

Announcements

- Final is Friday at 4:00 in **CSI 1115**
- Course evaluations are on the web

What is an Operating System?

- **Resource Manager**

- Resources include: CPU, memory, disk, network
- OS allocates and de-allocates these resources

- **Virtual Machine**

- provides an abstraction of a larger (or just different machine)
- Examples:
 - Virtual memory - looks like more memory
 - Java - pseudo machine that looks like a stack machine
 - IBM VM - a complete virtual machine (can boot multiple copies of an OS on it)

- **Multiplexor**

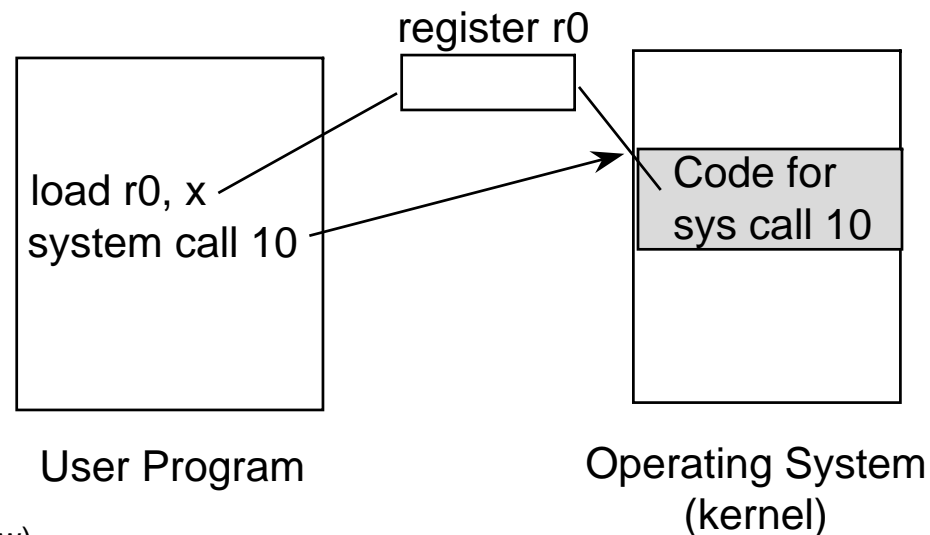
- allows sharing of resources and protection
- motivation is cost: consider a \$40M supercomputer

What is an OS (cont)?

- **Provider of Services**
 - includes most of the things in the above definition
 - provide “common” subroutines for the programmer
 - windowing systems
 - memory management
- **The software that is always loaded/running**
 - generally refers to the *Os kernel*.
 - small protected piece of software
- **All of these definitions are correct**
 - **but** not all operating have all of these features

System Calls

- Provide the interface between application programs and the kernel
- Are like procedure calls
 - take parameters
 - calling routine waits for response
- Permit application programs to access protected resources



System Call Mechanism

- Use numbers to indicate what call is made
- Parameters are passed in registers or on the stack
- Why do we use indirection of system call numbers rather than directly calling a kernel subroutine?
 - provides protection since the only routines available are those that are export
 - permits changing the size and location of system call implementations without having to re-link application programs

Policy vs. Mechanism

- Policy - what to do
 - users should not be able to read other users files
- Mechanism- how to accomplish the goal
 - file protection properties are checked on open system call
- Want to be able to change policy without having to change mechanism
 - change default file protection
- Extreme examples of each:
 - micro-kernel OS - all mechanism, no policy
 - MACOS - policy and mechanism are bound together

Processes

- What is a process?
 - a program in execution
 - “An execution stream in the context of a particular state”
 - a piece of code along with all the things the code can affect or be affected by.
 - this is a bit too general. It includes all files and transitively all other processes
 - only one thing happens at a time within a process
- What's not a process?
 - program on a disk - a process is an active object, but a program is just a file

Process Creation

- Who creates processes?
 - answer: other processes
 - operations is called fork (or spawn)
 - what about the first process?
- Have a tree of processes
 - parent-child relationship between processes
- what resources does the child get?
 - new resources from the OS
 - a copy of the parent resources
 - a subset of the parent resources
- What program does the child run?
 - a copy of the parent (UNIX fork)
 - a process may change its program (execve call in UNIX)
 - a new program specified at creation (VMS spawn)

