Chapter 1: Introduction



STUDENTS-HUB com Operating System Concepts – 10^h Edition

shiplaade, GRVn Malakino baid



- What Operating Systems Do
- Computer-System Organization
- Computer-System Architecture
- Operating-System Operations
- Resource Management
- Security and Protection
- Virtualization
- Distributed Systems
- Free/Libre and Open-Source Operating Systems





- Describe the general organization of a computer system and the role of interrupts
- Describe the components in a modern, multiprocessor computer system
- Illustrate the transition from user mode to kernel mode
- Discuss how operating systems are used in various computing environments
- Provide examples of free and open-source operating systems





- An operating system is "fill in the blanks"
- What about:
 - Car
 - Airplane
 - Printer
 - Washing Machine
 - Toaster
 - Compiler
 - Etc.





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





- 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



Abstract View of Components of Computer







What Operating Systems Do

- Depends on the point of view
- Users want convenience, ease of use and good performance
 - Don't care about resource utilization
- But shared computer such as mainframe or minicomputer must keep all users happy
 - Operating system is a resource allocator and control program making efficient use of HW and managing execution of user programs
- Users of dedicate systems such as workstations have dedicated resources but frequently use shared resources from servers
- Mobile devices like smartphones and tables are resource poor, optimized for usability and battery life
 - Mobile user interfaces such as touch screens, voice recognition
- Some computers have little or no user interface, such as embedded computers in devices and automobiles
 - Run primarily without user intervention



Operating System Definition

- No universally accepted definition
- "Everything a vendor ships when you order an operating system" is a good approximation
 - But varies wildly
- "The one program running at all times on the computer" is the kernel, part of the operating system
- Everything else is either
 - A system program (ships with the operating system, but not part of the kernel), or
 - An *application program*, all programs not associated with the operating system
- Today's OSes for general purpose and mobile computing also include middleware – a set of software frameworks that provide additional services to application developers such as databases, multimedia, graphics

shelsade Gavin Malakine Ba



Overview of Computer System Structure





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





- 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
- Each device controller type has an operating system device driver to manage it
- 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





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
- A trap or exception is a software-generated interrupt caused either by an error or a user request
- An operating system is **interrupt driven**





Interrupt Timeline



รหมอาจสมรุงสมที่ เมืองสมกัด



- The operating system preserves the state of the CPU by storing the registers and the program counter
- Determines which type of interrupt has occurred:
- Separate segments of code determine what action should be taken for each type of interrupt





Interrupt-drive I/O Cycle



STUDENTS-HUB.com Operating System Concepts – 10th Edition shiplaade, GRVn Malak no baid



I/O Structure

- Two methods for handling I/O
 - After I/O starts, control returns to user program only upon I/O completion
 - After I/O starts, control returns to user program without waiting for I/O completion





I/O Structure (Cont.)

- 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
- After I/O starts, control returns to user program without waiting for I/O completion
 - System call request to the OS 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
 - OS indexes into I/O device table to determine device status and to modify table entry to include interrupt





Storage Structure







Storage Structure

- Main memory only large storage media that the CPU can access directly
 - Random access
 - Typically volatile
 - Typically random-access memory in the form of Dynamic Random-access Memory (DRAM)
- Secondary storage extension of main memory that provides large nonvolatile storage capacity





Storage Structure (Cont.)

- Hard Disk Drives (HDD) 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
- Non-volatile memory (NVM) devices— faster than hard disks, nonvolatile
 - Various technologies
 - Becoming more popular as capacity and performance increases, price drops





- Storage systems organized in hierarchy
 - Speed
 - Cost
 - Volatility
- Caching copying information into faster storage system; main memory can be viewed as a cache for secondary storage
- Device Driver for each device controller to manage I/O
 - Provides uniform interface between controller and kernel





Storage-Device Hierarchy



How a Modern Computer Works



A von Neumann architecture

shiplaade, GRVn Malak no baid



Direct Memory Access Structure

- Used for high-speed I/O devices able to transmit information at close to memory speeds
- Device controller transfers blocks of data from buffer storage directly to main memory without CPU intervention
- Only one interrupt is generated per block, rather than the one interrupt per byte





