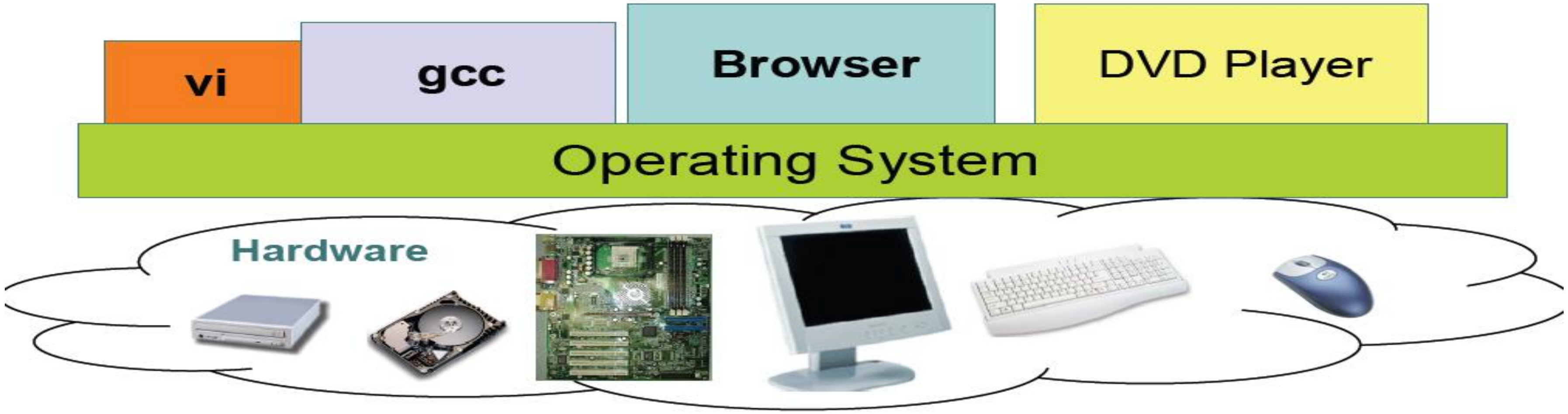
MODERN OPERATING SYSTEMS

Fourth Edition

ANDREW S. TANENBAUM

Chapter 1 Introduction

What Is An Operating System



- Software that sits between applications and hardware
  - (Also between different applications, and between different users, see later)
- Provides services and interfaces to applications
- Has privileged access to hardware

- User applications call OS routines for this access and for the services

What Is An Operating System (2)

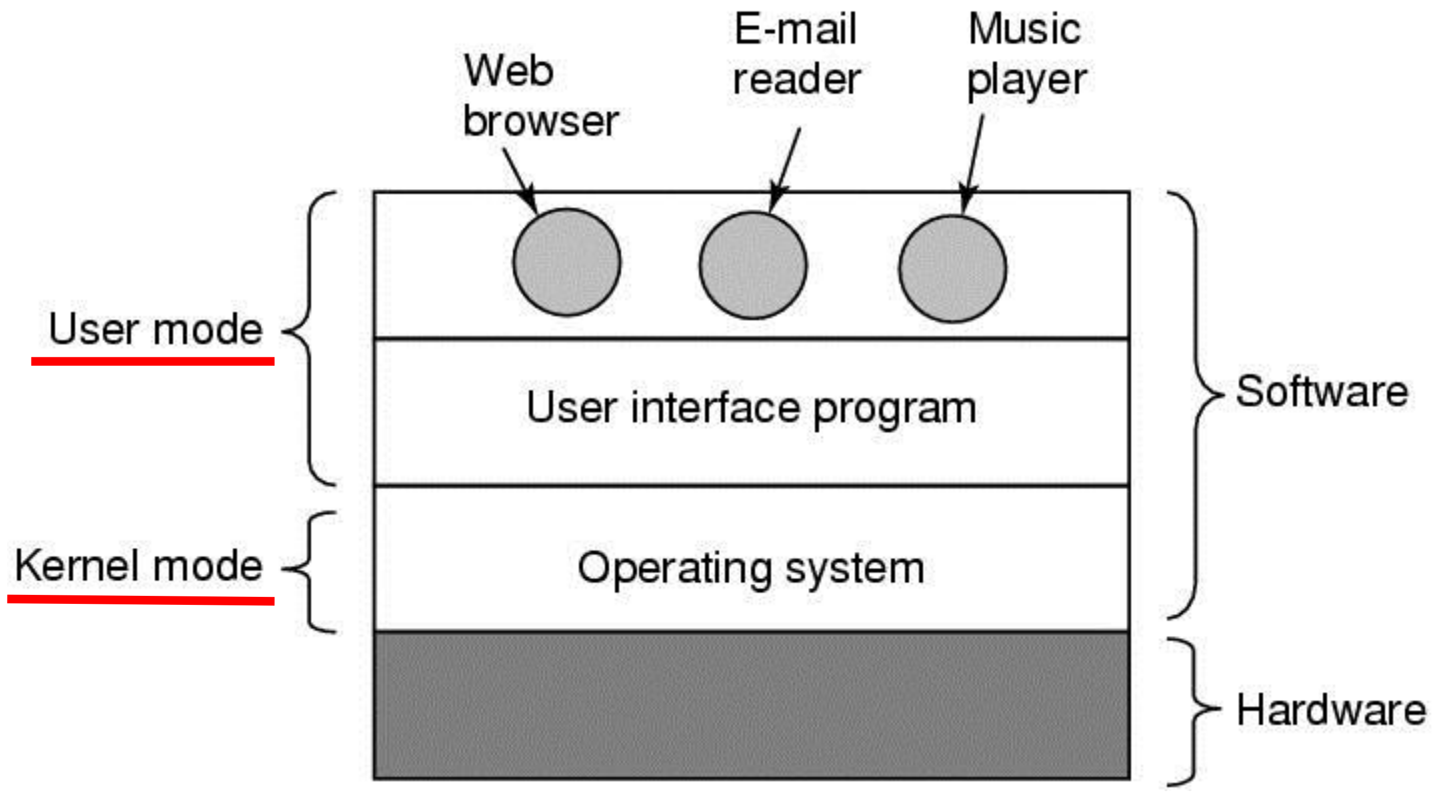


Figure 1-1. Where the operating system fits in.

What Does an Operating System Do?

- Provides a layer of abstraction for hardware resources
  - Allows user programs to deal with higher-level, simpler and more portable concepts than the vagaries of raw hardware
    - » E.g. files rather than disk blocks
  - Makes finite resources seem infinite

- Manages these resources
  - Manages complex resources and their interactions for an application
  - Allows multiple applications to share resources without hurting one another
  - Allows multiple users to share resources without hurting one another

The Operating System as an Extended Machine

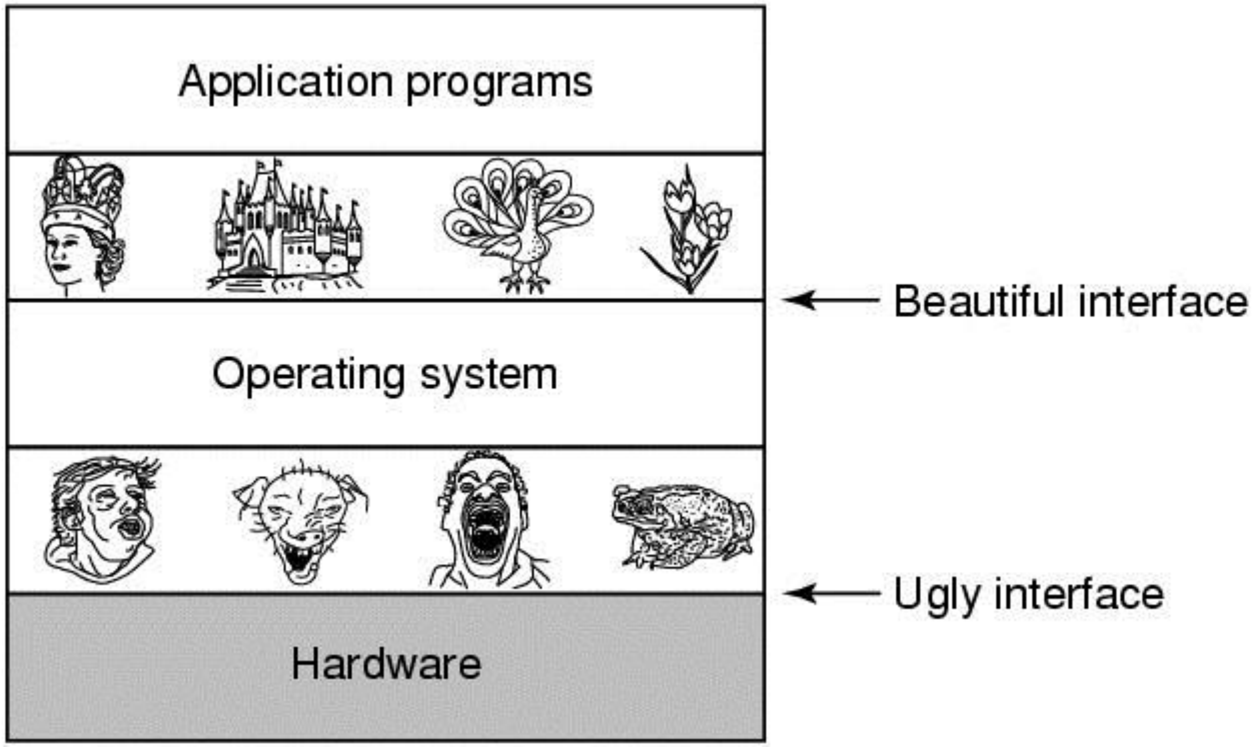


Figure 1-2. Operating systems turn ugly hardware into beautiful abstractions.

The Operating System as a Resource Manager

- Allow multiple programs to run at the same time
- Manage and protect memory, I/O devices, and other resources
- Includes multiplexing (sharing) resources in two different ways:
  - In time – CPU sharing
  - In space – Memory sharing

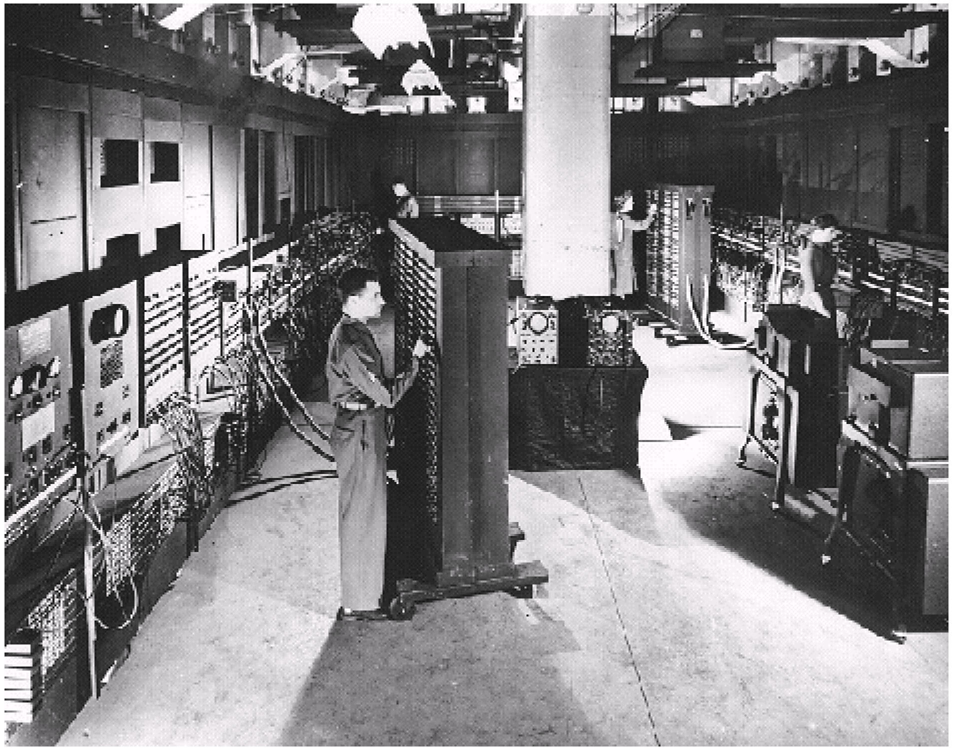
History of Operating Systems

- First generation 1945 - 1955
  - vacuum tubes, plug boards
- Second generation 1955 - 1965
  - transistors, batch systems
- Third generation 1965 – 1980
  - ICs and multiprogramming
- Fourth generation 1980 – present
  - personal computers
- Fifth generation 1990–present
  - mobile computers

Key Technical Ideas

- Binary rather than decimal
- Vacuum tubes for logic rather than storage
- Inexpensive rotating capacitors for (cheap) temporary storage
- Punch cards for input/output

