**True:**

- I/O completion triggers an interrupt

- the bounded buffer problem can be solved with just semaphores to control concurrency

- a thread requesting a resource never causes a deadlock as long as that thread doesn’t currently have any other resources assigned to it

- an application reading two files located in the same directory is a demonstration of path locality

- it’s possible for a soft link to point to a file that doesn’t exist

- NFS doesn’t solve all cache consistency corner cases + pushes some problems on the application to resolve

- in file system journaling, the checkpoint must always come after the journal commit

- denial of service is an example of an attack on the security concept of availability

- NSF clients simply retry sending command after a timeout because it would be too difficult to figure out why no response was received by the client

- LRU never experiences Belady’s Anomaly

- purpose of priority boost is to prevent starvation

- segment can be shared by multiple processes

- paging has the problem of internal fragmentation

- bounded buffer problem can be solved using only semaphores to control concurrency

- binary semaphore is equivalent to a mutex lock

- thread requesting a resource never causes a deadlock if that thread doesn’t currently have any other resources assigned to it

**False:**

- in access control matrix, a file would be an example of a subject

- when a router detects a packet loss it requests the host to resent the packet

- one cause of latency in SSD is seek time

- a software bug that can be used to gain access to a system is an example of a threat

- in symmetric encryption algorithms, one key is used to encrypt and a different one is used to decrypt

- In RR, best performance is when time quantum is small w/ respect to CS time.

- Linux Completely Fair Scheduler processes w/ low nice values always run before high nice values

- Thrashing is result of insufficient multiprogramming

- If a page’s contents in memory are different than they are on disk, the valid bit must be set to invalid

- illegal memory access results in page fault

- translation lookaside buffer (TLB) is used to search for free space in physical memory

**Consider a file system that uses Unix-style inodes. The system uses 8-KB blocks and 4- byte pointers. Multi-level indexing is used with an inode containing 8 direct pointers, 1 single indirect pointer, 1 double indirect pointer and 1 triple indirect pointer. What is the first file address that will be mapped to the double indirect pointers**

The 8 direct pointers point to addresses from 0 to 8\*8K-1

The single indirect pointers point to addresses from 8\*8K to 8\*8K+(8K\*8K/4)-1

The double indirect pointers to addresses starting at 8\*8K+(8K\*8K/4) = 64K + 8K\*2048 = 16,842,752.

**Matching:**

bitmap: data structure for tracking free blocks

Asymmetric: a cryptographic key system in which some keys are intentionally shared publicly

Track: a physical circular path around a disk where the data is stored

TCP: an example of the transport layer in networking

Directory: contains a list of file names and the inodes they are associated with

Idempotent: principle that makes it safe to retry sending commands any number of times

Socket: the software structure to which a server binds

Mutex: synchronization mechanism that provides only locking, not signaling

Fork(): makes a copy of the current process

CPU bound: a process that frequently exceeds its time slice

Semaphore: mechanism that can provide both locking and signaling

TCB: created as a result of calling pthread\_create()

Kernel mode: system calls are implemented using traps because they require this

Pipe: a queue for inter-process communication

Locality: reason caching can improve performance

TestAndSet: machine instruction that can be used to implement mutex locks

**Assume a HDD is being used. What’s an advantage to putting inodes at the start of the disk? What’s an advantage to putting inodes at the start of the first data blocks of the file?**

When inodes are together at the start of the disk there may be less seek time to find multiple files in a directory. When inodes are close to the contents of the file there is less seek time to go from the inode to the data of the fill.

**In the FSS the inodes for a file are placed near the start of the same block group as the file’s data blocks. Assuming a HDD, explain an advantage to this approach rather than placing all the inodes near the start of the disk**

This has the same advantages of putting inodes at the start of the disk for files that are close together in the directory tree. But there I an additional advantage that the inodes are closer to the file data then they would be if placed at the start of the disk.

**Assume SSD is being used. Does the FFS approach still have any advantages or would putting the inode at the start of the first data blocks of the file be better?**

SSD still benefits from faster sequential access, but there is not seek latency so whether blocks are close or far in the logical address space doesn’t make a difference. So there would be no difference in either of the two approaches.

**How many disk operations needed to fetch inode for */var/log/boot.log***

Assuming the inode for root directory is in memory, the reads will be:

(1) root data block (2) var inode block (3) var data block (4) log inode block (5) log data block (6) boot.log inode block**.**

There are 6 read operations

**What if the bitmap containing the info about the free disk blocks was lost due to a crash? Is there any way to recover from this disaster?**

It’s possible to completely recover the bitmap. Starting at the root directory every directory and file can be recursively visited. When the inode of a file or directory is visited it’s possible to discover what other blocks tha inode is pointing to. If the bits in the bitmap are marked as used for each block encountered, at the end the bitmap will be recovered.

