4. What is *Belady's anomaly* and what property does LRU have (and FIFO lack) that causes FIFO to exhibit it.

In computer storage, Bélády's anomaly is the phenomenon in which increasing the number of page frames results in an increase in the number of page faults for certain memory access patterns. This phenomenon is commonly experienced when using the First in First Out (FIFO) page replacement algorithm. The discovery of Belady's anomaly lead to the search for an optimal page-replacement algorithm, which is simply that which yields the lowest of all possible page-faults, and which does not suffer from Belady's anomaly. algorithm is that the page that has not been used in the longest time is the one that will not be used again in the near future. (Note the distinction between FIFO and LRU: The former looks at the oldest load time, and the latter looks at the oldest use time.) **1. Why is it critical for a microkernel operating system to provide very fast context-switching and interprocess communication (IPC)?** Context-switching is the switching of the CPU (central processing unit) from one process or thread to another. A process (also sometimes referred to as a task) is an executing (i.e., running) instance of a program. And IPC is a set of programming interfaces that allow a programmer to coordinate activities among different program processes that can run concurrently in an operating system. This allows a program to handle many user requests at the same time. It is important for a microkernel OS to provide very fast context-switching and interprocess communication. Shared memory is strictly speaking also an inter-process communication mechanism, but the abbreviation IPC usually only refers to message passing, and it is the latter that is particularly relevant to microkernels. IPC allows the operating system to be built from a number of small programs called servers, which are used by other programs on the system, invoked via IPC. Most or all support for peripheral hardware is handled in this fashion, with servers for device drivers, network protocol stacks, file systems, graphics, etc. **2. What is it that makes a context-switch so expensive on many architectures? How about changing from user mode to kernel mode?** More expensive CPUs perform noticeably better. Context switching itself has a cost in performance, due to running the task scheduler, TLB flushes, and indirectly due to sharing the CPU cache between multiple tasks. Switching between threads of a single process can be faster than between two separate processes, because threads share the same virtual memory maps, so a TLB flush is not necessary. When a transition between user mode and kernel mode is required in an operating system, a context switch is not necessary; a mode transition is not by itself a context switch. However, depending on the operating system, a context switch may also take place at this time. **10. What fundamental action that we take for granted in a centralized system must occur for access control lists to function in a distributed system?** Here one entity (dept or an individual) is responsible for overseeing access to all corporate resources. This type of administration provides a consistent and uniform method of controlling users access rights. Access controls are security features that control how users and systems communicate and interact with other systems and resources. Access is the flow of information between a subject and a resource. A subject is an active entity that requests access to a resource or the data within a resource. E.g.: user, program, process etc. An resource is an entity that contains the information. E.g.: Computer, Database, File, Program, Printer etc. Access controls give organization the ability to control, restrict, monitor, and protect resource availability, integrity and confidentiality **11. What advantage does the elevator algorithm have over shortest seek first (SSF)?** SSF might go back with the arm. The next seek time might be longer due to this. Also some data might stay in the buffer longer than expected: Assume this: Data comes into the buffer. SSF for byte 1: 10 SSF for byte 2: 20. SSF writes byte 1. New data arrived in the buffer: SSF for byte to: 25 (changed due to the write of byte 1) SSF for byte 3: 10. Byte 3 is written. The elevator algorithm writes all data in one continuous line, resulting in more direct clearing of the buffer and preventing some data staying in the buffer longer than anticipated. So it is just more predictable. In real world: Only important in high disk load situations. **15. A RAID system uses many disks in parallel to increase the transfer rate to and from the disks. Ideally, with n disks in parallel the throughput would improve by a factor of n. Unfortunately, there is the small write problem. What is it? How does it cause performance to suffer?