File Management

Madiba Hudson-Quansah

Contents

Chapter 1

Introduction

- The efficiency of the file manager is directly affected by:
  - File Organization
  - File Storage
  - File Record Structure
  - User Access Control
- The responsibilities of the file manager include:

  File Storage Tracking -
    - Keep track of used and free space
    - Allocate space when files are created

  Policy Implemetation -
    - Determine where and how files are stored
    - efficiently use available storage space
    - Provide efficient file access

  File Allocation if user access cleared, i.e. recording file use

  File Deallocation -
    - Return file to storage
    - Communicate file availability
- Policy determines -
  - File storage location
  - System and user access, using device independent commands
- Access to material depends on two factors -

  Flexibility of access to information -
    - Share files
    - Provide distributed access
    - Allow users to browse public directories

  Subsequent Protection -
    - Prevent system malfunctions

    - Security checks such as account numbers and passwords

- File allocation -
    1. Activate secondary storage device
    2. Load file into memory
    3. Update records
- File deallocation -
    - Update file tables
    - Rewrite file if revised
    - Notify waiting processes of file availability

Definition 1.0.1: Field

A group of related bytes, identified by user, e.g., name, type, size

Definition 1.0.2: Record

A group of related fields

Definition 1.0.3: File

A group of related records, used by specific applications.

Definition 1.0.4: Flat File

A file that is not connected in any way to other files, i.e. no dimensionality.

Definition 1.0.5: Database

A group of related files, that are interconnected at various levels, allowing users to access related stored data.

Definition 1.0.6: Program File

A file that contains executable code / instructions.

Definition 1.0.7: Data File

A file that contains data to be used by programs.

Definition 1.0.8: Directory / Folder

Listings of files and their attributes.

Definition 1.0.9: Attribute

Metadata about a file, e.g., name, type, location, size, date created, date modified, access permissions.k

The file manager is the software responsible for creating deleting, modifying, controlling access to files, and managing resources used by files. It provides support for libraries of programs and data, doing so in close collaboration with the Device Manager.

The file manager performs the following tasks:

- Keeps track of where each file is stored.
- Uses a policy that will determine where and how the files will be stored, making sure to efficiently use the available storage space and provide easy access to the files.
- Allocate each file when a user has been cleared for access to it and then track its use.
- Deallocate the file when the file is to be returned to storage and communicate its availability to others that may be waiting to use it.

The file manager's policy determines where each file is stored and how the system and its suers will be able to access them, via device independent commands. The policy also determines who will have access to what material, which involves two factors:

- Flexibility of access to the information
- Protection against subsequent unauthorized access

The file manager handles this by allow access to shared files, providing distributed access, and allowing users to browser through public directories. Meanwhile the operating system must protect its files against system malfunctions and provide security checks via account numbers and passwords to preserve the integrity of the data and safeguard against tampering.

The computer system allocates a file by activating the appropriate secondary storage devices and loading the file into memory, while updating the records of who is using what file. And deallocates a file by updating the file tables and rewriting the file, if modified, to the secondary storage device, and notifying any waiting processes of the file's availability.

Chapter 2

Interacting with the File Manager

- Device Independent Commands, i.e. doesn't need to know the specifics of the device.
- Logical Commands -
  - Broken into lower-level signals
- System monitors for error conditions

The file manager responds to specific propriety commands to perform necessary actions, with the most common being:

- OPEN
- DELETE
- RENAME
- COPY
- CREATE

2.1 Typical Volume Configuration

- Secondary storage unit
- Multi-file volumes may contain many files
- Multi-volume files are large files that span several volumes
- Volumes are identified by volume names, stored in the volume descriptor
- Master File Directory (MFD) - Is stored immediately after the volume descriptor and contains:

  Lists -
    - Names and characteristic of every file on the volume
    - Subdirectories

2.2 Subdirectories

- File managers create MFD for each volume. The MFD contains file and subdirectory entries
- Subdirectories
  - Created upon account opening
  - Treated as files by the file manager, but flagged in MFD as a subdirectory
- File Descriptors in subdirectories contain:

- File name
- File type
- File size
- File location
- Date and time of creation
- Owner
- Protection information
- Record size

Chapter 3

File Organization

File organization refers to the arrangement of records within a file, when a user give a command to modify the contents of a file, it's really a command to access records within the file.

