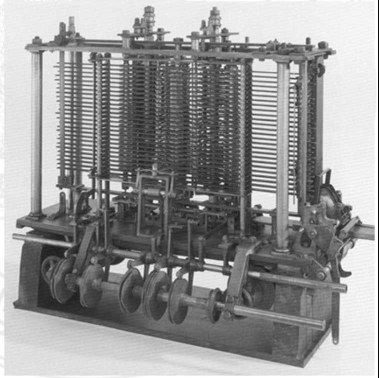
CIIC 4050 / ICOM 5007
Operating Systems

Lecture 2:
OS Structures

87

Pre-history

- Babbage's Analytical Engine
  - arithmetic logic unit
  - control flow
    - conditional branching
    - loops
  - integrated memory



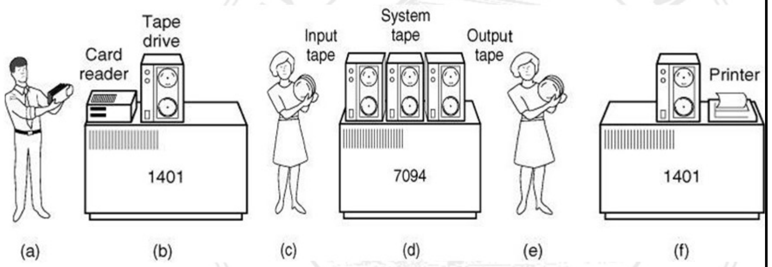
88

Operating System Generations

- Generation 1 (1945 – 55)
  Vacuum tubes and plugboards
- Generation 2 (1955 – 65)
  Transistors and batch systems
- Generation 3 (1965 – 80)
  ICs and multiprogramming
- Generation 4 (1980 – Present)
  Personal computers

89

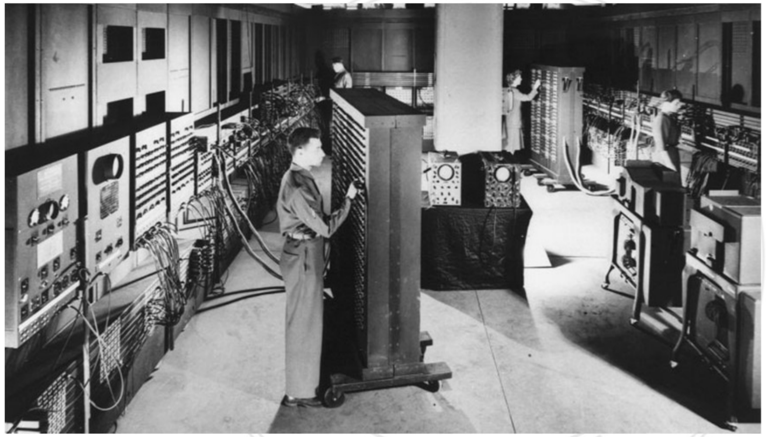
Early Batch Systems (1)



Early batch system. (a) Programmers bring cards to 1401. (b)1401 reads batch of jobs onto tape.

Based on Slides by: Tanenbaum & Woodhull, Operating Systems: Design and Implementation, (c) 2006 Prentice-Hall, Inc. All rights reserved. 0-13-142938-8

90

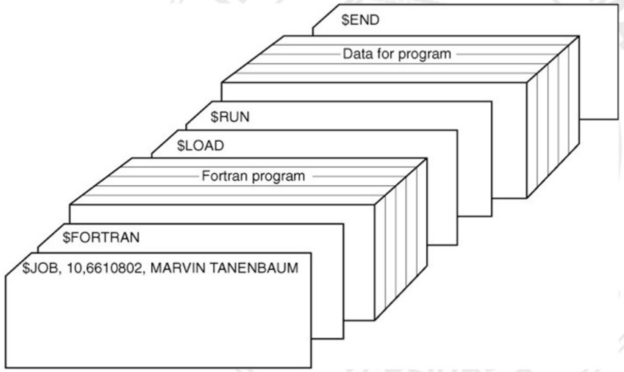


ENIAC, the world's first general-purpose electronic computer.

