

Operating Systems: Lecture 2

1

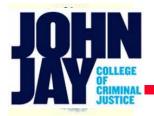
Operating System Structures

Jinwoo Kim

jwkim@jjay.cuny.edu



- Operating System Services
- User Operating System Interface
- System Calls
- Types of System Calls
- System Programs
- Operating System Design and Implementation
- Operating System Structure
- Virtual Machines
- Operating System Generation
- System Boot



- To describe the services an operating system provides to users, processes, and other systems
- To discuss the various ways of structuring an operating system
- To explain how operating systems are installed and customized and how they boot



- 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), 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. Obviously, programs need to read and write files and directories, create and delete them, search them, list file Information, permission management



- 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



- 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
 - 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



A View of Operating System Services

user and other system programs									
GUI batch command line									
user interfaces									
	system calls								
program execution	ons	ns file communication				1.1000755	resource allocation		
error detection			services				a	ection nd urity	
			(operating	, system				
hardware									



- 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



Bourne Shell Command Interpreter

000	Defau	ılt	
📮 🥡 💮 🦳			🗩 🎝 🗍
New Info Close		Execute	Bookmarks
Default	Default		
PBG-Mac-Pro:~ pbg\$ w			
15:24 up 56 mins, 2 users, lo	ad averages: 1.51 1.53	1.65	
USER TTY FROM	LOGIN@ IDLE WHAT	r	
pbg console -	14:34 50 -		
pbg s000 -	15:05 - w		
PBG-Mac-Pro:~ pbg\$ iostat 5			
disk0 disk		cpu load average	
		4B/s us sy id 1m 5m 15m	
33.75 343 11.30 64.31 1	4 0.88 39.67 0 0	0.02 11 5 84 1.51 1.53 1.65	
5.27 320 1.65 0.00	0 0.00 0.00 0 0	0.00 4 2 94 1.39 1.51 1.65	
4.28 329 1.37 0.00	0 0.00 0.00 0 0	0.00 5 3 92 1.44 1.51 1.65	
^C			
PBG-Mac-Pro:~ pbg\$ ls			
Applications	Music	WebEx	
Applications (Parallels)	Pando Packages	config.log	
Desktop	Pictures	getsmartdata.txt	
Documents	Public	imp	
Downloads	Sites	log	
Dropbox	Thumbs.db	panda-dist	
Library	Virtual Machines	prob.txt	
Movies	Volumes	scripts	
PBG-Mac-Pro:~ pbg\$ pwd			
/Users/pbg			
PBG-Mac-Pro:~ pbg\$ ping 192.16	8.1.1		
PING 192.168.1.1 (192.168.1.1)	: 56 data bytes		
64 bytes from 192.168.1.1: icm			
64 bytes from 192.168.1.1: icm ^C	p_seq=1 ttl=64 time=1.2	262 ms	
192.168.1.1 ping statistic	s		
2 packets transmitted, 2 packe		et loss	
round-trip min/avg/max/stddev			
PBG-Mac-Pro:~ pbg\$			



- 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 as "Aqua" GUI interface with UNIX kernel underneath and shells available
 - Unix and Linux have CLI with optional GUI interfaces (CDE, KDE, GNOME)



Touchscreen Interfaces

- Touchscreen devices require new interfaces
 - Mouse not possible or not desired
 - Actions and selection based on gestures
 - Virtual keyboard for text entry
 - Voice commands

📖 Verizon 🙃	10:31	1 🕴 42% 🔳
App Store	Settings	Contacts
Now The Contract of Contract o	Maps Reminders	Notes
HICALC + 2 × = Calculator Pro	Google Pocket	kindle
Wunderlist	Evernote	Reference
Apple2	Entertainment Music	Social
	Apple News W S Safari	Monday 9 Calendar Mail