** The "write hole" effect can happen if a power failure occurs during the write. It happens in all the array types, including but not limited to RAID5, RAID6, and RAID1. In this case it is impossible to determine which of data blocks or parity blocks have been written to the disks and which have not. In this situation the parity data does not match to the rest of the data in the stripe. Also, you cannot determine with confidence which data is incorrect - parity or one of the data blocks. **7. Briefly describe the similarities and the differences between capabilities and access control lists** Both constructed in a matrix || Both describe permissions || Access control lists - contain information about user access (add/remove user) - better for systems that have fewer user-oriented operations || Capability lists - contain information about file/object permissions **8. What are two advantages of bit-mapped disk block allocation? How does each relate to file system performance?** Only need to keep track of one bit per sector/block (simple) - one bit per sector || Random access is really fast || Deletion - just need to flip the bit **9. Deadlock can be prevented by negating any one (or more) of four necessary and sufficient conditions (Coffman 1971). Briefly describe the consequences of negating each of these conditions** 1. Mutual Exclusion Condition--The resources involved are non-shareable.Explanation: At least one resource (thread) must be held in a non-sharable mode, that is, only one process at a time claims exclusive control of the resource. If another process requests that resource, the requesting process must be delayed until the resource has been released. 2. Hold and Wait Condition--Requesting process hold already, resources while waiting for requested resources.Explanation: There must exist a process that is holding a resource already allocated to it while waiting for additional resource that are currently being held by other processes. 3. No-Preemptive Condition --Resources already allocated to a process cannot be preempted.Explanation: Resources cannot be removed from the processes are used to completion or released voluntarily by the process holding it. 4. Circular Wait Condition The processes in the system form a circular list or chain where each process in the list is waiting for a resource held by the next process in the list. **14. In a few sentences each, give two methods for deadlock recovery. What are the immediate consequences of each method?** Abort a process or preempt some resources when deadlocks are detected. Ignore the problem all together - If deadlocks only occur once a year or so, it may be better to simply let them happen and reboot as necessary than to incur the constant overhead and system performance penalties associated with deadlock prevention or detection. This is the approach that both Windows and UNIX take. **6. Briefly describe the clock page replacement algorithm.** In a computer operating system that uses paging for virtual memory management, page replacement algorithms decide which memory pages to page out (swap out, write to disk) when a page of memory needs to be allocated. Paging happens when a page fault occurs and a free page cannot be used to satisfy the allocation, either because there are none, or because the number of free pages is lower than some threshold. **12. The shortest seek first (SSF) tries to minimize the total amount of arm motion. Unfortunately, it has a serious flaw: what is that flaw? How does the elevator algorithm address this flaw?** Shortest seek first (or shortest seek time first) is a secondary storage scheduling algorithm to determine the motion of the disk's arm and head in servicing read and write requests. However, since the buffer is always getting new requests, these can skew the service time of requests that may be farthest away from the disk head's current location, if the new requests are all close to the current location; in fact, starvation may result, with the faraway requests never being able to make progress. **14. What is the major disadvantage of using a linked list to maintain a list of free disk blocks? How would you address this issue?** The major problem is that it is inefficient to support direct-access; it is effective only for sequential-access files. To find the ith block of a file, it must start at the beginning of that file and follow the pointers until the ith block is reached. Note that each access to a pointer requires a disk read. **10. What are two advantages that the elevator algorithm have over shortest seek-time first (SSF)?** Elevator algorithm reaches all tracks - lower chance of starvation. Smaller variance in response time **4. The shortest job first (SJF) scheduling CPU algorithm and the shortest seek-time first (SSF) disk arm scheduling algorithm have a common benefit, and a common problem. The common benefit is that they are provably optimal. In what way are they considered optimal? What is their common problem?