The ENIAC



Electronic Numerical Integrator and Computer

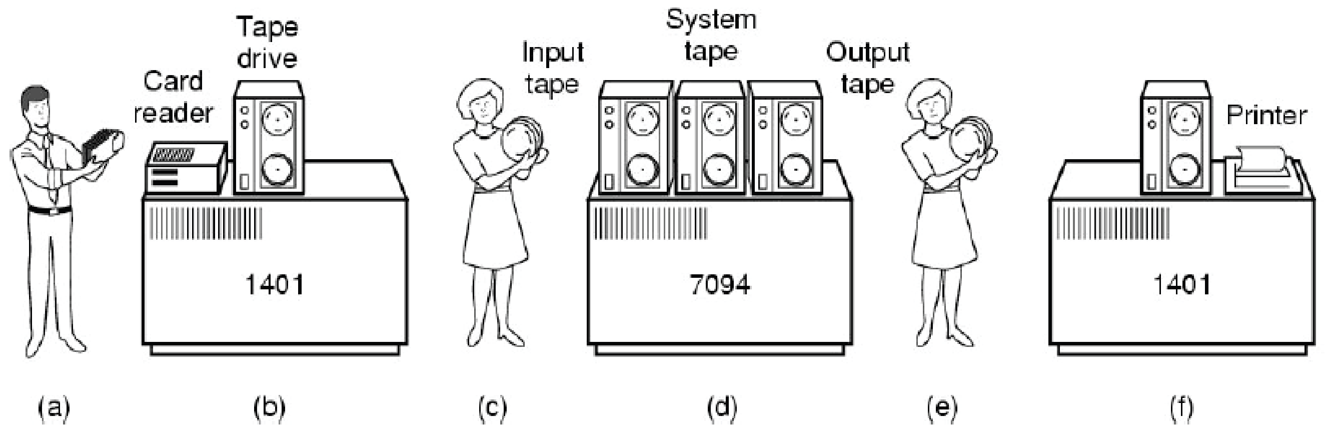
- **Started in 1942**
- **Completed in 1945**
- **Key stats**
  - **$500K**
  - **30 tons**
  - **1800 square feet**
  - **19000 vacuum tubes**
  - **175 Kw/power**
  - **5000 ops/second**

- **Followed 2 years later by EDVAC.**
  - **Von Neumann**
  - **Stored Program**

History of Operating Systems

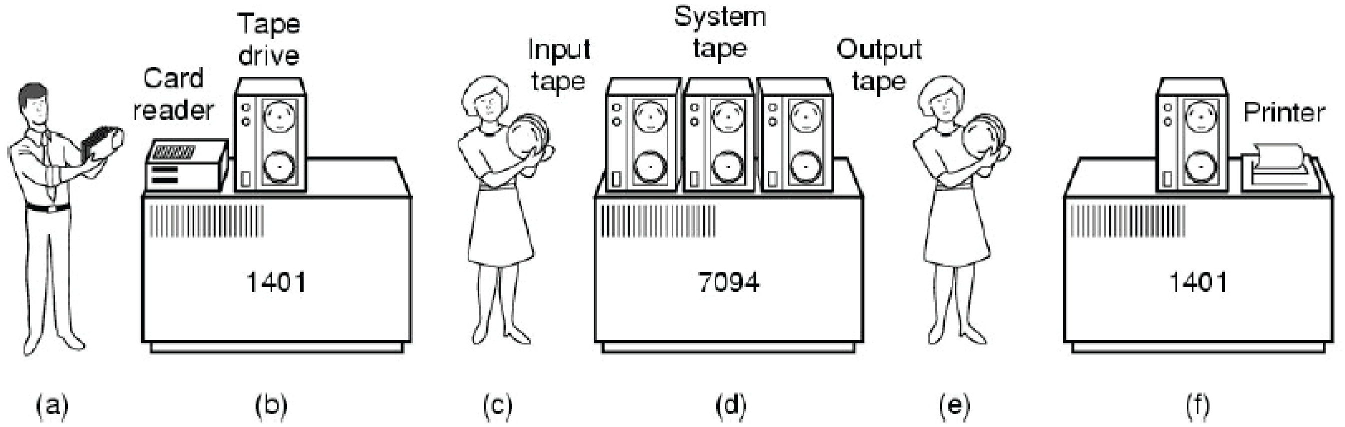
- **First generation 1945 - 1955**
  - **vacuum tubes, plug boards**
- **Second generation 1955 - 1965**
  - **transistors, batch systems**
- **Third generation 1965 – 1980**
  - **ICs and multiprogramming**
- **Fourth generation 1980 – present**
  - **personal computers**
- **Fifth generation 1990–present**
  - **mobile computers**

Phase 2: Transistors and Batch Systems



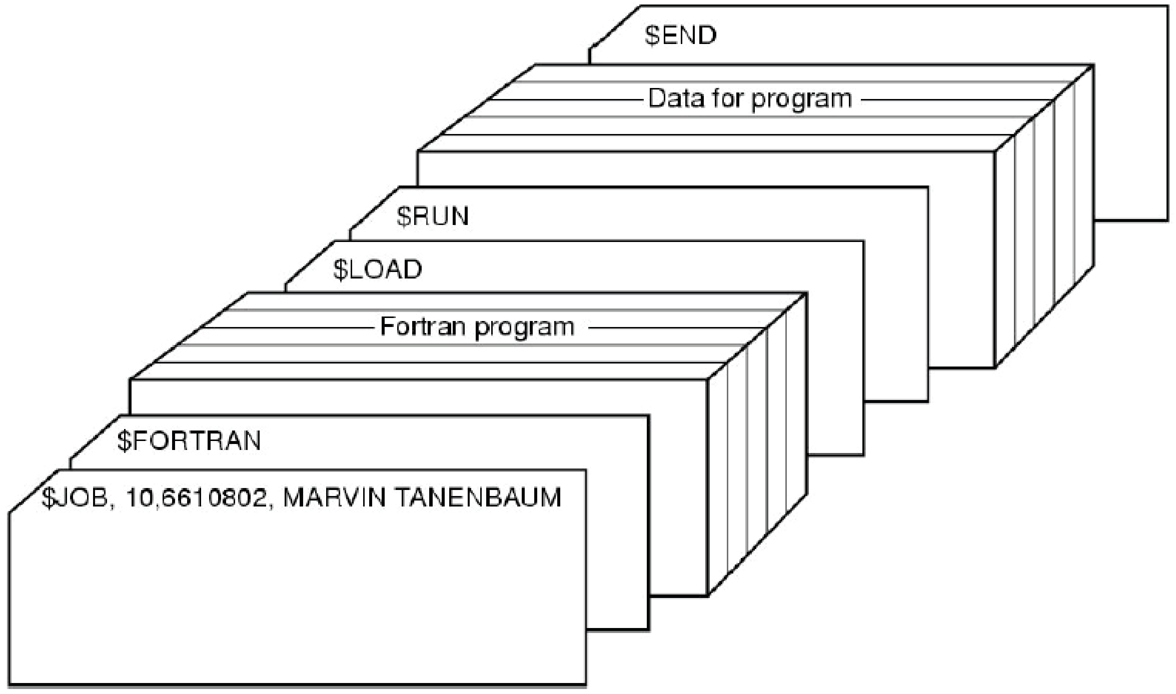
- Hardware still expensive, humans relatively cheap
- An early batch system
  - Programmers bring cards to reader system
  - Reader system puts jobs on tape

Phase 2: Transistors and Batch Systems



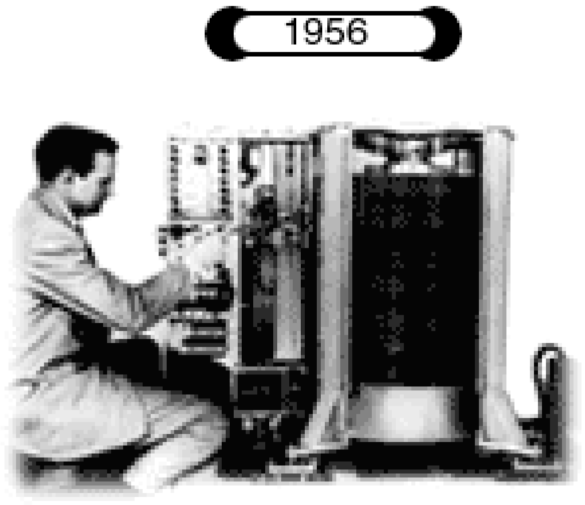
- An early batch system
  - Operator carries input tape to main computer
  - Main computer computes and puts output on tape
  - Operator carries output tape to printer system, which prints output

Punch cards and Computer Jobs



IBM Invents The Disk in 1956

- IP#4: Disks, which allow Random Access to Storage
  - –Run multiple programs at "once" by swapping image to disk
    - OS must choose which to run next when current ends, or performs I/O (eg, fetch data from disk)
    - job scheduling. EARLY POLICY/MECHANISM SEPARATION EXAMPLE
  - –Problem: CPU still idle during I/O



First computer disk storage system. The 305 RAMAC (Random Access Method of Accounting and Control) could store five million characters (five megabytes) of data on 50 disks, each 24 inches in diameter. RAMAC's revolutionary recording head could go directly to any location on a disk surface without reading the information in between. This IBM innovation made it possible to use computers for airline reservations, automated banking, medical diagnosis, and space flights.

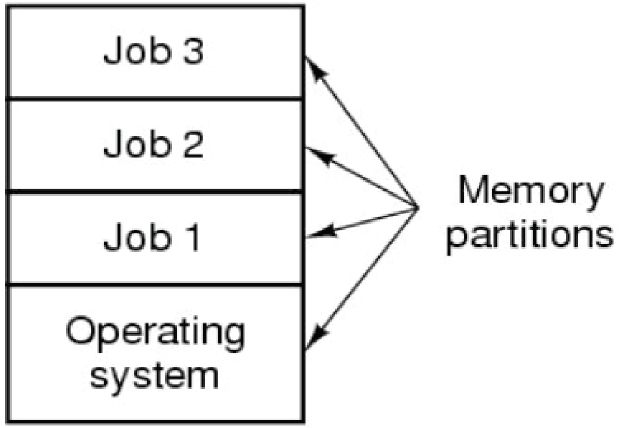
History of Operating Systems

- First generation 1945 - 1955
  - vacuum tubes, plug boards
- Second generation 1955 - 1965
  - transistors, batch systems
- Third generation 1965 – 1980
  - ICs and multiprogramming
- Fourth generation 1980 – present
  - personal computers
- Fifth generation 1990–present
  - mobile computers

Multiprogramming by 1960

- IP#5: Big, Cheap Memory, to keep multiple programs in core at the same time
  - Again, to increase system utilization
  - keeps multiple runnable jobs loaded in memory at once
  - overlaps I/O of a job with computing of another
    - » while one job waits for I/O completion, OS runs instructions from another job
  - to benefit, need asynchronous I/O devices
    - » need some way to know when devices are done
      - interrupts
      - polling
  - goal: optimize system throughput
    - » perhaps at the cost of response time…
  - Yielded the invention of the "PROCESS" – a program which is executing in memory
- Problem: Computer is *NOT interactive*

Phase 3: ICs and Multiprogramming



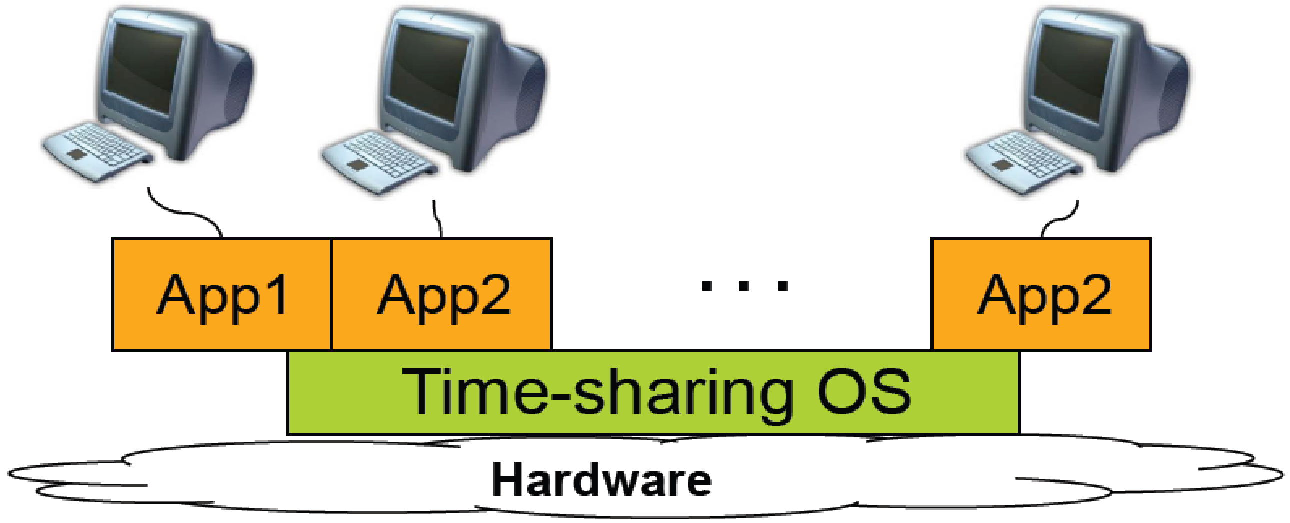
- Multiple jobs resident in computer's memory
- Hardware switches between them (interrupts)
- Hardware protects from one another (mem protection)
- Computer reads jobs from cards as jobs finish (spooling)
- Still batch systems: can't debug online
- Solution: time-sharing