Operating-System Operations

- Bootstrap program simple code to initialize the system, load the kernel
- Kernel loads
- Starts system daemons (services provided outside of the kernel)
- Kernel interrupt driven (hardware and software)
 - Hardware interrupt by one of the devices
 - Software interrupt (exception or trap):
 - Software error (e.g., division by zero)
 - Request for operating system service system call
 - Other process problems include infinite loop, processes modifying each other or the operating system





Multiprogramming (Batch system)

- Single user cannot always keep CPU and I/O devices busy
- 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 job has to wait (for I/O for example), OS switches to another job





Multitasking (Timesharing)

- A logical extension of Batch systems
 – the 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



Memory Layout for Multiprogrammed System

max	operating system
	process 1
	process 2
	process 3
0	process 4





Dual-mode Operation

- Dual-mode operation allows OS to protect itself and other system components
 - User mode and kernel mode
- Mode bit provided by hardware
 - Provides ability to distinguish when system is running user code or kernel code.
 - When a user is running ⇒ mode bit is "user"
 - When kernel code is executing ⇒ mode bit is "kernel"
- How do we guarantee that user does not explicitly set the mode bit to "kernel"?
 - System call changes mode to kernel, return from call resets it to user
- Some instructions designated as privileged, only executable in kernel mode



Transition from User to Kernel Mode







Timer

- Timer to prevent infinite loop (or process hogging resources)
 - Timer is set to interrupt the computer after some time period
 - Keep a counter that is decremented by the physical clock
 - Operating system set the counter (privileged instruction)
 - When counter zero generate an interrupt
 - Set up before scheduling process to regain control or terminate program that exceeds allotted time



Process Management

- A process is a program in execution. It is a unit of work within the system. Program is a *passive entity;* process is an *active entity*.
- Process needs resources to accomplish its task
 - CPU, memory, I/O, files
 - Initialization data
- Process termination requires reclaim of any reusable resources
- Single-threaded process has one program counter specifying location of next instruction to execute
 - Process executes instructions sequentially, one at a time, until completion
- Multi-threaded process has one program counter per thread
- Typically system has many processes, some user, some operating system running concurrently on one or more CPUs
 - Concurrency by multiplexing the CPUs among the processes / threads





Process Management Activities

The operating system is responsible for the following activities in connection with process management:

- Creating and deleting both user and system processes
- Suspending and resuming processes
- Providing mechanisms for process synchronization
- Providing mechanisms for process communication
- Providing mechanisms for deadlock handling





Memory Management

- To execute a program all (or part) of the instructions must be in memory
- All (or part) of the data that is needed by the program must be in memory
- Memory management determines what is in memory and when
 - Optimizing CPU utilization and computer response to users
- Memory management activities
 - Keeping track of which parts of memory are currently being used and by whom
 - Deciding which processes (or parts thereof) and data to move into and out of memory
 - Allocating and deallocating memory space as needed





- OS provides uniform, logical view of information storage
 - Abstracts physical properties to logical storage unit file
 - Each medium is controlled by device (i.e., disk drive, tape drive)
 - Varying properties include access speed, capacity, datatransfer rate, access method (sequential or random)
- File-System management
 - Files usually organized into directories
 - Access control on most systems to determine who can access what
 - OS activities include
 - Creating and deleting files and directories
 - Primitives to manipulate files and directories
 - Mapping files onto secondary storage
 - Backup files onto stable (non-volatile) storage media




Mass-Storage Management

- Usually disks used to store data that does not fit in main memory or data that must be kept for a "long" period of time
- Proper management is of central importance
- Entire speed of computer operation hinges on disk subsystem and its algorithms
- OS activities
 - Mounting and unmounting
 - Free-space management
 - Storage allocation
 - Disk scheduling
 - Partitioning
 - Protection





Caching

- 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



Characteristics of Various Types of Storage

Level	1	2	3	4	5
Name	registers	cache	main memory	solid-state disk	magnetic disk
Typical size	< 1 KB	< 16MB	< 64GB	< 1 TB	< 10 TB
Implementation technology	custom memory with multiple ports CMOS	on-chip or off-chip CMOS SRAM	CMOS SRAM	flash memory	magnetic disk
Access time (ns)	0.25-0.5	0.5-25	80-250	25,000-50,000	5,000,000
Bandwidth (MB/sec)	20,000-100,000	5,000-10,000	1,000-5,000	500	20-150
Managed by	compiler	hardware	operating system	operating system	operating system
Backed by	cache	main memory	disk	disk	disk or tape

