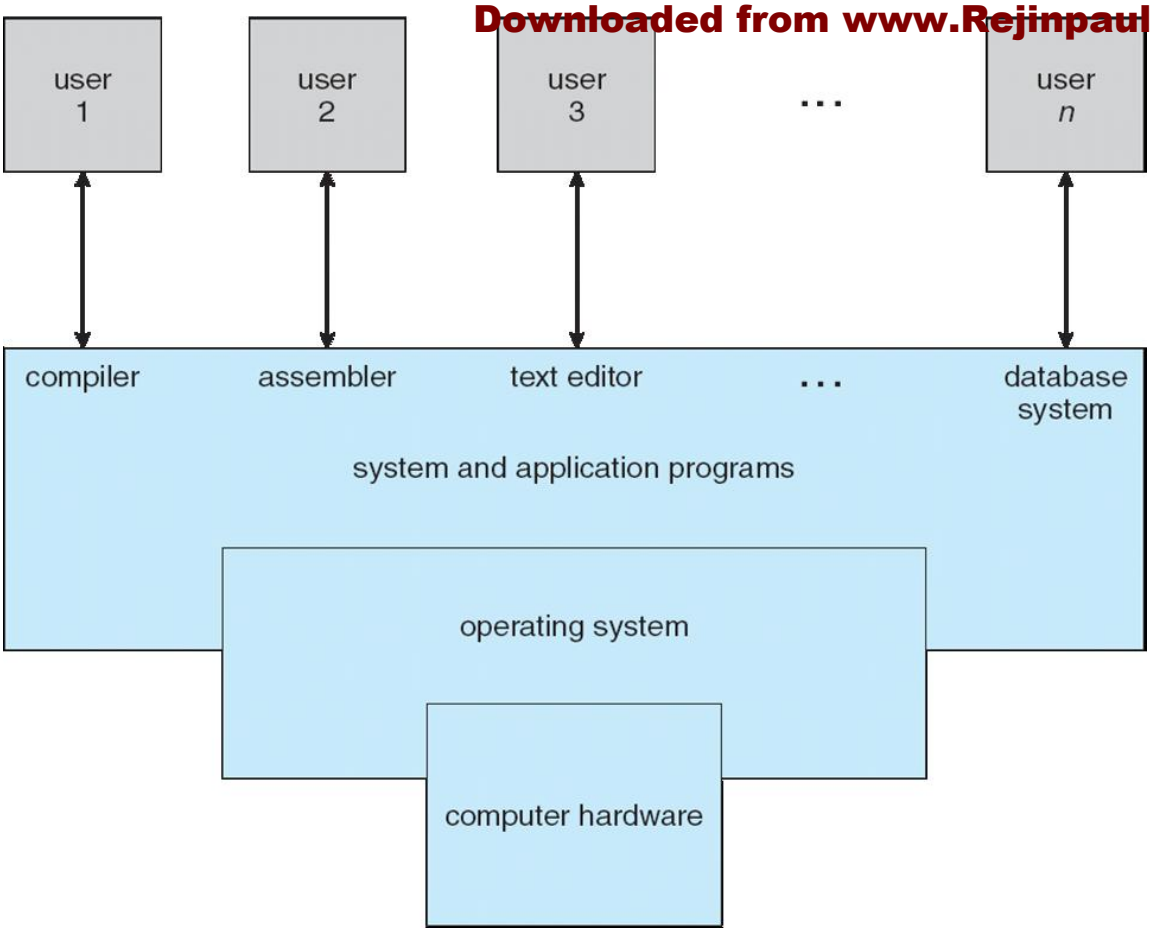
OS is a program that acts as an intermediary between a user of a computer and the computer hardware

Operating system goals:

- Execute user programs and make solving user problems easier
- Make the computer system convenient to use
- Use the computer hardware in an efficient manner

COMPUTER SYSTEM ARCHITECTURE

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



- Depends on the point of view
- Users want convenience, **ease of use** and **good performance**
    - Don't care about **resource utilization**
- But shared computer such as **mainframe** or **minicomputer** must keep all users happy
- Users of dedicate systems such as **workstations** have dedicated resources but frequently use shared resources from **servers**
- Handheld computers are resource poor, optimized for usability and battery life
- Some computers have little or no user interface, such as embedded computers in devices and automobiles
- OS is a **resource allocator**
    - Manages all resources
    - Decides between conflicting requests for efficient and fair resource use
- OS is a **control program**
    - Controls execution of programs to prevent errors and improper use of the computer
- No universally accepted definition

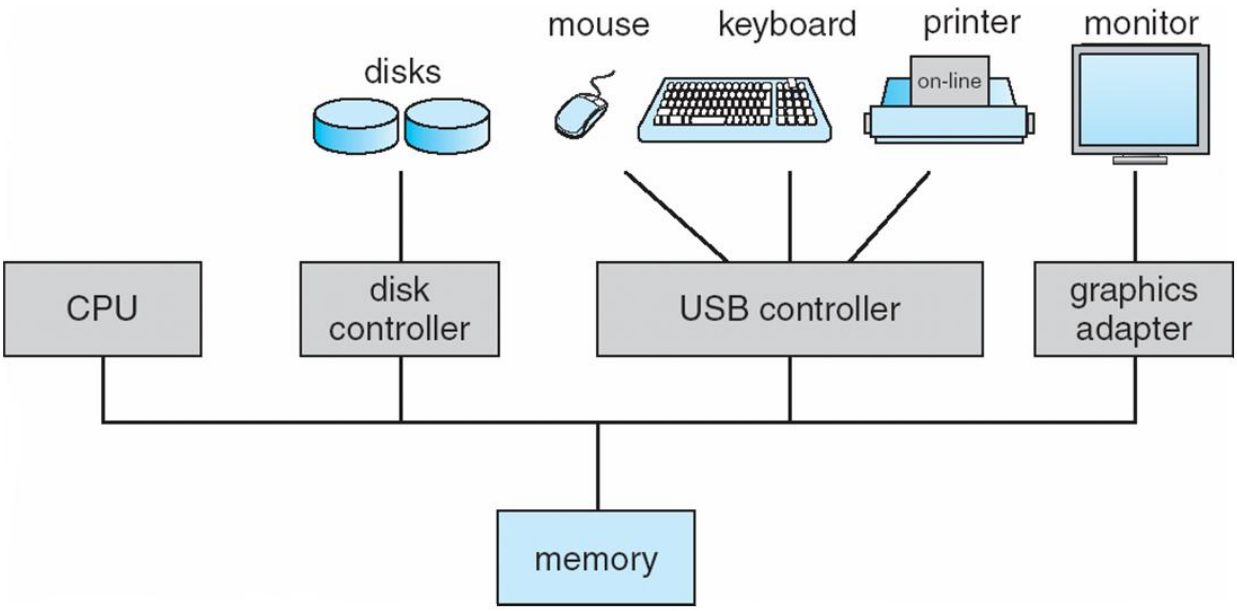
- "Everything a vendor ships when you order an operating system" is a good approximation
    - But varies wildly
- "The one program running at all times on the computer" is the **kernel**.
- Everything else is either
    - a system program (ships with the operating system) , or
    - an application program.

COMPUTER STARTUP

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

COMPUTER-SYSTEM OPERATION

- One or more CPUs, device controllers connect through common bus providing access to shared memory



- I/O devices and the CPU can execute concurrently
- Each device controller is in charge of a particular device type
- Each device controller has a local buffer
- CPU moves data from/to main memory to/from local buffers
- I/O is from the device to local buffer of controller

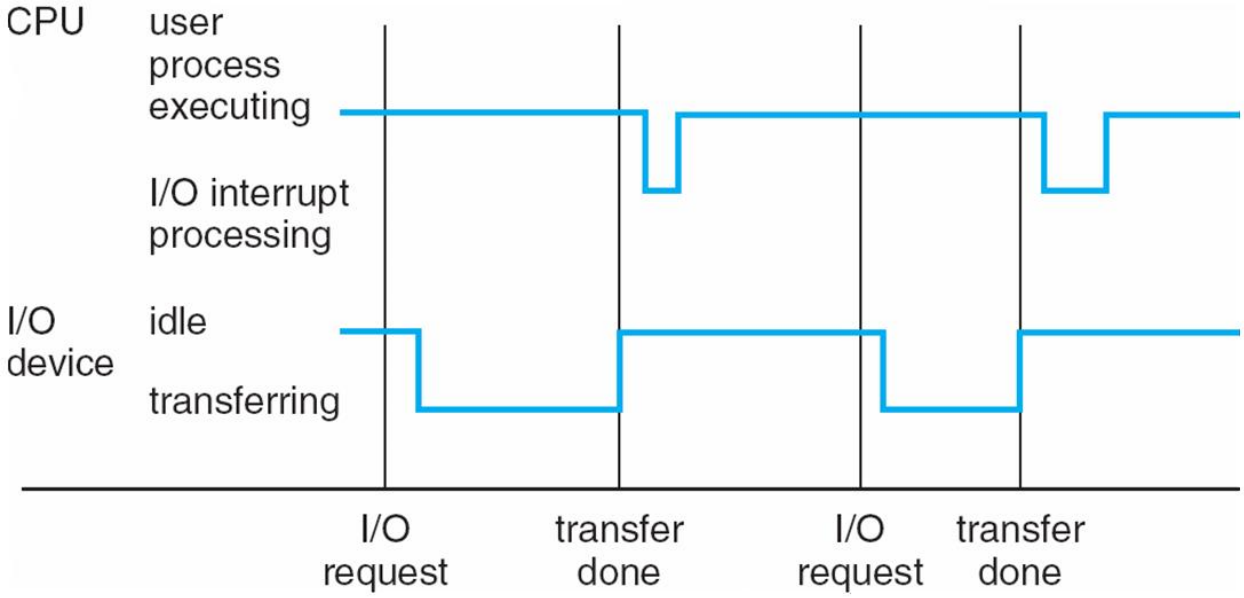
INTERRUPTS

- Device controller informs CPU that it has finished its operation by causing an interrupt
- Interrupt transfers control to the interrupt service routine generally, through the **interrupt vector**, which contains the addresses of all the service routines
- Interrupt architecture must save the address of the interrupted instruction
- A **trap** or **exception** is a software-generated interrupt caused either by an error or a user request
- An operating system is **interrupt driven**

INTERRUPT HANDLING

- The OS preserves the state of the CPU by storing registers and the program counter
- Determines which type of interrupt has occurred:
  - **polling**
    - The interrupt controller polls (send a signal out to) each device to determine which one made the request
  - **vectored** interrupt system
- Separate segments of code determine what action should be taken for each type of interrupt

INTERRUPT TIMELINE



I/O STRUCTURE

- Synchronous (blocking) I/O
    - Waiting for I/O to complete
    - Easy to program, not always efficient
    - Wait instruction idles the CPU until the next interrupt
    - At most one I/O request is outstanding at a time
        - no simultaneous I/O processing
- Asynchronous (nonblocking) I/O
    - After I/O starts, control returns to user program without waiting for I/O completion
    - Harder to program, more efficient
    - **System call** – request to the OS to allow user to wait for I/O completion (polling periodically to check busy/done)
    - **Device-status table** contains entry for each I/O device indicating its type, address, and state

STORAGE HIERARCHY

The basic unit of computer storage is the **bit**. A bit can contain one of two values, 0 and 1. All other storage in a computer is based on collections of bits. Given enough bits, it is amazing how many things a computer can represent: numbers, letters, images, movies, sounds, documents, and programs, to name a few. A **byte** is 8 bits, and on most computers it is the smallest convenient chunk of storage. For example, most computers don't have an instruction to move a bit but do have one to move a byte. A less common term is **word**, which is a given computer architecture's native unit of data. A word is made up of one or more bytes. For example, a computer that has 64-bit registers and 64-bit memory addressing typically has 64-bit (8-byte) words. A computer executes many operations in its native word size rather than a byte at a time.

Computer storage, along with most computer throughput, is generally measured and manipulated in bytes and collections of bytes.

A **kilobyte**, or **KB**, is 1,024 bytes

a **megabyte**, or **MB**, is $1{,}024^2$ bytes

a **gigabyte**, or **GB**, is $1{,}024^3$ bytes

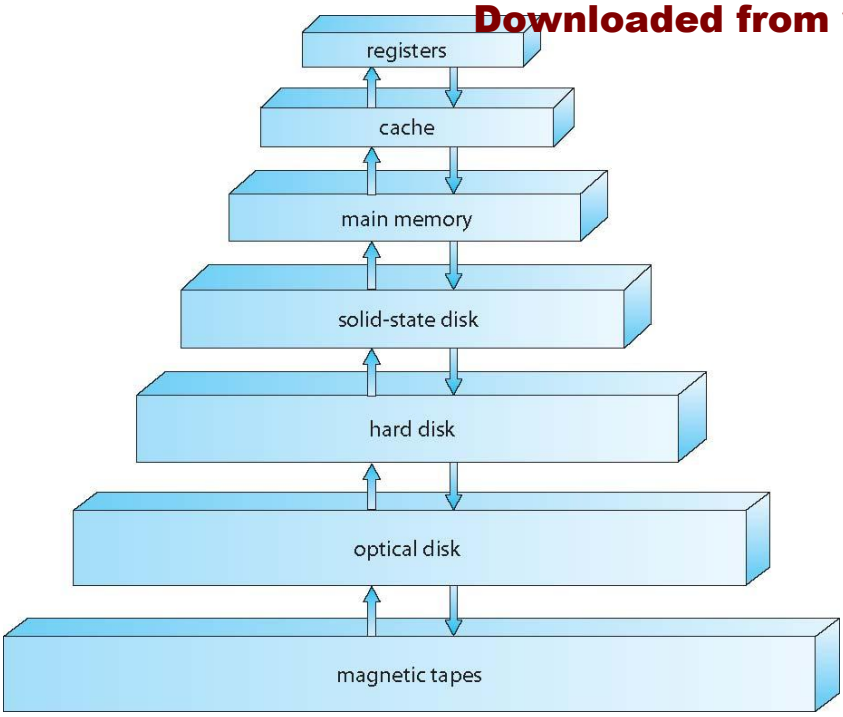
a **terabyte**, or **TB**, is $1{,}024^4$ bytes

a **petabyte**, or **PB**, is $1{,}024^5$ bytes

Computer manufacturers often round off these numbers and say that a megabyte is 1 million bytes and a gigabyte is 1 billion bytes. Networking measurements are an exception to this general rule; they are given in bits (because networks move data a bit at a time).

STORAGE STRUCTURE

- Main memory – only large storage media that the CPU can access directly
    - **Random access**
    - Typically **volatile**
- Secondary storage – extension of main memory that provides large **nonvolatile** storage capacity
- Hard disks – rigid metal or glass platters covered with magnetic recording material
    - Disk surface is logically divided into **tracks**, which are subdivided into **sectors**
    - The **disk controller** determines the logical interaction between the device and the computer
- **Solid-state disks** – faster than hard disks, nonvolatile
    - Various technologies
    - Becoming more popular
- Storage systems organized in hierarchy
    - Speed
    - Cost (per byte of storage)
    - Volatility
- **Device Driver** for each device controller to manage I/O
    - Provides uniform interface between controller and kernel



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

CACHING

- Important principle
- Performed at many levels in a computer
    - in hardware,
    - operating system,
    - software
- Information in use copied from slower to faster storage temporarily
    - Efficiency
- Faster storage (cache) checked first to determine if information is there

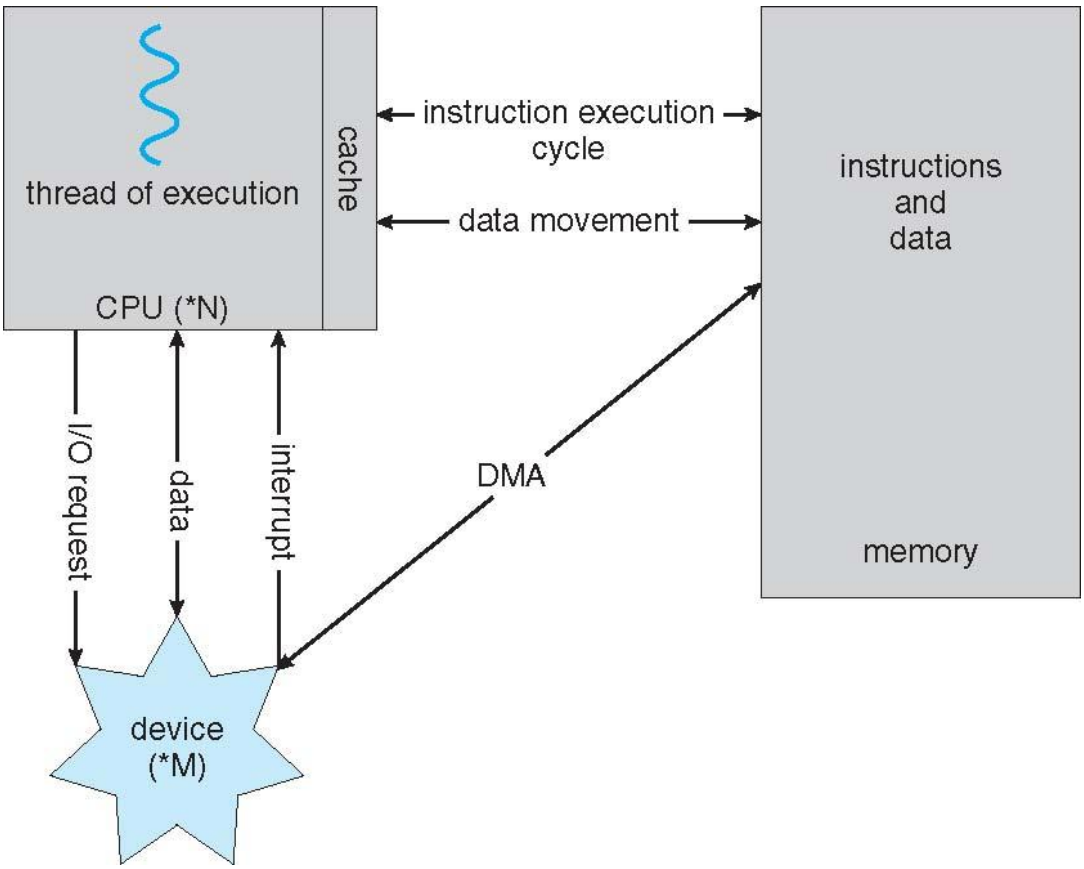
- If it is, information used directly from the cache (fast)
- If not, data copied to cache and used there

- Cache smaller than storage being cached
  - Cache management important design problem
  - Cache size and replacement policy

DIRECT MEMORY ACCESS STRUCTURE

- Typically used for I/O devices that generate data in blocks, or generate data fast
- Device controller transfers blocks of data from buffer storage directly to main memory without CPU intervention
- Only one interrupt is generated per block, rather than the one interrupt per byte

HOW A MODERN COMPUTER SYSTEM WORKS

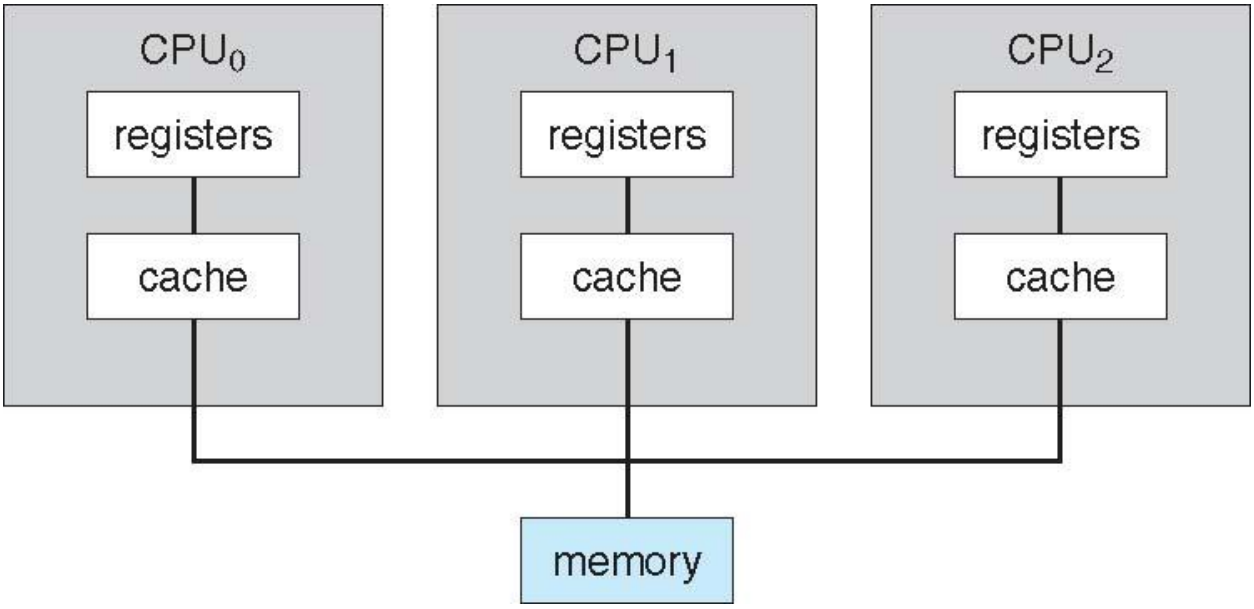


TYPES OF SYSTEMS

- Most systems use a single general-purpose processor
  - Most systems have special-purpose processors as well
- **Multiprocessors** systems growing in use and importance

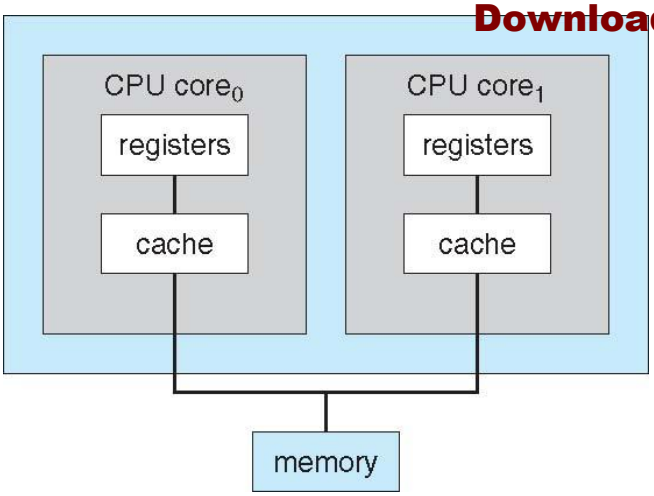
- Also known as **parallel systems, tightly-coupled systems**
- Advantages include:
    1. **Increased throughput**
    2. **Economy of scale**
    3. **Increased reliability** – graceful degradation or fault tolerance
- Two types:
    - **Asymmetric Multiprocessing** – each processor is assigned a specific task
    - **Symmetric Multiprocessing** – each processor performs all tasks

SYMMETRIC MULTIPROCESSING ARCHITECTURE

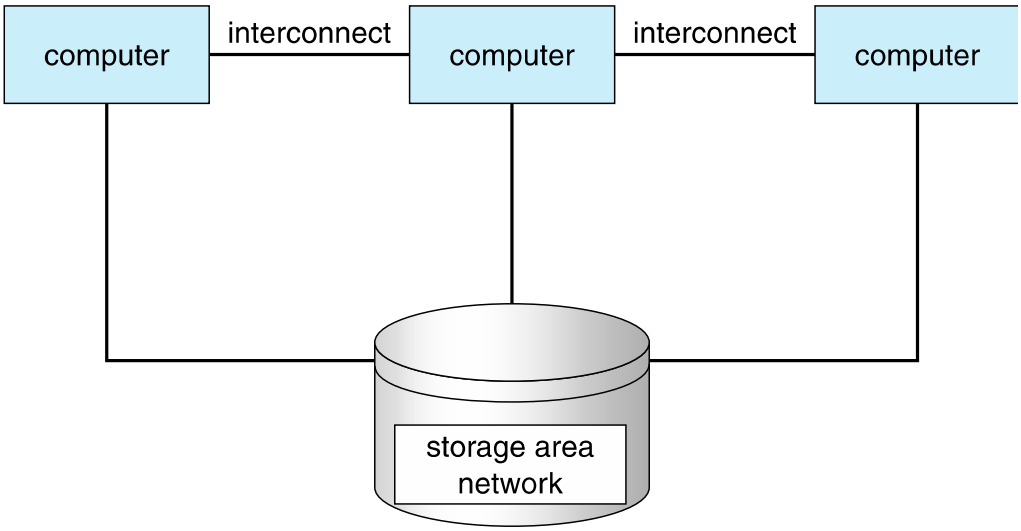


DUAL CORE DESIGN

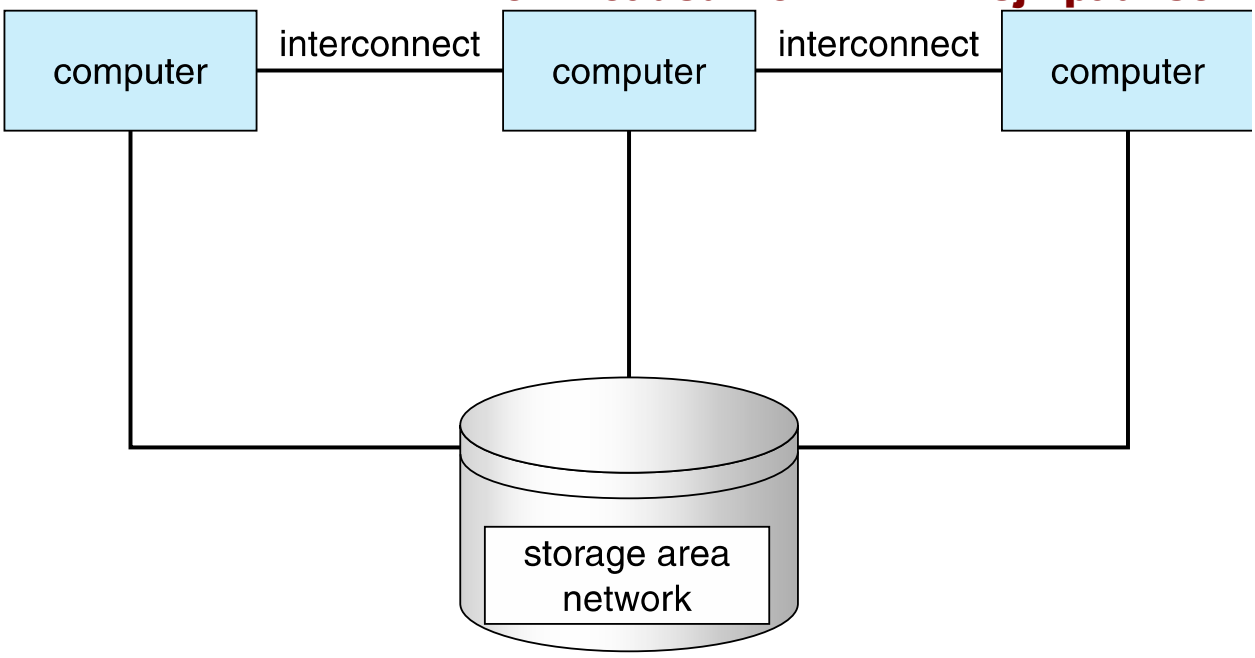
- **Multicore**
    - Several cores on a single chip
    - On chip communication is faster than between-chip
    - Less power used



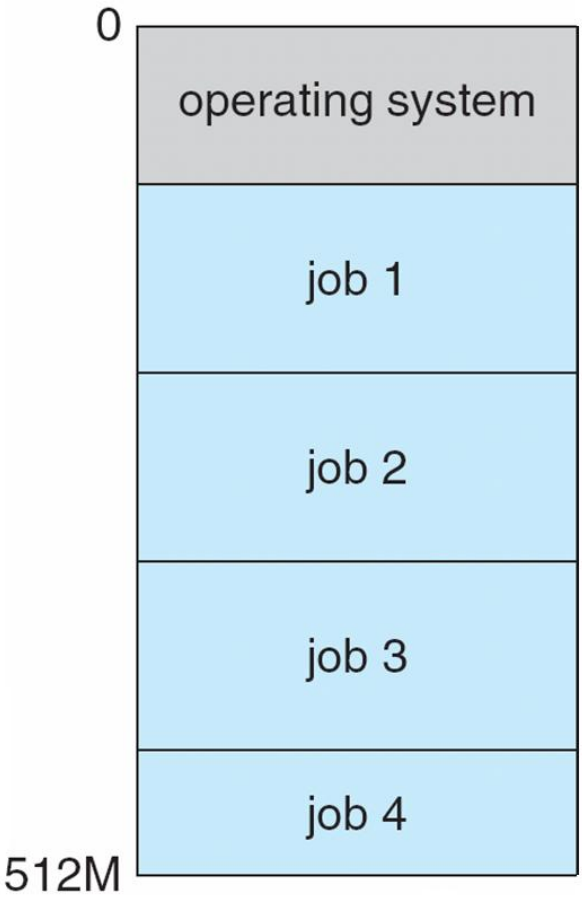
CLUSTERED SYSTEMS



- Like multiprocessor systems, but multiple systems working together
- Provides a **high-availability** service which survives failures
- **Asymmetric clustering** has one machine in hot-standby mode
- **Symmetric clustering** has multiple nodes running applications, monitoring each other
- Some clusters are for **high-performance computing (HPC)**
- Applications must be written to use **parallelization**



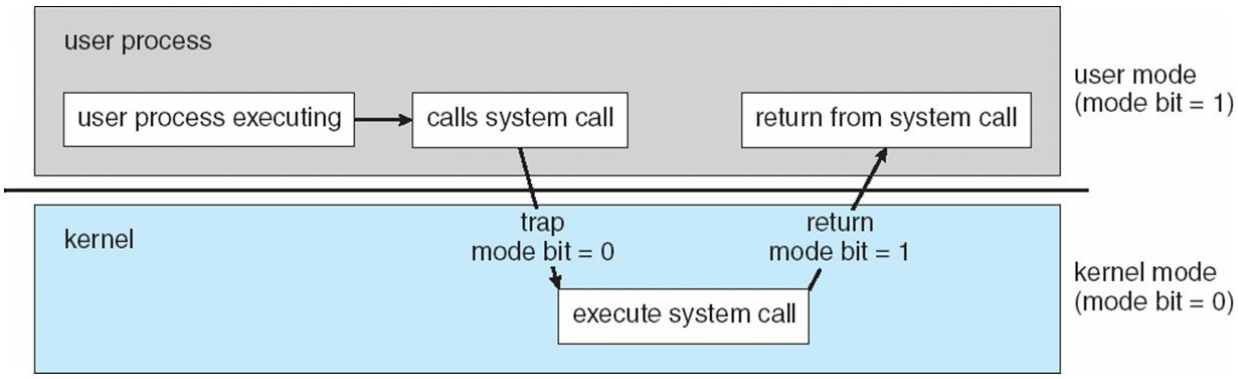
MEMORY LAYOUT FOR MULTIPROGRAMMED SYSTEMS



OPERATING SYSTEM OPERATIONS

- **Interrupt driven** (hardware and software)
    - Hardware interrupt by one of the devices

- Software interrupt (**exception** or **trap**):
  - Software error (e.g., division by zero)
  - Request for operating system service
  - Other process problems include infinite loop, processes modifying each other or the operating system



- **Dual-mode** operation allows OS to protect itself and other system components
- **User mode** and **kernel mode**
- **Mode bit** provided by hardware
- Provides ability to distinguish when system is running user code or kernel code
- Some instructions designated as **privileged**, only executable in kernel mode
- System call changes mode to kernel, return from call resets it to user
- Timer to prevent infinite loop / process hogging resources
- Timer is set to interrupt the computer after some time period
- Keep a counter that is decremented by the physical clock
- Operating system set the counter (privileged instruction)
- When counter zero generate an interrupt
- Set up before scheduling process to
- regain control, or
- terminate program that exceeds allotted time

PROCESS MANAGEMENT

- A process is a program in execution. It is a unit of work within the system. Program is a ***passive entity***, process is an ***active entity***.
- Process needs resources to accomplish its task

- CPU, memory, I/O, files
- Initialization data

- Typically system has many processes, some user, some operating system running concurrently on one or more CPUs
  - Concurrency by multiplexing the CPUs among the processes / threads

ACTIVITIES

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

MEMORY MANAGEMENT

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

STORAGE MANAGEMENT

- OS provides uniform, logical view of information storage
  - Different devices, same view
  - Abstracts physical properties to logical storage unit - **file**
  - Each medium is controlled by device (i.e., disk drive, tape drive)
    - Varying properties include access speed, capacity, data-transfer rate, access method (sequential or random)
    - File-System management

- Files usually organized into directories
- Access control on most systems to determine who can access what
- OS activities include
  - Creating and deleting files and directories
  - Primitives to manipulate files and directories
  - Mapping files onto secondary storage
  - Backup files onto stable (non-volatile) storage media

MASS STORAGE MANAGEMENT

- Usually disks used to store
- data that does not fit in main memory, or
- data that must be kept for a “long” period of time
- Proper management is of central importance
- Entire speed of computer operation hinges on disk subsystem and its algorithms
- Disk is slow, its I/O is often a bottleneck
- OS activities
- Free-space management
- Storage allocation
- Disk scheduling
- Multitasking environments must be careful to use most recent value, no matter where it is stored in the storage hierarchy
- Multiprocessor environment must provide **cache coherency** in hardware such that all CPUs have the most recent value in their cache
- Distributed environment situation even more complex
- Several copies of a datum can exist

I/O SUBSYSTEM

- One purpose of OS is to hide peculiarities of hardware devices from the user
- I/O subsystem responsible for
  - Memory management of I/O including buffering (storing data temporarily while it is being transferred),

    - Caching (storing parts of data in faster storage for performance),
    - General device-driver interface
    - Drivers for specific hardware devices
- **Protection** – any mechanism for controlling access of processes or users to resources defined by the OS
- **Security** – defense of the system against internal and external attacks
    - Huge range, including denial-of-service, worms, viruses, identity theft, theft of service
- Systems generally first distinguish among users, to determine who can do what
    - Access control for users and groups

COMPUTING ENVIRONMENTS

TRADITIONAL

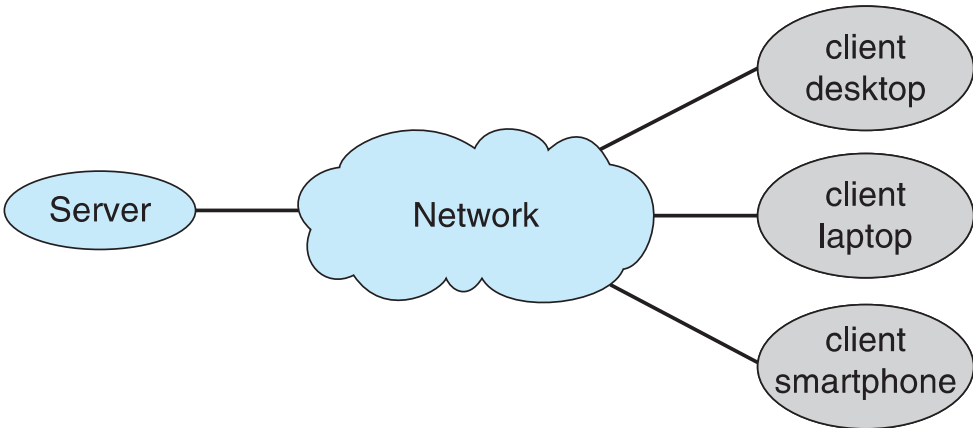
- Stand-alone general purpose machines
- But blurred as most systems interconnect with others (i.e., the Internet)
- **Portals** provide web access to internal systems
- **Network computers** (**thin clients**) are like Web terminals
- Mobile computers interconnect via **wireless networks**
- Networking becoming ubiquitous – even home systems use **firewalls** to protect home computers from Internet attacks

MOBILE

- Handheld smartphones, tablets, etc
- What is the functional difference between them and a "traditional" laptop?
- Extra feature – more OS features (GPS, gyroscope)
- Allows new types of apps like ***augmented reality***
- Use IEEE 802.11 wireless, or cellular data networks for connectivity
- Leaders are **Apple iOS** and **Google Android**

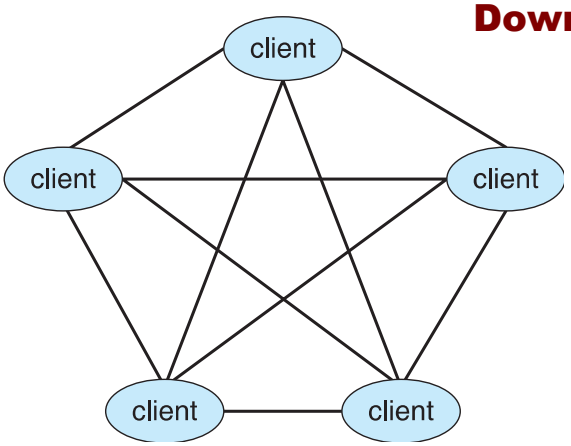
- Distributed computing
- Collection of separate, possibly heterogeneous, systems networked together

- **Network** is a communications path, **TCP/IP** most common
- **Local Area Network** (**LAN**)
- **Wide Area Network** (**WAN**)
- **Metropolitan Area Network** (**MAN**)
- **Personal Area Network** (**PAN**)
- **Network Operating System** provides features between systems across network
- Communication scheme allows systems to exchange messages
- Illusion of a single system
- Client-Server Computing
- Dumb terminals supplanted by smart PCs
- Many systems now **servers**, responding to requests generated by **clients**
- **Compute-server system** provides an interface to client to request services (i.e., database)
- **File-server system** provides interface for clients to store and retrieve files



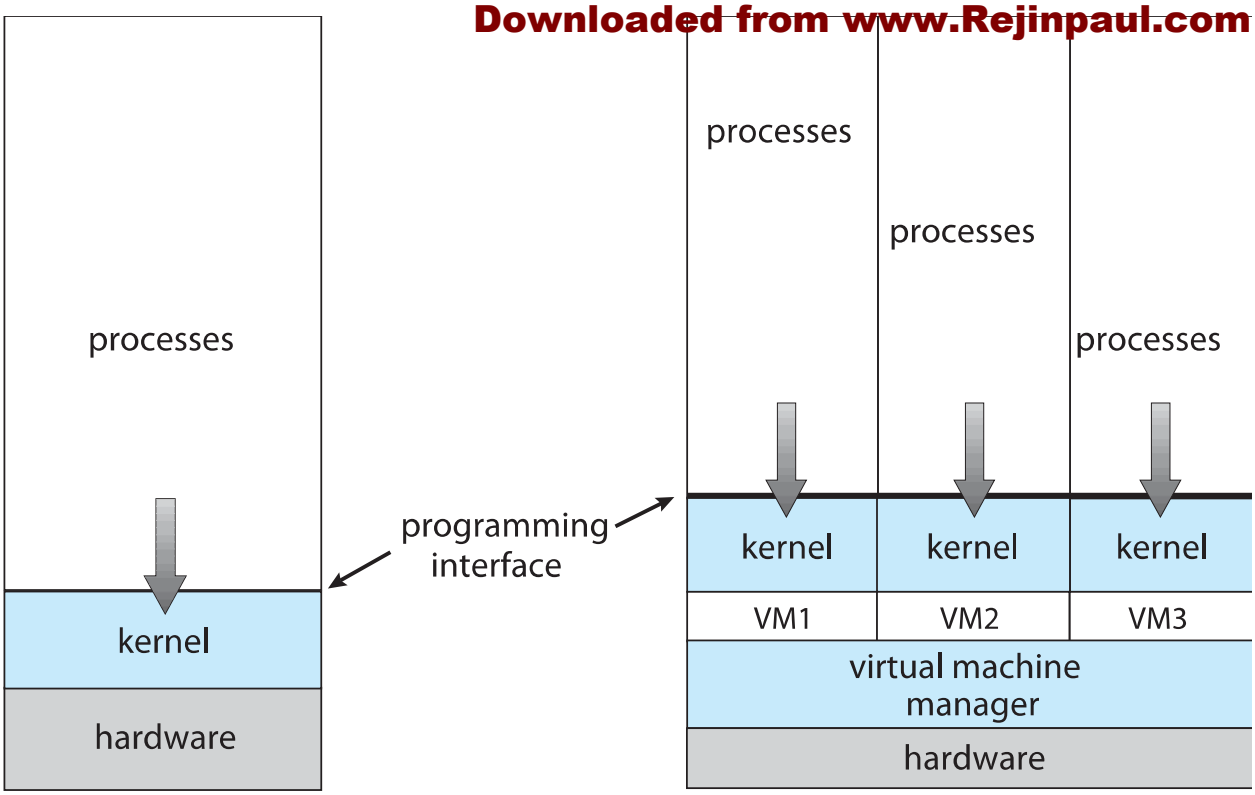
P2P does not distinguish clients and servers

- Instead all nodes are considered peers
- May each act as client, server or both
- Node must join P2P network
  - Registers its service with central lookup service on network, or
  - Broadcast request for service and respond to requests for service via ***discovery protocol***
- Examples include Napster and Gnutella, **Voice over IP** (**VoIP**) such as Skype



Virtualization

- **Host** OS, natively compiled for CPU
- **VMM** - virtual machine manager
- Creates and runs virtual machines
- VMM runs **guest** OSes, also natively compiled for CPU
- Applications run within these guest OSes
- Example: Parallels for OS X running Win and/or Linux and their apps
- Some VMM'es run within a host OS
- But, some act as a specialized OS
- Example. VMware ESX: installed on hardware, runs when hardware boots, provides services to apps, runs guest OSes
- Vast and growing industry
- Use cases
- Developing apps for multiple different OSes on 1 PC
- Very important for **cloud computing**
- Executing and managing **compute environments** in data centers



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

OPERATING SYSTEM SERVICES

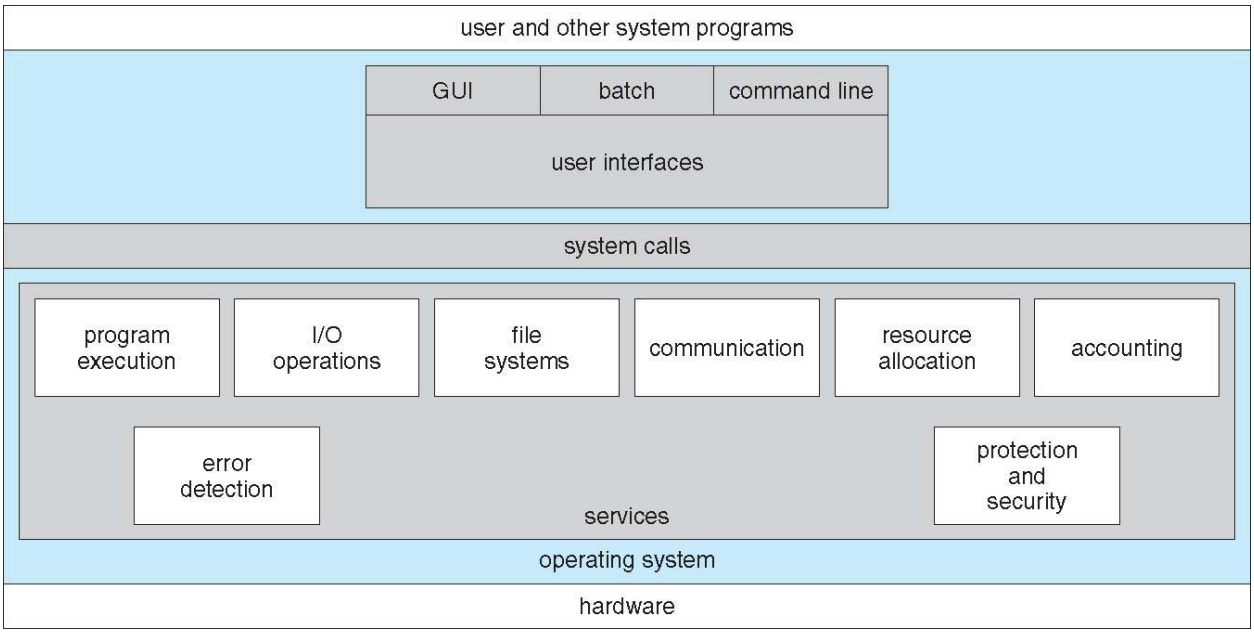
- Operating System Services
- User Operating System Interface
- System Calls
- Types of System Calls
- System Programs
- Operating System Design and Implementation