** Common benefit: optimal because in both cases, jobs that take the shortest time (or shortest time to get) are completed first - leaving longer, bigger processes for later. Common problem: long jobs (or jobs that take long to seek) will delay every job after them. **8. Briefly describe the LRU and FIFO page replacement algorithms** The prediction behind LRU, the Least Recently Used, algorithm is that the page that has not been used in the longest time is the one that will not be used again in the near future. The first-in, first-out (FIFO) page replacement algorithm is a low-overhead algorithm that requires little bookkeeping on the part of the operating system. The idea is obvious from the name – the operating system keeps track of all the pages in memory in a queue, with the most recent arrival at the back, and the oldest arrival in front. **2. What is a cache miss and why is it desirable to avoid them?** Much larger than the L1 cache. Since L2 contains the same data as L1, making L2 about the same size as L1 causes it to have a high local miss rate. This is true since if we miss in L1, it is likely that we'll miss in L2 as well resulting in performance that is not much better than using main memory alone. Therefore, it must be much larger. Cache miss results in a page fault which means we have to access main memory - takes too long. **11. How does a journaling file system (JFS) speed system start-up?** A journaling file system is a file system that keeps track of changes not yet committed to the file system's main part by recording the intentions of such changes in a data structure known as a "journal", which is usually a circular log. In the event of a system crash or power failure, such file systems can be brought back online more quickly with a lower likelihood of becoming corrupted. **12. What are the two dominant sources of delay when reading a sector from disk?** Three separate steps take place when reading a particular byte or series of bytes of data from a hard disk. First, the I/O head moves so that it is positioned over the track containing the data. This movement is called a seek. Second, the sector containing the data rotates to come under the head. When in use the disk is always spinning. At the time of this writing, typical disk spin rates are 7200 rotations per minute (rpm). The time spent waiting for the desired sector to come under the I/O head is called rotational delay or rotational latency. **15. Three ways of allocating space for a file are: contiguous (all the blocks are allocated contiguously), using extents (files are allocated in contiguous extents), or on demand (where writes are placed in the nearest available block). Briefly describe the advantages and disadvantages of each of these approaches.** Contiguous: Data in each file is stored in consecutive blocks on disk. It is simple and efficient indexing. Random access well-supported, but difficult to grow files. Must pre-allocate all needed space, and wasteful of storage. Linked Allocation: • Idea: Linked list of blocks, each pointing to next. It is to grow and fast sequential access, but slow non-sequential access. Indexed Allocation: • Idea: Store ordered list of block pointers. It is good for random access, not bad for sequential, but size is limited, not as fast for sequential access. Contiguous - everything is allocated at once - easier to find (random access); disadvantages: wasteful. Extent-based: Advantages: can eliminate metadata overhead, less file fragmentation; storage efficiency and performance aren't so good. On demand: Advantages: takes less time to place file; disadvantages: fragmentation. **15. What are the critical issues in providing fast file system start-up when the system is booted following a crash? What would you do to speed this process (running fsck is not considered fast)?** File system info needs to be repaired and cleaned. Issues: a modification could be partially executed - i.e. some modifications written, others not, Creating link to a file: new directory entry refers to file descriptor, but reference count wasn't updated in file descriptor. Ordered writes. Write-ahead logging. **A microkernel is an operating system technique that places minimal functionality in the kernel, but instead provides that functionality through user-state servers. What is a major performance obstacle that microkernel designers must face?** Drivers can touch only authorized I/O ports, but access to kernel calls is also controlled on a per-process basis, as is the ability to send messages to other processes. Processes can also grant limited permission for other processes to have the kernel access their address spaces. As an example, a file system can grant permission for the disk driver to let the kernel put a newly read-in disk block at a specific address within the file system's address space. **Briefly describe why round robin scheduling is appropriate for an interactive environment, while non- preemptive priority scheduling is not.