc media; although on some machines it is entirely unused and redundant.

Boot Step 2: Loading the OS



Boot Step 2: BIOS searches for and begins to load an operating system (in this example, Windows 9x is the OS)

Loading the MS-DOS Core of Windows 9x

- Brings OS to real-mode command prompt
- Relevance: Real-mode DOS core often used as a troubleshooting tool

Loading the MS-DOS Core of Windows 9x (continued)

- Files necessary to boot to command prompt
  - Io.sys
  - Msdos.sys
  - Command.com
- To customize 16-bit portion of load process
  - Autoexec.bat (Autoexec.nt – NT, 2000, XP)
  - Config.sys (Config.nt – NT, 2000, XP)

Boot Step 3: OS Initializes Itself

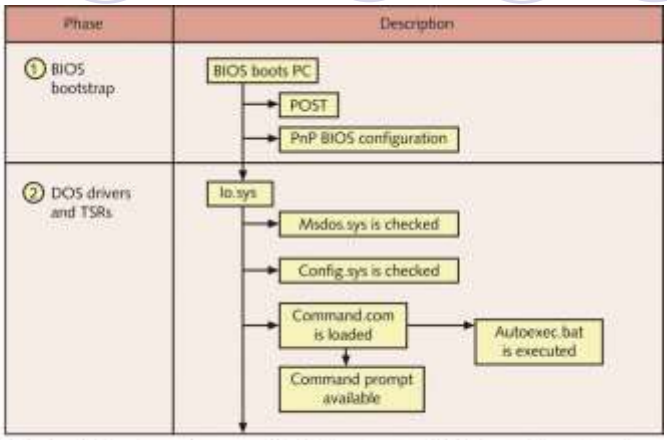


Figure 3-4 Boot Step 3: Operating system completes the boot process; MS-DOS core is loaded and command prompt presented to user

Emergency Startup Disks

- Bootable disks with some utility programs to troubleshoot a failed hard drive
- Each OS provides automated method to create a rescue disk (Windows 9x) or set of disks (Windows 2000)

Emergency Startup Disks (continued)

- Creating a Windows 9x startup disk
  - Add/Remove Programs icon in Control Panel
- Using a Windows 9x startup disk with another OS

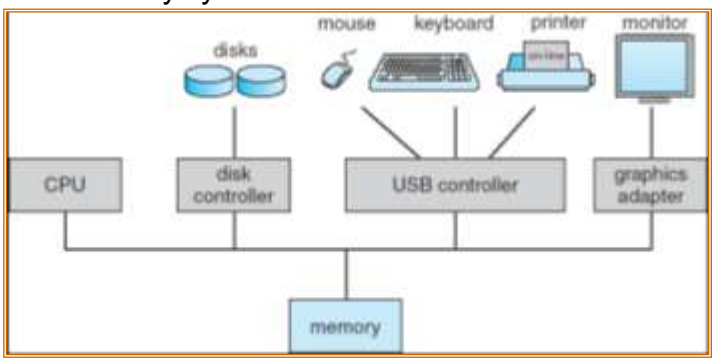
Windows 9x Startup Disks

Figure 3-5 Windows might use the Windows CD to create a startup disk

Computer System Organization

Computer System Organization

- Computer-system operation
  - One or more CPUs, device controllers connect through common bus providing access to shared memory
  - Concurrent execution of CPUs and devices competing for memory cycles

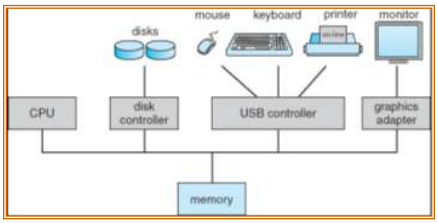


27

27

Basic Elements

- Processor
- Main Memory
  - volatile
  - referred to as real memory or primary memory
- I/O modules
  - secondary memory devices
  - communications equipment
  - terminals
- System bus
  - communication among processors, memory, and I/O modules



28

28

Processor

- Internal registers
  - Memory address register (MAR)
    - Specifies the address for the next read or write
  - Memory buffer register (MBR)
    - Contains data written into memory or receives data read from memory
  - I/O address register
  - I/O buffer register

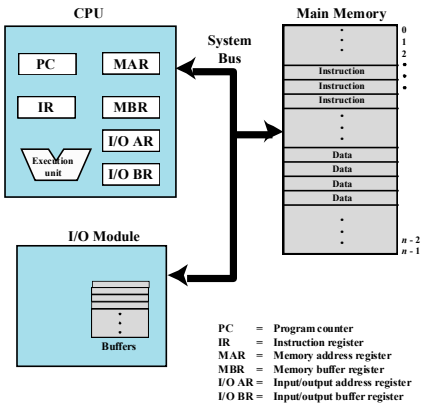


Figure 1.1 Computer Components: Top-Level View

29

29

User-Visible Registers

- Enable programmer to minimize main-memory references by optimizing register use
- May be referenced by machine language
- Available to all programs - application programs and system programs
- Types of registers
  - Data
  - Address
    - Index
    - Segment pointer
    - Stack pointer

30

30

User-Visible Registers

- Address Registers
  - Index
    - Involves adding an index to a base value to get an address
  - Segment pointer
    - When memory is divided into segments, memory is referenced by a segment and an offset
  - Stack pointer
    - Points to top of stack

31

31

Control and Status Registers

- Used by processor to control operating of the processor
- Used by privileged operating-system routines to control the execution of programs