**How many disk operations needed to fetch inode for */var/log/boot.log***

Assuming the inode for the root directory is in memory, the reads will be:

(1) root data block (2) var inode block (3) var data block (4) log inode block (5) log data block (6) boot.log inode block**.**

There are 6 read operations

**Suppose that a computer can read or write a memory word in 10 nsec. Also suppose that when an interrupt occurs, all 32 CPU registers, plus the program counter are pushed onto the stack. What is the max # of interrupts per second this machine can process**

To service an interrupt the CPU registers need to be written to memory and before the interrupt returns the registers need to be read back from memory. Assuming there is no caching 10 nsec is taken for each read or write of each word.

33 words \* 10 nsec \* 2 = 660 nsec

The max # of interrupts per second is 1 / 660 nsec = 1.5 million

A table of data with black text

Description automatically generated with medium confidenceA table of numbers and letters

Description automatically generated with medium confidenceA number of numbers and a mathematical equation

Description automatically generated with medium confidence

**Suppose a process is running on an OS that is a guest of a VM. Assume the page size is 1KB (1024 bytes) for both OS’s. If the process reads from virtual address 2,050, what address will be read from the machine’s memory?**

Virtual address 2050 is located on page 2 because that page starts at address 1KB\*2 = 2048.

It has an offset from the start of the page of 2050 – 2048 = 2

From table, virtual page 2 is stored at “physical frame” 1

But the guest OS’s physical frames are themselves virtualized, “physical frame” 1 is stored at machine frame 0

MFN 0 start at address 1KB\*0 = 0

Adding the offset the final machine address is 0+2 = 2

**In a man-in-the-middle attack the attacker secretly relays and possibly alters communications between two parties who believe that they are in direct communication with each other. Describe CIA with respect to man-in-middle attack. Which are compromised?**

Confidentiality: the attacker is impersonating both sides of the communication to see and possibly decrypt all information

Integrity: the attacker can modify the messages being passed between each side

Availability: the attacker can selectively drop messages being passed between the two sides making a service not available

A white grid with black text

Description automatically generated

1. Read the inode of os.txt to get size and location of first data block

2. Because a new data block needs to be allocated for the file, read the data bitmap to find the location of a free data block

3. After finding a free data block, update the data bitmap by setting a bit to indicate the block has been taken

4. Write the data from char\_buffer to the new data block

5. Update the inode to point to the new data block. Also, update other meta data such as the length of the file and time of last modification.

**Explain how the distributed file system AFS uses callbacks. What’s an alternative approach? What are the advantages and disadvantages of callback over the alternative approach?**

When a client fetches a file, the whole file is cached on the client. This means all future reads and write of the file are performed only from the cached version. Because the cached version can become out of date the client depends on the server notifying it of changes to the file by sending a notification callback message. Callback has the advantage of fewer message from the client to the server because the client doesn’t need to poll the server for file updates like in NFS. Callback has the disadvantage that the server needs to be stateful, specifically it needs to remember all of the client that have files in their cache. This can be a problem if the server crashes and forgets its state.

