CHAPTER 11

In this chapter we discuss various methods for storing information on secondary storage. The basic issues are device directory, free space management, and space allocation on a disk.

Exercises

11.1 Consider a file system that uses a modified contiguous-allocation scheme with support for extents. A file is a collection of extents, with each extent corresponding to a contiguous set of blocks. 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 extents are of the same size, and the size is predetermined.
b. Extents can be of any size and are allocated dynamically.
c. Extents can be of a few fixed sizes, and these sizes are predetermined.

Answer:
If all extents are of the same size, and the size is predetermined, then it simplifies the block allocation scheme. A simple bit map or free list for extents would suffice. If the extents can be of any size and are allocated dynamically, then more complex allocation schemes are required. It might be difficult to find an extent of the appropriate size and there might be external fragmentation. One could use the Buddy system allocator discussed in the previous chapters to design an appropriate allocator. When the extents can be of a few fixed sizes, and these sizes are predetermined, one would have to maintain a separate bitmap or free list for each possible size. This scheme is of intermediate complexity and of intermediate flexibility in comparison to the earlier schemes.

11.2 Contrast the performance of the three techniques for allocating disk blocks (contiguous, linked, and indexed) for both sequential and random file access.
Answer:

- Contiguous Sequential - Works very well as the file is stored contiguously.
- Sequential access - Simply involves traversing the contiguous disk blocks.
- Contiguous Random -Works very well as you can easily determine the adjacent disk block containing the position you wish to seek to.
- Linked Sequential - Satisfactory as you are simply following the links from one block to the next.
- Linked Random - Poor as it may require following the links to several disk blocks until you arrive at the intended seek point of the file.
- Indexed Sequential - Works well as sequential access simply involves sequentially accessing each index.
- Indexed Random - Works well as it is easy to determine the index associated with the disk block containing the position you wish to seek to

11.3 What are the advantages of the variation of linked allocation that uses a FAT to chain together the blocks of a file?
Answer:
The advantage is that while accessing a block that is stored at the middle of a file, its location can be determined by chasing the pointers stored in the FAT as opposed to accessing all of the individual blocks of the file in a sequential manner to find the pointer to the target block. Typically, most of the FAT can be cached in memory and therefore the pointers can be determined with just memory accesses instead of having to access the disk blocks.

11.4 Consider a system where free space is kept in a free-space list.

a. Suppose that the pointer to the free-space list is lost. Can the system reconstruct the free-space list? Explain your answer.
b. Consider a file system similar to the one used by UNIX with indexed allocation. How many disk I/O operations might be required to read the contents of a small local file at */a/b/c*? Assume that none of the disk blocks is currently being cached.
c. Suggest a scheme to ensure that the pointer is never lost as a result of memory failure.

Answer:

a. In order to reconstruct the free list, it would be necessary to perform "garbage collection." This would entail searching the entire directory structure to determine which pages are already allocated to jobs. Those remaining unallocated pages could be relinked as the free-space list.
b. Reading the contents of the small local file */a/b/c* involves 4 separate disk operations: (1) Reading in the disk block containing the root directory /, (2) & (3) reading in the disk block containing the directories *b* and *c*, and reading in the disk block containing the file *c*.
c. The free-space list pointer could be stored on the disk, perhaps in several places.

11.5 Some file systems allow disk storage to be allocated at different levels of granularity. For instance, a file system could allocate 4 KB of disk space as a single 4-KB block or as eight 512-byte blocks. How could we take advantage of this flexibility to improve performance? What modifications would have to be made to the free-space management scheme in order to support this feature?

Answer:

Such a scheme would decrease internal fragmentation. If a file is 5 KB, then it could be allocated a 4 KB block and two contiguous 512-byte blocks. In addition to maintaining a bitmap of free blocks, one would also have to maintain extra state regarding which of the subblocks are currently being used inside a block. The allocator would then have to examine this extra state to allocate subblocks and coalesce the subblocks to obtain the larger block when all of the subblocks become free.

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

Answer:

The primary difficulty that might arise is due to delayed updates of data and metadata. Updates could be delayed in the hope that the same data might be updated in the future or that the updated data might be temporary and might be deleted in the near future. However, if the system were to crash without having committed the delayed updates, then the consistency of the file system is destroyed.