Critical Section Problem

- processes must
 - request permission to enter the region
 - notify when leaving the region
- protocol needs to
 - provide mutual exclusion
 - only one process at a time in the critical section
 - ensure progress
 - no process outside a CS may block another process
 - guarantee bounded waiting time
 - limited number of times other processes can enter the critical section while another process is waiting
 - not depend on number or speed of CPUs
 - or other hardware resources
- May assume that some instructions are atomic
 - typically load, store, and test word instructions

Deadlocks

- **System contains finite set of resources**
 - Process requests resource before using it, must release resource after use
 - Process is in a deadlock state when every process in the set is waiting for an event that can be caused only by another process in the set
- **4 *necessary* deadlock conditions:**
 - Mutual exclusion - at least one resource must be held in a non-sharable mode
 - Hold and wait
 - No preemption
 - Circular wait

Deadlock Prevention

- Ensure that one conditions for deadlock never holds
- Hold and wait
 - guarantee that when a process requests a resource, it does not hold any other resources
 - Each process could be allocated all needed resources before beginning execution
- Mutual exclusion
 - Sharable resources
- Circular wait
 - make sure that each process claims all resources in increasing order of resource type enumeration
- No Preemption
 - virtualize resources and permit them to be preempted. For example, CPU can be preempted.

Banker's Algorithm

- Each process must declare the maximum number of instances of each resource type it may need
- Maximum cannot exceed resources available to system
- Variables: (n is the number of processes, m is the number of resource types)
 - Available - vector of length m indicating the number of available resources of each type
 - Max - n by m matrix defining the maximum demand of each process
 - Allocation - n by m matrix defining number of resources of each type currently allocated to each process
 - Need: n by m matrix indicating remaining resource needs of each process

Short-term scheduling algorithms

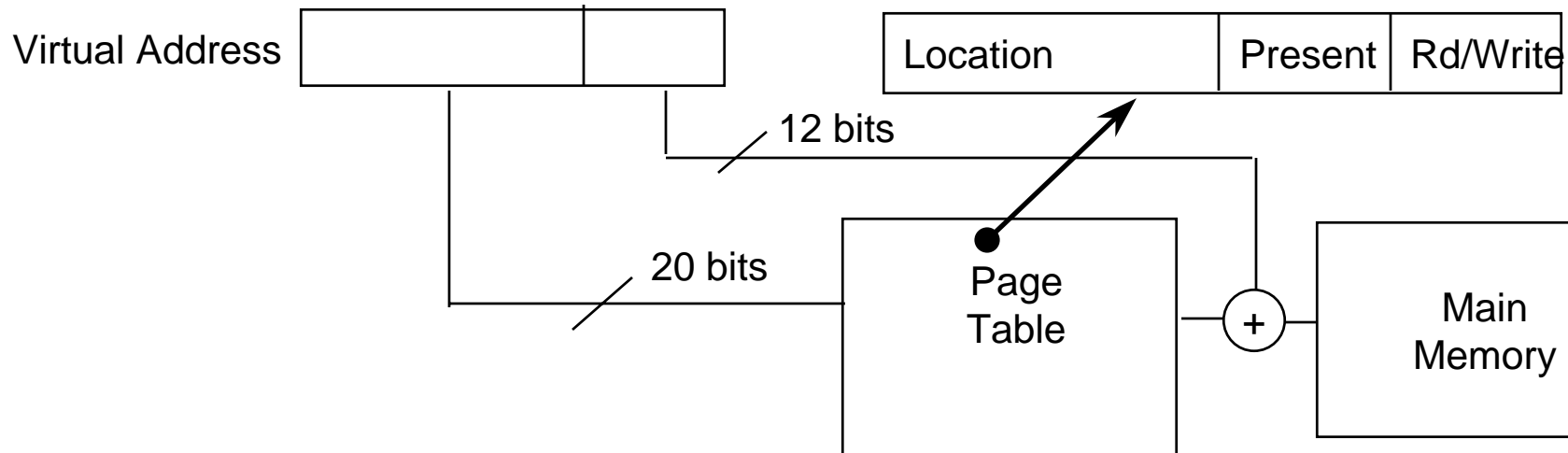
- **First-Come, First-Served (FCFS, or FIFO)**
 - as process becomes ready, join Ready queue, scheduler always selects process that has been in queue longest
- **Round-Robin (RR)**
 - use preemption, based on clock - time slicing
- **Shortest Process Next (SPN)**
 - non-preemptive
 - select process with shortest expected processing time
- **Shortest Remaining Time (SRT)**
 - preemptive version of SPN
 - scheduler chooses process with shortest expected remaining process time
- **Priorities**
 - assign each process a priority, and scheduler always chooses process of higher priority over one of lower priority