** A big advantage of round robin scheduling over non-preemptive schedulers is that it dramatically improves average response times. By limiting each task to a certain amount of time, the operating system can ensure that it can cycle through all ready tasks, giving each one a chance to run. With round robin scheduling, interactive performance depends on the length of the quantum and the number of processes in the run queue. A very long quantum makes the algorithm behave very much like first come, first served scheduling since it's very likely that a process with block or complete before the time slice is up. A small quantum lets the system cycle through processes quickly. This is wonderful for interactive processes. Unfortunately, there is an overhead to context switching and having to do so frequently increases the percentage of system time that is used on context switching rather than real work. Advantage: Round robin scheduling is fair in that every process gets an equal share of the CPU. It is easy to implement and, if we know the number of processes on the run queue, we can know the worst-case response time for a process. Disadvantage for non-preemptive priority: This will not execute later-arriving high-priority processes in a timely manner, because it does not check for new processes until the current process is finished. Non-preemptive priority scheduling is therefore better for a fixed, idle environment. **In terms of deadlock avoidance, there are three states: safe, unsafe and deadlocked. Briefly define each of these states, and say why it is unwise to enter the unsafe state. Which is**

possible, safe->unsafe, unsafe->deadlocked, safe->deadlocked Safe State: The key to a state being safe is that there is at least one way for all users to finish. In other analogy, the state of figure 2 is safe because with 2 units left, the banker can delay any request except C's, thus letting C finish and release all four resources. With four units in hand, the banker can let either D or B have the necessary units and so on. An unsafe state may not necessarily lead to deadlock, it just means that we cannot guarantee that deadlock will not occur. Thus, it is possible that a system in an unsafe state may still allow all processes to complete without deadlock occurring. This is not a good state to be in because we cannot be sure that the process will be able to finish. Deadlock occurs when multiple processes (two or more) are dependent on the result of another waiting process. No processes can run, release resources, or be awakened in this state. Safe->unsafe possible, unsafe->deadlocked possible, safe->deadlocked not possible. **4. Briefly describe a page replacement algorithm that approximates least-recently used (LRU) but uses only the referenced bit** Second Chance Page algorithm. A modified form of the FIFO page replacement algorithm, known as the Second-chance page replacement algorithm, fares relatively better than FIFO at little cost for the improvement. It works by looking at the front of the queue as FIFO does, but instead of immediately paging out that page, it checks to see if its referenced bit is set. If it is not set, the page is swapped out. Otherwise, the referenced bit is cleared, the page is inserted at the back of the queue (as if it were a new page) and this process is repeated. **There are many policies for variable-sized partition memory management. In a sentence or two each, describe and rank (in terms of wasted memory): first-fit, worst-fit and best-fit.** First-fit: In the first fit algorithm, the allocator keeps a list of free blocks (known as the free list) and, on receiving a request for memory, scans along the list for the first block that is large enough to satisfy the request. If the chosen block is significantly larger than that requested, then it is usually split, and the remainder added to the list as another free block. Worst-fit: The first block on the free list will always be large enough, if a large enough block is available. This approach encourages external fragmentation, but allocation is very fast. Highest amount of memory wasted (used to avoid fragmentation within memory -- where there are small chunks of memory that cannot be used for anything) Best-fit: the free block with the "tightest fit" is always chosen. The fit is usually sufficiently tight that the remainder of the block is unusably small. Least amount of memory wasted. **The five (basic) states that a process can be in are: new, ready, blocked, running and terminated. Describe the transitions among these states, and state when they occur. Hint: The best way to do this is to draw a diagram.** The OS needs to know which state a process is in so it can control which uses the processor next. A running process has control of CPU. A ready process is in a queue (called the ready queue) and will eventually get access to CPU. A blocked process is waiting for some event. When it happens successfully the process will be ready and will join the ready queue. A new process has just been born and will be admitted to the ready queue as soon as it is fully set up. An exit process is finished and will be removed from memory. A suspend process is swapped out and when there is room in memory it will be swapped back in. **13. Suppose that you have a clock chip operating at 100kHz, that is, every ten microseconds the clock will tick and subtract one from a counter. When the counter reaches zero, an interrupt occurs. Suppose that the counter is 16 bits, and you can load it with any value from 0 to 65535. How would you implement a time of day clock with a resolution of one second. Hint: you only need addition to implement the time of day clock.