 91

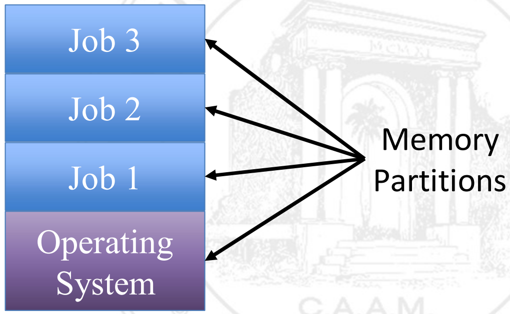
91

Early Batch Systems (2)



92

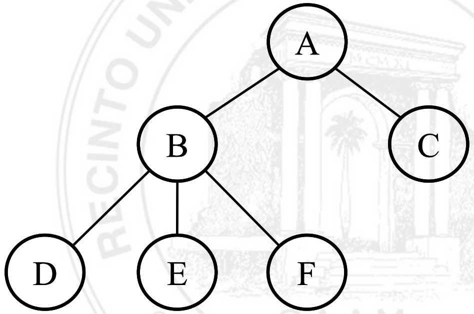
Multiprogramming



Multiprogramming system with three jobs in memory.

93

Processes (1)



- A process tree. Process A created two child processes, B and C. Process B created three child processes, D, E, and F.

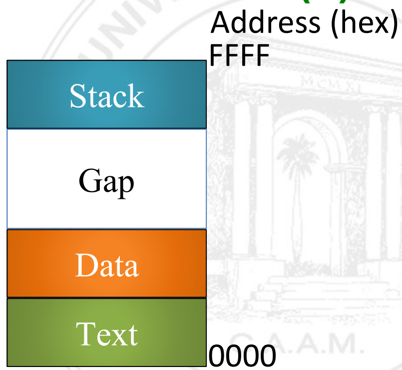
94

Processes (2)

- Two processes connected by a pipe.

95

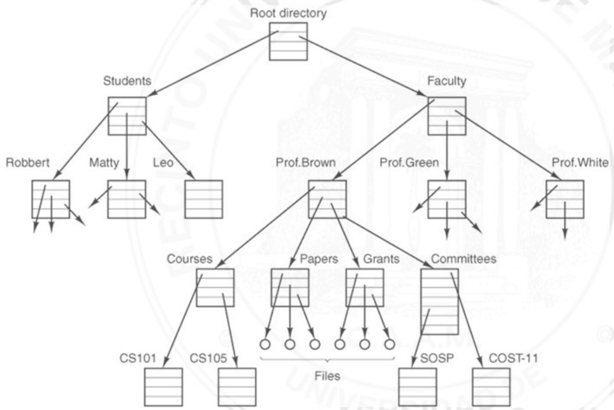
Processes (3)



- Processes have three segments: text, data, and stack. In this example, all three are in one address space, but separate instruction and data space is also supported.

96

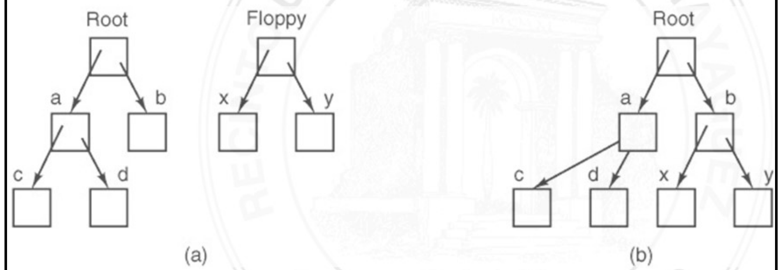
File Systems (1)



- Figure 1-6. A file system for a university department.

97

File Systems (2)



- Figure 1-7. (a) Before mounting, the files on drive 0 are not accessible. (b) After mounting, they are part of the file hierarchy.

98

LINUX

Directories in the Root of the Kernel Source Tree

Directory	Description
arch	Architecture-specific source
block	Block I/O layer
crypto	Crypto API
Documentation	Kernel source documentation
drivers	Device drivers
firmware	Device firmware needed to use certain drivers
fs	The VFS and the individual filesystems
include	Kernel headers
init	Kernel boot and initialization
ipc	Interprocess communication code
kernel	Core subsystems, such as the scheduler
lib	Helper routines
mm	Memory management subsystem and the VM
net	Networking subsystem
samples	Sample, demonstrative code
scripts	Scripts used to build the kernel
security	Linux Security Module
sound	Sound subsystem
usr	Early user-space code (called initramfs)
tools	Tools helpful for developing Linux
virt	Virtualization infrastructure