**Deadlock Conditions**: Mutual Exclusion, Hold-and-Wait, No Preemption, Circular wait **Mutual Exclusion**: Threads claim exclusive control of resources that they require **Hold and Wait**: Threads hold resources allocated to them while waiting for other resources **No Preemption**: Resources cannot be forcibly removed from threads that are holding them **Circular Wait**: There exists a circular chain of threads such that each thread holds one or more resources that are being requested by the next thread in the chain **Total Ordering**: Prevents circular wait; force threads to lock files in specific order **Deadlock Avoidance**: We only need to stop one of the four conditions to prevent deadlocks; we can modify the scheduler to not allow two threads to be scheduled concurrently **Detect and Recover**: Fallback strategy; detect deadlock and reboot system when it happens **File Systems**: Collection of files and directories that provide persistent storage **CPU**: Central Processing Unit of a computer; primary control of the machine **Memory**: Where instructions and data are stored within a computer **Bus**: Communication system that transfers data between components; groups of wires **Peripheral/Device**: Internal or external device that connects to a computer to do something **Device Controller**: Bridge between CPU and I/O devices; handles signals(both in and out) **Device Driver**: Software that operates a device; gives computer access to functionalities **Interrupt**: Signal emitted by a device/program that signals the OS to do something **DMA(Direct Memory Access)**: How the OS interacts with a device; allows CPU to run a process during the copying process and I/O of a different process **Hard Disk Drive(HDD)**: Main form of persistent data storage **Sectors**: Blocks within the address space to break down storage further **Track**: Each circle around the center of the drive, a surface has thousands of tracks **Rotational Delay**: Time for a sector to rotate around the disk head **Seek Time**: Time for the disk arm to change position to the correct track **Random Access**: The software requests addresses in random order, causing random latency **Sequential Access**: Software requests addresses in increasing order **Shortest Seek Time First**: Pick requests on the nearest track to complete first **File**: An array of bytes that has an identifier assigned to it called an inode number **Directory**: A list of pairs that also has an inode. Pairs have a name and an inode number **Root Directory**: The head of the directory tree; root directory has no parent directory **Absolute Path**: The path through the directory tree to get to a file; /bar/foo/bar.txt **File Descriptor**: A file’s index in the computer’s array of files(**Process Control Block**) **open()**: Takes a file name and options, opens a file in the current directory **read(int fd, void \*buf, int count)**: Reads count bytes from file fd into the buffer buf **write(int fd, void \*buf, int count)**: Writes count bytes from buffer buf to the file fd **close(int fd)**: Closes file fd, file descriptor no longer refers to a file and can be reused **lseek(int fd, int offset, int whence)**: Changes the location of the next read/write **Hard Link**: Connects a filename to an inode; filename1 filename2 **Soft(Symbolic) Link**: Gives an alias to a name; -s filename1 filename2 **Metadata Region**: A few blocks on the disk reserved for the file system metadata **Data Region**: The rest of the blocks on the disk that are used to store data of the files **Contiguous Allocation**: Similar to memory allocation for data; causes external fragmentation **Inode**: Contains information about a particular file **Data Bitmap**: Stores which blocks on the disk are free/used within the data region **Inode Bitmap**: Stores which blocks are free/used in the metadata region **Direct Indexing**: A way to index where the inode of a file points to the data blocks of the file **Indirect Indexing**: An indirect pointer points to an inode with more pointers **Double/Triple Indirect Indexing**: Double has two levels of indirection, triple has three, etc **Caching**: Hold popularly used blocks to decrease number of times blocks are read from disk **Write Buffering**: Batch multiple updates into one smaller set of I/O operations **Berkeley Fast File System(FFS)**: Keeps related stuff together to limit searching **Block Groups**: A grouping of blocks that has a superblock, inode blocks, and data blocks **Path Locality**: Consecutive file accesses are likely to be near each other **Journaling**: Before making a change, store a note to the disk about what is going to happen **Transaction**: One thing that will happen to the system, should be on the note **Commit**: Write the transaction to the commit block **Checkpoint**: Write the metadata and data to their final locations in the system **Flash-Based Solid State Drive(SSD):** Made from flash memory, no moving parts **Cell**: Transistor creates this; stores a bit in a single-level cell or up to three bits in trip-level cell **Page**: Holds several cells, typically is around 4 KB **Block**: Consists of multiple pages, typically around 128 KB in size **Read Page**: Client provides a page number to read; doesn’t depend on location of page **Erase Block**: Block must be set to all 1; tends to be slower than read **Program Page**: Writes the pages by setting 1’s and 0’s; time is between read and erase **Flash Translation Layer(FTL)**: Where the flash controller virtualizes the flash **Log-Structured FTL**: Uses an in-memory table to map virtual pages to physical pages; Every write moves the page to a different physical location **Garbage Collection**: Removes old content that was left on the disk but isn’t needed anymore **Wear Leveling**: Flash block has limited number of erases before it can’t be used anymore **Virtual Machine Monitor/Hypervisor(VMM)**: Creates illusion of being hardware to an OS **Supervisor Mode**: VMM reduces the privilege mode before calling guest OS trap handler **Information Gap**: If the OS has no useful work, it just spins in its scheduler loop **Para-Virtualization**: Guest OS has small modifications to operate more effectively **Confidentiality**: Only allow information maintained by a system to be accessed by authorized **Integrity**: Only allow a resource to be modified by authorized parties **Availability**: System can be accessed at requested times by authorized parties **Authenticity**: A computer system can verify the identity of a user **Intruder**: Those who attempt to breach the security of a system **Vulnerability**: Weakness in the security of a system **Threat**: Anything that leads to a loss or corruption of data/physical damage to hardware **Attack**: Attempt to break the security; can be either accidental or