- Operating System Structure
- Operating System Debugging
- Operating System Generation
- System Boot
- Operating systems provide an environment for execution of programs and services (helpful functions) to programs and users
- **User services:**
    - **User interface**
        - **No UI**, **Command-Line (CLI)**, **Graphics User Interface (GUI)**, **Batch**
    - **Program execution** - Loading a program into memory and running it, 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
    - User services (Cont.):
    - **File-system manipulation** - Programs need to read and write files and directories, create and delete them, search them, list file Information, permission management.
    - **Communications** – Processes may exchange information, on the same computer or between computers over a network
        - Communications may be via shared memory or through message passing (packets moved by the OS)
    - **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

System services:

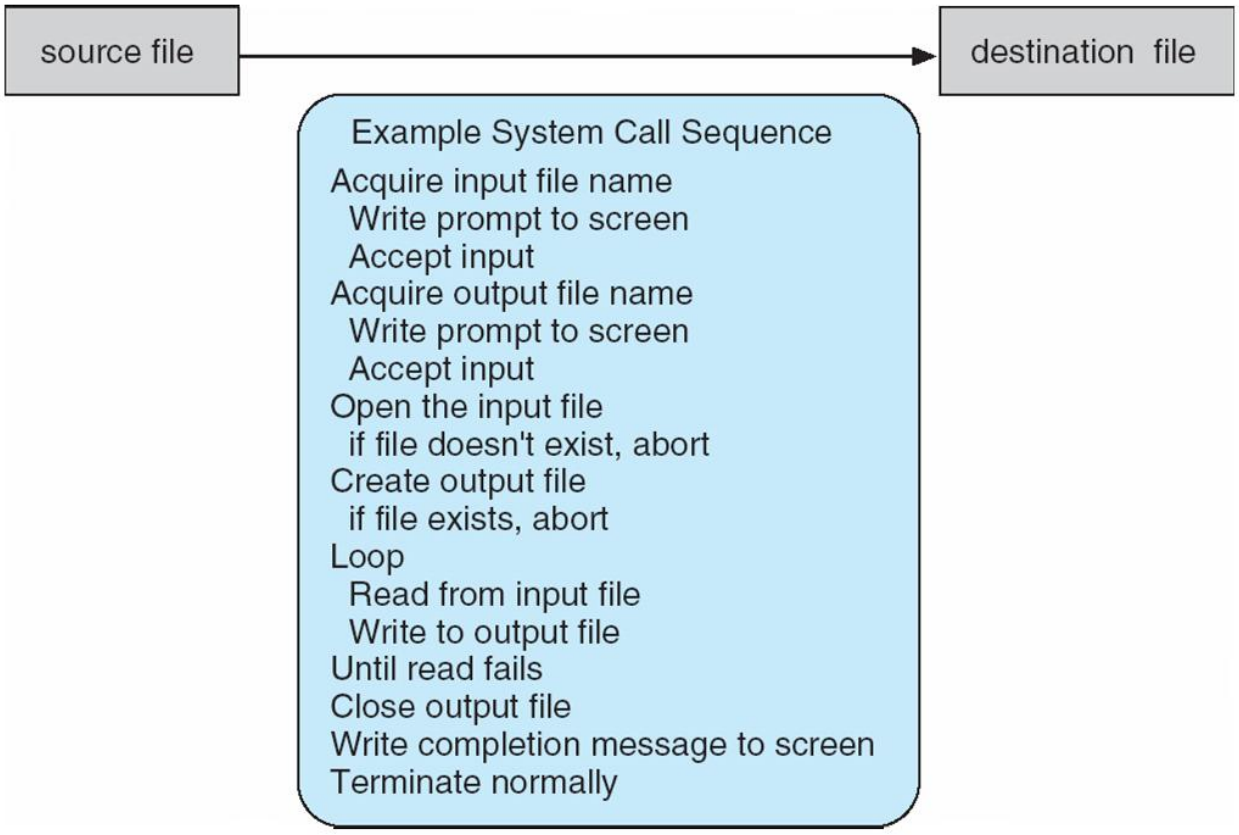
- For ensuring the efficient operation of the system itself via resource sharing
- **Resource allocation** - When multiple users or multiple jobs running concurrently, resources must be allocated to each of them

        - Many types of resources - CPU cycles, main memory, file storage, I/O devices.
    - **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 multiuser or networked computer system may want to control use of that information, concurrent processes should not interfere with each other
        - **Protection** involves ensuring that all access to system resources is controlled
        - **Security** of the system from outsiders requires user authentication, extends to defending external I/O devices from invalid access attempts

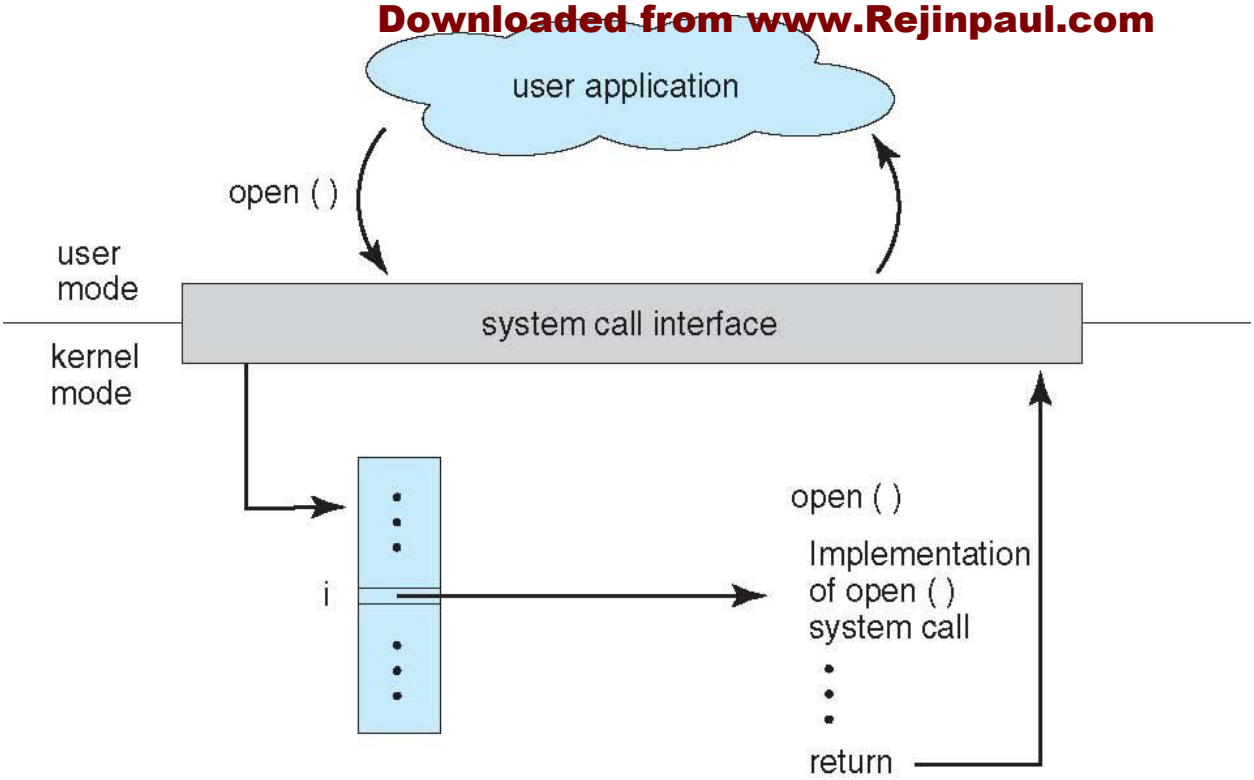


- CLI or **command interpreter** allows direct command entry
    - Sometimes implemented in kernel, sometimes by systems program
    - Sometimes multiple flavors implemented – **shells**
    - Primarily fetches a command from user and executes it
    - Sometimes commands built-in, sometimes just names of programs
        - If the latter, adding new features doesn't require shell modification
- Systems calls: programming interface to the services provided by the OS
- Typically written in a high-level language (C or C++)
- Mostly accessed by programs via a high-level **Application Programming Interface (API)** rather than direct system call use
- Three most common APIs are

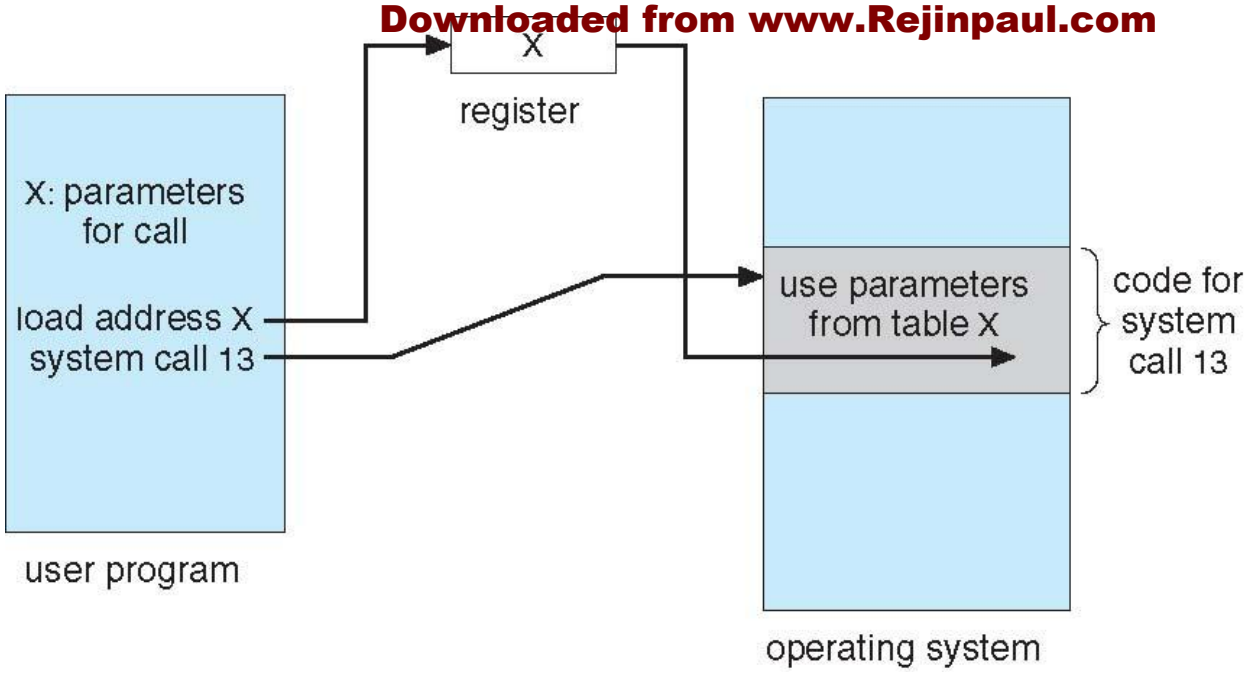
- Win32 API for Windows,
- POSIX API for POSIX-based systems
  - including virtually all versions of UNIX, Linux, and Mac OS X,
- Java API for the Java virtual machine (JVM)



- Typically, a number associated with each system call
  - **System-call interface** maintains a table indexed according to these numbers
- The system call interface invokes the intended system call in OS kernel and returns status of the system call and any return values
- The caller need know nothing about how the system call is implemented
  - Just needs to obey API and understand what OS will do as a result call
  - Most details of OS interface hidden from programmer by API
    - Managed by run-time support library (set of functions built into libraries included with compiler)



- Three general methods used to pass parameters to the OS in system calls
  - Simplest: 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

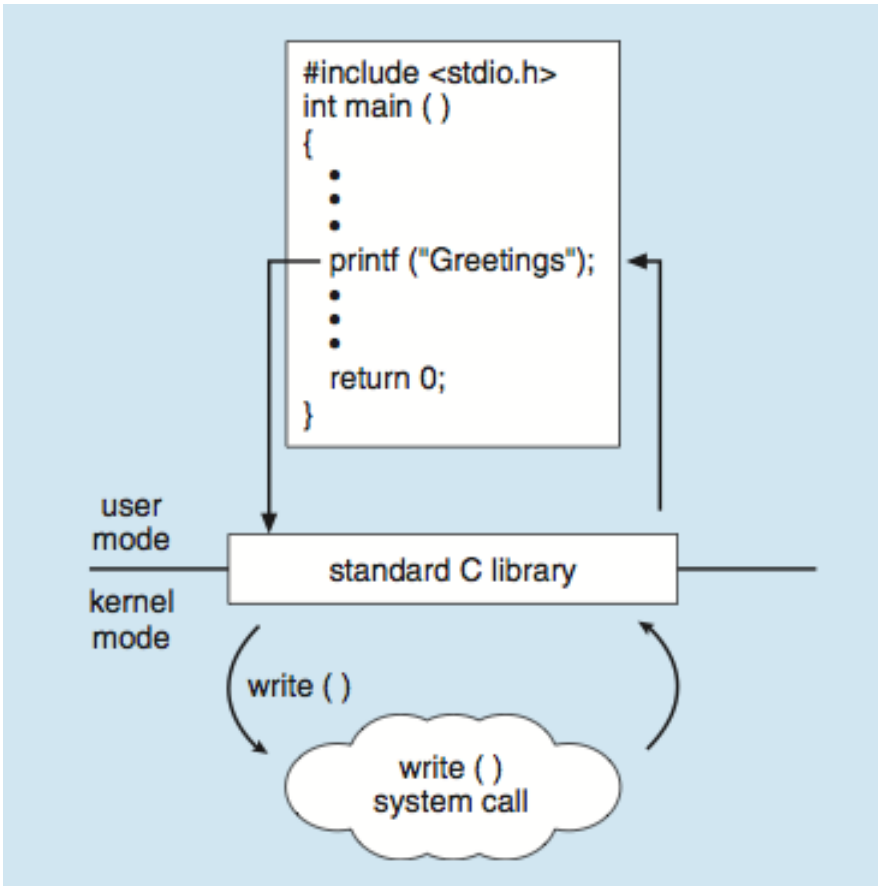


TYPES OF SYSTEM CALLS

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

	Windows	Unix
Process Control	CreateProcess() ExitProcess() WaitForSingleObject()	fork() exit() wait()
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()	pipe() shmget() mmap()
Protection	SetFileSecurity() InitializeSecurityDescriptor() SetSecurityDescriptorGroup()	chmod() umask() chown()



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

- Provide a convenient environment for program development and execution
    - Some of them are simply user interfaces to system calls; others are considerably more complex
    - **File management** - Create, delete, copy, rename, print, dump, list, and generally manipulate files and directories
- **Status information**
    - Some ask the system for info - date, time, amount of available memory, disk space, number of users
    - Others provide detailed performance, logging, and debugging information
    - Typically, these programs format and print the output to the terminal or other output devices
    - Some systems implement a **registry** - used to store and retrieve configuration information
- **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 the design 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 (example – timer)
- 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++
- Actually 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
- General-purpose OS is a **very large program**
  - How to implement and structure it?
  - Can apply many ideas from software engineering
    - Software engineering - a separate area in CS
    - Studies design, development, and maintenance of software
- A common approach is to partition OS into modules/components
  - Each modules is responsible for one (or several) aspect of the desired functionality
  - Each module has carefully defined interfaces
  - **Advantages:**
    - Traditional advantages of modular programming:

- Simplifying development and maintenance of computer programs, etc
- Modules can be developed independently from each other

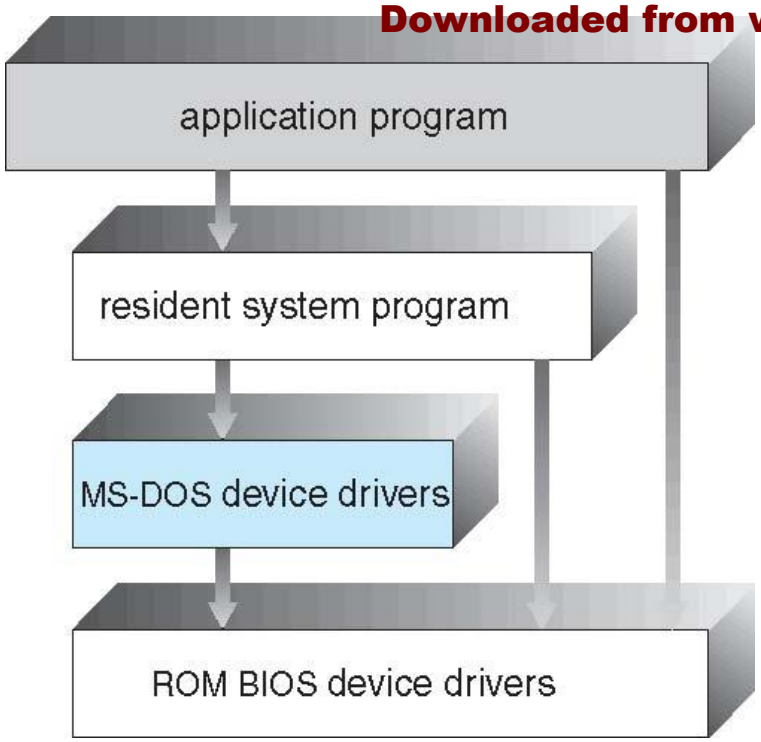
- **Disadvantages:**
  - Efficiency can decrease (vs monolithic approach)

OPERATING SYSTEM STRUCTURES

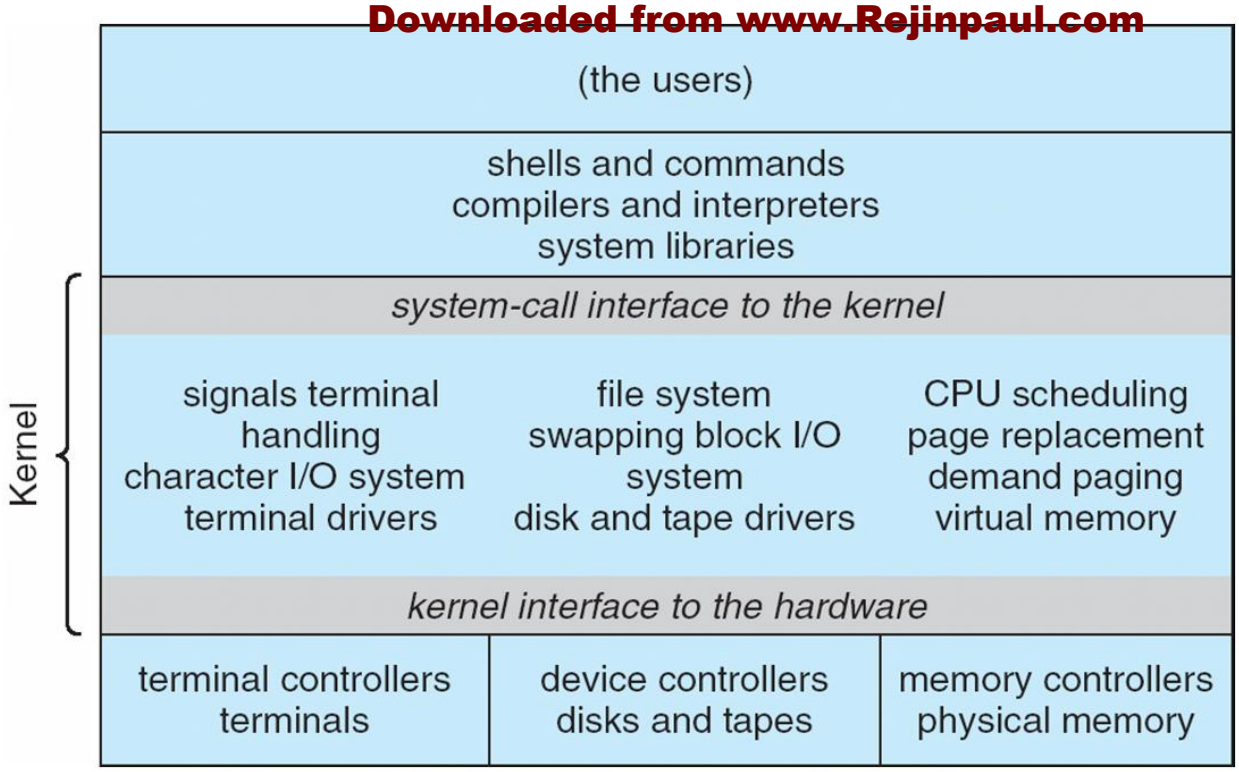
- In general, various ways are used to structure OSes
- Many OS'es don't have well-defined structures
  - Not one pure model: Hybrid systems
  - Combine multiple approaches to address performance, security, usability needs
- Simple structure – MS-DOS
- More complex structure - UNIX
- Layered OSes
- Microkernel OSes

SIMPLE STRUCTURE

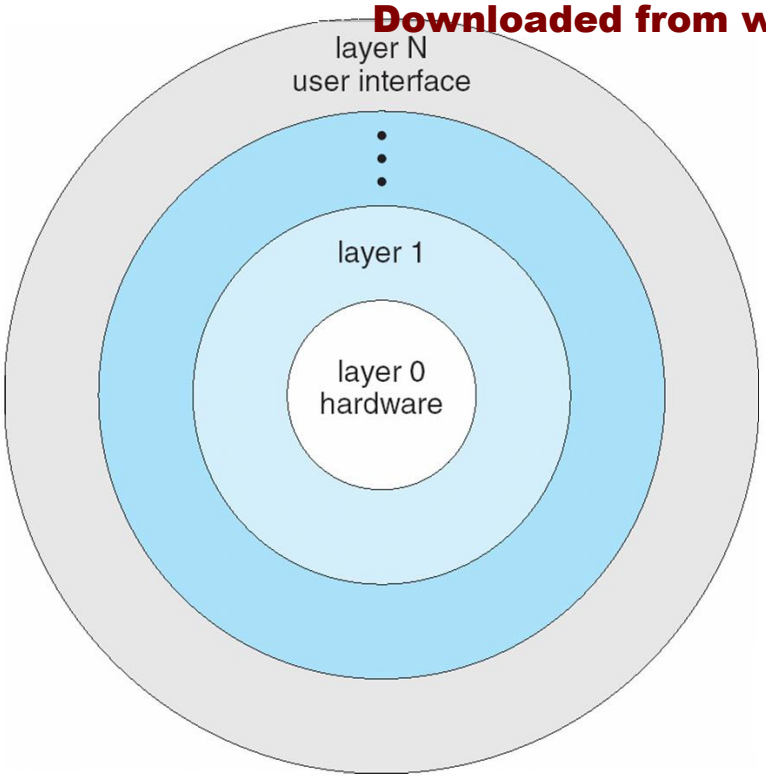
- MS-DOS was created to provide the most functionality in the least space
- Not divided into modules
- MS-DOS has some structure
  - But its interfaces and levels of functionality are not well separated
- No dual mode existed for Intel 8088
  - MS-DOS was developed for 8088
  - Direct access to hardware is allowed
    - System crashes possible

- 

- Traditional UNIX has limited structuring
- UNIX consists of 2 separable parts:
    - Systems programs
    - Kernel
- UNIX Kernel
    - Consists of everything that is
        - below the system-call interface and
        - above the physical hardware
    - Kernel provides
        - File system, CPU scheduling, memory management, and other operating-system functions
        - This is a lot of functionality for just 1 layer
        - Rather monolithic
            - But fast -- due to lack of overhead in communication inside kernel



- LAYERED APPROACH
- One way to make OS modular – layered approach
- The OS is divided into a number of layers (levels)
- Each layer is built on top of lower layers
- The bottom layer (layer 0), is the hardware
- The highest (layer N) is the user interface
- Layers are selected such that each uses functions (operations) and services of only lower-level layers
- Advantages:
    - simplicity of construction and debugging
- Disadvantages:
    - can be hard to decide how to split functionality into layers
    - less efficient due to high overhead



OPERATING SYSTEM DEBUGGING

- **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."

PERFORMANCE TUNNING

- 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

OS SYSGEN

- Operating systems are designed to run on any of a class of machines

- The system must be configured for each specific computer site
- The process of configuration is known as system generation **SYSGEN**
    - How to format partitions
    - Which hardware is present
    - Etc
- Used to build system-specific compiled kernel or system-tuned
- Can general more efficient code than one general kernel

SYSTEM BOOT

- How OS is loaded?
- When power is initialized on system, execution starts at a predefined memory location
    - Firmware ROM is used to hold initial bootstrap program (=bootstrap loader)
- Bootstrap loader
    - small piece of code
    - locates the kernel, loads it into memory, and starts it
- Sometimes 2-step process is used instead
    - Simple bootstrap loader in ROM loads a more complex boot program from (a fixed location on) disk
    - This more complex loader loads the kernel

UNIT II

PROCESSES - PROCESS CONCEPT

- An operating system executes a variety of programs:
  - Batch system – "**jobs**"
  - Time-shared systems – "**user programs**" or "**tasks**"
- We will use the terms ***job*** and ***process*** almost interchangeably
- **Process** – is a program in execution (informal definition)
- Program is ***passive*** entity stored on disk (**executable file**), process is ***active***
  - Program becomes process when executable file loaded into memory
- Execution of program started via GUI, command line entry of its name, etc
- One program can be several processes
  - Consider multiple users executing the same program
- In memory, a process consists of **multiple parts:**
  - **Program code**, also called **text section**
  - **Current activity** including
    - **program counter**
    - processor registers
  - **Stack** containing temporary data
    - Function parameters, return addresses, local variables
  - **Data section** containing global variables
  - **Heap** containing memory dynamically allocated during run time

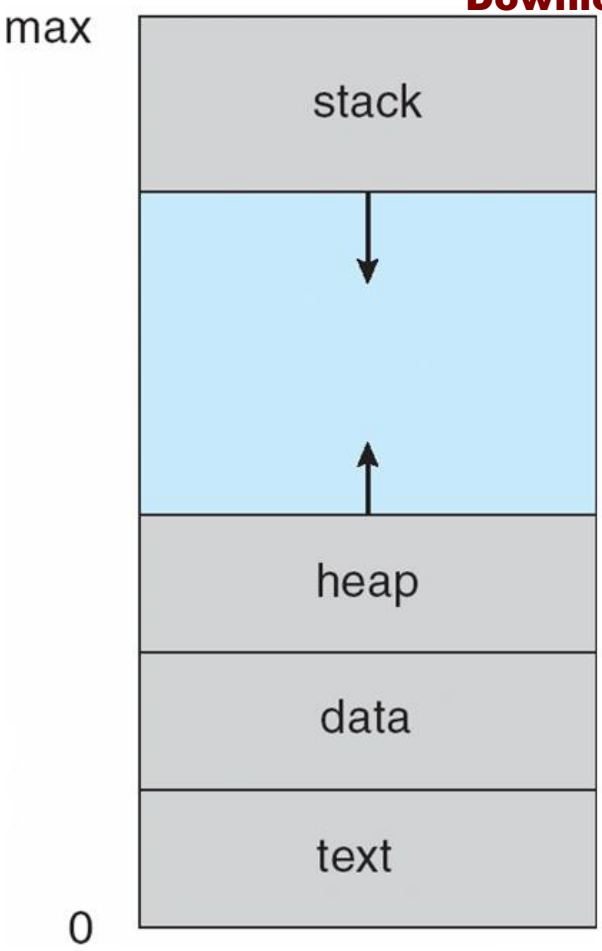
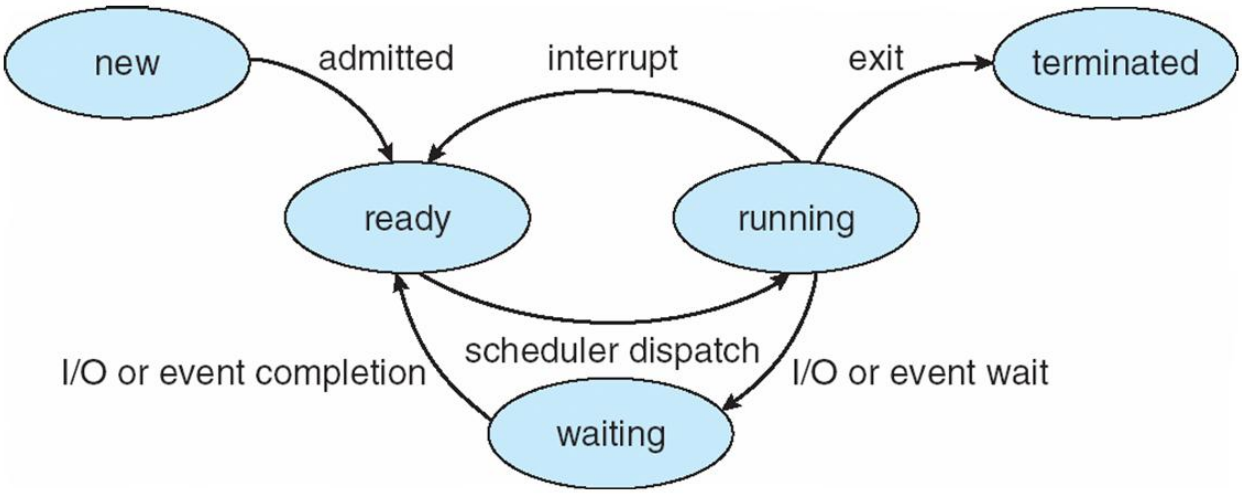


DIAGRAM OF PROCESS STATE



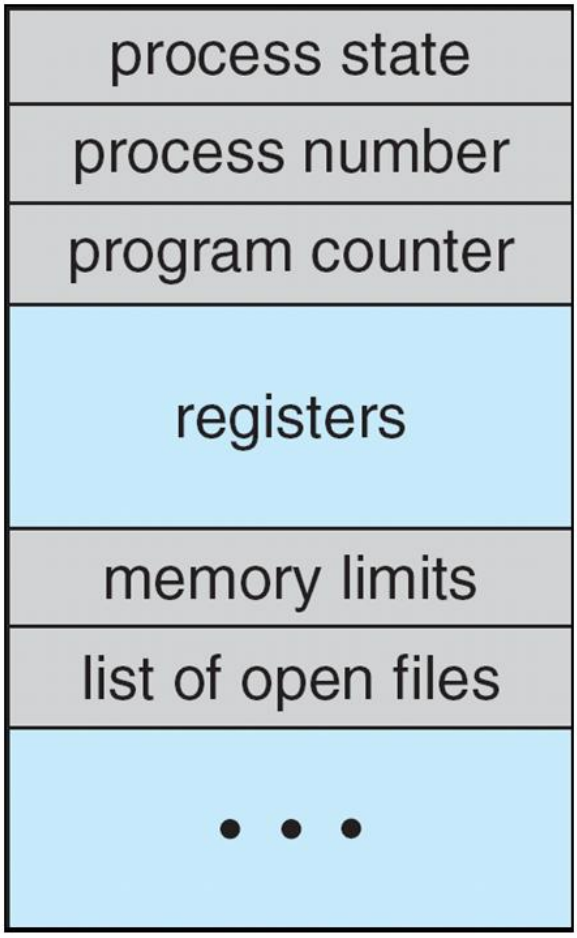
- As a process executes, it changes **state**
  - **new**: The process is being created
  - **ready**: The process is waiting to be assigned to a processor
  - **running**: Instructions are being executed
  - **waiting**: The process is waiting for some event to occur

- **terminated**: The process has finished execution

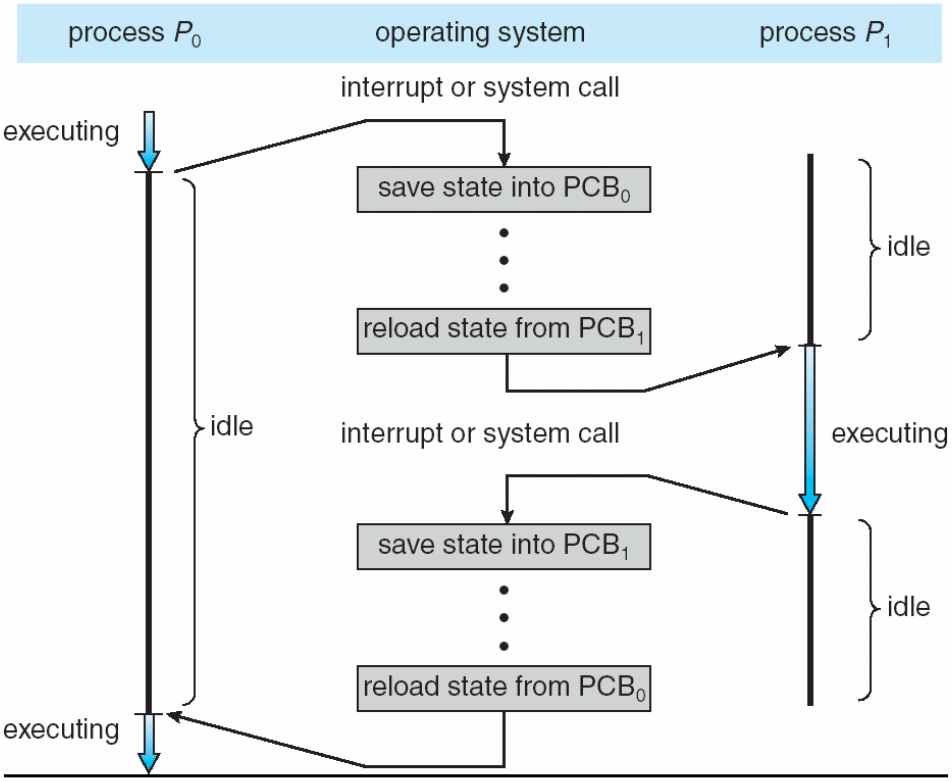
PROCESS CONTROL BLOCK(PCB)

Each process is represented in OS by PCB

- PCB - info associated with the process
- Also called **task control block**
- Process state – running, waiting, etc
- Program counter – location of instruction to next execute
- CPU registers – contents of all process-centric registers
- CPU scheduling information- priorities, scheduling queue pointers
- Memory-management information – memory allocated to the process
- Accounting information – CPU used, clock time elapsed since start, time limits
- I/O status information – I/O devices allocated to process, list of open files



CPU SWITCH FROM PROCESS TO PROCESS



THREADS

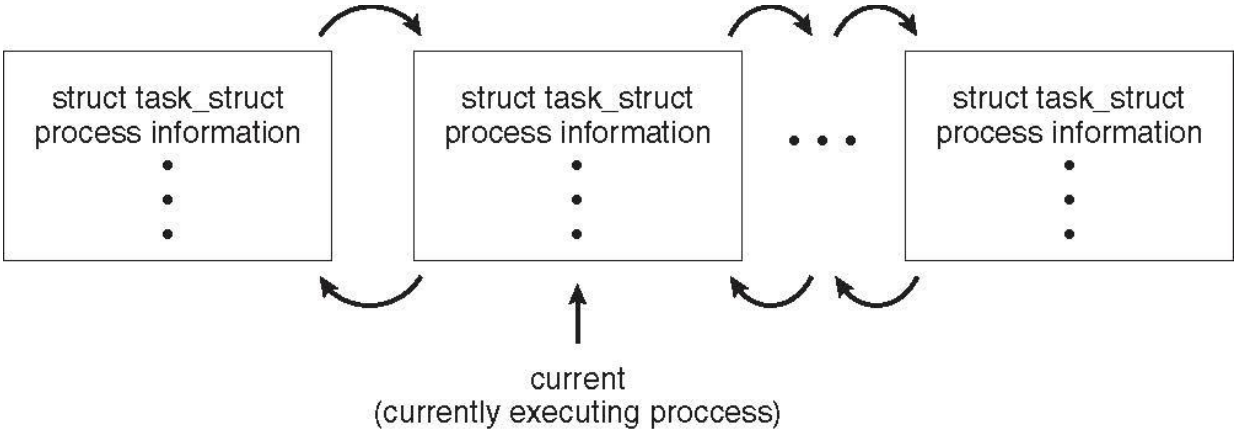
- So far, process has a single thread of execution
  - One task at a time
- Consider having multiple program counters per process
  - Multiple locations can execute at once
  - Multiple tasks at a time
  - Multiple threads of control -> **threads**
- PCB must be extended to handle threads:
  - Store thread details
  - Multiple program counters

PROCESS REPRESENTATION IN LINUX

Represented by the C structure task_struct

pid t_pid; /* process identifier */
long state; /* state of the process */
unsigned int time_slice /* scheduling information */
struct task_struct *parent; /* this process's parent */
struct list_head children; /* this process's children */

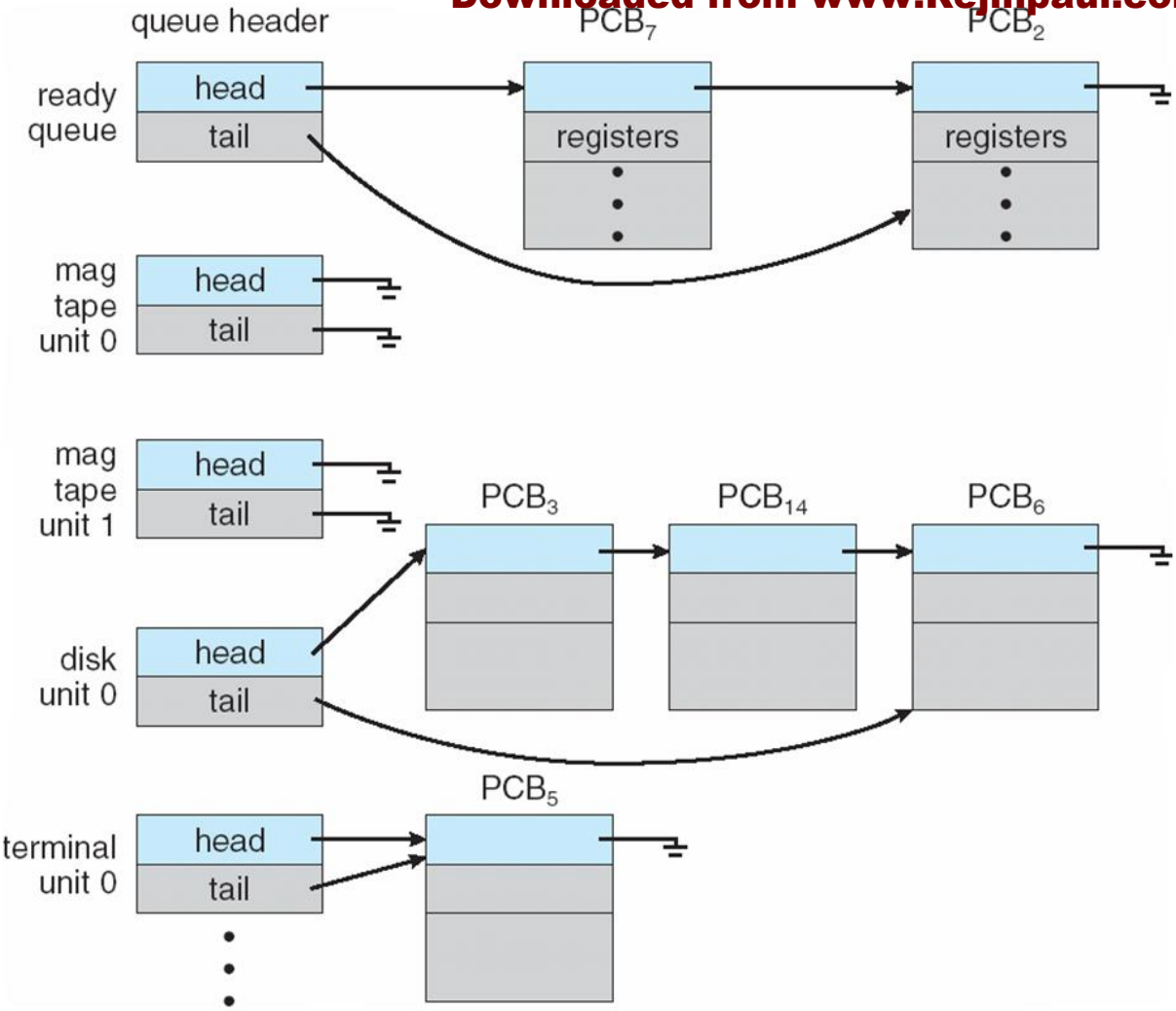
struct files_struct *files; /* list of open files */
struct mm_struct *mm; /* address space of this process */



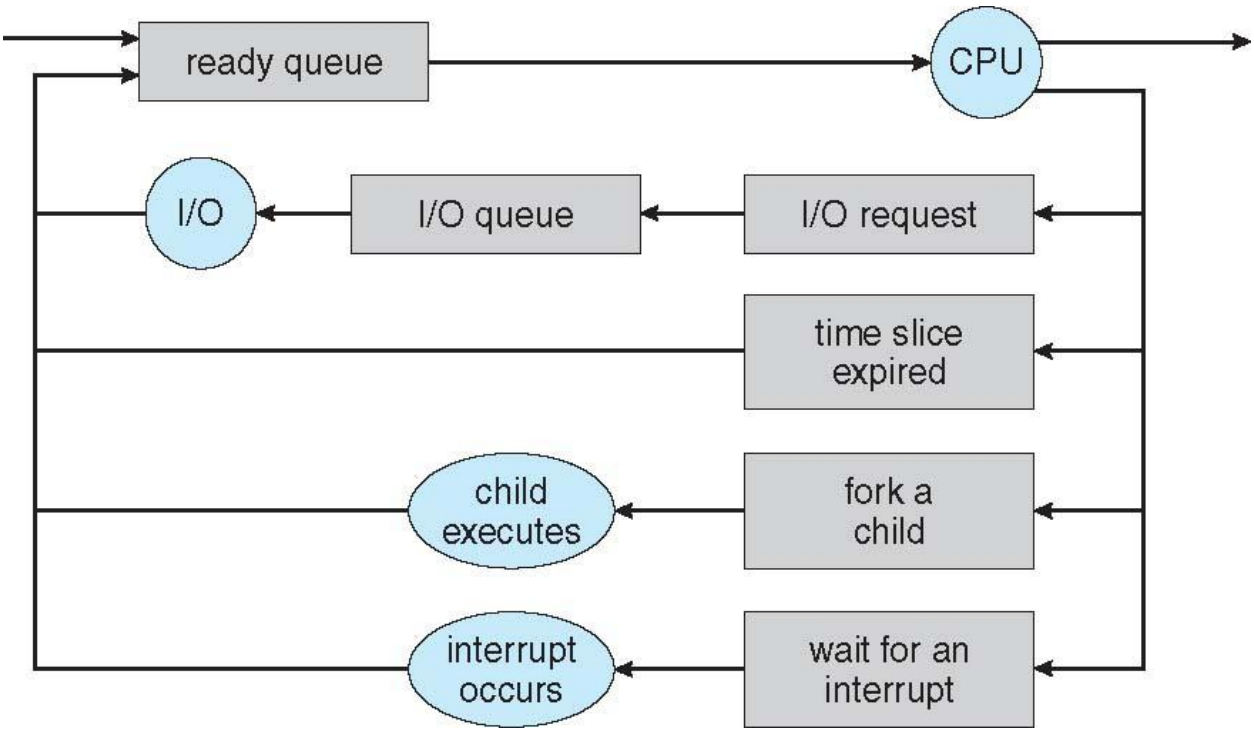
PROCESS SCHEDULING

- Goal of multiprogramming:
  - Maximize CPU use
- Goal of time sharing:
  - Quickly switch processes onto CPU for time sharing
- **Process scheduler** – needed to meet these goals
  - Selects 1 process to be executed next on CPU
  - Among available processes
- Maintains **scheduling queues** of processes
  - **Job queue** – set of all processes in the system
  - **Ready queue** – set of all processes residing in main memory, ready and waiting to execute
  - **Device queues** – set of processes waiting for an I/O device
  - Processes migrate among the various queues

READY QUEUE AND VARIOUS QUEUES



REPRESENTATION OF PROCESS SCHEDULING



- **Queuing diagram**

- a common representation of process scheduling
- represents queues, resources, flows

SCHEDULERS

- **Scheduler –** component that decides how processes are selected from these queues for scheduling purposes

  Long-term scheduler (or **job scheduler**)