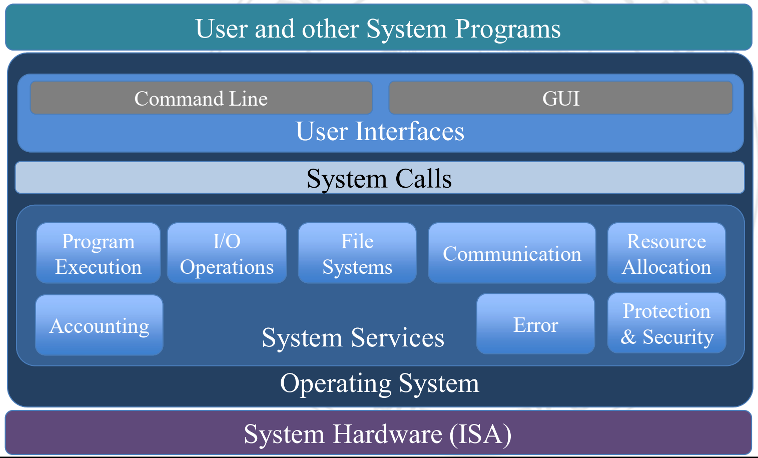
99

System Calls

- Interface that the Operating System uses to provide services to processes:
  - Process Management
  - Signals
  - File Management
  - File System and Directory Management
  - Protection
  - Time Management

100

A View of Operating System Services



101

Operating System Services (1)

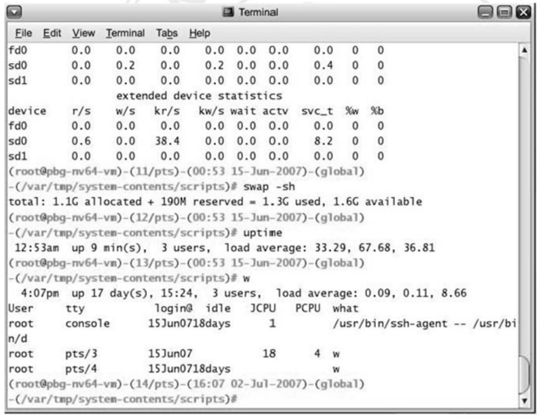
- One set of operating-system services provides functions that are helpful to the user:
  - **User interface** - Almost all operating systems have a user interface (UI)
    - Varies between Command-Line (CLI), Graphical User (GUI), Batch

102

User OS Interfaces - CLI

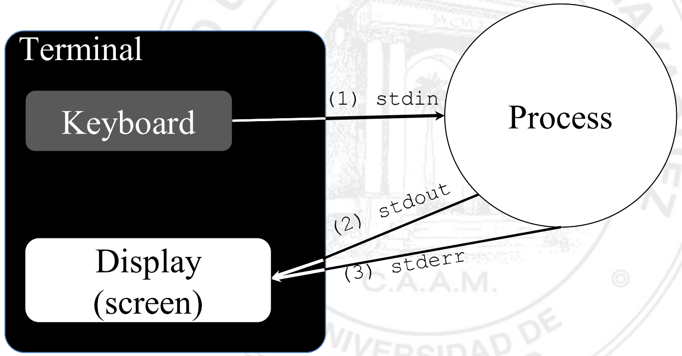
- Command Line Interface (CLI) or command interpreter allows direct command entry
  - Sometimes implemented in kernel, sometimes by systems program, sometimes multiple flavors implemented – shells
  - Primarily fetches a command from user and executes it
    - Standard streams:
      - standard in (stdin) keyboard, standard out (stdout) and standard error (stderr)

UNIX Shell Command Line Interface



```
Terminal
File  Edit  View  Terminal  Tabs  Help
fd0       0.0    0.0    0.0    0.0  0.0  0.0    0.0   0   0
sd0       0.0    0.2    0.0    0.2  0.0  0.0    0.4   0   0
sd1       0.0    0.0    0.0    0.0  0.0  0.0    0.0   0   0
                 extended device statistics
device    r/s    w/s   kr/s   kw/s wait actv  svc_t  %w  %b
fd0       0.0    0.0    0.0    0.0  0.0  0.0    0.0   0   0
sd0       0.6    0.0   38.4    0.0  0.0  0.0    8.2   0   0
sd1       0.0    0.0    0.0    0.0  0.0  0.0    0.0   0   0
(root@pbg-nv64-vm)-(11/pts)-(00:53 15-Jun-2007)-(global)
-(/var/tmp/system-contents/scripts)# swap -sh
total: 1.1G allocated + 190M reserved = 1.3G used, 1.6G available
(root@pbg-nv64-vm)-(12/pts)-(00:53 15-Jun-2007)-(global)
-(/var/tmp/system-contents/scripts)# uptime
 12:53am  up 9 min(s),  3 users,  load average: 33.29, 67.68, 36.81
(root@pbg-nv64-vm)-(13/pts)-(00:53 15-Jun-2007)-(global)
-(/var/tmp/system-contents/scripts)# w
  4:07pm  up 17 day(s), 15:24,  3 users,  load average: 0.09, 0.11, 8.66
User     tty           login@  idle   JCPU   PCPU  what
root     console      15Jun0718days      1         /usr/bin/ssh-agent -- /usr/bi
n/d
root     pts/3        15Jun07           18      4  w
root     pts/4        15Jun0718days                w
(root@pbg-nv64-vm)-(14/pts)-(16:07 02-Jul-2007)-(global)
-(/var/tmp/system-contents/scripts)#
```