malicious **Malware**: Software designed to exploit, disable, or damage a system **Trojan Horse**: Program that looks legitimate but can take control of your computer **Spyware**: Program frequently installed with legit software, but displays ads/captures data **Ransomware**: Locks up data via encryption, demands payment to unlock it **Sniffing**: Intruder “sniffs” communication between users to get secret data and information **Masquerading**: Intruder communicates with the receiver to gain intel or information **Man-in-the-Middle**: Intruder communicates with both the sender and receiver to gain info **Denial of Service(DoS)**: Overload the target computer to prevent it from doing work **Access Control**: Regulating what actions subjects can perform on general objects **Object Access Control**: Resources on which an action can be performed(files, tables, etc.) **Subject Access Control**: Human users representing users from which objects are protected **Access Mode**: Controllable actions of subjects on objects **Access Control Matrix**: Describes the access modes granted to subjects on objects **Access Control List:** Alternative to matrix; each column of matrix is stored in separate list **Capability**: Group of three things containing a subject, object, and access mode **Encryption**: Passing plaintext through an algorithm to get an unreadable group of characters **Decryption**: Passing the unreadable characters through an algorithm to get readable text **Plaintext**: Human readable text, most files are in plaintext **Ciphertext:** What comes out of an encryption algorithm; unreadable by humans **Symmetric Key System**: Same key is used to encrypt and decrypt; key must be secret **Asymmetric Key System**: Public-key encryption based on each user having two keys **Public Key**: Published key that is used to encrypt data **Private Key**: Key known only to an individual user that is used to decrypt data **Authentication**: Identify users and verify that they are who they say they are **Nonce**: A once in a lifetime number to use in one session, new number for every session **Defense in Depth**: Use multiple layers of defense; multiple independent methods used **Layered Architecture**: A way to design software that used defense in depth **Trust**: A trusted system is relied upon to an extent to enforce a specific security policy **Trusted Computing Base(TCB):** Everything in trusted system necessary to enforce policy **Trusted Platform Module(TPM)**: Hardware that controls what can be done on the machine **Virtual Machine**: Present the users with only the resources they need; fake an OS **Sandbox**: Protected environment where a program can run and not endanger the system **Honeypot**: Fake environment that is intended to lure in an attacker **Protocol**: Defines the interfaces between layers in the same system/same layers **Link**: Physical connection between nodes **Packet**: Block of data being communicated **Host**: Computer/Device connected to the network **Switch**: Nodes with multiple links that forward data/packets from one link to another **Router**: Forwards data between networks **Client/Server**: Has two types of communication channels; request/reply and message stream **OSI Architecture**: Open Systems Interconnection, seven different layers **Physical Layer**: Handles transmission of raw bits over a communication link **Data Link Layer**: Collects streams of bits into a frame, delivers frames to hosts **Network Layer**: Handles routing among nodes within a packet-switched network **Transport Layer**: Implements a process-to-process channel, exchanges messages **Session Layer**: Provides a namespace; ties together different streams of one application **Presentation Layer**: Concerned about the format of data exchanged between users **Application Layer**: Standardizes the common type of exchanges **Internet Architecture**: Does not imply strict layering; Wide at the top/bottom, narrow in the middle; needs to be protocol specification and at least one representative implementation **Socket**: The point where a local application process attaches itself to a network **IP Address**: Identifies a single device within a network **Port**: Port numbers are used to identify individual processes; 16-bits unsigned **TCP**: Transmission Control Protocol; communication standard, allows message exchanges **UDP**: User Datagram Protocol; Used for communication throughout the internet **Bind**: Binds newly created socket to a specified address, address combines IP and port **Listen**: Defines how many connection can be pending on the specified socket **Accept**: Carries out the passive open; does not return until connection is established **Connect**: Does not return until TCP has established connection; invokes send and receive **Bandwidth**: Width of the frequency band **Latency**: Transmit + Propagation + Queue **Propagation**: Distance / Speed of light **Stateful Server**: Maintains information about the client’s state; slow recovery, complex **Stateless Server**: A server that does not have a client state **NFS**: Stateless protocol, server remembers nothing from the previous requests **File Handle**: Has a volume identifier, inode number, and generation number **LOOKUP**: Obtain a file handle **READ**: Read from a file at specified location for x bytes **WRITE**: Write to file at specified location for x bytes **GETATTR**: Get file attributes **Generation Number**: Identifies the version of the inode **Retry on Failure**: Client simply retries request after a timeout **Idempotency**: Performing operations multiple times has the same effect of performing once **Cache Consistency**: Client can’t get most recent version or reads out-of-date cache **Update Visibility**: Problem when the client gets the wrong version of a file from the server **Stale Cache**: When the client reads an out-of-date cache **Flush-on-Close**: Cache is always flushed after the application closes a file **Attribute Cache**: Local cache that updates contents only after a timeout **AFS**: Client side code is called Venus and server side code is called Vice **Scalability**: It is easily able to grow based on modularity **TestAuth**: Test whether a file has changed(used to validate cache entries) **Fetch**: Fetch the contents of the file **Store**: Store this file on the server **Whole-File Caching**: After open(), all contents are copied to local disk, read() and write() commands are performed only on the local copy, on close() file is flushed back to the server **Callback**: Server promises to inform client when file is modified, reduces server interactions **Heartbeat Protocol**: Client sends periodic message to server and expects a response | **Deadlock** | **File Systems** | **VMs** | **Security** | **Networking** | **Distributed Systems** |

