John Hodson

EEL4736

09/16/2019

Homework 2

A.1) One way to speed up the resolving of names is to implement a cache that remembers recently looked-up {name, object} pairs.

a) What problems do synonyms pose for cache designers, as compared with caches that don't support synonyms?

When there are synonyms for a particular object, that means that there are multiple names (keys) for a given object. This presents itself as an issue when updates happen to the object. Say a given object has two names, a and b. When an update comes across to name a, the object is updated, and the updated object is cached under name a. However, an old version of our object may be cached under name b. Subsequent requests to name b will have actions performed against the old version of the object, resulting in undesired behavior. This is not an issue for a cache that doesn't support synonyms, as within such a cache, you cannot have multiple references to the same object, so it is never possible to have multiple copies that are potentially out of date.

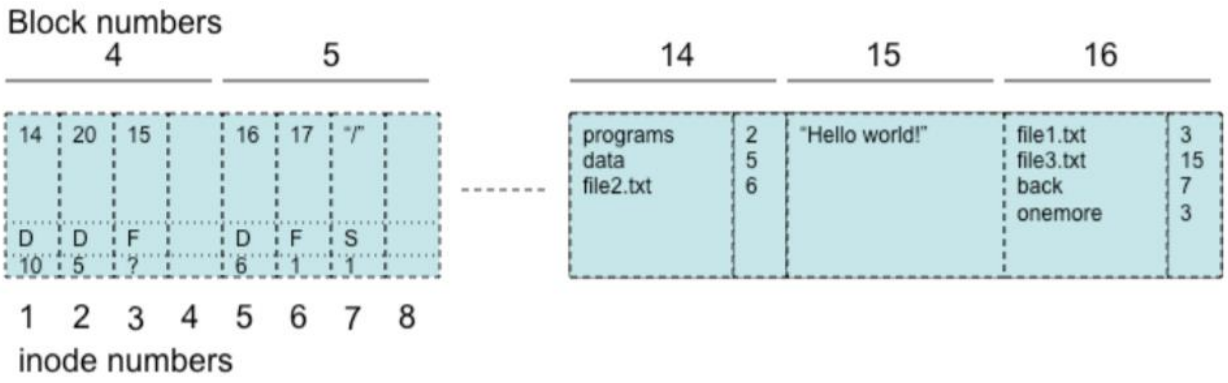
b) Propose a way of solving the problems if every object has a unique ID.

One way that this could be solved is mapping the name to the unique ID and then mapping the unique ID to the object that was cached. This way, when you have multiple synonyms for a particular object, all of these synonyms will map to the same unique ID corresponding to the object of interest. This way, caches will not store the synonyms of the object and only the unique IDs.

A.2) Louis Reasoner has become concerned about the efficiency of the search rule implementation in the Eunuchs system (an emasculated version of the Unix system). He proposes to add a referenced object table (ROT), which the system will maintain for each session of each user, set to be empty when the user logs in. Whenever the system resolves a name through of use a search path, it makes an entry in the ROT consisting of the name and the path name of that object. The “already referenced” search rule simply searches the ROT to determine if the name in question appears there. If it finds a match, then the resolver will use the associated path name from the ROT. Louis proposes to always use the “already referenced” rule first, followed by the traditional search path mechanism. He claims that the user will detect no difference, except for faster name resolution. Is Louis right?

I am assuming that Louis implemented the ROT correctly and there is never an issue with the ROT being out of date due to potential updates to the search path. When a particular name is resolved, then the correct reference is cached in the ROT. This initial name resolution will take more time, as we will have to first search the ROT, then the search path, and then place the resolution into the ROT. For subsequent name resolutions, this will result in faster lookups as we will not have to iterate through the search path to resolve a particular name.