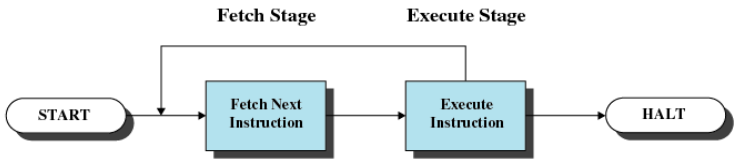
- Program Counter (PC)
  - Contains the address of an instruction to be fetched
- Instruction Register (IR)
  - Contains the instruction most recently fetched
- Program Status Word (PSW)
  - Condition codes
  - Interrupt enable/disable
  - Supervisor/user mode
- Condition Codes or Flags
  - Bits set by the processor hardware as a result of operations
  - Examples: Positive result, Negative result, Zero, Overflow

32

32

Instruction Execution

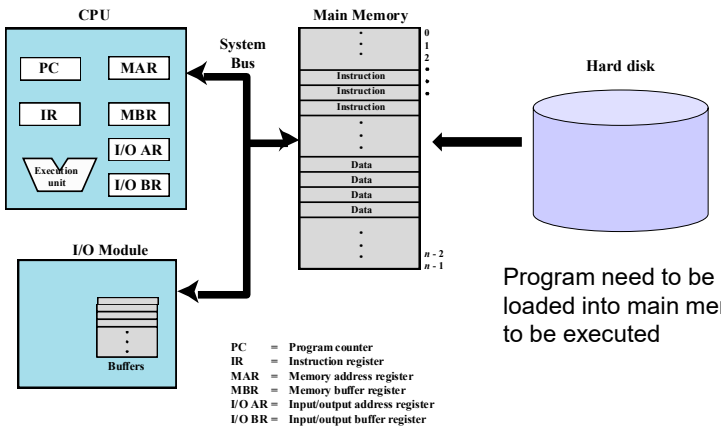
- Two steps
  - Processor fetches (reads) instructions from memory
  - Processor executes each instruction
- The processor fetches the instruction from memory
- Program counter (PC) holds address of the instruction to be fetched next
  - Program counter is incremented after each fetch



33

33

Instruction Fetch and Execute



Program need to be loaded into main memory to be executed

Instructions in program will be fetched one by one from main memory to IR (in CPU) to be executed

34

34

Instruction Register

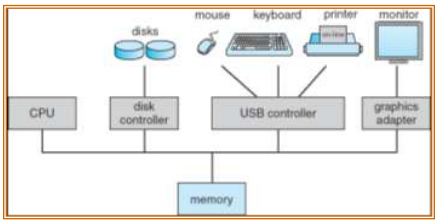
- Fetched instruction is placed in the instruction register
- Categories
  - Processor-memory
    - Transfer data between processor and memory
  - Processor-I/O
    - Data transferred to or from a peripheral device
  - Data processing
    - Arithmetic or logic operation on data
  - Control
    - Alter sequence of execution

35

35

Computer-System Operation

- I/O devices and the CPU can execute concurrently.
- Each device controller is in charge of a particular device type.
- Each device controller has a local buffer.
- CPU moves data from/to main memory to/from local buffers
- I/O is from the device to local buffer of controller.
- Device controller informs CPU that it has finished its operation by causing an *interrupt*.



36

36

Interrupts

- Interrupt the normal sequencing of the processor
- Most I/O devices are slower than the processor
  - Processor must pause to wait for device

Program	Generated by some condition that occurs as a result of an instruction execution, such as arithmetic overflow, division by zero, attempt to execute an illegal machine instruction, and reference outside a user's allowed memory space.
Timer	Generated by a timer within the processor. This allows the operating system to perform certain functions on a regular basis.
I/O	Generated by an I/O controller, to signal normal completion of an operation or to signal a variety of error conditions.
Hardware failure	Generated by a failure, such as power failure or memory parity error.

Classes of interrupts

37

37

Common Functions of Interrupts

- Interrupt transfers control to the interrupt service routine generally, through the *interrupt vector*, which contains the addresses of all the service routines.
- Interrupt architecture must save the address of the interrupted instruction.
- Incoming interrupts are *disabled* while another interrupt is being processed to prevent a *lost interrupt*.
- A *trap* is a software-generated interrupt caused either by an error or a user request.
- An operating system is *interrupt* driven.

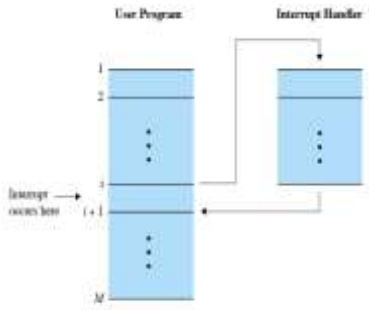


Figure 1.6 Transfer of Control via Interrupts

38

38

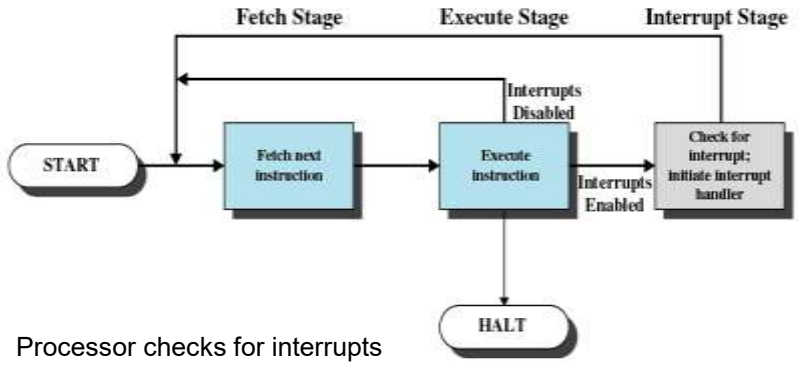
Interrupt Handling

- When CPU is interrupted, it stops what it is doing and immediately transfers execution to a fixed location.
- The operating system preserves the state of the CPU by storing registers and the program counter.
- Determines which type of interrupt has occurred:
  - *polling*
  - *vectored* interrupt system
- Separate segments of code determine what action should be taken for each type of interrupt