Unix Shell Standard Streams



User OS Interfaces - 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

- **The Xerox Alto: the primary inspiration for the modern desktop computer.**
- It included:
  - mouse,
  - bit-mapped scheme,
  - windows-based user interface,
  - local network connection.

107

The Apple IIe Plus. Designed by Steve Wozniak.

108

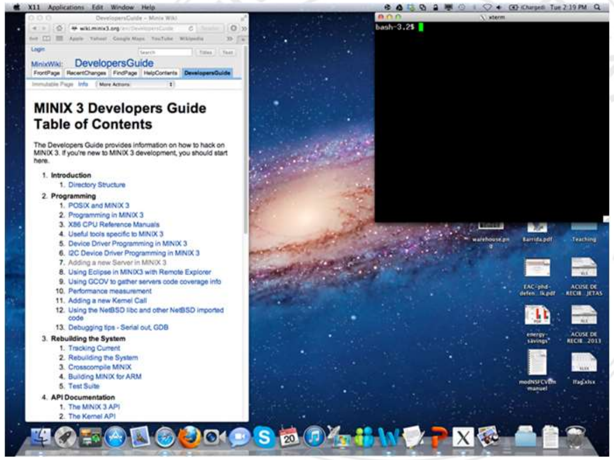
108

User OS Interfaces

- Many systems now include both CLI (emulated) 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)

109

The Mac OS X GUI (Aqua)



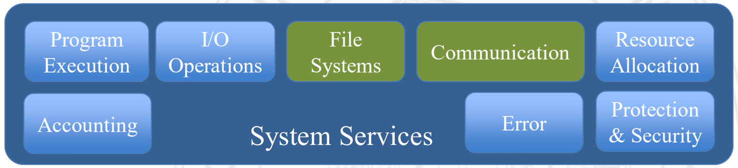
110

Operating System Services (2)

- **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

111

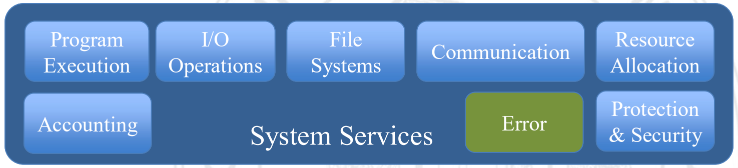
Operating System Services (3)



- **File-system manipulation** - Programs 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 message passing (packets moved by the OS)

112

Operating System Services (4)



- **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

113

Operating System Services (5)

- **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

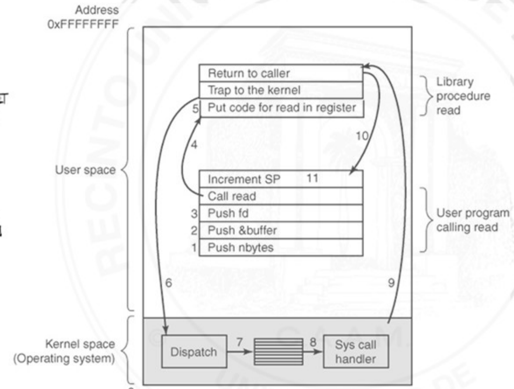
114

Operating System Services (6)

- **Accounting -** To keep track of system behavior - log
- **Protection and security -**
  - **Protection** ensuring that all access to system resources is controlled
  - **Security** requires user authentication, 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.

115

Operating System Structure

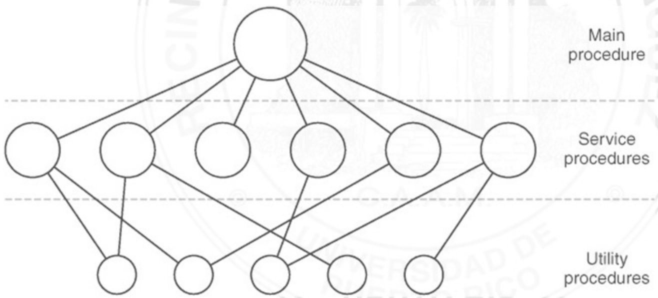


- Figure 1-16. The 11 steps in making the system call read(fd, buffer, nbytes).