Movement between levels of storage hierarchy can be explicit or implicit





Computer System Architecture







- Most systems use a single general-purpose processor
 - Most systems have special-purpose processors as well
- **Multiprocessors** systems growing in use and importance
 - Also known as parallel systems, tightly-coupled systems
 - Advantages include:
 - 1. Increased throughput
 - 2. Economy of scale
 - 3. Increased reliability graceful degradation or fault tolerance
 - Two types:
 - Asymmetric Multiprocessing each processor is assigned a specie task.
 - Symmetric Multiprocessing each processor performs all tasks

Symmetric Multiprocessing Architecture



shulelsaale, G. R.V.in Malakin Daaida





- Multi-chip and multicore
- Systems containing all chips
 - Chassis containing multiple separate systems



STUDENTS-HUB.com Operating System Concepts – 10th Edition shippisaale, Gavin Malakin Dezia

Non-Uniform Memory Access System



STUDENTS-HUB.com Operating System Concepts – 10th Edition รหมอาจสมรุงธริงาท เมลายุสุกอออาจี

Free and Open-Source Operating Systems

- Operating systems made available in source-code format rather than just binary closed-source and proprietary
- Counter to the copy protection and Digital Rights Management (DRM) movement
- Started by Free Software Foundation (FSF), which has "copyleft" GNU Public License (GPL)
 - Free software and open-source software are two different ideas championed by different groups of people
 - https://www.gnu.org/philosophy/open-source-misses-thepoint.en.html
- Examples include GNU/Linux and BSD UNIX (including core of Mac OS X), and many more
- Can use VMM like VMware Player (Free on Windows), Virtualbox (open source and free on many platforms - http://www.virtualbox.com)
 - Use to run guest operating systems for exploration



Chapter 2: Operating-System Services



STUDENTS-HUB com Operating System Concepts – 10^h Edition shiplaade, davin Malakino baid



Outline

- Operating System Services
- User and Operating System-Interface
- System Calls
- System Services
- Linkers and Loaders
- Why Applications are Operating System Specific
- Design and Implementation
- Operating System Structure
- Building and Booting an Operating System
- Operating System Debugging





- 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
- Illustrate the process for booting an operating system
- Apply tools for monitoring operating system performance
- Design and implement kernel modules for interacting with a Linux kernel



A View of Operating System Services





STUDENTS-HUB.com Operating System Concepts – 10th Edition



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), touch-screen, 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
 - 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.

shilersaale, Galvin Malaking beig

STUDENTS-HUB.com Operating System Concepts – 10th Edition

Operating System Services (Cont.)

- One set of operating-system services provides functions that are helpful to the user (Cont.):
 - **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)
 - 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
 - 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.
 - Logging To keep track of which users use how much and what kinds of computer resources
 - 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



- 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



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
- 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)