**Calculate total time to transfer a 1000KB file. Assume an RTT of 50 ms, a packet size of 1KB data. Bandwidth is 1.5 Mbps, and data packets sent continuously.**

Bandwidth = 1.5 Mbps = 0.1875 MB/s

Packet size = 1KB = 1024 B

Transmit time per packet = 1024B / 187500B/s = 5.461 ms

Transmit time for 1000 = 5461 ms

Propagation + queuing time = RTT / 2 = 25 ms

Transfer time = 5461ms + 25ms = 5486ms

**Must wait 1 RTT before sending next packet.**

(5.461ms + 50ms) \* 1000 = 55461ms

**Explain the difference between the transport layer and the network layer.** Both are part of the network stack, and the transport layer relies on the network layer to complete its tasks. The transport layer provides communication between processes, the processes are identified by port numbers, and it is expected to be resilient to packet loss or out of order packets. The network layer provides communication between hosts, the hosts are identified by IP addresses, and no attempt is made to resent, or reorder lost or out of order packets.

**Why do the routers not have a transport layer?** Routers participate at network layer and below, they look at each packet’s IP address and forward packet to next appropriate node, they don’t try to check for lost/out of order packets. Adding transport layer mechanisms at the router level makes routing slow, expensive, and inflexible.

**For efficient sharing of the network infrastructure, TCP breaks long application layer messages into small packets (typically less than 1500 bytes). Explain why the packets arriving at the webserver can be out of order.** The hosts have no control over the route packets will take over the network, some packets might go to router A and others to router B. Communication over the network is asynchronous, some packets may be lost, some packets may take a longer/slower route, so packets from router A and router B could arrive at the webserver in any order.

**The arrows in the diagram show an example TCP communication from the PC to the Webserver, the layer boundaries have been labeled (a) to (h). List all boundaries (a to h) where packets might arrive out of order.** On the PC, the application must provide the correctly ordered message to the transport layer, where there would be no reason for the transport layer, network layer or network interface to change the order of packets. Once the packets reach the network there is not guarantee over which route packets will take or how long a particular route will take. The transport layer (TCP) on the webserver maintains a receive window, it collects packets in the window and uses acknowledgements to have lost packets resent. Only a complete (contiguous) window of packets is forwarded to the application layer.

A diagram of a network router

Description automatically generated

A **stateless server** like NFS doesn’t remember what it told a client in a previous response. For security, check file permission on each request.

**FIFO:** no preemption. **SJF (Shortest Job First)**: CPU assigned to the process that has the smallest next CPU burst. No preemption. **STCF (Shortest Time to Completion First)**: priority queue sorted by job length. If new job arrives with a shorter time to completion, it preempts. **RR**: FIFO queue, job on CPU gets time slice, preempt when time expired.