The Mac OS X GUI

🍯 Grab File Edit Ca	pture Window Help			_ Ø • =	🖲 🗑 🖷 🖬 🕼 🖉 👘 🖉	📥 🌒 15:06 EDT Mon 2 Jul 🖉 🎕 💿
0.0.0		(⊯ fig-dir	- fater and		<u>a</u>	
		the state of the s			Nex .	a inclusion days
	1 🖾 🖉 🖉 🖗		0 % / 100 00 00	Selection) ID Preventer de	5500 (ALCO - 52 (ALCO
Dirop Stack +	🚔 • Faverites • Documents •	Music - Movies - Pictures - Dasktop -	Applications * ZPBG * ZPBGE * jDir	0.+	RECEIPTION OF THE PARTY OF THE	
		O Desktop O Computer O Ha-d	ir.			
Empire	▲ 2PBG > Imp > book > es8-dir					Alsonitoriate
	Name big-1.05a.pd	# Kind © FDF	Date Modified 6/24/07, 1:05PM	Size Application 106 LKB Skm		up a servanu
🖲 Network	i fig-1.0Ba svg	O SVG	5/18/07, 5:50PM	7.7 KB Inkscape		
Freedom Stripe	🔮 fig-2.0 png	Fortable Network	Craphics Image Today, 1:06PM	39.2 KB Freview		
Macintosh HD	€ hg-2.0.svg	O SVG	Today, 1:06PM	L4 3 KB Inkscape	6.4	000063
Untitled Untitled 2	👻 6.3-2.0 s	O THE Occurrent	Today, 2.53PM	8.80 1.KB Freddw		16 - C
e ZPBG ±					199	
ZPBGE					TIFF	Unitide 9
🖞 iDisk 🗴						tative of the data
* Peter Baer Galvin's iPod 🛛 🛔	é l					
						0
						ULEL
					(195719)	SPECIAL GRANTING
					Prezier His	
State	-				Info.+	0
r pbg	ő				Name: fig-2.0a fig-2.0a tiff	Rate Building
Applications	i l				Kind: TIFF Document	VEHICL SHEET FLO
Decuments	i l				UTH sublicitiff URL: dfs://	
Games	i l				CALVENT: PEC/OBLACKS, N	4886
I Utilities I tmp	i l				/ZPEG Size: 385.1 K8 (901,236 bytes)	Security of the second
iii Desktop	i l				data: 901.236 bytes Physical: 681 KE (902.144	1
Favorites	i l				aytes) Modified: Today, 2:53PM	19
Music	10				Modified: Today, 2:53PM Attributes: Today, 2:53PM	7P3G1
Movies		File History +		ecoor Path +	Owner: abg (101)	19737 (E) 30 (E) fa
Pictures	😤 fig-2.0a		🔮 fig-2.Da		Crowp: admin (50) Permission: -rwx (700)	
I Sites	1		💭 fig-dir 💭 os9-dir		Path: /Volumes/2P3G-1/imp/	
# Public	/1		teok		posk/os8-cir, fig-dir/ fig-2,0a.tiff	Journier Brep Box
Preferences Elbrary	1		imp		Application: Preview Volume: ZPBG	
il cti	1		E ZPEG		Capacity: 735.4 CB	N N
projects	L.				Free: 734.3 CB Format: SMB	Nevada bigi
🖉 consult	á				Mount Point: /Volumes/2P8G-1	TOTAL CONTRACTOR
1	B B D G Sitems	1 of 5 Items selected - 734.	3 G3 available: 5.1 G3 uses	- 1	/	
						Solaris (Qualitas
		000	Address Book			
		10 C (.	QA		O O Dictionary and Thesau	
		Construction of the second sec		4	A A Q oper	ating system 💿 📜 🔛
		Group Name	Annula Comm	1		Professional Mac
000 - 20	alc 🔿	Apple Comput		iter inc.	Dictionary —	and the second se
10	are	Directories MaApple Comput	er In		op-er-at-ing sys-tem	a)
				1.00		G-Force 3.7.1 (Platinum).dmg
	8	The Last Import				
Constant State	8.	Last Import			noun the software that supports a computer	(Platinum).dmg
	U.	미 Las: Import			the software that supports a computer	s base hinchons.
Dec Hex Oct Bin Deg R	U.	Las: Import	main 1-800-MY-APPLE		the software that supports a computer such as scheduling tasks, executing	s base hinchons.
	Rad () RPN Irfo Tape	Las: Import	1		the software that supports a computer	s base hinchons.
x ² x ³ x ⁿ x ¹ Rr	Rad () RPN Irio Tape nd x-y z meells ¹²	The Last Import	main 1-800-MY-APPLE other 800-275-2273		the software that supports a computer such as scheduling tasks, executing	s base hinchons.
	Rad () RPN Irio Tape nd x-y z meells ¹²	C Last Import	1	17	the software that supports a computer such as scheduling tasks, executing	s base hinchons.
x ² x ³ x ⁶ x ¹ Rr √x √x x^{1/2} 1/x 3	Rad () RPN Info Taps and $x - y$ z $prod_{E}^{(2)}$	C Las: Import	other 800-275-2273	275	the software that supports a computer such as scheduling tasks, executing	s base hinchons.
$\begin{array}{c} x^{2} x^{3} x^{6} x^{1} x^{1} Rr \\ \sqrt{x} \sqrt{x} \sqrt{x} x^{1/2} 1/x 3 \\ \hline 7 8 9 + A \end{array}$	kal () RPN Ir for Tape nd x-y z μμε/με ⁽²⁾ % x+M π μμε/με ⁽²⁾ %C M- Sim Cos Ta ²	C Las: Import	other 800-275-2273 home page http://www.appie.cr work 1 Infinite Loop		the software that supports a computer such as scheduling tasks, executing	s base hinchons.
x ² x ³ x ⁶ x ¹ Rr √x √x x^{1/2} 1/x 3	kal () RPN Ir for Tape nd x-y z μμε/με ⁽²⁾ % x+M π μμε/με ⁽²⁾ %C M- Sim Cos Ta ²	C Las: Import	other 800-275-2273 home page http://www.app.e.cr work 1 Infinite Loop C.pert no CA 9501-		the software that supports a computer such as scheduling tasks, executing	s base hinchons.
$\begin{array}{c} x^{2} x^{3} x^{6} x^{1} x^{1} Rr \\ \sqrt{x} \sqrt{x} \sqrt{x} x^{1/2} 1/x 3 \\ \hline 7 8 9 + A \end{array}$	kad () RPN frio Taps nd x-y z jiiiii jiiiii jiiiii x x-W π jiiiii jiiiii jiiiii jiiiii jiiiii jiiiii jiiiiiiii jiiiiii jiiiiiiiiii jiiiiiiiiiii jiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiii	Las: Import	other 800-275-2273 home page http://www.appie.cr work 1 Infinite Loop		the software that supports a computer such as scheduling tasks, executing	's basic hinchons.
x ² x ³ x ⁶ x1 Rr y x y y x y	Lat () RPA Info Tape ad x-y, a y x-y a () X-10 Tape () X-10 Tape () X-5 in Cos Ta- () M-5 in Cos 1 Tan () M-5 in Cos 1 Tan () M-5 in Cos 1 Tan	Las: Import	other 800-275-2273 home page http://www.app.e.cr work 1 Infinite Loop C.pert no CA 9501-	4	the software that supports a computer such as scheduling tasks, executing	s base hinchons.