Managing Memory

- Main memory is big, but what if we run out
 - use virtual memory
 - keep part of memory on disk
 - bigger than main memory
 - slower than main memory
- Want to have several program in memory at once
 - keeps processor busy while one process waits for I/O
 - need to protect processes from each other
 - have several tasks running at once
 - compiler, editor, debugger
 - word processing, spreadsheet, drawing program
- Use *virtual addresses*
 - look like normal addresses
 - hardware translates them to *physical addresses*

Paging

- Divide physical memory into fixed sized chunks called *pages*
 - typical pages are 512 bytes to 64k bytes
 - When a process is to be executed, load the pages that *are actually used* into memory
- Have a table to map virtual pages to physical pages
- Consider a 32 bit addresses
 - 4096 byte pages (12 bits for the page)
 - 20 bits for the page number



Inverted Page Tables

- Solution to the page table size problem
- One entry per page frame of physical memory

<process-id, page-number>

- each entry lists process associated with the page and the page number
- when a memory reference:
 - <**process-id,page-number,offset**> occurs, the inverted page table is searched (usually with the help of a hashing mechanism)
 - if a match is found in entry *i* in the inverted page table, the physical address <**i,offset**> is generated
- The inverted page table does not store information about pages that are not in memory
 - page tables are used to maintain this information
 - page table need only be consulted when a page is brought in from disk

What Happens when a virtual address has no physical address?

- called a *page fault*
 - a trap into the operating system from the hardware
- caused by: the first use of a page
 - called *demand paging*
 - the operating system allocates a physical page and the process continues
 - read code from disk or init data page to zero
- caused by: a reference to an address that is not valid
 - program is terminated with a “segmentation violation”
- caused by: a page that is currently on disk
 - read page from disk and load it into a physical page, and continue the program
- caused by: a copy on write page

Page State (hardware view)

- Page frame number (location in memory or on disk)
- *Valid Bit*
 - indicates if a page is present in memory or stored on disk
- *A modify or dirty bit*
 - set by hardware on write to a page
 - indicates whether the contents of a page have been modified since the page was last loaded into main memory
 - if a page has not been modified, the page does not have to be written to disk before the page frame can be reused
- *Reference bit*
 - set by the hardware on read/write
 - cleared by OS
 - can be used to approximate LRU page replacement
- **Protection attributes**
 - read, write, execute

Page Replacement Algorithms

- **FIFO**

- Replace the page that was brought in longest ago
- However
 - old pages may be great pages (frequently used)
 - number of page faults may increase when one increases number of page frames (discouraging!)
 - called belady's anomaly
 - 1,2,3,4,1,2,5,1,2,3,4,5 (consider 3 vs. 4 frames)

- **Optimal**

- Replace the page that will be used furthest in the future
- Good algorithm(!) but requires knowledge of the future
- With good compiler assistance, knowledge of the future is sometimes possible

Page Replacement Algorithms

- LRU

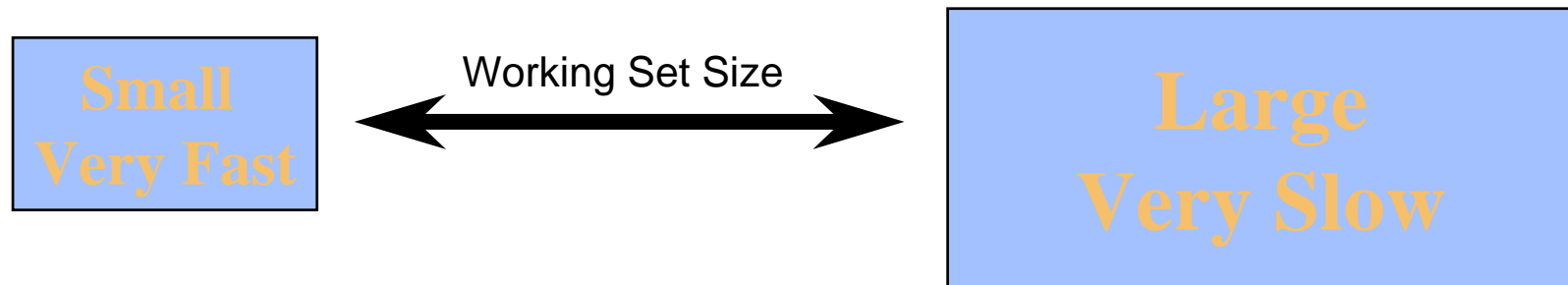
- Replace the page that was actually used longest ago
- Implementation of LRU can be a bit expensive
 - e.g. maintain a stack of nodes representing pages and put page on top of stack when the page is accessed
 - maintain a time stamp associated with each page

