HOMEWORK 4
SOLUTIONS

CSE 120 OPERATING SYSTEMS

FALL 2006

NB: Questions are in boldface and the solutions follow them.

1) Explain briefly how NFS attempts to keep the server 'stateless.' List 2 disadvantages of using such a design.
[1+2 points]

Ans:

NFS implements a stateless DFS. It assumes that a client request for a file read or write would not have occurred unless the file system had been remotely mounted and the file had been previously open. The NFS protocol carries all the information needed to locate the appropriate file and perform the requested operation.
Disadvantages of being stateless:

1) Server doesn't keep track of concurrent access to same file
2) Multiple clients might be modifying a file at the same time. NFS does not provide any consistency guarantees.
3) Heavy load on the server.
4) Incur round trip on every open.
5) Performance issues.

2) [Silberschatz Galvin Gagne – 7th Edition – Chapter 10 – Problem 10.1] Consider a file system where a file can be deleted and its disk space reclaimed while links to that file still exist. What problems may occur if a new file is created in the same storage area or with the same absolute path name? How can these problems be avoided?
[2+1 points]

Ans:

Problems:
- Dangling pointers
- Illegal references to other existing files
Solutions:
- Backpointers solve the dangling pointer problem.
- If the file entry itself is deleted, we can search for the dangling links and remove them, but unless a list of the associated links is kept with each file, this search can be

expensive. Alternatively, we can leave the links until an attempt is made to use them. At that time, we can determine that the file of the name given by the link does not exist and the access is treated just as with any other illegal file name.
- Another approach is to preserve the file until all references to it are deleted,

3) Why might a file system that uses linked allocation perform poorly for applications that require direct access?
[2 points]

Ans:

To find the ith block of a file, we must start at the beginning of that file and follow the pointers until we get to the ith block. Each access to a pointer requires a disk read, and some require a disk seek. Consequently, it is inefficient to support a direct-access capability for linked-allocation files.

4) Consider a UNIX-style i-node with 10 direct pointers, one single-indirect pointer, and one double-indirect pointer only. Assume that the block size is 8 Kbytes, and that the size of a pointer is 4 bytes. How large a file can be indexed using such an i-node?
[4 points]

Ans:

10 direct pointers. Block size = 8KB. Thus, 80 KB can be directly accessed.
1 single-indirect pointer. Pointer size is 4 bytes. Block size is 8KB. Thus, there can be 2048 4-byte pointers in an 8KB index block. Thus, with the single-indirect pointer, (2048 x 8 x 1024) = 16 MB can be directly addressed.
1 double-indirect pointer. Pointer size is 4 bytes. Block size is 8KB. Thus, there can be 2048 4-byte pointers in an 8KB index block. Two levels of index allow (2048 x 2048) data blocks and hence (2048 x 2048 x 8 x 1024) = 32 GB can be directly accessed.
Thus, with all the index schemes,
The total allowable file size would be = 8KB + 16MB + 32GB ≈ 32GB

5) Give a scenario where choosing a large file system block size might be a benefit; and give an example where it might be a hindrance. What are the tradeoffs associated in choosing large or small block sizes?
[2 points]

Ans:

If a system administrator knows the file system is going to be used to store large files it would make sense to use the largest possible logical block size, thereby reducing external fragmentation. Conversely, if the file system is used for small files, a small block size makes sense, and helps to reduce internal fragmentation.

6) Why might file systems managing external storage devices do write-through caching (avoid buffering writes) even though there is a detrimental affect on performance?
[2 points]

Ans:

When data is written to the cache, it must at some point be written to main memory as well. The timing of this write is controlled by what is known as the write policy. In a write-through cache, every write to the cache causes a write to main memory. Alternatively, in a write-back cache, writes are not immediately mirrored to memory. Instead, the cache tracks which locations have been written over (these locations are marked dirty). The data in these locations is written back to main memory when that data is evicted from the cache. For this reason, a miss in a write-back cache will often require two memory accesses to service: one to read the new location from memory and the other to write the dirty location to memory.

7) Log-structured file systems have a data structure called an 'i-node map' – which is used to determine the location of each i-node on the disk. Regular (non-log-structured) UNIX file systems do not. What is the reason for this distinction?
[2 points]

Ans:

Unlike UFS, inodes in LFS do not have fixed locations. An inode map – a flat list of inode block locations – is used to track them. As with everything else, inode map blocks are also written to the log when they are changed. An inode is a data structure on a traditional Unix-style file system. An inode stores basic information about a regular file, directory, or other file system object. When a file system is created, data structures that contain information about files are created. Each file has an inode and is identified by an inode number (i-number) in the file system where it resides. inodes store information on files such as user and group ownership, access mode (read, write, execute permissions) and type of file. There is a fixed number of inodes, which indicates the maximum number of files each file system can hold.

8) The kernel typically maintains a 'file-open count' along with other pieces of data associated with an open file. The 'file-open count' is simply a counter indicating the number of times a particular file is open. Why do we need to store this information?
[2 points]

Ans:

As files are closed, the operating system must reuse its open-file table entries, or it could run out of space in the table. Because multiple processes may have opened a file, the system must wait for the last file to close before removing the open-file table entry. The file-open counter tracks the number of opens and closes and reaches zero on the last close. The system can then remove the entry. Also, if a file is deleted but is still open, it is not removed from disk until all references have closed it.

9) Consider a disk with a sector size of 512 bytes, 100 sectors per track. Given a rotational speed of 7200 revolutions per second, what is the maximum rotational delay to the start of a sector? Assuming that one track of data can be transferred per revolution, what is the transfer rate?
[1+1 points]

Ans:

- Maximum rotational delay:1/7200 sec
- Transfer rate: 512*100*7200 bytes/sec.

10) Assuming a normal UNIX i-node file system, what are the fewest number of disk blocks that must be read from disk to access the 1048th byte of the file /foo/bar/baz? Assume that the i-node for the root file system is already cached, but the disk buffer cache is otherwise empty. Consider a UNIX-style i-node with 12 direct pointers, one single-indirect pointer, and one double-indirect pointer. Assume that the block size is 2 Kbytes, and that the size of a pointer is 4 bytes.
[3 points]

Ans:

Total number of blocks read = Number of directory blocks read + INode blocks read for each of the directories + INode block read for the file + Index block (if any) to access ith byte of file + Block containing data

Number of directory blocks read = 1 for '/' (root) + 1 for 'foo' + 1 for 'bar' =3
Now, it is given that the inode for the root directory is already cached.
INode blocks read for each of the other directories = 1 + 1 = 2
INode block read for the file "baz" = 1

Since the block size is 2KB and there are 12 direct pointers, the first 24Kbytes of the file can be accessed directly. The 1048th byte would fall in the second block which would be accessed directly using the second direct pointer.

Hence, there is no need to access the index block.

Hence, total number of disk blocks read = 3+2+1+0+1 = 7