

Name:

ID:

Final Exam
CS402
14 December 1999

You have 2 hours for this exam. The exam has 6 pages. There are 100 possible points. Show all your work for partial credit. This exam has been brought to you by the letter π and the number e.

Definitions

Each question includes the number of points. Answer all questions in this section.

1. Give one or more terms that describe the following:
 - a) The four requirements for deadlock (4 pts)
Answer: Mutual exclusion, non-preemption, hold and wait, circular wait
 - b) The name for the condition where increasing the number of physical pages in a VM system results in more page faults (1 pt)
Answer: Belady's Anomaly
 - c) Two possible states of a **process** (2 pts)
Answer: Ready, Running, Blocked
 - d) The structure that UNIX® uses to keep track of the sectors allocated to a given file (1 pt)
Answer: i-node
 - e) The structure that MS-DOS or windows uses to keep track of the sectors allocated to a file (1 pt)
Answer: FAT (File Allocation Table)
 - f) Four layers of the ISO OSI networking stack (4 pts)
Answer: Application, Presentation, Session, Transport, Network, Link, Physical
 - g) A program that provides a service (or pretends to) to steal a password or shared secret (1 pt)
Answer: trojan horse
 - h) Two elements of the Virtual Memory system (hardware or software) (2 pts)
Answer: MMU, page table, page replacement algorithm, page fault handler
 - i) Smallest addressable piece of data on a disk drive (1 point)
Answer: sector or block
 - j) Three memory protection systems (3 pts)
Answer: base/limit registers, software keys, virtual memory

Short Answer

Each question is worth 10 points. Do all questions in this section.

2. In older UNIX systems the password file was readable by any user. Why does this not allow any user who can read this file the ability to log in as any other user? Your answer should include a description of how a user's password is verified by a UNIX system. (6 pts)

Answer: The password information is the result of applying a one-way function to the user's password. Because such a function is not easily invertible, translating from hash value to password is not feasible. Verifying the password consists of hashing it and comparing the hashed values.

These days, most UNIX `/etc/passwd` files do not include the password information. Give 2 reasons for hiding that information, even considering the answer to the first part of this question. (4 pts)

Answer: Bad passwords remain too common, and putting the password information in the clear invites a dictionary attack. Furthermore, it having the information in the clear allows an attacker to make the attack from a different machine, not even giving the attacked machine a chance to detect it.

3. Consider a FAT-based (File Allocation Table) file system. Entries in the table are 16 bits wide. A user wants to install a disk with 131072 512-byte sectors. What is a potential problem? (3 pts)

Answer: Each 16 byte entry in the table is an address of a sector on the disk. The OS can address 65536 sectors, and the disk has more than that.

Describe a solution to this problem (4 pts) and explain the trade-offs involved. (3 pts)

Answer: Make each FAT entry access a logical sector that is 2 physical sectors. This trades increased internal fragmentation against maintaining the size of the FAT, and backward compatibility.

Alternatively, you can increase the size of the FAT to be 131072 17-bit entries, which increases the table size to 278528 bytes (at least) and increases the complexity of the decoding process. It also ruins any backward compatibility.

Name:

ID:

4. In a network, both the link layer and the transport layer may provide error detection facilities. Give 2 reasons that a transport-level error detection mechanism is needed to ensure end-to-end data integrity. (6 pts)

Answer: The link layer is not guaranteed to provide error detection, and the transport can't be sure which link levels it's running over. Even if all the link levels provide error detection, this does not detect errors in the intermediate systems in the network.

Most operating systems do not perform error detection on disk blocks, but rather trust the disk hardware itself to detect its own errors. Why do you think this is the case? (4 pts)

Answer: The only source of corruption outside the disk itself is the operating system, processor or memory. If the processor or memory is malfunctioning, it will not be possible to perform a reliable checksum calculation. Similar arguments apply to OS. In the network case, the OS can check another system's faulty checksum calculation, it cannot check it's own. (Although, it might be possible on a multiprocessor...)

Most people made an efficiency argument here, and that was worth 3 points. You should consider why the OS can't check the disk blocks.

5. One of the reasons that a UNIX `fork` system call is expensive is that it duplicates the entire parent's address space for the child process. In most cases the child will access only a very small portion of the address space copied from the parent before replacing it with a new address space (for example, by making an `exec` system call). A common solution to this problem is to create an address space for the child that initially shares the pages of the parent's address space, but copies any pages that child writes when the child writes them. This way the changes appear only in the child's address space, not the parent's. The result is that the child's address space mostly consists of pages shared with the parent process but also includes a few unshared pages that are modified copies of the parent's pages. This strategy is called copy-on-write.

You're given a paging system where each page has its own backing store, and each page's read and write permissions are determined by the current page table (a physical page may appear in 2 page tables with different permissions). There is no TLB (translation lookaside buffer); all page references are resolved through the page table. Any time a page is not in memory, or is accessed with the wrong permissions, the page fault routine is called. The OS handles the problems of maintaining consistency of shared pages - if a shared page is evicted or brought into memory, all page tables that reference it are properly updated. You have room in the page table to add a few flags to each entry.