- Approximate LRU algorithms

- maintain reference bit(s) which are set whenever a page is used
- at the end of a given time period, reference bits are cleared

Working Sets and Page Replacement

- Programs usually display reference locality
 - temporal locality
 - repeated access to the same memory location
 - spatial locality
 - consecutive memory locations access nearby memory locations
 - memory hierarchy design relies heavily on locality reference
 - sequence of nested storage media
- Working set
 - set of pages referenced in the last delta references

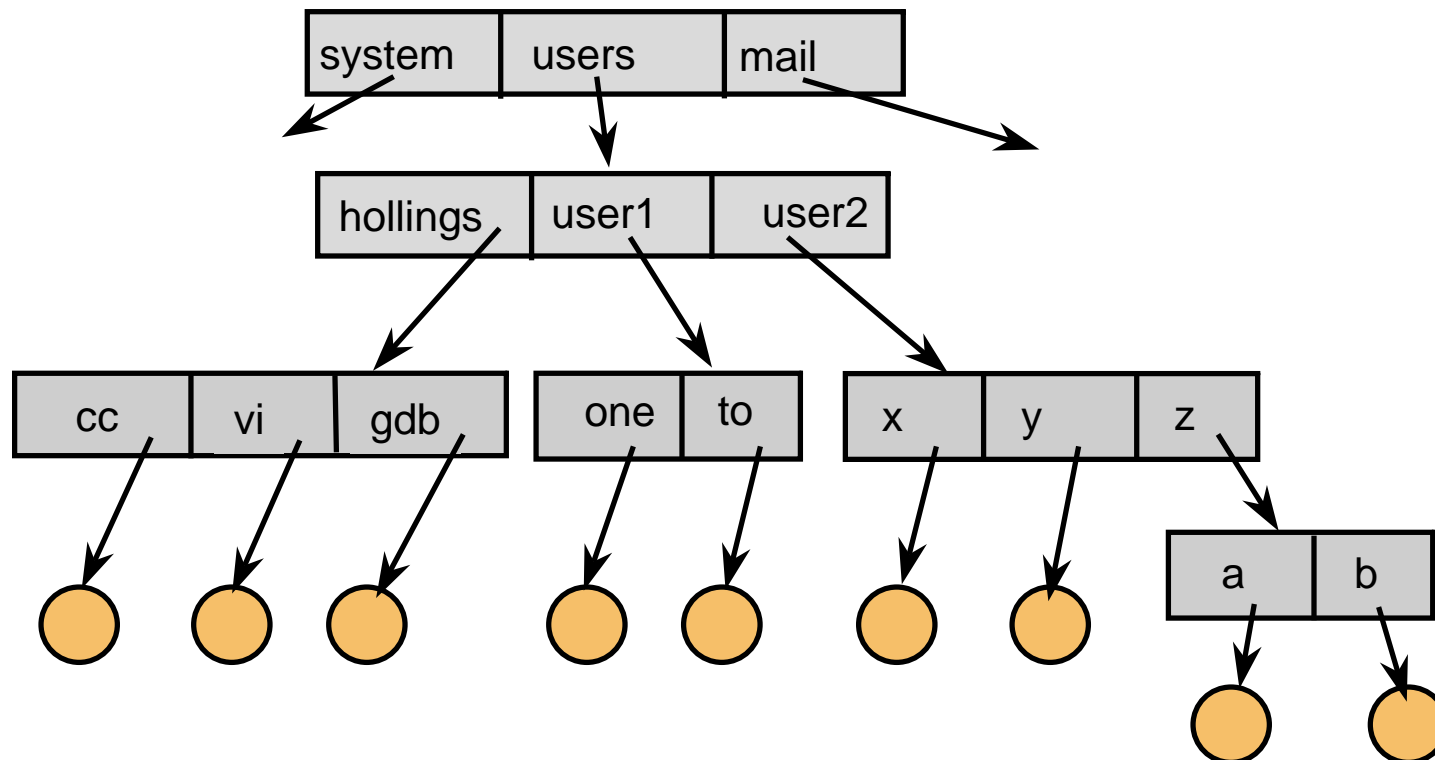


File Abstraction

- What is a file?
 - A named collection of information stored on secondary storage
- Properties of a file
 - non-volatile
 - can read, read, or update it
 - has meta-data to describe attributes of the file
- File Attributes
 - name: a way to describe the file
 - type: some information about what is stored in the file
 - location: how to find the file on disk
 - size: number of bytes
 - protection: access control
 - may be different for read, write, execute, append, etc.
 - time: access, modification, creation
 - version: how many times has the file changed

Tree Directories

- create a tree of files
- each directory can contain files or directory entries
- each process has a current directory
 - can name files relative to that directory
 - can change directories as needed



File Protection

- How to give access to some users and not others?
- Access types:
 - read, write, execute, append, delete, list
 - rename: often based on protection of directory
 - copy: usually the same as read
- Degree of control
 - access lists
 - list for each user for each file the permitted operations
 - groups
 - enumerate users in a list called a group
 - provide same protection to all members of the group
 - depending on system:
 - files may be in one or many groups