Definition 3.0.1: Sequential File

A file in which records are stored one after the other and only have a coherent meaning when considered as a whole. For example an image file.

Definition 3.0.2: Direct Access File

A file in which records can be accessed in any order, with each record having a coherent meaning by itself. For example a database file.

3.1 Record Format

Within each file the records are all presumed to have the same format, either fixed-length or variable length. These records regardless of their format can be accessible individually or grouped in to blocks, with variable length fields taking longer to access:

Fixed-length Records -

- Easiest to access
- If the configured record length is too small to hold the entire record, then the leftover data is truncated, thereby corrupting the data.
- If the record length is too large, i.e. larger than the actual record, then the leftover space is wasted.

Variable-length Records -

- Don't leave empty storage space and don't truncate data
- Can easily be accessed sequentially but take longer to access directly because it is not easy to calculate where each record is located.
- Most frequently used in files that are likely to be accessed sequentially.

The record format, how it's blocked and other related information are kept in the file descriptor. The amount of space that's actually used to store supplementary information varies from system to system and comforts to the physical limitations of the storage medium.

3.2 Physical File Organization

Physical file organization refers to how records are arranged and the characteristics of the medium used to store them. On magnetic disks, files can be organized in one of several ways, i.e.:

- Sequential
- Direct
- Indexed Sequential

To select the best of these file organizations the system designer must consider these practical characteristics:

Volatility of the data - The frequency with which additions and deletions are made

Activity of the file - The percentage of records accessed during a given run

Size of the file - The number of records in the file

Response time - The amount of time the user is willing to wait before the requested operation is completed, which is especially crucial when doing time-sensitive searches.

3.2.1 Sequential Record Organization

The easiest file organization to implement because the variable length records are stored and retrieved serially, one after the other. To find a specific record the file manager can search the file from it's beginning until the requested record is found.

To optimize the search process a key field can be selected from the record and then the records can be sorted by that field before storing them. Later, when a user requests a specific record, the system searches only the key field of each record in the file. The search is ended when either an exact match is found, or the key field for the requested record is smaller than the value of the record of the last compared.

Although this optimization aids the search process, it complicates file maintenance because the original order must be preserved every time records are added or deleted. And to preserve the physical order the file must be completely rewritten or maintained in a sorted fashion every time it is updated.

3.2.2 Direct Record / Random Organization

This file organization uses direct access files, which can only be implemented on Direct Access Storage Devices (DASDs). Theses files allow users to access any record in any order without having to begin a search from the beginning of the file.

To implement direct access a field or combination of fields in the record format is designated as the key field. The program used to store the data follows a set of instructions called a hashing algorithm, which transforms each key into a number, i.e. the record's logical address. This is given to the file manager, which takes the necessary steps to translate the logical address into a physical address, .e. Cylinder, Surface, and record number, preserving the file organization. The same process is used to retrieve records.

Direct access files can also be accessed sequentially by starting at a relative address and going to each record down the line. Direct access files can be updated quicker than sequential files because records can quickly be rewritten to their original addresses after modifications have been made, and there is no need to preserve the order of the records, so adding or deleting them takes very little time.

The problem with hashing algorithm is that several records which unique keys, may generate the same logical address, known as a hash collision. To resolve this the program must generate another logical address before presenting it to the file manager for storage. Records that collide are stored in an overflow area that was set aside when the file was created. Although the program does all the work of linking the records from the overflow area to their corresponding logical address, the file manager must handle the physical allocation of space of the overflow area.