39

39

Interrupt Cycle

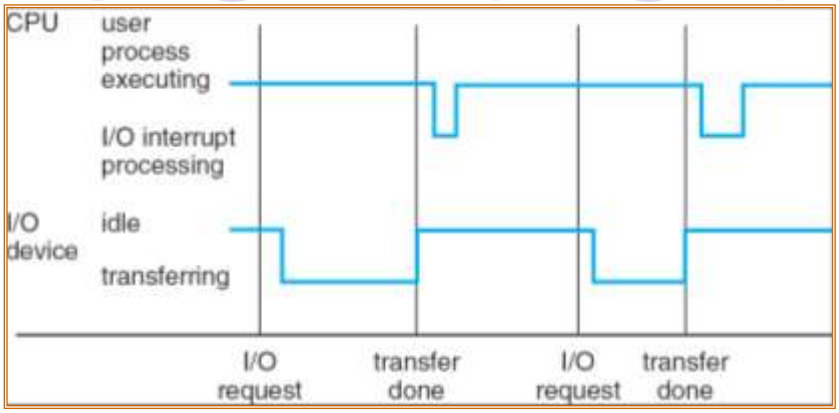


- Processor checks for interrupts
- If no interrupts fetch
  the next instruction for the current program
- If an interrupt is pending, suspend execution of the current program, and execute the interrupt-handler routine

40

40

Interrupt Timeline



41

41

I/O Structure

Synchronous

- After I/O starts, control returns to user program only upon I/O completion.
  - Wait instruction idles the CPU until the next interrupt
  - Wait loop (contention for memory access).
  - At most one I/O request is outstanding at a time, no simultaneous I/O processing.

Asynchronous

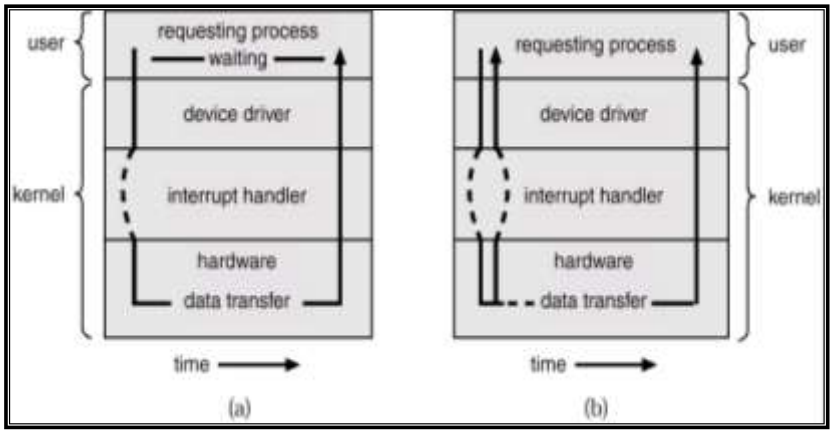
- After I/O starts, control returns to user program without waiting for I/O completion.
- *System call* – request to the operating system to allow user to wait for I/O completion.
- *Device-status table* contains entry for each I/O device indicating its type, address, and state.
- Operating system indexes into I/O device table to determine device status and to modify table entry to include interrupt.

42

42

Two I/O Methods

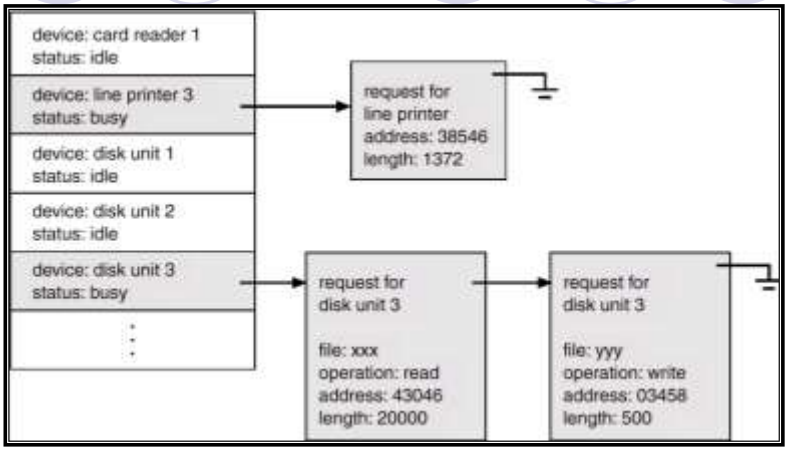
Synchronous Asynchronous



43

43

Device-Status Table



44

44

Storage Structure

- **Main memory** – the only large storage media that the CPU can access directly.
- **Secondary storage** – extension of main memory that provides large nonvolatile storage capacity.
  - Magnetic disks – rigid metal or glass platters covered with magnetic recording material
  - Disk surface is logically divided into *tracks*, which are subdivided into *sectors*.
  - The *disk controller* determines the logical interaction between the device and the computer.