 - users may be in one or many groups
 - per file passwords (tedious and a security problem)

Filesystems

- **Raw Disks can be viewed as:**
 - a linear array of fixed sized units of allocation, called blocks
 - assume that blocks are error free (for now)
 - typical block size is 512 to 4096 bytes
 - can update a block in place, but must write the entire block
 - can access any block in any desired order
 - blocks must be read as a unit
 - for performance reasons may care about “near” vs. “far” blocks (but that is covered in a future lecture)
- **A Filesystem:**
 - provides a hierarchical namespace via directories
 - permits files of variable size to be stored
 - provides disk protection by restricting access to files based on permissions

Allocation Methods

- How do we select a free disk block to use?
- Contiguous allocation
 - allocate a contiguous chunk of space to a file
 - directory entry indicates the starting block and the length of the file
 - easy to implement, but
 - how to satisfy a given sized request from a list of free holes?
 - two options
 - first fit (find the first gap that fits)
 - best fit (find the smallest gaps that is large enough)
 - What happens if one wants to append to file?
 - from time to time, one will need to repack files

Indexed Allocation

- Bring all pointers together in an *index block*
 - Each file has its own index block - i th entry of index block points to i th block making up the file
- How large to make an index block?
 - unless one only wants to support fixed size files, index block scheme needs to be extensible
- Linked scheme:
 - maintain a linked list of indexed blocks
- Multilevel index:
 - Index block can point to other index blocks (which point to index blocks), which point to files
- Hybrid multi-level index
 - first n blocks are from a fixed index
 - next m blocks from an indirect index
 - next o blocks from a double indirect index