116

Basic Structure for Operating System (Monolithic)

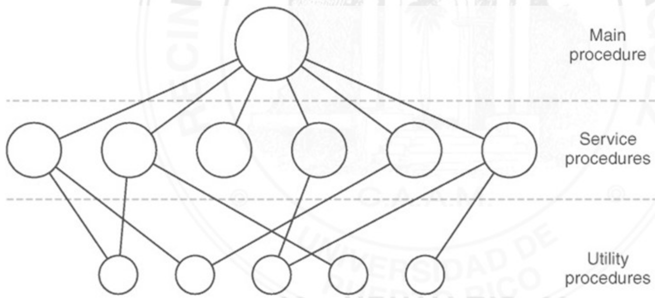
1. A main program that invokes the requested service procedure
2. A set of service procedures that carry out the system calls
3. A set of utility procedures that help the service procedures



117

Monolithic Systems

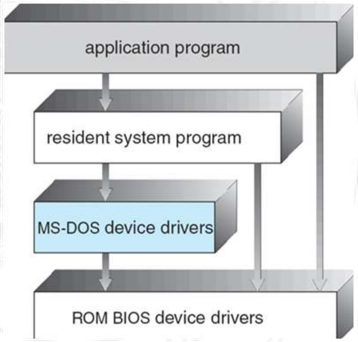
- Figure 1-17. A simple structuring model for a monolithic system.



118

Monolithic (Simple) Structure

- 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



119

Layered Systems (1)

- 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

120

Layered Systems (2)

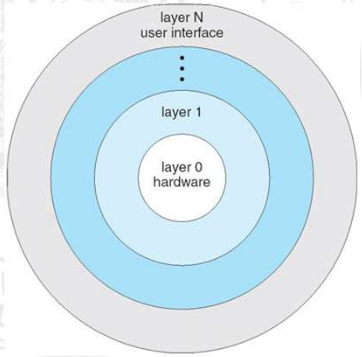
- Figure 1-18. Structure of the THE Operating System (THEOS).

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

121

Layered OS: The Multiplexed Information and Computer Services (MULTICS)

- Many novel ideas:
  - Memory mapped files
  - Dynamic linking
  - Online reconfiguration
  - Hierarchical filesystem
  - Too ambitious (complex) for its era



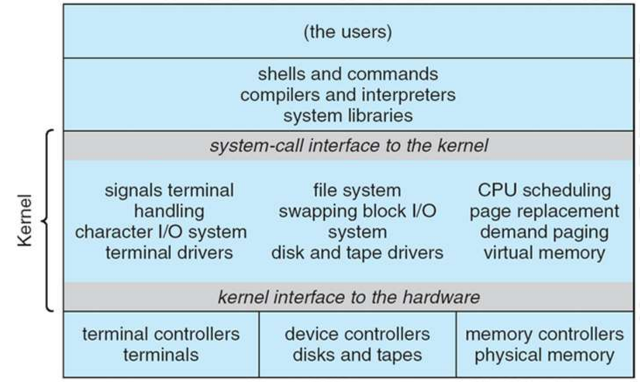
122

UNICS – UNIplexed Information and Computer Services (D. Ritchie and K. Thompson)

- 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

123

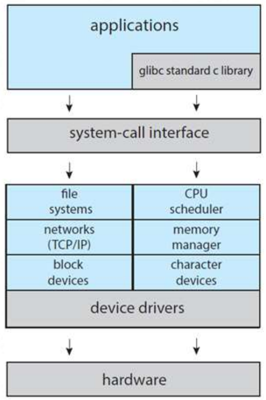
Traditional UNIX System Structure



124

The UNIX 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



Linux system structure

125

Microkernel System Structure (1) Mini-Unix (MINIX)

- Moves as much from the kernel into "*user*" space
- Take advantage of the Client-Server model
  - Communication takes place between user modules using message passing

