Final Examination
CS 111, Fall 2016
UCLA

Name: _____

This is an open book, open note test. You may use electronic devices to take the test, but may not access the network during the test. You have three hours to complete it. Please remember to put your name on all sheets of your answers.

There are 3 questions on the test, each on a separate page. You must answer all of them. Each problem you answer is worth 33% of the total points on the test. You get 1% for putting your name on the test.

You must answer every part of every problem. Read each question CAREFULLY, make sure you understand EXACTLY what question is being asked and what type of answer is expected, and make sure that your answer clearly and directly responds to the asked question.

I am looking for depth of understanding and the ability to solve real problems. I want to see specific answers. Vague generalities will receive little or no credit (e.g., zero credit for an answer like "no, due to the relocation problem."). Superficial answers will not be sufficient on this exam.

Organize your thoughts before writing out the answer. If the correct part of your answer is buried under a mountain of words, I may have trouble finding it.

1. Many file systems end up storing the same bit patterns over and over again for many different files, effectively wasting a lot of disk space storing redundant data. Some file system deal with this issue by using deduplication, a technique that tries to ensure that any bit pattern occurring more than once anywhere in the file system is only stored on disk one time. Other files containing the same bit pattern do not store unique copies of the data, but instead "point to" (in some fashion), the single copy. Of course, one must choose some granularity (in terms of length of bit patterns) at which one deduplicates. There would be no benefit in deduplicating all occurrences of the bit pattern "0", for example. (Note: if you propose some technique other than deduplication for dealing with this problem and do not address the points below concerning deduplication, you will get 0 on this question. Don't do that.)
    a. What granularity would you suggest deduplicating at? Why?
    b. What algorithms and data structures will you use to detect duplicates?
    c. Should deduplication be applied to directories or just to regular files? Why?
    d. Does the use of deduplication cause any new security problems? Why?
    e. Describe how you would alter the FFS to allow it to perform some useful form of deduplication. Describe the changes to metadata structures that your alteration would require. How would the following file operations change (in their underlying OS implementation) in your design: `create()`, `open()`, `read()`, `write()`, `unlink()`?
    f. Will deduplication improve, degrade, or not alter file system consistency concerns? Explain.
    g. Will deduplication perform better when the underlying storage medium is a hard disk drive or a flash drive, or will it have the same performance for both? Why?

    a. *(5 points) Deduplication at high granularity won't find much to work with. For instance, file deduplication will only help for totally identical files. Deduplication at a low granularity (such as a few bytes) won't scale, since you will need to keep too many indices for different patterns. If one is going with a fixed size, a disk block/IO cache block size is reasonable. It has the advantage of fitting in easily with other file system elements. Probably doing a variable sized granularity not aligned on a particular boundary would provide the most bang for the buck, but adds much complexity.*
    b. *(7 points) Deduppping by comparing full patterns will be an immense loser. You will be best off deduppping by comparing the outputs of a hash algorithm applied to the bit patterns. Avoiding collisions is vital, and you don't want the hash to be too long, both because you'll be storing the hash and you will be doing lots of comparisons on it. A cryptographic hash is a good choice. In terms of data structures, you need a general data structure shared by the entire file system that keeps track of the hashes you've already seen. It needs to scale well and not be a performance*

bottleneck, particularly on file reads. Either a database or perhaps a hash table (of the hashes themselves) might be a good choice. In addition to a pointer to the actual data with that hash, it should contain a reference count of how many places in the file system use that block, since once all such references are either deleted or altered to have different contents, the data pointed to by the entry should be freed. Other data structures designed to handle fast pattern lookups, like tries, might also work well. The key to getting full points is a complete description of how you will use the data structure.

c. *(2 points) There will be little value in dedupping directories. They usually contain only directory names and inode pointers. Both are too short to benefit from dedupping. The chance that there will be identical names and inodes spanning more than one entry in two directories is low.*

d. *(2 points) Other than introducing more code that might contain security flaws, dedupping a file system should not cause security problems. It does not grant users access to anything they do not already have the right to access. They only get access to a dedupped block in a file they can't read if they already have a copy of that dedupped block in a file they can access. It would be better to hide the deduplication from users, since if they know that a particular block of their own file is duplicated in some other file, there is a minor privacy leak.*

e. *(12 points: 7 for general discussion, 1 each for proper description of the operations) Assuming we are working at the block level, each pointer to a block in an inode might point either to a block or to a dedup entry in the dedup database/table. The latter point to a single instance of the duplicated block. Dedupping can happen as files are written, on close after a write, when a written block is flushed from the block cache to the disk, or at some point in the background. Any (or some reasonable combination) are acceptable. Dedupping as writes occur is probably not a good idea, though, as it may require multiple hashes and table lookups when a program performs multiple writes to a single block.*

   There are small details of how this approach would impact the FFS inode layout, such as how are you going to keep the information about whether a block pointer is a true block pointer or a pointer to a dedup entry. There is currently no room in either the pointer or the inode for that. You didn't need to address such issues.