A.3) Consider the figure below representing data stored on a computer’s hard disk and mounted as the root file system. At the bottom of each inode, its type (D (directory), F (file), and S (symlink)) and reference count are shown (e.g. D, 10 for inode 1), based on the UNIX file system described in class.



i) Suppose blocks are 8KBytes (8192 Bytes) in size, and there are 8 inodes per block. Assume an inode uses 8 Bytes to store type and reference count; the rest of the inode space is used to store block numbers. Suppose the disk has 2^32 blocks, and 2048 blocks are used to store inodes.

a. What is the largest number of files that can be stored?

Number of blocks * amount of inodes per block = amount of inodes. You must have at least one inode for each file.

2^11 * 2^3 = 2^14 possible files

b. What is the largest file size that can be stored (in KBytes)?

~~(total blocks – inode blocks) * size of block in KBytes = total KBytes available. The largest file size possible will take up all of the memory available for files.~~

~~(2^32 - 2^11) * 2^14 = 70368710623232 KBytes~~

8 KBytes / 8 Inodes = 1 KByte/Inode

(1 KByte of total space – 8 bytes reserved for Inode metadata) = 1016 bytes available for block numbers.

Since there are 2^32 blocks, you need a 32-bit address to index. 32-bit address means you need 4 bytes per block number.

Therefore, you have 1016 total bytes available / 4 bytes per block number = 254 blocks.

254 blocks * 8 Kbytes/block = 2032 KBytes available for files.

ii) Consider the LOOKUP resolution procedure as discussed in class.

a. What is returned from LOOKUP(“data”,14)?

Undefined behavior / FAILURE, there is an attempt to look up a directory with inode number 14 and such an inode number does not exist.

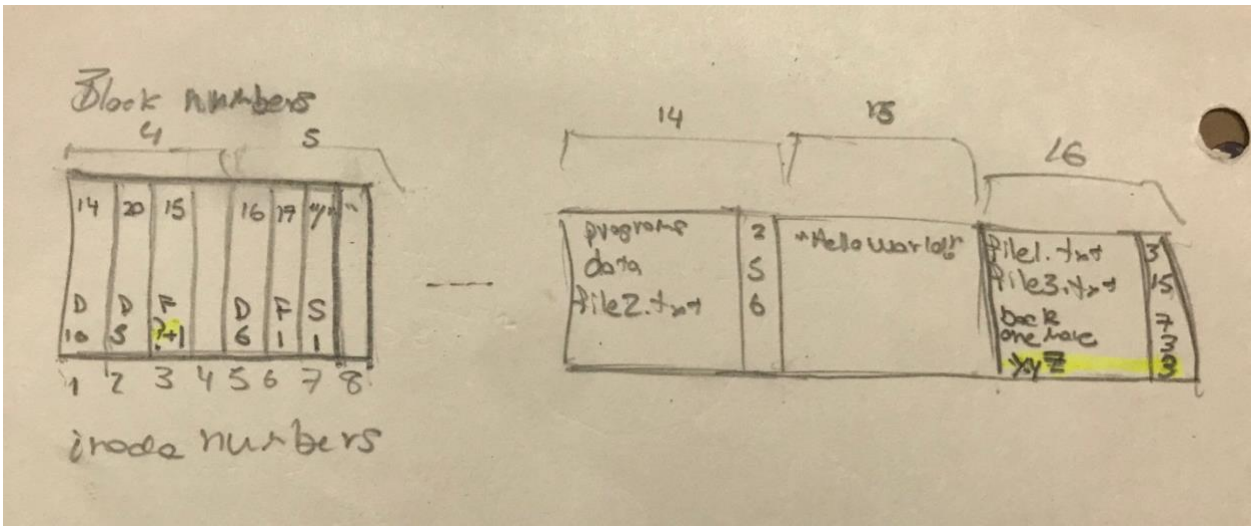
b. What is returned from LOOKUP("file1.txt",5)?

3, this function returns the inode number of the file that is within the directory corresponding to the inode number dir (5 in this example).

c. What is returned from LOOKUP("programs",5)?