The maximum size of the file is determined when it's created, and if either the file fills to its maximum capacity, or the number of records in the overflow area becomes to large, then retrieval efficiency is lost. When this happens the file must be reorganized and rewritten by the file manager.

3.2.3 Indexed Sequential Record Organization

This file organization combines the advantages of both sequential and direct access. It's created and maintained using an Indexed Sequential Access Method (ISAM) application, which handles handling overflows and preserving the record order.

This type of organization doesn't use the result of the hashing algorithm to generate a record's address, instead it uses it to generate an index file, through which records are retrieved, preventing hash collisions. This organization divides an ordered sequential file into blocks of equal size, with the size determined by the file manager to take advantage of physical storage devices and to optimize retrieval strategies. Each entry in the index file contains the highest record key and the physical location of the data block where this record and the records with smaller keys are stored.

Therefore to access any record in a file the system goes through this process:

1. The index file is searched sequentially to find the block containing the requested record.
2. The physical location of the block is sent to the file manager, which retrieves it from secondary storage.

Here the index file acts like a pointer to the data file, allowing quick access to any record in the file.

An indexed sequential file also has overflow areas, but they're spread throughout the file, so that existing records can expand, and new records can be located in close physical sequence as well as in logical sequence. Another overflow area is located apart from the main data area, but is used only when the other overflow areas are completely filled, called the overflow of last resort.

Definition 3.2.1: Overflow Of Last Resort

An overflow area located apart from the main data area, used only when the other overflow areas are completely filled. It can store records added during the lifetime of the file, where records are kept in logical order by the software without much effort on the part of the programmer. However, accessing records in this area is slow because it requires searching through the entire overflow area so if nay records end up here, it can lead to a degradation of performance.

When retrieval time becomes too slow the file must be reorganized which is usually handled by maintenance software.

Chapter 4

Physical Storage Allocation

The file manager must work with files not just as whole units, but also as logical units or records. By grouping individual records into blocks, some efficiency can be gained. Records within a file must have the same forma but they can vary in length.

In turn records are subdivided into fields, with their structure, in most cases, managed by application programs and not the operating system. An exception is made for those systems that are heavily oriented towards database applications, where the file manager handles field structure.

So when files are stored the only thing actually being handled by the file manager is the storage of records, i.e. files usually don't exist as physical constructs on secondary storage devices, only logically within the file manager.

4.0.1 Contiguous Storage

Storing records one after the other. This was the scheme used in early operating systems and is simple to implement and manage. This has the following advantages:

- Any record can be found and read once its starting address and size are known, making the directory streamlined.
- Direct access is easy because every part of the file is stored in the same place.

However, this scheme has several disadvantages:

- Files can't grow unless there's empty space available immediately following it. Therefore room for expansion must be provided when the file is created. If the file runs out of room then the entire file must be recopied to larger section of the disk every time records are added.
- Fragmentation can occur, which can be overcome by compaction and rearranging files, but files can't be accessed while compaction is happening.

The file manager keeps track of the empty storage areas by treating them as files, called free space files, flagged to differentiate them from regular files. Usually the directory is kept in order by sector number, so adjacent empty areas can be combined into one large free space.

4.0.2 Non-contiguous Storage

Allows files to be stored on any storage space available on the disk, which a file's records only being stored contiguously if there's enough empty space. Any remaining records, and all other additions to the file are stored in other sections of the disk, called the extents of the file and linked together via pointers. The physical size of each extent is determined by the operating system and is usually 256 bytes or some other power of 2 bytes.

File extents are usually linked in one of two ways:

Storage Level -

- Each extent points to the next one in the sequence.