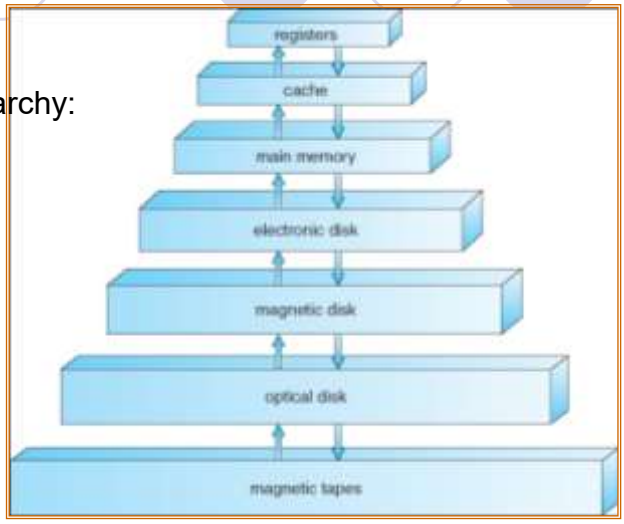
45

45

Storage-Device Hierarchy

Storage systems organized in hierarchy:

- Speed
- Cost
- Volatility



46

46

Caching

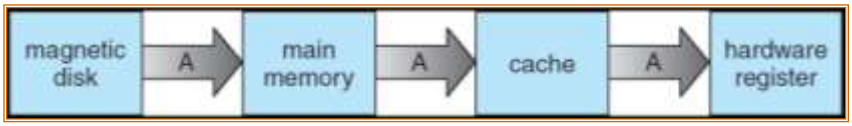
- *Caching* – copying information into faster storage system; main memory can be viewed as a last *cache* for secondary storage.
- Important principle, performed at many levels in a computer (in hardware, operating system, software)
- Information in use copied from slower to faster storage temporarily
- Faster storage (cache) checked first to determine if information is there
  - If it is, information used directly from the cache (fast)
  - If not, data copied to cache and used there
- Cache smaller than storage being cached
  - Cache management important design problem
  - Cache size and replacement policy

47

47

Migration of Integer A from Disk to Register

- Multitasking environments must be careful to use most recent value, not matter where it is stored in the storage hierarchy



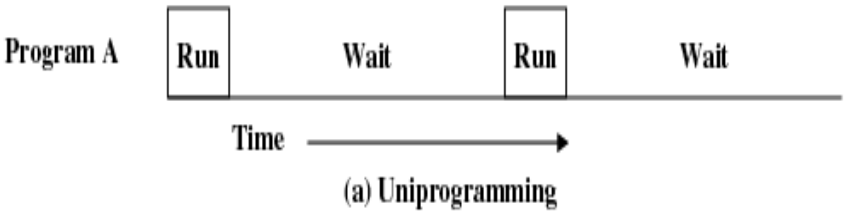
- Multiprocessor environment must provide cache coherency in hardware such that all CPUs have the most recent value in their cache
- Distributed environment situation even more complex
  - Several copies of a datum can exist

48

48

Uniprogramming

- Processor must wait for I/O instruction to complete before preceding



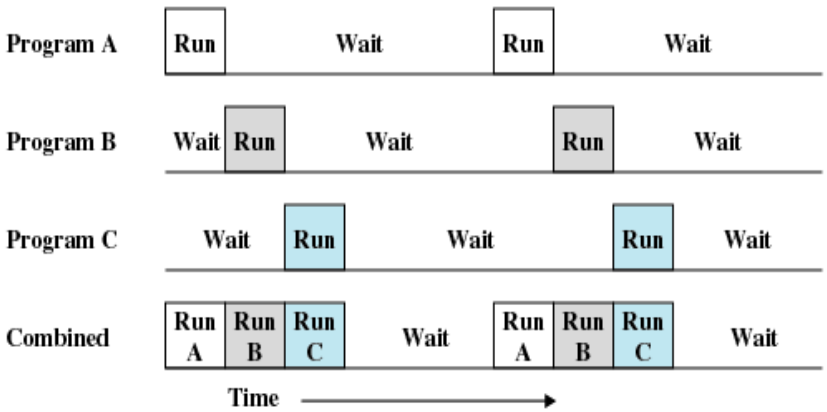
(a) Uniprogramming

49

49

Multiprogramming

- When one job needs to wait for I/O, the processor can switch to the other job



(c) Multiprogramming with three programs

50

Multiprogramming

- **Multiprogramming** needed for efficiency
  - Single user cannot keep CPU and I/O devices busy at all times
  - Multiprogramming organizes jobs (code and data) so CPU always has one to execute
  - A subset of total jobs in system is kept in memory
  - One job selected and run via **job scheduling**
  - When it has to wait (for I/O for example), OS switches to another job