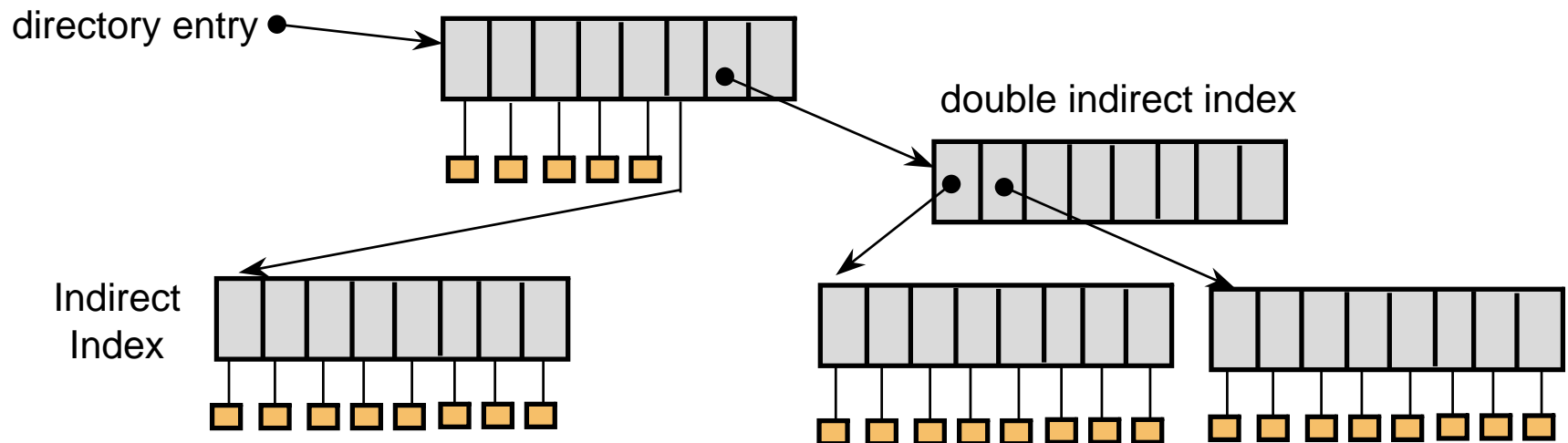
Hybrid Multi-level Index (UNIX)

- Observations

- most files are small
- most of the space on the disk is consumed by large files

- Want a flexible way to support different sized

- assume 4096 byte block
- first 12 blocks (48KB) are from a fixed index
- next 1024 blocks (1MB) from an indirect index
- next 1024^2 blocks (1GB) from a double indirect index
- final 1024^3 blocks (1TB) from a triple indirect index



Disk Cache

- Buffer in main memory for disk sectors
- Cache contains copy of some of the sectors on a disk. When I/O request is made for a sector, a check is made to find out if sector is in the disk cache
- Replacement strategy:
 - Least recently used: block that has been in the cache longest with no reference gets replaced
 - Least frequently used: block that experiences fewest references gets replaced

Disk Scheduling

- **First come, first served**
 - ordering may lead to lots of disk head movement
- **Shortest seek time first: select request with the minimum seek time from current head position**
 - potential problem with distant tracks not getting service for an indefinite period
- **Scan scheduling**
 - read-write head starts at one end of the disk, moves to the other, servicing requests as it reaches each track
- **C-Scan (circular scan)**
 - disk head sweeps in only one direction
 - when the disk head reaches one end, it returns to the other

Who do you trust?

- It's easy to get paranoid
- Do I trust a login prompt?
- Do I trust the OS that I got from the vendor?
- Do I trust the system staff?
 - should I encrypt all my files?
- Networking
 - do you trust the network provider?
 - do you trust the phone company?
- How do you bootstrap security?
 - always need one “out of band” transfer to get going

Authentication

- How does the computer know who is using it?
 - need to exchange some information to verify the user
 - types of information exchanged:
 - pins
 - numeric passwords
 - too short to be secure in most cases
 - passwords
 - a string of letters and numbers
 - often easy to guess
 - challenge/response pairs
 - user needs to be apply to apply a specific algorithm
 - often involve use of a calculator like device
 - can be combined with passwords
 - unique attributes of the person
 - i.e. signature, thumb print, DNA?
 - sometimes these features can change during life