Timesharing by the mid-60s

- IP#6: Even cheaper memory, modems, and inexpensive CRTs, allowing multiple users to “interact” with the computer at the same time
  - To support interactive use, create a timesharing OS:
    - » multiple terminals into one machine
    - » each user has illusion of entire machine to him/herself
    - » optimize response time, perhaps at the cost of throughput
  - Timeslicing
    - » divide CPU equally among the users
    - » if job is truly interactive (e.g. editor), then can jump between programs and users faster than users can generate load
    - » permits users to interactively view, edit, debug running programs
  - MIT Multics system (mid-1960’s) was the first large timeshared system (based on the CTSS for the IBM 7094 from 1961)
    - » nearly all OS concepts can be traced back to Multics
    - » UNIX is just a simpler MULTICS
      - Core of Mac OS X, Linux, etc.
- Problem: The computer ran faster at night

Phase 3: ICs and Multiprogramming

- Time-sharing:
  - Users at terminals simultaneously
  - Computer switches among active 'jobs'/sessions
  - Shorter, interactive commands serviced faster



The Multics System

1968: Honeywell 645

1975: Honeywell 6180

Last one decommissioned in 2000

Unix

- Ken Thompson, PDP-7, C language
- AT&T – System V, UC Berkeley – BSD
- Sun – Sun OS, Solaris
- IBM – AIX, HP – HP UNIX
- Minix, Xenix
- Linux, FreeBSD, NetBSD

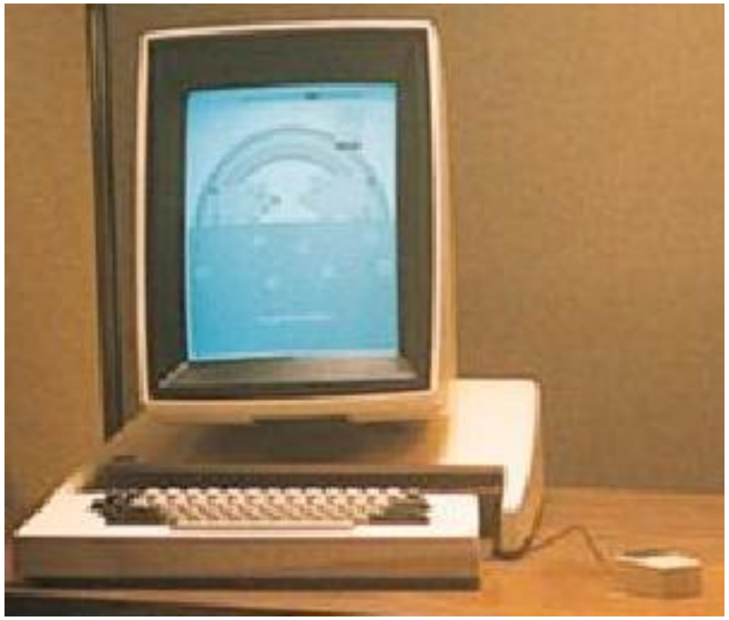
History of Operating Systems

- First generation 1945 - 1955
  - vacuum tubes, plug boards
- Second generation 1955 - 1965
  - transistors, batch systems
- Third generation 1965 – 1980
  - ICs and multiprogramming
- Fourth generation 1980 – present
  - personal computers
- Fifth generation 1990–present
  - mobile computers

Personal and Distributed Computing by the 1970s

- **IP#7: Plummeting cost of silicon and networking allows everyone to have their own computer**
  - Memory was under a penny/bit
  - "Share everything but the time"
  - distributed systems using geographically distributed resources
    - » workstations on a LAN
    - » servers across the Internet
  - OS supports communications between jobs
    - » interprocess communication
    - » networking stacks
  - OS supports sharing of distributed resources (hardware, software)
    - » load balancing, authentication and access control, …
  - speedup isn't the issue
    - » access to diversity of resources is goal
- **Problem: there's never a time that the machine runs faster**

XEROX ALTO -- 1972



< $50K

First personal workstation First wide deployment of:

- •Bit-map graphics
- •Mouse
- •WYSIWG editing

Hosted the invention of:

- •Local-area networking
- •Laser printing
- •All of modern client / server distributed computing

Parallel Systems by the 80's

- IP#8: High Speed Interconnects allow multiple processors to cooperate on a single program
  - Some applications can be written as multiple parallel threads or processes
  - can speed up the execution by running multiple threads/processes simultaneously on multiple CPUs
  - need OS and language primitives for dividing program into multiple parallel activities
  - need OS primitives for fast communication between activities
    - » degree of speedup dictated by communication/computation ratio
  - many flavors of parallel computers
    - » SMPs (symmetric multi-processors)
    - » MPPs (massively parallel processors)
    - » NOWs (networks of workstations)
    - » computational grid (SETI @home)

Now: Multiple Processors per Machine

- Multiprocessors
  - SMP: Symmetric MultiProcessor
  - ccNUMA: Cache-Coherent Non-Uniform Memory Access
  - General-purpose, single-image OS with multiproccesor support
- Multicomputers
  - Supercomputer with many CPUs and high-speed communication
  - Specialized OS with special message-passing support
- Clusters
  - A network of PCs
  - Commodity OS

Internet by the 90s

- IP#9: WEB, HTTP, TCP at Scale
  - TCP+HTTP+Web Browser has changed the way that the world looks at computing
  - BUT, there have been relatively few Operating System Advances to support this
    - » The internet was functional by the 70s
  - Mostly, it’s been a time of leveraging the last 50 years to deploy MASSIVELY SCALABLE SYSTEMS

History of Operating Systems

- First generation 1945 - 1955
  - vacuum tubes, plug boards
- Second generation 1955 - 1965
  - transistors, batch systems
- Third generation 1965 – 1980
  - ICs and multiprogramming
- Fourth generation 1980 – present
  - personal computers
- Fifth generation 1990–present
  - mobile computers

Ubiquitous Computing in the 21st Century

- **IP#10:** Massive miniaturization and integration of computers + Wireless
  - Ubiquitous computing
    - » cheap processors embedded everywhere and anywhere
    - » how many are on your body now? in your car?
    - » cell phones, PDAs, network computers, …
  - Typically very constrained hardware resources
    - » (relatively) slow processors
    - » very small amount of memory (e.g. 8 MB)
    - » no disk
  - OS researchers are working in this area today to understand what we'll be using tomorrow.

Personal Computer OS and More

- Digital Research - CP/M
- Seattle Computer Products - DOS
- Microsoft – MS-DOS for IBM PC
- Apple Mac OS – GUI, mouse
- Microsoft
  - Windows 3.1, 95, 98, Me
  - Windows NT, 2000, XP, Vista, 7
- iPhone OS, Android, Symbian OS, 바다, VxWork, ….

1.3 Computer Hardware Review

CPU Pipelining

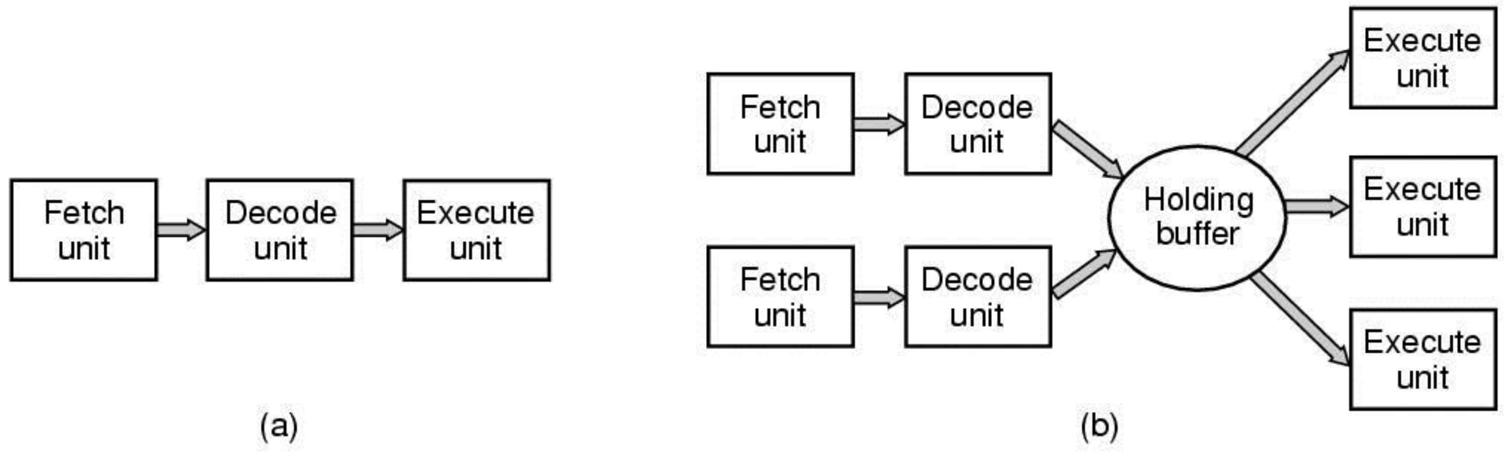


Figure 1-7. (a) A three-stage pipeline. (b) A superscalar CPU.

- Kernel mode and User mode
- System call – Trap
- Registers – program counter, stack pointer, general purpose register, Program Status Word

Multithreaded and Multicore Chips

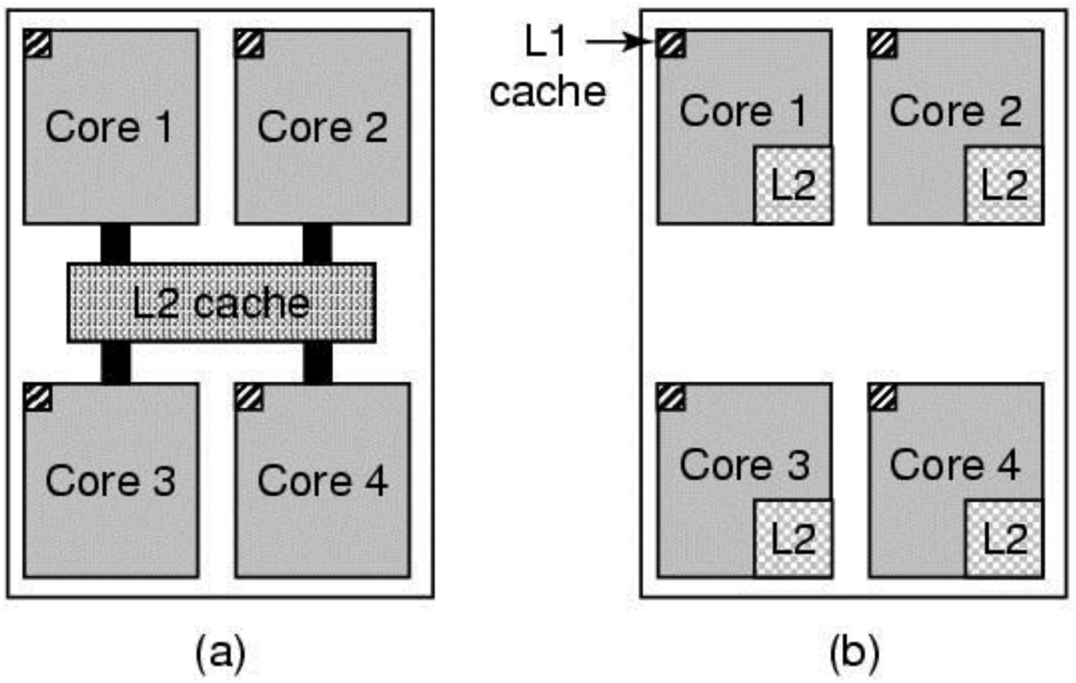


Figure 1-8. (a) A quad-core chip with a shared L2 cache.
(b) A quad-core chip with separate L2 caches.