126

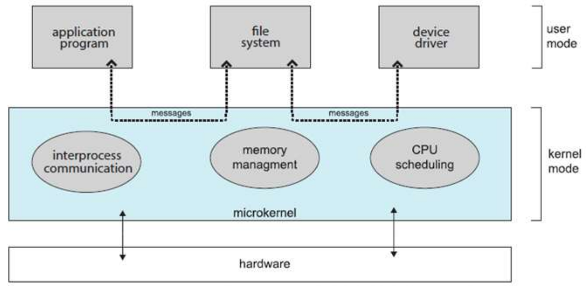
Microkernel System Structure (2) Mini-Unix (MINIX)

- 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

127

Microkernel

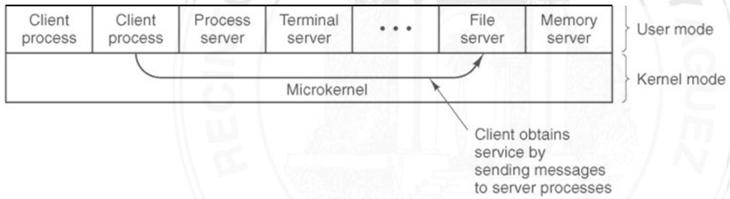
- Darwin/Mach – iOS / MacOS X
- Minix



128

Use of the Client-Server Model (1)

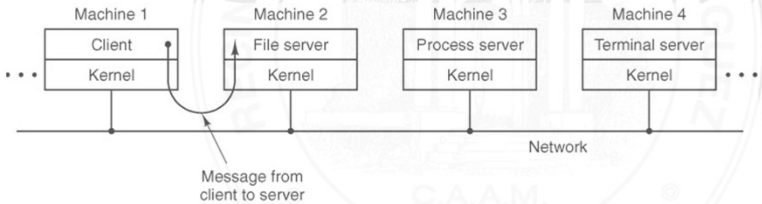
- Figure 1-20. The client-server model.



129

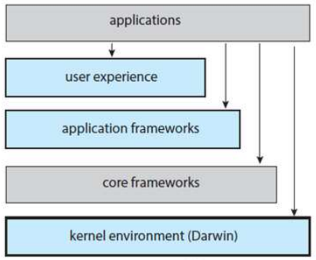
Use of the Client-Server Model (2)

- Figure 1-21. The client-server model in a distributed system.

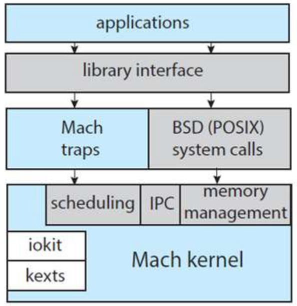


130

MacOS X and iOS Structure



Architecture of macOS and iOS



Structure of Darwin

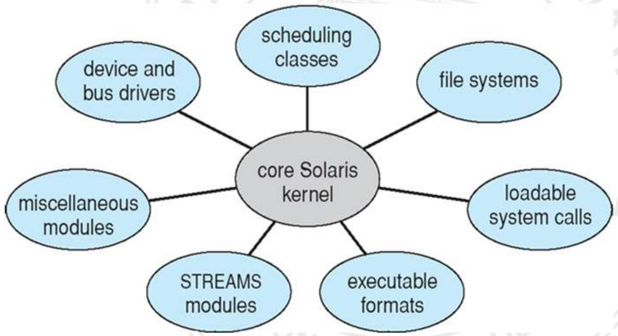
131

Modules

- Most modern operating systems implement kernel modules
  - Uses object-oriented approach
  - Each core component is separate
  - Each talks to the others over known interfaces
  - Each is loadable as needed within the kernel
- Overall, similar to layers but more flexible

132

Solaris Modular Approach



133

Virtual Machines

- A virtual machine (VM) takes the layered approach to its logical conclusion. It treats hardware and the OS kernel as they were all hardware
- A VM provides an interface *identical* to the underlying bare hardware
- The OS host creates the illusion that a process has its own processor and (virtual) memory
- Each guest has a (virtual) copy of underlying computer

134

Virtual Machines History and Benefits (1)

- First appeared in IBM mainframes (VM/370 1972)
- Fundamentally, multiple execution environments (different operating systems) can share the same hardware
- Protect from each other
- Some sharing of file can be permitted, controlled

135

IBM System/360 computers: models 40, 50, 65, and 75 were all introduced in 1964.