**System uses 16KB blocks and 8-byte pointers. Multi-level indexing is used with an inode containing 12 direct pointers, 1 single indirect pointer, 1 double indirect pointer and 1 triple indirect pointer. What is the max size of a file that can be stored in the file system?**

A block can hold 16KB / 8 bytes = 2048 pointers

(12 + 2048 + 2048^2 + 2048^3) \* 16KB

(12 + 2^11 + 2^22 + 2^33) \* 2^14 = (8594130956 \* 16384) / 2^40TB=128TB

**Describe how many blocks need to be read to perform a read of the data at the address**

4096 / 16384 = block # 0

The block < 12, so its directly pointed to by the inode, so 2 reads

32768 / 16384 = block # 2

The block < 12, so its directly pointed to by the inode, so 2 reads

1048576 / 16384 = block #64

Block is >= 12, but < 2060, so pointed to by single indirect pointer, 3 reads

268435456 / 16384 = block # 16384

Block is >= 2060 but < 4196364, pointed by double indirect pointer, 4 reads

**Why rm unlinks file?** May be more than one file name mapped to the inode. If the file is deleted, those other directory mappings would be come invalid. Unlinking removes the directory mapping. Only when all files to the inode are unlinked will the inode be deleted

**Consider 320 GB SATA drive. Suppose workload is 10,000 reads to sectors randomly scattered across the disk. How long with these 10,000 requests take (total) assuming the disk services requests in FIFO order?**

Disk access time = seek time + rotation time + transfer time

Seek time: each request requires a seek from a random starting track to a random ending track. For seek time use the average of 12 ms.

Rotation time: once the disk head settles on the right track, it must wait for the desired sector to rotate under it. The average time it will take is ½ rotation. 0.5 \* 60s/5400RPM = 5.56 ms

Transfer time: the disks surface bandwidth is at least 100 MB/s, so transferring one sector (512 bytes) take at most 5120ns (000512 ms)

Total time: # requests \* (seek + rotation + transfer) = 10,000 \* (12ms + 5.56 ms + 0.00512ms) = 10,000 \* 17.56512ms = 175.6s

**Suppose the workload is 10,000 reads to 10,000 sequential sectors on the outer most tracks of the disk. How long will these 10,000 requests take (total) assuming the disk services requests in FIFO order?**

Seek time and rotation time per request are the same as above.

Transfer time: 10,000 sectors \* 512 bytes/sector \* 1/(100 MB/s) = 51.2ms

So 10,000 requests will take 12ms + 5.56ms + 51.2ms = 68.76ms

**How do OSs that use paged virtual memory prevent processes from reading or writing the physical memory or other processes?**

All memory accesses by a process are treated as virtual. The hardware MMU translates virtual addresses to physical addresses by looking in the page table. If the page table doesn’t have a translation, then the memory access isn’t allowed. The page table only maps to addresses the process is allowed to access

**Why is there need for a machine mode?** Allows adding an extra level of protection such as putting the OS in a virtual machine, hypervisor, or sandbox or using a trusted platform module

**An OS provides access control which prevents unauthorized reading of data, what are the advantages of encrypting data stored in the computer system?**

Security depth. Even though the OS has access control, it shouldn’t be relied on as the only security mechanism for sensitive data like passwords. Systems are hacked, if the attacker gets access to the password file its better if the file is encrypted to add an extra layer of defense

**Suppose a distributed system is using acknowledgement-based communication. How does the sender distinguish between the message being lost, the receiver being not available, or the acknowledgement being lost?** The sender has no way to know.

**Explain how the sender deals with a lost acknowledgement.** After a timeout, it resends the original message. Messages have a sequence # attached so the receiver will know if it receives the same message twice

**Explain how the sender deals with an acknowledgement that is significantly delayed.** From the perspective of the sender this is the same as a lost acknowledgement. If the acknowledgement doesn’t arrive before the timeout, it resends the original message

**3 parts of a socket? Which layer of the 7-layer OSI model does each part deal with?**

Protocol – transport layer

Port – transport layer

IP address – network layer

**Suppose a per-subject access control list is used. Deleting an object in such a system is inconvenient b/c all changes must be made to the control lists of all subjects who did have access to the object. Alternative?**Keep both a per-subject and per-object access control list. The per-object control list can be used to efficiently find all subjects that have access to the object. Then removing the object for all subject lists is more efficient.

**Some file systems allow disk storage to be allocated at different levels of granularity. How could we take advantage of this flexibility to improve performance?**

The advantage is less internal fragmentation will lead to less wasted space. Suppose a file only uses half of a block, then the other half would have been wasted space. With 512 byte subblocks the file will occupy exactly two subblocks and waste no space.

**What modifications could be made to the bitmap free space management scheme in order to support this feature?**

There any many possible ideas. The most naïve approach would be to make a bit map that is 4 times larger so that every bit represents one sub-block. A more efficient approach might be to combine the ideas of a bitmap and free list. The bitmap would indicate if the entire block is free. When no free blocks remain a linked list indicates which sub-blocks are free