CS 390
Chapter 14 Homework Solutions

14.1 `Consider a file currently consisting ...`

Assume in this question that the FCB and index blocks, if any, are already in memory. Do not count the writes required to update the FCB, index blocks, or free list in your answer.

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

14.2 `Why must the bit map ...`

Anything stored in primary memory is lost when the machine shuts down or power is lost. If the file allocation bitmap was stored in primary memory, then after a reboot, the kernel would have no way of determining which blocks on secondary storage held file system data or metadata, and which were available to allocate.[1]

Note that while the bitmap's primary "home" must be the secondary storage device, in order to use the bitmap, we must read it into memory. A reliable kernel will want to quickly commit any changes of the bitmap to disk.

14.3 `Consider a system that supports ...`

You should think about three criteria to answer this question: What is known about the file when it is created, how will the file be read, and how will the file be written.

Contiguous allocation is the best for static files. These are files whose size is known when they are created, and will not change after the

[1] Actually, if the kernel can determine the location of the root or top level directory on the disk, then the list of free blocks can in fact be reconstructed by traversing the entire filesystem tree from the top down and noting which blocks are in use. This operation is done using the program "fsck" on Linux, and "chksdk" on Windows

initial write. Such files can be read efficiently using either sequential or direct access. Although these files can be re-written in place, they may be difficult or impossible to append.

Linked allocation is good for files where the maximum size is not known. These files can be read efficiently using sequential access, but not direct access. They can be appended efficiently, once the location of the last block is determined. Information could be inserted at the beginning or in the middle of the file, but only in chunks that are the same size as a block, and really, how often would this happen?

Indexed allocation is good for all types of files. Not much needs to be known at file creation time because file growth can be easily accommodated. It supports sequential and direct access, both for reading and writing. The only drawback of indexed allocation is that is carries slightly more overhead than the other two systems, because the system uses space to store the index.

14.5 `How do caches help improve ...`

Caches allow components of differing speeds to communicate more quickly by storing data from the slower device, temporarily, in a faster device (the cache). Since the speed of any computing operation is determined by the speed of the slowest component that is being accessed during the operation, caching improves the overall speed of the system.

Caches are, by definition, more expensive than the device they are caching for, so increasing the number of size of caches would increase the system cost.

A In this question, assume that the file system is stored on a hard-disk drive. Consider a system that uses modified contiguous allocation with extents. Under this system, a file is a collection of extents, with each extent corresponding to a contiguous set of disk blocks. The addresses of the extents which make up the file are stored in the FCB.

A key issue in such systems is the degree of variability in the size of the extents. What are the advantages and disadvantages of the following schemes?

a. All the extents consist of the same number of disk blocks, and this size is fixed when the disk is formatted.

b. Extents can be any size as long as that size is some multiple of the disk block size.

c. Extents can be a few fixed sizes, each of which a multiple of the disk block size, and these sizes are determined when the disk is formatted.

B Consider a system where the free space on a disk is kept in a free space list.

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

b. Suggest a scheme to make sure that the free space list pointer is never lost as the result of disk block corruption or system crash.

C Fragmentation on a storage device can be eliminated by recompaction. However, typical storage devices do not have base or relocation registers like the memory management hardware of the CPU does. Given this, how can files be relocated? Give three reasons why file compaction is generally avoided.

D Suppose a computer system uses indexed allocation. The storage device has 1KB blocks, and block addresses are 32 bits. No data block addresses are stored in the FCB.

1. If a single index block is used, what is the maximum file size that can be supported?
2. If a three-level indexing scheme (index block, indirect-index block, and doubly-indirect index block) is used, what is the maximum file size that can be supported?
3. Suppose the block size in increased to 2KB. What is the maximum file size that can be supported with these two schemes?

A **a.** Advantages: No external fragmentation, since we can always reuse the space occupied by a deleted extent.

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 a simple indexed scheme.

b. Advantages: Little internal fragmentation, as only the last extent of a file will not be 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. This is similar to the "buddy system" of dynamic memory allocation. (See section 10.8.1.)

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 could eventually run short of smaller extents. If it does, there will be lots of internal fragmentation when large extents are used for small files. On the other hand, the system could run short of large extents. If it does, large files will be stored in a large number of small extents, reducing internal fragmentation, but causing files to be scattered across the disk.

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

a. 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 of the file system tree. (The location of the root directory is stored in the volume control block.) As each directory and file is scanned, the OS marks the associated disk blocks as used. 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 mark these blocks as free, since they are not part of the file system structure. The danger here is that these blocks might later be allocated to a file, even though we know they are bad. Thus, the OS would need to keep a separate list of bad 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. This file was placed in a special directory. This directory ("/lost+found") still exists on Linux systems.

b. Keep copies of the pointer to the free space list in several places on the disk. In Linux, this pointer is stored in the volume control block (called the "superblock" in Linux). Multiple copies of the superblock are stored on the disk in physically separate locations.

C Memory management hardware requires base and/or relocation registers, because address translation must be done in hardware. It would slow the system down to an unacceptable level if we did address translation in software, because every memory reference needs to be translated.

When we access data on a storage device, however, the address translation (from logical offset in a file to physical location on disk) only needs to be done once, when the file is transferred between the device and memory. The time required to do this translation is minuscule compared to the time required to move the file between memory and the device, so we don't mind doing it in software.

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 and rewrite 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 is being moved? What if the crash happens when the file that contains the kernel executable is being moved?

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

2. The index block can hold 256 block addresses. The indirect index block can hold 256 index block addresses, each of which holds 256 data block addresses. The double-indirect index block holds pointers to 256 indirect index blocks. The maximum file size is $(256 + 256^2 + 256^3) \cdot 1\text{KB} \approx 16\text{GB}$.

3. $\approx 1\text{MB}$, and $\approx 256\text{GB}$, respectively.