136

136

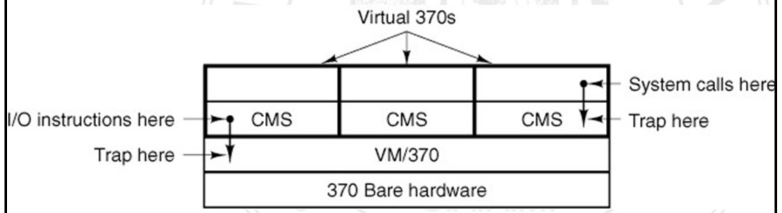
Virtual Machines History and Benefits (2)

- Commutate with each other, other physical systems via networking
- Useful for development, testing
- Consolidation of many low-resource use systems onto fewer busier systems
- “Open Virtual Machine Format”, standard format of virtual machines, allows a VM to run within many different virtual machine (host) platforms

137

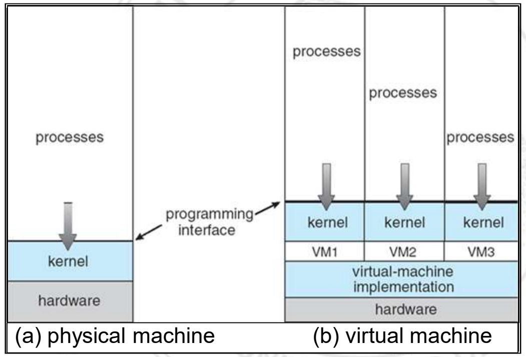
IBM Virtual Machines

- Figure 1-19. The structure of VM/370 with CMS.



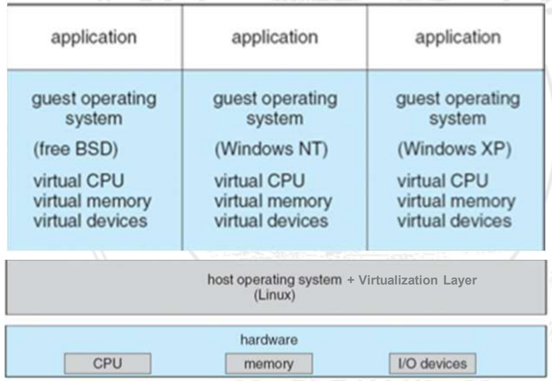
138

Virtual Machines (Cont)



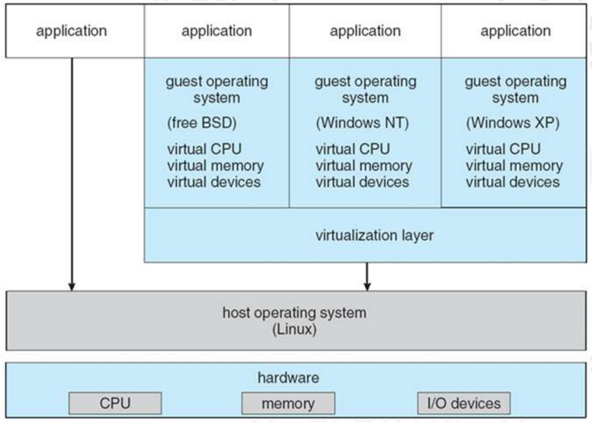
139

VMware Architecture (type-1,hostless, *unhosted*)



140

VMware Architecture (type-2, hosted)



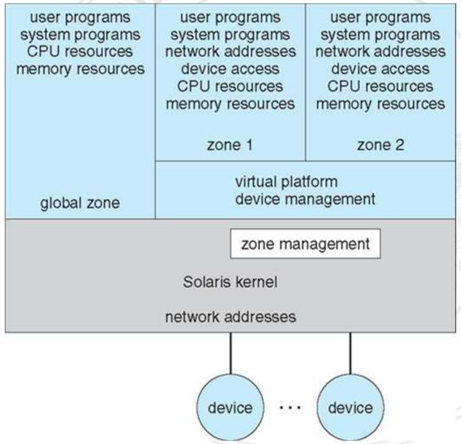
141

Para-virtualization

- Presents guest with system similar but not identical to hardware
- Guest must be modified to run on paravirtualized hardware
  - Performance limitations
  - Portability issues
- Guest can be an OS, or in the case of Solaris 10 applications running in containers

