Frederick Kennedy
404-667-930
COM SCI 111 Discussion 1C
Final Exam – December 5, 2016

Question 1

a) I will de-duplicate the files when the bit pattern takes up a block.

   i. De-duplicating too few bits may not show any obvious results. Another reason we don't want to de-duplicate at a small length of bit patterns is because the bit pattern may just match at a coincidence. For example, we have a file that contains "Hello" and another file that contains "Hello World!". Though both files have the same start to It (both containing "Hello"), they are two completely different files and we don't want to de-duplicate one of them.

   ii. De-duplicating a bit pattern with too many bits may restrict the number of files being de-duplicated. Though it may seem like an ideal and optimal solution, because we are truly minimizing the amount of redundant data we may have, it is very difficult to find files which contain the same bit pattern as another file.

   iii. De-duplicating files by block may result in a good balance/amount in between a fine-grained granularity and a coarse-grained granularity. This balance helps us achieve the perfect amount of deduplication, ensuring that files are not stored redundantly on the disk.

b)

   i. Algorithm:
   We can use the checksum algorithm to check for the bit patterns. A checksum is simply the result of a function that takes a chunk of data as input and computes a function over said data, producing a small summary of the contents of the data. With this, we can compare one file's checksum with another file's checksum and it may result in the same value.
   One example checksum function is based on exclusive or (XOR). With XOR-based checksums, the checksum is computed by taking the XOR of each chunk of the data block being that we are currently checking the sum of, thus producing a single value that represents the XOR of the entire block.
   Another function that we can use is the cyclic redundancy check. All you do is treat the data block D as if it is a large binary number (it is just a string of bits after all) and divide it by an agreed upon value (k). The remainder of this division is the value of the CRC.

   ii. Data Structure:
   I will use a hash table that maps the block contents to block ID's. In determining whether a certain block is a "duplicate", I will check the contents of the block, run it through the hash table, and search the block element for a match (searching bit-by-bit). Though, this is especially difficult because collisions may occur when using the functions to compute the checksum of different files – collisions are when two data blocks with non-identical contents will have identical checksums.

c) Deduplication should only be applied to files. If a certain directory contains the same contents as another directory in the system, essentially we are wasting a lot of disk space storing redundant data, but because directories don't take up too much space, we can disregard them for being a duplicate with another directory. Thus, we will only de-duplicate regular files. By creating a soft link to one of the original files, we then are able to minimize the amount of data used in the disk. In addition to that, directories contain inodes of different files and we want to preserve this.
Another reason why we shouldn't apply deduplication to directories is because there is a possibility for a directory to share the same size/name with another directory, but have completely different files inside. This may result in us opening the wrong directory that we intended to.

d) One of the security problems that we can encounter is the fact that we can de-duplicate another user's file by accident. The reason is because we are only checking the bit pattern of the file and not checking its metadata. Another security problem that we can face is that bit patterns may seem the same but we are duplicating files that are made not read-only. Also, while de-duplicating files, there may be a timer interrupt happening or a power outage, and without any backup, these files may be lost in the process of de-duplicating and there won't be any soft links after all.

e) FFS is a file system where related things are kept together. I would include the metadata of files in the hash table used to detect duplicates. With this, we are able to preserve original content of files and its data so that when we do file operations, the metadata of files can be obtained as well.
   i. Create() will not be changed at all. The reason is because we are just creating a new file, with no content whatsoever.
   ii. Open() may be changed because we are also obtaining the metadata of files from the hash table. When we open a file that is a soft link, we can go through the hash table and search for the block that contains the file and we can obtain the metadata and also the original file.
   iii. Read()will not be changed at all. The reason is because we are just reading a file, with no content whatsoever.
   iv. Write() will be different than before because instead of writing byte per byte, but we have to write files block by block.
   v. Unlink() will be different now that we have the metadata of files. By having the FFS metadata of files in the hash table, we have to make sure that when we remove the hash table entry, we have to remove all its other contents as well, including the FFS metadata structures.

f) Deduplication may degrade file system consistency concerns. By de-duplicating all files that have a similar bit pattern, we are essentially creating soft links of files, pointing to a single original file. Thus, this creates some more concerns:
   i. What if we wish to change the data of one of the files? That means we have to search for the original file and update it. There are multiple pointers pointing to this original (singular) data block) and hence, when we change one of the files, all the other files will also change (and that is probably not our intention).