- 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

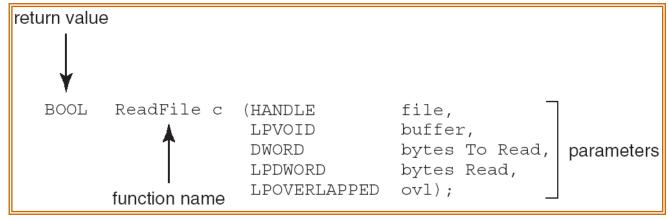


• System call sequence to copy the contents of one file to another file

source file	►	destination file
	Example System Call Sequence Acquire input file name Write prompt to screen Accept input Acquire output file name Write prompt to screen Accept input Open the input file if file doesn't exist, abort Create output file if file exists, abort Loop Read from input file Write to output file Until read fails Close output file Write completion message to screen Terminate normally	



- 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



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, siz	e_t count)
return value	function name	parameters	

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 where the data will be read into
- size_t count—the maximum number of bytes to be read into the buffer

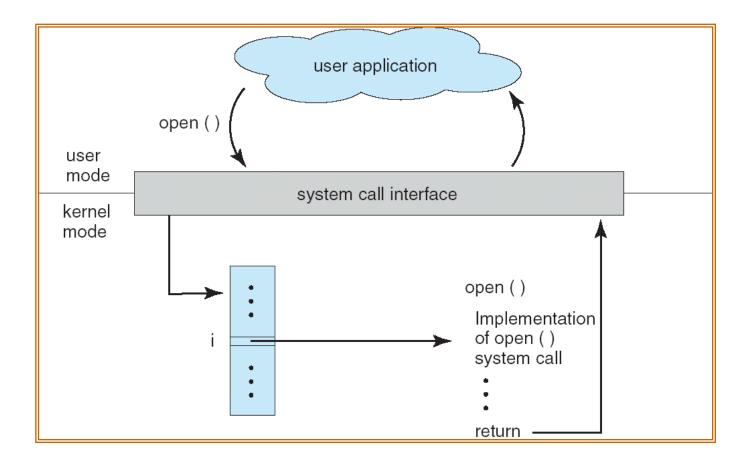
On a successful read, the number of bytes read is returned. A return value of 0 indicates end of file. If an error occurs, read() returns -1.



- 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 to 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

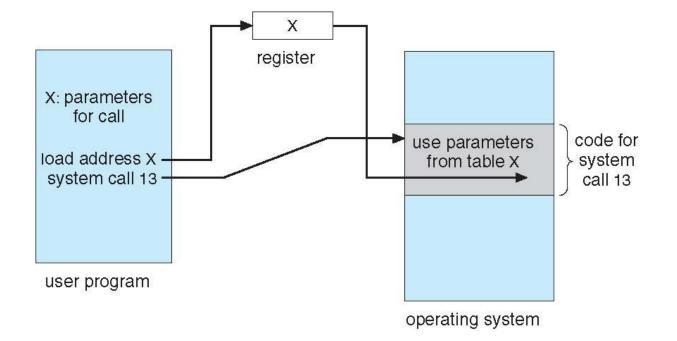




- Often, more information is required than simply identity of desired system call
 - Exact type and amount of information vary according to OS and call
- Three general methods used to pass parameters to the OS
 - Simplest: pass the parameters in registers
 - In some cases, may be more parameters than 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



Parameter Passing via Table





Types of System Calls

- Process control
 - create process, terminate process
 - end, abort
 - load, execute
 - get process attributes, set process attributes
 - wait for time
 - wait event, signal event
 - allocate and free memory
 - Dump memory if error
 - **Debugger** for determining **bugs**, **single step** execution
 - Locks for managing access to shared data between processes



- File management
 - create file, delete file
 - open, close file
 - read, write, reposition
 - get and set file attributes
- Device management
 - request device, release device
 - read, write, reposition
 - 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
 - get and set process, file, or device attributes
- Communications
 - create, delete communication connection
 - send, receive messages if message passing model to host name or process name
 - From client to server
 - Shared-memory model create and gain access to memory regions
 - transfer status information
 - attach and detach remote devices



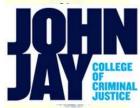
Types of System Calls (Cont.)

- Protection
 - Control access to resources
 - Get and set permissions
 - Allow and deny user access

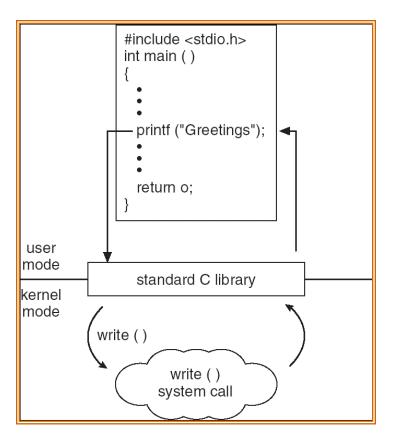


Examples of Windows and Unix System Calls

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

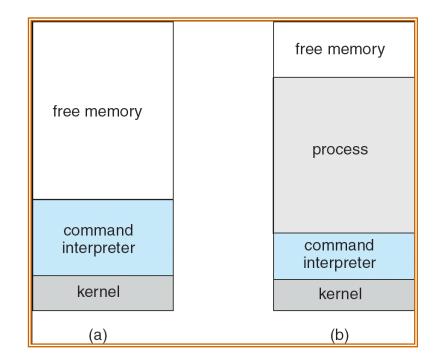


 C program invoking printf() library call, which calls write() system call





- Single-tasking
- Shell invoked when system booted
- Simple method to run program
 - No process created
- Single memory space
- Loads program into memory, overwriting all but the kernel
- Program exit -> shell reloaded



(a) At system startup (b) running a program



FreeBSD Running Multiple Programs

- Unix variant
- Multitasking
- User login -> invoke user's choice of shell
- Shell executes fork() system call to create process
 - Executes exec() to load program into process
 - Shell waits for process to terminate or continues with user commands
- Process exits with:
 - code = 0 -no error
 - code > 0 error code

process D
free memory
process C
interpreter
process B
kernel



- 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 operating system is defined by system programs, not the actual system calls



- 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



- 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



Background Services

- Launch at boot time
 - Some for system startup, then terminate
 - Some from system boot to shutdown
- Provide facilities like disk checking, process scheduling, error logging, printing
- Run in user context not kernel context
- Known as services, subsystems, daemons

• Application programs

- Don't pertain to system
- Run by users
- Not typically considered part of OS
- Launched by command line, mouse click, finger poke