Memory (1)

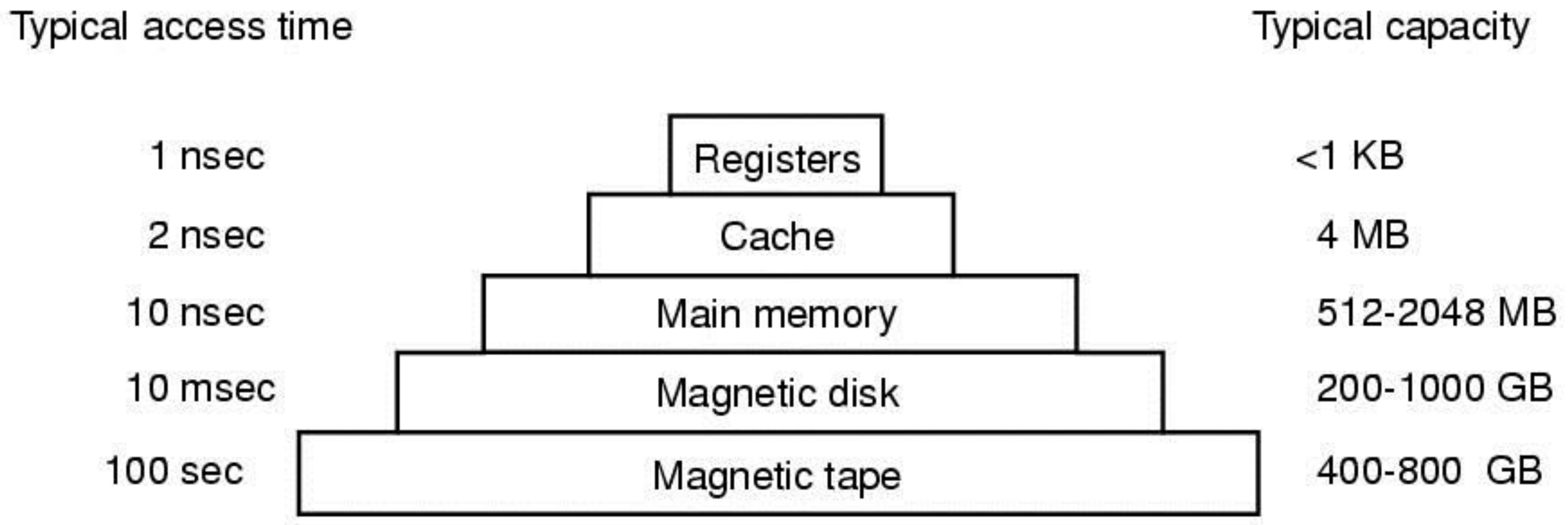


Figure 1-9. A typical memory hierarchy.
The numbers are very rough approximations.

Disks

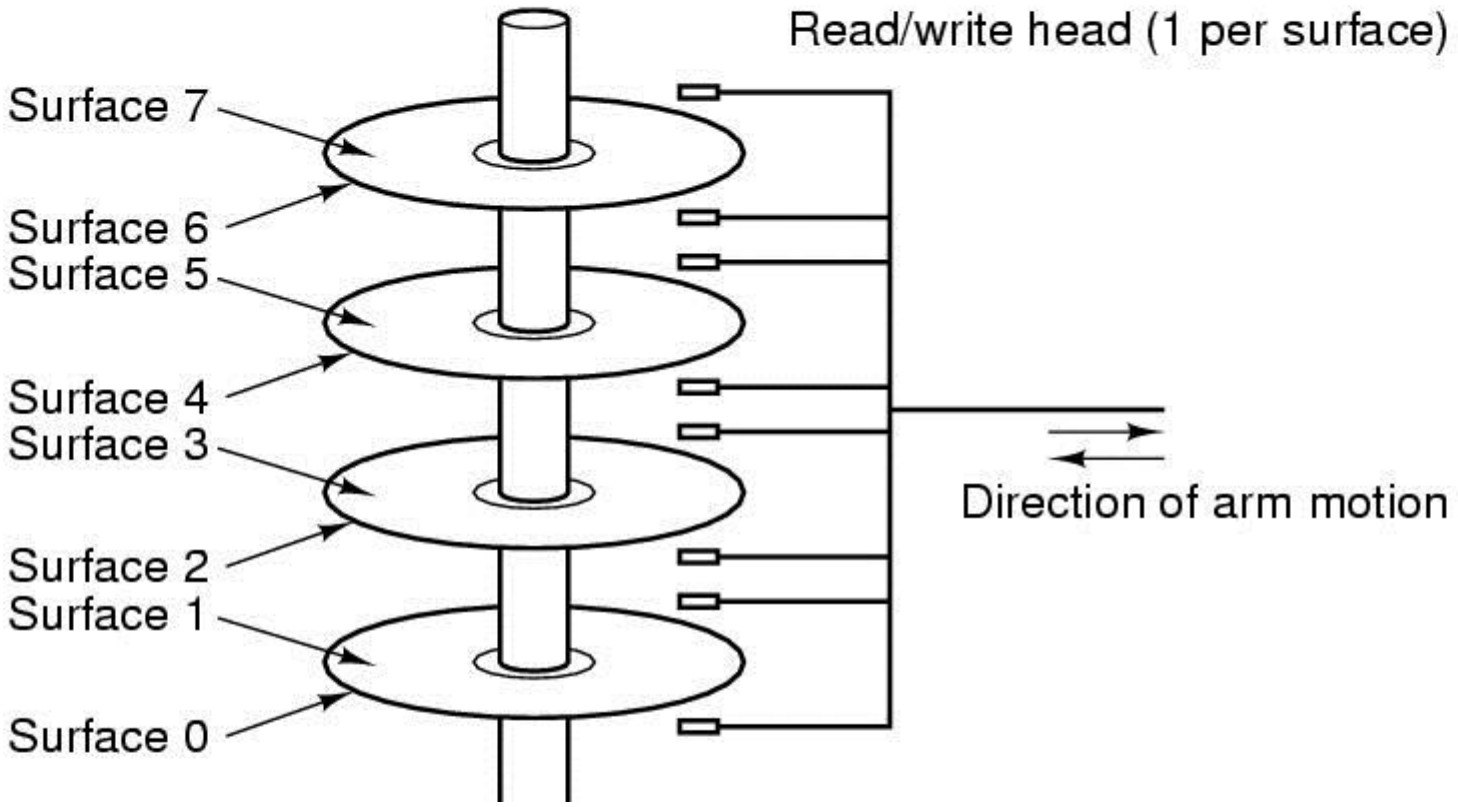
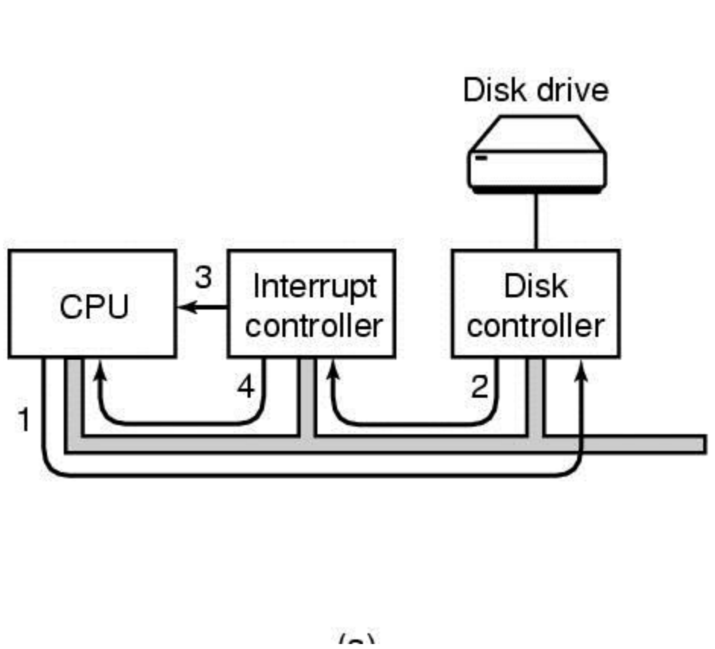


Figure 1-10. Structure of a disk drive.

Disks

- Cylinder, track
- Virtual memory
- MMU (Memory Management Unit)
- Context Switch

I/O Devices



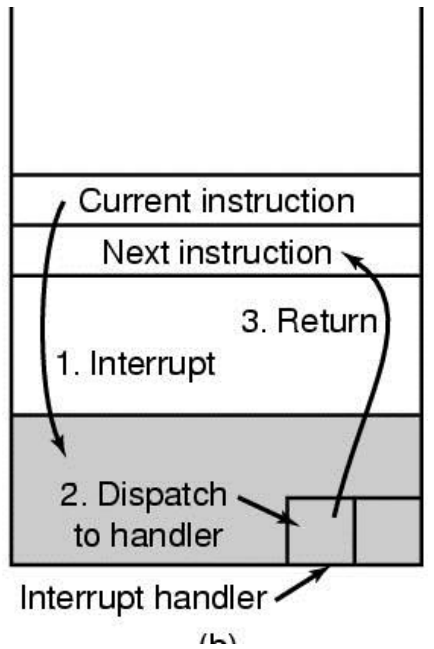


Figure 1-11. (a) The steps in starting an I/O device and getting an interrupt.

I/O Devices

- Device Driver – Device Controller
- I/O port space for device register
- I/O Access – busy waiting, interrupt, DMA (Direct Memory Access)
- Interrupt Vector – addresses of interrupt handlers