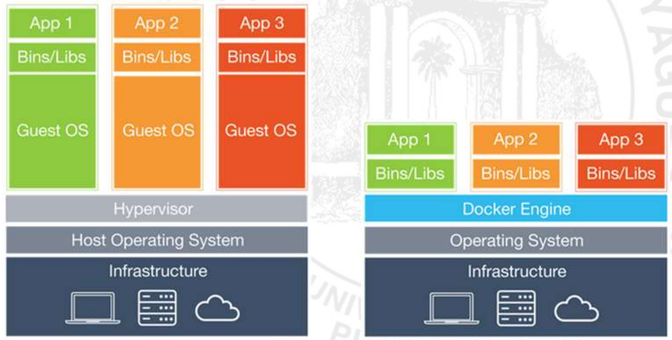
142

Solaris 10 with Two Containers



143

Using Containers: Docker, Kubernetes



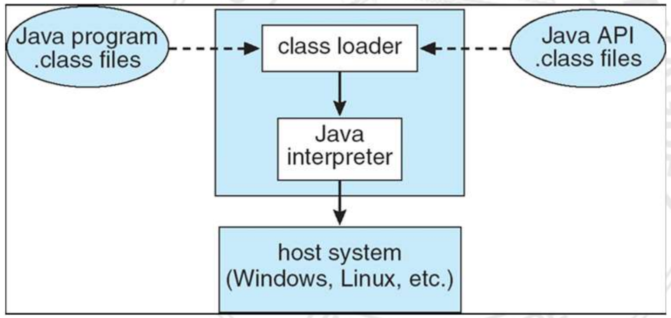
144

Java

- Java consists of
  - Programming language specification
  - Application programming interface (API)
  - Virtual machine specification

145

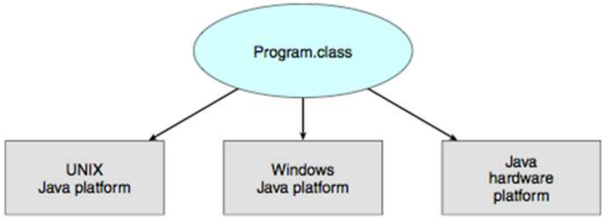
The Java Virtual Machine



146

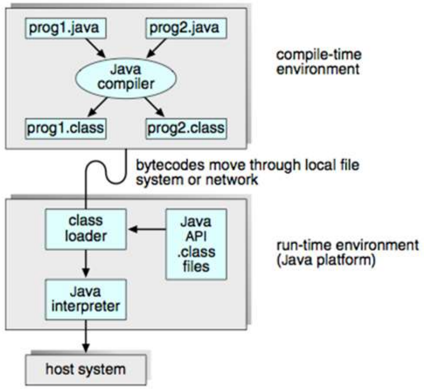
The Java Virtual Machine

- Java portability across platforms.



147

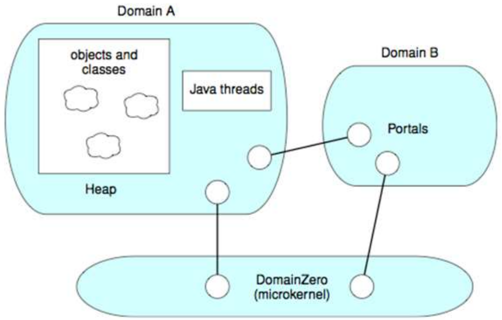
The Java Development Kit



148

Java Operating Systems

- The JX operating system



149

Operating-System Debugging (1)

- Debugging is finding and fixing errors, or bugs
- OSes 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

150

Operating-System Debugging (2)

- 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.”

151

DTrace Following System Call

```
# ./all.d 'pgrep xclock' XEventsQueued
dtrace: script './all.d' matched 52377 probes
CPU FUNCTION
  0 -> XEventsQueued                       U
  0   -> _XEventsQueued                    U
  0     -> _X11TransBytesReadable          U
  0     <- _X11TransBytesReadable          U
  0     -> _X11TransSocketBytesReadable    U
  0     <- _X11TransSocketBytesreadable    U
  0     -> ioctl                           U
  0       -> ioctl                         K
  0         -> getf                        K
  0           -> set_active_fd             K
  0           <- set_active_fd             K
  0         <- getf                        K
  0         -> get_udatamodel              K
  0         <- get_udatamodel              K
...
  0         -> releasef                    K
  0           -> clear_active_fd           K
  0           <- clear_active_fd           K
  0           -> cv_broadcast              K
  0           <- cv_broadcast              K
  0         <- releasef                    K
  0       <- ioctl                         K
  0     <- ioctl                           U
  0   <- _XEventsQueued                    U
  0 <- XEventsQueued                       U
```

- DTrace tool in Solaris, FreeBSD, Mac OS X allows live instrumentation on production systems
  - Probes fire when code is executed, capturing state data and sending it to consumers of those probes

152

Operating System Generation

- 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
- *Booting* – start computer by loading the kernel
  - *Bootstrap program* – code stored in ROM that is able to locate the kernel, load it into memory, and start its execution

153

System Boot

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

154