Bourne Shell Command Interpreter

● ● ● 1. root@r6181-d5-us01:~ (ssh)							
× root@r6181-d5-u • #1	I X	ssh 🦻	\$ #2 ×	root@r618	81-d5-u	s01 #3	
Last login: Thu Jul 14	08:47:01 or	ttys002					
iMacPro:~ pbg\$ ssh root	iMacPro:~ pbg\$ ssh root@r6181-d5-us01						
root@r6181-d5-us01's password:							
Last Login: Thu Jul 14	06:01:11 20	16 from 17	2.16.16.1	162			
LLCOT@r6181-d5-us01 ~]#	f uptime		in the state of the last	. 120 52	00.0		
06:57:48 up 16 adys, 1	10:52, 3 US	ers, Load	average	: 129.52	, 80.3	53, 56.55	
Eilosystom	F UT -KN	vail llco	Mounted	00			
/dev/manner/va ks_lv ra	ot Used A	vull use	Mourrea	on			
/uev/ilupper/vg_ks=tv_rc	506 196	286 41%	1				
tmpfs 1	27G 520K	1276 1%	/dev/shm				
/dev/sda1 4	77M 71M	381M 16%	/boot				
/dev/dssd0000 1	.0T 480G	545G 47%	/dssd_xfs	S			
tcp://192.168.150.1:333	34/orangefs						
	12T 5.7T	6.4T 47%	/mnt/ora	ngefs			
/dev/gpfs-test	/gpfs-test 23T 1.1T 22T 5% /mnt/gpfs						
[root@r6181-d5-us01 ~]#	ŧ						
[root@r6181-d5-us01 ~]# ps aux sort -nrk 3,3 head -n 5							
root 97653 11.2 6	6.6 42665344	17520636	? S <l1< td=""><td>Jul13 10</td><td>66:23</td><td>/usr/lpp/mmfs/bin/mmfsd</td><td></td></l1<>	Jul13 10	66:23	/usr/lpp/mmfs/bin/mmfsd	
root 69849 6.6 @	0.0	0 ?	S	Jul12 18	81:54	[vpthread-1-1]	
root 69850 6.4 0	0.0	0 ?	S	Jul12 1	77:42	[vpthread-1-2]	
root 3829 3.0 0	0.0 0	0 ?	S	Jun27 73	30:04	[rp_thread 7:0]	
root 3826 3.0 0	0.0 0	0 ?	S	Jun27 72	28:08	[rp_thread 6:0]	
[root@r6181-d5-us01 ~]# ls -l /usr/lpp/mmfs/bin/mmfsd							
-r-x 1 root root 2066/161 Jun 3 2015 /usr/lpp/mmts/bin/mmtsa							
Lrooter6181-d5-us01 ~J#							

รหมองสุรุธสี่มา เมื่อเสม



- System programs provide a convenient environment for program development and execution. They can be divided into:
 - File manipulation
 - Status information sometimes stored in a file
 - Programming language support
 - Program loading and execution
 - Communications
 - Background services
 - Application programs
- Most users' view of the operating system is defined by system programs, not the actual system calls.
- But what are **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), and Java API for the Java virtual machine (JVM)

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





Example of System Calls

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







Example of Standard API

EXAMPLE OF STANDARD API

As an example of a standard API, consider the read() function that is available in UNIX and Linux systems. The API for this function is obtained from the man page by invoking the command

man read

on the command line. A description of this API appears below:

#include	<unistd.h></unistd.h>					
ssize_t	read(int	fd,	void	*buf,	size_t	count)
return value	function name		р	aramete	ers	

A program that uses the read() function must include the unistd.h header file, as this file defines the ssize_t and size_t data types (among other things). The parameters passed to read() are as follows:

- int fd—the file descriptor to be read
- void *buf a buffer into which the data will be read
- size_t count—the maximum number of bytes to be read into the buffer

On a successful read, the number of bytes read is returned. A return value of 0 indicates end of file. If an error occurs, read() returns -1.

shippingade Gavin Malaking bai



System Call Implementation

- Typically, a number is 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)



API – System Call – OS Relationship





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







STUDENTS-HUB.com Operating System Concepts – 10th Edition



Types of System Calls

- Process control
 - create process, terminate process, end, abort, load, execute,...
- File management
 - create file, delete file, open, close file, read, write, reposition,...
- Device management
 - request device, release device, read, write, get device attributes, set device attributes, logically attach or detach devices, ...
- Information maintenance
 - get time or date, set time or date, get system data, set system data, ...
- Communications
 - create, delete communication connection, send, receive messages if message passing model to host name or process name, Sharedmemory model create and gain access to memory regions, ...
- Protection
 - Control access to resources, Get and set permissions, Allow and deny user access

sheredate Gawn Matakine att

STUDENTS-HUB.com Operating System Concepts – 10th Edition



EXAMPLES OF WINDOWS AND UNIX SYSTEM CALLS

The following illustrates various equivalent system calls for Windows and UNIX operating systems.

