CS 111: Operating Systems Final
Eric Sehun Oh
304184918

Question 1

A. The granularity approach to deduplication should be more fine than it is coarse. If it is too coarse, too many bit patterns will be matched as candidates for deduplication and thus accumulate too much overhead; if it is too fine then only the largest will match and there will be no real optimization done.
B. For data structures and algorithms
   a. One completely inefficient approach for detecting duplicates would be to run through the file system of bit patterns and finding bit patterns of a certain length and storing them in a two-dimensional array of bit patterns to memory locations.
   b. Another data structure that we can use would be a linked list in which the nodes contain the bit pattern and an array of memory locations in which they occur.
C. Deduplication should be applied just to regular files instead of directories for the same reason you cannot create hard links to directories. Hard links are not allowed to a directory, for fear that you will create a cycle in the directory tree. As hard links, link a name to an inode, what we seek to accomplish by having multiple data structure refer to a bit pattern is similar in nature.
D. Deduplication would cause security problems as the different file containing the bit pattern may have different have different permissions allowing them to change the bit pattern which will then propagate to the other patterns that share that bit pattern if it is not handled correctly. In addition, a malicious attacker may target these bit patterns as they are a prime target for a small manipulation that creates a great amount of change and havoc within the file system with a small amount of code.
E. Some changes would have to be made to the FFS in order to perform some useful form of deduplication. For example the create file operation will need some way to check for the bit pattern that is being created within the file, and if it exists, link those bit patterns to the file rather than allocating new bits for the file. Open and read will need a way to access or traverse through the list of the file's mapped bit patterns and combine them together in order to open the file or fetch the contents. Write must be changed in a way such that if the write affects the bits that are pointed to by other files, it must allocate new bits for the file and point to those bits instead of writing to bits that are pointed to by multiple files. Unlink must now be able to access each and every one of those mapped bit patterns of the file and unlink itself.
F. Deduplication will undoubtedly complicate file system consistency. When there are complications, there are consistency concerns. This is because multiple files may share the same bit patterns and the files may have different permissions, etc. When these dependencies of bit patterns to files are not handled properly, a change to a bit pattern may affect many files, or at worst case all files, of the file system.

G. Deduplication will perform better on the flash drive as the flash performs much more quickly than disk drives due to the mechanical seek and rotation costs that the disk must perform.

Question 2

Conditions for a deadlock: the four conditions that must hold for deadlock to occur are as follows: mutual exclusion, hold-and-wait, no pre-emption, and circular wait.

1. Cellular communications processor
    a. The best approach to allow preemption, removing one of these conditions listed above with the use of leases, which are commonly used in distributed applications, is a contract that gives its holder specified rights to some resource for a limited time.
    b. When dealing with systems that must deal with millions of calls and hundreds of concurrent operations, it becomes incredibly difficult to find out lock dependencies and remove circular waits. Mutual exclusion is a property that must hold as updates must occur. Hold-and-wait is also another property which must hold as we are dealing with a system in which it is specified that locks to multiple objects must be acquired. If hold-and-wait is removed in order to prevent deadlock, then there is a chance that we will never get the correct combination of locks to perform the update.
    c. In an update, there are multiple objects that require locks. For each kind of object, we set different lease times, for different levels of de facto lock granularity. If we contain a lock to a call, but need the lock for a sector, we will hold the call lock for a period of time in which we hope to acquire the lock for the sector. If we do acquire that lock then we can perform the update and release the locks. However, if the sector lock is not acquired then we must notify the lease holder of the expiration of their lease and release the lock. Since there are multiple objects for which locks must be acquired, there should be consideration of lease renewals if the chain of locks required is long enough. Consider the case in which the lease for lock A is timed out after lock B is acquired. If the process needs two more locks C, D, there should be consideration of lease times for locks A and B such that there is padding for locks C and D to be acquired. Either, lock A and B lease times must be lengthened, which is bad overall for concurrency or introduce a lease renewal approach such as that the leases for previous locks are renewed if new locks in the lock chain are acquired.
2. Swap on secondary storage

    Consider the case in which there is no room on the swap device to swap out one of the in-memory processes, and is unable to make room to swap in and run any of the processes that are swapped out. The best approach in dealing with this situation is by maintaining a backup swap space that is never used for any other purpose than for a situation such as this. It is a waste of space in that it may never be used in certain scenarios, however, when we enter a blocking/deadlocking scenario such as this, the only other way would be to kill processes manually or even shutdown the system which is a huge loss in performance and reliability. The other approach in killing processes would require additional algorithm of a page replacement policy in having to choose which processes to kill such that we will run into the least amount of page faults possible as making the wrong decision in kicking out pages could cause the program to run ten thousand to a hundred thousand times slower. The other alternative is to run a swap daemon that frees memory, evicting pages until there is a watermark value of pages available. All of these approaches are inefficient and complex. It is just easier to allocate a backup swap space that is utilized only when it needs to because KISS. If not, swap daemon is better than running a page replacement algorithm since its always a great idea to do work in the background to increase efficiency.

3. Network lock manager
    a. The best approach is by seemingly remove mutual exclusion by allowing clients to locally store server resources and cache their updates to the local resource. Thus, even if there is a lock on the resource in the network, the client is not bound to the lock of the network resource and can compute on their local resource.
    b. Here we explore the four conditions that we could choose to negate from the locks of the server in order to prevent deadlocks in clients.
       This is a large distributed system in which dependencies are seemingly incredibly difficult to figure out and thus remove circular wait will take massive amounts of code review, which is not a viable option.
       Removing preemption is a recipe for disaster for server resources. Consider the case in which the update from a client A is preempted for an update from another client, client B. This changes the resources that the previous client A used to compute the update and now those resources have now been changed by client B. The client A must receive a new version of those resources and perform the update again only to hope that this chain of preemption does not happen again. Not viable, horribly inefficient.
    c. Mutual exclusion is a condition that can be compromised if the clients are given a copy of the server's resources which can be stored on their cache. Thus mutual exclusion is seemingly removed by creating two copies of the resources and the client is allowed to access the cached resource, even if the server might have the lock on that resource. This requires that the client communicate with the server before sending them the update in order to ensure that the cached content is up to date with the server's resources. If it is not, then the client just computes again with the new resource of the server. However, consider that there are over a thousand servers which need the server's resources; there is a downside as the clients will constantly ask the server whether or not the contents are up to date and become flooded with these kinds of requests. This was one of the problems encountered AFSv1.

Question 3

The access control mechanism to support operations for the company mentioned in the example would be an RBAC system. The RBAC system allows for a more formal approach than merely including individuals in groups as it is mentioned that all services will need access control for all possible elements.

In order to implement this kind of access control system, I would use something similar to the android access control model which uses permission labels that are like capabilities and thus possession of those capabilities allow for the app to do something. Similarly, if the user in the company has those capabilities, then they are allowed to perform those tasks. Permission labels allow for extreme customization while as access control lists just checks to see whether the user is on the list or no; this allows great flexibility and an immense variation of possible permission labels and can manifest for a company. Because the company needs access control for all services and all elements (services * elements), there will be too many access control lists to cover all the users and access control combinations and thus is not scalable with ACL's. The security issues is that once a permission label is given out, it is hard to revoke the permission label from the system service, although the user may be able to. Thus if the security protocols for the service changes, additional implementation to the service must be added to either give out new permissions based on permissions they already have or reject once accepted permission labels.