Buses

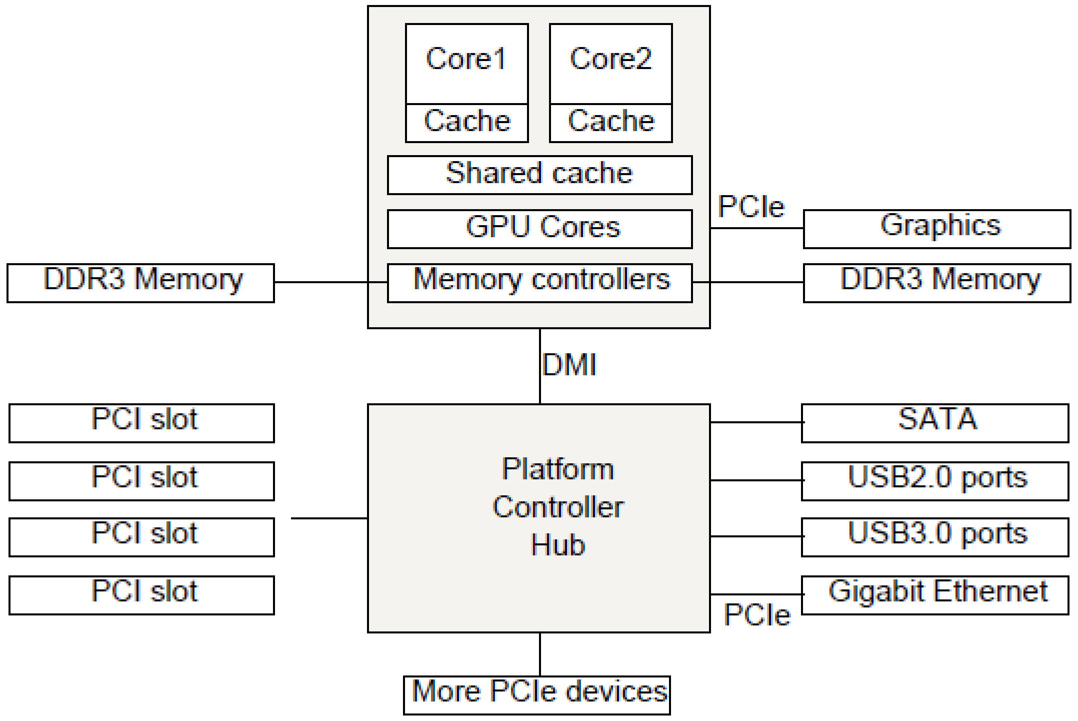


Figure 1-12. The structure of a large x86 system

Buses

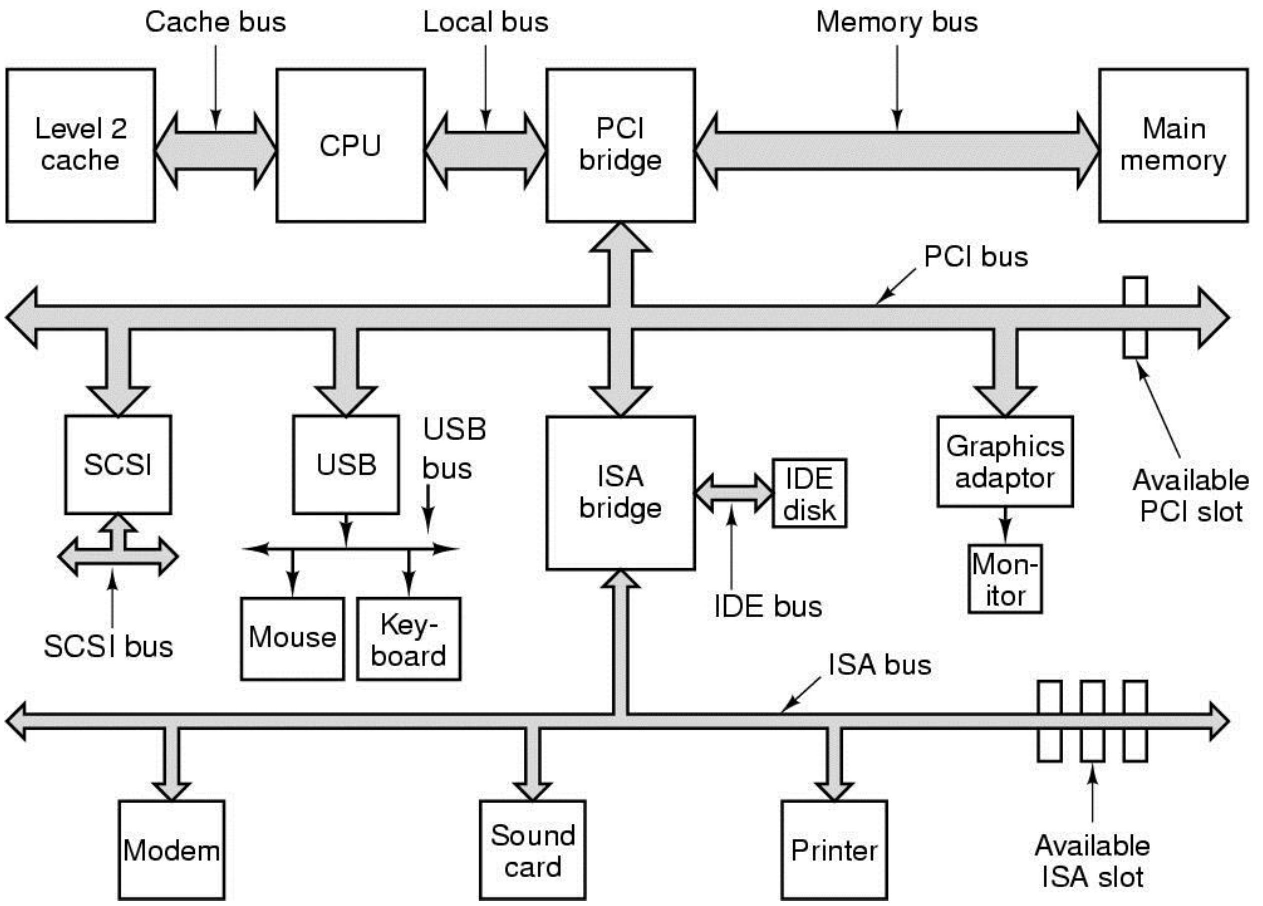


Figure 1-12. The structure of a large Pentium system

The Operating System Zoo

- Mainframe operating systems
- Server operating systems
- Multiprocessor operating systems
- Personal computer operating systems
- Handheld operating systems
- Embedded operating systems
- Sensor node operating systems
- Real-time operating systems
- Smart card operating systems

The Operating System Zoo

- Mainframe operating systems
  - Heavily oriented toward processing many jobs at once, most of which need large amounts of I/O
  - High-end Web servers, servers for large-scale electronic commerce sites, and servers for business-to-business transactions
  - Three kinds of service
    - Batch
      - Processes routine jobs without any interactive user present
      - Claims processing in an insurance company or sales reporting for a chain of stores
    - Transaction processing
      - Each unit of work is small, but the system must handle hundreds or thousands per second
      - Check processing at a bank or airline reservations
    - Timesharing
      - Allows multiple remote users to run jobs on the computer at once
      - Querying a big database
    - Mainframe operating systems often perform all of them.
  - IBM OS/390

The Operating System Zoo

- Server operating systems
  - Run on servers (very large personal computers, workstations, or even mainframes)
  - Serve multiple users at once over a network and allow users to share hardware and software resources
  - Print service, file service, Web service
  - UNIX, Windows

- Multiprocessor operating systems
  - Connect multiple CPUs into a single system to get a major-league computing power
  - Variations on the server operating systems, with features for communication and connectivity

- Personal computer operating systems
  - To provide a good interface to a single user
  - Used for word processing, spreadsheets, and Internet access
  - Windows, Macintosh, and Linux

The Operating System Zoo

- Real-time operating systems
  - Time is a key parameter
  - Hard real-time system
    - The action absolutely must occur within the deadline.
    - E.g. car assembly line, flight control, military device
  - Soft real-time system
    - Missing an occasional deadline is acceptable.
    - E.g. digital audio, multimedia systems
  - VxWorks, QNX

The Operating System Zoo

- **Embedded operating systems**
  - PDA(Personal Digital Assistant)
  - Embedded systems
    - » TV sets, microwave ovens, and mobile telephones
    - » some characteristics of real-time systems
    - » restrictions on size, memory, and power
    - » PalmOS, Windows CE(Consumer Electronics)

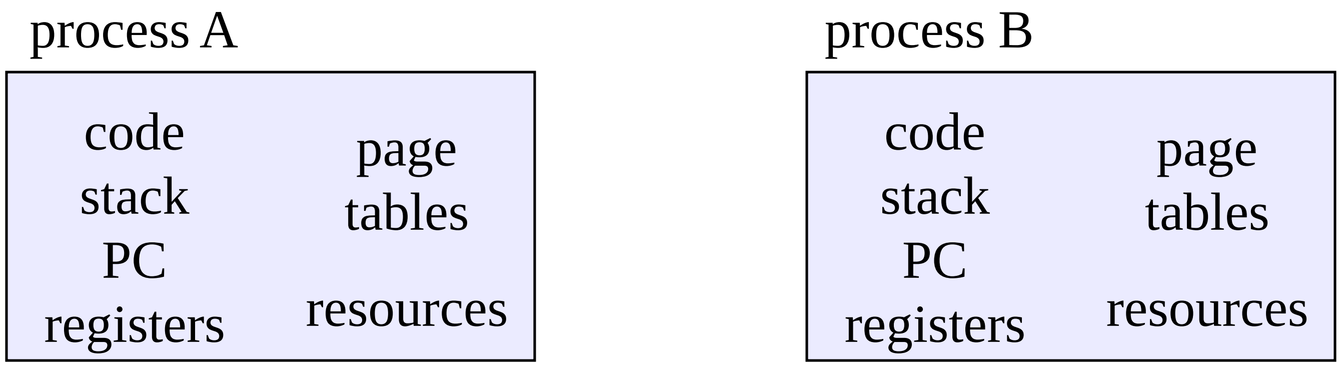
- **Smart card operating systems**
  - Run on smart cards, which are credit card-sized devices containing a CPU chip
  - Very severe processing power and memory constraints
  - Single function(e.g. electronic payments) or multiple functions on the same smart card
  - Some smart cards have JVM on which multiple applets run at the same time.

Operating System Concepts

- Processes
- Address spaces
- Files
- Input/Output
- Protection
- The shell
- Ontogeny recapitulates phylogeny
  - Large memories
  - Protection hardware
  - Disks
  - Virtual memory

Processes

- Note that a program is totally passive
  - just bytes on a disk that contain instructions to be run
- A process is an instance of a program being executed
  - at any instance, there may be many processes running copies of the same program (e.g. an editor); each process is separate and (usually) independent
  - Linux: `ps -auwwx` to list all processes



Process Management

- An OS executes many kinds of activities:
  - users' programs
  - batch jobs or scripts
  - system programs
    - » print spoolers, name servers, file servers, network daemons, …
- **Each of these activities is encapsulated in a process**
  - **a process includes the execution context**
    - » **PC, registers, VM, OS resources (e.g. open files), etc…**
    - » **plus the program itself (code and data)**
  - **the OS's process module manages these processes**
    - » **creation, destruction, scheduling, …**

Process operations

- **The OS provides the following kinds operations on processes (I.e. the process abstraction interface):**
  - create a process
  - delete a process
  - suspend a process
  - resume a process
  - clone a process
  - inter-process communication
  - inter-process synchronization
  - create/delete a child process (subprocess)

Processes

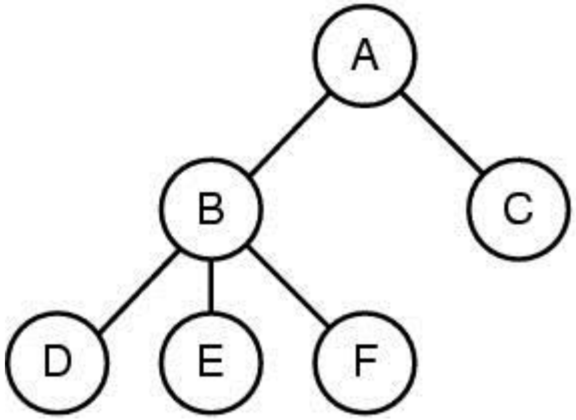


Figure 1-13. A process tree. Process A created two child processes, B and C. Process B created three child processes, D, E, and F.

Memory management

- The primary memory (or RAM) is the directly accessed storage for the CPU
  - programs must be stored in memory to execute
  - memory access is fast (e.g. 60 ns to load/store)
    - » but memory doesn't survive power failures
- OS must:
  - allocate memory space for programs (explicitly and implicitly)
  - deallocate space when needed by rest of system
  - Address space management – virtual memory systems
    - » Address space – set of addresses a process may reference
  - maintain mappings from physical to virtual memory
    - » through page tables
  - decide how much memory to allocate to each process
    - » a policy decision
  - decide when to remove a process from memory
    - » also policy

File Systems

- Secondary storage devices are crude and awkward
  - e.g. “write 4096 byte block to sector 12”
- File system: a convenient abstraction of disks and I/Os
  - defines logical objects like files and directories
    - » hides details about where on disk files live
  - as well as operations on objects like read and write
    - » read/write byte ranges instead of blocks
- A file is the basic long-term storage unit
  - file = named collection of persistent information
- A directory is just a special kind of file
  - directory = named file that contains names of other files and metadata about those files (e.g. file size)

File system operations

- The file system interface defines standard operations:
  - file (or directory) creation and deletion
  - manipulation of files and directories (read, write, extend, rename, protect)
  - copy
  - lock
- File systems also provide higher level services
  - accounting and quotas
  - backup
  - (sometimes) indexing or search
  - (sometimes) file versioning

Files (1)

- Path name
- File descriptor
- Root, working directory

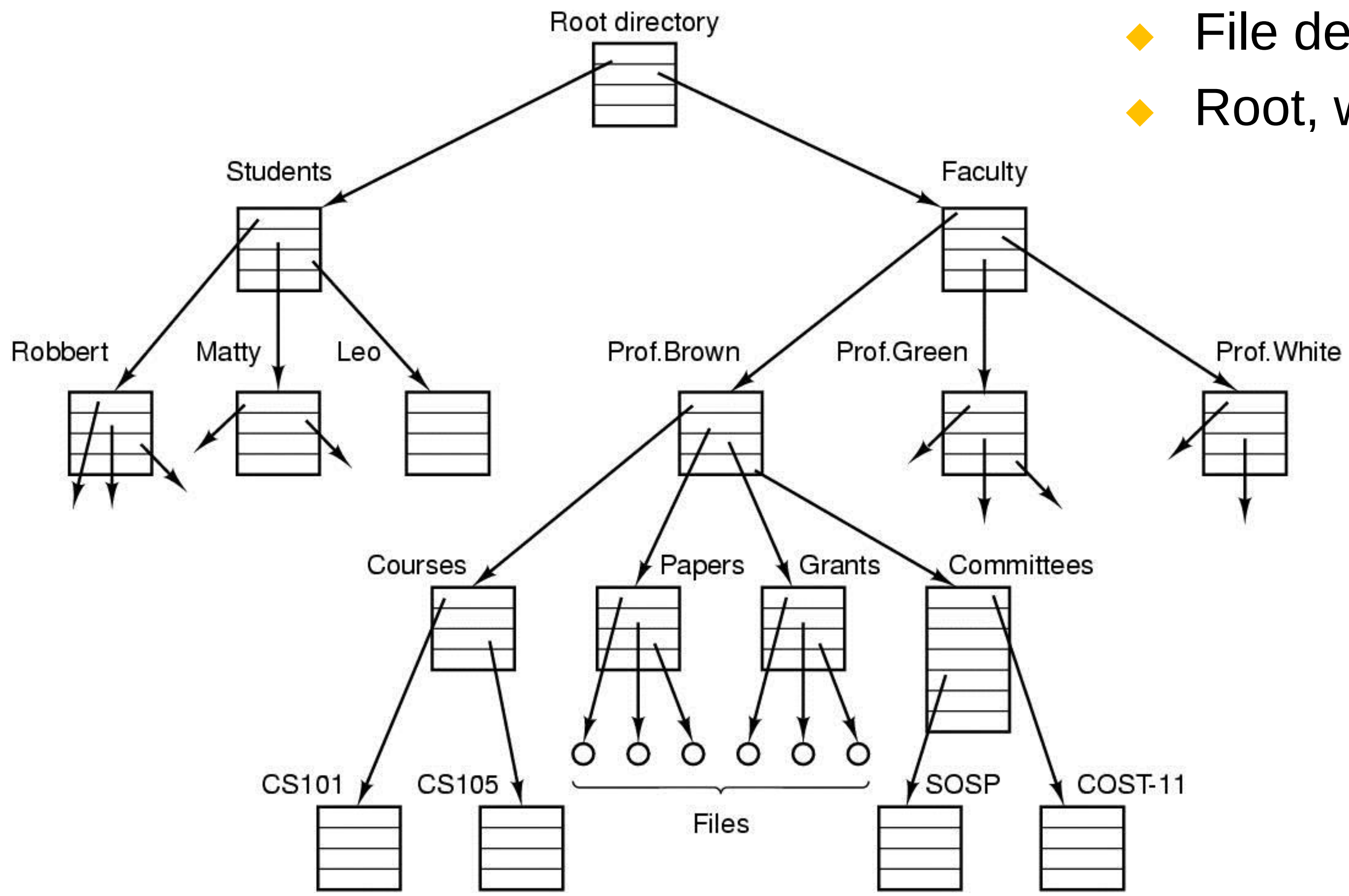


Figure 1-14. A file system for a university department.