- On this slide - "LTS" (LTS is not a common notation)
- In a **batch system**, more processes are submitted then can be executed in memory
- They are spooled to disk
- LTS selects which processes should be brought into the ready queue
- LTS is invoked infrequently
- (seconds, minutes) ⇒ (may be slow, hence can use advanced algorithms)
- LTS controls the **degree of multiprogramming**
- The number of processes in memory
- Processes can be described as either:

  I/O-bound process

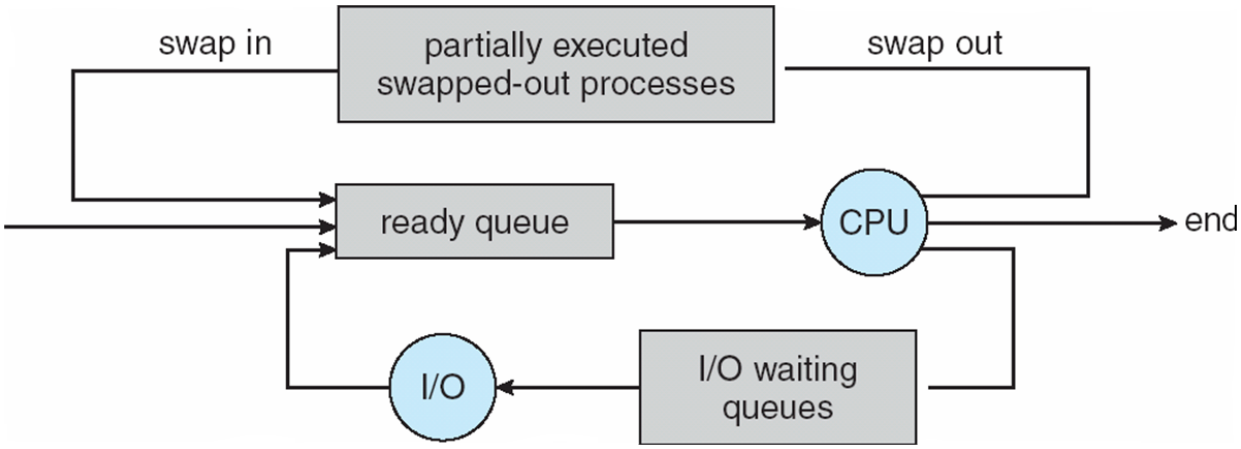
- Spends more time doing I/O than computations, many short CPU bursts

  CPU-bound process

- Spends more time doing computations; few very long CPU bursts
- LTS strives for good ***process mix***
- **Short-term scheduler** (or **CPU scheduler**)
  - Selects 1 process to be executed next
    - Among ready-to-execute processes
      - From the ready queue
    - Allocates CPU to this process
  - Sometimes the only scheduler in a system
  - Short-term scheduler is invoked frequently
    - (milliseconds) ⇒ (must be fast)

    - Hence cannot use costly selection logic

- **Medium-term scheduler** can be added if degree of multiple programming needs to decrease
    - Used by time-sharing OSes, etc
        - Too many programs ➔poor performance ➔users quit
- Key idea:
    - Reduce the degree of multiprogramming by **swapping**
    - **Swapping** removes a process from memory, stores on disk, brings back in from disk to continue execution



CONTEXT SWITCH

- **Context** of a process represented in its PCB
- **Context switch**
    - When CPU switches to another process, the system must:
        - **save the state** of the old process, and
        - load the **saved state** for the new process
- Context-switch time is overhead
    - The system does no useful work while switching
    - The more complex the OS and the PCB ➔
        - the longer the context switch
- This overhead time is dependent on hardware support

- Some hardware provides multiple sets of registers per CPU
    - multiple contexts are loaded at once
    - switch requires only changing pointer to the right set

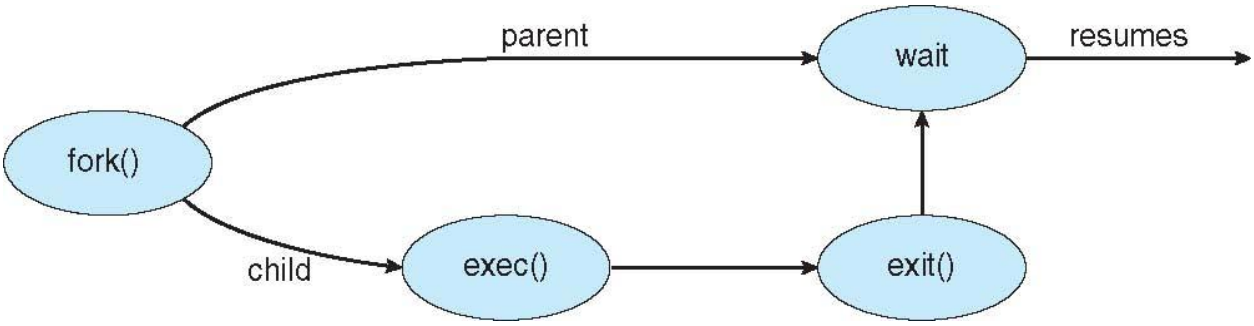
OPERATIONS ON PROCESSES

- System must provide mechanisms for:
    - process creation,
    - process termination,

PROCESS CREATION

- A **(parent)** process can create several (**children)** processes
- Children can, in turn, create other processes
- Hence, a **tree** of processes forms
- Generally, process identified and managed via a **process identifier** (**pid**)
- Resource sharing options (of process creation)
- Parent and children share all resources
- Children share subset of parent's resources
- One usage is to prevent system overload by too many child processes
- Parent and child share no resources
- Execution options
- Parent and children execute concurrently
- Parent waits until children terminate
- Address space options
    - Child duplicate of parent
    - Child has a program loaded into it
- UNIX examples
    - **fork()** system call creates new process
        - Child is a copy of parent's address space
            - except fork() returns 0 to child and nonzero to parent

- **exec()** system call used after a **fork()** to replace the process' memory space with a new program



PROCESS TERMINATION

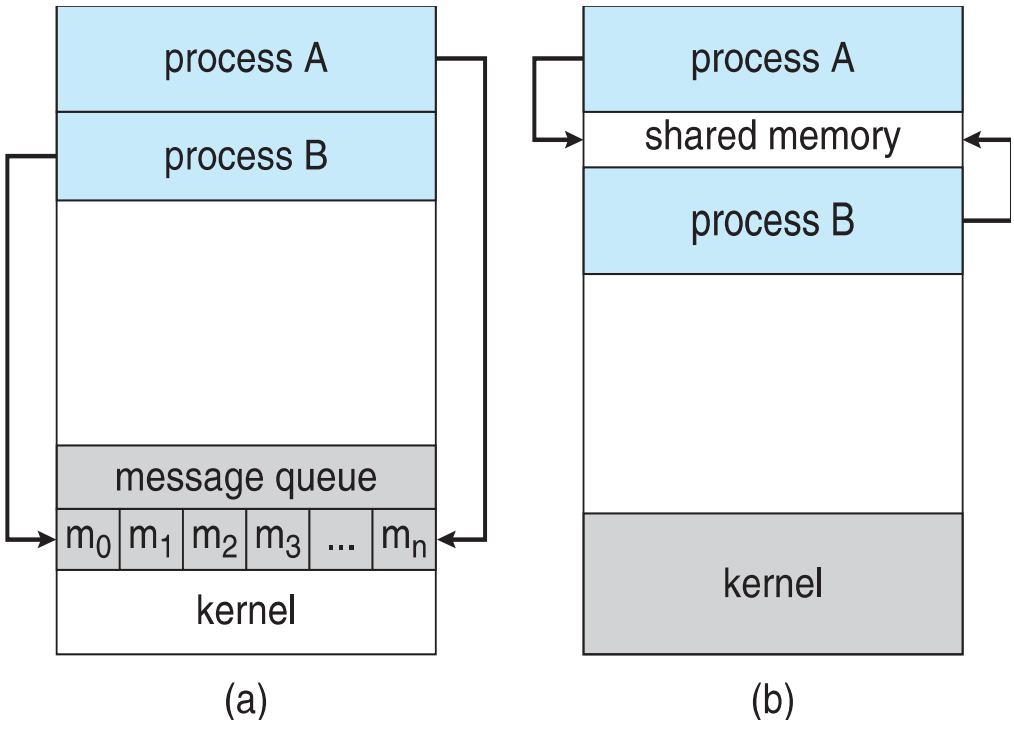
- Process executes last statement and then asks the operating system to delete it using the **exit()** system call.
    - Returns status data from child to parent (via **wait()**)
    - Process' resources are deallocated by operating system
- Parent may terminate the execution of children processes using the **abort()** system call. Some reasons for doing so:
    - Child has exceeded allocated resources
    - Task assigned to child is no longer required
    - The parent is exiting and the operating systems does not allow a child to continue if its parent terminates
    - Some OSes don't allow child to exists if its parent has terminated
    - **cascading termination -** if a process terminates, then all its children, grandchildren, etc must also be terminated.
    - The termination is initiated by the operating system
- The parent process may wait for termination of a child process by using the **wait()**system call**.**
    - The call returns status information and the pid of the terminated process
    - **pid = wait(&status);**
- If no parent waiting (did not invoke **wait()**) process is a **zombie**
    - All its resources are deallocated, but exit status is kept
- If parent terminated without invoking **wait** , process is an **orphan**
    - UNIX: assigns **init** process as the parent
    - Init calls wait periodically

CO OPERATING PROCESSES

- Processes within a system may be ***independent*** or ***cooperating***
  - When processes execute they produce some **computational results**
  - ***Independent*** process cannot affect (or be affected) by such results of another process
  - ***Cooperating*** process can affect (or be affected) by such results of another process
- Advantages of process cooperation
  - Information sharing
  - Computation speed-up
  - Modularity
  - Convenience

INTERPROCESS COMMUNICATION

- For fast exchange of information, cooperating processes need some **interprocess communication** (**IPC**) mechanisms
- Two models of IPC
  - **Shared memory**
  - **Message passing**



- An area of memory is shared among the processes that wish to communicate

- The communication is under the control of the users processes, not the OS.
- Major issue is to provide mechanism that will allow the user processes to synchronize their actions when they access shared memory.
- **Producer-consumer problem** – a common paradigm for cooperating processes
    - Used to exemplify one common generic way/scenario of cooperation among processes
    - We will use it to exemplify IPC
    - Very important!
- ***Producer*** process
    - produces some information
    - incrementally
- ***Consumer*** process
    - consumes this information
    - as it becomes available
- Challenge:
    - Producer and consumer should run concurrently and efficiently
    - Producer and consumer must be synchronized
        - Consumer cannot consume an item before it is produced
- Shared-memory solution to producer-consumer
    - Uses a buffer in shared memory to exchange information
        - **unbounded-buffer:** assumes no practical limit on the buffer size
        - **bounded-buffer** assumes a fixed buffer size
- Shared data

```
#define BUFFER_SIZE 10

typedef struct {

    . . .

} item;

item buffer[BUFFER_SIZE];

int in = 0;
```

```
int out = 0;

item next_produced;

while (true) {

        next_produced = ProduceItem();


        while (((in + 1) % BUFFER_SIZE) == out)

                ; /* do nothing, no space in buffer */

              //wait for consumer to get items and          //free up some space


    /* enough space in buffer */

        buffer[in] = next_produced; //put item into buffer

        in = (in + 1) % BUFFER_SIZE;

}
```

MESSAGE PASSING

- Mechanism for processes to communicate and to synchronize their actions
- Message system – processes communicate with each other without resorting to shared variables
- IPC facility provides two operations:
    - **send**(*message*)
    - **receive**(*message*)
- The *message* size is either fixed or variable
    - If processes *P* and *Q* wish to communicate, they need to:
    - Establish a ***communication link*** between them
    - Exchange messages via send/receive
    - Implementation issues:
    - How are links established?
    - Can a link be associated with more than two processes?
    - How many links can there be between every pair of communicating processes?
    - What is the capacity (buffer size) of a link?

- Is the size of a message that the link can accommodate fixed or variable?
- Is a link unidirectional or bi-directional?

Logical implementation of communication link

- Direct or indirect
- Synchronous or asynchronous
- Automatic or explicit buffering

DIRECT COMMUNICATION

- Processes must name each other explicitly:
  - **send** (*P, message*) – send a message to process P
  - **receive**(*Q, message*) – receive a message from process Q
  - Properties of a direct communication link
  - Links are established automatically
  - A link is associated with exactly one pair of communicating processes
  - Between each pair there exists exactly one link
  - The link may be unidirectional, but is usually bi-directional

INDIRECT COMMUNICATION

- Messages are directed and received from mailboxes (also referred to as ports)
- Each mailbox has a unique id
- Processes can communicate only if they share a mailbox
- Properties of an indirect communication link
- Link established only if processes share a common mailbox
- A link may be associated with many processes
- Each pair of processes may share several communication links
- Link may be unidirectional or bi-directional
- Operations
  - create a new mailbox (port)
  - send and receive messages through mailbox
  - destroy a mailbox

- Primitives are defined as:

send(*A, message*) – send a message to mailbox A

receive(*A, message*) – receive a message from mailbox A

SYNCHRONIZATION

- Message passing may be either
  - Blocking, or
  - Non-blocking
- **Blocking** is considered **synchronous**
  - **Blocking send** -- the sender is blocked until the message is received
  - **Blocking receive** -- the receiver is blocked until a message is available
- **Non-blocking** is considered **asynchronous**
  - **Non-blocking send** -- the sender sends the message and continues
  - **Non-blocking receive** -- the receiver receives:
    - A valid message, or
    - Null message
- Different combinations possible
  - If both send and receive are blocking – called a **rendezvous**
- Producer-consumer is trivial via rendezvous

```
message next_produced;


while (true) {
        ProduceItem(&next_produced);


        send(next_produced);
}
```

BUFFERING

- Queue of messages is attached to the link.
- Implemented in one of three ways

1. Zero capacity

– no messages are queued on a link.

- Sender must wait for receiver (rendezvous)

2.Bounded capacity

– finite length of *n* messages
- Sender must wait if link full

3.Unbounded capacity

– infinite length
- Sender never waits

THE CRITICAL SECTION PROBLEM

- Consider system of ***n*** processes {P_0, P_1, … ,P_{n-1}}
- Each process has **critical section** segment of code
    - Process may be changing common variables, updating table, writing file, etc
    - When one process in critical section, no other is allowed to be in its critical section
- ***Critical section problem*** is to design protocol to solve this
- Each process
    - must ask permission to enter critical section in **entry section**,
    - may follow critical section with **exit section**,
    - then **remainder section**

```
do {
    entry section
        critical section
    exit section
        remainder section
} while (true);
```

- Each process $\mathbf{P_i}$ has its own critical section
    - Accesses to shared data only in CS
- Lock – will consider later

- In shared data, accessible by all
- Often 1 lock for all n processes
- Guards access to all critical sections
- Ensures mutual exclusivity

- Assume that each process executes at a nonzero speed
- No assumption concerning **relative speed** of the ***n*** processes

1. **Mutual Exclusion**
    - l Only one process can be in the critical section at a time –
        - otherwise what critical section?
2. **Progress**
    - l **Intuition:** No process is forced to wait for an available resource –
        - otherwise very wasteful.
    - l **Formal:** If no process is executing in its critical section and there exist some processes that wish to enter their critical section, then the selection of the processes that will enter the critical section next cannot be postponed indefinitely
3. **Bounded Waiting**
    - l **Intuition:** No process can wait forever for a resource –
        - otherwise an easy solution: no one gets in
    - l **Formal:** A bound must exist on the number of times that other processes are allowed to enter their critical sections after a process has made a request to enter its critical section and before that request is granted

Algorithm for Process P_i

- Assume 2-process case: processes Pi and Pj

```
do {

        while (turn == j); //j is not my turn => wait
                critical section
        turn = j;  //change turn to j
                remainder section
} while (true);
```

- Mutual exclusion?
- Progress?
- Bounded waiting?

- Previous solutions are complicated and generally inaccessible to application programmers
- OS designers build software tools to solve critical section problem
- Simplest is mutex lock
- Protect a critical section by:
    - first **acquire()** a lock
    - then **release()** the lock
    - Boolean variable indicating if lock is available or not
- Calls to **acquire()** and **release()** must be atomic
    - Usually implemented via hardware atomic instructions
- But this solution requires **busy waiting**
    - This lock therefore called a **spinlock**

```
acquire() {
    while (!available)
        ; /* busy wait */
    available = false;
}
release() {
    available = true;
}
while (true) {
acquire lock
    critical section
release lock
  remainder section
}
```

SEMAPHORE

- Synchronization tool that provides more sophisticated ways (than Mutex locks) for process to synchronize their activities.
- Semaphore ***S*** – integer variable
- Can only be accessed via two indivisible (atomic) operations
  - **wait()** and **signal()**
  - Originally called **P()** and **V()--** important to remember, used often
    - **P()** (wait) – from Dutch proberen: "to test"
    - **V()** (signal) – from Dutch verhogen: "to increment"
- Definition of the **wait() operation**

```
wait(S) {
  while (S <= 0)
    ; // Notice, busy waits, spending CPU cycles, not (always) good
  S--;
}
```

- Definition of the **signal() operation**

signal(S) {

S++;

}

- **Counting semaphore** – integer value can range over an unrestricted domain
  - Can control access to resource that has a finite number of instances
  - Initialized to the number of resources available
- **Binary semaphore** – integer value can range only between 0 and 1
  - Same as a **mutex lock**
- Can solve various synchronization problems
- Consider P_1 and P_2 that require S_1 to happen before S_2

Create a semaphore "**synch**" initialized to 0

P1:

S_1;

signal(synch);

P2:

wait(synch);

S_2;

- Can implement a counting semaphore ***S*** via binary semaphores
- Data Structures

int S = n; // semaphore count

mutex S_guard = 1; // initial value 1 (FREE)

mutex delay = 0; // for delaying other processes

- Operations **wait()** and **signal()** must be executed atomically
- Thus, the implementation becomes the critical section problem
  - **wait** and **signal** code are placed in the critical section
- Hence, can now have **busy waiting** in critical section implementation
  - If critical section rarely occupied
    - Little busy waiting

- But, applications may spend lots of time in critical sections
  - Hence, the busy-waiting approach is not a good solution

- A **waiting queue** is associated with each semaphore
  - It stores the processes waiting on the semaphore

- typedef struct{

  int value;

  struct process *list; // waiting queue

  } semaphore;

- Two operations:
  - **block** – place the process invoking the operation on the appropriate waiting queue
  - **wakeup** – remove one of processes in the waiting queue and place it in the ready queue

```
wait(semaphore *S) {
  S->value--;
  if (S->value < 0) {
    add this process to S->list;
    block(); //suspends self, sleeps, avoids CPU cycles
  }
}
signal(semaphore *S) {
  S->value++;
  if (S->value <= 0) {
    remove a process P from S->list;
    wakeup(P);
  }
}
```

- S->value can be negative
- |S->value| is then the number of processes waiting on the semaphore
- It is never negative in classical definition

DEADLOCK

- **Synchronization problems**
- **Deadlock** – two or more processes are waiting indefinitely for an event that can be caused by only one of the waiting processes
- Let ***S*** and ***Q*** be two semaphores initialized to 1

P_0	P_1
wait(S);	**wait(Q);**
wait(Q);	**wait(S);**
...	**...**
signal(S);	**signal(Q);**
signal(Q);	**signal(S);**

- **Starvation** (= **indefinite blocking)**
  - Related to deadlocks (but different)
  - Occurs when a process waits indefinitely in the semaphore queue
  - For example, assume a LIFO semaphore queue
    - Processed pushed first into it might not get a chance to execute
- **Priority Inversion**
  - A situation when a higher-priority process needs to wait for a lower-priority process that holds a lock

Classical problems used to test newly-proposed synchronization schemes

- Bounded-Buffer Problem
- Readers and Writers Problem
- Dining-Philosophers Problem

MONITORS

monitor monitor-name

{

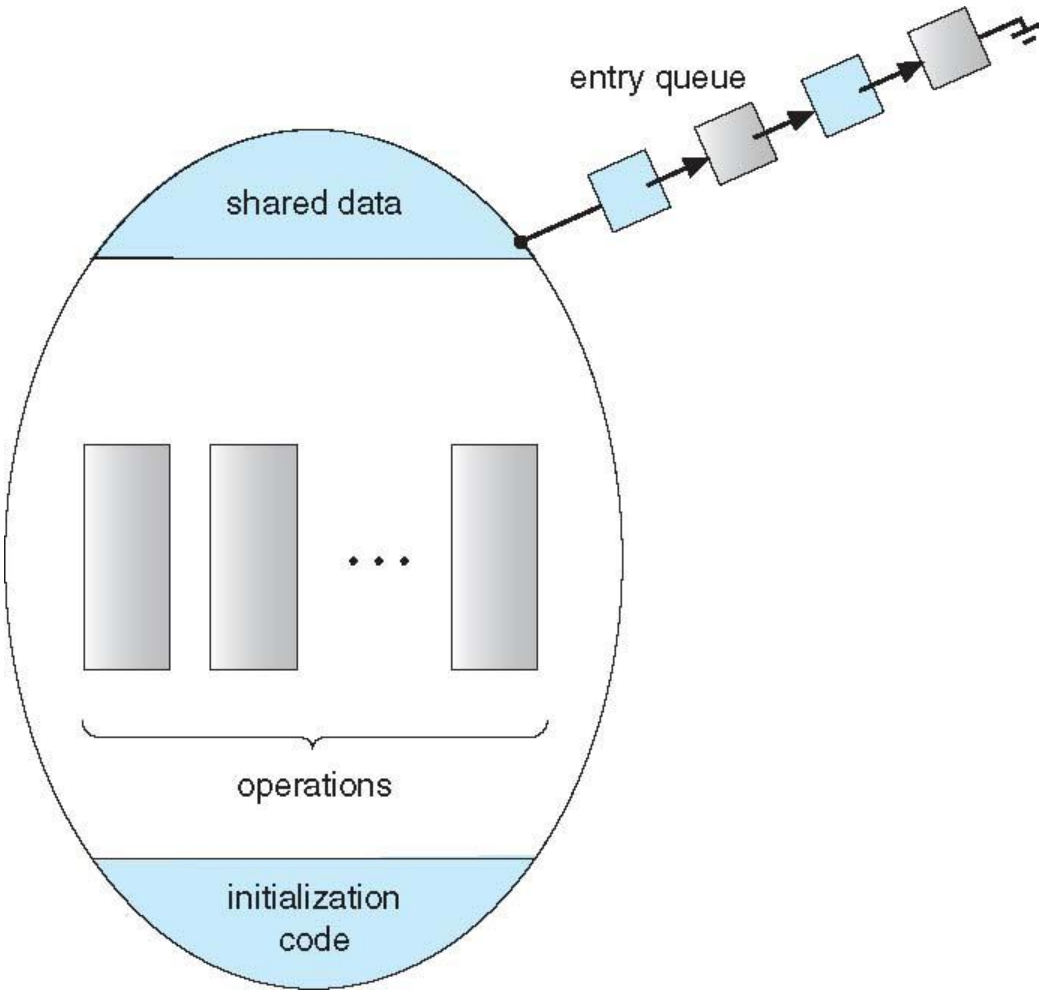
// shared variable declarations

function P1 (…) { … }

…

```
    function Pn (…) {……}
  initialization_code (…) { … }
}
```

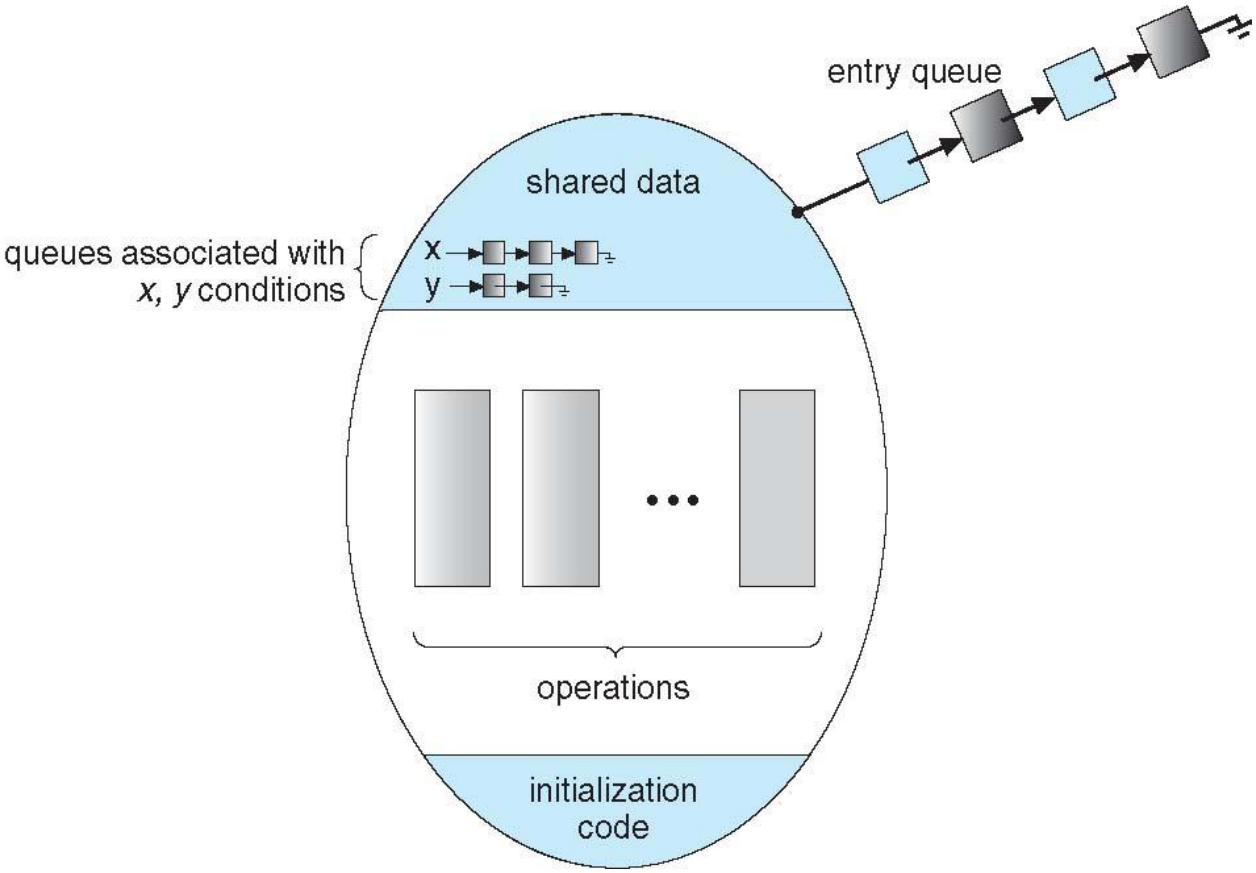
- Recall that: an *Abstract data type (ADT)* -- encapsulates data
- internal variables only accessible by code within the procedure
- A monitor type – is an ADT that provides a convenient and effective mechanism for process synchronization
- For several concurrent processes
- Encapsulation
  - Local variables accessed only via local functions
  - Local functions access only local vars and params
- Only one process at a time may be active within the monitor
- A lock guards shared data
- Hence, the programmer does not need to code this constraint



- Monitors not powerful enough to model some synchr. schemes
- Hence, additional synchr. mechanisms are added:

  Provided by the **condition** construct

  condition x, y;
- Two operations are allowed on a condition variable **x**:
- **x.wait()**
  - A process that invokes the operation is suspended (sleeps)
    - until **x.signal()**is called
  - Lets other processes enter the monitor
    - releases the lock to shared data, atomically with sleep
- **x.signal()**
  - Resumes one of processes (if any) that invoked **x.wait()**
  - If no **x.wait()** was called, then **x.signal()**has no effect **x**



Issues with monitors: assume

- Process P invokes **x.signal(),** and
- Process Q is suspended in **x.wait()**

Who proceeds next?

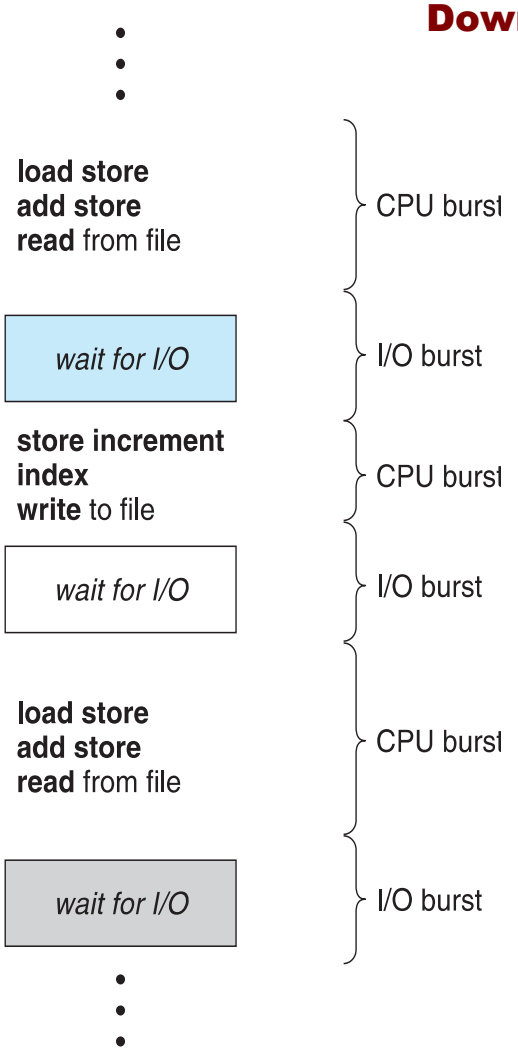
- Both Q and P cannot execute in parallel
- Because they are within a monitor

Options include

- **Signal and wait** – P waits until Q either leaves the monitor or it waits for another condition
- **Signal and continue** – Q waits until P either leaves the monitor or it waits for another condition
- Both have pros and cons – language implementer can decide

CPU SCHEDULING

- Maximum CPU utilization obtained with multiprogramming
- waiting for I/O is wasteful
- 1 thread will utilize only 1 core
- CPU–I/O Burst Cycle
- Process execution consists of:
  - a **cycle** of CPU execution
  - and I/O wait
- **CPU burst** followed by **I/O burst**
- CPU burst distribution is of main concern



SCHEDULING LEVELS

High-Level Scheduling

- See Long-term scheduler or Job Scheduling from Chapter 3
- Selects jobs allowed to compete for CPU and other system resources.

Intermediate-Level Scheduling

- See Medium-Term Scheduling from Chapter 3
- Selects which jobs to temporarily suspend/resume to smooth fluctuations in system load.

Low-Level (CPU) Scheduling or Dispatching

- Selects the ready process that will be assigned the CPU.
- Ready Queue contains PCBs of processes.
- **Short-term scheduler**
- Selects 1 process from the ready queue

- then allocates the CPU to it
- Queue may be ordered in various ways
- CPU scheduling decisions may take place when a process:
- Switches from running to waiting state
- Switches from running to ready state
- Switches from waiting to ready
- Terminates
- Scheduling under 1 and 4 is called **nonpreemptive (=cooperative)**
- All other scheduling is called **preemptive**
- Process can be interrupted and must release the CPU
- Special care should be taken to prevent problems that can arise
- Access to shared data – race condition can happen, if not handled

Dispatcher

- a module that gives control of the CPU to the process selected by the short-term scheduler; this involves:
  - switching context
  - switching to user mode
  - jumping to the proper location in the user program to restart that program

Dispatch latency

- Time it takes for the dispatcher to stop one process and start another running
- This time should be as small as possible

- How do we decide which scheduling algorithm is good?
- Many criteria for judging this has been suggested
  - Which characteristics considered can change significantly which algo is considered the best
- **CPU utilization** – keep the CPU as busy as possible
- **Throughput** – # of processes that complete their execution per time unit

- **Turnaround time** – amount of time to execute a particular process
- **Waiting time** – amount of time a process has been waiting in the ready queue
- **Response time** – amount of time it takes to stat responding
    - Used for interactive systems
    - Time from when a request was submitted until the first response is produced
- Max CPU utilization
- Max throughput
- Min turnaround time
- Min waiting time
- Min response time

Process	Burst Time
P_1	24
P_2	3
P_3	3

Suppose that the processes arrive in the order: P_1 , P_2 , P_3
The Gantt Chart for the schedule is:

Waiting time for P_1 = 0; P_2 = 24; P_3 = 27

Average waiting time: (0 + 24 + 27)/3 = 17

FCFS

Suppose that the processes arrive in the order:

P_2 , P_3 , P_1

- The Gantt chart for the schedule is:
- Waiting time for P1 = 6; P2 = 0; P3 = 3
- Average waiting time: (6 + 0 + 3)/3 = 3
- Much better than previous case
- Hence, average waiting time of FCFS not minimal

- And it may vary substantially
- FCFS is nonpreemptive
- Not a good idea for timesharing systems
- FCFS suffers from the convoy effect,

- **Convoy effect** – when several short processes wait for long a process to get off the CPU
- Assume
    - 1 long CPU-bound process
    - Many short I/O-bound processes
- Execution:
    - The long one occupies CPU
        - The short ones wait for it: no I/O is done at this stage
        - No overlap of I/O with CPU utilizations
    - The long one does its first I/O
        - Releases CPU
        - Short ones are scheduled, but do I/O, release CPU quickly
    - The long one occupies CPU again, etc
- Hence low CPU and device utilization

SJF

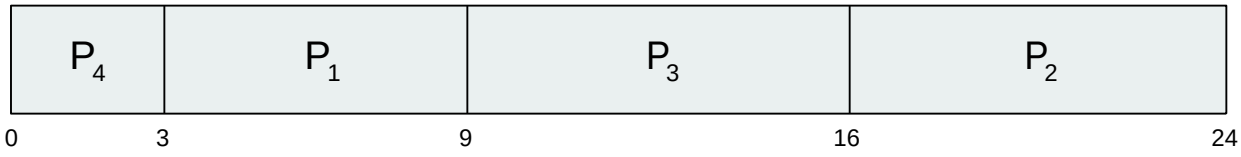
- Associate with each process the length of its next CPU burst
    - SJF uses these lengths to schedule the process with the shortest time
- Notice, the burst is used by SJF,
    - **not** the process end-to-end running time
        - implied by word “job” in SJF
    - Hence, it should be called ``Shorted-Next-CPU-Burst”
    - However, “job” is used for historic reasons
- Two versions of SJF: preemptive and nonpreemptive

- Assume
  - A new process Pnew arrives while the current one Pcur is still executing
  - The burst of Pnew is less than what is left of Pcur
- Nonpreemptive SJF – will let Pcur finish
- Preemptive SJF – wil preempt Pcur and let Pnew execute
  - This is also called shortest-remaining-time-first scheduling

- **Advantage:**
  - SJF is **optimal** in terms of the average waiting time
- **Challenge of SJF:**
  - Hinges on knowing the length of the next CPU burst
    - But how can we know it?
    - Solutions: ask user or estimate it
  - In a batch system and long-term scheduler
    - Could ask the user for the job time limit
    - The user is motivated to accurately estimate it
      - Lower value means faster response
      - Too low a value will cause time-limit violation and job rescheduling
  - In a short-term scheduling
    - Use estimation

Process	Arrival Time	Burst Time
P_1	0.0	6
P_2	2.0	8
P_3	4.0	7
P_4	5.0	3

SJF scheduling chart

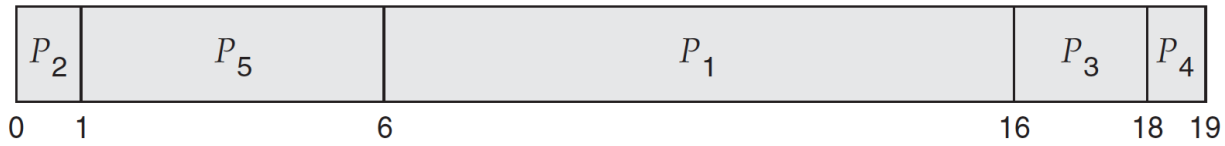


Average waiting time = (3 + 16 + 9 + 0) / 4 = 7

PRIORITY

- A priority number (integer) is associated with each process
- The CPU is allocated to the process with the highest priority (smallest integer ≡ highest priority)
    - Preemptive
    - Nonpreemptive
    - SJF is priority scheduling where priority is the inverse of predicted next CPU burst time
- Problem ≡ **Starvation** – low priority processes may never execute
- Solution ≡ **Aging** – as time progresses increase the priority of the process

Process	Burst Time	Priority
P_1	10	3
P_2	1	1
P_3	2	4
P_4	1	5
P_5	5	2

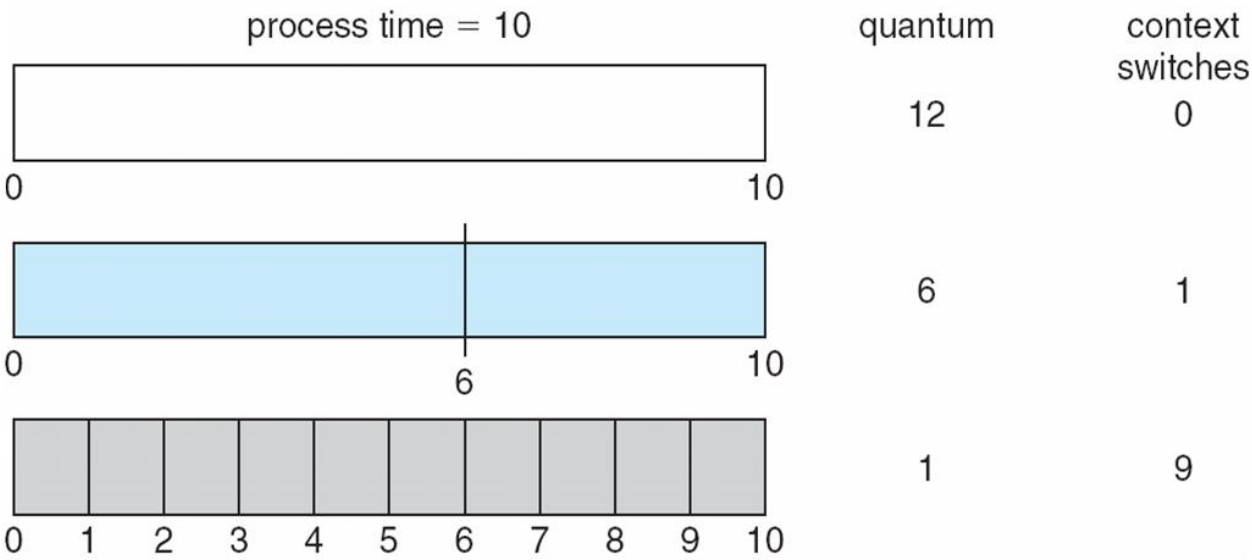


Average waiting time = 8.2 msec

ROUND ROBIN

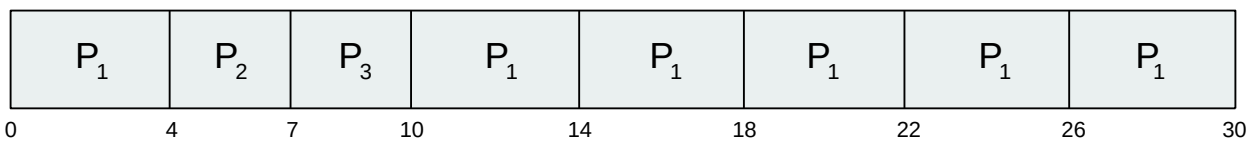
- Each process gets a small unit of CPU time
    - Time quantum q
    - Usually 10-100 milliseconds
- After this time has elapsed:
    - the process is preempted and

- added to the end of the ready queue
- The process might run for ≤ q time
  - For example, when it does I/O
- If
  - n processes in the ready queue, and
  - the time quantum is q
- then
  - "Each process gets 1/n of the CPU time"
    - Incorrect statement from the textbook
  - in chunks of ≤ q time units at once
  - Each process waits ≤ (n-1)q time units



Process	Burst Time
P_1	24
P_2	3
P_3	3

The Gantt chart is:



Deadlock – two or more processes are waiting indefinitely for an event that can be caused by only one of the waiting processes

Example: Let ***S*** and ***Q*** be two semaphores initialized to 1

P_0	P_1
wait(S);	**wait(Q);**
wait(Q);	**wait(S);**
...	**...**
signal(S);	**signal(Q);**
signal(Q);	**signal(S);**

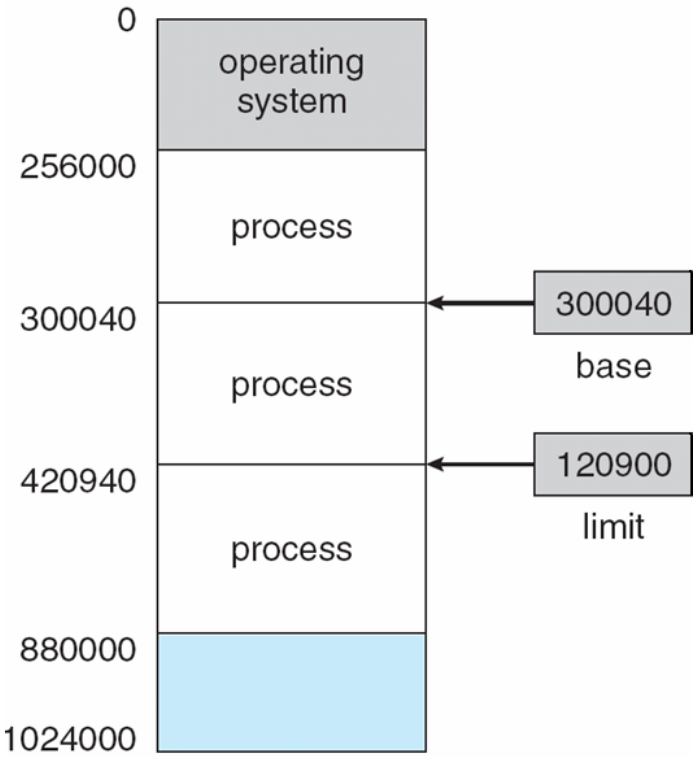
- System consists of resources
- Resource types $R_1, R_2, \ldots, R_m$
- *CPU cycles, files, memory space, I/O devices, etc*
- Each resource type R_i has W_i instances.
    - Type: CPU
        - 2 instances - CPU1, CPU2
    - Type: Printer
        - 3 instances - printer1, printer2, printer3
- Each process utilizes a resource as follows:
    - **Request** resource
        - A process can request a resource of a given type
            - E.g., "I request any printer"
        - System will then assign a instance of that resource to the process
            - E.g., some printer will be assigned to it
        - If cannot be granted immediately, the process must wait until it can get it
    - **Use** resource
        - Operate on the resource, e.g. print on the printer
    - **Release** resource
- Mutexes and Semaphores

- Special case:
  - Each mutex or semaphor is treated as a separate resource **type**
  - Because a process would want to get not just "any" lock among a group of locks, but a specific lock that guards a specific shared data type
    - e.g., lock that guards a specific queue