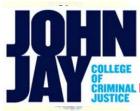
- Design and Implementation of OS not "solvable", but some approaches have proven successful
- Internal structure of different Operating Systems can vary widely
- Start by defining goals and specifications
- Affected by choice of hardware, type of system
- User goals and System goals
 - User goals operating 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



- Important principle to separate
 Policy: What will be done?
 Mechanism: How to do it?
- 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 maximum flexibility if policy decisions are to be changed later
- Specifying and designing an OS is highly creative task of software engineering



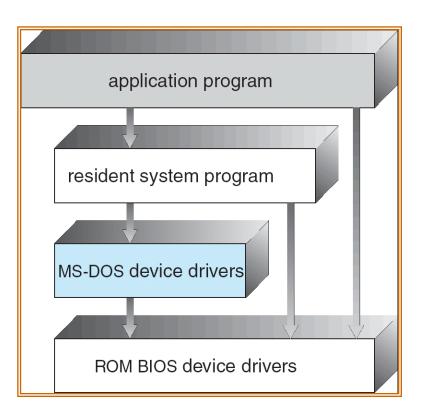
- Much variation
 - Early OSes in assembly language
 - Then system programming languages like Algol, PL/1
 - Now C, C++
- Usually a mix of languages
 - Lowest levels in assembly
 - Main body in C
 - Systems programs in C, C++, scripting languages like PERL, Python, shell scripts
- More high-level language easier to **port** to other hardware
 - But slower
- Emulation can allow an OS to run on non-native hardware



- General-purpose OS is very large program
- Various ways to structure ones
 - Simple structure MS-DOS
 - More complex -- UNIX
 - Layered an abstraction
 - Microkernel -Mach

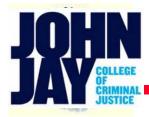


- MS-DOS written to provide the most functionality in the least space
 - Not divided into modules
 - Although MS-DOS has some structure, its interfaces and levels of functionality are not well separated





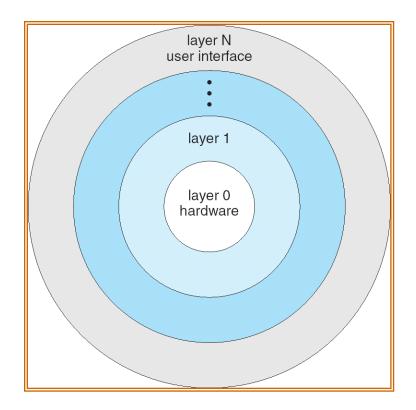
- UNIX limited by hardware functionality, the original UNIX operating system had limited structuring
- The UNIX OS consists of two separable parts
 - Systems programs
 - The kernel
 - Consists of everything below the system-call interface and above the physical hardware
 - Provides the file system, CPU scheduling, memory management, and other operating-system functions; many functions for one level



Beyond simple but not fully layered

	(the users)			
	shells and commands compilers and interpreters system libraries			
	system-call interface to the kernel			
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	

- The operating system is divided into several 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

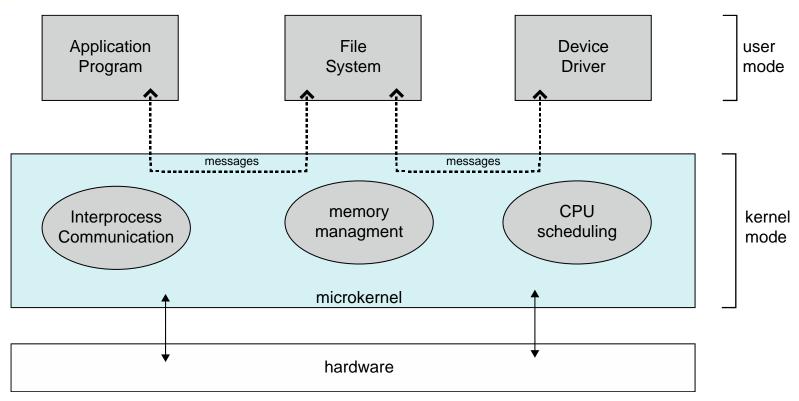


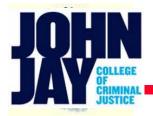


- Moves as much from the kernel into user space
- Mach example of microkernel
 - Mac OS X kernel (Darwin) partly based on Mach
- Communication takes place between user modules using message passing
- Benefits:
 - Easier to extend a microkernel
 - Easier to port the operating system to new architectures
 - More reliable (less code is running in kernel mode)
 - More secure
- Detriments:
 - Performance overhead of user space to kernel space communication



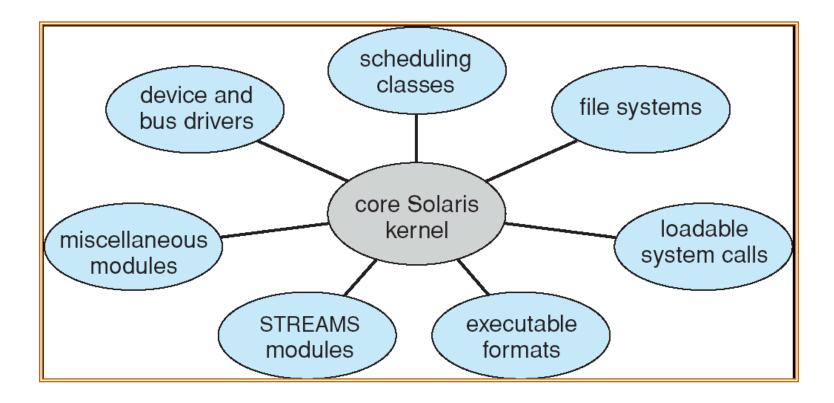
Microkernel System Structure





- Many modern operating systems implement loadable 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 approach* 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)



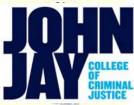
Mac OS X Structure

graphical user interface Aqua					
application environments and serv	application environments and services				
Java Cocoa		Quicktime	BSD		
kernel environment					
	BSD				
Mach					
I/O kit		kernel extensions			