Files (2)

- **Root file system**
- **mount**

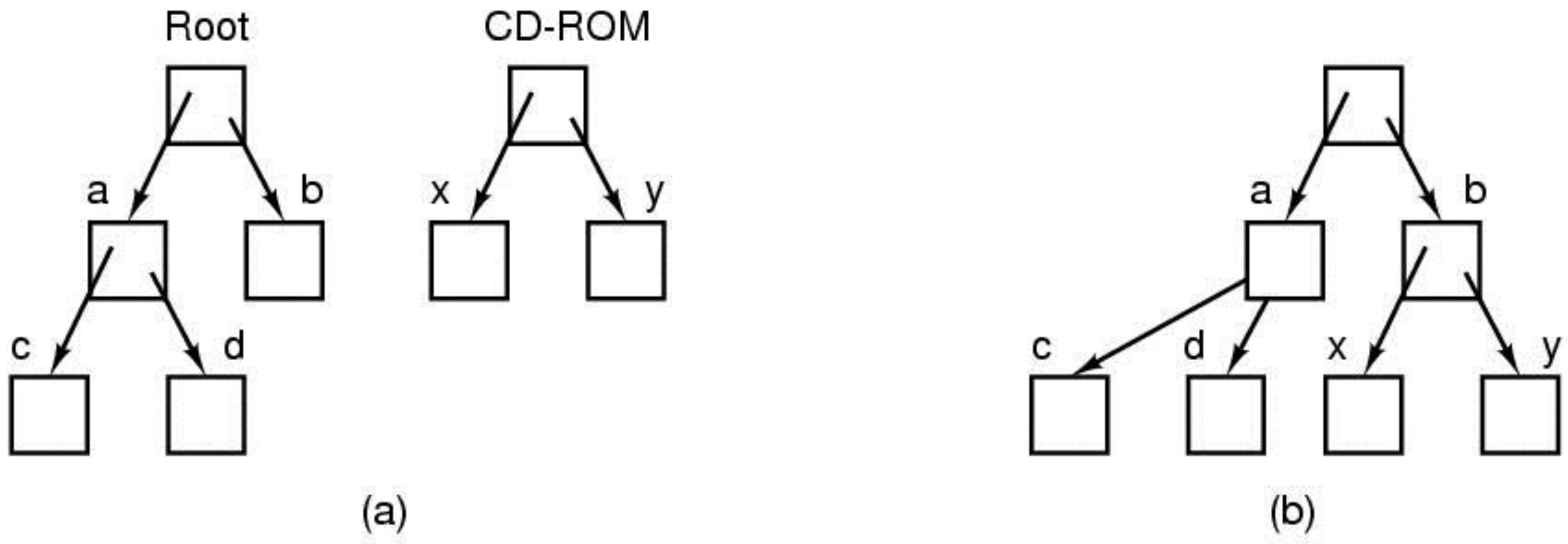


Figure 1-15. (a) Before mounting, the files on the CD-ROM are not accessible. (b) After mounting, they are part of the file hierarchy.

Files (3)

- Special file
  - Block special files
  - Character special files

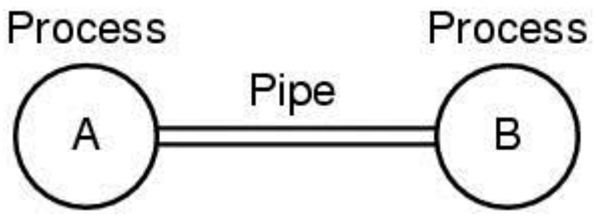


Figure 1-16. Two processes connected by a pipe.

I/O

- A big chunk of the OS kernel deals with I/O
  - hundreds of thousands of lines in NT
- The OS provides a standard interface between programs (user or system) and devices
  - file system (disk), sockets (network), frame buffer (video)
- Device drivers are the routines that interact with specific device types
  - encapsulates device-specific knowledge
  - e.g., how to initialize a device, how to request I/O, how to handle interrupts or errors
  - examples: SCSI device drivers, Ethernet card drivers, video card drivers, sound card drivers, …

Command Interpreter (shell)

- a particular program that handles the interpretation of users' commands and helps to manage processes
  - user input may be from keyboard (command-line interface), from script files, or from the mouse (GUIs)
  - allows users to launch and control new programs
- on some systems, command interpreter may be a standard part of the OS (e.g. MSDOS, Apple II)
- on others, it's just a non-privileged process that provides an interface to the user
  - e.g. bash/csh/tcsh/zsh on UNIX
- on others, there may be no command language
  - e.g. Mac OS

Protected Instructions

- some instructions are restricted to the OS
  - known as protected or privileged instructions
- e.g., only the OS can:
  - directly access I/O devices (disks, network cards)
    - » why?
  - manipulate memory state management
    - » page table pointers, TLB loads, etc.
    - » why?
  - manipulate special ‘mode bits’
    - » interrupt priority level
    - » why?
  - halt instruction
    - » why?

OS Protection

- So how does the processor know if a protected instruction should be executed?
  - the architecture must support at least two modes of operation: kernel mode and user mode
    - » VAX, x86 support 4 protection modes
  - mode is set by status bit in a protected processor register
    - » user programs execute in user mode
    - » OS executes in kernel mode (OS == kernel)
- Protected instructions can only be executed in the kernel mode
  - what happens if user mode executes a protected instruction?

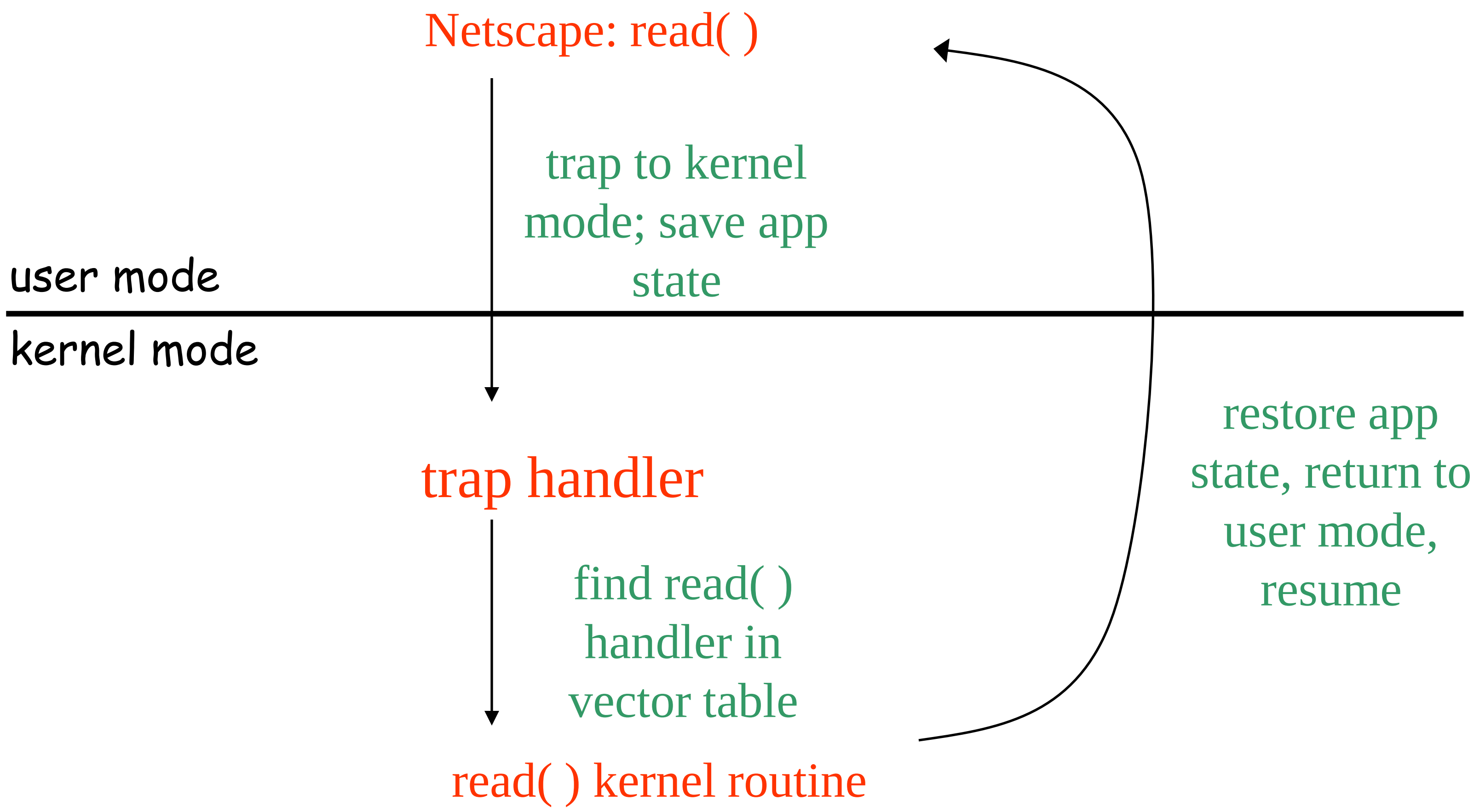
Crossing Protection Boundaries

- So how do user programs do something privileged?
  - e.g., how can you write to a disk if you can't do I/O instructions?
- User programs must call an OS procedure
  - OS defines a sequence of system calls
  - how does the user-mode to kernel-mode transition happen?
- There must be a system call instruction, which:
  - causes an exception (throws a software interrupt), which vectors to a kernel handler
  - passes a parameter indicating which system call to invoke
  - saves caller's state (regs, mode bit) so they can be restored
  - OS must verify caller's parameters (e.g. pointers)
  - must be a way to return to user mode once done

System Calls

- Interface that programs can use in order to obtain a variety of services provided by the operating system
- Most CPUs have two modes, kernel mode and user mode.
  - A bit in the PSW(Program Status Word) usually controls the mode.
  - When running in kernel mode, the CPU can execute every instruction in its instruction set and use every feature of the hardware
  - The operating system runs in kernel mode, giving it access to the complete hardware
  - User programs run in user mode, which permits only a subset of the instructions to be executed and a subset of features to be accessed.
  - Generally, all instructions involving I/O and memory protection are disallowed in user mode.
- To obtain services from the operating system, a user program must make a system call which traps into the kernel and invokes the operating system
- The Trap instruction switches from user mode to kernel mode and starts the operating system.

A Kernel Crossing Illustrated

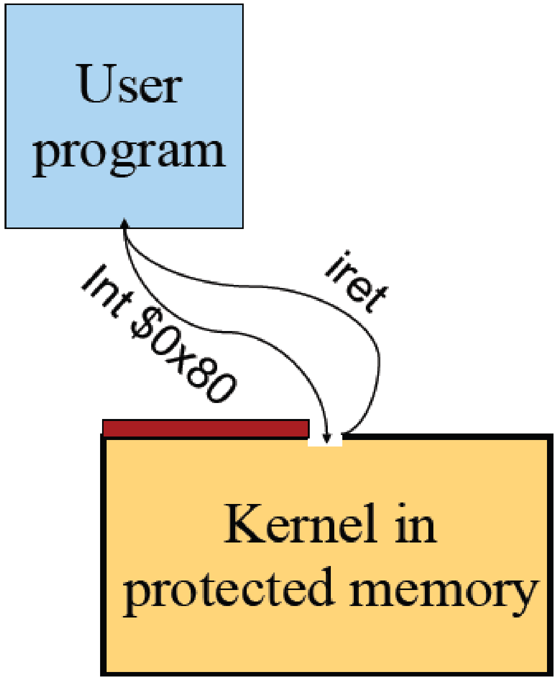

Netscape: read( )
trap to kernel mode; save app state
user mode
kernel mode
trap handler
find read( ) handler in vector table
read( ) kernel routine
restore app state, return to user mode, resume

Library Stubs for System Calls

- Example:

```
int read( int fd, char * buf, int size)
{
      move fd, buf, size to R1, R2,
      R3
      move READ to R0
      int $0x80
      move result to Rresult
}
```