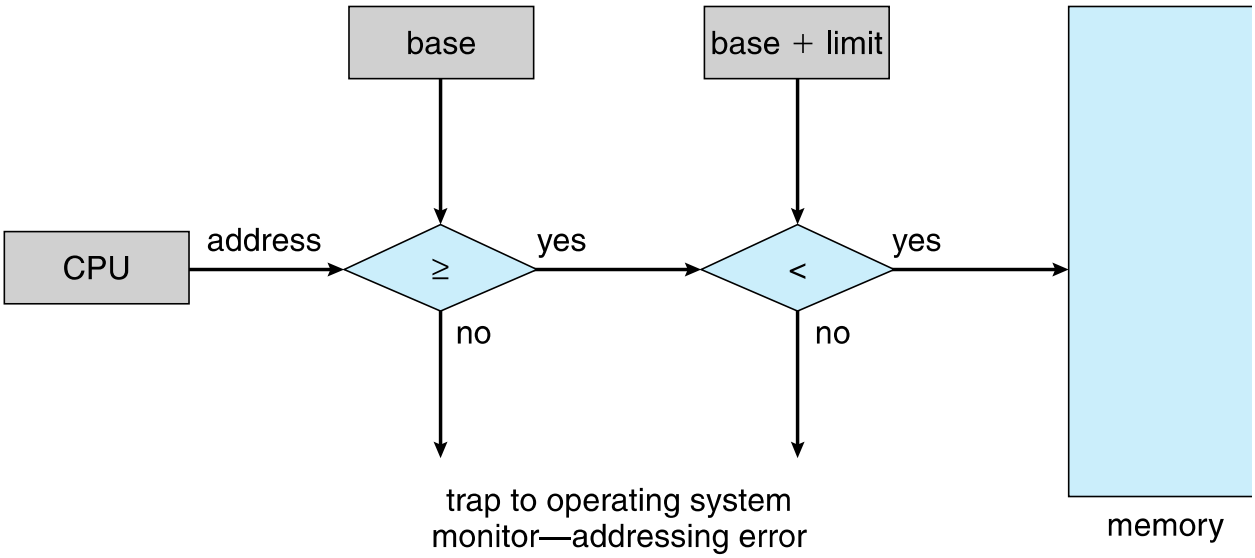
- **Mutual exclusion:** only one process at a time can use a resource
- **Hold and wait:** a process holding at least one resource is waiting to acquire additional resources held by other processes
- **No preemption:** a resource can be released only voluntarily by the process holding it, after that process has completed its task
- **Circular wait:** there exists a set $\{P_0, P_1, \ldots, P_n\}$ of waiting processes such that
  - P_0 is waiting for a resource that is held by P_1, and so on:
  - $P_0 \rightarrow P_1 \rightarrow P_2 \rightarrow \ldots \rightarrow P_{n-1} \rightarrow P_n \rightarrow P_0$
- Notice: "Circular wait" implies "Hold and Wait"
  - Why then not test for only the "Circular wait"?
  - Because, computationally, "Hold and wait" can be tested much more efficiently than "Circular wait"
  - Some algorithms we consider only need to check H&W
- Deadlocks can occur via system calls, locking, etc.
- Trivial example
  - Mutexes A, B – unlocked initially
  - Process P1: wait(A); wait(B);
  - Process P2: wait(B); wait(A);

MAIN MEMORY

- Program must be brought (from disk) into memory and placed within a process for it to be run
- Main memory and registers are only storage CPU can access directly
- Memory unit only sees a stream of addresses + read requests, or address + data and write requests
- Register access in one CPU clock (or less)
- Main memory can take many cycles, causing a **stall**
- **Cache** sits between main memory and CPU registers
- Protection of memory required to ensure correct operation
- A pair of **base** and **limit registers** define the logical address space
- CPU must check every memory access generated in user mode to be sure it is between base and limit for that user



HARDWARE PROTECTION

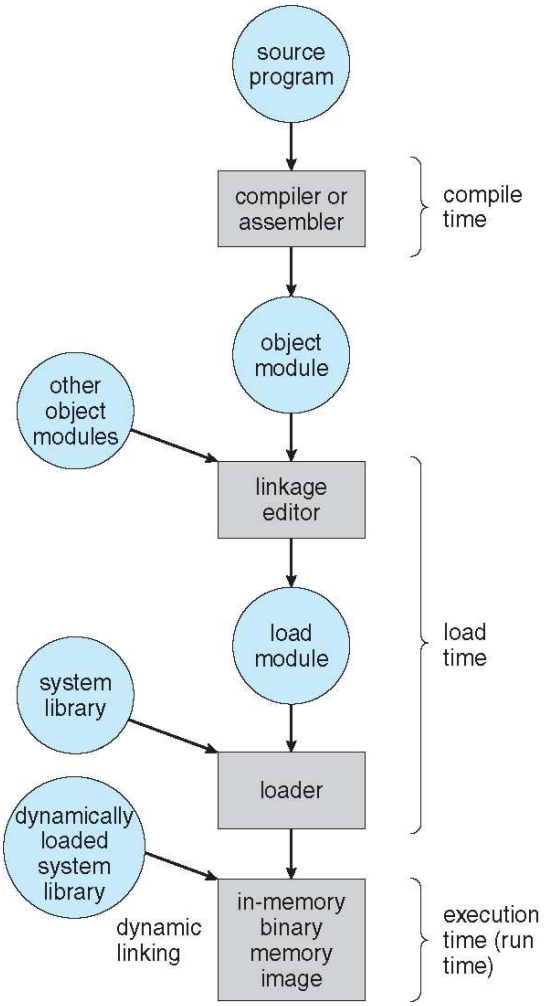


ADDRESS BINDING

- Programs on disk, ready to be brought into memory to execute form an **input queue**
    - Without support, must be loaded into address 0000
- Inconvenient to have first user process physical address always at 0000
    - How can it not be?
- Further, addresses represented in different ways at different stages of a program's life
    - Source code addresses usually symbolic
    - Compiled code addresses **bind** to relocatable addresses
        - i.e. "14 bytes from beginning of this module"
    - Linker or loader will bind relocatable addresses to absolute addresses
        - i.e. 74014
    - Each binding maps one address space to another
- Address binding of instructions and data to memory addresses can happen at three different stages
    - **Compile time**: If memory location known a priori, **absolute code** can be generated; must recompile code if starting location changes

- **Load time**: Must generate **relocatable code** if memory location is not known at compile time
- **Execution time**: Binding delayed until run time if the process can be moved during its execution from one memory segment to another
    - Need hardware support for address maps (e.g., base and limit registers)

MULTISTEP PROCESSING



LOGICAL VS PHYSICAL ADDRESS

- The concept of a logical address space that is bound to a separate **physical address space** is central to proper memory management
    - **Logical address** – generated by the CPU; also referred to as **virtual address**
    - **Physical address** – address seen by the memory unit
- Logical and physical addresses are the same in compile-time and load-time address-binding schemes; logical (virtual) and physical addresses differ in execution-time address-binding scheme
- **Logical address space** is the set of all logical addresses generated by a program

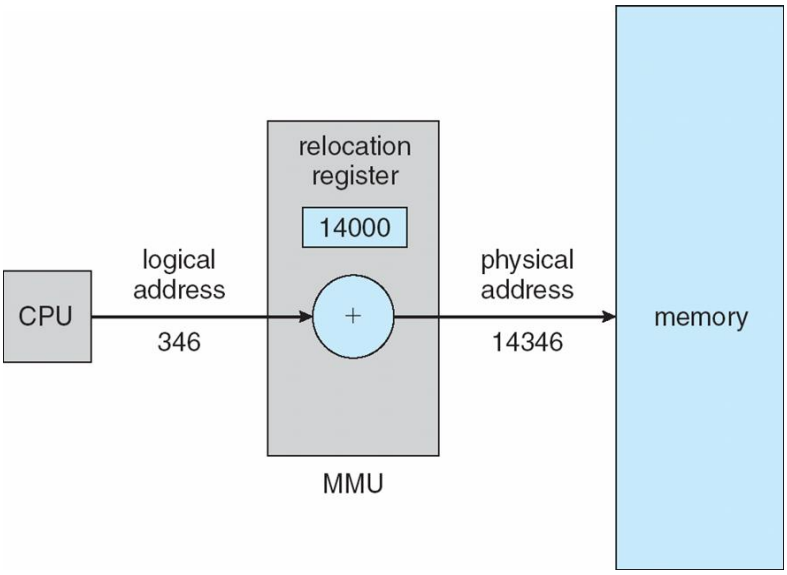
- **Physical address space** is the set of all physical addresses generated by a program

MMU

- Hardware device that at run time maps virtual to physical address
- Many methods possible, covered in the rest of this chapter
- To start, consider simple scheme where the value in the relocation register is added to every address generated by a user process at the time it is sent to memory
    - Base register now called **relocation register**
    - MS-DOS on Intel 80x86 used 4 relocation registers
- The user program deals with *logical* addresses; it never sees the *real* physical addresses
    - Execution-time binding occurs when reference is made to location in memory
    - Logical address bound to physical addresses

DYNAMIC ALLOCATION

- Routine is not loaded until it is called
- Better memory-space utilization; unused routine is never loaded
- All routines kept on disk in relocatable load format
- Useful when large amounts of code are needed to handle infrequently occurring cases
- No special support from the operating system is required
    - Implemented through program design
    - OS can help by providing libraries to implement dynamic loading

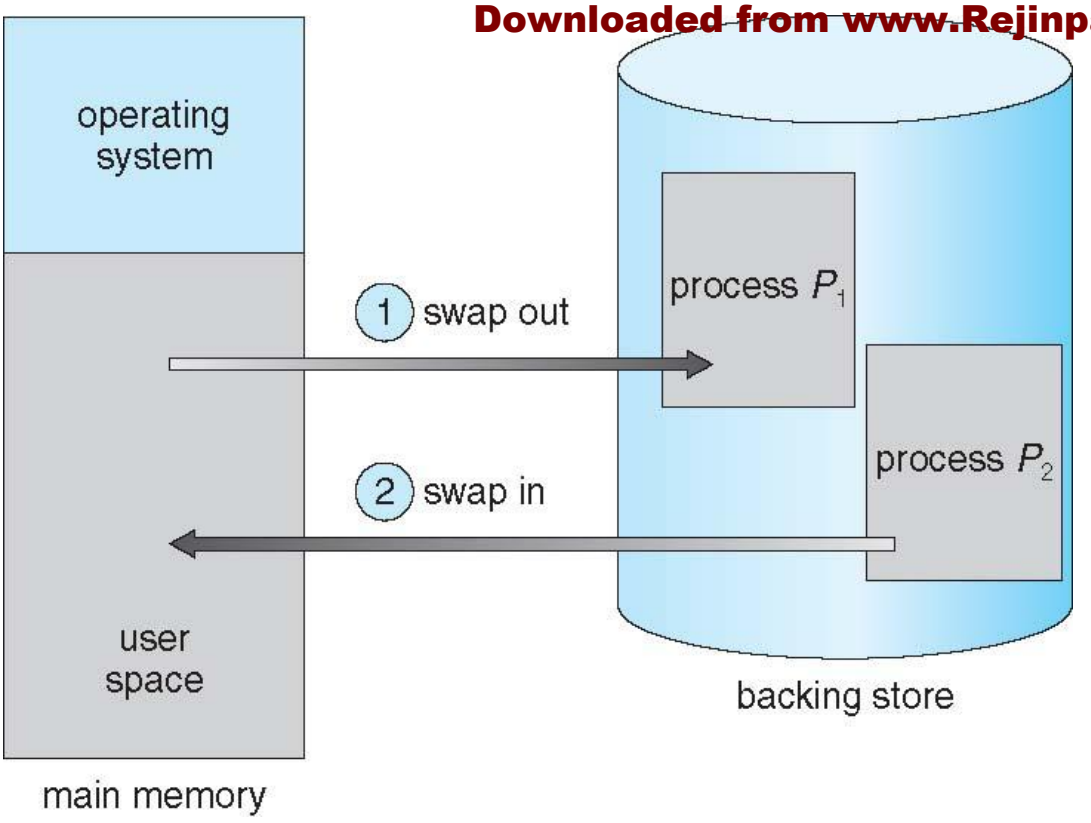


DYNAMIC LINKING

- **Static linking** – system libraries and program code combined by the loader into the binary program image
- Dynamic linking –linking postponed until execution time
- Small piece of code, **stub**, used to locate the appropriate memory-resident library routine
- Stub replaces itself with the address of the routine, and executes the routine
- Operating system checks if routine is in processes' memory address
    - If not in address space, add to address space
- Dynamic linking is particularly useful for libraries
- System also known as **shared libraries**
- Consider applicability to patching system libraries
    - Versioning may be needed

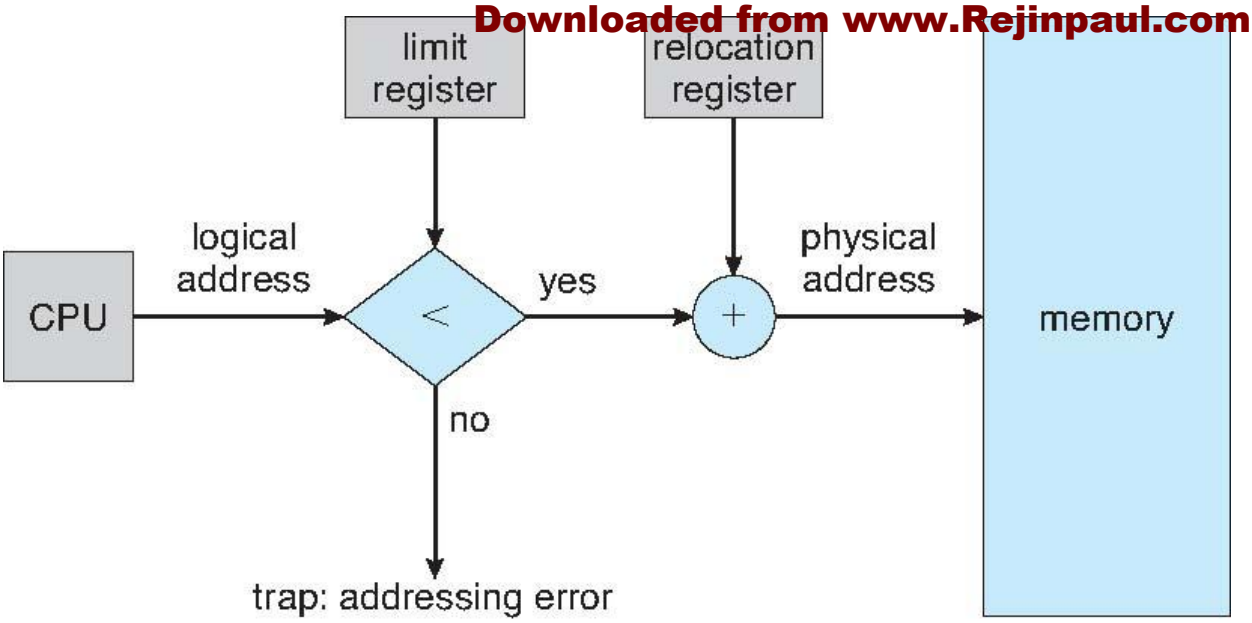
SWAPPING

- A process can be **swapped** temporarily out of memory to a backing store, and then brought back into memory for continued execution
    - Total physical memory space of processes can exceed physical memory
- **Backing store** – fast disk large enough to accommodate copies of all memory images for all users; must provide direct access to these memory images
- **Roll out, roll in** – swapping variant used for priority-based scheduling algorithms; lower-priority process is swapped out so higher-priority process can be loaded and executed
- Major part of swap time is transfer time; total transfer time is directly proportional to the amount of memory swapped
- System maintains a **ready queue** of ready-to-run processes which have memory images on disk
- Does the swapped out process need to swap back in to same physical addresses?
- Depends on address binding method
    - Plus consider pending I/O to / from process memory space
- Modified versions of swapping are found on many systems (i.e., UNIX, Linux, and Windows)
    - Swapping normally disabled
    - Started if more than threshold amount of memory allocated
    - Disabled again once memory demand reduced below threshold



CONTIGUOUS ALLOCATION

- Main memory must support both OS and user processes
- Limited resource, must allocate efficiently
- Contiguous allocation is one early method
- Main memory usually into two **partitions**:
    - Resident operating system, usually held in low memory with interrupt vector
    - User processes then held in high memory
    - Each process contained in single contiguous section of memory
- Relocation registers used to protect user processes from each other, and from changing operating-system code and data
    - Base register contains value of smallest physical address
    - Limit register contains range of logical addresses – each logical address must be less than the limit register
    - MMU maps logical address *dynamically*
    - Can then allow actions such as kernel code being **transient** and kernel changing size



FRAGMENTATION

- **External Fragmentation** – total memory space exists to satisfy a request, but it is not contiguous
- **Internal Fragmentation** – allocated memory may be slightly larger than requested memory; this size difference is memory internal to a partition, but not being used
- First fit analysis reveals that given *N* blocks allocated, 0.5 *N* blocks lost to fragmentation
    - 1/3 may be unusable -> **50-percent rule**
- Reduce external fragmentation by **compaction**
    - Shuffle memory contents to place all free memory together in one large block
    - Compaction is possible *only* if relocation is dynamic, and is done at execution time
    - I/O problem
        - Latch job in memory while it is involved in I/O
        - Do I/O only into OS buffers
- Now consider that backing store has same fragmentation problems

SEGMENTATION

Memory-management scheme that supports user view of memory

A program is a collection of segments

A segment is a logical unit such as:

main program

procedure

function

method

object

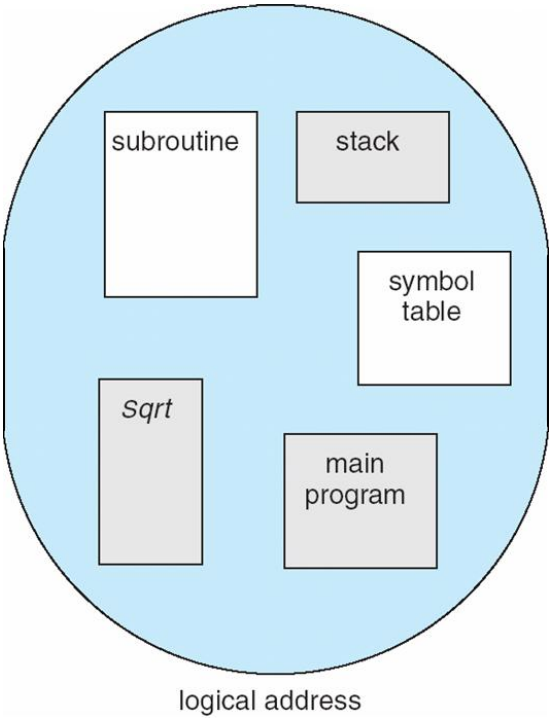
local variables, global variables

common block

stack

symbol table

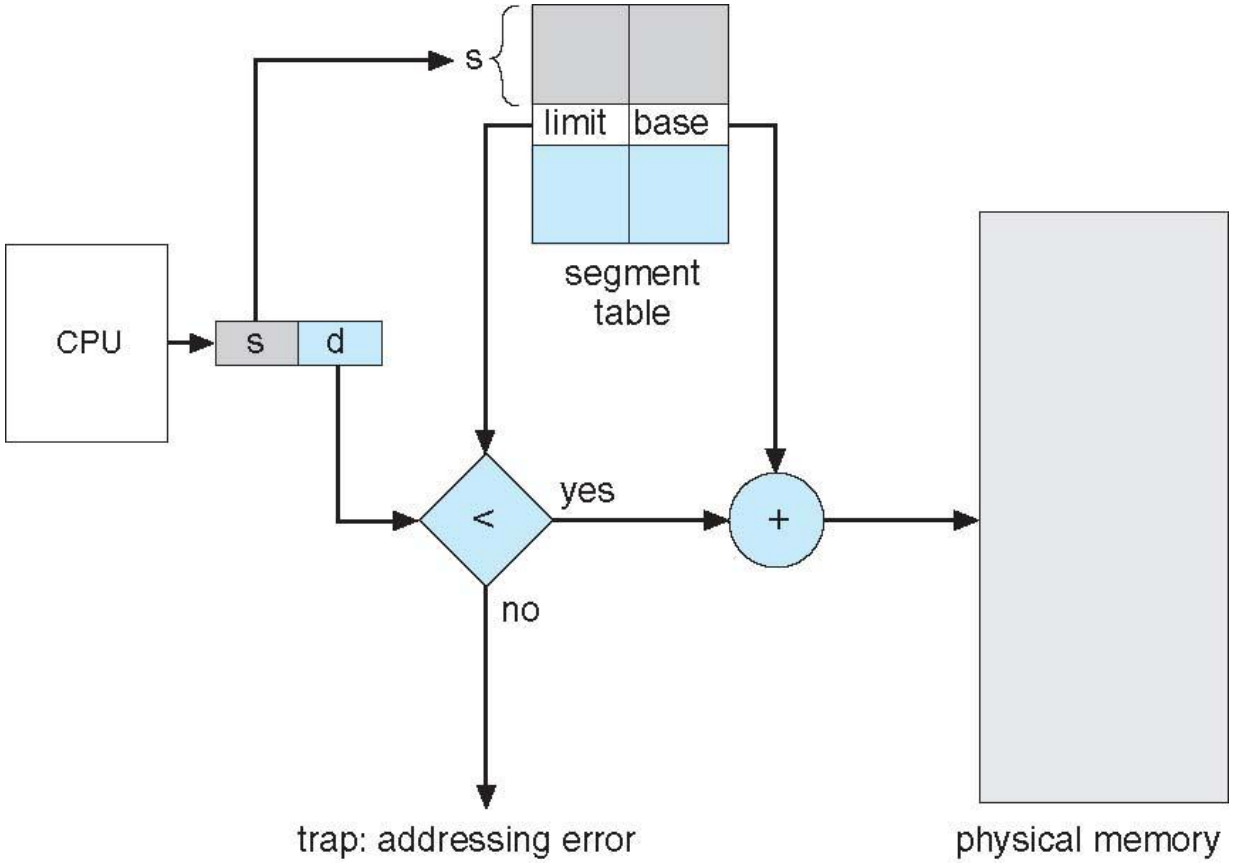
arrays



- Logical address consists of a two tuple:
    - <segment-number, offset>,
- **Segment table** – maps two-dimensional physical addresses; each table entry has:
    - **base** – contains the starting physical address where the segments reside in memory
    - **limit** – specifies the length of the segment
    - **Segment-table base register (STBR)** points to the segment table's location in memory
- **Segment-table length register (STLR)** indicates number of segments used by a program;

    - segment number **s** is legal if **s** < **STLR**

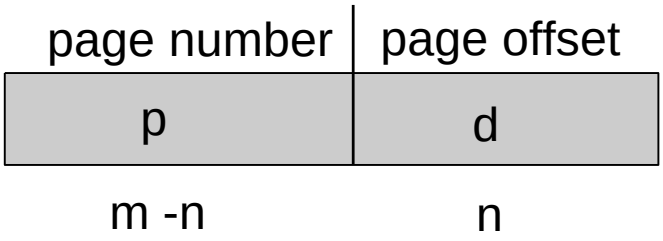
- Protection
    - With each entry in segment table associate:
        - validation bit = 0 ⇒ illegal segment
        - read/write/execute privileges
- Protection bits associated with segments; code sharing occurs at segment level
- Since segments vary in length, memory allocation is a dynamic storage-allocation problem
- A segmentation example is shown in the following diagram



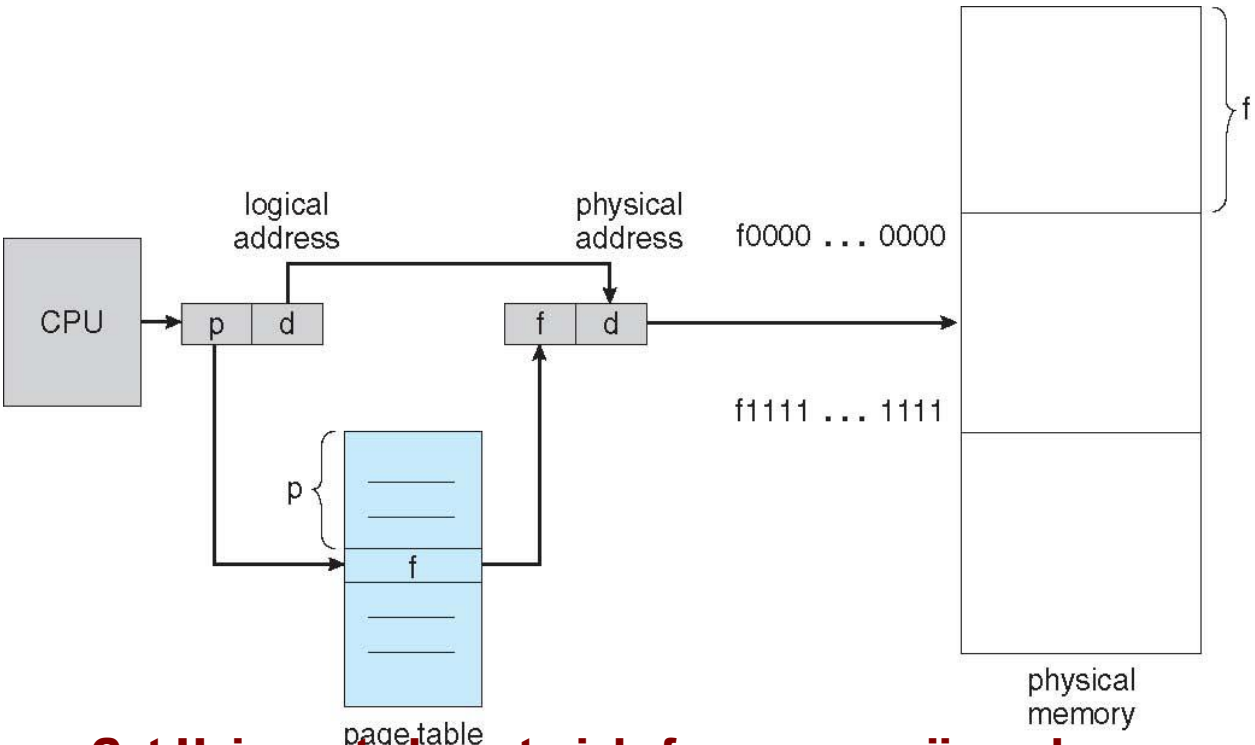
PAGING

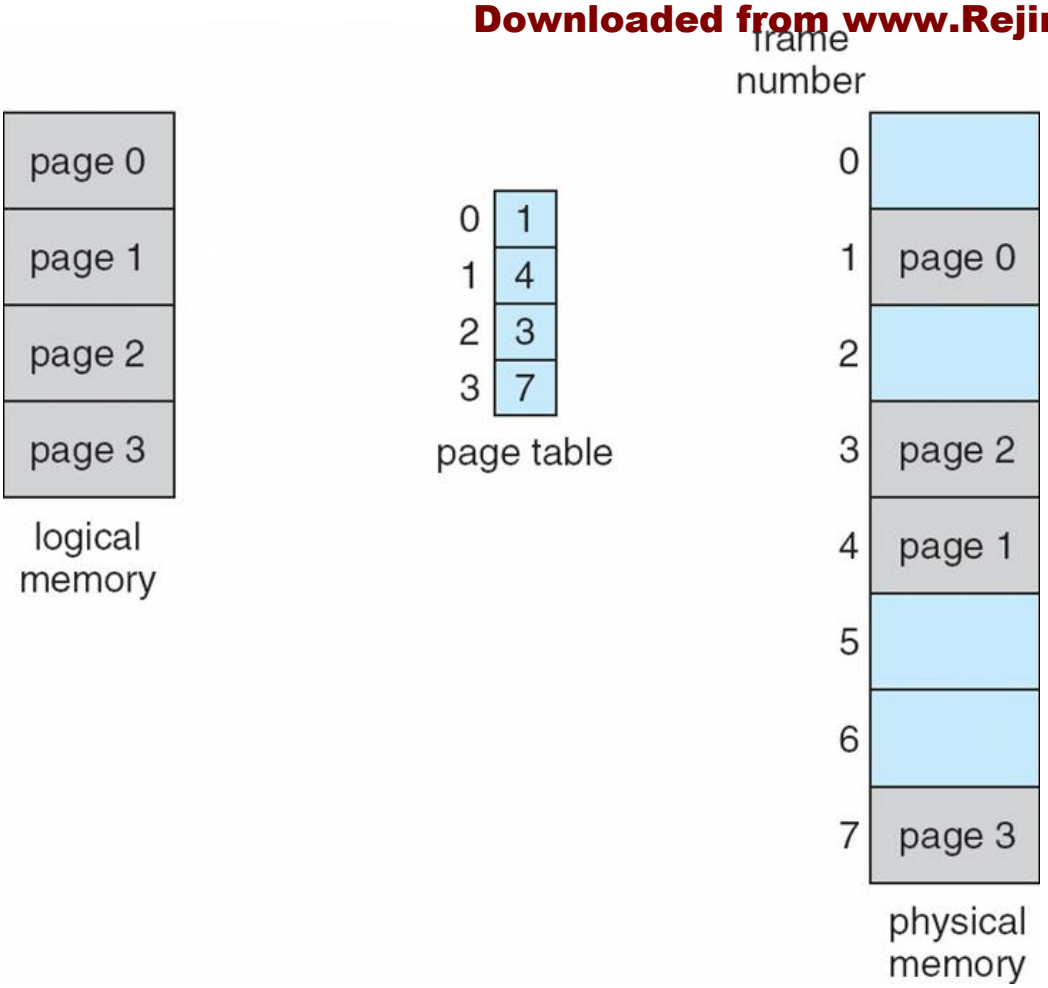
- Physical address space of a process can be noncontiguous; process is allocated physical memory whenever the latter is available
    - Avoids external fragmentation
    - Avoids problem of varying sized memory chunks
- Divide physical memory into fixed-sized blocks called **frames**
    - Size is power of 2, between 512 bytes and 16 Mbytes

- Divide logical memory into blocks of same size called **pages**
- Keep track of all free frames
- To run a program of size ***N*** pages, need to find ***N*** free frames and load program
- Set up a **page table** to translate logical to physical addresses
- Backing store likewise split into pages
- Still have Internal fragmentation
- Address generated by CPU is divided into:
    - **Page number** (***p***) – used as an index into a **page table** which contains base address of each page in physical memory
    - **Page offset** (***d***) – combined with base address to define the physical memory address that is sent to the memory unit
    - For given logical address space 2^m and page size 2^n

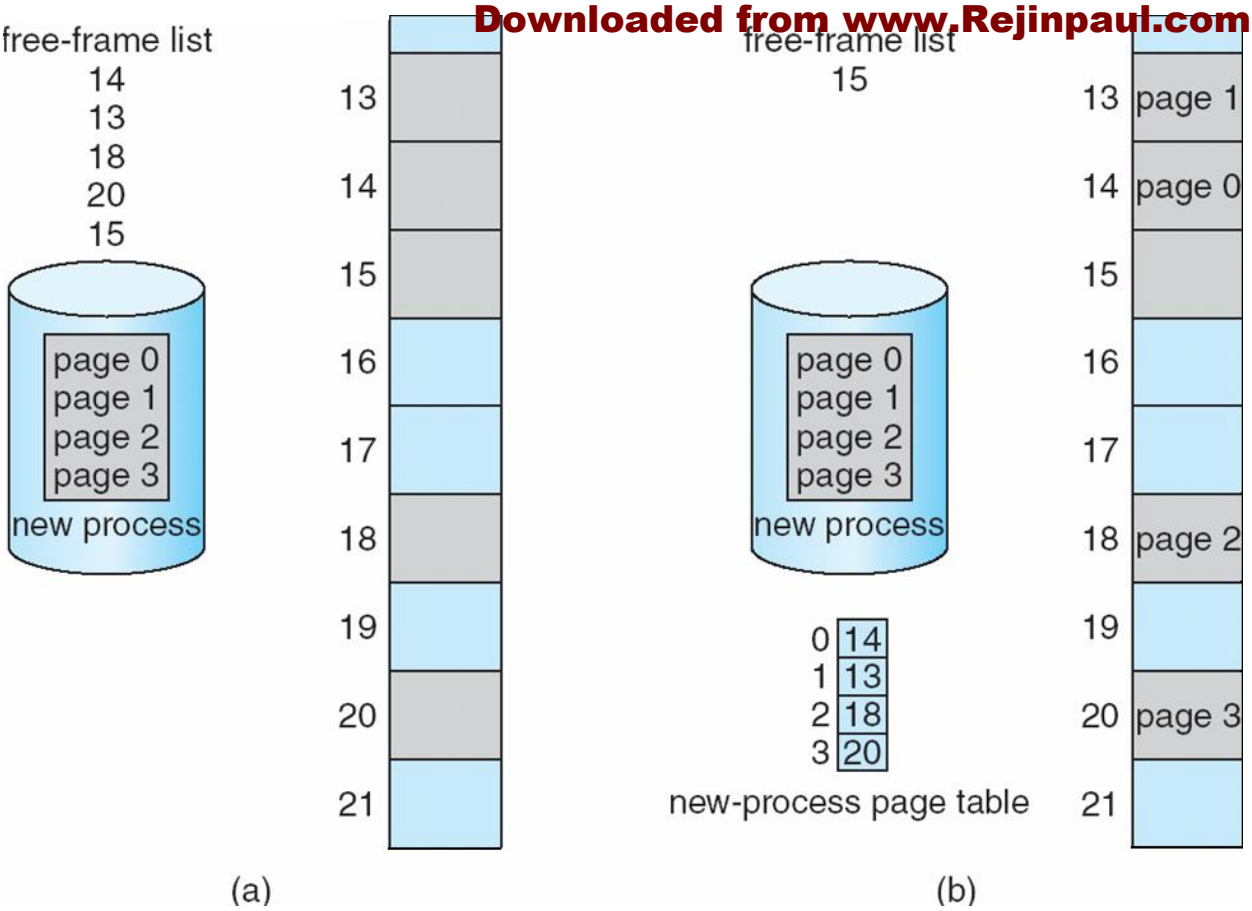


PAGING STRUCTURE





- Calculating internal fragmentation
  - Page size = 2,048 bytes
  - Process size = 72,766 bytes
  - 35 pages + 1,086 bytes
  - Internal fragmentation of 2,048 - 1,086 = 962 bytes
  - Worst case fragmentation = 1 frame – 1 byte
  - On average fragmentation = 1 / 2 frame size
  - So small frame sizes desirable?
  - But each page table entry takes memory to track
  - Page sizes growing over time
    - Solaris supports two page sizes – 8 KB and 4 MB
- Process view and physical memory now very different
- By implementation process can only access its own memory



- Page table is kept in main memory
- **Page-table base register** (**PTBR**) points to the page table
- **Page-table length register** (**PTLR**) indicates size of the page table
- In this scheme every data/instruction access requires two memory accesses
    - One for the page table and one for the data / instruction
- The two memory access problem can be solved by the use of a special fast-lookup hardware cache called **associative memory** or **translation look-aside buffers** (**TLBs**)
- Some TLBs store **address-space identifiers** (**ASIDs**) in each TLB entry – uniquely identifies each process to provide address-space protection for that process
    - Otherwise need to flush at every context switch
- TLBs typically small (64 to 1,024 entries)
- On a TLB miss, value is loaded into the TLB for faster access next time
    - Replacement policies must be considered
    - Some entries can be **wired down** for permanent fast access

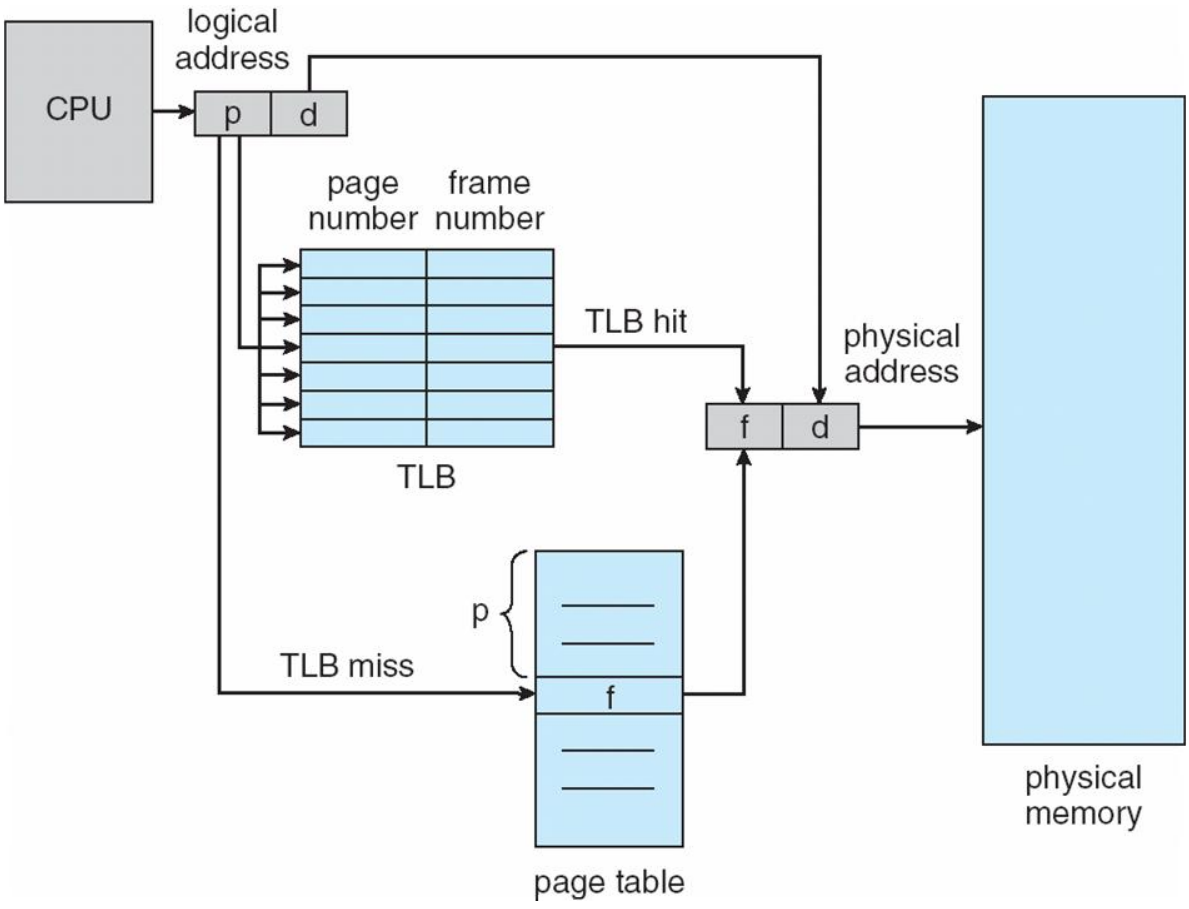
ASSOCIATIVE MEMORY

- Associative memory – parallel search

- Address translation (p, d)
    - If p is in associative register, get frame # out
    - Otherwise get frame # from page table in memory

Page #	Frame #

PAGING WITH TLB



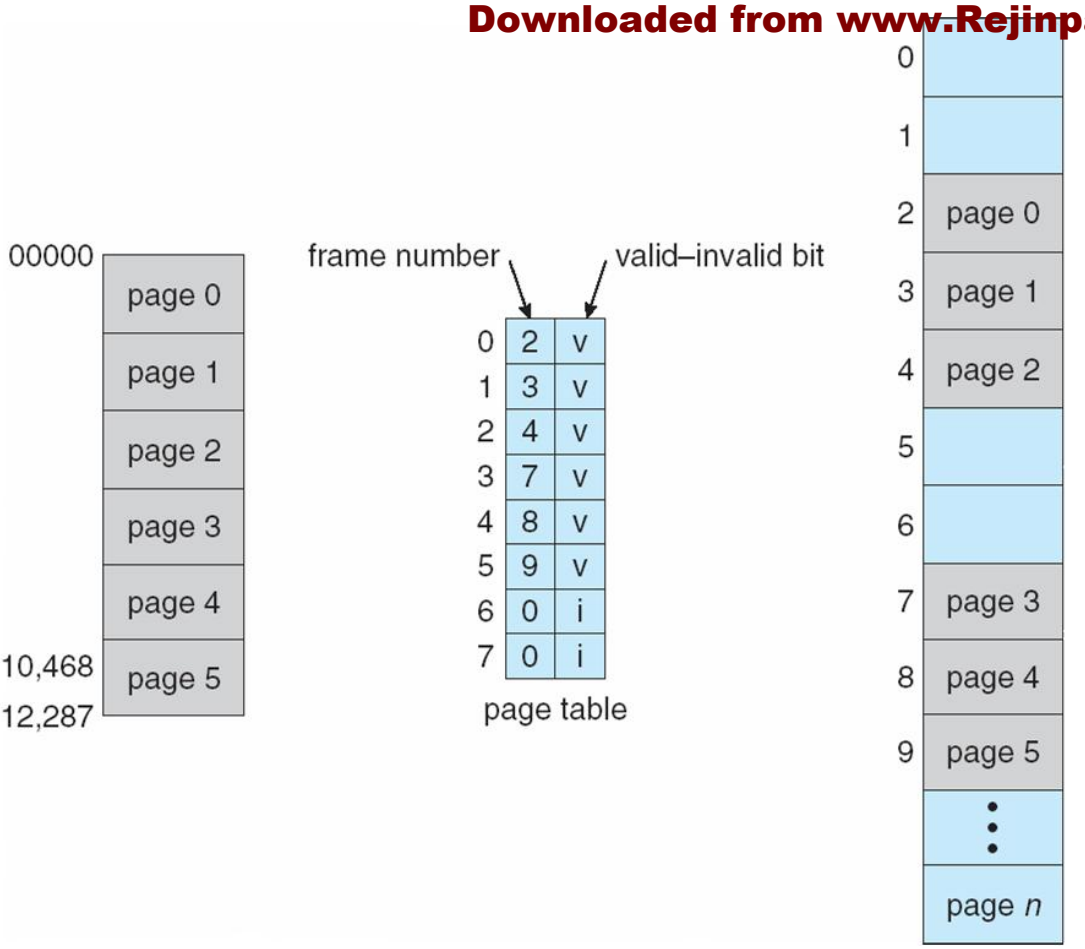
- Associative Lookup = ε time unit
    - Can be < 10% of memory access time
- Hit ratio = α
    - Hit ratio – percentage of times that a page number is found in the associative registers; ratio related to number of associative registers

- Consider α = 80%, ε = 20ns for TLB search, 100ns for memory access
- **Effective Access Time** (**EAT**)
    - EAT = $(1 + \varepsilon)\ \alpha + (2 + \varepsilon)(1 - \alpha)$
        - $= 2 + \varepsilon - \alpha$
- Consider α = 80%, ε = 20ns for TLB search, 100ns for memory access
    - EAT = 0.80 x 100 + 0.20 x 200 = 120ns
- Consider more realistic hit ratio -> α = 99%, ε = 20ns for TLB search, 100ns for memory access
    - EAT = 0.99 x 100 + 0.01 x 200 = 101ns

MEMORY PROTECTION

- Memory protection implemented by associating protection bit with each frame to indicate if read-only or read-write access is allowed
    - Can also add more bits to indicate page execute-only, and so on
- **Valid-invalid** bit attached to each entry in the page table:
    - "valid" indicates that the associated page is in the process' logical address space, and is thus a legal page
    - "invalid" indicates that the page is not in the process' logical address space
    - Or use **page-table length register** (**PTLR**)
- Any violations result in a trap to the kernel