51

51

Time Sharing

- Using multiprogramming to handle multiple interactive jobs
- Processor's time is shared among multiple users
- Multiple users simultaneously access the system through terminals
- a number of users would get small slices of computer time, at a rate at which it appeared they were each connected to their own, slower, machine.
- **Timesharing (multitasking)** is logical extension in which CPU switches jobs so frequently that users can interact with each job while it is running, creating **interactive** computing
  - **Response time** should be < 1 second
  - Each user has at least one program executing in memory ⇨**process**
  - If several jobs ready to run at the same time ⇨ **CPU scheduling**
  - If processes don't fit in memory, **swapping** moves them in and out to run
  - **Virtual memory** allows execution of processes not completely in memory

52

52

Time Sharing

- Issues:
  - users (particularly at universities where the systems were being developed) seemed to want to hack the system to get more CPU time. For this reason, security and access control became a major focus of the Multics project in 1965.
  - proper handling of computing resources: users spent most of their time staring at the screen and thinking instead of actually using the resources of the computer, and a time-sharing system should give the CPU time to an active user during these periods.
  - the systems typically offered a memory hierarchy several layers deep, and partitioning this expensive resource led to major developments in virtual memory systems.

53

53

Compatible Time-Sharing System (CTSS)

- First time-sharing system developed at MIT

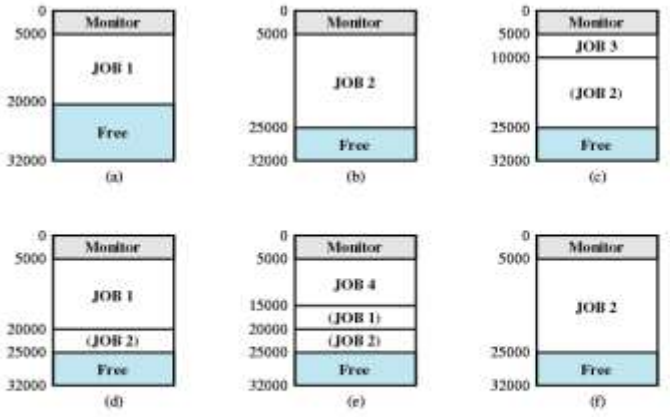


Figure 2.7 CTSS Operation

54

54

Operating-System Operations

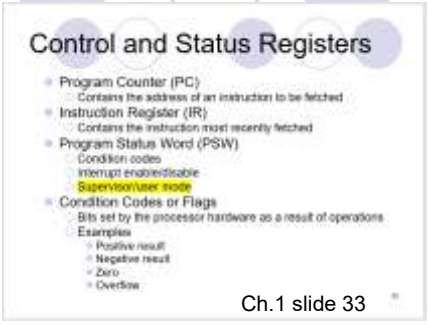
- Interrupt driven by hardware
- Software error or request creates **exception** or **trap**
  - Division by zero, request for operating system service
- Other process problems include infinite loop, processes modifying each other or the operating system
- **Dual-mode** operation allows OS to protect itself and other system components
  - **User mode**
  - **Kernel mode**

55

55

Modes of Execution

- User mode
  - Less-privileged mode
  - User programs typically execute in this mode
- Kernel mode, system mode, control mode, or supervisor mode
  - More-privileged mode
  - Kernel of the operating system



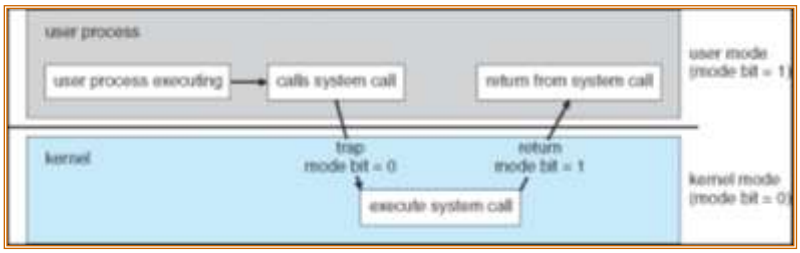
Mode bit provided by hardware
Provides ability to distinguish when system is running user code or kernel code
Some instructions designated as **privileged**, only executable in kernel mode
System call changes mode to kernel, return from call resets it to user

56

56

Transition from User to Kernel Mode

- Timer to prevent infinite loop / process hogging resources
  - Set interrupt after specific period
  - Operating system decrements counter
  - When counter zero generate an interrupt
  - Set up before scheduling process to regain control or terminate program that exceeds allotted time



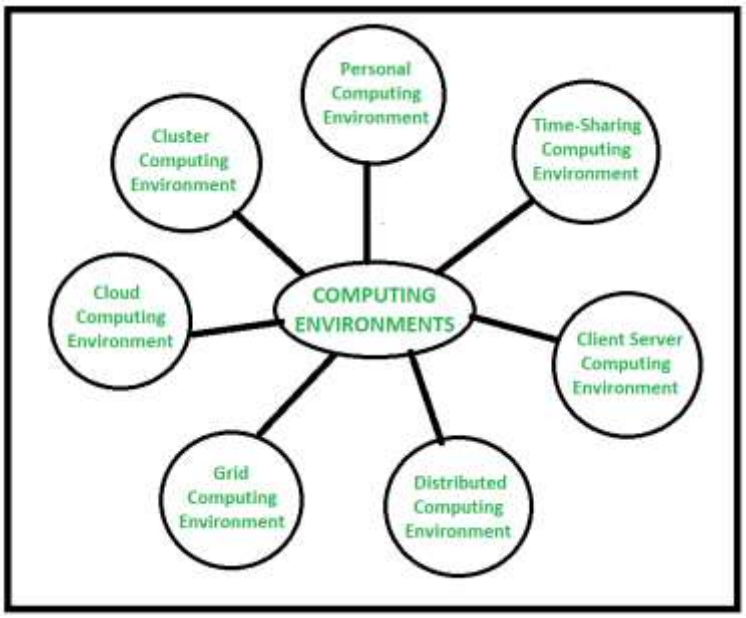
57

57

Computing Environments

58

58



https://www.geeksforgeeks.org/computing-environments/ 59

59

Computing Environments

Computing environments refer to the technology infrastructure and software platforms that are used to develop, test, deploy, and run software applications.