Linux: 80
NT: 2E



System Calls

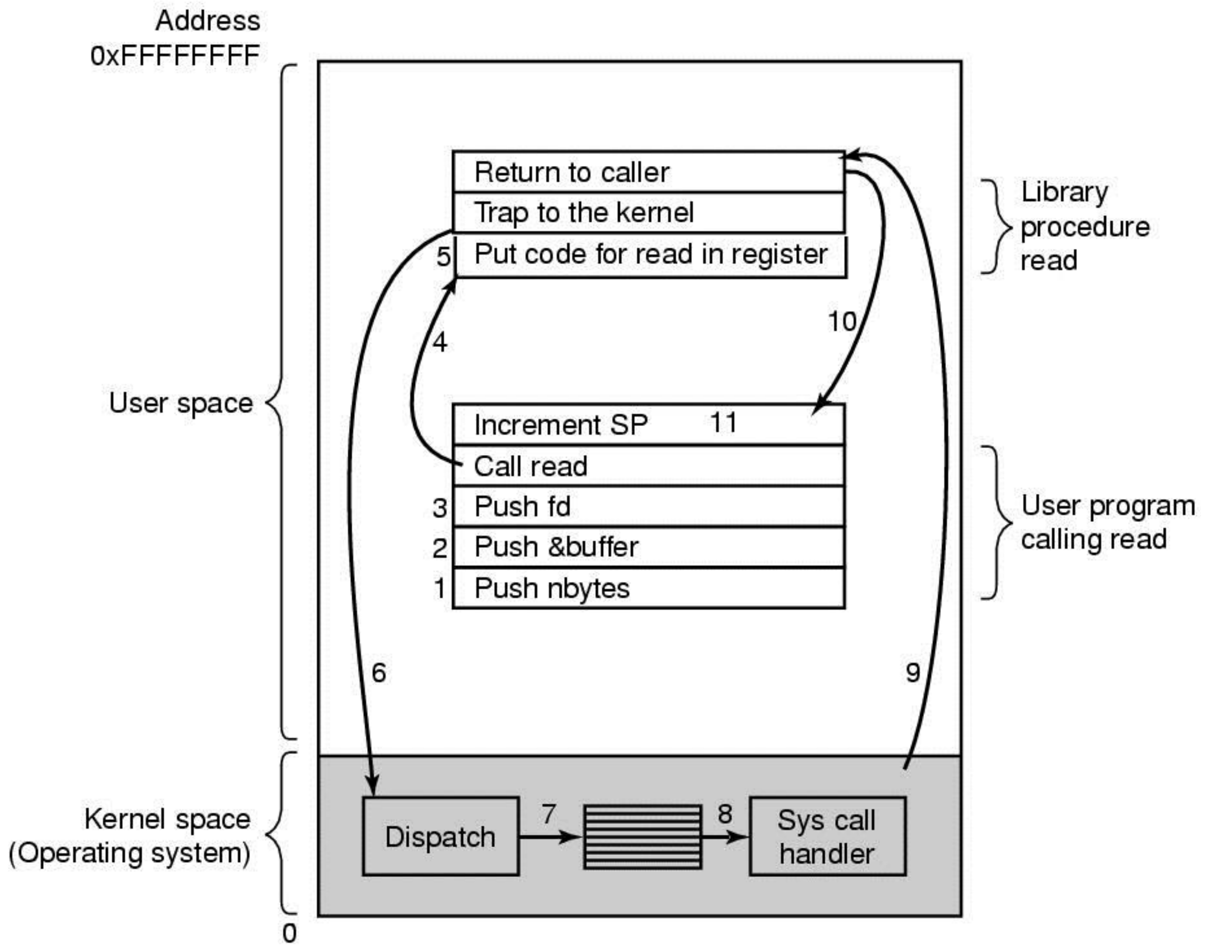


Figure 1-17. The 11 steps in making the system call read(fd, buffer, nbytes).

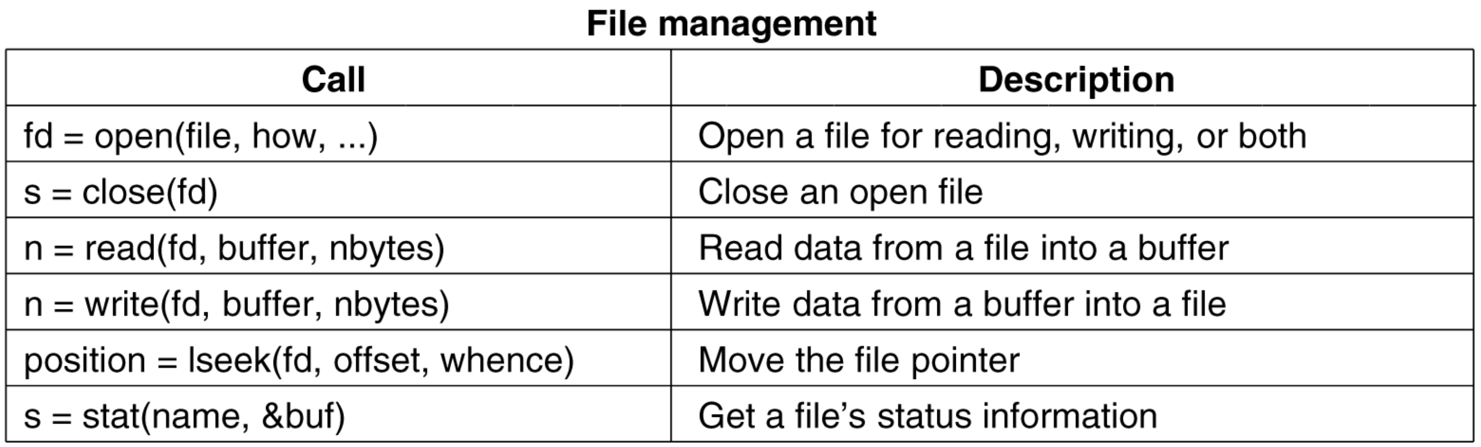
System Calls for Process Management

Process management

Call	Description
pid = fork( )	Create a child process identical to the parent
pid = waitpid(pid, &statloc, options)	Wait for a child to terminate
s = execve(name, argv, environp)	Replace a process' core image
exit(status)	Terminate process execution and return status

Figure 1-18. Some of the major POSIX system calls.

System Calls for File Management (1)



File management

Call	Description
fd = open(file, how, ...)	Open a file for reading, writing, or both
s = close(fd)	Close an open file
n = read(fd, buffer, nbytes)	Read data from a file into a buffer
n = write(fd, buffer, nbytes)	Write data from a buffer into a file
position = lseek(fd, offset, whence)	Move the file pointer
s = stat(name, &buf)	Get a file's status information

Figure 1-18. Some of the major POSIX system calls.

System Calls for File Management (2)

Call	Description
s = mkdir(name, mode)	Create a new directory
s = rmdir(name)	Remove an empty directory
s = link(name1, name2)	Create a new entry, name2, pointing to name1
s = unlink(name)	Remove a directory entry
s = mount(special, name, flag)	Mount a file system
s = umount(special)	Unmount a file system

Figure 1-18. Some of the major POSIX system calls.

Miscellaneous System Calls

Call	Description
s = chdir(dirname)	Change the working directory
s = chmod(name, mode)	Change a file's protection bits
s = kill(pid, signal)	Send a signal to a process
seconds = time(&seconds)	Get the elapsed time since Jan. 1, 1970

Figure 1-18. Some of the major POSIX system calls.

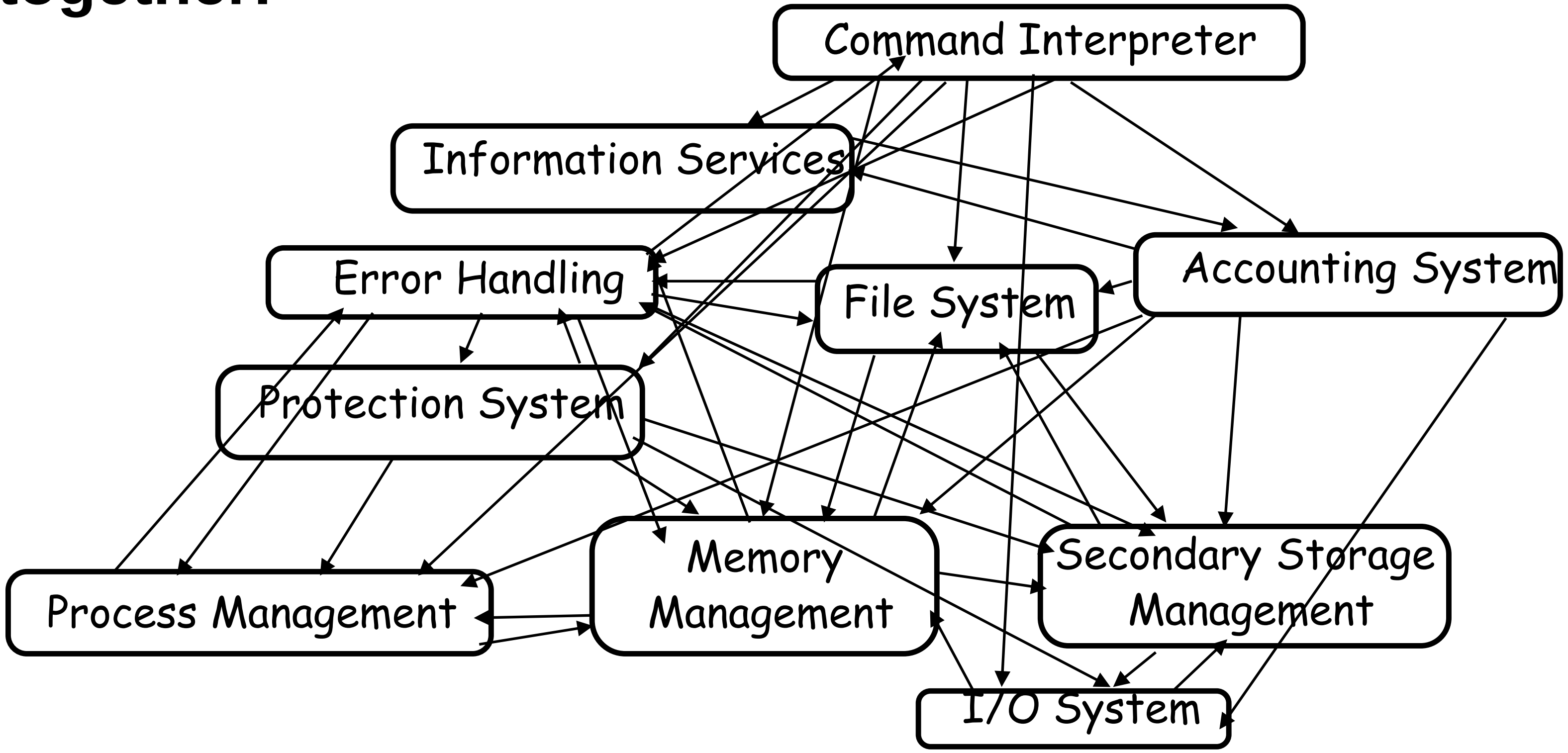
A Simple Shell

```
#define TRUE 1

while (TRUE) {                                      /* repeat forever */
     type_prompt( );                                /* display prompt on the screen */
     read_command(command, parameters);             /* read input from terminal */

     if (fork( ) != 0) {                            /* fork off child process */
          /* Parent code. */
          waitpid(−1, &status, 0);                  /* wait for child to exit */
     } else {
          /* Child code. */
          execve(command, parameters, 0);           /* execute command */
     }
}
```

Figure 1-19. A stripped-down shell.