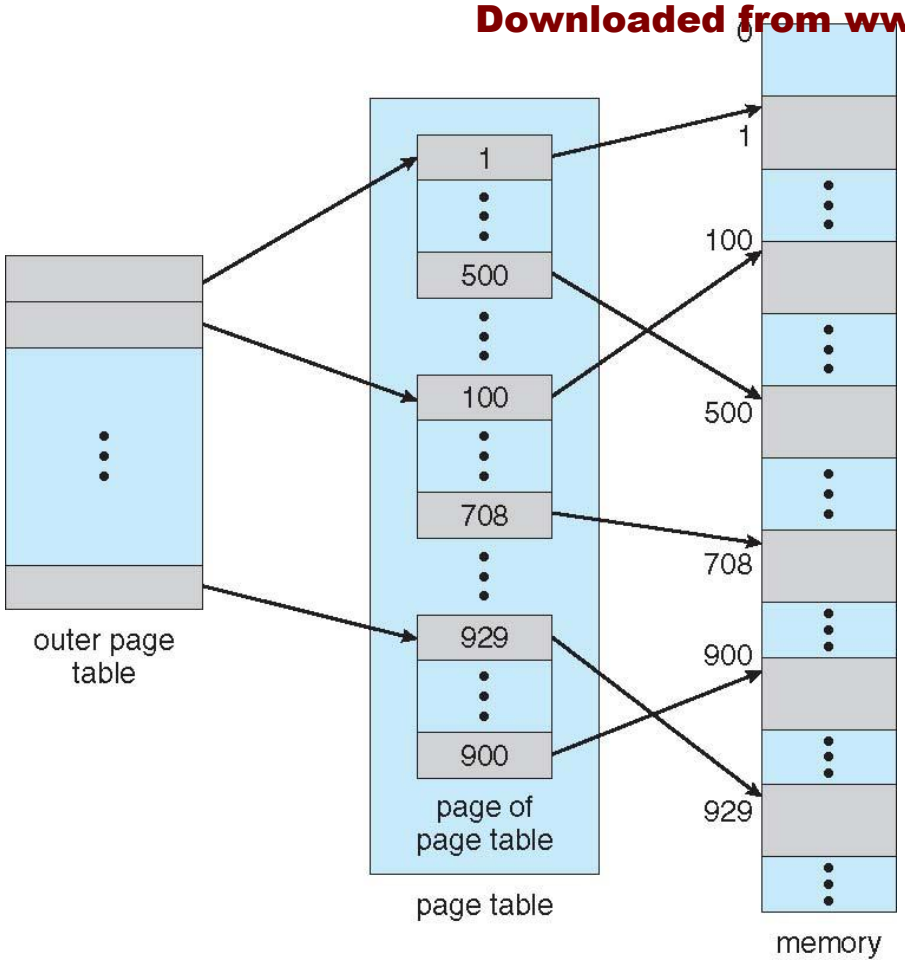
VALID OR INVALID BIT IN PAGE TABLE



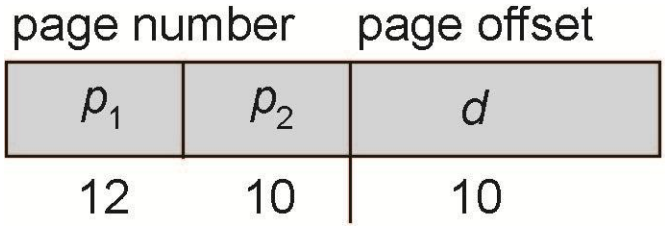
STRUCTURE OF THE PAGE TABLE

- Memory structures for paging can get huge using straight-forward methods
    - Consider a 32-bit logical address space as on modern computers
    - Page size of 4 KB (2^{12})
    - Page table would have 1 million entries ($2^{32} / 2^{12}$)
    - If each entry is 4 bytes -> 4 MB of physical address space / memory for page table alone
        - That amount of memory used to cost a lot
        - Don’t want to allocate that contiguously in main memory
- Hierarchical Paging
- Hashed Page Tables
- Inverted Page Tables

TWO LEVEL PAGE SCHEME

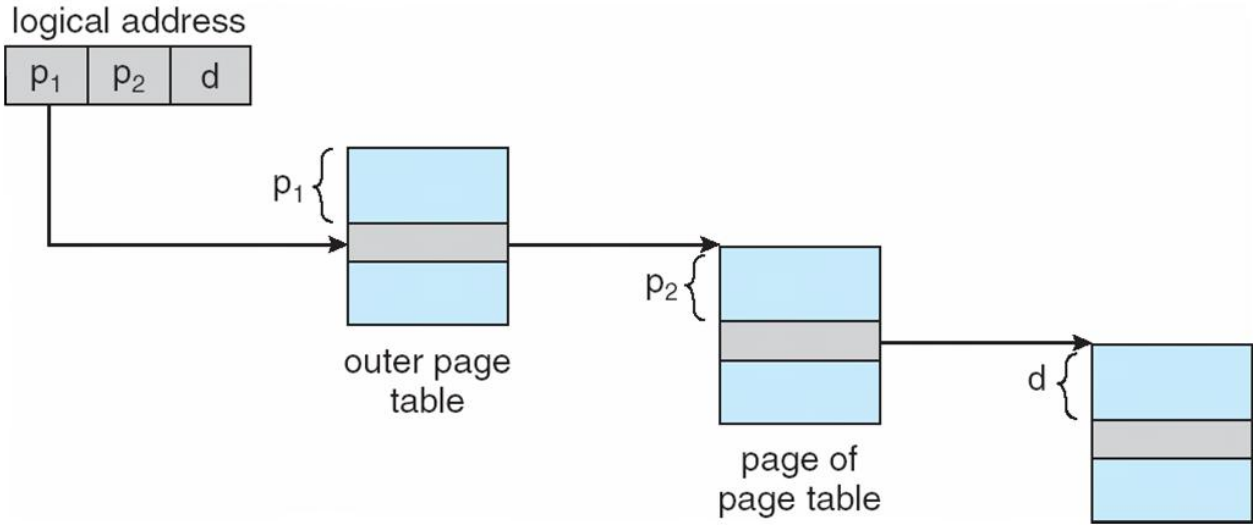


- A logical address (on 32-bit machine with 1K page size) is divided into:
  - a page number consisting of 22 bits
  - a page offset consisting of 10 bits
  - Since the page table is paged, the page number is further divided into:
  - a 12-bit page number
  - a 10-bit page offset
  - Thus, a logical address is as follows:



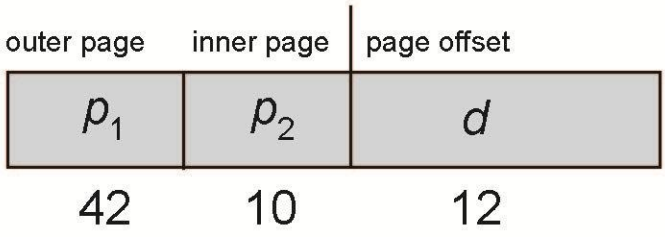
- where p_1 is an index into the outer page table, and p_2 is the displacement within the page of the inner page table
- Known as **forward-mapped page table**

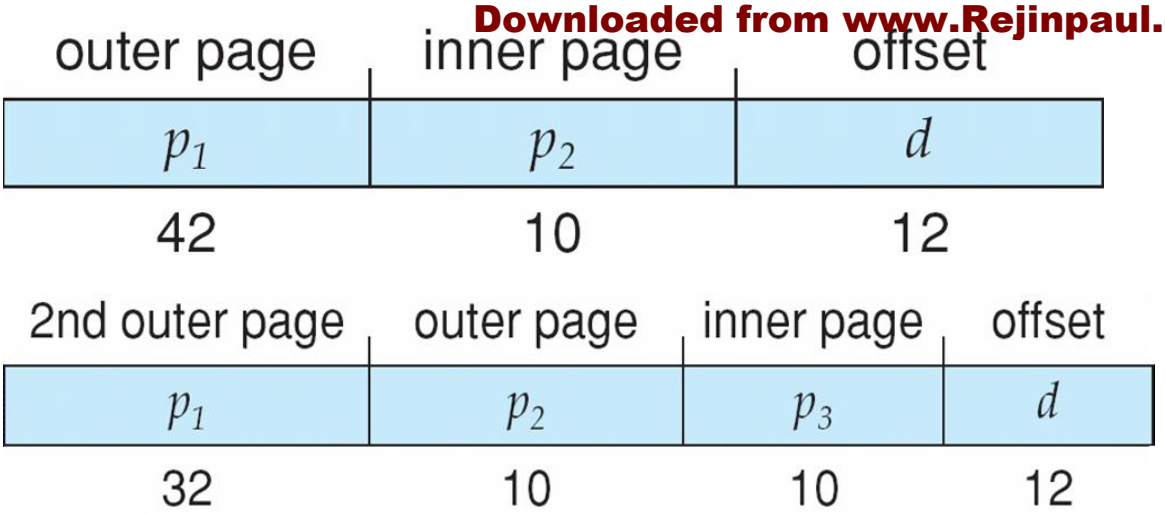
ADDRESS TRANSLATION SCHEME



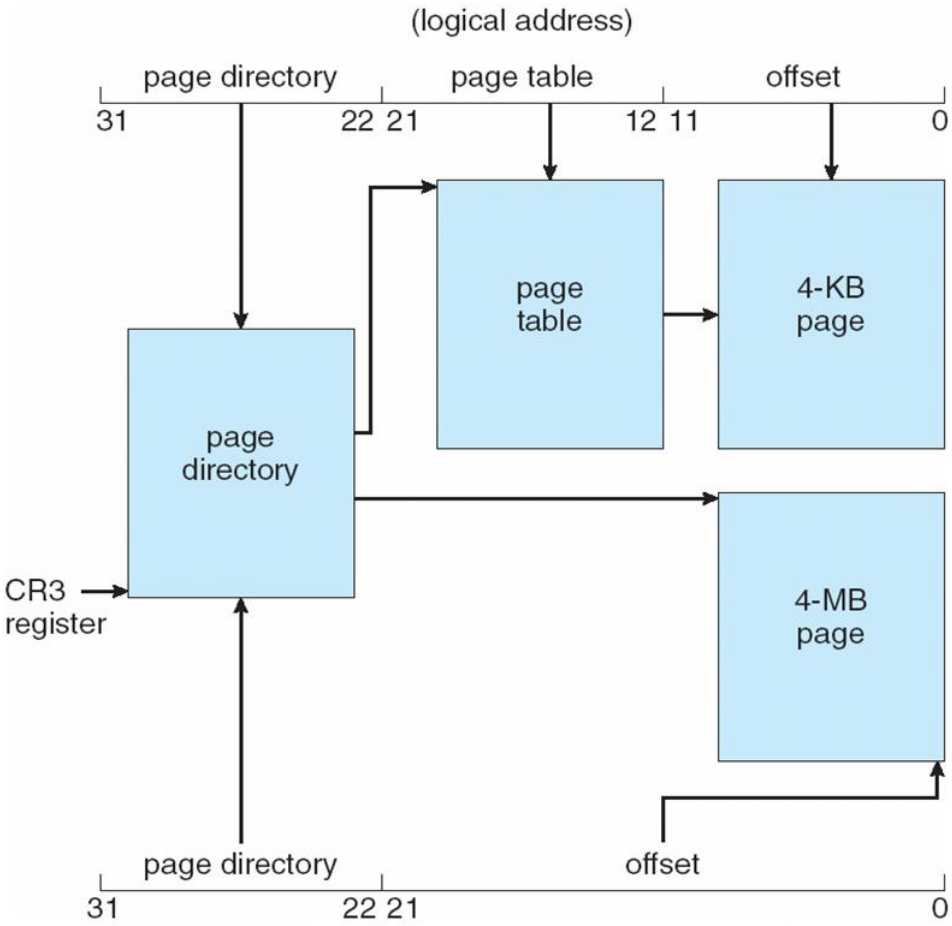
64 BIT ARCHITECTURE

- Even two-level paging scheme not sufficient
- If page size is 4 KB (2^{12})
  - Then page table has 2^{52} entries
  - If two level scheme, inner page tables could be 2^{10} 4-byte entries
  - Address would look like
  - Outer page table has 2^{42} entries or 2^{44} bytes
  - One solution is to add a 2^{nd} outer page table
  - But in the following example the 2^{nd} outer page table is still 2^{34} bytes in size
    - And possibly 4 memory access to get to one physical memory location

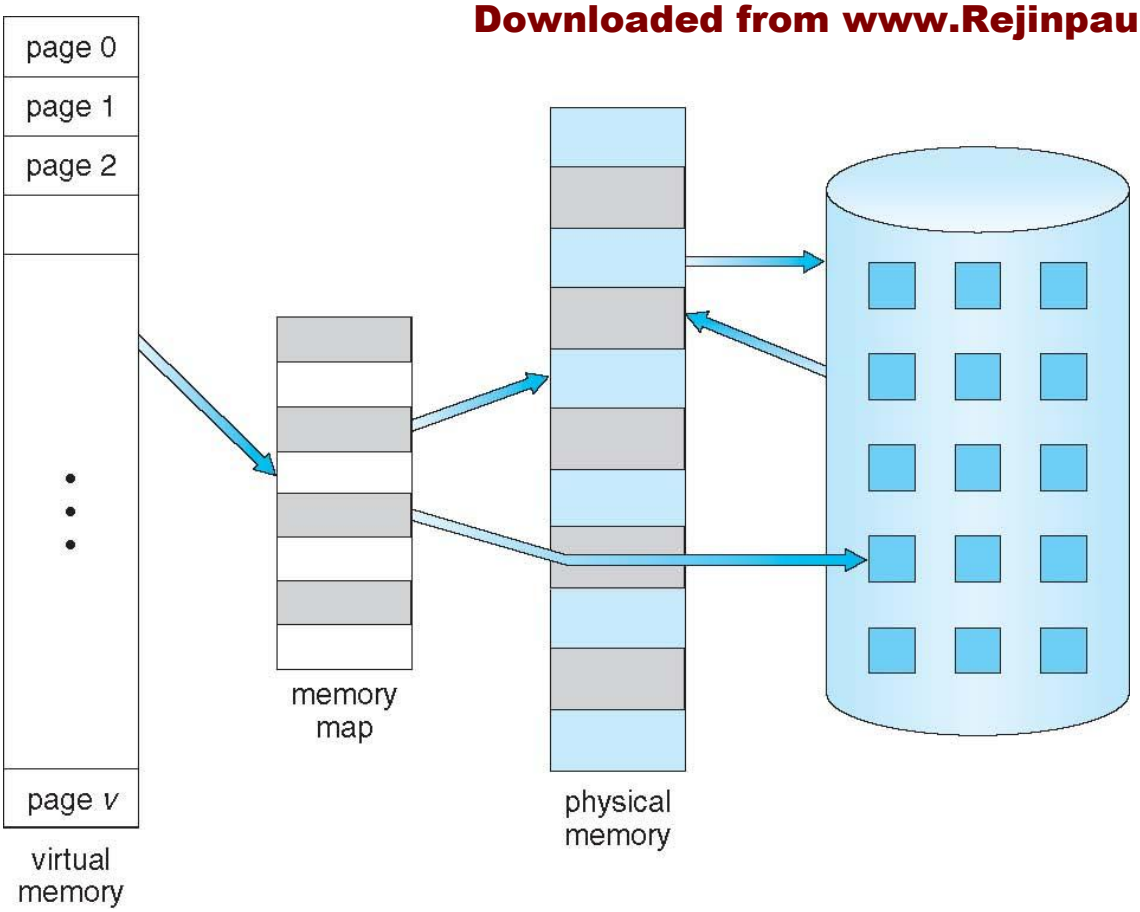




INTEL 32 BIT ARCHITECTURE

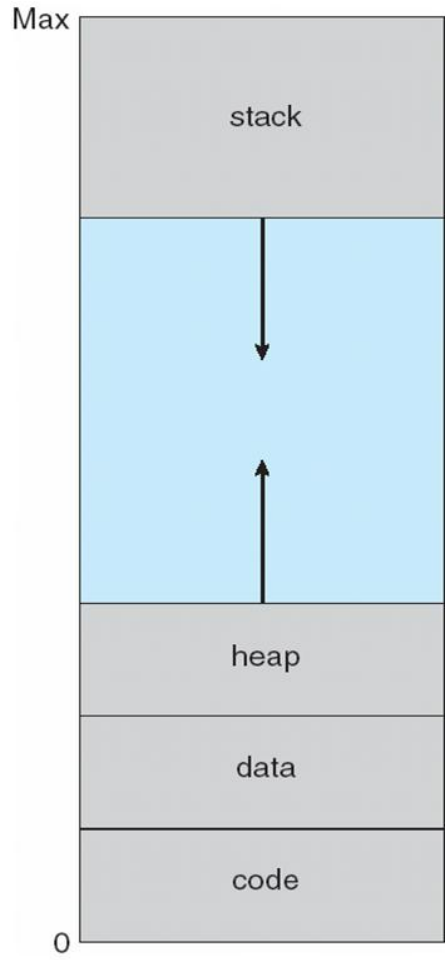


VIRTUAL MEMORY

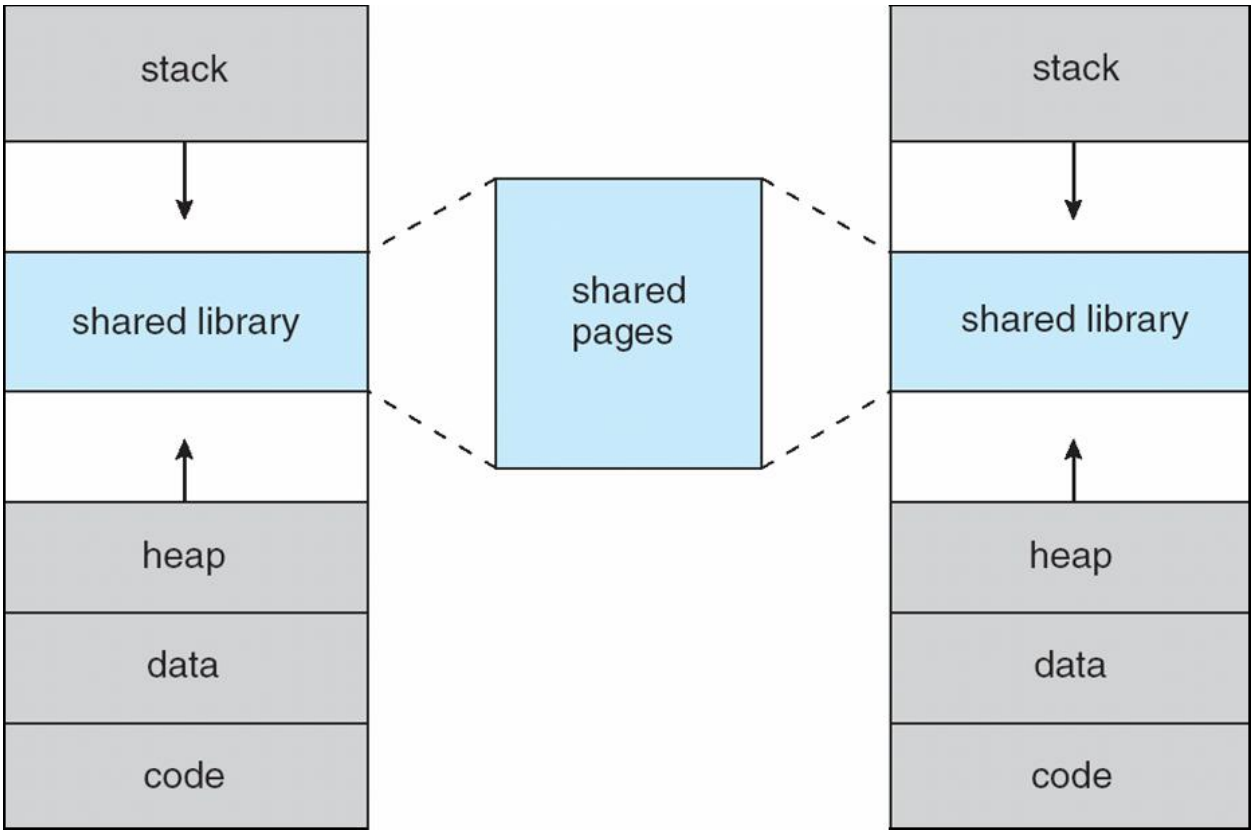


VIRTUAL ADDRESS SPACE

- Usually design logical address space for stack to start at Max logical address and grow "down" while heap grows "up"
    - Maximizes address space use
    - Unused address space between the two is hole
        - No physical memory needed until heap or stack grows to a given new page
- Enables **sparse** address spaces with holes left for growth, dynamically linked libraries, etc
- System libraries shared via mapping into virtual address space
- Shared memory by mapping pages read-write into virtual address space
- Pages can be shared during fork(), speeding process creation

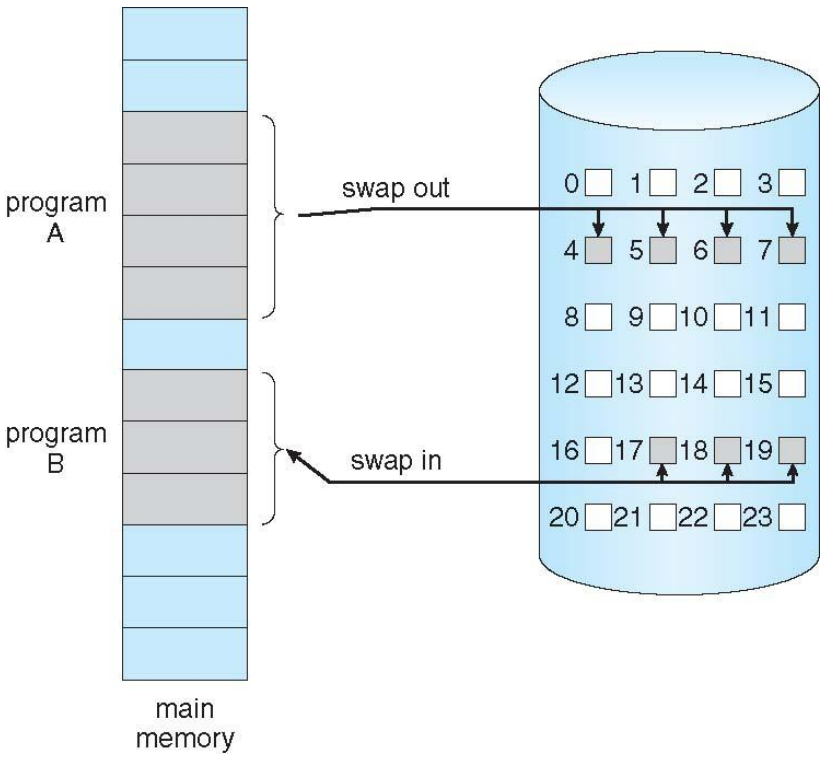


SHARED LIBRARY USING VIRTUAL MEMORY



DEMAND PAGING

- Could bring entire process into memory at load time
- Or bring a page into memory only when it is needed
    - Less I/O needed, no unnecessary I/O
    - Less memory needed
    - Faster response
    - More users
- Similar to paging system with swapping (diagram on right)
- Page is needed ⇒ reference to it
    - invalid reference ⇒ abort
    - not-in-memory ⇒ bring to memory
- **Lazy swapper** – never swaps a page into memory unless page will be needed
    - Swapper that deals with pages is a **pager**

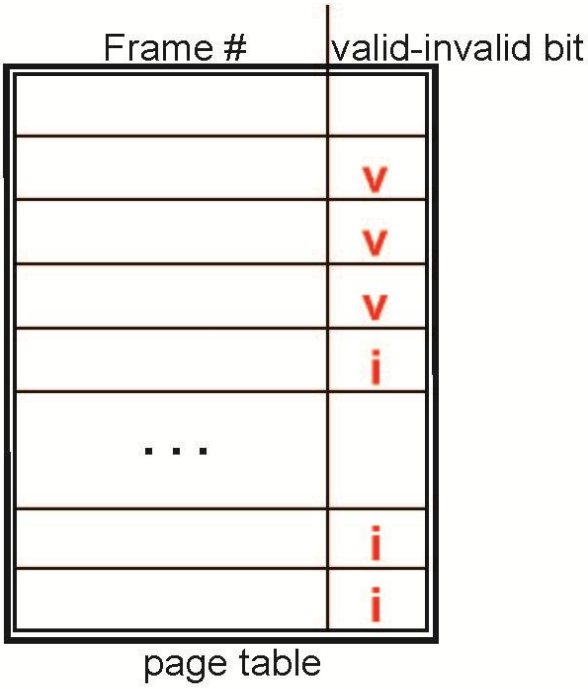


- With swapping, pager guesses which pages will be used before swapping out again
- Instead, pager brings in only those pages into memory
- How to determine that set of pages?
    - Need new MMU functionality to implement demand paging

- If pages needed are already **memory resident**
    - No difference from non demand-paging
- If page needed and not memory resident
    - Need to detect and load the page into memory from storage
        - Without changing program behavior
        - Without programmer needing to change code

VALID OR INVALID BIT

- With each page table entry a valid–invalid bit is associated
  (**v** ⇒ in-memory – **memory resident**, **i** ⇒ not-in-memory)
- Initially valid–invalid bit is set to **i** on all entries
- Example of a page table snapshot:



During MMU address translation, if valid–invalid bit in page table entry is **i** ⇒ page fault

If there is a reference to a page, first reference to that page will trap to operating system:

PAGE FAULT

1. Operating system looks at another table to decide:
    - l Invalid reference ⇒ abort
    - l Just not in memory
2. Find free frame

3. Swap page into frame via scheduled disk operation
4. Reset tables to indicate page now in memory
   Set validation bit = **v**
5. Restart the instruction that caused the page fault
6. Extreme case – start process with *no* pages in memory
   - OS sets instruction pointer to first instruction of process, non-memory-resident -> page fault
   - And for every other process pages on first access
   - **Pure demand paging**
7. Actually, a given instruction could access multiple pages -> multiple page faults
   - Consider fetch and decode of instruction which adds 2 numbers from memory and stores result back to memory
   - Pain decreased because of **locality of reference**
8. Hardware support needed for demand paging
   - Page table with valid / invalid bit
   - Secondary memory (swap device with **swap space**)
   - Instruction restart

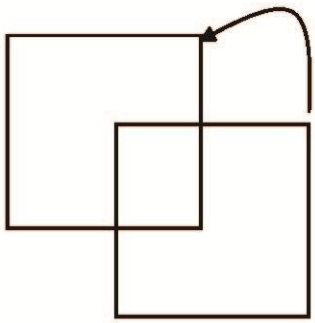
INSTRUCTION RESTART

Consider an instruction that could access several different locations

block move
auto increment/decrement location

Restart the whole operation?

- What if source and destination overlap?



PERFORMANCE

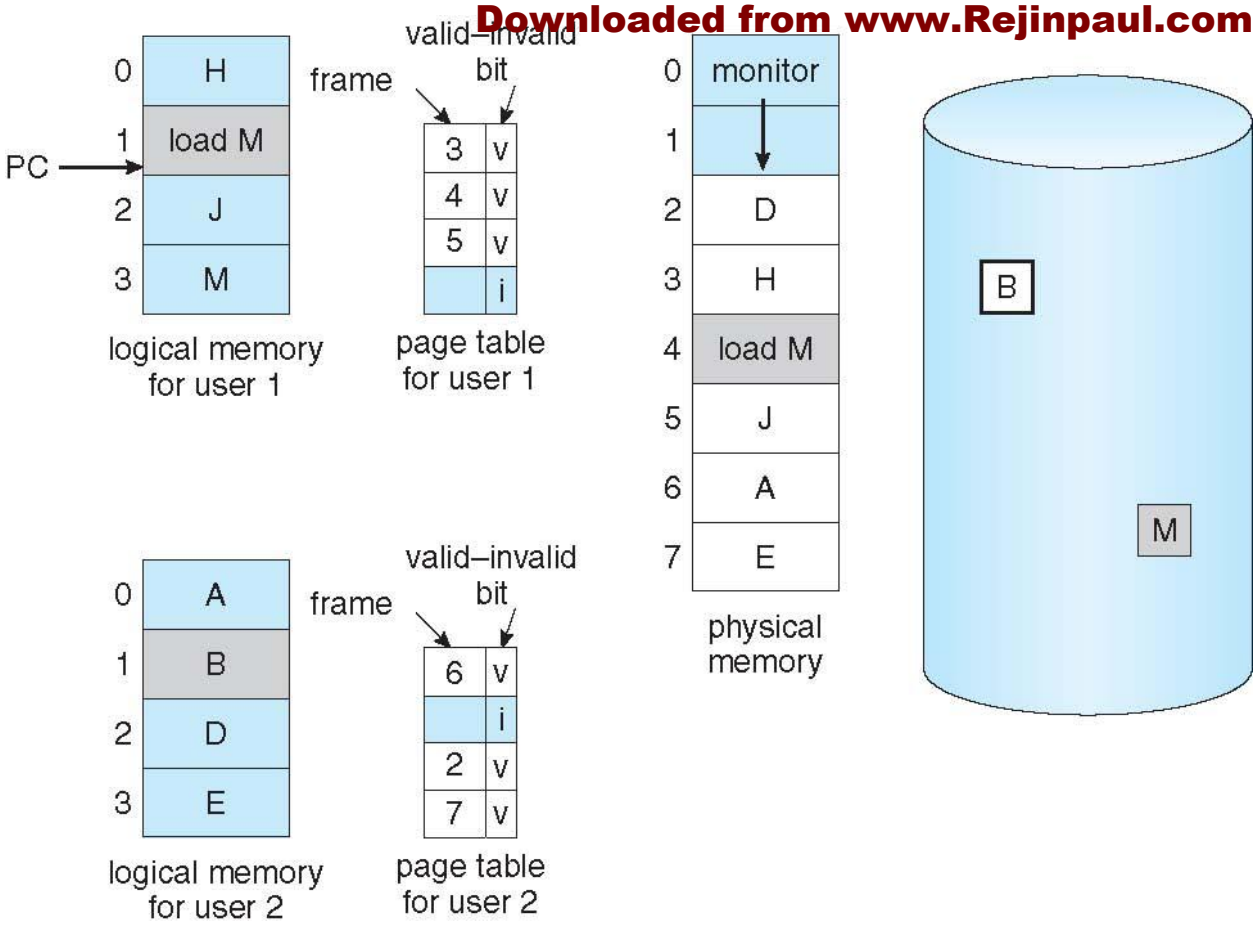
- Stages in Demand Paging (worse case)

1. Trap to the operating system
2. Save the user registers and process state
3. Determine that the interrupt was a page fault
4. Check that the page reference was legal and determine the location of the page on the disk
5. Issue a read from the disk to a free frame:
    1. Wait in a queue for this device until the read request is serviced
    2. Wait for the device seek and/or latency time
    3. Begin the transfer of the page to a free frame
6. While waiting, allocate the CPU to some other user
7. Receive an interrupt from the disk I/O subsystem (I/O completed)
8. Save the registers and process state for the other user
9. Determine that the interrupt was from the disk
10. Correct the page table and other tables to show page is now in memory
11. Wait for the CPU to be allocated to this process again
12. Restore the user registers, process state, and new page table, and then resume the interrupted instruction

PAGE REPLACEMENT

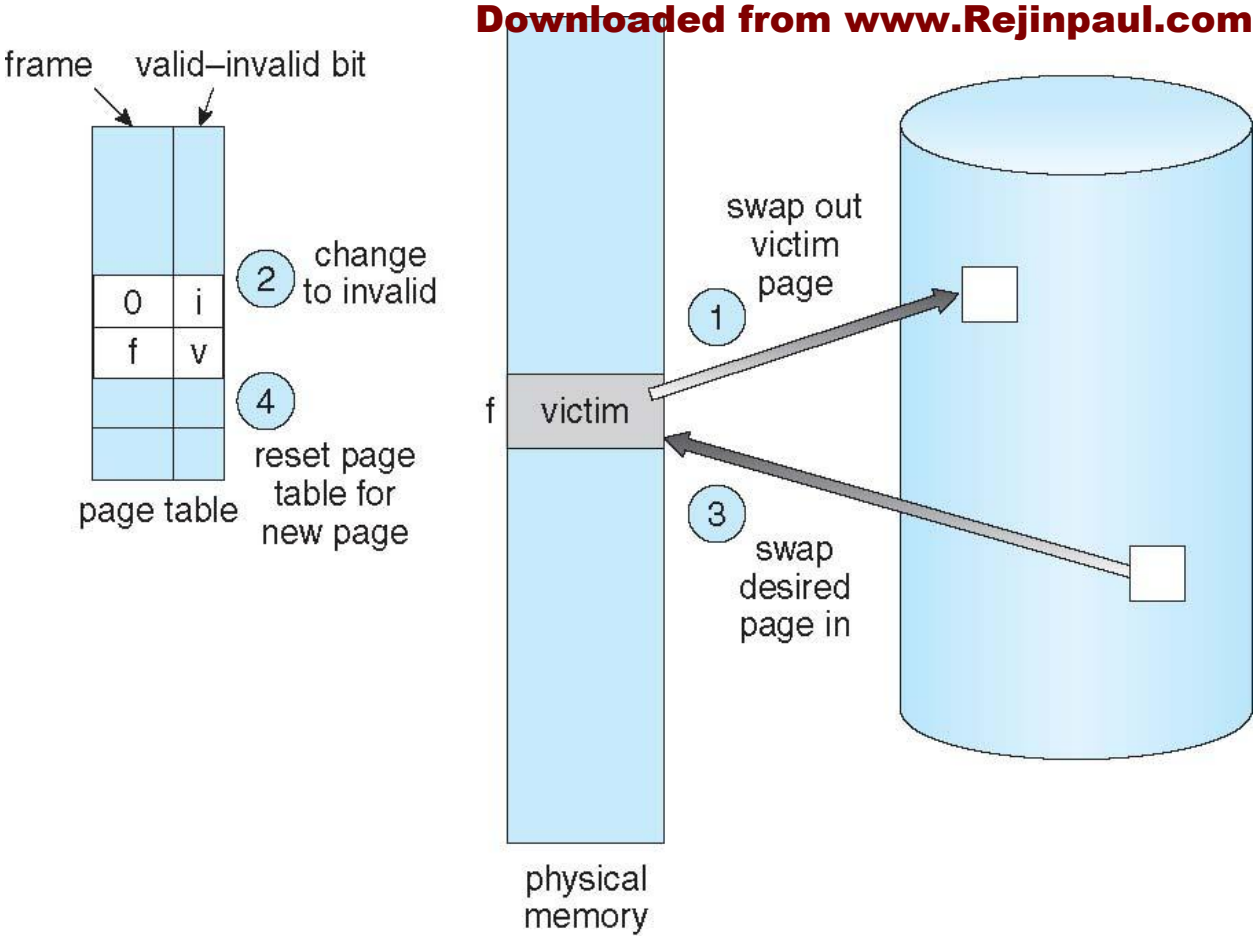
- Prevent **over-allocation** of memory by modifying page-fault service routine to include page replacement
- Use **modify** (**dirty**) **bit** to reduce overhead of page transfers – only modified pages are written to disk
- Page replacement completes separation between logical memory and physical memory – large virtual memory can be provided on a smaller physical memory

NEED



BASIC PAGE REPLACEMENT

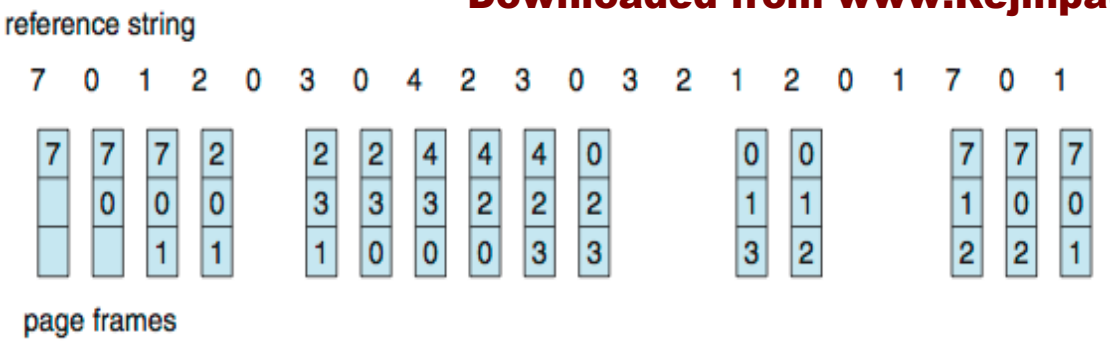
1. Find the location of the desired page on disk
2. Find a free frame:
   - If there is a free frame, use it
   - If there is no free frame, use a page replacement algorithm to select a **victim frame**
     - Write victim frame to disk if dirty
3. Bring the desired page into the (newly) free frame; update the page and frame tables
4. Continue the process by restarting the instruction that caused the trap



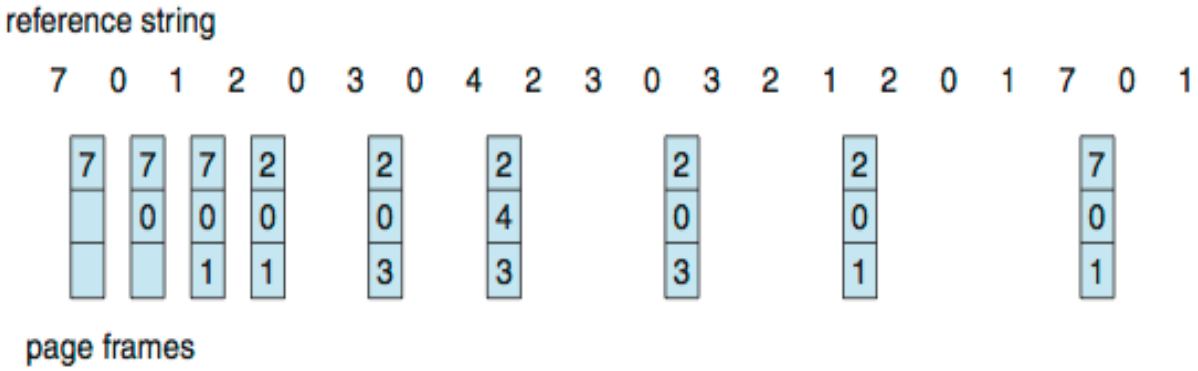
ALGORITHMS

- **Frame-allocation algorithm** determines
  - How many frames to give each process
  - Which frames to replace
- **Page-replacement algorithm**
  - Want lowest page-fault rate on both first access and re-access
- Evaluate algorithm by running it on a particular string of memory references (reference string) and computing the number of page faults on that string
  - String is just page numbers, not full addresses
  - Repeated access to the same page does not cause a page fault
  - Results depend on number of frames available
- In all our examples, the **reference string** of referenced page numbers is
    - **7,0,1,2,0,3,0,4,2,3,0,3,0,3,2,1,2,0,1,7,0,1**

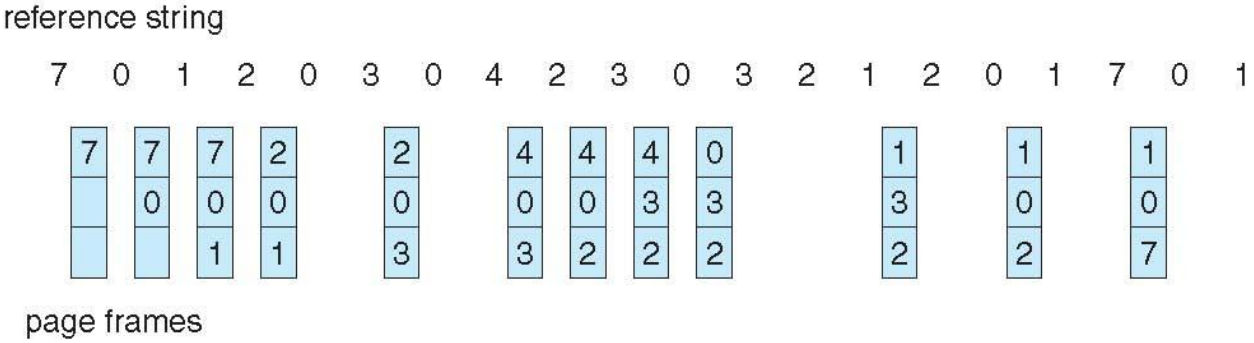
FIFO



OPTIMAL ALGORITHM



LRU

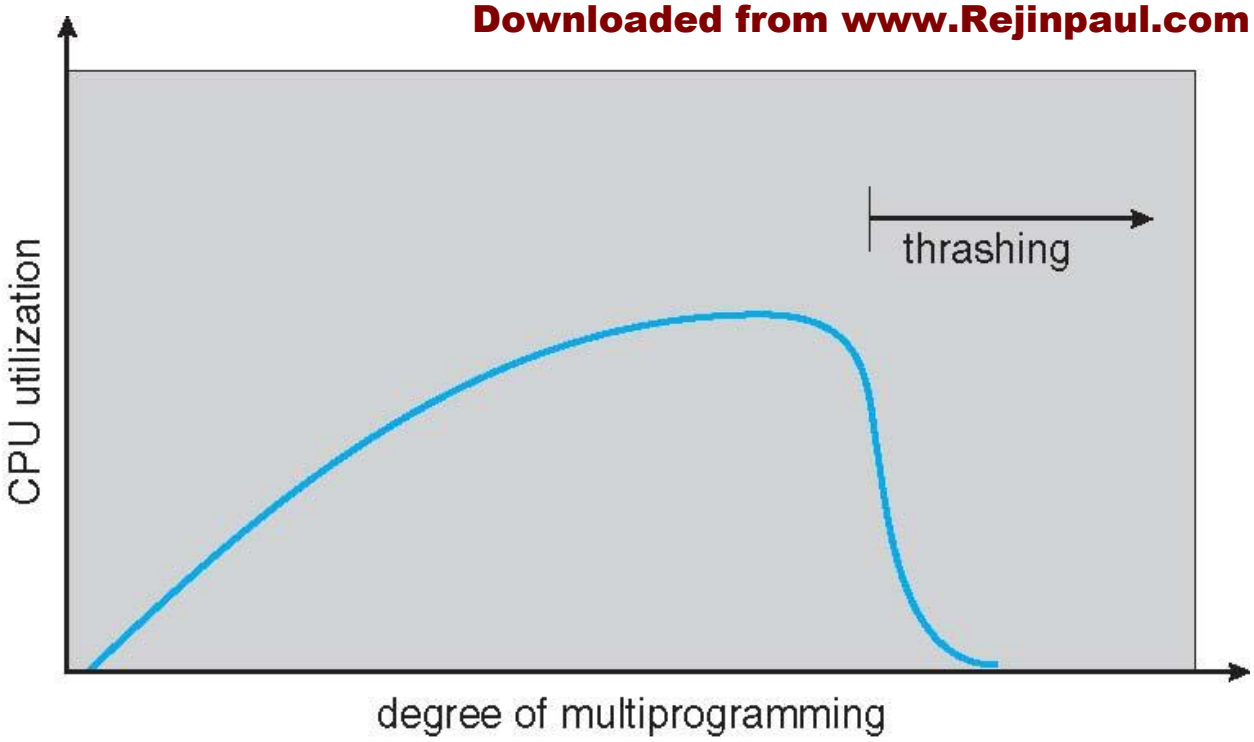


- Keep a pool of free frames, always
  - Then frame available when needed, not found at fault time
  - Read page into free frame and select victim to evict and add to free pool
  - When convenient, evict victim
- Possibly, keep list of modified pages
  - When backing store otherwise idle, write pages there and set to non-dirty
- Possibly, keep free frame contents intact and note what is in them
  - If referenced again before reused, no need to load contents again from disk
  - Generally useful to reduce penalty if wrong victim frame selected

- Each process needs ***minimum*** number of frames
- Example: IBM 370 – 6 pages to handle SS MOVE instruction:
    - instruction is 6 bytes, might span 2 pages
    - 2 pages to handle *from*
    - 2 pages to handle *to*
- ***Maximum*** of course is total frames in the system
- Two major allocation schemes
    - fixed allocation
    - priority allocation
- Many variations

THRASHING

- If a process does not have "enough" pages, the page-fault rate is very high
    - Page fault to get page
    - Replace existing frame
    - But quickly need replaced frame back
    - This leads to:
        - Low CPU utilization
        - Operating system thinking that it needs to increase the degree of multiprogramming
        - Another process added to the system
- **Thrashing** ≡ a process is busy swapping pages in and out
- Why does demand paging work?
  Locality model
- Process migrates from one locality to another
- Localities may overlap
- Why does thrashing occur?
  Σ size of locality > total memory size
- Limit effects by using local or priority page replacement



FILE

- Contiguous logical address space
- Types:
  - Data
    - numeric
    - character
    - binary
  - Program
- Contents defined by file's creator
  - Many types
    - Consider **text file, source file, executable file**

FILE ATTRIBUTES

- **Name** – only information kept in human-readable form
- **Identifier** – unique tag (number) identifies file within file system
- **Type** – needed for systems that support different types
- **Location** – pointer to file location on device
- **Size** – current file size
- **Protection** – controls who can do reading, writing, executing
- **Time, date, and user identification** – data for protection, security, and usage monitoring
- Information about files are kept in the directory structure, which is maintained on the disk
- Many variations, including extended file attributes such as file checksum
- Information kept in the directory structure

OPERATIONS

- File is an **abstract data type**
- **Create**
- **Write –** at **write pointer** location

- **Read –** at **read pointer** location
- **Reposition within file - seek**
- **Delete**
- **Truncate**
- ***Open(F_i)*** – search the directory structure on disk for entry ***F_i***, and move the content of entry to memory
- ***Close (F_i)*** – move the content of entry ***F_i*** in memory to directory structure on disk
- Several pieces of data are needed to manage open files:
    - **Open-file table**: tracks open files
    - File pointer: pointer to last read/write location, per process that has the file open
    - **File-open count**: counter of number of times a file is open – to allow removal of data from open-file table when last processes closes it
    - Disk location of the file: cache of data access information
    - Access rights: per-process access mode information