   Pointing to a data block without being concerned about whether it is a dedup block or not is possible, but it leads to further issues that must be addressed if the answer goes in this direction. These include how do you make sure when you write a block that you're only writing your own block, and how do you determine when you can garbage collect a block.

   For the operation descriptions, details may vary depending on how the student has suggested the system be designed.

(a) *On create, nothing special must happen. So far, there is no data in the file, so there are no duplicates. Of course, the inode structure used to hold the file's pointers is allocated in a different format.*

(b) *On open, no special action is required. The file's inode will not itself be deduplicated, and open will only reference the inode.*

(c) *On read, the pointer for the requested block is followed either to a standard data block or to a dedup pointer, which in turn is followed to the duplicated data block. In either case, the actual data is returned to the user without indication of whether it went through a dedup pointer.*

(d) *On write, there are several possibilities. A write to a brand new block should wait till the block is filled. Once filled, the block's data is hashed and checked against the dedup table. If a duplicate is found, the file's pointer for the block is set to the dedup table entry. On write to a block that is currently dedupped, a new copy of the block must be made and the final state of the block run through deduplication. It is unlikely to match the old hash (since some data was written), but it might match another. On a write to a block that is not currently deduplicated, we check for duplication at the usual moment, possibly replacing the pointer to a data block to a point to a dedup hash entry.*

(e) *On unlink, we must check all the data blocks of the file to see which, if any, were deduplicated. If some were, we need to decrement the reference count of those blocks in the table of hash entries. Should any reference counts go to zero there as a result, we need to free the actual data block associated with it and free the entry in the table.*

f. *(2 points) In many ways, dedupping will not affect consistency. It will have some minor costs. First, the update of a file's state is now potentially a bit more complex, since not only a file's inode and data block must be consistently written, but also the dedup table entry. Second, if there is corruption in one block of the disk, and that block is either in the dedup table or in one of the duplicated blocks, more files will be affected. On the up side, corruption in a duplicated block will be easy to detect, since the dedup table contains a hash of the proper contents of the block.*

g. *(3 points) Dedup should perform very well on flash, but might run into some problems on hard disks. By its nature, it will be impossible to ensure that a duplicated block is located in close physical proximity to all files that share it. So for some files, some blocks will require long seeks to access them. But it doesn't really matter where the data is located for a file in the flash memory, since access times are constant for all of them. As a minor point, flash wear issues will likely be improved by dedup, since we can avoid writing multiple copies of a block when it is duplicated.*

2. Each of the following three situations poses interesting allocation or locking problems (figuring out what or how to lock while avoiding deadlocks, hangs and bottlenecks). For each resource, (1) describe the best approach (2) justify why the situation demanded this approach, and (3) tell me specifically how you would apply that approach to the problem.

(a) A cellular communications processor that controls calls, power-levels, and the movement of calls across adjacent cells supports millions of calls and hundreds of concurrent operations. To prevent conflicting updates, there are locks associated with each transceiver, call, and cellular sector. Some operations start by locking a single object (e.g., a call) but as they progress it becomes clear that it will also be necessary to lock other objects (e.g., one or more sectors). How can we prevent concurrent multi-object operations from deadlocking?

In this situation, I would use total ordering (since sharing is not an option, all-up-front is precluded by the problem, and revocation is always nasty).

I would define a relative ordering among the types of objects (e.g. transceivers, calls, sectors) and an ordering within each type of object, and then require processes that locked objects to always lock them in order. If a process holds a higher numbered mutex and needs to get a lower numbered one, it must release the higher numbered one, take the lower numbered one, and then re-take the higher numbered one (lock dancing). Note that when releasing a mutex for a lock dance, there is a possibility that someone else will get it, and therefore the resource must be in a completely consistent state when the mutex is released. Similarly, when it is reobtained, the program cannot assume that it was not changed in the interim.

(b) Swap space on secondary storage. If there is not room on the swap device to swap out one of the in-memory processes, we will be unable to make room to swap in and run any of the processes that are currently swapped out. How can we prevent such a situation?

Swap space is a commodity resource, and as such much easier to manage. I would avoid deadlocks by requiring reservations be obtained whenever a process is created or expands the size of its virtual address space. The banker's algorithm would work in this situation, but I would not allow any over-booking. I would only allow processes virtual addresses to be created/expanded as long as there was guaranteed swap-space for them ... after which I would fail subsequent requests.

(c) A network lock manager provides locking services for a very wide range of distributed resources that are shared by thousands of clients. The clients are a wide variety of applications running on many different operating systems, and not all of the resources they need are managed by the network lock manager. How can we ensure network locks will not contribute to deadlocks among the client applications?