ii. What if data corruption happens during deduplication (such as a power outage or a timer interrupt)? When this happens, the pointers may result in pointing to different files and in the end, may not give us the correct result of files.

iii. If indeed the solution in deduplication is to have a hash table to store all the duplicated data, we now have to maintain the hash table and keep track of the block reference counts. Though this isn't a file system consistency concern, it degrades the overall file system.

g) Deduplication may perform better in flash drives compared to hard disk drives. The reason is because in flash drives, we can simply just copy-on-write files unlike hard disk drives, where reading and writing files depend on a spindle that reads data from a platter. Hard disk drives may take up a lot of time in reading and writing compared to flash drives. However, in terms of space, both of them really do not make a significant difference and may perform identically.

Question 2

a)

1. The best approach in this situation is to remove the "hold-and-wait" condition of deadlocks.
2. The situation demands this approach because in a phone call, we don't want to be waiting on the phone for long hours just because we are "on hold" and we have to wait.
3. If the system is unable to obtain the locks necessary to make the call, it should just fail the call instead of "holding and waiting" – on the user's end, it may seem like he or she is waiting for to reach the other end in call when the caller really isn't on a call (and just being placed on hold, thus, having to wait).

b) The swap space is located on a disk. This swap space is usually used for moving pages back and forth out of memory into the disk or into memory from the disk. The operating system is responsible to remember the disk address of a given page.

1. The best approach in this situation is to make sure the swap space does not run out of room. One way is to use the low watermark and high watermark technique on the memory (not disk/secondary storage) and another way is to prevent new memory allocations in the disk.
2. The reason why "ensuring the swap space does not run out of storage" is the best approach is because we cannot delete files from the secondary storage to make room for the swap space. Instead, we can only minimize the usage of the swap space.
3. By using the low watermark and high watermark technique, we are making sure that the memory has enough room to keep all its pages within its memory (and there will be no use in swapping the pages out to put in the swap space). When the OS notices that there are fewer than LW pages available, a background thread that is responsible for freeing memory runs. This thread will evict pages until there are HW pages available. This background thread is usually called the swap daemon (or page daemon).

c)

1. The best approach in this situation is to remove the “no preemption” condition of deadlocks.
2. Because network locks are usually managed by the network lock manager, we cannot trust remote computers to release the locks. That is why, by removing the “no preemption” condition of deadlocks, network locks contributing to the deadlocks will ensure that client will no longer have to wait for other locks while holding locks themselves.
3. One of the ways to implement this approach is instead of using locks, we use a scheduling system that allows the clients to use the network for a certain amount of time before renewing their request for more time in using the network.

Question 3

In order to solve the access control problem, I propose in using an Access Control List with each element for each service.

1. Scale: If we decided to use capabilities for access control, the system will need to maintain its own protected capability list for each service and the list is likely to be fairly long, and perhaps fairly complex. Hence, using access control lists provide a practical solution in maintaining access control.
2. Flexibility: Control is easily maintained by access control lists. By having a list for each service, it allows the flexibility of granting access to a group of users. An example would be:
    a. A company owns a printer for its users.
    b. Access is given to a certain group of users.
    c. The company buys a new computer for printing purposes only.
    d. The access is given to the group of users that are allowed to print.
3. Performance: This approach will have good performance in terms of scaling of services. But when it comes to adding/removing users, it may be much more complex. First, we would have to add the user into the lists that we want the user to be on. Then, if we were to remove a user, we would have to go through all the access control lists and remove them. Overall, Access Control Lists make it easy to figure out who can access a resource, easy to revoke or change access permissions, but as I mentioned before, it is hard to figure out what a subject can access and changing access rights requires getting to the object.
4. Distributed nature of the problem: The distributed nature of the problem is that we need to ensure all the data that we have is unique – basically, all the users should have a unique username/password combination. This is to ensure that the right users have the right access and doesn’t have any access to services it (intendedly) doesn’t have.
5. Security issues: The security issue in maintaining this access control list is not that big of a deal because it is a relatively small company (only has 500 people). So, we can have a database for the users (username and password combination) which will be hosted on a secure server. Also, because it is relatively small, it will be easily maintained – changing, removing, adding users is not that hard compared to a company with almost 10,000 employees.