OPEN FILE LOCKING

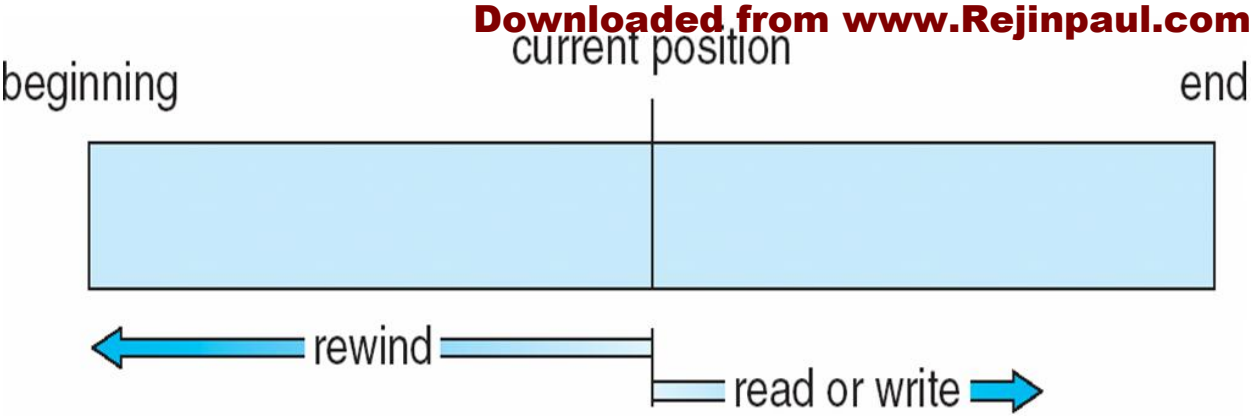
- Provided by some operating systems and file systems
    - Similar to reader-writer locks
    - **Shared lock** similar to reader lock – several processes can acquire concurrently
    - **Exclusive lock** similar to writer lock
- Mediates access to a file
- Mandatory or advisory:
    - **Mandatory** – access is denied depending on locks held and requested
    - **Advisory** – processes can find status of locks and decide what to do

file type	usual extension	function
executable	exe, com, bin or none	ready-to-run machine-language program
object	obj, o	compiled, machine language, not linked
source code	c, cc, java, pas, asm, a	source code in various languages
batch	bat, sh	commands to the command interpreter
text	txt, doc	textual data, documents
word processor	wp, tex, rtf, doc	various word-processor formats
library	lib, a, so, dll	libraries of routines for programmers
print or view	ps, pdf, jpg	ASCII or binary file in a format for printing or viewing
archive	arc, zip, tar	related files grouped into one file, sometimes com-pressed, for archiving or storage
multimedia	mpeg, mov, rm, mp3, avi	binary file containing audio or A/V information

STRUCTURE

- None - sequence of words, bytes
- Simple record structure
    - Lines
    - Fixed length
    - Variable length
- Complex Structures
    - Formatted document
    - Relocatable load file
- Can simulate last two with first method by inserting appropriate control characters
- Who decides:
    - Operating system
    - Program

SEQUENTIAL FILE ACCESS



ACCESS METHODS

Sequential Access

read next

write next

reset

no read after last write

(rewrite)

Direct Access – file is fixed length logical records

read *n*

write *n*

position to *n*

read next

write next

rewrite *n*

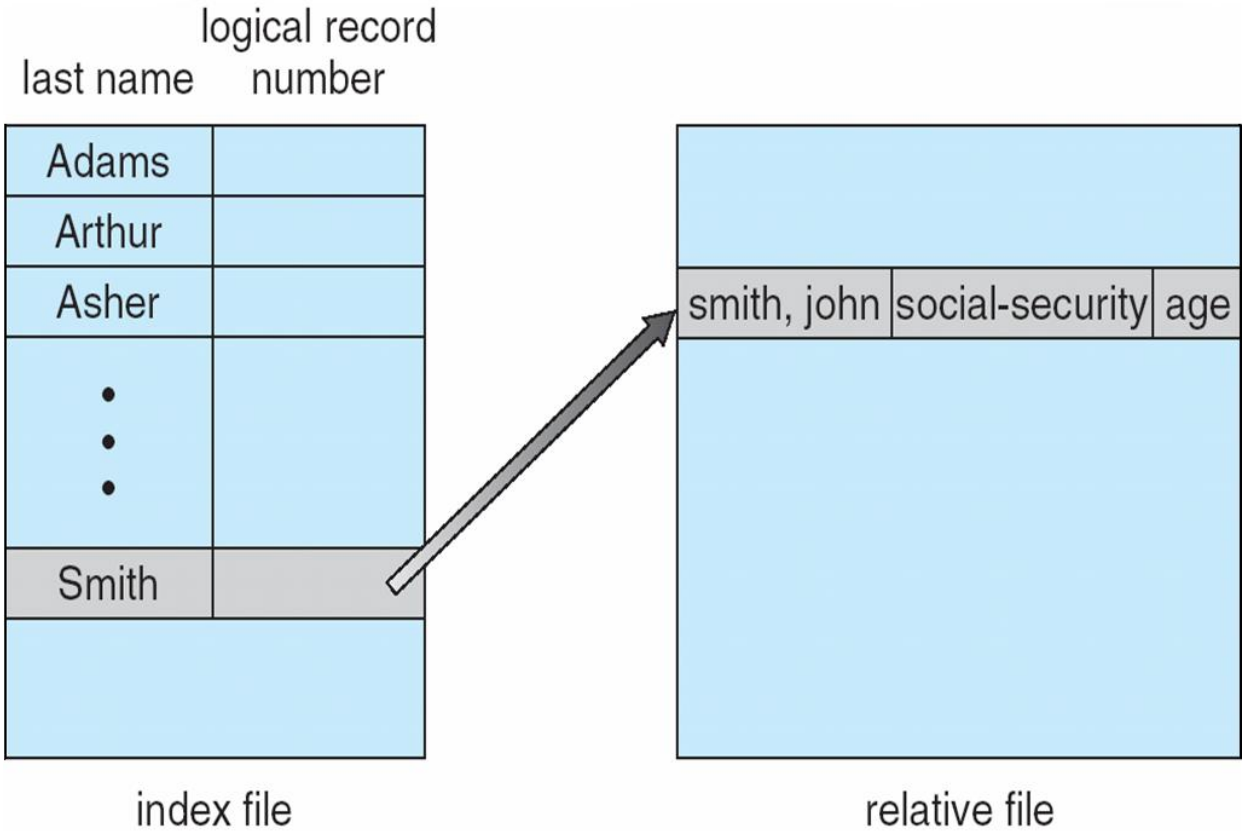
n = relative block number

Relative block numbers allow OS to decide where file should be placed

Simulation of Sequential Access on Direct-access File

sequential access	implementation for direct access
reset	*cp* = 0;
read next	*read cp*; *cp* = *cp* + 1;
write next	*write cp*; *cp* = *cp* + 1;

- Can be built on top of base methods
- General involve creation of an index for the file
- Keep index in memory for fast determination of location of data to be operated on (consider UPC code plus record of data about that item)
- If too large, index (in memory) of the index (on disk)
- IBM indexed sequential-access method (ISAM)
    - Small master index, points to disk blocks of secondary index
    - File kept sorted on a defined key
    - All done by the OS
- VMS operating system provides index and relative files as another example



DIRECTORY STRUCTURE

A collection of nodes containing information about all files

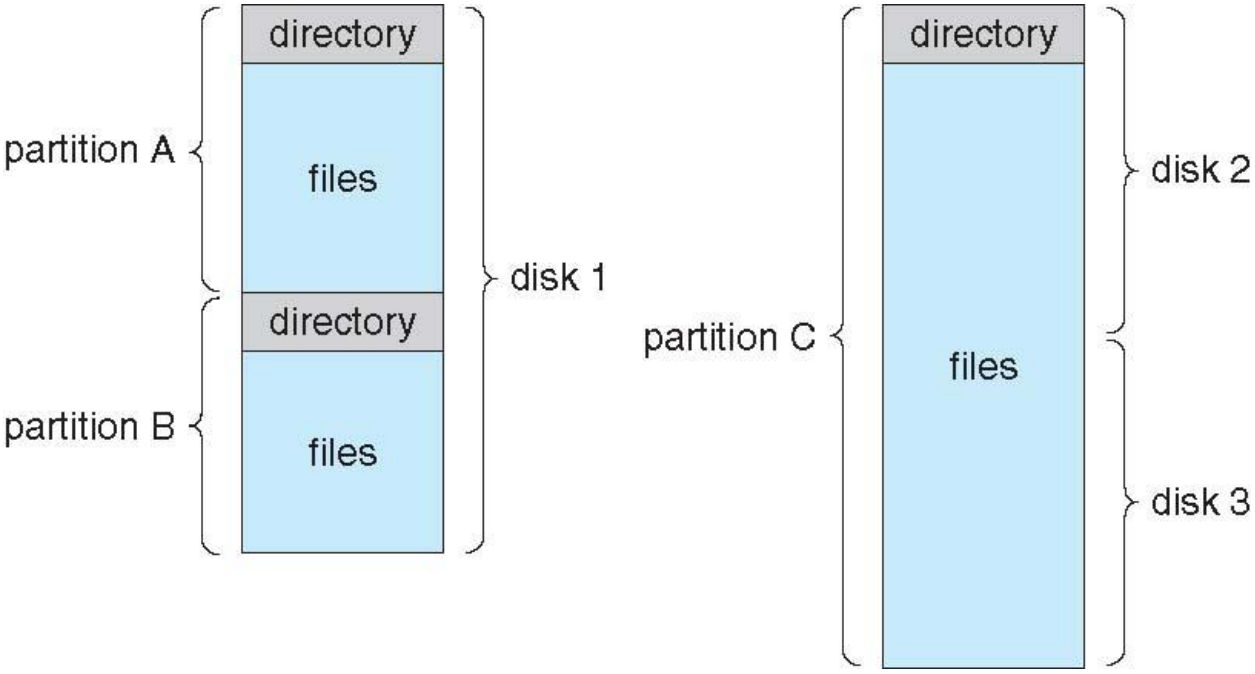
Both the directory structure and the files reside on disk

DISK STRUCTURE

- Disk can be subdivided into **partitions**

- Disks or partitions can be **RAID** protected against failure
- Disk or partition can be used **raw** – without a file system, or **formatted** with a file system
- Partitions also known as minidisks, slices
- Entity containing file system known as a **volume**
- Each volume containing file system also tracks that file system's info in **device directory** or **volume table of contents**
- As well as **general-purpose file systems** there are many **special-purpose file systems**, frequently all within the same operating system or computer

FILE SYSTEM ORGANIZATION



TYPES

- We mostly talk of general-purpose file systems
- But systems frequently have may file systems, some general- and some special- purpose
- Consider Solaris has
    - tmpfs – memory-based volatile FS for fast, temporary I/O
    - objfs – interface into kernel memory to get kernel symbols for debugging
    - ctfs – contract file system for managing daemons
    - lofs – loopback file system allows one FS to be accessed in place of another
    - procfs – kernel interface to process structures
    - ufs, zfs – general purpose file systems

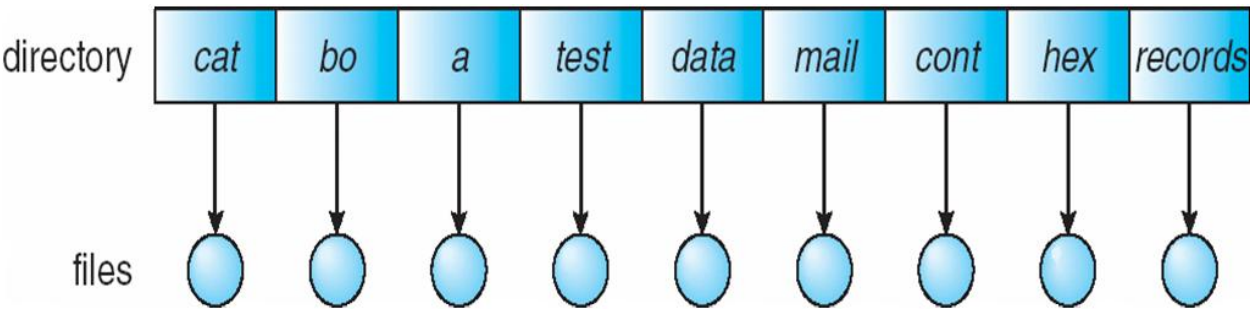
DIRECTORY OPERATIONS

- Search for a file
- Create a file
- Delete a file
- List a directory
- Rename a file
- Traverse the file system

DIRECTORY ORGANIZATION

- Efficiency – locating a file quickly
- Naming – convenient to users
    - Two users can have same name for different files
    - The same file can have several different names
- Grouping – logical grouping of files by properties, (e.g., all Java programs, all games, …)

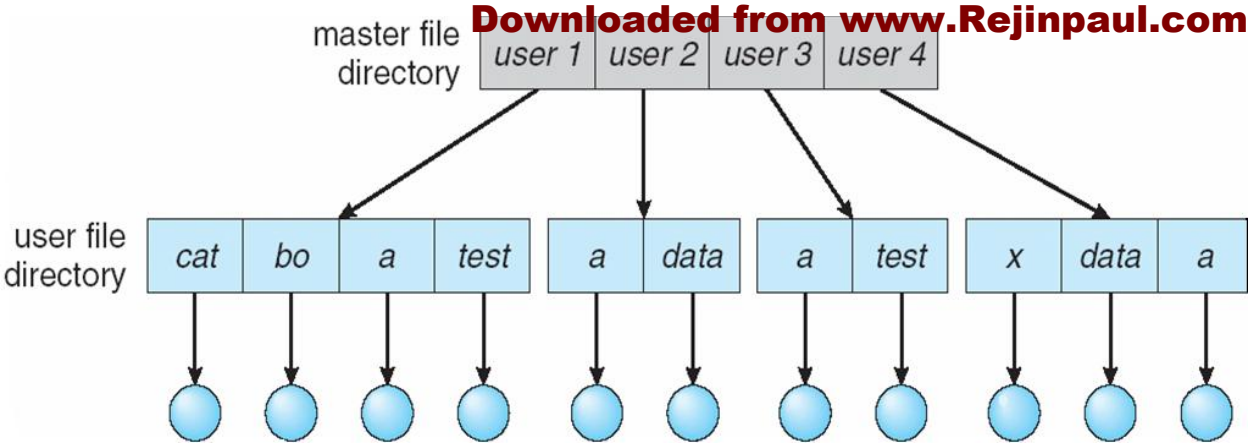
SINGLE LEVEL DIRECTORY



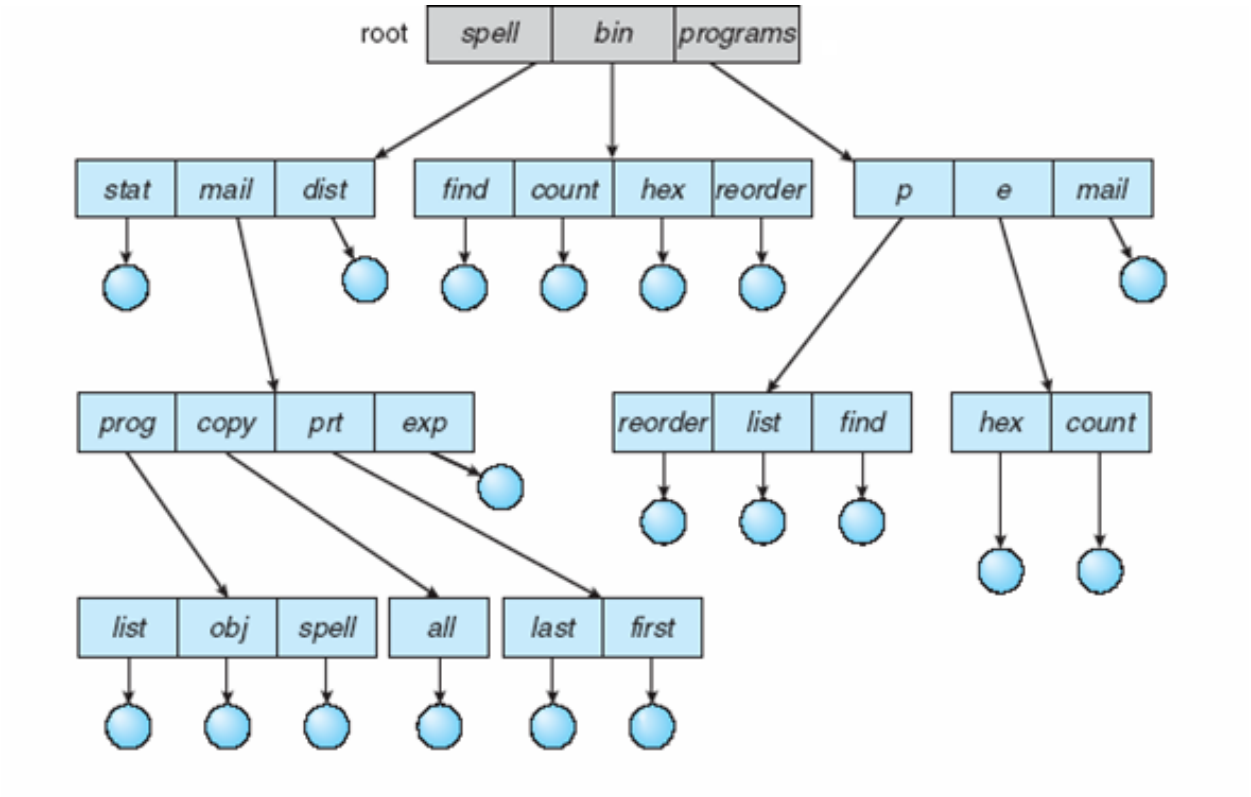
- A single directory for all users
- Naming problem
- Grouping problem

TWO LEVEL DIRECTORY

- Path name
- Can have the same file name for different user
- Efficient searching
- No grouping capability

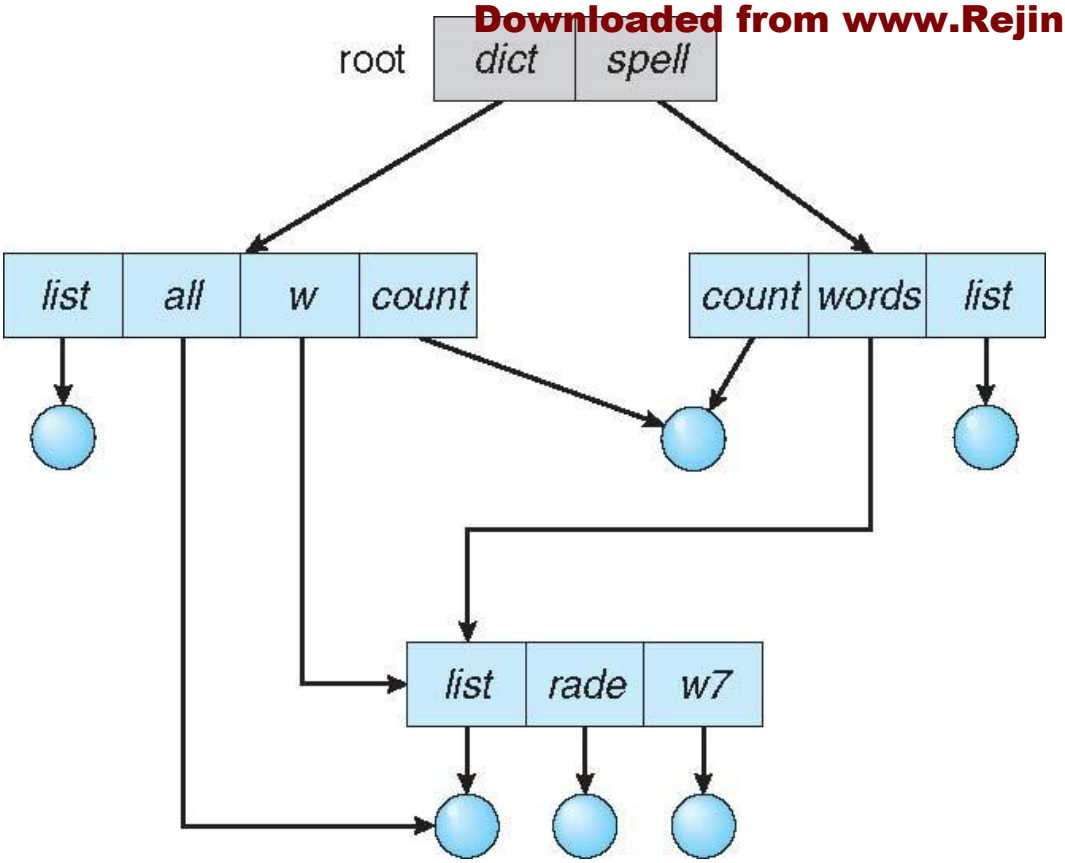


TREE STRUCTURED DIRECTORIES



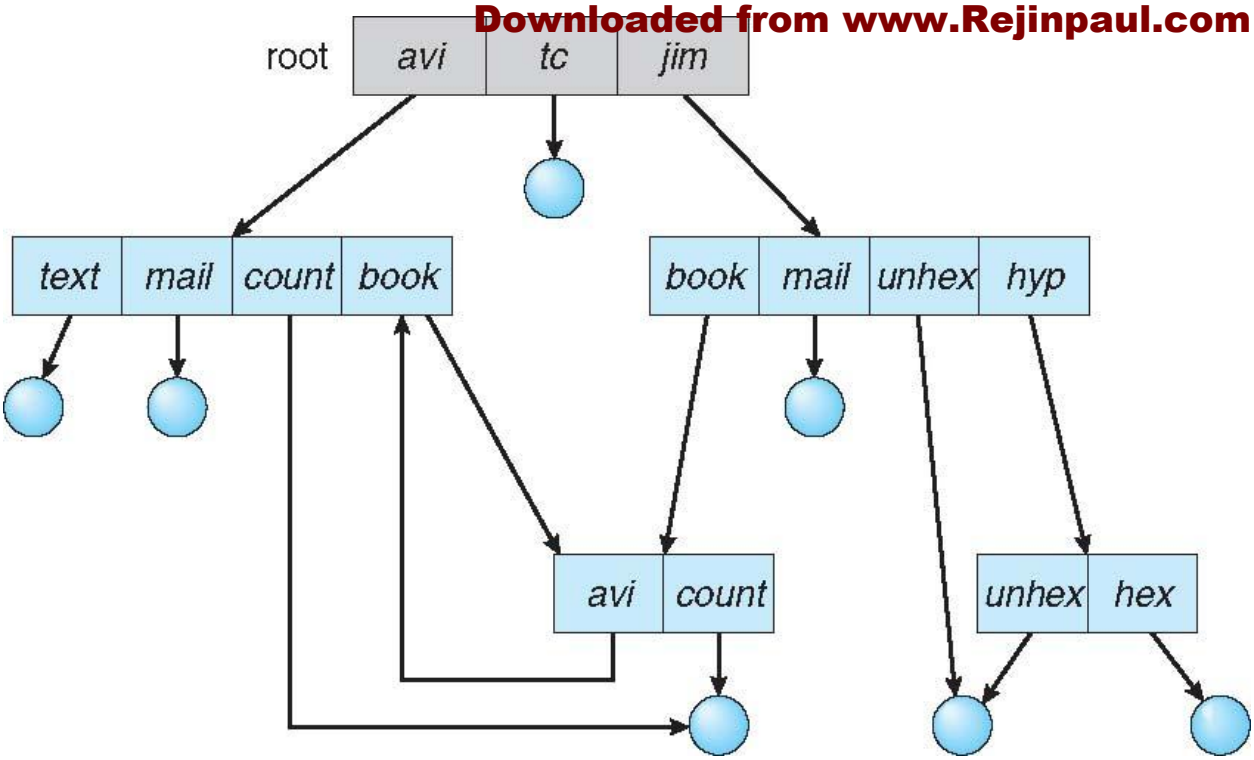
- Efficient searching
- Grouping Capability
- Current directory (working directory)
    - **cd /spell/mail/prog**
    - **type list**

ACYCLIC GRAPH DIRECTORIES

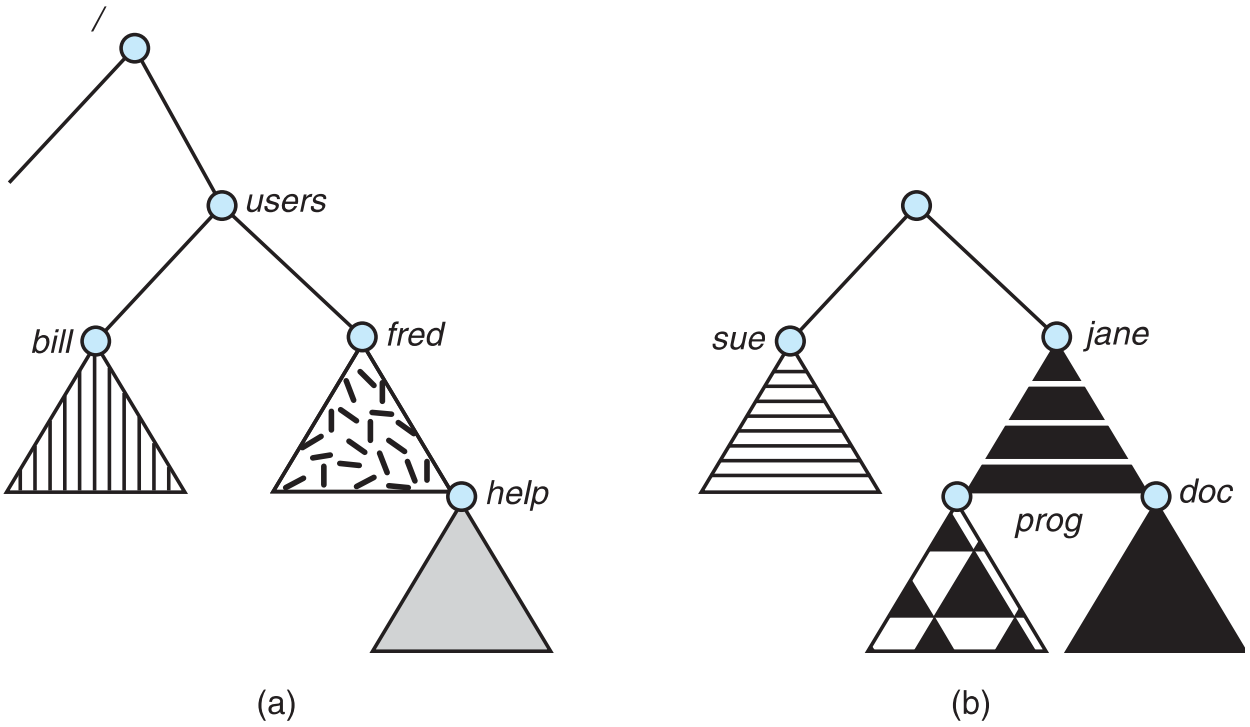


- Two different names (aliasing)
- If ***dict*** deletes ***list*** ⇒ dangling pointer
    - Solutions:
    - Backpointers, so we can delete all pointers
      Variable size records a problem
    - Backpointers using a daisy chain organization
    - Entry-hold-count solution
- New directory entry type
    - **Link** – another name (pointer) to an existing file
    - **Resolve the link** – follow pointer to locate the file

GENERAL GRAPH DIRECTORY



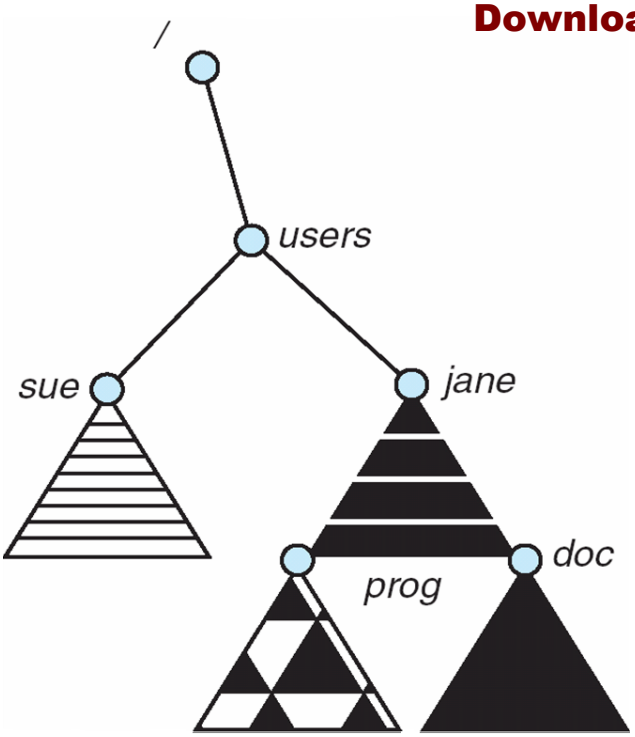
FILE SYSTEM MOUNTING



(a) (b)

- A file system must be **mounted** before it can be accessed
- A unmounted file system is mounted at a **mount point**

MOUNT POINT



FILE SHARING

- Sharing of files on multi-user systems is desirable
- Sharing may be done through a **protection** scheme
- On distributed systems, files may be shared across a network
- Network File System (NFS) is a common distributed file-sharing method
- If multi-user system
    - **User IDs** identify users, allowing permissions and protections to be per-user **Group IDs** allow users to be in groups, permitting group access rights
    - Owner of a file / directory
    - Group of a file / directory
- Uses networking to allow file system access between systems
    - Manually via programs like FTP
    - Automatically, seamlessly using **distributed file systems**
    - Semi automatically via the **world wide web**
- **Client-server** model allows clients to mount remote file systems from servers
    - Server can serve multiple clients
    - Client and user-on-client identification is insecure or complicated
    - **NFS** is standard UNIX client-server file sharing protocol

- **CIFS** is standard Windows protocol
- Standard operating system file calls are translated into remote calls

- Distributed Information Systems **(distributed naming services)** such as LDAP, DNS, NIS, Active Directory implement unified access to information needed for remote computing
- All file systems have failure modes
  - For example corruption of directory structures or other non-user data, called **metadata**
- Remote file systems add new failure modes, due to network failure, server failure
- Recovery from failure can involve **state information** about status of each remote request
- **Stateless** protocols such as NFS v3 include all information in each request, allowing easy recovery but less security

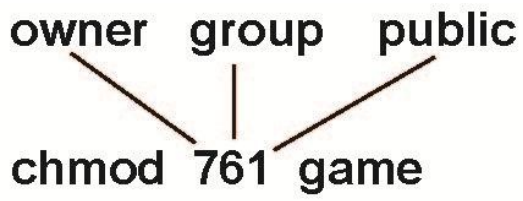
PROTECTION

- File owner/creator should be able to control:
  - what can be done
  - by whom
- Types of access
  - **Read**
  - **Write**
  - **Execute**
  - **Append**
  - **Delete**
  - **List**
- Mode of access: read, write, execute
- Three classes of users on Unix / Linux

				RWX
▪	a) **owner access**	7	⇒	1 1 1
				RWX
▪	b) **group access**	6	⇒	1 1 0
				RWX

- c) **public access** 1 ⇒ 0 0 1

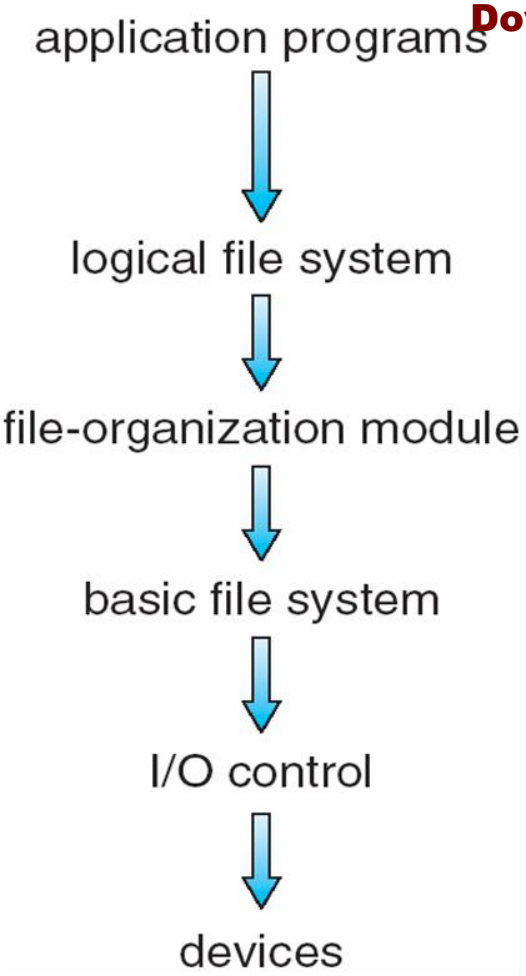
- Ask manager to create a group (unique name), say G, and add some users to the group.
- For a particular file (say *game*) or subdirectory, define an appropriate access.



FILE SYSTEM STRUCTURE

- File structure
  - Logical storage unit
  - Collection of related information
- **File system** resides on secondary storage (disks)
  - Provided user interface to storage, mapping logical to physical
  - Provides efficient and convenient access to disk by allowing data to be stored, located retrieved easily
- Disk provides in-place rewrite and random access
  - I/O transfers performed in **blocks** of **sectors** (usually 512 bytes)
- **File control block** – storage structure consisting of information about a file
- **Device driver** controls the physical device
- File system organized into layers

LAYERED FILE SYSTEM



FILE SYSTEM LAYERS

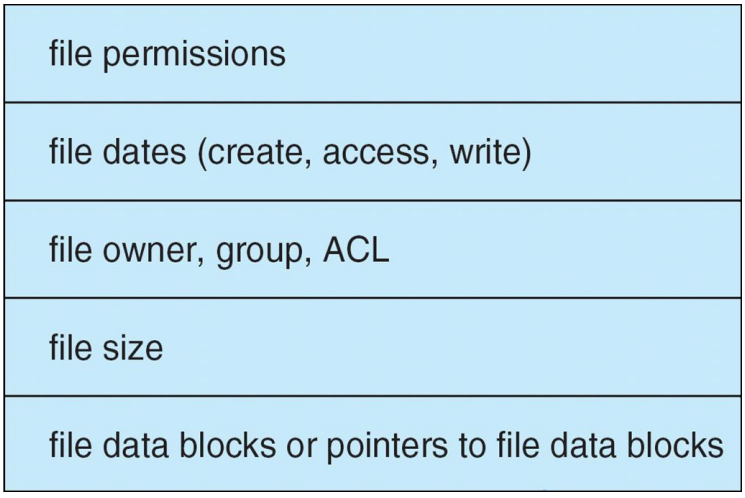
- **Device drivers** manage I/O devices at the I/O control layer
    - Given commands like "read drive1, cylinder 72, track 2, sector 10, into memory location 1060" outputs low-level hardware specific commands to hardware controller
- **Basic file system** given command like "retrieve block 123" translates to device driver
- Also manages memory buffers and caches (allocation, freeing, replacement)
    - Buffers hold data in transit
    - Caches hold frequently used data
- **File organization module** understands files, logical address, and physical blocks
    - Translates logical block # to physical block #
    - Manages free space, disk allocation
- **Logical file system** manages metadata information
    - Translates file name into file number, file handle, location by maintaining file control blocks (**inodes** in UNIX)

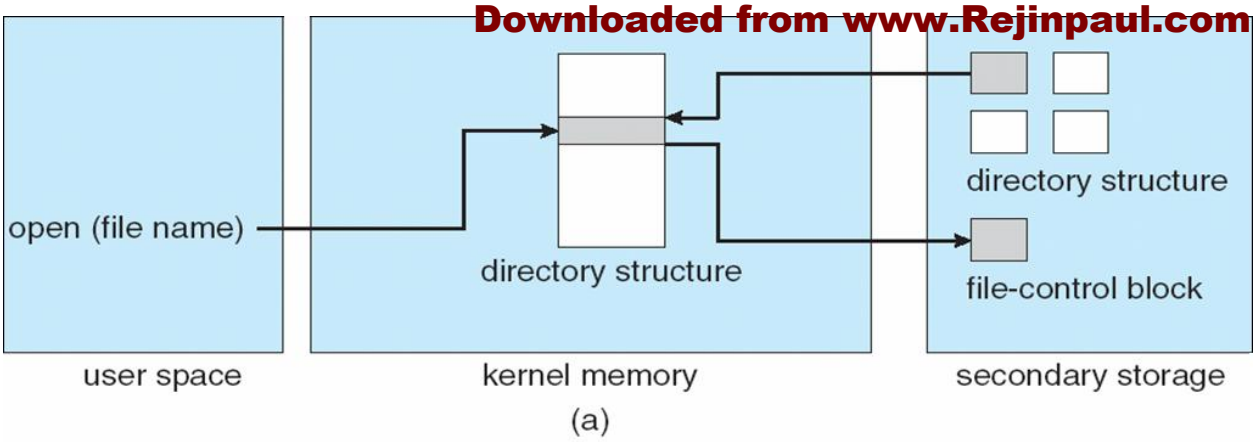
- Directory management
- Protection

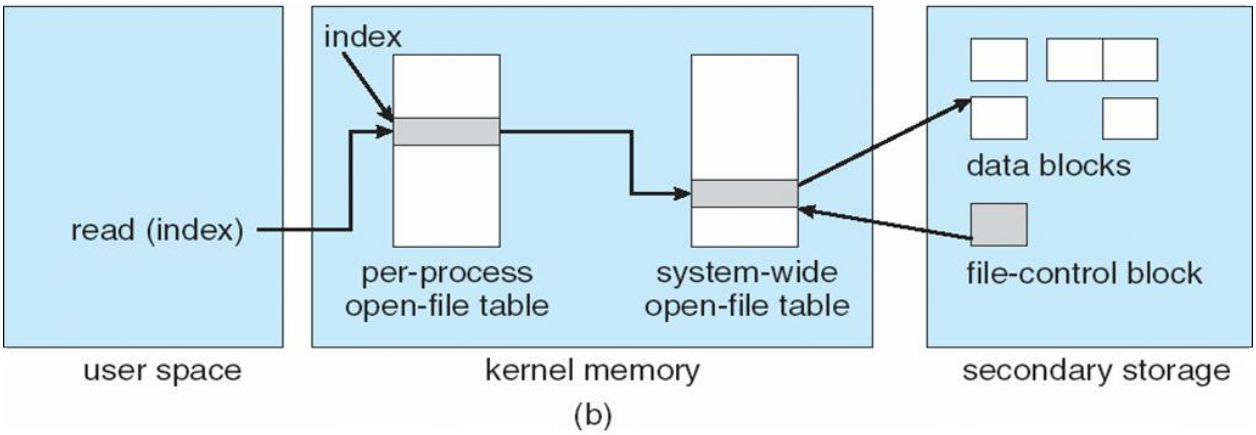
- Layering useful for reducing complexity and redundancy, but adds overhead and can decrease performanceTranslates file name into file number, file handle, location by maintaining file control blocks (**inodes** in UNIX)
    - Logical layers can be implemented by any coding method according to OS designer

FILE SYSTEM IMPLEMENTATION

- We have system calls at the API level, but how do we implement their functions?
    - On-disk and in-memory structures
- **Boot control block** contains info needed by system to boot OS from that volume
    - Needed if volume contains OS, usually first block of volume
- **Volume control block (superblock, master file table)** contains volume details
    - Total # of blocks, # of free blocks, block size, free block pointers or array
- Directory structure organizes the files
    - Names and inode numbers, master file table
- Per-file **File Control Block (FCB)** contains many details about the file
    - inode number, permissions, size, dates
    - NFTS stores into in master file table using relational DB structures

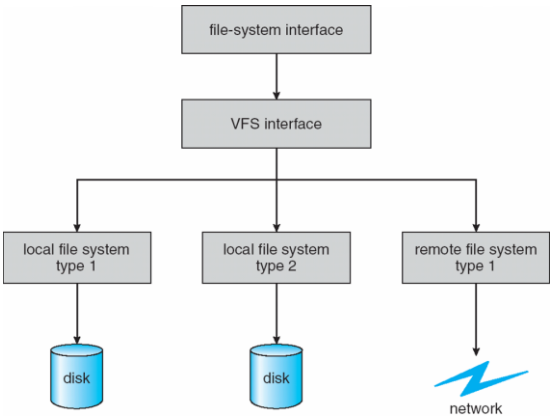






VIRTUAL FILE SYSTEMS

- **Virtual File Systems** (**VFS**) on Unix provide an object-oriented way of implementing file systems
- VFS allows the same system call interface (the API) to be used for different types of file systems
    - Separates file-system generic operations from implementation details
    - Implementation can be one of many file systems types, or network file system
        - Implements **vnodes** which hold inodes or network file details
    - Then dispatches operation to appropriate file system implementation routines



VFS IMPLEMENTATION

- For example, Linux has four object types:
    - inode, file, superblock, dentry
- VFS defines set of operations on the objects that must be implemented
    - Every object has a pointer to a function table
        - Function table has addresses of routines to implement that function on that object
        - For example:
- **int open(. . .)**—Open a file
- **int close(. . .)**—Close an already-open file
- **ssize t read(. . .)**—Read from a file
- **ssize t write(. . .)**—Write to a file
- **int mmap(. . .)**—Memory-map a file

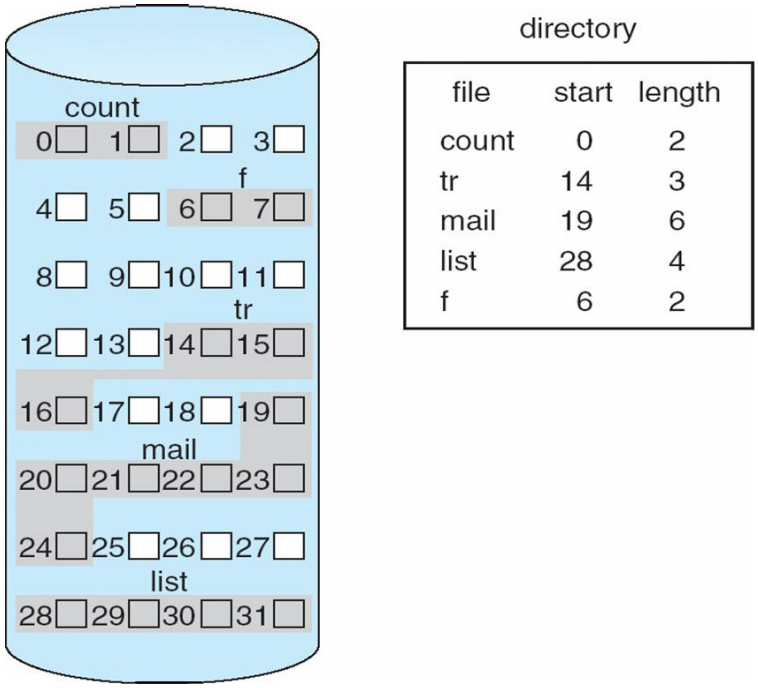
DIRECTORY IMPLEMENTATION

- **Linear list** of file names with pointer to the data blocks
    - Simple to program
    - Time-consuming to execute
        - Linear search time
        - Could keep ordered alphabetically via linked list or use B+ tree
- **Hash Table** – linear list with hash data structure
    - Decreases directory search time
    - **Collisions** – situations where two file names hash to the same location
    - Only good if entries are fixed size, or use chained-overflow method

ALLOCATION METHOD CONTIGUOUS

- An allocation method refers to how disk blocks are allocated for files:
- **Contiguous allocation** – each file occupies set of contiguous blocks
    - Best performance in most cases
    - Simple – only starting location (block #) and length (number of blocks) are required