- 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

Cocoa Touch		
Media Services		
Core Services		
Core OS		

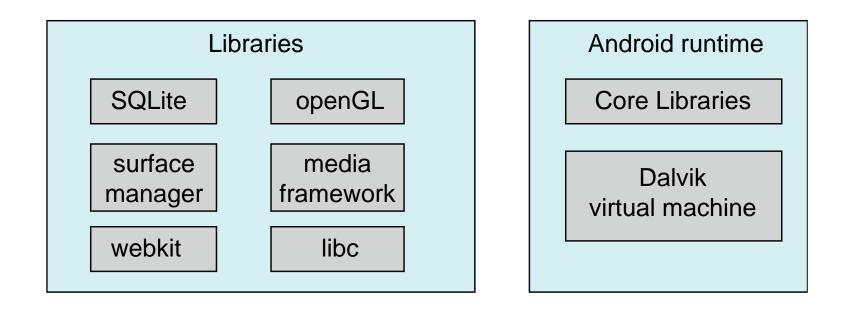


Android

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



Application Framework





- **Debugging** is finding and fixing errors, or **bugs**
- OS 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

Kernighan's Law: "Debugging is twice as hard as writing the code in the first place. Therefore, if you write the code as cleverly as possible, you are, by definition, not smart enough to debug it."



- Improve performance by removing bottlenecks
- OS must provide means of computing and displaying measures of system behavior
- For example, "top" program or Windows Task Manager

🗏 Windows Task Manager 📃 🗆 🗙				
<u>File O</u> ptions <u>V</u> i	ew <u>H</u> elp			
Applications Pr	ocesses Performanc	e Networking		
CPU Usage CPU Usage H		fistory		
0%				
PF Usage	Page File Usa	age History		
627 MB				
Totals		Physical Memory (0	
Handles	12621	Total	2096616	
Threads	563	Available	1391552	
Processes	50	System Cache	1584184	
Commit Charge (K)		Kernel Memory (K)		
Total	642128	Total	118724	
Limit	4036760	Paged	85636	
Peak	801216	Nonpaged	33088	
rocesses: 50	CPU Usage: 0%	Commit Charge:	627M / 3942M	



DTrace

- DTrace tool in Solaris, FreeBSD, Mac OS X allows live instrumentation on production systems
- Probes fire when code is executed within a provider, capturing state data and sending it to consumers of those probes
- Example of following XEventsQueued system call move from libc library to kernel and back

```
dtrace: script './all.d' matched 52377 probes
CPU FUNCTION
  0 -> XEventsQueued
                                           U
      -> XEventsQueued
  0
                                           U
        -> X11TransBytesReadable
  0
                                           U
        <- X11TransBytesReadable
                                           U
  0
           X11TransSocketBytesReadable U
  0
         ->
        <- X11TransSocketBytesreadable U
  0
        -> ioctl
  0
                                           U
  0
           -> ioctl
                                           Κ
  0
             -> getf
                                           Κ
  0
               -> set active fd
                                           Κ
               <- set active fd
  0
                                           Κ
  0
             <- getf
                                           Κ
  0
             -> get udatamodel
                                           Κ
             <- get udatamodel
  0
                                           Κ
 . .
             -> releasef
  0
                                           Κ
               -> clear active fd
  0
                                           Κ
               <- clear active fd
  0
                                           Κ
               -> cv broadcast
                                           Κ
  0
               <- cv broadcast
  0
                                           Κ
             <- releasef
  0
                                           Κ
           <- ioctl
  0
                                           Κ
  0
        <- ioctl
                                           U
      <- XEventsQueued
                                           U
  0 <- XEventsQueued
                                           U
```

./all.d 'pgrep xclock' XEventsQueued



• DTrace code to record amount of time each process with UserID 101 is in running mode (on CPU) in nanoseconds

```
sched:::on-cpu
uid == 101
{
   self->ts = timestamp;
}
```

```
sched:::off-cpu
self->ts
{
    @time[execname] = sum(timestamp - self->ts);
    self->ts = 0;
}
```

# dtrace -s sched.d			
dtrace: script 'sched.d' matched 6 probes			
^C	•		
gnome-settings-d	142354		
gnome-vfs-daemon	158243		
dsdm	189804		
wnck-applet	200030		
gnome-panel	277864		
clock-applet	374916		
mapping-daemon	385475		
xscreensaver	514177		
metacity	539281		
Xorg	2579646		
gnome-terminal	5007269		
mixer_applet2	7388447		
java	10769137		
/			

Figure 2.21 Output of the D code.



Operating systems are designed to run on any of a class of machines; the system must be configured for each specific computer site

SYSGEN program obtains information concerning the specific configuration of the hardware system Used to build system-specific compiled kernel or system-tuned Can general more efficient code than one general kernel

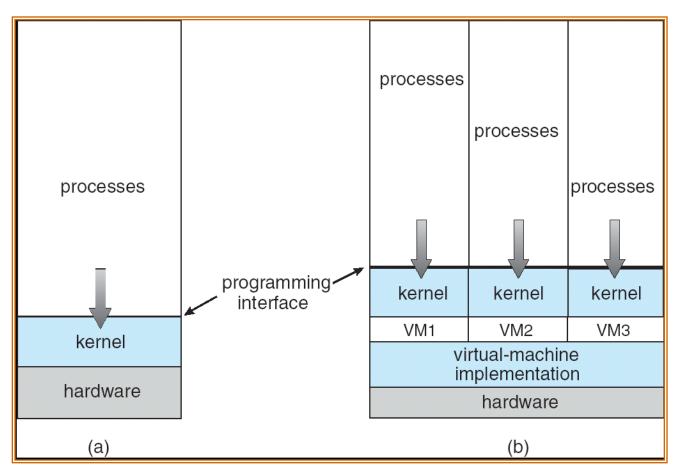


- A *virtual machine* takes the layered approach to its logical conclusion. It treats hardware and the operating system kernel as though they were all hardware
- A virtual machine provides an interface *identical* to the underlying bare hardware
- The operating system creates the illusion of multiple processes, each executing on its own processor with its own (virtual) memory