** Crystal oscillator with fixed frequency (only when computer is on) • Typically used to time short intervals (~ 1 second) • May be used to correct time-of-day clock. Time-of-day clock (runs when system is off) • Keeps minutes, hours, days • May not be too accurate...• Used to load system clock at startup. Time kept in seconds and ticks (often 102–106 per second) • Number of seconds since a particular time • For many versions of Unix, tick 0 was on January 1, 1970 • Number of ticks this second • Advance ticks once per clock interrupt • Advance seconds when ticks "overflow". **2. What are the critical issues in providing fast file system start-up when the system is booted following a crash?** OS must protect data structures used for free space management. OS must keep in-memory and on-disk structures consistent • Update free list when block is removed: change a pointer in the previous block in the free list • Update bit map when block is allocated • Caution: on-disk map must never indicate that a block is free when it's part of a file • Solution: set bit[j] in free map to 1 on disk before using block[j] in a file and setting bit[j] to 1 in memory • New problem: OS crash may leave bit[j] == 1 when block isn't actually used in a file • New solution: OS checks the file system when it boots up...Managing free space is a big source of slowdown in file systems **4. Given that you have a RAID-4 system composed of five disks, fill in the formula for P = D0 ⊕ D1 ⊕ D2 ⊕ D3 and for D1 = D0 ⊕ D2 ⊕ D3 ⊕ D4 . Now, suppose that D3 fails, give the formula for D3 and briefly list the steps for reconstructing its data. 11. Suppose that you have a UNIX file system where the disk block size is 1kB, and an i-node takes 64 bytes. Disk addresses take 32 bits, and the i-node contains 8 direct addresses, one indirect, one double indirect and one triple-indirect (the rest of the space in the i-node is taken up with other information). An index block is the same size as a disk block. Suppose you write one byte each at offsets 0, 1024, 65536 (64k), and 1048576 (1M). How much total disk space does the file consume (including overhead)?**
Single indirect block: 1KB(8+2^8+2^16+2^24) = – Assuming 32-bit addresses, we have 4 bytes per block pointer, so 1 KB/4 = 256 blocks – So … 256 * 1 KB = 256 KB • Double-indirect block: – 256 * 256 * 1 KB = 64 MB • Triple Indirect block: – 256 * 256 * 256 * 1 KB = 16 GB • Total: ~16 GB Disk blocks are 1024 bytes Inodes are 128 byte Disk addresses are 4 bytes Inodes contain space for 64 bytes of data, 8 direct addresses, 1 indirect, 1 double- indirect, and 1 triple-indirect. An index block is 1024 bytes. A single indirect pointer can point to 1024/4 = 256 disk addresses = 256 KB data. A double indirect pointer can hold 256 * 256 = 65536 disk addresses = 64 MB data. A triple indirect pointer can hold 256 ^ 3 disk addresses = 16 GB data So, 1 byte file : inode stores the data : so 128 bytes (inode size) 1025 bytes : inode + 1 data block : 128 + 1024 = 1152 bytes 65536 bytes : inode + 8 direct data blocks + 1 indirect pointer block + 56 data blocks : total (128 + 65K bytes) = 66688 bytes 1 MB : inode + 8 direct data blocks + 1 indirect pointer block + 256 data blocks + (note: 760 K are still left) 4 double indirect pointer blocks of overhead + 760 data blocks : total : 128 + 1029 KB = 1053824 byte. 6. The LRU page replacement algorithm has the property that for memory M of capacity m and a reference string r, M(r, m) ⊆ M(r, m + 1). What is this property called and why does this preclude LRU from suffering Belady's anomaly? What exactly is Belady's anomaly? Property called the inclusion property. **13. We all must run programs that we didn't write. Either we trust them, or we attempt to confine them in some way. But, confinement is quite difficult since there can be covert channels for leaking information. Give a simple example of a covert channel that a malevolent program might use to leak information to its cohort.** a covert channel is a type of computer security attack that creates a capability to transfer information objects between processes that are not supposed to be allowed to communicate by the computer security policy. Exchange information using file locking Assume n+1 files accessible to both A and B A sends information by • Locking files 0..n-1 according to an n-bit quantity to be conveyed to B • Locking file n to indicate that information is available B gets information by • Reading the lock state of files 0..n+1 • Unlocking file n to show that the information was received ❖ May not even need access to the files (on some systems) to detect lock status!