Mutual exclusion is surely necessary, and since the applications are running on different operating systems and machines, we probably can't prevent them from blocking while holding network locks. Because the managed resources are arbitrary (and only some of them are owned by the lock manager), that probably makes ordering impractical. This leaves us with preemption. We should implement leases with lock breaking.

2. A company with around 500 employees in several offices around the globe wishes to provide its employees with a wide range of software services hosted at several company server machines. They expect to add, remove, and alter services regularly, and they expect to host hundreds of such services. They wish to maintain tight access control on the services, with much more specificity than merely enforcing that particular users are or are not allowed to use the service. Instead, for each service, the service designer must be allowed to specify particular elements of the service that are under access control. (For example, one service might want access control on whether the user is allowed to print results from the service, while another might want access control on whether a particular user is allowed to escalate a problem to a higher ranking employee, and a third service might use access control to permit only system administrators to determine how much of a server's CPU use a particular user is expending, while not allowing the administrators to see the specific tasks being performed.) Since the company doesn't know all services they will ultimately offer, they cannot specify all service elements that might need access control, but they know that not all services will need access control for all possible elements. The service designer should be able to specify which actions offered by his service are under access control. (For instance, service A might want access control on printing, while service B does not.) The access control mechanism is to be built into the operating system, to ensure trustworthiness, uniformity, and to protect from application bugs. Users will occasionally be added or deleted, and access permissions for particular users and services will change often.

   How would you design the operating system mechanism to provide this kind of access control? Consider the required scale, flexibility, performance, distributed nature of the problem, and security issues, at the minimum.

   It is vital to notice that we are asking for the OS to perform access control on arbitrary pieces of user code, which means that the OS must take over when those pieces of code are invoked. Thus, a good answer to this question requires that you describe how the operating system will become involved so that it can perform access control. One general approach is to register this code with the OS and add system calls to obtain access to it. Another approach is to use the distributed nature of the system to force actions through the OS, by requiring messaging to access the protected code. When the message is delivered, the OS will have a chance to perform access control before invoking the code. Other approaches are possible.

   Assuming the second approach, we should have a new set of system calls related to these types of access permission. Probably it would be best to specify that there are particular portions of a program that can only be accessed by remote users by going through the operating system first. The cost won't be that high, since the remote users are getting to the services by sending messages to the server, which already go through the OS. The messages can be delivered first to the OS component that performs the access control and, if allowed, then delivered to the application.

One set of system calls will require the applications to register a service as under access control. This might be something similar to setting up a socket, but with the understanding that access control has been applied to anything that comes through it. We will also need calls that change the particular access permissions associated with the socket or whatever it is. While one could limit such calls to being made from inside the program that provides the service, it would be easier to build such programs and manage the system if the access permissions could be changed by a system administrator. That implies some kind of persistent information about the access permissions for the service, since they need to be around even if the service is not running. An obvious choice would be to store them in a file that is only accessible by trusted administrators and the system itself. The tools would then read and write the files.

From the scalability perspective, this would work reasonably well. Each service (or service element) being controlled would have one file with its access permissions, which would need to be stored on the server that runs the service. File systems handle scale well. Only access control files associated with active services would need to be opened and accessed, so the costs of the I/O would be low. Presumably services in high use would have their access permissions stored in the block I/O cache, limiting further the I/O required.

In terms of performance, this approach would work well. There need not be any extra kernel boundary crossings, since the permissions would be checked when messages arrive, which are handled in the OS itself. The permission information would usually be cached in RAM. If the checking mechanism is designed to be cheap, in the manner of Unix file access permissions, the costs of the checks themselves would be low. Applications would not need to perform internal checks, since any requests for protected services that were delivered to the application had already been checked.

Distribution issues are reasonable. Services are accessed by messages, and access control is applied to each incoming message. This would require some kind of common name space for remote users, which is reasonable in the environment in question. We can replicate services, as necessary for scaling and load management, since each server will handle its own access control. There would be issues of ensuring consistency of access permission across replicated servers, if we support them, but if we assume changes of access permissions are not common and can be handled with a consensus protocol, that can work. If each message is self-contained and comes with the necessary trusted identity information, we don't need to worry about failure or saving state.

Security will depend on proper authentication. The method used is more or less orthogonal to the rest of the design, but something must be done. Presumably we will want some kind of primal authentication operation for users. This would generate credentials useful for ongoing authentication. We could use authentication servers. We could require each service to perform a login operation (perhaps to invoke a common one used by all services) that generates credentials that expire. We could require remote users to establish an authenticated session with each server, with ongoing crypto providing the

authentication after that point. Other approaches are possible. Any reasonable choice is sufficient. The student must also indicate that messages cannot be tampered with in transit, since that could lead to improper use of a controlled service.

This sample answer contains a lot of detail, more than I would expect from any student. However, each issue that was mentioned in the question had to be discussed in at least a little detail to get full points.

There are entirely different approaches to this problem, which are also OK, to the extent they are developed correctly and address the required issues.