- 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
 - 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



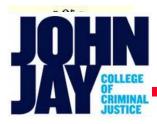


(a) Nonvirtual machine

(b) virtual machine

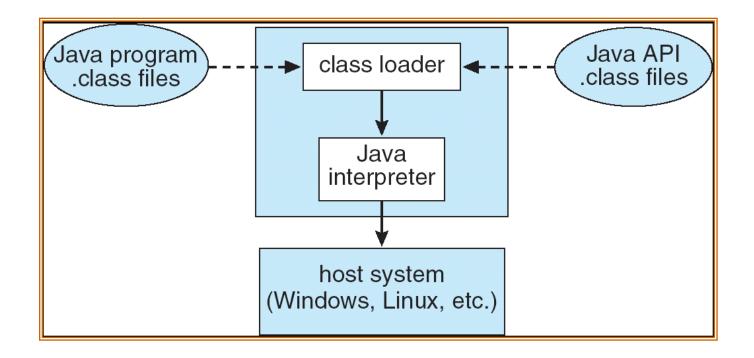


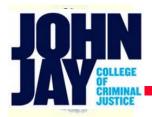
- The virtual-machine concept provides complete protection of system resources since each virtual machine is isolated from all other virtual machines
 - This isolation, however, permits no direct sharing of resources
- A virtual-machine system is a perfect vehicle for operatingsystems research and development.
 - 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



application	application	application	application	
	guest operating system (free BSD) virtual CPU virtual memory virtual devices	guest operating system (Windows NT) virtual CPU virtual memory virtual devices virtualization layer	guest operating system (Windows XP) virtual CPU virtual memory virtual devices	
host operating system (Linux)				
hardware CPU memory I/O devices				







- 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



- Each time a computer is turned on, it must familiarize itself with its internal components and the peripheral world
 - This start-up process is called the "boot process"
- The basic steps of "Boot Process"
 - CPU reset and run the bootstrap code in ROM BIOS (Basic Input Output System)
 - BIOS: firmware containing code for Input/output between OS and hardware
 - Stores inside ROM to maintain code without power
 - Pre-POST (Power-on Self Test)
 - POST
 - Disk boot
 - MBR
 - VBR



- POST results are compared with CMOS chip
- CMOS chip stores information about the computer drives, keyboard, monitor, current date and time
- Problem may result in beeps (example code on next slide), or error messages or simply fail to boot up
- If BIOS POST completes, BIOS instructs the CPU to look for a disk containing an Operating System



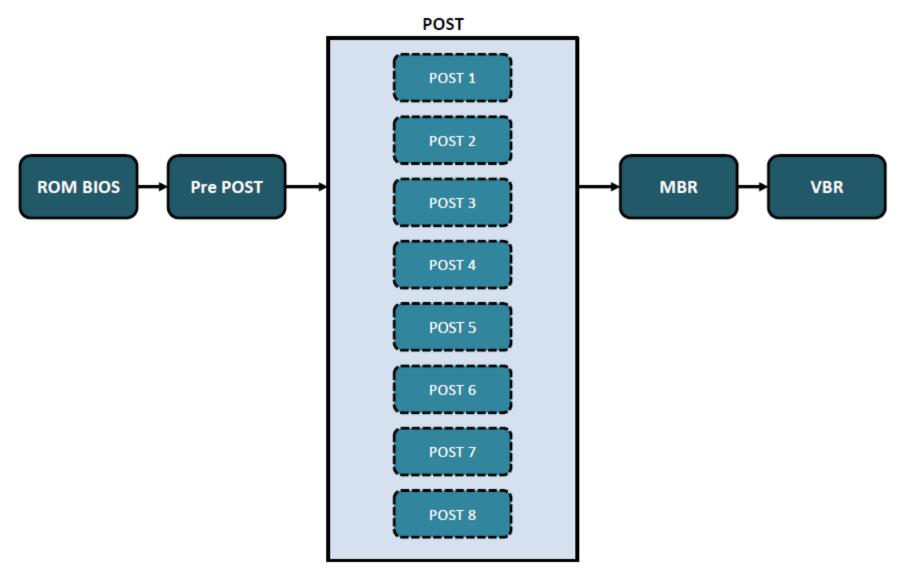
- BIOS vendors used a sequence of beeps from the motherboard-attached loudspeaker to signal error codes
 - Original IBM POST beep codes
 - 1 short beep Normal POST system is OK
 - 2 short beeps POST error error code shown on screen
 - · No beep Power supply, system board problem, disconnected CPU, or disconnected speaker,
 - · Continuous beep Power supply, system board, or keyboard problem
 - · Repeating short beeps Power supply or system board problem or keyboard
 - 1 long, 1 short beep System board problem
 - 1 long, 2 short beeps Display adapter problem (MDA, CGA)
 - 1 long, 3 short beeps Enhanced Graphics Adapter (EGA)
 - 3 long beeps 3270 keyboard card

Intel-based Macs

- 1 beep = no RAM installed
- 2 beeps = incompatible RAM types
- 3 beeps = no good banks
- 4 beeps = no good boot images in the boot ROM (and/or bad sys config block)
- 5 beeps = processor is not usable



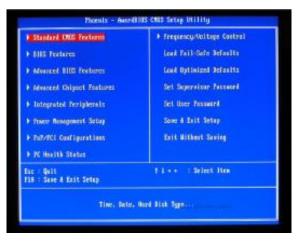
Common Boot Process





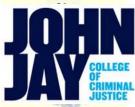
- ROM BIOS stage: the first stage in "Boot Process" is to get the CPU started (reset) with an electrical pulse
 - By power on switch button or over network
 - Once CPU is reset, PC is initialized with 0xF000
 - Address of bootstrap program in the ROM BIOS (Basic Input and Output System)
 - Bootstrap program in ROM BIOS chips that contain computer startup instructions starts