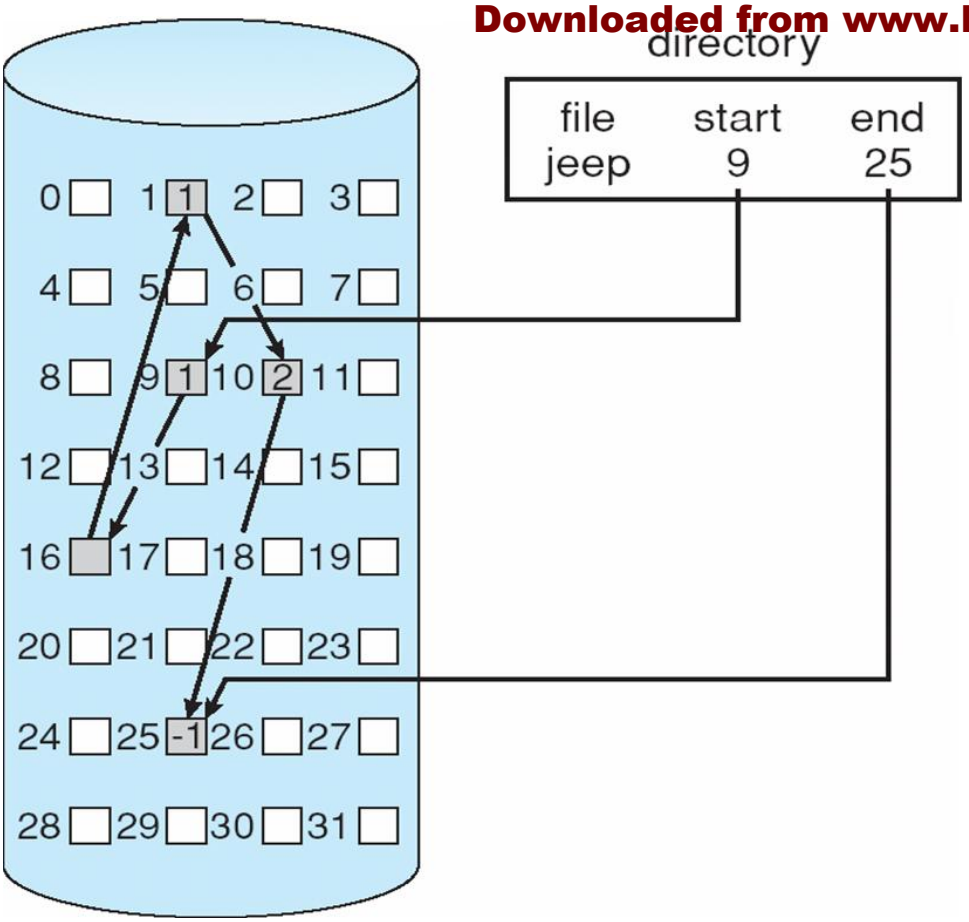
- Problems include finding space for file, knowing file size, external fragmentation, need for **compaction off-line** (**downtime**) or **on-line**



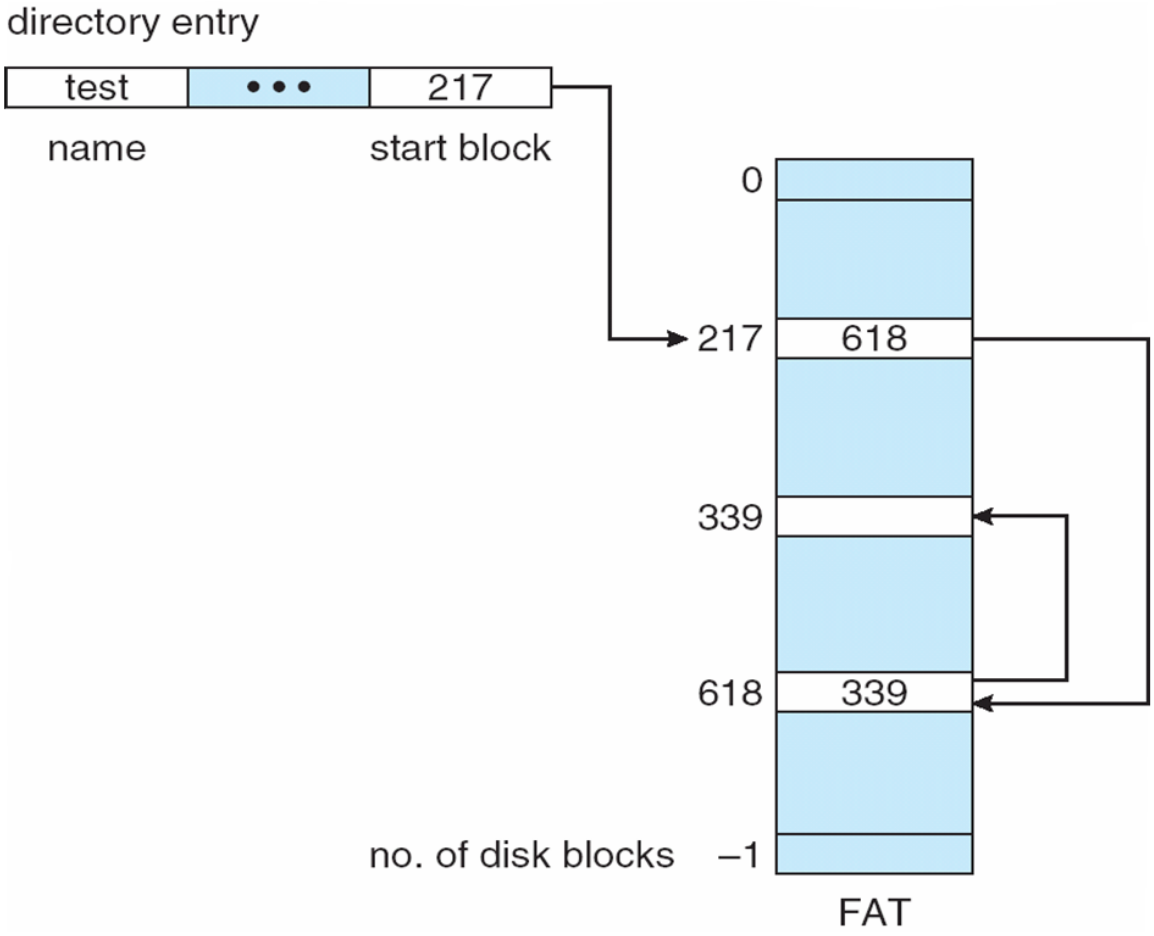
directory

file	start	length
count	0	2
tr	14	3
mail	19	6
list	28	4
f	6	2

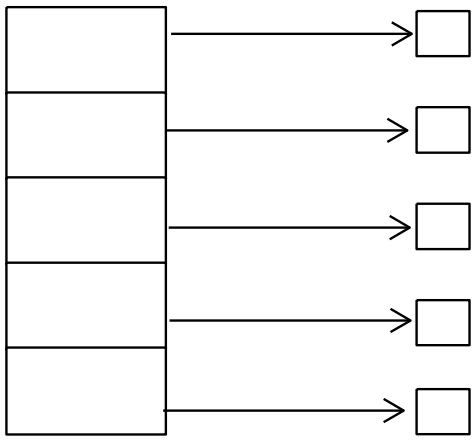
- **Linked allocation** – each file a linked list of blocks
  - File ends at nil pointer
  - No external fragmentation
  - Each block contains pointer to next block
  - No compaction, external fragmentation
  - Free space management system called when new block needed
  - Improve efficiency by clustering blocks into groups but increases internal fragmentation
  - Reliability can be a problem
  - Locating a block can take many I/Os and disk seeks



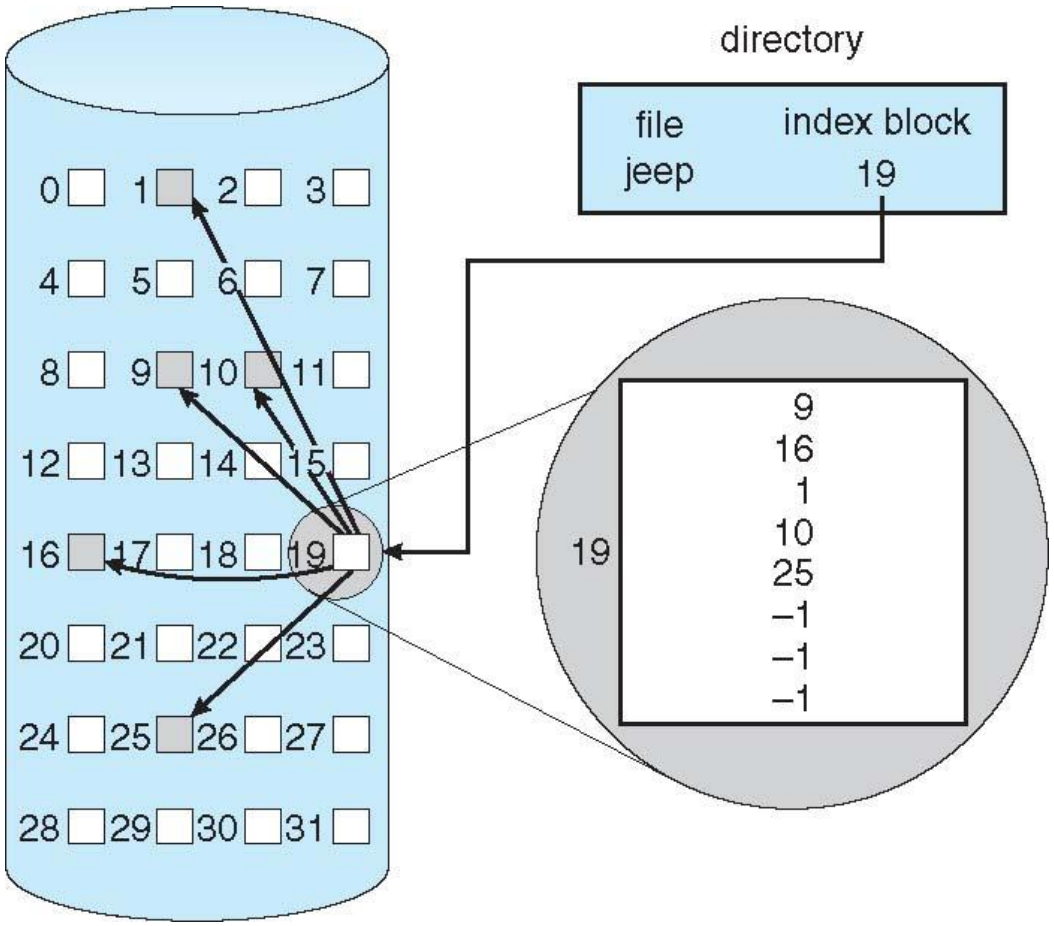
FILE ALLOCATION TABLE



- **Indexed allocation**
- Each file has its own **index block**(s) of pointers to its data blocks
- Logical view



- index table

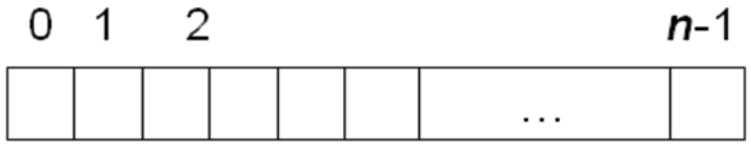


- Need index table
- Random access
- Dynamic access without external fragmentation, but have overhead of index block

- Mapping from logical to physical in a file of maximum size of 256K bytes and block size of 512 bytes. We need only 1 block for index table

FREE SPACE MANAGEMENT

- File system maintains **free-space list** to track available blocks/clusters
    - (Using term "block" for simplicity)
- **Bit vector** or **bit map** (***n*** blocks)



$$bit[i] = \begin{cases} 1 \Rightarrow block[i] \text{ free} \\ 0 \Rightarrow block[i] \text{ occupied} \end{cases}$$

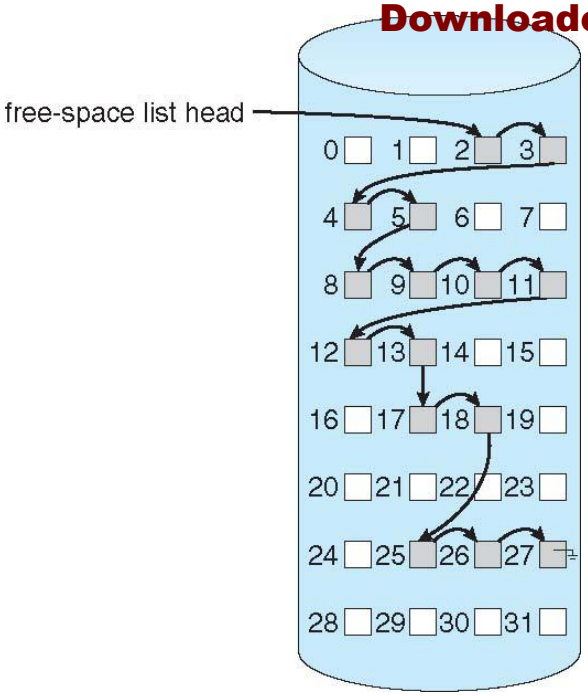
Block number calculation

(number of bits per word) *(number of 0-value words) +offset of first 1 bit

- Bit map requires extra space
    - Example:
        - block size = 4KB = 2^{12} bytes
        - disk size = 2^{40} bytes (1 terabyte)
        - $n = 2^{40}/2^{12} = 2^{28}$ bits (or 32MB)
        - if clusters of 4 blocks -> 8MB of memory
- Easy to get contiguous files

Linked list (free list)

- Cannot get contiguous space easily
- No waste of space
- No need to traverse the entire list (if # free blocks recorded)



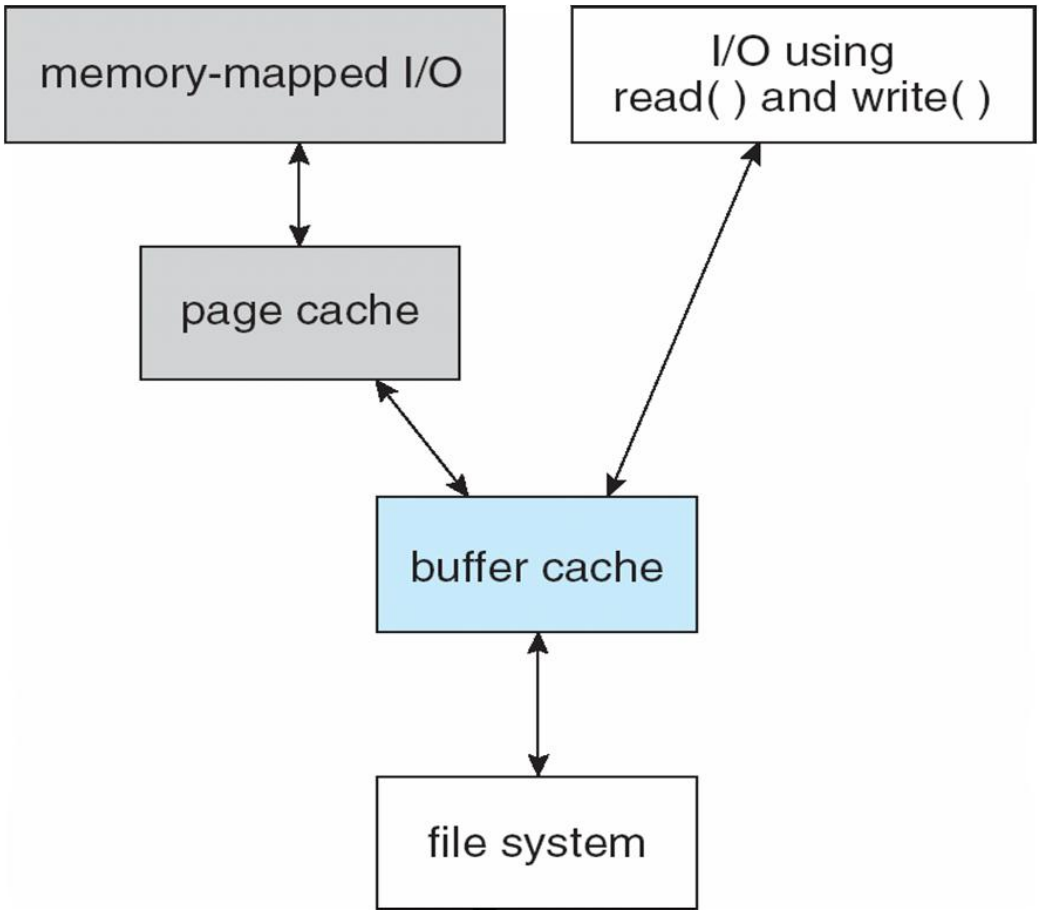
- Grouping
  - Modify linked list to store address of next *n-1* free blocks in first free block, plus a pointer to next block that contains free-block-pointers (like this one)
- Counting
  - Because space is frequently contiguously used and freed, with contiguous-allocation allocation, extents, or clustering
    - Keep address of first free block and count of following free blocks
    - Free space list then has entries containing addresses and counts
- Space Maps
  - Used in **ZFS**
  - Consider meta-data I/O on very large file systems
    - Full data structures like bit maps couldn't fit in memory -> thousands of I/Os
  - Divides device space into **metaslab** units and manages metaslabs
    - Given volume can contain hundreds of metaslabs
  - Each metaslab has associated space map
    - Uses counting algorithm
  - But records to log file rather than file system
    - Log of all block activity, in time order, in counting format

- Metaslab activity -> load space map into memory in balanced-tree structure, indexed by offset
    - Replay log into that structure
    - Combine contiguous free blocks into single entry

- Efficiency dependent on:
    - Disk allocation and directory algorithms
    - Types of data kept in file's directory entry
    - Pre-allocation or as-needed allocation of metadata structures
    - Fixed-size or varying-size data structures

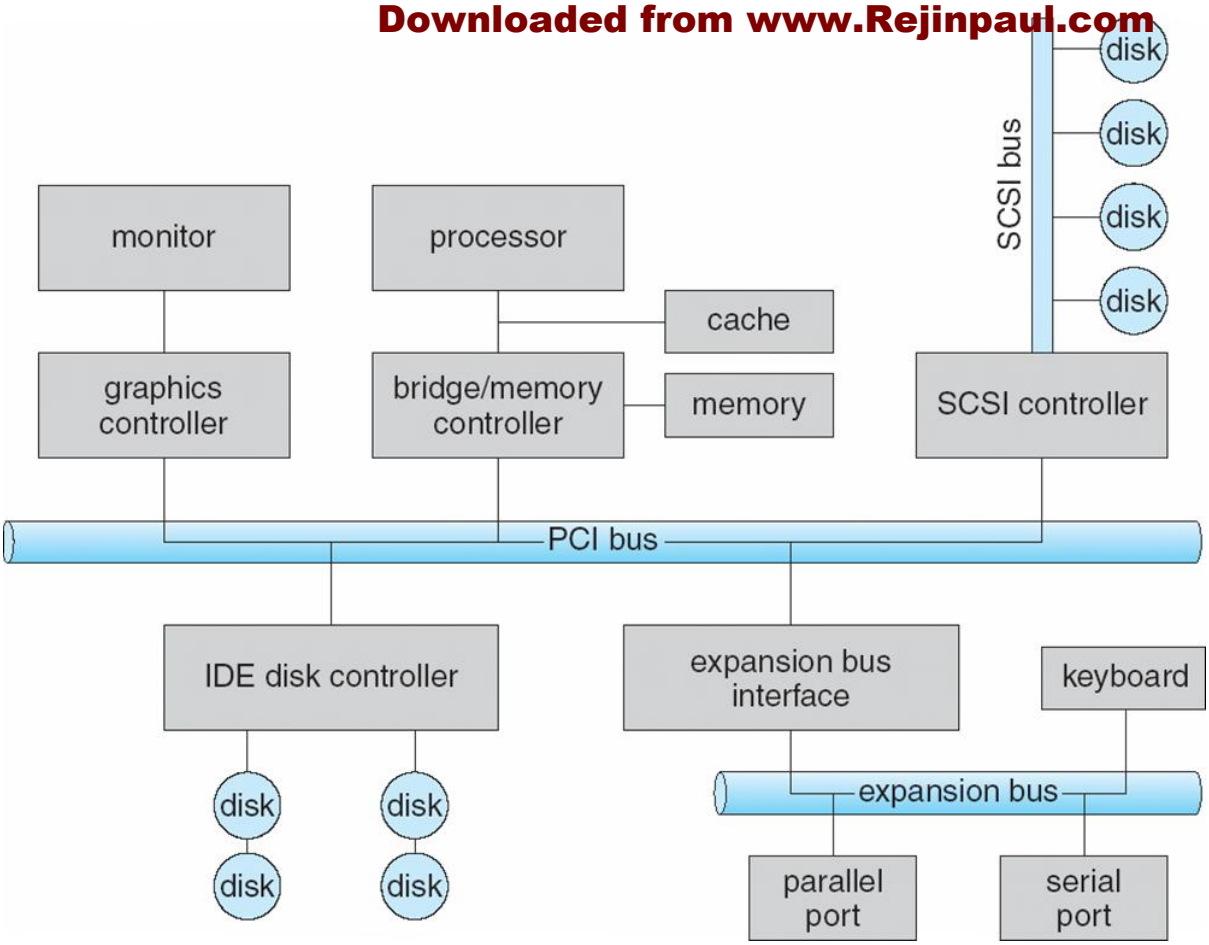
PAGE CACHE

- A **page cache** caches pages rather than disk blocks using virtual memory techniques and addresses
- Memory-mapped I/O uses a page cache
- Routine I/O through the file system uses the buffer (disk) cache
- This leads to the following figure



IO SYSTEMS

- Incredible variety of I/O devices
    - Storage
    - Transmission
    - Human-interface
- Common concepts – signals from I/O devices interface with computer
    - **Port** – connection point for device
    - **Bus - daisy chain** or shared direct access
        - **PCI** bus common in PCs and servers, PCI Express (**PCIe**)
        - **expansion bus** connects relatively slow devices
    - **Controller** (**host adapter**) – electronics that operate port, bus, device
        - Sometimes integrated
        - Sometimes separate circuit board (host adapter)
        - Contains processor, microcode, private memory, bus controller, etc
            - Some talk to per-device controller with bus controller, microcode, memory, etc



- I/O instructions control devices
- Devices usually have registers where device driver places commands, addresses, and data to write, or read data from registers after command execution
    - Data-in register, data-out register, status register, control register
    - Typically 1-4 bytes, or FIFO buffer
- Devices have addresses, used by
    - Direct I/O instructions
    - **Memory-mapped I/O**
        - Device data and command registers mapped to processor address space
        - Especially for large address spaces (graphics)

I/O address range (hexadecimal)	device
000–00F	DMA controller
020–021	interrupt controller
040–043	timer
200–20F	game controller
2F8–2FF	serial port (secondary)
320–32F	hard-disk controller
378–37F	parallel port
3D0–3DF	graphics controller
3F0–3F7	diskette-drive controller
3F8–3FF	serial port (primary)

UNIT V

LINUX SYSTEM – BASIC CONCEPTS

Linodes run **Linux**. Linux is an operating system, just like Windows and Mac OS X. As an operating system, Linux manages your Linode's hardware and provides services your other software needs to run.

Linux is a very hands-on operating system. If running Windows is like driving an automatic, then running Linux is like driving a stick. It can take some work, but once you know your way around Linux, you'll be using the command line and installing packages like a pro. This article aims to ease you into the world of Linux.

This guide is intended to be very beginner-friendly. It takes a Linux 101 approach to explanations for basic concepts. There are a few how-to sections as well, which are intended to get you on your feet with your Linode. At times we'll link off to a different guide that has more details on a particular topic.

Everything on a Linux system is case-sensitive. That means that `photo.jpg`, `photo.JPG`, and `Photo.jpg` are all different files. Usernames and passwords are also case-sensitive.

Linux, like Mac OS X, is based on the Unix operating system. A research team at AT&T's Bell Labs developed Unix in the late 1960s and early 1970s with a focus on creating an operating system that would be accessible and secure for multiple users.

Corporations started licensing Unix in the 1980s and 1990s. By the late 1980s, there was interest in building a free operating system that would be similar to Unix, but that could be tinkered with and redistributed. In 1991, Linus Torvalds released the Linux kernel as free, *open-source* software. Open source means that the code is fully visible, and can be modified and redistributed.

Strictly speaking, Linux is the *kernel*, not the entire operating system. The kernel provides an interface between your Linode's hardware and the input/output requests from applications. The rest of the operating system usually includes many GNU libraries, utilities, and other software, from the Free Software Foundation. The operating system as a whole is known as GNU/Linux.

A Little Bit About Servers

Your Linode is a type of *server*. What's a server? A server is a type of computer that provides services over a *network*, or connected group of computers. When people think about servers, they're usually thinking of a computer that is:

- Always (or almost always) on
- Connected to the Internet
- Contains programs and files for websites and/or other Internet content

Since a server is a type of computer, there are a lot of similarities between a Linode and your home computer. Some important similarities include:

- The physical machine: Your Linode is hosted on a physical machine. It's sitting in one of our data centers.

- The operating system: As we mentioned in the introduction, Linodes use the Linux operating system. It's just another type of operating system like Windows or Mac OS X.
- Applications: Just like you can install applications on your home computer or smartphone, you can install applications on your Linode. These applications help your Linode do things like host a website. WordPress is a popular website application, for example. Applications are also known as *software* and *programs*.
- Files and directories: In the end, whether it's an application or a photo, everything on your Linode is a file. You can create new files, edit and delete old ones, and navigate through directories just like you would on your home computer. In Linux, folders are called *directories*.
- Internet access: Your Linode is connected to the Internet. That's how you connect to it to get everything set up, and how your users connect to it to view your website or download your app.

SYSTEM ADMINISTRATION

Linux System Administration Basics

This presents a collection of common issues and useful tips for Linux system administration. Whether you're new to system administration or have been maintaining systems for some time, we hope these tips are helpful regardless of your background or choice in Linux distributions.

Basic Configuration

These tips cover some of the basic steps and issues encountered during the beginning of system configuration. We provide a general getting started guide for your convenience if you're new to Linode and basic Linux system administration. Additionally, you may find some of our Introduction to Linux Concepts guide useful.

Set the Hostname

Please follow our instructions for setting your hostname. Issue the following commands to make sure it is set properly:

```
1 hostname
2 hostname -f
```

The first command should show your short hostname, and the second should show your fully qualified domain name (FQDN).

Set the Timezone

When setting the timezone of your server, it may be best to set it to the timezone of the bulk of your users. If you're unsure which timezone would be best, consider using universal coordinated time or UTC (i.e. Greenwich Mean Time).

By default, Linode base installs are set to Eastern Standard Time. The following process will set the timezone manually, though many operating systems provide a more elegant method for changing timezones. To change the time zone manually, you must find the proper zone file in `/usr/share/zoneinfo/` and link that file to `/etc/localtime`. See the example below for

common possibilities. Please note that all contents following the double hashes (eg: `##`) are comments and should not be copied into your terminal.

```
ln -sf /usr/share/zoneinfo/UTC /etc/localtime ## for Universal Coordinated Time

ln -sf /usr/share/zoneinfo/EST /etc/localtime ## for Eastern Standard Time

ln -sf /usr/share/zoneinfo/US/Central /etc/localtime ## for American Central time (including DST)

ln -sf /usr/share/zoneinfo/US/Eastern /etc/localtime ## for American Eastern (including DST)
```

To change the time zone in Debian and Ubuntu systems, issue the following command and answer the questions as prompted by the utility:

```
dpkg-reconfigure tzdata
```

In Arch Linux, set the timezone in the `/etc/rc.conf` file by configuring the `TIMEZONE=` setting in the "Localization" section. This line will resemble the following:

/etc/rc.conf

TIMEZONE="America/New_York"

Note that the string specified in `TIMEZONE` refers to the "zoneinfo" file located in or below the `/usr/share/zoneinfo/` directory.

Use the /etc/hosts File

The `/etc/hosts` file provides a list of IP addresses with corresponding hostnames. This allows you to specify hostnames for an IP address once on the local machine, and then have multiple applications connect to external resources via their hostnames. The system of host files predates DNS, and hosts files are *always* checked before DNS is queried. As a result, `/etc/hosts` can be useful for maintaining small "internal" networks, for development purposes, and for managing clusters.

Some applications require that the machine properly identify itself in the `/etc/hosts` file. As a result, we recommend configuring the `/etc/hosts` file shortly after deployment. Here is an example file:

/etc/hosts

127.0.0.1 localhost.localdomain localhost 12.34.56.78 squire.example.com squire

You can specify a number of hostnames on each line separated by spaces. Every line must begin with one and only one IP address. In this case, replace `12.34.56.78` with your machine's IP address. Let us consider a few additional `/etc/hosts` entries:

/etc/hosts

74.125.67.100 test.com 192.168.1.1 stick.example.com

In this example, all requests for the test.com hostname or domain will resolve to the IP address `74.125.67.100`, which bypasses the DNS records for `test.com` and returns an alternate website.

The second entry tells the system to look to `192.168.1.1` for the domain `stick.example.com`. These kinds of host entries are useful for using "private" or "back channel" networks to access other servers in a cluster without needing to access the public network.

Network Diagnostics

The following tips address the basic usage and functionality of a number of tools that you can use to assess and diagnose network problems. If you suspect connectivity issues, including output of the relevant commands in your support ticket can help our staff diagnose your issue. This is particularly helpful in cases where networking issues are intermittent.

Using the Ping Command

The `ping` command tests the connection between the local machine and a remote address or machine. The following command "pings" `google.com` and `74.125.67.100`:

```
1 ping google.com ping 74.125.67.100
```

These commands send a bit of data (i.e. an ICMP packet) to the remote host, and wait for a response. If the system is able to make a connection, for every packet it will report on the "round trip time." Here is the output of four pings of google.com:

```
  64 bytes from yx-in-f100.1e100.net (74.125.45.100): icmp_seq=1 ttl=50
  time=33.8 ms
1 64 bytes from yx-in-f100.1e100.net (74.125.45.100): icmp_seq=2 ttl=50
2 time=53.2 ms
3 64 bytes from yx-in-f100.1e100.net (74.125.45.100): icmp_seq=3 ttl=50
4 time=35.9 ms
  64 bytes from yx-in-f100.1e100.net (74.125.45.100): icmp_seq=4 ttl=50
  time=37.5 ms
```

In this case `yx-in-f100.1e100.net` is the reverse DNS for this IP address. The `time` field specifies in milliseconds that the round trip takes for an individual packet. When you've gathered the amount of information you need, send `Control+C` to interrupt the process. At this juncture, you'll be presented with some statistics. This will resemble:

```
1 --- google.com ping statistics ---
2 4 packets transmitted, 4 received, 0% packet loss, time 3007ms
3 rtt min/avg/max/mdev = 33.890/40.175/53.280/7.679 ms
```

There are several important data points to notice. They are:

- *Packet Loss*, or the discrepancy between the number of packets sent and the number of packets that return successfully.
- *Round Trip Time* statistics on the final line report important information about all the ping responses. For this ping we see that the fastest packet round trip took 33.89 milliseconds. The longest packet took 53.28 milliseconds. The average round trip took 40.175 milliseconds. A single standard deviation unit for these four packets is 7.67 milliseconds.

Use the ping tool to contact a server and ensure that you are able to make a connection. Furthermore, ping is useful as an informal diagnostic tool to measure point-to-point network latency, and as a network connection testing tool.

Using the traceroute Command

The `traceroute` command expands on the functionality of the ping command. `traceroute` provides a report on the path that the packets take to get from the local machine to the remote machine. Route information is useful when troubleshooting a networking issue: if there is packet loss in one of the first few "hops" the problem is often related to the user's local area network (LAN) or Internet service provider (ISP). By contrast, if there is packet loss near the end of the route, the problem may be caused by an issue with the server's connection.

Here is an example of output from a `traceroute` command:

```
1 traceroute to google.com (74.125.53.100), 30 hops max, 40 byte packets
```

1 207.192.75.2 (207.192.75.2) 0.414 ms 0.428 ms 0.509 ms 2 vlan804.tbr2.mmu.nac.net (209.123.10.13) 0.287 ms 0.324 ms 0.397 ms 3 0.e1-1.tbr2.tl9.nac.net (209.123.10.78) 1.331 ms 1.402 ms 1.477 ms 4 core1-0-2-0.lga.net.google.com (198.32.160.130) 1.514 ms 1.497 ms 1.519 ms 5 209.85.255.68 (209.85.255.68) 1.702 ms 72.14.238.232 (72.14.238.232) 1.731 ms 21.031 ms 6 209.85.251.233 (209.85.251.233) 26.111 ms 216.239.46.14 (216.239.46.14) 23.582 ms 23.468 ms 7 216.239.43.80 (216.239.43.80) 123.668 ms 209.85.249.19 (209.85.249.19) 47.228 ms 47.250 ms 8 209.85.241.211 (209.85.241.211) 76.733 ms 216.239.43.80 (216.239.43.80) 73.582 ms 73.570 ms 9 209.85.250.144 (209.85.250.144) 86.025 ms 86.151 ms 86.136 ms 10 64.233.174.131 (64.233.174.131) 80.877 ms 216.239.48.34 (216.239.48.34) 76.212 ms 64.233.174.131 (64.233.174.131) 80.884 ms 11 216.239.48.32 (216.239.48.32) 81.267 ms 81.198 ms 81.186 ms 12 216.239.48.137 (216.239.48.137) 77.478 ms pw-in-f100.1e100.net (74.125.53.100) 79.009 ms 216.239.48.137 (216.239.48.137) 77.437 ms

Often the hostnames and IP addresses on either side of a failed jump are useful in determining who operates the machine where the routing error occurs. Failed jumps are designated by line with three asterisks (e.g. `* * *`).

Furthermore, including `traceroute` information in tickets to Linode support is sometimes useful when trying to diagnose network issues. You may also want to forward `traceroute` information to your Internet Service Provider (ISP) if you suspect that the connectivity issue is with your ISP's network. Recording `traceroute` information is particularly useful if you are experiencing an intermittent issue.

Using the mtr Command

The "mtr" command, like the traceroute tool, provides information about the route that Internet traffic takes between the local system and a remote host. However, `mtr` provides additional information about the round trip time for the packet. In a way, you can think of `mtr` as a combination of traceroute and ping.

Here is the example output of an `mtr` command:

```
1 HOST: squire.example.com  Loss%   Snt   Last   Avg  Best  Wrst StDev
```

1. 256.129.75.4 0.0% 10 0.4 0.4 0.3 0.6 0.1

2. vlan804.tbr2.mmu.nac.net 0.0% 10 0.3 0.4 0.3 0.7 0.1
3. 0.e1-1.tbr2.tl9.nac.net 0.0% 10 4.3 4.4 1.3 11.4 4.1
4. core1-0-2-0.lga.net.google.c 0.0% 10 64.9 11.7 1.5 64.9 21.2
5. 209.85.255.68 0.0% 10 1.7 4.5 1.7 29.3 8.7
6. 209.85.251.9 0.0% 10 23.1 35.9 22.6 95.2 27.6
7. 72.14.239.127 0.0% 10 24.2 24.8 23.7 26.1 1.0
8. 209.85.255.190 0.0% 10 27.0 27.3 23.9 37.9 4.2
9. gw-in-f100.1e100.net 0.0% 10 24.1 24.4 24.0 26.5 0.7

Used without the `--report` flag, `mtr` tracks the speed of the connection in real time until you exit the program. Additionally, be aware that `mtr` will pause for a few moments before generating output. For more information regarding `mtr` consider our guide to diagnosing network issues with mtr.

System Diagnostics

If you're having an issue with your Linode that is neither related to networking, nor another easily diagnosable application issue, it is worthwhile to rule out "hardware" and operating system level issues. Use the following tools to better diagnose and resolve these kinds of issues.

If you determine that you have a problem with memory usage, please reference our document regarding resolving memory usage issues. Use the following tools and approaches to determine the specific cause of your troubles.

Check Current Memory Usage

If you need to see how much memory your system is using at the current moment issue the following command:

```
1 free -m
```

On a moderately utilized Linode 1GB, this command will generate output that resembles the following:

```
1 total       used       free     shared    buffers     cached
```

Mem: 1002 956 46 0 171 357 -/+ buffers/cache: 427 575 Swap: 127 39 88

This output takes a little bit of careful reading to interpret correctly. Out of a total 1002 megabytes of memory (RAM), the system is using 956 megabytes, and has 46 megabytes free. **However**, the system also has 427 megabytes of "stale" data buffered and stored in cache. The operating system will "drop" the caches when and if it needs the space, but retains the cache if there is no other need for the space. It is totally normal for a Linux system to leave old data in RAM until the space is needed, and you should not be alarmed if only a small amount of memory is actually "free."

In the above example, there are 575 megabytes of memory that are actually *free*. This means 575 megabytes are available to your system when you start an additional process or a running application needs more memory.

Monitor IO Usage with vmstat

The `vmstat` tool provides information about memory, swap utilization, IO wait, and system activity. It is particularly useful for diagnosing I/O-related issues.

If you think you're having an I/O issue then run the following command:

```
vmstat 1 20
```

This runs a vmstat every second, twenty times. We find this gives a pretty good sample of the current state of the system. The output generated resembles the following:

```
   procs -----------memory---------- ---swap-- -----io---- -system-- ----cpu--
 1 --
 2  r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id
 3 wa
 4  0  0      4  32652  47888 110824    0    0     2    15   15    0  0 100  0
 5  0  0      4  32644  47888 110896    0    0     4   106  123    0  0 100  0
 6  0  0      4  32644  47888 110912    0    0     0    70  112    0  0 100  0
 7  0  0      4  32644  47888 110912    0    0     0    92  121    0  0 100  0
 8  0  0      4  32644  47888 110912    0    0    36    97  136    0  0 100  0
 9  0  0      4  32644  47888 110912    0    0     0    96  119    0  0 100  0
10  0  0      4  32892  47888 110912    0    0     4    96  125    0  0 100  0
11  0  0      4  32892  47888 110912    0    0     0    70  105    0  0 100  0
12  0  0      4  32892  47888 110912    0    0     0    97  119    0  0 100  0
13  0  0      4  32892  47888 110912    0    0    32    95  135    0  0 100  0
14  0  0      4  33016  47888 110912    0    0     0    75  107    0  0 100  0
15  0  0      4  33512  47888 110912    0    0    24   113  134    0  0 100  0
16  0  0      4  33512  47888 110912    0    0     0   175  244    0  0 100  0
17  0  0      4  33512  47888 110912    0    0     0    92  148    0  0 100  0
18  0  0      4  33512  47888 110912    0    0     0   114  162    0  0 100  0
19  0  0      4  33512  47888 110912    0    0    36   100  157    0  0 100  0
20  0  0      4  33388  47888 110912    0    0     0   116  166    0  0 100  0
21  0  0      4  33388  47888 110912    0    0     0    97  157    0  0 100  0
22  0  0      4  33388  47888 110912    0    0     0    89  144    0  0 100  0
    0  0      4  33380  47888 110912    0    0     0   181  185    0  0 99  0
```

The memory and swap columns provide the same kind of information provided by the "free -m" command, albeit in a slightly more difficult to comprehend format. The most salient information produced by this command is the `wa` column, which is the final column in most implementations. This field displays the amount of time the CPU spends waiting for IO operations to complete.

If this number is consistently and considerably higher than 0, you might consider taking measures to address your IO usage. However, if the `vmstat` output resembles the above, you can be sure in the knowledge that you're not experiencing an IO-related issues.

If you are experiencing an intermittent issue, you will need to run `vmstat` *when* you experience the issue in order to properly diagnose or rule out an IO issue. `vmsat` output can sometimes help support diagnose problems.

Monitor Processes, Memory, and CPU Usage with htop

If you want a more organized and real-time view of the current state of your system, we recommend a tool called `htop`. This is not installed by default on most systems. To install `htop`, issue one of the following commands, depending on which distribution you use:

```
apt-get install htop
yum install htop
pacman -S htop
```

```
4 emerge sys-process/htop
```

Now, at a command line, issue the following command:

```
1 htop
```

You can quit at any time by pressing the `F10` or `Q` keys. There are a couple of `htop` behaviors that may not be initially intuitive. Take note of the following:

- The memory utilization graph displays used memory, buffered memory, and cached memory. The numbers displayed at the end of this graph reflect the total amount of memory available and the total amount memory on the system as reported by the kernel.
- The default configuration of `htop` presents all application threads as independent processes, which is non-intuitive. You can disable this by selecting the "setup" option with `F2`, then "Display Options," and then toggling the "Hide userland threads" option.
- You can toggle a "Tree" view with the `F5` key that usefully displays the processes in a hierarchy and shows which processes were spawned by which other processes. This is helpful in diagnosing a problem when you're having trouble figuring out what processes are what.

File System Management

Historically, web developers and editors have used the FTP protocol to transfer and manage files on a remote system. FTP, however, is very insecure and inefficient for managing the files on a system when you have SSH access.

If you're new to administering systems and the Linux world, you might consider our "Tools & Reference" section and articles including: "installing and using WinSCP" using rsync to synchronize files and "using SSH and the terminal."

As always, if you are giving other users access to upload files to your server, it would be wise to consider the security implications of all additional access that you grant to third parties seriously.

How to Upload files to a Remote Server

If you're used to using an FTP client, OpenSSH (which is included and active with all of the Linode provided installation templates) allows you to use an FTP-like interface over the SSH protocol. Known as "SFTP," many clients support this protocol, including: "WinSCP" for Windows, "Cyberduck" for Mac OS X, and "Filezilla" for Linux, OS X, and Windows desktops.

If you are accustomed to FTP, SFTP is great option. Do note that by default, whatever access a user has to a file system at the command line, they will also have over SFTP. Consider file permissions very carefully.

Conversely, you can use Unix utilities including `scp` and rsync to securely transfer files to your Linode. On local machine, a command to copy `team-info.tar.gz` would look like:

```
1 scp team-info.tar.gz squire@lollipop.example.com:/home/squire/backups/
```

The command, `scp`, is followed by the name of the file on the local file system to be transferred. Next is the username and hostname of the remote machine, separated by an "at" sign (e.g. `@`). Following the hostname, there is a colon (e.g. `:`) and the path on the remote server where the file should be uploaded to. Taken another way, this command would be:

```
1 scp [/path/to/local/file] [remote-username]@[remote-
  hostname]:[/path/to/remote/file]
```

This command is available by default on OS X and Linux machines. You can use it to copy files to a Linode, as well as between remote servers. If you have SSH keys deployed, you can use the `scp` command without entering a password for every transfer.

The syntax of `scp` follows the form `scp [source] [destination]`. You can copy files from a remote host to the local machine by reversing the order of the paths in the above example.

How to Protect Files on a Remote Server

Because Linode servers are network accessible and often have a number of distinct users, maintaining the security of files is often an important concern. We recommend you familiarize yourself with our basic security guide. Furthermore, our documentation of access control with user accounts and permissions may provide additional insight.

Additionally, we suggest the following best practices for maintaining security:

- Only give users the permission to do what they need to. This includes application specific users.
- Only run services on public interfaces that you are actively using. One common source of security vulnerabilities are in daemons that are left running and unused. This includes database servers, HTTP development servers, and FTP servers.
- Use SSH connections whenever possible to secure and encrypt the transfer of sensitive information.

Understanding and Using Sym Linking

"Symbolic Linking," colloquially "sym linking," allows you to create an object in your filesystem that points to another object on your filesystem. This is useful when you need to provide users and applications access to specific files and directories without reorganizing your folders. This way you can provide restricted users access to your web-accessible directories without moving your `DocumentRoot` into their home directories.

To create a symbolic link, issue a command in the following format:

```
1 ln -s /home/squire/config-git/etc-hosts /etc/hosts
```

This creates a link of the file `etc-hosts` at the location of the system's `/etc/hosts` file. More generically. this command would read:

```
1 ln -s [/path/to/target/file] [/path/to/location/of/sym/link]
```

Note the following features of the link command:

- The final term, the location of the link, is optional. If you opt to omit the link destination, a link will be created in the current directory with the same name as the file you're linking to.
- When specifying the location of the link, ensure that path does not have a final trailing slash. You can create a sym link that *targets* a directory, but sym links cannot terminate with slashes.
- You may remove a symbolic link without affecting the target file.
- You can use relative or absolute paths when creating a link.

How to Manage and Manipulate Files on a Linux System

If you're new to using Linux and manipulating files on the terminal interface we encourage you to consider our using the terminal document. This tip provides an overview of basic file management operations.