Encryption: protecting info from being read

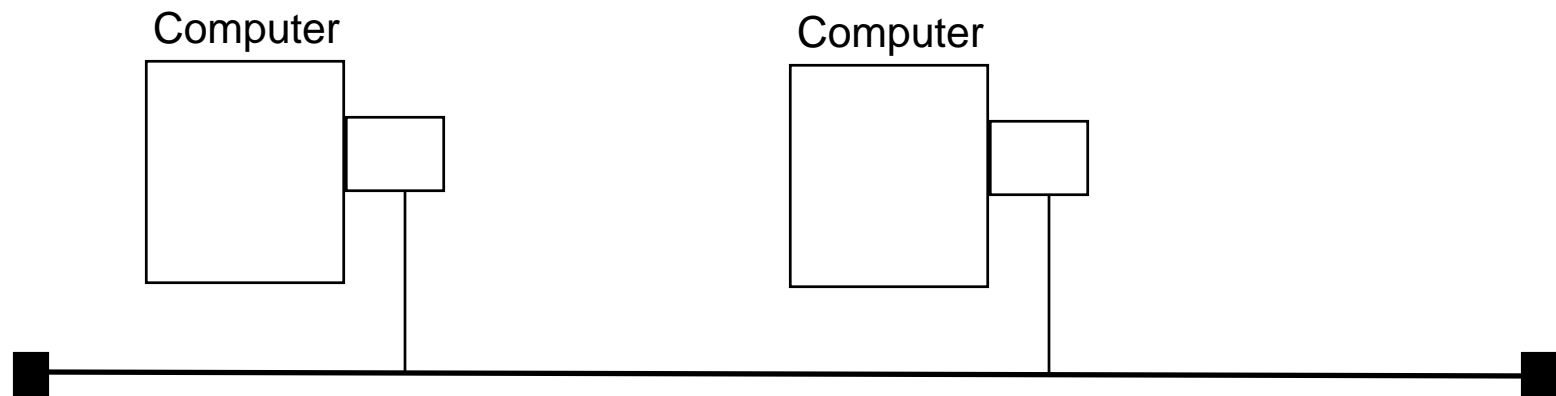
- Given a message m
 - use a key k , and function E_k to compute $E_k(m)$
 - store or send only $E_k(m)$
 - use a second key k' and function $D_{k'}$ such that
 - $D_{k'}(E_k(m)) = m$
 - E_k and $D_{k'}$ need not be kept a secret
- If $k=k'$ it's called private key encryption
 - need to keep k secret
 - example DES
- if $k \neq k'$, it's called public key encryption
 - need only keep one of them secret
 - if k' is secret, anyone can send a private message
 - if k is secret, it is possible to “sign” a message
 - still need a way to authenticate k or k' for a user
 - example RSA

Sending Data

- Data is split into *packets*
 - limited size units of sending information
 - can be
 - fixed sized (ATM)
 - variable size (Ethernet)
- Need to provide a destination for the packet
 - need to identify two levels of information
 - machine to send data to
 - comm abstraction (e.g. process) to get data
 - address may be:
 - a globally unique destination
 - for example every host has a unique id
 - may unique between hops
 - unique id between two switches

Ethernet

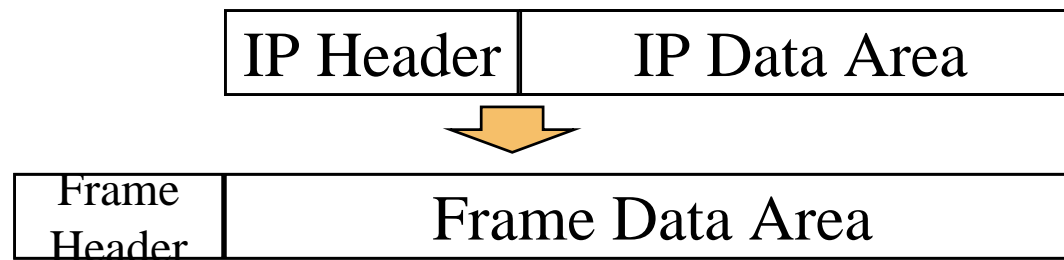
- 10 Mbps (to 100 Mbps)
- mili-second latency
- limited to several kilometers in distance
- variable sized units of transmission
- bus based protocol
 - requests to use the network can collide
- addresses are 48 bits
 - unique to each interface



Encapsulation

How do we send higher layer packets over lower layers?

- Higher level info is opaque to lower layers
 - it's just data to be moved from one point to another



- Higher levels may support larger sizes than lower
 - could need to *fragment* a higher level packet
 - split into several lower level packets
 - need to re-assemble at the end
 - examples:
 - ATM cells are 48 bytes, but IP packets can be 64K
 - IP packets are 64K, but files are megabytes

Routing

- How does a packet find its destination?
 - problem is called routing
- Several options:
 - source routing
 - end points know how to get everywhere
 - each packet is given a list of hops before it is sent
 - hop-by-hop
 - each host knows for each destination how to get one more hop in the right direction
- Can route packets:
 - per session
 - each packet in a connection takes same path
 - per packet
 - packets may take different routes
 - possible to have out of order delivery