OS Structure

- **It's not always clear how to stitch OS modules together:**

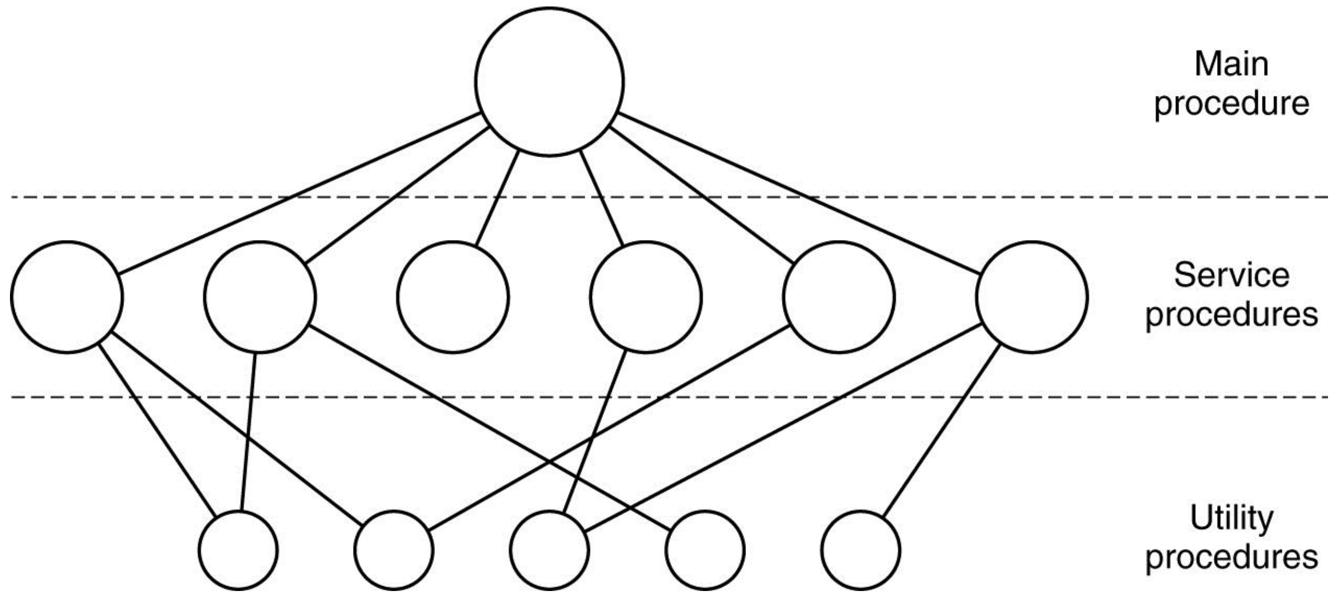


OS Structure

- An OS consists of all of these components, plus:
  - many other components
  - system programs (privileged and non-privileged)
    - » e.g. bootstrap code, the init program, …
- Major issue:
  - how do we organize all this?
  - what are all of the code modules, and where do they exist?
  - how do they cooperate?
- Massive software engineering and design problem
  - design a large, complex program that:
    - » performs well, is reliable, is extensible, is backwards compatible, …

Monolithic kernels – basic structure:

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



Monolithic kernels

- **Major advantage:**
  - cost of module interactions is low (procedure call)
  - Good performance
- **Disadvantages:**
  - hard to understand
  - hard to modify
  - unreliable (no isolation between system modules)
  - hard to maintain
- **What is the alternative?**
  - find a way to organize the OS in order to simplify its design and implementation

Layering

- The traditional approach is layering
  - implement OS as a set of layers
  - each layer acts as a 'virtual machine' to the layer above
- The first description of this approach was Dijkstra's THE system
  - layer 0: hardware
  - layer 1: CPU scheduling
  - layer 2: memory management (sees virtual processors)
  - layer 3: console device (sees VM segments)
  - layer 4: I/O device buffering (sees 'virtual console')
  - layer 5: user programs (sees 'virtual I/O drivers')
- Each layer can be tested and verified independently

Microkernels

- Popular in the late 80's, early 90's
  - recent resurgence of popularity for small devices
- Goal:
  - minimize what goes in kernel
  - organize rest of OS as user-level processes
- This results in:
  - better reliability (isolation between components)
  - ease of extension and customization
  - poor performance (user/kernel boundary crossings)
- First microkernel system was Hydra (CMU, 1970)
  - follow-ons: Mach (CMU), Chorus (French UNIX-like OS), and in some ways NT (Microsoft), OSX (Apple)

Microkernels

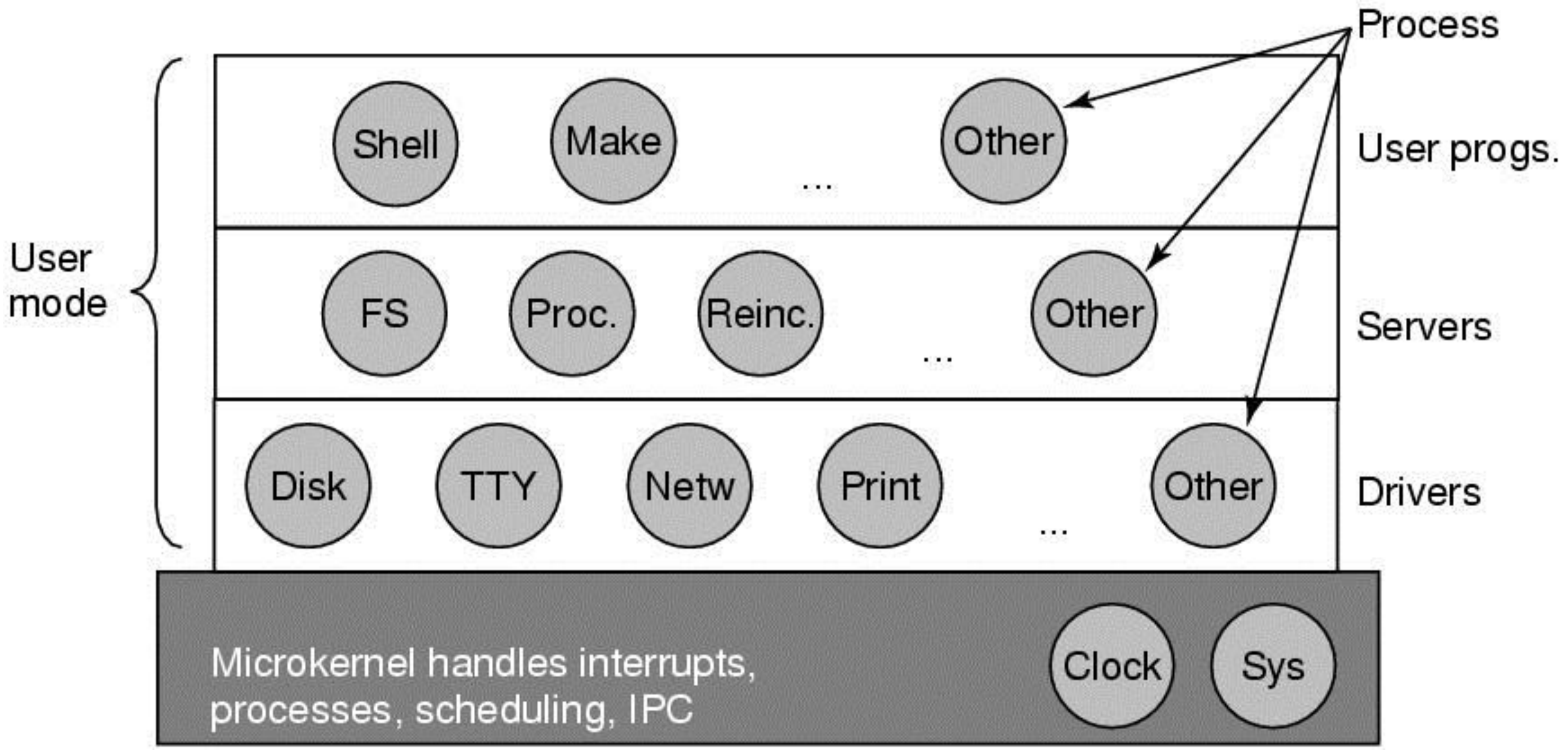
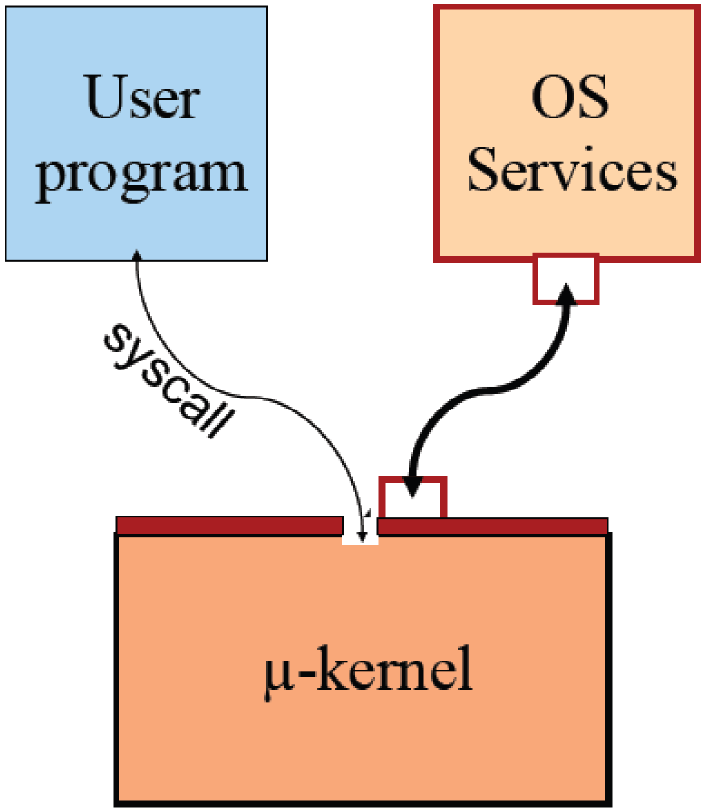


Figure 1-26. Structure of the MINIX 3 system.

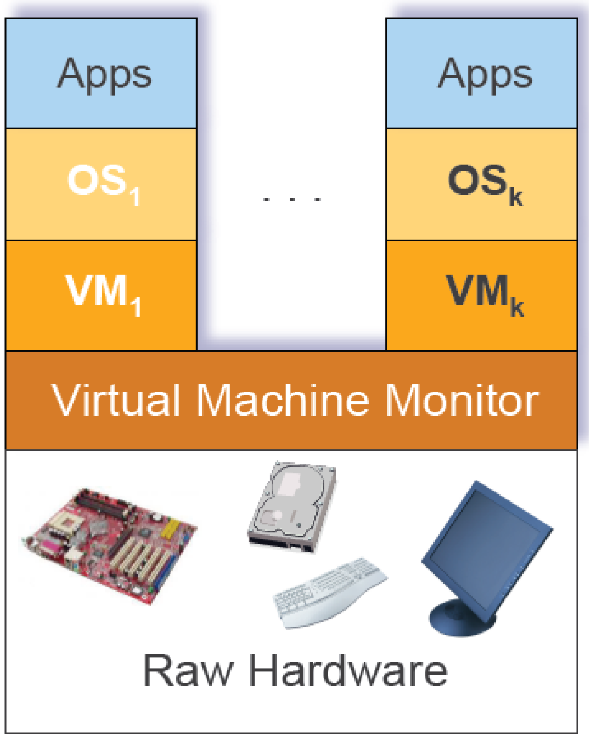
Microkernel

- Put less in kernel mode: only small part of OS; reduce kernel bugs
- Services are regular processes; one file system crashing doesn't crash full system; can't corrupt kernel memory
- µ-kernel gets svcs on behalf of users by messaging with service processes
- Pros
  - Flexibility, Fault isolation and reliability
- Cons
  - Inefficient (boundary crossings)
  - Inconvenient to share data between kernel and services



Virtual Machines

- Separate out multiprogramming from abstraction; VMM provides former
- Virtual machine monitor
  - Virtualize hardware, but expose it as multiple instances of 'raw' hw
  - Run several OSes, one on each set
  - Examples
    - IBM VM/370
    - Java VM
    - VMWare, Xen
- What would you use virtual machine for?



Virtual Machines Rediscovered

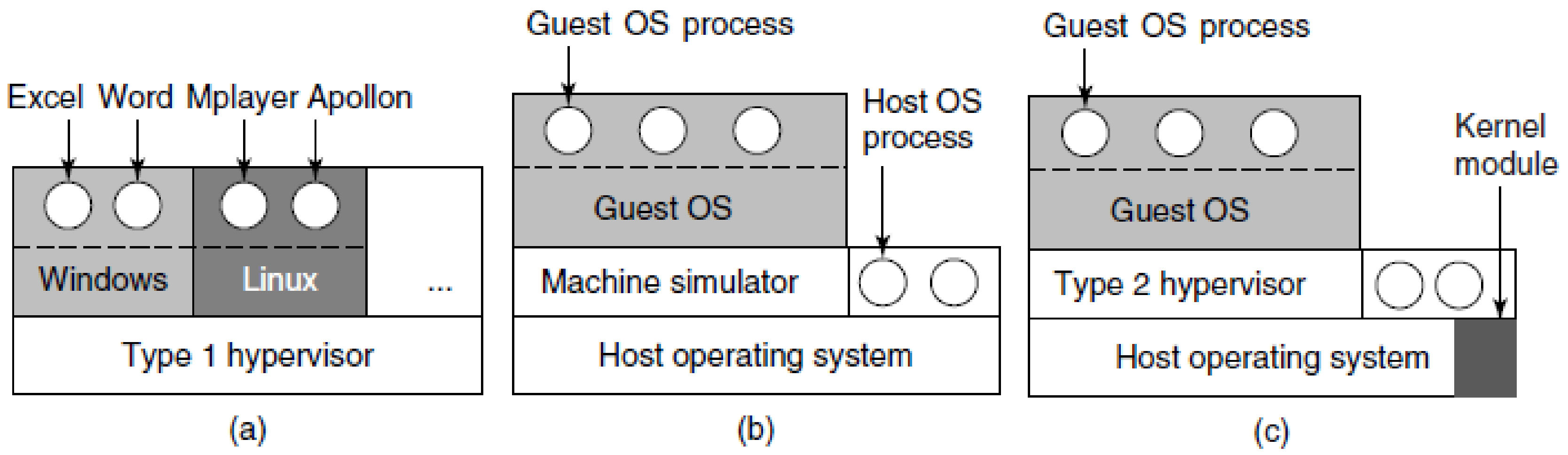


Figure 1-29. (a) A type 1 hypervisor. (b) A pure type 2 hypervisor. (c) A practical type 2 hypervisor.