- Mainframe:
  - A large and powerful computer system used for critical applications and large-scale data processing.
- Client-Server:
  - client devices access resources and services from a central server.
- Cloud Computing:
  - resources and services are provided over the Internet and accessed through a web browser or client software.

https://www.geeksforgeeks.org/computing-environments/ 60

60

Computing Environments

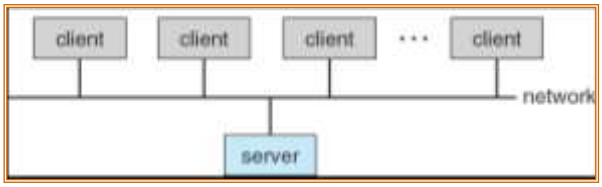
- Mobile Computing:
  - users access information and applications using handheld devices such as smartphones and tablets.
- Grid Computing:
  - resources and services are shared across multiple computers to perform large-scale computations.
- Embedded Systems:
  - software is integrated into devices and products, often with limited processing power and memory.

https://www.geeksforgeeks.org/computing-environments/ 61

61

Client-Server Computing

- Dumb terminals supplanted by smart PCs
- Many systems now **servers**, responding to requests generated by **clients**
  - **Compute-server** provides an interface to client to request services (i.e. database)
  - **File-server** provides interface for clients to store and retrieve files

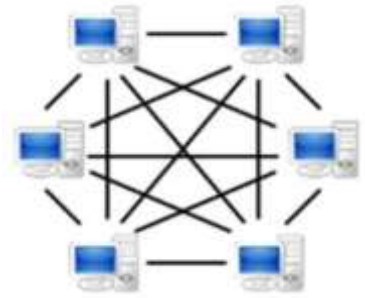


62

62

Peer-to-Peer Computing

- Another model of distributed system
- P2P does not distinguish clients and servers
  - Instead all nodes are considered peers
  - May each act as client, server or both
  - Node must join P2P network
    - Registers its service with central lookup service on network, or
    - Broadcast request for service and respond to requests for service via *discovery protocol*
  - Examples include *Napster* and *Gnutella*



63

63

Operating System Services

64

64

Operating System Services

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 development**
  - Editors and debuggers
- **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)

65

65

Operating System Services

- **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.
  - Obviously, 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 moved by the OS)

66

66

Operating System Services (Cont.)

- **Error detection** – OS needs to be constantly aware of possible errors
  - May occur in the CPU and memory hardware, in I/O devices, in user program
  - Software errors
    - Arithmetic overflow
    - Access forbidden memory locations
  - 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

67

67

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 - Some (such as CPU cycles, main memory, and file storage) may have special allocation code, others (such as I/O devices) may have general request and release code.
- **Accounting -** To keep track of which users use how much and what kinds of computer resources

68

68

Operating System Services (Cont.)

- **Protection and security -** The owners of information stored in a multi-user 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
  - If a system is to be protected and secure, precautions must be instituted throughout it. A chain is only as strong as its weakest link.

69

69

Kernel

- Portion of operating system that is in main memory
- Contains most frequently used functions
- Also called the nucleus

70

70

Early OS Kernel

- OS (and thus, a kernel) is not *required* to run a computer. Programs can be directly loaded and executed on the "bare metal" machine, provided that the authors of those programs are willing to work without any hardware abstraction or OS support.
- Most early computers (1950s and early 1960s) - were reset and reloaded between the execution of different programs.
- Eventually, small ancillary programs such as program loaders and debuggers were left in memory between runs, or loaded from ROM. As these were developed, they formed the basis of what became early OS kernels.
- The "bare metal" approach is still used today on some video game consoles and embedded systems, but in general, newer computers use modern OS and kernels.
- In 1969 the RC 4000 Multiprogramming System introduced the system design philosophy of a small nucleus "upon which OSs for different purposes could be built in an orderly manner", what would be called the microkernel approach.

71

71

Microkernels

- Small operating system core
- Contains only essential core operating systems functions
- Many services traditionally included in the operating system are now external subsystems
  - Device drivers
  - File systems
  - Virtual memory manager
  - Windowing system
  - Security services

