CS 390
Chapter 12 Homework Solutions

12.1 `Consider a file currently consisting ...`

	contiguous	linked	indexed
a.	100 reads, 101 writes	1 write	2 writes
b.	(100 - m) reads, (101 - m) writes	(100 - m) reads, 2 writes	2 writes
c.	1 write	100 reads, 2 writes	2 writes
d.	no operations	no operations	1 write
e.	(99 - m) reads, (99 - m) writes	(99 - m) reads, 2 writes	1 write
f.	no operations	99 reads, 1 write	1 write

12.3 `Why must the bit map ...`

The bit map may be many MB in size. For example, a 1TB disk with 512 byte blocks will have 2^{31} blocks. At 8 bits / byte it will require a 256MB bit map. This bit map will be accessed relatively infrequently, and only a single bit will be accessed at once. It is a wasteful to keep the entire bitmap in memory.

The bitmap must maintain its contents between reboots, so it must be stored on secondary storage. The logical place to store the bitmap is on the disk that is being mapped.

While parts of the bitmap must be placed in memory to be manipulated by the kernel, we don't want to keep it there for long. A crash could corrupt the bitmap if it is not current on disk, corrupting the file system free-block list.

12.9 `Consider a file system that ...`

a. `All extents are of the same size, and the size is predetermined.`

Advantages: No external fragmentation, since we can always reuse the space occupied by a deleted extent. With this scheme, the number of extents is a function of the number of blocks in the file, rather than a product of the current fragmentation state of the disk.

Disadvantages: Internal fragmentation, which will be severe if extents are large. If extents are the same size as a disk block, this system degenerates into an indexed scheme.

b. `Extents can be of any size and are allocated dynamically.`

Advantages: Little internal fragmentation, as only the last extent of a file will contain a block that is not completely filled. Some files (specifically, the first ones created on the disk) will be stored contiguously, so access to them will be fast.

Disadvantages: After some time, the free space on the disk will become fragmented. In the worst case, each free block will be in its own extent, and reading a file will require a seek for each block, making access slow.

c. `Extents can be of a few fixed sizes, and these sizes are predetermined.`

This is similar to the "buddy system" of dynamic memory allocation. (See section 9.8.1.)

Advantages: As long as files rarely exceed the size of the largest extent, this allocation scheme is fast because the file system will attempt to place files into a single extent that is just large enough to hold it.

Disadvantages: Internal fragmentation will occur, since files will rarely fit exactly into an extent. Since the number and sizes of extents are fixed, the system may eventually run short of smaller extents, resulting in lots of internal fragmentation when large extents are used for small files. On the other hand, the system could run short of large extents, reducing internal fragmentation, but causing files to be scattered across the disk.

12.12 `Consider a system where free space is kept in a free space list.`

Let's assume in this question that the free-space list is a bitmap of free and used blocks stored in the disk. The pointer to this list would be kept in the volume control block.

a. `Suppose that the pointer to the free-space list is lost. Can the system reconstruct the free-space list? Explain your answer.`

Yes, but the amount of time required will grow linearly with the number of files in the file system. To reconstruct the free-space

list, the OS must scan the file system beginning at the root. As each directory and file is scanned, the OS marks the associated disk blocks as used. (Notice that this includes not only file data, but any file-control blocks and index blocks associated with the file, as well as blocks (both data and meta-data) used by directories. In addition, blocks occupied by volume- and boot-control blocks must be accounted for.)

One problem with this scheme is that some OS's deal with damaged (unreadable and/or unwritable) disk blocks simply by removing them from the free list. The scanning technique described above would identify these blocks as free, since they are not part of the file system structure. The danger here is that these blocks might be allocated to a file, even though we know they are bad. Thus, the OS would need to keep a separate list of these blocks on disk. Early versions of Unix solved this problem by allocating these bad blocks to a special file whose only purpose was to be composed of bad blocks, and placing this file in a special directory. This directory ("/lost+found") still exists on some systems.

b. `Consider a file system similar to the one used by Unix with ...`

Remember that in Unix, directories are just regular files that contain a list of file/directory names and pointers to FCB's (inodes). In the best case, reading each directory or file requires three disk block accesses: one to read the inode for the file or directory, one to read the index block pointed to by the inode, and then one to read the first data block of file or directory itself.