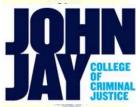
- BIOS does a series of tests to test computer hardware to be sure it is connected and operating correctly
 - Pre-POST: basic test for POST
- Pre-POST (Power on Self Test) freeze is indicative of some sort of hardware failure
- If test results matches with stored value in ROM BIOS, continue to POST



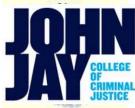
- 1st Step: System bus test
 - Power-On Self Test (POST)
 - Send special signal to the system buses to ensure that the bus is properly functioning
 - If it passes, POST continues to the next step
- 2nd Step: Real-Time Clock (RTC) or System clock test
 - Check system clock
 - Stores system date and time and also keeps all system electrical signals in synchronization
 - Inside the CMOS chip as a form of RTC/NVRAM
 - NVRAM (Non-Volatile RAM): stores basic system information for booting such as size of memory, drive type, etc







- 3rd Step: System's Video Components test
 - The video memory is tested, as are the signals sent by this device
 - If it passes, POST continues to the next step
- 4th Step: Main Memory (RAM) test
 - The data is written to RAM
 - The data is read and compared to the original data sent
 - If it matches, it passes and go to the next step
- 5th Step: Keyboard test
 - Check whether the keyboard is properly attached and whether any keys are pressed
 - If it passes, go to next step



The Boot Process – POST (Continued)

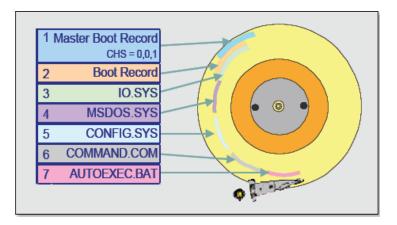
- 6th Step: Drive test
 - Sends signals over specific bus pathways to determine which drives (floppies, CD & DVDs, hard disk drives) are available to the system
 - If it passes, go to next step
- 7th Step: Check the POST result
 - POST results are compared to the expected system settings store in CMOS
 - If it passes, go to next step



- 8th Step: Additional BIOS loading
 - Load additional BIOS (SCSI BIOS, etc.) to RAM if necessary



- If BIOS POST completes, the bootstrap code in ROM BIOS instructs the CPU to look for a disk containing an Operating System according to the order set forth in the boot sequence
 - The place where this information is stored is called the "master boot record" (*MBR*)
 - Also referred to as the "master boot sector" or "boot sector"
- The MBR is always located as cylinder 0, head 0, and sector 1

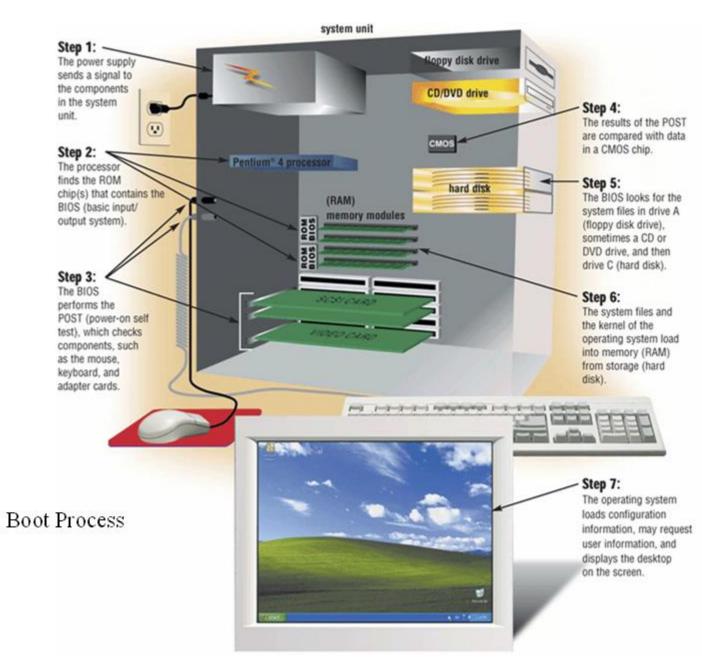




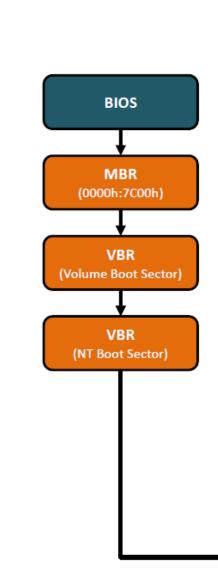
The Boot Process -- Disk Boot (Continued)

- The MBR contains following structure
 - Master Partition Table
 - This small table contains the description of the partitions that are contained on the hard disk
 - There is only room for the information describing 4 (primary) partitions
 - Master Boot Code
 - This small initial boot program loaded and executed to start the boot process by BIOS
 - Since the master boot code is the first program executed when you turn on your PC, this is a favorite place for virus writers to target
- Jump to the Volume Boor Record (VBR) of bootable partition
 - VBR code searches for and runs the OS on that volume