12. Capabilities have two well-known problems: the capability propagation problem and the capability revocation problem. In a few sentences (at most), describe both of these problems. Propagation: easy for capabilities to be transmitted across security domains. Revocation: easy for domain access rights to be taken away. **14. Suppose that you have a clock chip that operating at 3 MHz, that is, three times per microsecond the clock will subtract one from a counter. When the counter reaches zero, an interrupt occurs. The counter is 16 bits. What value do you use to reset the counter? Use C-like pseudocode to show how to implement a time of day clock.** 3000000 65535. Keep software counter uint32_t 0-3000000 - when it reaches either 3000000 or 0, increment seconds. 16.A Diffie-Hellman key exchange requires a base a, and a prime p. A must choose an x and B must choose a y. It then proceeds as follows: A → B : a, p, ax (mod p) B → A : a y (mod p) (a) What secret do A and B now share? a^(xy) mod p (b) Why doesn't the adversary also know this secret? Adversary doesn't know because they do not know x and do not know y (c) What would be necessary for the adversary to learn this secret? - either x or y. **11. An RSA (Rivest-Shamir-Adelman) cryptosystem is composed of e, d, n = pq and where d × e ≡ 1 mod ϕ(n). The public key is e, the private key is d, n is also public but p, q and ϕ(n) are private.** (a) n = pq and φ(n) = (p-1)(q-1). (b) If the adversary can discover p or q, the formula for computing d is. (c) The formula for E(m) = m^(e) mod n and for D(c) = c^d mod n. (d) Why is it safe for n to be made public? Because it is hard to factor a large number. Usually it is a big number.

Some Terminologies:

Because seeks are so expensive (milliseconds!), it helps to schedule disk requests that are queued waiting for the disk – FCFS/FIFO (do nothing) • Reasonable when load i s low • Long waiting times for long request queues – SSTF (shortest seek time first) • Minimize arm movement (seek time), maximize request rate • Favors middle tracks – SCAN (elevator) • Service requests in one direction until done, then reverse • Discriminates against the highest and lowest tracks – C-SCAN • Like SCAN, but only go in one direction (typewriter) • Reduce variance in seek times. In general, unless there are request queues, disk scheduling does not have much impact– Important for servers, less so for PCs• Modern disks often do the disk scheduling themselves– Disks know their layout better than OS, can optimize better– Ignores, undoes any scheduling done by the OS **RSA algorithm for public key encryption** Private, public key pair consists of KR = (d,n), KU = (e,n) • n = p×q (p and q are large prime numbers)• e is a randomly chosen integer with GCD (e, (p–1)×(q–1)) = 1 • d is an integer such that (e×d) ≡ 1 (mod (p–1)×(q–1))❖ p & q aren't published, and it's hard to compute them: factoring large numbers is "difficult"❖ Public key is published, and can be used by anyone to send a message to the private key's owner❖ Encryption & decryption are the same algorithm: E(KU,M) = Me MOD n (similar for KR)• Methods exist for doing the above calculation quickly, but... • Exponentiation is still very slow• Public key encryption not usually done with large messages **Buffer overflow** Buffer overflow is a big source of bugs in operating systems• Most common in user-level programs that help the OS do something • May appear in "trusted" daemons❖ Exploited by modifying the stack to• Return to a different address than that intended • Include code that does something malicious ❖ Accomplished by writing past the end of a buffer on the stack **Not Recently Used Page Replacement Algorithm** The not recently used (NRU) page replacement algorithm is an algorithm that favours keeping pages in memory that have been recently used. This algorithm works on the following principle: when a page is referenced, a referenced bit is set for that page, marking it as referenced. **Busy waiting:** ❖ Use a shared variable (turn) to keep track of whose turn it is ❖ Waiting process continually reads the variable to see if it can proceed • Called a spin lock: the waiting process "spins" in a tight loop reading the variable ❖ Avoids race conditions, but doesn't satisfy criterion 3 for critical regions. **Aging Replacement Algorithm** The aging algorithm is a descendant of the NFU algorithm, with modifications to make it aware of the time span of use. Instead of just incrementing the counters of pages referenced, putting equal emphasis on page references regardless of the time, the reference counter on a page is first shifted right (divided by 2), before adding the referenced bit to the left of that binary number.