If we assume we know the address of the root inode when we begin, then 12 reads are required:

1. read the root inode,
2. read the index block of the root directory,
3. read the root directory,
4. read `a`'s inode,
5. read `a`'s index block,
6. read directory `a`,
7. read `b`'s inode,

8. read `b`'s index block,
9. read directory `b`,
10. read `c`'s inode,
11. read `c`'s index block,
12. read first block of file `c`.

If we assume a double-indirect block scheme, than in the worse case, four reads are required to get the appropriate block of a directory or file into memory: read the inode, read the double indirect block, read the indirect block, read the data block. This gives us a total of 16 reads.

c. `Suggest a scheme to ensure that the pointer is never lost as a result of memory failure.`

Store the pointer to the free space list in several different locations on disk.

12.14 `Discuss how performance optimizations for file systems might result in difficulties in maintaining the consistency of the systems in the event of computer crashes.`

When parts of the file system are cached, blocks that are in the cache but which have not yet written to disk would be lost in a crash. Imagine that a new file is created consisting of one block. If the file-control block is written to disk, but the data block is not, the file-control block will point to a disk block of garbage after a crash.

Suppose that free space is managed using a bitmap. The system could allocate a new block to a file, write the file to disk, update the bitmap in memory, but then crash before the bitmap can be written to the disk. This scenario requires some type of consistency checker (e.g. "fsck" in Unix and "chkdsk" in NT.)

12.17 `Fragmentation on a storage device ...`

To compact disk blocks, we need to read each file and its associated FCB into memory, free up the blocks that the file occupies on disk, locate a contiguous section of free blocks large enough to hold the file data, rewrite the data blocks, and then update the FCB.

1. It is time consuming, since each change requires two block accesses - a read of the original block followed by a write to its new location. Suppose transferring a 1K block requires 0.1 microseconds. (That is, the disk has a transfer rate of 10 MB / sec.) Then processing a disk with a 100GB file system would require $2 \cdot \frac{10 \cdot 2^{30}}{2^{10}} \cdot 1\mu\text{sec} \approx 36$ minutes. This assumes that the compaction algorithm would only have to move a file once.
2. If a disk is nearly full, we may have to read many files into memory before we can free up enough contiguous space to rewrite even one of them. These files may not fit in memory, or they may have to be read and re-written several times.
3. If a crash occurs while the disk is being compacted, some data may be lost, since the file system is in an inconsistent state while a file is being moved. What would happen if the system crashes while the root directory of the file system or the file that contains the kernel is being relocated?

11.a `Suppose a computer system uses indexed allocation. The disk has 1KB blocks, and block addresses are 32 bits.`

1 `If a single direct block is used, what is the maximum file size that can be supported?`

We can fit $\frac{2^{10}}{4} = 256$ addresses into the direct block. Thus, the maximum file size is $256 \cdot 1\text{KB} = 256\text{KB}$.

2 `If a double indirect block scheme is used, what is the maximum file size that can be supported?`

The direct block can hold 256 block addresses. The indirect block can hold 256 direct block addresses, each of which holds 256 block addresses. The maximum file size is $(256 + (256 \cdot 256)) \cdot 1\text{KB} = 65792\text{KB} \approx 64\text{MB}$.

3 `Suppose the block size in part 2 is increased to 2KB. What is the maximum file size that can be supported?`

$\approx 513\text{MB}$

4 `Suppose we use 1KB blocks with a triple indirect block scheme. What is the maximum file size?`

$(256 + 256^2 + 256^3) * 1\text{KB} \approx 16\text{GB}$

5 What about a triple indirect block scheme with 8KB blocks?

$(2048 + 2048^2 + 2048^3) * 8\text{KB} \approx 64$ TB.