Christopher Brant
EEL 5737 HW2partA
Due 9/16/19

HW2 Part A

1) Textbook exercise 2.3
    a. Synonyms in cache can cause issues around duplicate objects, duplicate references, as well as issues regarding how to correctly update, validate or invalidate those objects in the cache, and create efficiency issues surrounding these extra operations necessary for dealing with multiple copies/extra synonyms in the cache.
    b. If every object has a unique ID then binding a name to an object would only need to lookup the object at its reference value instead of having multiple copies of the instance of the object and having to update and validate each instance, would only need to update and validate the single instance as all synonyms only reference the object's unique ID instead of the object's many copies/instances. Basically just binds the name to a reference which is much better for this purpose.

2) Textbook exercise 2.4
    This idea is basically an implementation of a cache for search paths to objects and should mean that Louis is correct, however there are two different ways to look at this. Louis would be correct in the instance that the user is consistently looking up the same files or using the same places in the file path over and over and isn't constantly "missing" when searching the ROT first. This would happen if a user is looking up many different files and does not lookup the same files multiple times. In that case he would be wrong, but his theory would have a basis in fact, as in the following case he would be correct. Louis would be correct in the case that a user is constantly looking up or using the same files over and over and if the principles of spatial and temporal locality are in effect, this is a good system. The question does not say how often a user is utilizing files or how many files he is constantly looking up so I believe there are two possibilities, one in which he is right, and one in which he is wrong, which are both described here.

3) Part A Question 3
    a. i.a) The largest number of files possible in this file system is dependent upon how many inodes are in the system. Assuming in this case that only 1 file per inode, with 8 inodes/block, and 2048 blocks reserved for inodes, that is a maximum of 16384 inodes or 16K files, and if we assume inode 1 is reserved for the root of the filesystem that is still 16K files, just 16383 files instead of 16384.
    b. i.b) 8 inodes per block, with 1024 Bytes/inode, assuming 8 Bytes/inode are reserved for the type and reference count for this example, that leaves 1016 Bytes to keep block numbers in per inode. With 32-bit block numbers that gives us 254 possible block numbers per inode, and 254 blocks is equivalent to 2080768 Bytes/file or is equal to 2,032 KB as a maximum file size.

Christopher Brant
EEL 5737 HW2partA
Due 9/16/19

c. ii.a) LOOKUP("data",14)?
   This would return a failure as 14 is not an inode number that we have available in our example of blocks.
d. ii.b) LOOKUP("file1.txt")?
   This would return inode number of 3.
e. ii.c) LOOKUP("programs",5)?
   This would also return a failure since the given filename is not in the directory at block 16 which is pointed to by inode 5 given in the arguments to the call.
f. iii) Block 16 would now contain 5 name/inode number combinations of file1.txt/3, file3.txt/15, back/7, onemore/3, and xyz/3. Just as well, block 4 would change as inode 3 would change since it's reference count would increase to 3 with file1.txt, onemore, and xyz all referencing that same file at inode 3 which is contained in block 4.
g. iv) This example was a little difficult at first to understand as it slightly contradicts itself. After going to the TA's office hours it was cleared up, as the question both states that A and B are *different processes* then references them as threads. I was told to treat them as different processes, and in that case, at Time T3, bufA would have completed its read and contain the characters "Hello", and at Time T5, bufB would have completed its read and also contain the characters "Hello". This is because they are separate processes and would each have their own file table entry for the file "data/file1.txt" opened, and therefore each file table entry would have its own cursor value and neither would affect the other as when process A reads and closes, its cursor would move to the offset after the fifth character, and close, and process B would reference a different entry in the file table with a cursor still at the beginning of the file as these are threads that are not sharing the same file table entry. If that was the case, B would have read the next 5 characters, but it did not.
h. v) It is impossible to create a hard link across multiple disks because of possible cross-system issues arising from the ambiguous nature of implementation across disks. Only symbolic links are possible in this given situation/example. To create a symbolic link for "file4.txt" under "/data". The way this would be done is that a new inode would need to be used in the original hard disk, lets say we use inode 4 in block 4, that inode would have "/mnt/file4.txt" as its data instead of a block number, and it would have a 1 for its reference count, and an S as its type for symbolic link. Then in block 16 we would have to add "file4.txt" as an entry with the inode 4 bound to it.