	Windows	Unix
Process control	CreateProcess() ExitProcess() WaitForSingleObject()	<pre>fork() exit() wait()</pre>
File management	CreateFile() ReadFile() WriteFile() CloseHandle()	open() read() write() close()
Device management	SetConsoleMode() ReadConsole() WriteConsole()	ioctl() read() write()
Information maintenance	GetCurrentProcessID() SetTimer() Sleep()	getpid() alarm() sleep()
Communications	CreatePipe() CreateFileMapping() MapViewOfFile()	pipe() shm_open() mmap()
Protection	SetFileSecurity() InitlializeSecurityDescriptor() SetSecurityDescriptorGroup()	chmod() umask() chown()

STUDENTS-HUB.com Operating System Concepts – 10th Edition รหมอายาสระ 6 สี่งา่า พลเลม เกอออาร์



Standard C Library Example

• C program invoking printf() library call, which calls write() system call

THE STANDARD C LIBRARY

The standard C library provides a portion of the system-call interface for many versions of UNIX and Linux. As an example, let's assume a C program invokes the printf() statement. The C library intercepts this call and invokes the necessary system call (or calls) in the operating system—in this instance, the write() system call. The C library takes the value returned by write() and passes it back to the user program:



STUDENTS-HUB.com Operating System Concepts – 10th Edition shippenale, GRVn Malakin Dezich

The Role of the Linker and Loader



STUDENTS-HUB.com Operating System Concepts – 10th Edition

Why Applications are Operating System Specific

- Apps compiled on one system usually not executable on other operating systems
- Each operating system provides its own unique system calls
 - Own file formats, etc.
- Apps can be multi-operating system
 - Written in interpreted language like Python, Ruby, and interpreter available on multiple operating systems
 - App written in language that includes a VM containing the running app (like Java)
 - Use standard language (like C), compile separately on each operating system to run on each
- Application Binary Interface (ABI) is architecture equivalent of API, defines how different components of binary code can interface for a given operating system on a given architecture, CPU, etc.





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

Monolithic Structure – Original UNIX

- UNIX limited by hardware functionality, the original UNIX operating system 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 the file system, CPU scheduling, memory management, and other operating-system functions; a large number of functions for one level





Beyond simple but not fully layered

	(the users)					
kernel	shells and commands compilers and interpreters system libraries					
	system-call interface to the kernel					
	signals terminal handling character I/O system terminal drivers	file system swapping block I/O system disk and tape drivers	CPU scheduling page replacement demand paging virtual memory			
	kernel interface to the hardware					
-	terminal controllers terminals	device controllers disks and tapes	memory controllers physical memory			



STUDENTS-HUB.com Operating System Concepts – 10th Edition



Monolithic plus modular design



STUDENTS-HUB.com Operating System Concepts – 10th Edition shiplaade, Gavin Malakin Dezig


Layered Approach

- The operating system is divided into a number of layers (levels), each built on top of lower layers. The bottom layer (layer 0), is the hardware; the highest (layer N) is the user interface.
- With modularity, layers are selected such that each uses functions (operations) and services of only lower-level layers







- Moves as much from the kernel into user space
- Mach is an 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
- Detriments:
 - Performance overhead of user space to kernel space communication





Microkernel System Structure







Modules

- Many modern operating systems implement loadable kernel modules (LKMs)
 - 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
 - Linux, Solaris, etc.



- Most modern operating systems are 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)



Building and Booting an Operating System

- Operating systems generally designed to run on a class of systems with variety of peripherals
- Commonly, operating system already installed on purchased computer
 - But can build and install some other operating systems
 - If generating an operating system from scratch
 - Write the operating system source code
 - Configure the operating system for the system on which it will run
 - Compile the operating system
 - Install the operating system
 - Boot the computer and its new operating system

Example: Building and Booting Linux

- Download Linux source code (http://www.kernel.org)
- Configure kernel via "make menuconfig"
- Compile the kernel using "make"
 - Produces vmlinuz, the kernel image
 - Compile kernel modules via "make modules"
 - Install kernel modules into vmlinuz via "make modules_install"
 - Install new kernel on the system via "make install"
 NTS-HUB.com
 System Concepts 10th Edition
 1.82





System Boot

- When power initialized on system, execution starts at a fixed memory location
- Operating system must be made available to hardware so hardware can start it
 - Small piece of code bootstrap loader, BIOS, stored in ROM or EEPROM locates the kernel, loads it into memory, and starts it
 - Sometimes two-step process where **boot block** at fixed location loaded by ROM code, which loads bootstrap loader from disk
 - Modern systems replace BIOS with Unified Extensible Firmware Interface (UEFI)
- Common bootstrap loader, GRUB, allows selection of kernel from multiple disks, versions, kernel options
- Kernel loads and system is then running
- Boot loaders frequently allow various boot states, such as single user mode