Describe a system for implementing copy-on-write. Specifically describe how you create a copy-on-write address space in the `fork` syscall (5 pts) and furthermore how you can detect that a copy-on-write page has been written to and must be copied (5 pts).

Answer: To create a copy on write address space, copy the parent's address space, make the permissions to any writable pages read-only, and add a copy-on-write flag to only those writable in the parent space. When the page fault handler is called because of a permission fault on a copy-on-write page, you must copy it and restart the fault.

At least one person noted that in reality this is a much thornier problem than the one presented here. In a real system, the address space should be copy-on-write for **both** parent and child. You didn't have to do the harder version for full credit, but you did have to mention using the read-only bit to get extra faults.

Long answer

Each question is worth 20 Points. Do all questions in this section.

6. This question asks you to consider adding links to the Nachos file system. **This is a design question; you should not write code.**

These questions refer to adding **hard** links to the system.

- a) What changes would you make to the internals (directory, fileheader, free map, etc.) of the file system to support hard links. (3 pts)

Answer: You would have to add a link count to the file header.

- b) How do the semantics of the `Remove` system call change (2 pts), and how do you implement that change? (3 pts)

Answer: Now a file is removed from the file system when there are no more links and no thread has the file open. Previously, a file could always be marked for deletion by `Remove`. `Remove` and `close` must be modified to check for both conditions, rather than only the condition that no thread has the file open.

- c) What new system calls, if any, must be added to the system. Give the signature of any new calls, e.g., `Open(char *file)` (2 pts)

Answer: Add a `Link(char *old, char *new)` to create a new link to an existing file.

These questions refer to adding **soft** links to the system. Soft links are also called symbolic links.

- d) What changes would you make to the internals (directory, fileheader, free map, etc.) of the file system to support soft links. (3 pts)

Answer: The directory would have to be modified to store the soft link's pathname in the directory.

- e) What existing system calls have to be modified and how to support soft links (5 pts)

Answer: `Open` needs to be changed to implement the recursive lookup implied by soft links. If the file being opened resolves to a soft link, `Open` must open the file pointed to by the soft link. A new error mode for a soft link that doesn't point to a file is introduced. `Remove` must be modified in a minor way to remove soft links correctly. (`Remove` is not required for full credit, and is only worth 1 point alone).

- f) What new system calls, if any, must be added to the system. Give the signature of any new calls, e.g., `Open(char *file)` (2 pts)

Answer: Add a `SymLink(char *old, char *new)` to create a new symbolic link to an existing file.

7. Generally we've talked about each operating system component in isolation. This question asks you to think about ways in which they interoperate. For each pair of systems below, give a specific way that they interact (or that they could interact). Telling me that the file system and I/O system interact because they both use the disk is not worth more than a point, and may be worth none. Telling me that the file system and I/O system interact when they determine the mapping from logical blocks → physical blocks which impacts the size of file system structures, and the efficiency of the disk usage because larger logical blocks imply more internal fragmentation on the disk is a more complete answer.

- a) How does the virtual memory system (say, demand paging) interact with the process system (scheduling or creation)? (5 pts)

Answer: A demand paging system will be at its worst at process creation, with an empty address space. The two systems have to work together to pre-fetch a reasonable working set without overloading the paging system by bringing in the entire address space. If the VM system supports shared text (or copy on write), the two need to work together to take advantage of it.

- b) How do the virtual memory system and the file system interact? (5 pts)

Answer: The VM system may page executables directly from the file system, which requires coordination to make sure that users cannot write to files being used as backing store and that deletion of such files is disallowed or handled properly. The advantage is that pages swapped from the file system do not need to be copied to swap space.

Swap files can generally be put in the file system, but if so, the VM and file systems need to coordinate such use (maybe with a quota system) to prevent the swap files from running the file system out of room. Using the file system interface simplifies the VM system, but adds per-swapfile overhead.

In a system that allows memory mapped files, the two systems have to interact closely to guarantee consistency between the file cache and the VM allocated to a memory mapped file.

- c) Name another way (not the example above) that the file system and the I/O system (probably the disk drivers) interact. (5 pts)

Answer: Feature coordination. If the disk implements interleaving, the file system implementation should not do its own interleaving. The result can be to make both systems worse than useless - actually detrimental. Similar negotiations have to be done for features like disk caching or bad block renaming.

- d) How do the security system (e.g., file permissions) and the virtual memory system (5 pts)?

Answer: This one's deliberately tricky. If the OS allows memory mapping of files, the memory mapping interface has to respect file system permissions - potentially every time that the file is read or written. This is a potentially very complex interaction.

Although I didn't have it in mind when I wrote the question, the TENEX bug for guessing passwords is an even better example of this interaction. The TENEX bug was worth full credit here.