Remote Procedure Calls

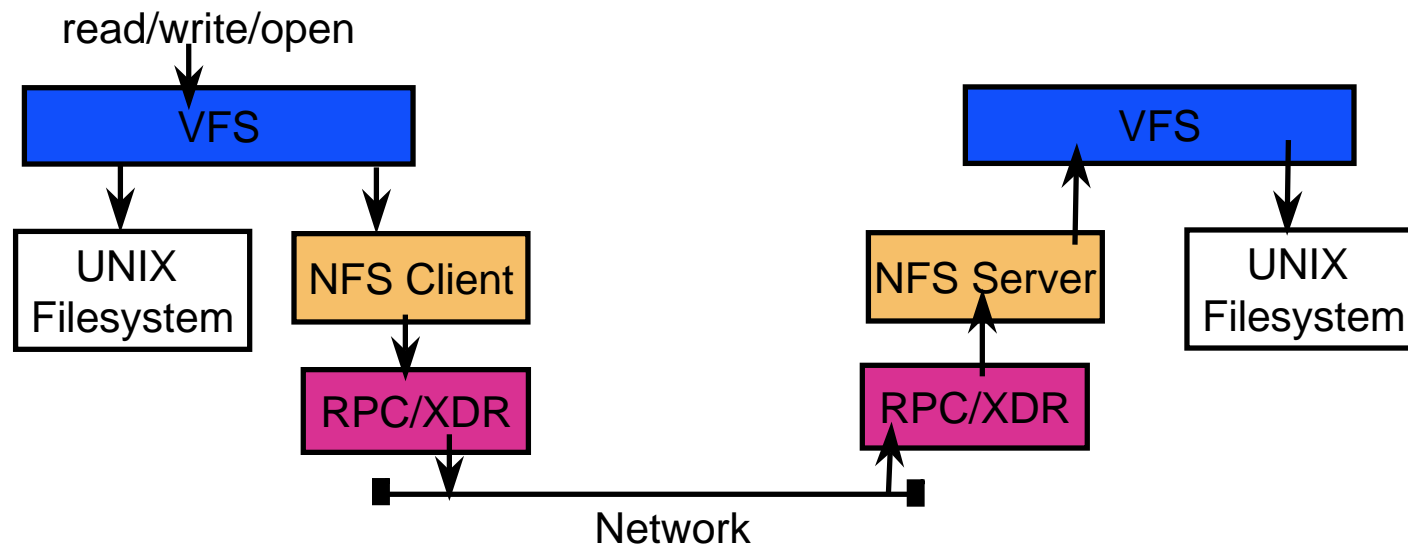
- Provide a way to access remotes services
- Look like “normal” procedure calls
- Issues:
 - binding functions to services
 - can use static binding (like kernel trap #'s)
 - can use a nameserver
 - data format
 - different machine may have different formats
 - translation is called *marshalling*
 - pick a common way to encode info (e.g. XDR)
 - always send in this common format
 - failures
 - what if a host dies while and RPC is active?

Distributed Filesystems

- Provide the same semantics as a local filesystem
 - data is stored at various locations in the system
 - often stored in central file servers
 - can be stored in serverless file servers
- Naming
 - location transparency
 - filenames don't imply information about location
 - location independence
 - can move the file without changing names
 - naming files
 - host:local-name
 - not transparent
 - global-name
 - transparent, requires something to coordinate names

NFS

- Provides a way to mount remote filesystems
 - can be done explicitly
 - can be done automatically (called an automounter)
 - clients are provided “file handle” by the server for future use
- Uses VFS: extended UNIX filesystem
 - inodes are replaced by vnodes
 - network wide unique inodes
 - can refer to local or remote files



AFS

- Designed to scale to 5,000 or more workstations
- Location independent naming
 - within a single cell
- volumes
 - basic unit of management
 - can vary in size
 - can be migrated among servers
- names are mapped to “fids”
 - 96 bit unique id’s for a file
 - three parts: volume, vnode, and uniqidentifier
 - location information is stored in a volume to location DB
 - replicated on every server

AFS (cont.)

- File Access

- open: file is transferred from server to client
 - very large files may only be partially transferred
- read/write: performed on the client
- close: file (if dirty) is written back to server
 - can fail if the disk is full

- Consistency

- clients have callbacks
- sever informs client when another client writes data
- only applies to open operation
- only requires communication when:
 - more than one client wants to write
 - one client wants to write and others to read