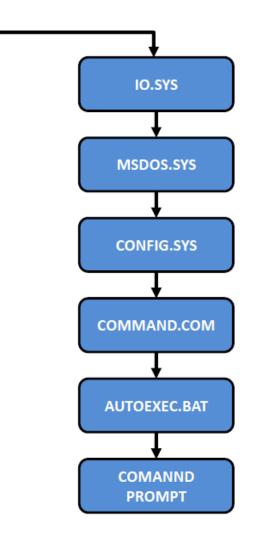






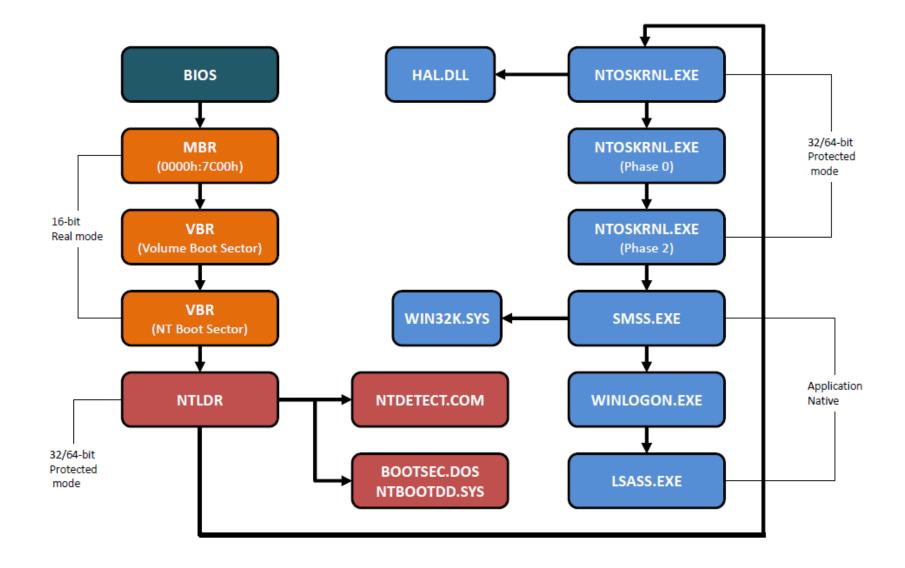
OH

COLLEGE OF CRIMINAL



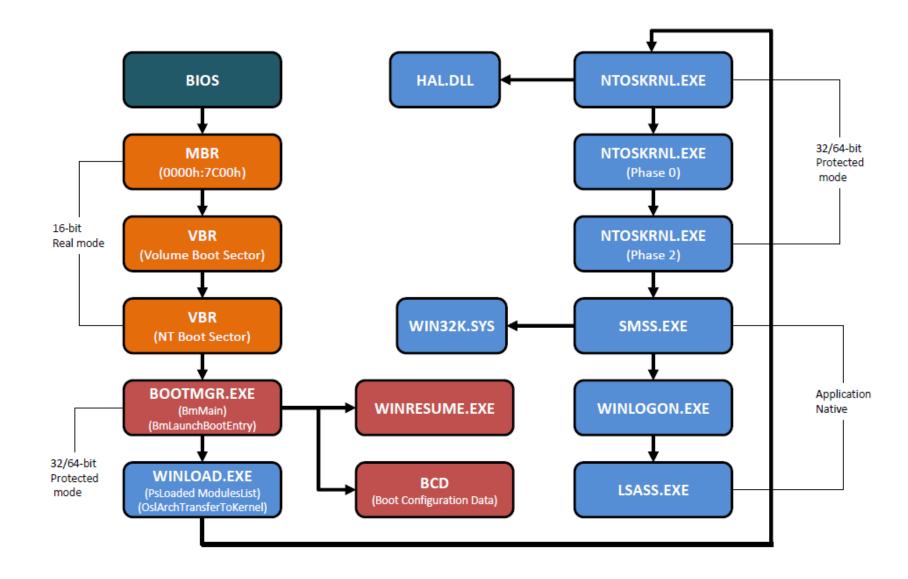


Windows NT/2000/XP Boot Process





Windows Vista/7 Boot Process





Windows 10 Boot sequence flowchart

				achiec nowenart
		Boot Sequence	Display	Typical NoBoot
BIOS phase	Pre Boot Windows	PC Power ON UEFI Boot POST (Power-on Self Tex) Launch UEFI Firmware Get boot informistion from SRAM (NVRAM or CMOS) -Boot entry -Boot order etc Bootstrap Code of MBR search Parition Tableto find active particin, then run Bootstrap Code of PBR (Partition Boot Record). Search and run bootmgr.		In this phase typically following reasons cause noboot issue - MBR corruption - PBR corruption - PBR corruption - Bootsector corruption - Bootsector corruption - Diak corruption (BIOS) Missing Boot Device Not failure. Reboot and Select proper Boot device or fuscer: Boot Hedia in selected Boot device [BIOS] Missing Bootmgr or corrupt Active Partition In operating system wasn't found. Try disconnecting any drives that don't press that an operating system. Press Ctrl Hitchel to restart -
ler phase	Windows Boot Manager	Launch Windows Boot Manager Histoxika Soot Manager Histoxika Soot Manager Visionade bootmar Read the BCD file Read the BCD file		In this phase, BCD, Registry or Driver files corruption might cause noboot. If there is no encrystion with disk, you probably can attach windbg to boomgr or bootloader for finding root cause. Bootmar Missing BCD An error occurred (ceeeed) while attempting to load the boot application Windows (system2)(winload.exe *** Fatal Error 0.00000001 : (0.0000002, 0x00000845, 0x0000000, 0x0000000)
Boot loader phase	Windows Boot Loader	Launch Windows Boot Loader WINDOWS\system32\winload.eti Load OS Ker nel into memory		The back interformation of the second of the
		Launch Windows NT OS Kernel	4	In this phase, various factors cause noboot''' Fortunately, from this phase 05 try to write down memory dump, so if you can get it you should analyze the dump first. If there is no dump, you should do typical troubleshooting. [NT OS Kemel] Missing registry hive Your PC ran into a problem and needs to restart. We're just collecting some error info, and then we'll restart for you.
ase	S Kernel	H/W enumistion Load drivers, create device node Launch smss.exe	You can see progress ring around here	O% complete Image: A start data data data data data data data
Kernel phase	Windows NT OS Kernel	Initialize Subaystem (load wind32k.sys) Create user session processes Launch Services Etc		
		Winlogon show logon screen Any Group Policy scripts run When the user bgs in, Windows creates a session for that user. (explorer, etc)	6:27 Saturday, November 10	



- Boot the computer and load an OS in a forensically sound manner so the evidentiary media is not changed
- When the disk contains evidence, the ability to prevent a computer from using the operating system on the hard disk is crucial
 - Ex) In Intel-based machine, a floppy diskette containing OS can be inserted to prevent the OS on the hard disk from loading
- Why is it important to have a controlled boot environment?