FAILURE, there is no such inode named programs referenced within the directory that corresponds to inode number 5.

iii) Suppose you create a new hard link "xyz" in directory "/data" that links to "/data/file1.txt". Describe (in words, or use a drawing based on the figure above) the new state of all blocks that may have changed due to this operation.



iv) Assume the sequence of events for two different processes (A and B) running in the same computer where this file system is mounted:

Time T1: A issues fdA=open("/data/file1.txt"), and obtains fdA=3

Time T2: B issues fdB=open("/data/file1.txt") and obtains fdB=3

Time T3: thread A issues read(fdA,bufA,n=5)

Time T4: thread A issues close(fdA)

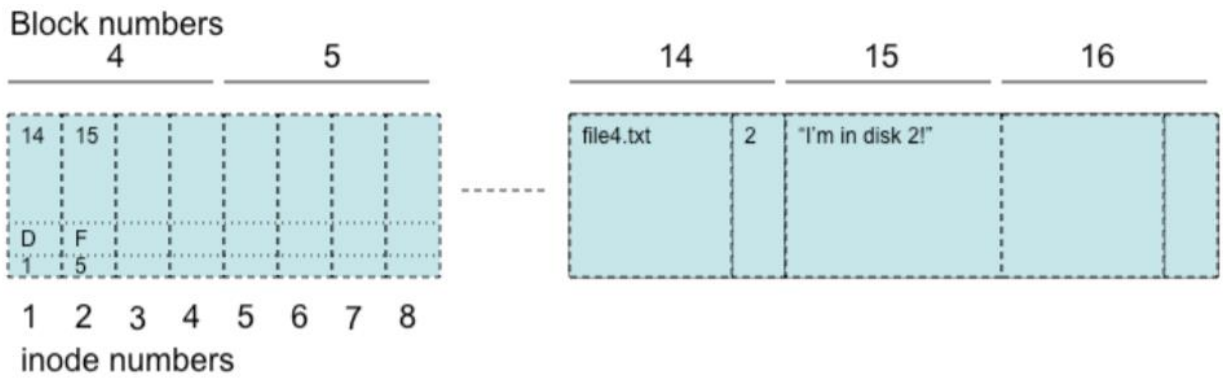
Time T5: thread B issues read(fdB,bufB,n=5)

Where fdA, fdB are file descriptors, bufA and bufB are memory buffers, and n is the number of characters to read. What values (if any) are returned in the buffer bufA of thread A at time T3, and bufB in thread B at time T5?

fdA and fdB are synonyms, as both point to the inode number 3. Both are in different threads, but within the same process. When you make a syscall to read at T3 in thread A, "Hello" is returned as we asked for the first 5 characters of the file. Then we make a call to close fdA at T4 in thread A.

At T5, we make a call to read fdB (inode number 3). This will result in a failure – file has been closed in thread A. There is also the possibility that the open call failed at T2, depending on the implementation of the OS. Some OSs do not allow for files being open concurrently to ensure atomicity.

v) Suppose now that the system has a second hard disk, with a second file system as depicted in the figure below. Suppose you mount this second file system under directory "/mnt" of the root file system. Is it possible to create a hard link "file4.txt" under "/data" that links to "file4.txt" in the second file system? Is it possible to create a symbolic link "file4.txt" under "/data" that links to "file4.txt" in the second file system? Describe (in words, or use a drawing based on the figure above) the new state of all blocks that may have changed due to these operations, if they are possible.



Is it possible to create a hard link “file4.txt” under “/data” that links to “file4.txt” in the second file system?

No, it is not possible to create a hard link as you do not have access to the inode numbers in the new drive from the old drive. Hard links cannot cross across drives.

Is it possible to create a symbolic link “file4.txt” under “/data” that links to “file4.txt” in the second file system?

Yes, we can create a symlink to cross the boundary between drives.

Describe (in words, or use a drawing based on the figure above) the new state of all blocks that may have changed due to these operations, if they are possible.

Add a new entry in block number 16 called “file4.txt” in the first drive that references a new symlink inode (perhaps number 8 in the first block drive) that points to “/mnt/file4.txt”