72

72

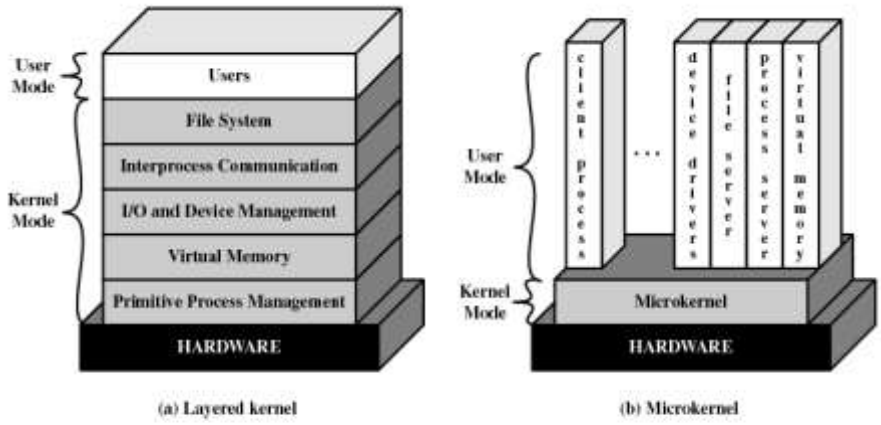
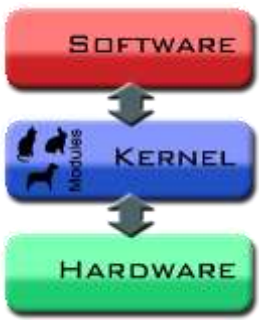


Figure 4.10 Kernel Architecture

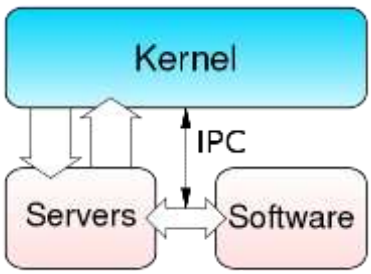
73

73

Monolithic kernel vs. Microkernel



Monolithic kernels



Microkernel

In the microkernel approach, the kernel itself only provides basic functionality that allows the execution of servers, separate programs that assume former kernel functions, such as device drivers, GUI servers, etc.

74

74

Monolithic kernel

- all OS services run along with the main kernel thread, thus also residing in the same memory area.
- provides rich and powerful hardware access.
- "easier to implement a monolithic kernel" than microkernels [Ken Thompson, UNIX developer].
- main disadvantages:
  - dependencies between system components – a bug in a device driver might crash the entire system
  - large kernels can become very difficult to maintain.

75

75

Microkernel System Structure

- Moves as much from the kernel into "*user*" space
- 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

76

76

Microkernel

- a simple abstraction over the hardware, with a set of primitives or system calls to implement minimal OS services (memory management, multitasking, and inter-process communication).
- Other services, including those normally provided by the kernel such as networking, are implemented in user-space programs, referred to as *servers*.
- easier to maintain than monolithic kernels,
- but the large number of system calls and context switches might slow down the system because they typically generate more overhead than plain function calls.
- A microkernel allows the implementation of the remaining part of the OSs a normal application program written in a high-level language, and the use of different OSs on top of the same unchanged kernel. It is also possible to dynamically switch among OSs and to have more than one active simultaneously.

77

77

Benefits of a Microkernel Organization

- Uniform interface on request made by a process
  - Don't distinguish between kernel-level and user-level services
  - All services are provided by means of message passing
- Extensibility
  - Allows the addition of new services
- Flexibility
  - New features added
  - Existing features can be subtracted
- Portability
  - Changes needed to port the system to a new processor is changed in the microkernel - not in the other services

78

78

Benefits of a Microkernel Organization

- Reliability
  - Modular design
  - Small microkernel can be rigorously tested
- Distributed system support
  - Message are sent without knowing what the target machine is
- Object-oriented operating system
  - Components are objects with clearly defined interfaces that can be interconnected to form software

79

79

Development of microkernels

- Although Mach, developed at Carnegie Mellon University from 1985 to 1994, is the best-known general-purpose microkernel, other microkernels have been developed with more specific aims. The L4 microkernel family (mainly the L3 and the L4 kernel) was created to demonstrate that microkernels are not necessarily slow. Newer implementations such as Fiasco and Pistachio are able to run Linux next to other L4 processes in separate address spaces.
- QNX is a real-time OS with a minimalistic microkernel design that has been developed since 1982, having been far more successful than Mach in achieving the goals of the microkernel paradigm.It is principally used in embedded systems and in situations where software is not allowed to fail, such as the robotic arms on the space shuttle and machines that control grinding of glass to extremely fine tolerances, where a tiny mistake may cost hundreds of thousands of dollars.

80

80

User Operating System Interface – Command Line Interface (CLI)

- CLI 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

81

81

User Operating System Interface – Graphical User 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

82

82

User Operating System Interface

- Many systems now include both CLI and GUI interfaces
  - Microsoft Windows is GUI with CLI “command” shell
  - Apple Mac OS X as “Aqua” GUI interface with UNIX kernel underneath and shells available
  - Solaris is CLI with optional GUI interfaces (Java Desktop, KDE)

83

83

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 Program 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), and Java API for the Java virtual machine (JVM)
- Why use APIs rather than system calls?

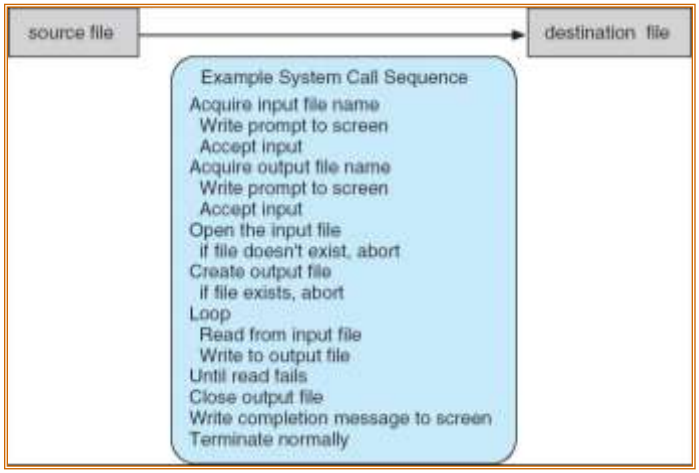
(Note that the system-call names used throughout this text are generic)