11.7 Consider a file system on a disk that has both logical and physical block sizes of 512 bytes. Assume that the information about each file is already in memory. For each of the three allocation strategies (contiguous, linked, and indexed), answer these questions:

a. How is the logical-to-physical address mapping accomplished in this system? (For the indexed allocation, assume that a file is always less than 512 blocks long.)
b. If we are currently at logical block 10 (the last block accessed was block 10) and want to access logical block 4, how many physical blocks must be read from the disk?

Answer:

Let *Z* be the starting file address (block number).

- **Contiguous**. Divide the logical address by 512 with *X* and *Y* the resulting quotient and remainder respectively.
    a. Add *X* to *Z* to obtain the physical block number. *Y* is the displacement into that block.
    b. 1
- **Linked**. Divide the logical physical address by 511 with *X* and *Y* the resulting quotient and remainder respectively.

a. Chase down the linked list (getting *X* + 1 blocks). *Y* + 1 is the displacement into the last physical block.
b. 4

- **Indexed**. Divide the logical address by 512 with *X* and *Y* the resulting quotient and remainder respectively.
  a. Get the index block into memory. Physical block address is contained in the index block at location *X*. *Y* is the displacement into the desired physical block.
  b. 2

11.8 Consider a file system that uses inodes to represent files. Disk blocks are 8-KB in size and a pointer to a disk block requires 4 bytes. This file system has 12 direct disk blocks, plus single, double, and triple indirect disk blocks. What is the maximum size of a file that can be stored in this file system?
Answer:
(12 * 8 /KB/) + (2048 * 8 /KB) + (2048 * 2048 * 8 /KB/) + (2048 * 2048 * 2048 * 8 /KB) = 64 terabytes

11.9 Fragmentation on a storage device could be eliminated by recompaction of the information. Typical disk devices do not have relocation or base registers (such as are used when memory is to be compacted), so how can we relocate files? Give three reasons why recompacting and relocation of files often are avoided.
Answer:
Relocation of files on secondary storage involves considerable overhead—data blocks have to be read into main memory and written back out to their new locations. Furthermore, relocation registers apply only to *sequential* files, and many disk files are not sequential. For this same reason, many new files will not require contiguous disk space; even sequential files can be allocated noncontiguous blocks if links between logically sequential blocks are maintained by the disk system.

11.10 Assume that in a particular augmentation of a remote-file-access protocol, each client maintains a name cache that caches translations from file names to corresponding file handles. What issues should we take into account in implementing the name cache?
Answer:
One issue is maintaining consistency of the name cache. If the cache entry becomes inconsistent, then either it should be updated or its inconsistency should be detected when it is used next. If the inconsistency is detected later, then there should be a fallback mechanism for determining the new translation for the name. Also, another related issue is whether a name lookup is performed one element at a time for each subdirectory in the pathname or whether it is performed in a single shot at the server. If it is performed one element at a time, then the client might obtain more information regarding the translations for all of the intermediate directories. On the other hand, it increases the network traffic as a single name lookup causes a sequence of partial name lookups.

11.11 Explain why logging metadata updates ensures recovery of a file system after a file system crash.
Answer:
For a file system to be recoverable after a crash, it must be consistent or must be able to be made consistent. Therefore, we have to prove that logging metadata updates keeps the file system in a consistent or able-to-be-consistent state. For a file system to become inconsistent, the metadata must be written incompletely or in the wrong order to the file system data structures. With metadata logging, the writes are made to a sequential log. The complete transaction is written there before it is moved to the file system structures. If the system crashes during file system data updates, the updates can be completed based on the information in the log. Thus, logging ensures that file system changes are made completely (either before or after a crash). The order of the changes is guaranteed to be correct because of the sequential writes to the log. If a change was made incompletely to the log, it is discarded, with no changes made to the file system structures. Therefore, the structures are either consistent or can be trivially made consistent via metadata logging replay.

11.12 Consider the following backup scheme:

- **Day 1**. Copy to a backup medium all files from the disk.
- **Day 2**. Copy to another medium all files changed since day 1.
- **Day 3**. Copy to another medium all files changed since day 1.

This differs from the schedule given in Section 11.7.4 by having all subsequent backups copy all files modified since the first full backup. What are the benefits of this system over the one in Section 11.7.4? What are the drawbacks? Are restore operations made easier or more difficult? Explain your answer.

Answer:

Restores are easier because you can go to the last backup tape, rather than the full tape. No intermediate tapes need be read. More tape is used as more files change.