To **copy** files, issue the following command:

cp /home/squire/todo.txt /home/squire/archive/todo.01.txt

This copies `todo.txt` to an archive folder, and adds a number to the file name. If you want to recursively copy all of the files and subdirectories in a directory to another directory, use the `-R` option. This command looks like:

```
1 cp -R /home/squire/archive/ /srv/backup/squire.01/
```

If you need to **move** a file or directory, use the following command:

```
1 mv /home/squire/archive/ /srv/backup/squire.02/
```

You can also use the `mv` command to rename a file.

To **delete** a file, issue a command in the following form:

```
1 rm scratch.txt
```

This will delete the `scratch.txt` file from the current directory.

For more information about file system navigation and manipulation, please consider our documentation of file system navigation in the using the terminal document.

Package Management

Contemporary Linux systems use package management tools to facilitate the installation and maintenance of all software on your system. For more in-depth coverage of this topic, please reference our package management guide.

While package management provides a number of powerful features, it is easy to obviate the benefits of package management. If you install software manually without package management tools, it becomes very difficult to keep your system up to date and to manage complex dependencies. For these reasons, we recommend installing all software through package management tools unless other means are absolutely necessary. The following tips outline a couple of basic package management tasks.

How to Know What Packages are Installed on Your System

Because packages are so easy to install, and often pull in a number of dependencies, it can be easy to lose track of what software is installed on your system. The following commands provide a list of installed packages on your system.

For Debian and Ubuntu systems, issue the following command:

```
dpkg -l
```

The following example presents the first few lines of the output of this command on a production Debian Lenny system.

```
  ||/ Name             Version               Description
  +++-================================-=====================================-
================================
ii  adduser          3.110                 add and remove users and groups
ii  apache2-mpm-itk      2.2.6-02-1+lenny2        multiuser MPM for Apache
2.2
ii  apache2-utils        2.2.9-10+lenny4          utility programs for
webservers
ii  apache2.2-common         2.2.9-10+lenny4          Apache HTTP Server
common files
ii  apt              0.7.20.2+lenny1          Advanced front-end for dpkg
ii  apt-utils            0.7.20.2+lenny1          APT utility programs
ii  bash             3.2-4                 The GNU Bourne Again SHell
```

For CentOS and Fedora systems, issue the following command:

```
yum list installed
```

The following example presents a few relevant lines of the output of this command:

```
MAKEDEV.i386                3.23-1.2              installed
SysVinit.i386               2.86-15.el5               installed
```

CentOS and Fedora systems provide the name of the package (e.g. `SysVinit`), the architecture it was compiled for (e.g. `i386`), and the version of the build installed on the system (e.g. `2.86-15.el5`).

For Arch Linux systems, issue the following command:

```
pacman -Q
```

This command provides a total list of all packages installed on the system. Arch also allows you to filter these results to display only packages that were explicitly installed (with the `-Qe` option) or that were installed as dependencies (with the `-Qd` option). The above command is thus the union of the output of the following commands:

```
pacman -Qe
pacman -Qd
```

The following is a sample of the format of this output:

```
perl-www-mechanize 1.60-
perl-yaml 0.70-1
pkgconfig 0.23-1
procmail 3.22-2
python 2.6.4-1
rsync 3.0.6-1
```

For Gentoo Linux systems, issue the following command:

```
emerge -evp --deep world
```

The following is a sample of the format of this output:

```
These are the packages that would be merged, in order:


   Calculating dependencies... done!
   [ebuild   R   ] sys-libs/ncurses-5.6-r2  USE="unicode -debug -doc -gpm -
minimal -nocxx -profile -trace" 0 kB
   [ebuild   R   ] virtual/libintl-0  0 kB
   [ebuild   R   ] sys-libs/zlib-1.2.3-r1  0 Kb
```

Because there are often a large number of packages installed on any given system, the output of these commands is often quite large. As a result, it is often useful to use tools like grep and `less` to make these results more useful. For example, the command :

```
dpkg -l | grep "python"
```

will return a list of all packages with the word python in their name or description. Similarly, the following command:

```
dpkg -l | less
```

will return the same list as the plain "`dpkg -l`; however, the results will appear in the `less` pager, which allows you to search and scroll more easily.

You can append `| grep "[string]"` to these commands to filter package list results, or `| less` to display the results in a pager, regardless of distribution.

How to Discover Package Names and Information

Sometimes the name of a package doesn't correspond to the name that you may associate with a given piece of software. As a result, most package management tools make provide an interface to search the package database. These search tools may be helpful if you're looking for a specific piece of software but don't know what it's called.

For Debian and Ubuntu systems, issue the following command:

```
apt-cache search [package-name]
```

This will search the local package database for a given term and generate a list with brief descriptions. An excerpt of the output for `apt-cache search python` follows:

```
txt2regex - A Regular Expression "wizard", all written with bash2 builtins
vim-nox - Vi IMproved - enhanced vi editor
vim-python - Vi IMproved - enhanced vi editor (transitional package)
vtk-examples - C++, Tcl and Python example programs/scripts for VTK
zope-plone3 - content management system based on zope and cmf
zorp - An advanced protocol analyzing firewall
groovy - Agile dynamic language for the Java Virtual Machine
python-django - A high-level Python Web framework
python-pygresql-dbg - PostgreSQL module for Python (debug extension)
python-samba - Python bindings that allow access to various aspects of
Samba
```

Note that `apt-cache search` queries the full records for all of the packages, and not simply the titles and the descriptions displayed here, hence the inclusion of `vim-nox` and `groovy` which both mention python in their descriptions. To see the full record on a package issue the following command:

```
1 apt-cache show [package-name]
```

This provides information regarding the maintainer, the dependencies, the size, the homepage of the upstream project, and a description of the software. This command can be used to provide additional information about a package from the command line.

For CentOS and Fedora systems, issue the following command:

```
1 yum search [package-name]
```

This generates a list of all packages available in the package database that match the given term. See the following excerpt for an example of the output of `yum search wget`:

```
1 Loaded plugins: fastestmirror
2  Loading mirror speeds from cached hostfile
3   * addons: centos.secsup.org
4   * base: centos.secsup.org
5   * extras: centos.secsup.org
6   * updates: styx.biochem.wfubmc.edu
7  ================================= Matched: wget
8 =================================
   wget.i386 : A utility for retrieving files using the HTTP or FTP protocols.
```

You can use the package management tools to discover more information about a specific package. Use the following command to get a full record from the package database:

```
1 yum info [package-name]
```

This output presents more in-depth information concerning the package, its dependencies, origins, and purpose.

For Arch Linux systems, issue the following command:

```
1 pacman -Ss [package-name]
```

This will perform a search of the local copy of the package database. Here is an excerpt of results for a search for “python:

```
1 extra/twisted 8.2.0-1
2     Asynchronous networking framework written in Python.
3 community/emacs-python-mode 5.1.0-1
4     Python mode for Emacs
```

The terms “extra” and “community” refer to which repository the software is located in. This level of specificity is unnecessary when specifying packages to install or display more information about. To request more information about a specific package issue a command in the following form:

```
1 pacman -Si [package-name]
```

Running `pacman` with the `-Si` option generates the package's record from the database. This record includes information about dependencies, optional dependencies, package size, and a brief description.

For Gentoo Linux systems, issue one of the following commands:

```
1 emerge --search [package-name]
2 emerge --searchdoc [package-name]
```

The first command only searches the database for package names. The second command searches through the database for package names and descriptions. These commands will allow you to search your local package tree (i.e. portage) for the specific package name or term. The output of either command is similar to the excerpt presented bellow.

```
 1 Searching...
 2  [ Results for search key : wget ]
 3  [ Applications found : 4 ]
 4
 5  *  app-emacs/emacs-wget
 6        Latest version available: 0.5.0
 7        Latest version installed: [ Not Installed ]
 8        Size of files: 36 kB
 9        Homepage:      http://pop-club.hp.infoseek.co.jp/emacs/emacs-wget/
10        Description:   Wget interface for Emacs
11        License:       GPL-2
```

Because the output provided by the `emerge --search` command is rather verbose, there is no "show more information" tool, unlike other distributions' tools. The `emerge --search` command accepts input in the form of a regular expression if you need to narrow results even further.

Since there are often a large number of results for package searches, these commands output a great quantity of text. As a result it is often useful to use tools like grep and `less` to make these results easier to scroll. For example, the command :

```
1 apt-cache search python | grep "xml"
```

will return the subset of the list of packages which matched for the search term "python," and that mention xml in their name or short description. Similarly, the following command:

```
1 apt-cache search python | less
```

will return the same list as the plain `apt-cache search python` but the results will appear in the `less` pager. This allows you to search and scroll more conveniently.

You can append `| grep "[string]"` to any of these commands to filter package search results, or `| less` to display the results in the `less` pager, regardless of distribution.

Text Manipulation

Among Linux and UNIX-like systems, nearly all system configuration information is stored and manipulated in plain text form. These tips provide some basic information regarding the manipulation of text files on your system.

How to Search for a String in Files with grep

The `grep` tool allows you to search a stream of text, such as a file or the output of a command, for a term or pattern matching a regular expression.

To use the `grep` tool, issue a command in the following format:

```
1 grep "^Subject:.*HELP.*" /home/squire/mbox
```

This will search the mail spool for subject lines (i.e. begins with the word "Subject:"), beginning with any number of characters, containing the word "help" in upper case, and followed by any number of additional characters. `grep` would then print these results on the terminal.

`grep` provides a number of additional options that, if specified, force the program to output the context for each match (e.g. with `-C 2` for two lines of context). With `-n`, `grep` outputs the line number of the match. With `-H`, `grep` prints the file name for each match, which is useful when you "grep" a group of files or "grep" recursively through a file system (e.g. with `-r`). Consider the output of `grep --help` for more options.

To grep a group of files, you can specify the file with a wildcard, as in the following example:

```
1 grep -i "morris" ~/org/*.txt
```

This will find and match against every occurrence of the word "morris," while ignoring case (because of the option for `-i`). `grep` will search all files in the `~/org/` directory with a .txt extension.

You can use `grep` to filter the results of another command that sends output to standard out (e.g. `stdout`). This is accomplished by "piping" the output of one command "into `grep`." For instance:

```
1 ls /home/squire/data | grep "1257"
```

In this example, we assume that the `/home/squire/data` directory contains a large number of files that have a UNIX time stamp in their file name. The above command will filter the output to only display those tiles that have the four digits "1257" in their file name. Note, in these cases `grep` only filters the output of `ls` and does not look into file contents. For more information regarding `grep` consider the full documentation of the grep command.

How to Search and Replace Across a Group of Files

While the grep tool is quite powerful for filtering text on the basis of regular expressions, if you need to edit a file or otherwise manipulate the text you may use the `sed` tool. `sed`, or the Stream EDitor, allows you search for a regular expression pattern and replace it with another string.

`sed` is extremely powerful, and we recommend that you back up your files and test your `sed` commands thoroughly before running them, particularly if you're new to using `sed`. Here is a very simple `sed` one-liner, intended to illustrate its syntax.

```
1 sed -i `s/^good/BAD/` morning-star.txt
```

This replaces occurrences of the word "good" occurring at the beginning of a file (as noted by the ^) with the string "BAD" in the file `morning-star.txt`. The option `-i` tells `sed` to perform the replacements "in place." `sed` can make backups of the files it edits if you specify a suffix after the `-i` option, as in `-iBAK`. In the above command this option would save the original file as `morning-star.txt.BAK` before making changes.

We can surmise that the format of a `sed` statement is:

```
1 's/[regex]/[replacement]/'
```

To match literal slashes (e.g. `/`), you must escape them with a backslash (e.g. `\`). As a result, to match a `/` character you would use `\/` in the `sed` expression. If you are searching for a string that has a number of slashes, you can replace the slashes which another character. For instance:

```
1 's|r/e/g/e/x|regex|'
```

This would strip the slashes from the string `r/e/g/e/x` so that this string would be `regex` after running the `sed` command on the file that contains the string.

The following example, from our migrating a server to your Linode document, searches and replaces one IP address with another. In this case `98.76.54.32` is replaced with `12.34.56.78`:

```
1 sed -i 's/98\.76\.54\.32/12\.34\.56\.78/'
```

In the above example, period characters are escaped as `\.`. In regular expressions the full-stop (period) character matches to any character.

Once again, `sed` is a very powerful and useful tool; however, if you are unfamiliar with it, we strongly recommend testing your search and replace patterns before making any edit of consequence. For more information about `sed` consider the full documentation of text manipulation with sed.

How to Edit Text Interactively

In many Linode Library documents, you may be instructed to edit the contents of a file. To do this, you need to use a text editor. Most of the distribution templates that Linode provides come with an implementation of the vi/vim text editor and the nano text editor. These are small, lightweight, and very powerful text editors that allow you manipulate the text of a file from the terminal environment.

There are other options for text editors, notably emacs and "zile." Feel free to install these programs using your operating system's package manager. Make sure you search your package database so that you can install a version compiled without GUI components (i.e. X11).

To open a file, simply issue a command beginning with the name of the editor you wish to run followed by the name of the file you wish to edit. Here are a number of example commands that open the `/etc/hosts` file:

```
1 nano /etc/hosts
2 vi /etc/hosts
3 emacs /etc/hosts
4 zile /etc/hosts
```

When you've made edits to a file, you can save and exit the editor to return to the prompt. This procedure varies between different editors. In emacs and zile, the key sequence is the same: depress control and type x and s to save. This operation is typically notated "C-x C-s" and then "C-x C-c" to close the editor. In nano, press Control-O (notated \^O) and confirm the file name to write the file, and type \^X to exit from the program.

Since vi and vim are modal editors, their operation is a bit more complex. After opening a file in vi, you can enter "insert" mode by pressing the "i" key; this will let you edit text in the conventional manner. To save the file, you must exit into "normal" mode by pressing the escape key (`Control-[` also sends escape), and then type `:wq` to write the file and quit the program.

This provides only the most basic outline of how to use these text editors, and there are numerous external resources which will provide a more thorough introduction for more advanced use of this software.

Web Servers and HTTP Issues

Linodes do not come with any particular web server installed by default. You have the choice and power to install and configure your web server as you see fit. This allows you to deploy a configuration in a way that makes sense for your application and desired use case. The Linode Library contains a number of documents regarding the installation and maintenance of various web servers.

The following tips cover a number of basic web serving tasks and functions, as well as some guidance for users new to the world of web servers.

How to Serve Websites

Web servers work by listening on a TCP port, typically port 80 for "http" and port 443 for "https." When a visitor makes a request for content, the servers respond by delivering the resource requested. Typically resources are specified with a URL that contains the protocol, `http` or `https`; a colon and two slashes, `://`; hostname or domain, `www.example.com` or `squire.example.com`; and the path to a file, `/images/avatar.jpg`, or `index.html`. Thus a full URL would resemble: `http://www.example.com/images/avatar.jpg`.

In order to provide these resources to connected users, your Linode needs to be running a web server. There are multiple different HTTP servers and countless configurations to provide support for various web development frameworks. The three most popular general use web servers are the Apache HTTP server, Lighttpd server ("Lighty"), and nginx server ("Engine X"). Each server has its strengths and weaknesses, and your choice depends largely on your experience and the nature of your needs.

Once you've chosen a web server, you need to decide what (if any) scripting support you need to install. Scripting support allows you to run dynamic content with your web server and program server side scripts in languages such as Python, PHP, Ruby, and Perl.

If you need a full web application stack, we encourage you to consider one of our more full-featured LAMP stack guides. If you need support for a specific web development framework, consult our tutorials for installing and using specific web development frameworks.

How to Choose a Web Server

In most situations, end users are completely oblivious to which web server you use. As a result, choosing a web server is often a highly personal decision based on the comfort of the administrator and the requirements of the deployment in question. This can be a challenge for the new systems administrator. This tip attempts to offer some guidance by providing some background and information which might be helpful during this process.

The Apache HTTP Server is considered by many to be the *de facto* standard web server. It is the most widely deployed open source web server, its configuration interface has been stable for many years, and its modular architecture allows it to function in many different types of deployments. Apache forms the foundation of the LAMP stack, and contains superb support for integrating dynamic server-side applications into the web server.

By contrast, web servers like Lighttpd and nginx are highly optimized for serving static content in an efficient manner. If you have a deployment where server resources are limited and are facing a great deal of demand, consider one of these servers. They are very functional and run very well with minimal systems resources. Lighttpd and nginx can be more complex to set up than Apache, and can be difficult to configure with regards to integration with dynamic content interpreters. Furthermore, as these servers are more directed at niche use cases, there are more situations and applications which remain undocumented.

Finally the Cherokee web server provides a general purpose web server with an easy to configure interface. Cherokee might be a good option for some basic deployments.

Remember that the choice of web servers is often contextually determined. Specific choices depend on factors like: the type of content you want to serve, the demand for that content, and your comfort with that software as an administrator.

Reading Apache Logs

Often, when there is something wrong with an Apache web sever configuration or site, it is difficult to determine what the cause of the error is from the behavior of the web server. There are a number of common issues with which you might begin your troubleshooting efforts. However, when more complex issues arise it is important to review the Apache error logs.

By default, error logs are located in the /var/log/apache2/error.log file. You can track or “tail” this log with the following command:

```
1 tail -F /var/log/apache2/error.log
```

In the default virtual host configurations suggested in our Apache installation and LAMP guides, we suggest the following error logging setup:

Apache Virtual Host Configuration

ErrorLog /srv/www/example.com/logs/error.log CustomLog
/srv/www/example.com/logs/access.log combined

Where `bucknell.net` represents the name of your virtual host, and the location of relevant files. These configuration directives make Apache create two log files that contain logging information specific to that virtual host. This allows you to easily troubleshoot errors on specific virtual hosts. To track or tail the error log, issue the following command:

```
1 tail -F /srv/www/example.com/logs/error.log
```

This will allow you to see new error messages as they appear. Often problems can be diagnosed by using specific parts of an error message from an Apache log as a term in Web search (e.g. Google.) Common errors to look for include:

- Missing files, or errors in file names.
- Permissions errors.
- Configuration errors.
- Dynamic code execution or interpretation errors.

DNS Servers and Domain Names

The *Domain Name System*, or DNS, is the service that the Internet uses to associate the hard to remember and manage IP addresses with more human-usable domain names. These tips address several specific DNS related tasks. To learn more about DNS consider our overview of the domain name system. If you are familiar with DNS and just need to figure out how to set up your DNS server, consider our documentation of the Linode DNS manager.

Using CNAMEs to Redirect DNS Queries

CNAME DNS records make it possible to redirect requests for one hostname or domain to another hostname or domain. This is useful in situations where you want to direct requests for one domain to another, but don't want to set up the web-server to handle requests.

CNAMEs are **only** valid when pointing from one domain to another. If you need to redirect a full URL, you will need to set up a web server and configure redirection and/or virtual hosting on the server level. CNAMEs will allow you to redirect subdomains, such as `team.example.com`, to other subdomains or domains, such as `jack.example.org`. CNAMEs must point a valid a domain that has a valid A Record, or to another CNAME.

Although limited in their capabilities, CNAMEs can be quite useful in some situations. In particular, if you need to change the hostname of a machine, CNAMEs are quite useful. To learn how to set up CNAME records with the Linode Manager, consult our documentation of the Linode DNS Manager.

How to Set Up Subdomains

When reading domain names, we commonly refer to parts before the main or first-level domain as "sub-domains." For example, in the domain `team.example.com`, `team` is a sub-domain for the root domain `example.com`.

If you want to create and host a sub-domain, consider the following process:

First we need to create an A Record

in the DNS zone for the domain. This is easily accomplished when using the Linode DNS Manager. As always, you may host the DNS for your domain with any provider you choose.

In order for your server to respond to requests for this domain, you must set up a server to respond to these requests. For web servers like Apache this requires configuring a new virtual

host. For XMPP Servers you must configure an additional host to receive the requests for this host. For more information, consult the documentation for the specific server you wish to deploy.

Once configured, subdomains function identically to first-level domains on your server in almost all respects. If you need to, you can set up HTTP redirection for the new sub domain.

SMTP Servers and Email Issues

We provide a number of guides that cover email-related issues. The following tips attempt to further demystify email management.

Choosing an Email Solution

There are two major components of the email stack that are typically required for basic email functionality. The most important part of the tool chain is the SMTP server or "Mail Transfer Agent." The MTA, as it is often called, sends mail from one server to another. The second crucial part of an email system is a server that permits users to access and download that mail from the server to their own machine. Typically these server use a protocol such as POP3 or IMAP to provide remote access to the mailbox.

There are additional components in the email server tool chain. These components may or may not be optional depending on the requirements of your deployment. They include filtering and delivery tools like procmail, anti-virus filters like ClamAV, mailing list managers like MailMan, and spam filters like SpamAssassin. These components function independently of which MTA and remote mailbox accessing server you chose to deploy.

The most prevalent SMTP servers or MTAs in the UNIX-like world are Postfix, Exim, and Sendmail. Sendmail has the longest history and many systems administrators have extensive experience with it. Postfix is robust and modern, and is compatible with many different deployment types. Exim is the default MTA in Debian systems, and many consider it to be easier to use for basic tasks. For remote mailbox access, servers like Courier and Dovecot are widely deployed to provide remote access to mailboxes.

If you are in need of an integrated and easy to install email solution we encourage you to consider the Citadel groupware server. Citadel provides an integrated "turnkey" solution that includes an SMTP server, remote mailbox access, real time collaboration tools including XMPP, and a shared calendar interface. Along similar lines, we also provide documentation for the installation of the Zimbra groupware server.

If, by contrast, you want a more simple and modular email stack, we urge you to consider one of our guides built around the Postfix SMTP server.

Finally, it's possible to outsource email service to a third party provider, such as Google Apps or FastMail.fm. This allows you to send and receive mail from your domain, without hosting email services on your Linode. Consult our documentation for setting up Google Apps for your domain.

Sending Email From Your Server

In many cases, administrators have no need for a complete email stack like those documented in our email guides. However, applications running on that server still need to be able to send mail for notifications and other routine purposes.

The configuration of applications to send notifications and alerts is beyond the scope of this tip, most applications rely on a simple "sendmail" interface. Nevertheless, the modern MTAs Postfix provides a sendmail-compatible interfaces located at `/usr/sbin/sendmail`.

You can install postfix on Debian and Ubuntu systems with the following command:

```
1 apt-get install postfix
```

On CentOS and Fedora systems you can install postfix by issuing the following command:

```
1 yum install postfix
```

Once Postfix is installed, your applications should be able to access the sendmail interface, located at `/usr/sbin/sendmail`. Most applications running on your Linode should be able to send mail normally with this configuration.

If you simply want to use your server to send email through an external SMTP server, you may want to consider a more simple tool like `msmtp`. Since `msmtp` is packaged in most distributions you can install using the command appropriate to your distribution:

```
1 apt-get install msmtp
2 yum install msmtp
3 pacman -S msmtp
```

Use the command `type msmtp`, to find the location of `msmtp` on your system. Typically the program is located at `/usr/bin/msmtp`. You can specify authentication credentials with command line arguments or by declaring SMTP credentials in a configuration file. Here is an example `.msmtprc` file.

.msmtprc example

account default host smtp.example.com from squire@example.com auth on user squire password s3cr37 tls on tls_certcheck off port 587

The `.msmptrc` file needs to be set to mode 600, and owned by the user account that will be sending mail. If the configuration file is located at `/srv/smtp/msmtprc`, you can call mstmp with the following command:

```
1 /usr/bin/msmtp --file=/srv/smtp/msmtprc
```

VIRTUALIZATION

Virtualization projects are the focus of many IT professionals who are trying to consolidate servers or data centers, decrease costs and launch successful "green" conservation initiatives. Virtualizing IT resources can be thought of as squeezing an enterprise's computer processing power, memory, network bandwidth and storage capacity onto the smallest number of hardware platforms possible and then apportioning those resources to operating systems and applications on a time-sharing basis. This approach aims to make the most efficient possible use of IT resources. It differs from historical computing and networking models, which have typically involved inextricably binding a given software

application or service to a specific operating system (OS), which, in turn, has been developed to run on a particular hardware platform. By contrast, virtualization decouples these components, making them available from a common resource pool. In this respect, virtualization prevents IT departments from having to worry about the particular
hardware or software platforms installed as they deploy additional services. The decoupling and optimization of these components is possible whether you are virtualizing servers, desktops, applications, storage devices or networks. To virtualize some or all of a computing infrastructure's resources, IT departments require special virtualization software, firmware or a third- party service that makes use of virtualization software or firmware. This software/firmware component, called the hypervisor or the virtualizattion layer, performs the mapping between virtual and physical resouces. It is what enables the various resources to be decoupled,then aggregated and dispensed, irrespective of the underlying hardware and, in some cases, the software OS. In effect, the hypervisor takes over hardware management from the OS. In addition to the hypervisor virtualization technology, the organization overseeing the virtualization project requires a virtualization management tool – which might be procured from the same or a different supplier – to set up and manage virtual devices and policies.
Why Virtualize?
One key reason why IT organizations are considering virtualization of some or all of their computing infrastructures is that the technology helps them to derive the biggest bang out of their computing buck.

SETTING UP XEN

Xen is a type 1, bare-metal virtual machine monitor (or hypervisor), which provides the ability to run one or more operating system instances on the same physical machine. Xen, like other types of virtualization, is useful for many use cases such as server consolidation and isolation of production and development environments (Eg.: corporate and personal environments on the same system).

As of Ubuntu 11.10 (Oneiric), the default kernel included in Ubuntu can be used directly with the Xen hypervisor as the management (or control) domain (dom0 or "Domain0" in Xen terminology).

Our example uses LVM for virtual disks and network bridging for virtual network cards. It also assumes Xen 4.1 (the version available in 12.04). It assumes a familiarity with general virtualization issues, as well as with the specific Xen terminology. Please see the Xen wiki (see http://wiki.xen.org/wiki/Xen_Overview) for more information.

During Installation of Ubuntu

- `Note: For VMware Workstation test users:`
- `On VM settings enable "Virtualize Intel VT-x/EPT or AMD-V/RVI"`

During the install of Ubuntu for the Partitioning method choose "Guided - use the entire disk and setup LVM". Then, when prompted to enter "Amount of volume group to use for guided partitioning" enter a value large enough for the Xen dom0 system, leaving the rest for virtual disks. Enter a value smaller than the size of your installation drive. For example 100 GB should be large enough for a minimal Xen dom0 system. Keep in mind that in our model stay inside that guest (dom0) all installation media for guest OSs and other useful files, so that guest must have enough space on it.

After Installation of Ubuntu

Install GUI

```
sudo apt-get update
sudo apt-get install ubuntu-desktop
```

To skip the login screen completely, boot into the console and then start the GUI

```
sudo gedit /etc/default/grub
```

Change line GRUB_CMDLINE_LINUX_DEFAULT="quiet splash" to GRUB_CMDLINE_LINUX_DEFAULT="text".

Save and exit.

```
sudo update-grub
```

Reboot and you should come up directly in tty1.

Login, and then "startx" to boot into the default desktop.

To logoff "Unity" (default Ubuntu desktop) from command line type "gnome-session-quit".

Install Windows Remote Desktop

We understand that this is the best approach to have remote access to "Ubuntu Server 12.04 LTS" into a cross-plataform environment. The "XRDP" is an implementation of the "Remote Desktop" standards from Microsoft and works on the same way as for Windows. Allows remote desktop access via native Windows client machines (or "RDESKTOP" on Ubuntu), does not require loading the "Ubuntu Server 12.04 LTS" GUI (Graphical User Interface) on boot and allows multiple simultaneous sessions. To use "RDESKTOP" on Ubuntu 12.04 LTS with "TSCLIENT" see my post on http://superuser.com/questions/420291/ubuntu-12-04-how-to-get-tsclient-back.

With "XRDP" you can easily use Microsoft RDP to connect to Ubuntu without any configuration. All you need to do is install the "xrdp" package, then open Remote Desktop Connection from Windows and connect. That's it, nothing to configure.

Without wasting anymore of your time, let’s get going.

- `sudo apt-get install xrdp`

Next, open Windows Remote Desktop Connection (RDP) and type Ubuntu Server hostname or IP address.

Off Course You Can Use the Great SSH

As you may already know, SSH is a secure communication protocol that lets you remotely access networked computers. It is known as a replacement for Telnet which is very unsecure. While Telnet sends traffic in plain text, SSH on the other hand uses a secure protocol to communicate.

Run the commands below to install SSH Server.

- `sudo apt-get install openssh-server`

To log in on remote machine type on your terminal

- `ssh <remote_user>@<ip_or_name>`

Installing Xen (XCP - Xen Cloud Platform)

XCP (Xen Cloud Platform) is the open source version similar to Citrix Xen Server that uses the Xen Hypervisor. XCP uses XAPI or XenAPI to manage Xen hosts. XCP is based on CentOS 5.5.

Project Kronos is an initiative to port the XAPI tool stack to Debian and Ubuntu. It is a management stack implemented in OCaml that configures and controls Xen hosts, attached storage, networking and virtual machine life cycle. It exposes a HTTP API and provides a command line interface (xe) for resource management.

XenCenter is Windows desktop application by Citrix that is distributed with XenServer for managing servers running XenServer (the equivalent of linux is OpenXenManager). It uses XAPI for talking to Xen resource pools. Since we are setting up XAPI, we can use XenCenter to manage the server.

Why use XCP-XAPI on Debian/Ubuntu when XCP appliance exists?

- Manage dom0 using a configuration management framework (Puppet, Chef...)
- Apply security updates to dom0 root file system
- Run Xen version 4.1.
- Ubuntu Server 12.04 is a LTS release that is supported for 5 years

Installing and configuring Xen Hypervisor

- Install the Xen Hypervisor
  - `sudo apt-get install xen-hypervisor`

  Setup GRUB to boot the Xen Hypervisor

  - `sudo sed -i 's/GRUB_DEFAULT=.*\+/GRUB_DEFAULT="Xen 4.1-amd64"/' /etc/default/grub`

  Disable apparmor at boot

  - `sudo sed -i 's/GRUB_CMDLINE_LINUX=.*\+/GRUB_CMDLINE_LINUX="apparmor=0"/' /etc/default/grub`

  Restrict "dom0" to 1GB of memory and 1 VCPU (example)

  sudo gedit /etc/default/grub

  After GRUB_CMDLINE_LINUX="apparmor=0" add the line
  GRUB_CMDLINE_XEN="dom0_mem=1G,max:1G dom0_max_vcpus=1"

Update Grub with the config changes we just made

- `sudo update-grub`

Reboot the server so that Xen boots on the server

- `sudo reboot`

Once the server is back online ensure that Xen is running

- `cat /proc/xen/capabilities should display "control_d"`
- Note: To stop or start xcp-xapi
  `sudo /etc/init.d/xcp-xapi stop (or start)`

Installing and configuring XAPI (XenAPI)

- Install XCP-XAPI
  - `sudo apt-get install xcp-xapi`

    -> Choose "bridge" when prompted for network backend

- Setup the default toolstack
  - `sudo gedit /etc/default/xen`

    -> Set "TOOLSTACK=xapi"

- Disable xend from starting at boot
  - `sudo sed -i -e 's/xend_start$/#xend_start/' -e 's/xend_stop$/#xend_stop/' /etc/init.d/xend`

    Note: Only xend the deamon needs to be disabled from starting, "/etc/init.d/xend" handles other things like modules and xenfs. Do not disable it from the runlevel. Disable service xendomains

  - `sudo update-rc.d xendomains disable`

    Fix for "qemu" which emulates the console does not have the keymaps in the correct location

  - `sudo mkdir /usr/share/qemu; sudo ln -s /usr/share/qemu-linaro/keymaps /usr/share/qemu/keymaps`

    Network configuration

  - This section describes how to set up Linux bridging in Xen. It assumes eth0 is both your primary interface to dom0 and the interface you want your VMs to use. It also assumes that you will use manually IP configuration.
    - `sudo apt-get install bridge-utils`

      Note: If not already installed

  - As you are working with a desktop install, disable "Network Manager"
    - `sudo stop network-manager`
      `sudo gedit /etc/NetworkManager/NetworkManager.conf`

- Now make sure the "managed" line contains "managed=false"
  Save and exit

Since Network Manager is not managing your network interfaces anymore, you must manually enter that information. Below we explain how.

- Setup bridge networking
  - `sudo gedit /etc/network/interfaces`
    - Create a bond called xenbr0. The file should look like this for a static network configuration:

```
# This file describes the network interfaces available on your system
# and how to activate them. For more information, see interfaces(5).

# The loopback network interface
auto lo
iface lo inet loopback


# Xen network interface for "dom0"
auto xenbr0
iface xenbr0 inet static

# IP address
address 192.168.1.111
# Subnet mask
netmask 255.255.255.0
# Default Gateway
gateway 192.168.1.1
# DNS Server
dns-nameservers 192.168.1.1

bridge_ports eth0
iface eth0 inet manual


# The primary network interface
# auto eth0
# iface eth0 inet dhcp
```

Configure xcp to use "bridge" networking instead of "openswitch"

- `sudo gedit /etc/xcp/network.conf`

-> Replace “openswitch” with “bridge”

```
Note: For VMware Workstation test users:
        - Configure with "bridge" for network adapter
and run these comands on host:

                sudo chmod ugo+rwx /dev/vmnet0
                sudo chown <username> /dev/vmnet0
                sudo chown :<usergroup> /dev/vmnet0

                Eg.:
                        sudo chmod ugo+rwx
/dev/vmnet0
```

```
                                        sudo chown eduardo
/dev/vmnet0
                                        sudo chown :eduardo
/dev/vmnet0
```

To test whether the network is working run the command

- `sudo /etc/init.d/networking restart`

All set! Ready to reboot and let xcp-xapi toolstack take over

- `sudo reboot`

On restart confirm that xcp is working

- `sudo xe vm-list`
  - This should list the control domain
    - "
    - `uuid (RO) : dbcf74d2-ee50-edd5-d44d-b81fc8ba1777`
    - `name-label (RW): Control domain on host: ubuntu-xenserver-1`
    - `power-state (RO): running`
    - "

    -> If your output looks similar, "xapi" is running on the server, if you get “Connection refused” then xapi is not setup correctly!

Configure Storage Repository for Use With XAPI

- NFS servers are a common form of shared filesystem infrastructure, and can be used as a storage repository substrate for virtual disks. As NFS storage repositories are shared, the virtual disks stored in them allow VMs to be started on any server in a resource pool and to be migrated between them using XenMotion. When you configure an NFS storage repository, you simply provide the hostname or IP address of the NFS server and the path to a directory that will be used to contain the storage repository (if this resource is on another machine, it is not our case). The NFS server must be configured to export the specified path to all servers in the pool.
  - To show your volume group (VG)
    - `sudo pvs`

Create a LV with X GBs

- `sudo lvcreate -L <X>GB -n <StorageRepositoryName> /dev/<VG>`
- `Eg1.: sudo lvcreate -L 25GB -n StorageRepository /dev/ubuntus1204`
  `Eg2.: sudo lvcreate -l 100%FREE -n StorageRepository /dev/ubuntus1204`

Register the logical volume for use with XAPI

- `sudo xe sr-create type=ext shared=true name-label=<StorageRepositoryName> device-config:device=/dev/<VG>/<StorageRepositoryName>`

```
Eg.: sudo xe sr-create type=ext shared=true
name-label=StorageRepository device-
config:device=/dev/ubuntus1204/StorageRepositor
y
```

This should display the Storage Repository

- sudo xe sr-list name-label=<StorageRepositoryName>
  Eg.: sudo xe sr-list name-label=StorageRepository
  - "
  - uuid (RO): 37bc5263-c9fc-8876-d24c-d5927f1bbed2
  - name-label (RW): StorageRepository
  - name-description (RW):
  - host (RO): ubuntus1204
  - type (RO): ext
  - content-type (RO):
  - "

Configure a ISO Repository for Use With XAPI

An ISO Repository contains ISOs (disk images) with operational systems to perform the installations.

Then the following example makes a storage repository called ISOs

```
sudo xe sr-create name-label=<LocalISORepositoryName> type=iso shared=true
device-config:location=<FolderPath> device-config:legacy_mode=true content-
type=iso
Eg.:
sudo mkdir -p /var/opt/xen/LocalISORepository/
sudo xe sr-create name-label=LocalISORepository type=iso shared=true device-
config:location=/var/opt/xen/LocalISORepository/ device-
config:legacy_mode=true content-type=iso
```

This should display the ISO Repository

```
sudo xe sr-list name-label=<LocalISORepositoryName>
Eg.: sudo xe sr-list name-label=LocalISORepository
"
uuid ( RO): 26edb27b-72fc-af56-ad2f-4d15a8d8e3f7
name-label ( RW): LocalISORepository
name-description ( RW):
host ( RO): ubuntus1204
type ( RO): iso
content-type ( RO): iso
"
```

VMWARE ON LINUX

VMware Workstation 5.5 Host System Requirements

What do you need to get the most out of VMware Workstation 5? Take the following list of requirements as a starting point. Like physical computers, the virtual machines running under VMware Workstation generally perform better if they have faster processors and more memory.

PC Hardware

- Standard x86-compatible or x86-64-compatible personal computer
- 400 MHz or faster CPU minimum (500 MHz recommended)

Compatible processors include

- Intel®: Celeron®, Pentium® II, Pentium III, Pentium 4, Pentium M (including computers with Centrino™ mobile technology), Xeon™ (including "Prestonia"), EM64T
- AMD™: Athlon™, Athlon MP, Athlon XP, Athlon 64, Duron™, Opteron™, Turion™ 64
- Experimental support for AMD Sempron™

For additional information, including notes on processors that are not compatible, see the VMware knowledge base at www.vmware.com/support/kb/enduser/std_adp.php?p_faqid=967.

- Multiprocessor systems supported
- AMD Opteron, AMD Athlon 64, AMD Turion 64, AMD Sempron, Intel EM64T; support for 64-bit guest operating systems is available only on the following specific versions of these processors:
  - AMD Athlon 64, revision D or later
  - AMD Opteron, revision E or later
  - AMD Turion 64, revision E or later
  - AMD Sempron, 64-bit-capable revision D or later (experimental support)
  - Intel EM64T VT-capable processors (experimental support)

Memory

- 128 MB minimum (256 MB recommended)

You must have enough memory to run the host operating system, plus the memory required for each guest operating system and for applications on the host and guest. See your guest operating system and application documentation for their memory requirements.

Display

- 16-bit or 32-bit display adapter recommended

Disk Drives

Guest operating systems can reside on physical disk partitions or in virtual disk files.

Hard Disk

- IDE and SCSI hard drives supported, up to 950GB capacity
- At least 1GB free disk space recommended for each guest operating system and the application software used with it; if you use a default setup, the actual disk space needs are approximately the same as those for installing and running the guest operating system and applications on a physical computer.
- **For Installation** — 80MB (Linux) or 250MB (Windows) free disk space required for basic installation. You can delete the installer afterwards to reclaim disk space.

Optical CD-ROM/DVD-ROM Drive

- IDE and SCSI optical drives supported
- CD-ROM and DVD-ROM drives supported
- ISO disk image files supported

Local Area Networking (Optional)

- Any Ethernet controller supported by the host operating system
- Non-Ethernet networks supported using built-in network address translation (NAT) or using a combination of host-only networking plus routing software on the host operating system

Host Operating System

VMware Workstation is available for both Windows and Linux host operating systems.

Windows Host Operating Systems (32-Bit)

Workstation supports the following Windows 32-bit host operating systems.

- Windows Server 2003 Standard Edition, SP1
Windows Server 2003 Web Edition, SP1
Windows Server 2003 Small Business Edition, SP1
Windows Server 2003 Enterprise Edition, SP1
Windows Server 2003 R2
(Listed versions are also supported with no service pack.)
- Windows XP Home Edition, SP1, SP2

Windows XP Professional, SP1, SP2
(Listed versions are also supported with no service pack.)

- Windows 2000 Server SP3, SP4

Windows 2000 Professional, SP3, SP4
Windows 2000 Advanced Server, SP3, SP4

Windows Host Operating Systems (64-Bit)

- Windows Server 2003 x64 Edition SP1

Windows Server 2003 x64 Edition R2

- Windows XP Professional x64 Edition

Internet Explorer 4.0 or higher is required for the Windows online help system.

Linux Host Operating Systems (32-Bit)

Supported distributions and kernels are listed below. VMware Workstation may not run on systems that do not meet these requirements.

Note: As newer Linux kernels and distributions are released, VMware modifies and tests its products for stability and reliability on those host platforms. We make every effort to add support for new kernels and distributions in a timely manner, but until a kernel or distribution is added to the list below, its use with our products is not supported. Look for newer prebuilt modules in the download area of our Web site. Go to www.vmware.com/download/.

- Mandriva Corporate Server 4

Mandriva Linux 2007 (experimental support)
Mandriva Linux 2006
Mandrake Linux 10.1
Mandrake Linux 9.0 — stock 2.4.19

- Red Hat Enterprise Linux 5.0, AS, ES, WS (experimental support)

Red Hat Enterprise Linux 4.0, AS, ES, WS, updates 1, 2, 3, 4
Red Hat Enterprise Linux 3.0, AS, ES, WS, updates 1, 2, 3, 4, 5, 6, 7, 8
Red Hat Enterprise Linux 2.1 — stock 2.4.9-e3

Red Hat Linux 9.0 — stock 2.4.20-8, upgrade 2.4.20-20.9
Red Hat Linux 8.0 — stock 2.4.18
Red Hat Linux 7.3 — stock 2.4.18
Red Hat Linux 7.2 — stock 2.4.7-10, upgrade 2.4.9-7, upgrade 2.4.9-13, upgrade 2.4.9-21, upgrade 2.4.9-31
Red Hat Linux 7.1 — stock 2.4.2-2, upgrade 2.4.3-12
Red Hat Linux 7.0 — stock 2.2.16-22, upgrade 2.2.17-14

- SUSE Linux Enterprise Server 10

SUSE Linux Enterprise Server 9, 9 SP1, 9 SP2, 9 SP3 — stock 2.6.5-797
(Listed versions are also supported with no service pack.)
SUSE Linux Enterprise Server 8, stock 2.4.19

SUSE Linux 10.1
SUSE Linux 10 — stock 2.6.13
SUSE Linux 9.3 — stock 9.3-2.6.11.4
SUSE Linux 9.2, SP1 — stock 9.2-2.6.8-24.11 (Listed versions are also supported with no service pack.)
SUSE Linux 9.1 — stock 2.6.4-52
SUSE Linux 9.0 — stock 2.4.21-99
SUSE Linux 8.2 — stock 2.4.20

- Ubuntu Linux 6.10(experimental support)

Ubuntu Linux 6.06
Ubuntu Linux 5.10
Ubuntu Linux 5.04

Platforms not listed above are not supported. A Web browser is required for the Help system.

Linux Host Operating Systems (64-Bit)

Supported distributions and kernels are listed below. VMware Workstation may not run on systems that do not meet these requirements.

Note: As newer Linux kernels and distributions are released, VMware modifies and tests its products for stability and reliability on those host platforms. We make every effort to add support for new kernels and distributions in a timely manner, but until a kernel or distribution is added to the list below, its use with our products is not supported. Look for newer prebuilt modules in the download area of our Web site. Go to www.vmware.com/download/.

- Mandriva Corporate Server 4

Mandriva Linux 2007 (experimental support)
Mandriva Linux 2006

- Red Hat Enterprise Linux 5.0, AS, ES, WS (experimental support)

Red Hat Enterprise Linux 4.0, AS, ES, WS, updates 1, 2, 3, 4
Red Hat Enterprise Linux 3.0, AS, ES, WS stock 2.4.21, updates 2.4.21-15, 6, 7, 8

SUSE Linux Enterprise Server 10
SUSE Linux Enterprise Server 9, SP1, SP2, SP3 — stock 2.6.5-797
(Listed versions are also supported with no service pack.)

SUSE Linux 10.1
SUSE Linux 10 — stock 2.6.13
SUSE Linux 9.3 — stock 9.3-2.6.11.4
SUSE Linux 9.2, SP1 — stock 9.2-2.6.8-24.11 (Listed versions are also supported with no service pack.)
SUSE Linux 9.1 — stock 2.6.4-52

- Ubuntu Linux 6.10 (experimental support)

Ubuntu Linux 6.06

Ubuntu Linux 5.10
Ubuntu Linux 5.04

Platforms not listed above are not supported. A Web browser is required for the Help system.