84

84

Example of System Calls

- System call sequence to copy the contents of one file to another file

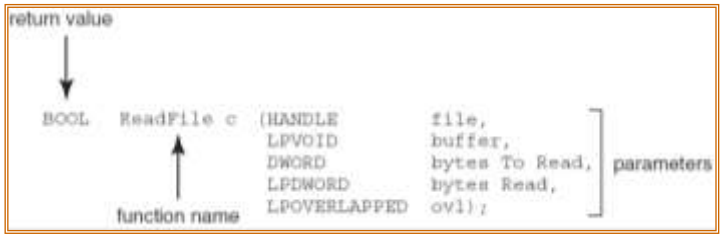


85

85

Example of Standard API

- Consider the ReadFile() function in the
- Win32 API—a function for reading from a file



- A description of the parameters passed to ReadFile()
  - HANDLE file—the file to be read
  - LPVOID buffer—a buffer where the data will be read into and written from
  - DWORD bytesToRead—the number of bytes to be read into the buffer
  - LPDWORD bytesRead—the number of bytes read during the last read
  - LPOVERLAPPED ovl—indicates if overlapped I/O is being used

86

86

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 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)

87

87

Types of System Calls

- Process control
- File management
- Device management
- Information maintenance
- Communications

88

88

System Programs

- System programs provide a convenient environment for program development and execution.
- They can be divided into:
  - File manipulation
  - Status information
  - File modification
  - Programming language support
  - Program loading and execution
  - Communications
  - Application programs
- Most users' view of the operation system is defined by system programs, not the actual system calls

89

89

System Programs

- Provide a convenient environment for program development and execution
  - Some of them are simply user interfaces to system calls; others are considerably more complex
- 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

90

90

System Programs (cont'd)

- 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

91

91

UNIX

- Unix programmers model every high-level device as a file, because they believed the purpose of computation was data transformation.
  - E.g. printers were represented as a "file" at a known location - when data was copied to the file, it printed out.
- In Unix, the OS consists of two parts;
  - (1) the huge collection of utility programs that drive most operations,
  - (2) kernel that runs the programs.
  - the kernel is a program running in supervisor mode that acts as a program loader and supervisor for the small utility programs making up the rest of the system, and to provide locking and I/O services for these programs; beyond that, the kernel didn't intervene at all in user space.
- Over the years the computing model changed, and Unix's treatment of everything as a file or byte stream no longer was as universally applicable:
  - Although a terminal could be treated as a file or a byte stream, which is printed to or read from, the same did not seem to be true for a GUI.
  - Networking - Even if network communication can be compared to file access, the low-level packet-oriented architecture dealt with discrete chunks of data and not with whole files.
  - As the capability of computers grew, Unix became increasingly cluttered with code. While kernels might have had 100,000 lines of code in the 70s and 80s, kernels of modern Unix successors like Linux have more than 4.5 million lines.

92

92

UNIX

- Modern Unix-derivatives are generally based on module-loading monolithic kernels. Examples of this are the Linux kernel in its many distributions as well as the Berkeley software distribution variant kernels such as FreeBSD, DragonflyBSD, OpenBSD and NetBSD.
- Apart from these alternatives, amateur developers maintain an active OS development community, populated by self-written hobby kernels which mostly end up sharing many features with Linux, FreeBSD, DragonflyBSD, OpenBSD or NetBSD kernels and/or being compatible with them.

93

93

Mac OS

- Apple Computer first launched Mac OS in 1984, bundled with its Apple Macintosh personal computer. For the first few releases, Mac OS (or System Software, as it was called) lacked many essential features, such as multitasking and a hierarchical filesystem. With time, the OS evolved and eventually became Mac OS 9 and had many new features added, but the kernel basically stayed the same.[*citation needed*] Against this, Mac OS X is based on Darwin, which uses a hybrid kernel called XNU, which was created combining the 4.3BSD kernel and the Mach kernel.

94

94

Microsoft Windows

- Microsoft Windows was first released in 1985 as an add-on to MS-DOS. Because of its dependence on another OS, initial releases of Windows, prior to Windows 95, were considered an operating environment (do not confuse with operating system).
- This product line continued to evolve through the 1980s and 1990s, culminating with release of the Windows 9x series (upgrading the system's capabilities to 32-bit addressing and pre-emptive multitasking) through the mid 1990s and ending with the release of Windows Me in 2000.
- Microsoft also developed Windows NT, an OS intended for high-end and business users. This line started with the release of Windows NT 3.1 in 1993, and has continued through the years of 2000 with Windows Vista and Windows Server 2008.

95

95

Microsoft Windows

- The release of Windows XP in October 2001 brought these two product lines together, with the intent of combining the stability of the NT kernel with consumer features from the 9x series. The architecture of Windows NT's kernel is considered a hybrid kernel because the kernel itself contains tasks such as the Window Manager and the IPC Manager, but several subsystems run in user mode. The precise breakdown of user mode and kernel mode components has changed from release to release, but with the introduction of the User Mode Driver Framework in Windows Vista, and user-mode thread scheduling in Windows 7, have brought more kernel-mode functionality into user-mode processes.

96

96