- The directory entry consists of the filename, the storage location of the first extent, the location of the last extent, and the total number of extents not counting the first.
- By following the pointer from the first extent, the system can locate each of the linked extents until it reaches the last one.
- Each storage block holds its address, a pointer to the next file block, and the file data

Directory Level -

- Each extent is listed with its physical address, size, and a pointer to the next extent
- A null pointer indicates the last extent

Although both non-contiguous allocation schemes eliminate external storage fragmentation and the need for compaction, they don't support direct access because there's no easy way to determine the exact location of a specific record.

Files are usually declared to be either sequential or direct when they're created so that the file manager can select the efficient method of storage allocation, i.e contiguous for direct files, and non-contiguous for sequential files. Operating systems must have the capability to support both storage allocation schemes.

Files can then be converted from one type to another by creating a file of the desired type and copying the contents of the old file into the new file, using a program designed for this purpose.

4.1 Indexed Storage

Indexed storage allocation allows direct record access by bringing together the pointers linking every extent of that file into an index block. Each file has its own index block which consists of the addresses of each disk sector that make up the file. The index lists each entry in the same order in which the sectors are linked.

When a file is created the pointers in the index block are all set to null, then as each sector is filled the pointer is set to the appropriate sector address. This supports both sequential and direct access, but it doesn't necessarily improve the use of storage space, because each file must have its own index block, usually the size of one disk sector. For larger files with more entries several levels of indexes can be generated. In this case to find a desired record, the file manger accesses the first index, the highest level, which points to a second index, lower level, which points to an even lower level index and eventually to the data block containing the desired record.

Chapter 5

Access Methods

Access methods are determined by a file's organization, with the most flexibility allowed with indexed sequential files, and the least with sequential files.

A file that has been organized in sequential fashion can support sequential access to its records, and these records can be of either fixed or variable length. The file manager uses the address of the last byte read to access the next sequential record, therefore the **current byte address (CBA)** must be updated every time a record is accessed, such as when the READ command is executed.

5.1 Sequential Access

For sequential access of fixed-length records, the CBA is updated by simply by incrementing it by the record length (RL) which is constant:

$$\text{CBA} = \text{CBA} + \text{RL}$$

For sequential access of Variable-length records, the file manager adds the length of the record (RL) plus the number of bytes used to store the record length (N) to the CBA:

$$\text{CBA} = \text{CBA} + \text{N} + \text{RL}$$

5.2 Direct Access

If the file is organized in a direct fashion, and if the records are of a fixed length, it can be accessed in either direct or sequential order. In the case of direct access with fixed-length records, the CBA can be computed directed from the record length and the desired record number, RN minus 1, because the first record starts at byte 0:

$$\text{CBA} = (RN - 1) \times RL$$

For example, if we're looking for the beginning of the eleventh record, and the fixed record length is 25 bytes, the CBA would be:

$$\text{CBA} = (11 - 1) \times 25 = 250$$

However if the files is organized for direct access with variable-length records, it's virtually impossible to access a record directly because the address of the desired record can't be easily computed. Therefore to access a record the file manger must do a sequential search through the records, it becomes a half-sequential read through the file because the file manager could save the address of the last record accessed and when the next requests arrives, it could search forward from the CBA ,if the address of the desired record was between the CBA and the end of the file. Otherwise the search would start from the beginning of the file.

An alternative method is for the file manager to keep a table of record numbers and their CBAs, then to fill a request, this table is searched for the exact storage location of the desired record, the direct access reduces to a table lookup followed by a read operation.

To avoid this problem many systems force users to have their files organized for fixed-length records, if the records are to be accessed directly.

5.3 Indexed Sequential Access

Records in an indexed sequential file can be accessed either sequentially or directly, so either of the approaches to compute the CBA would work, but with one extra step, i.e. the index file must be searched for the pointer to the block where the data is stored. Because the index file smaller than the data file, it can be kept in main memory, and a quick search can be performed to locate the block where the desired record is located. Then the block can be retrieved from secondary storage, and the beginning byte address of the record can be calculated. In system that support several levels of indexing to improve access to very large files, the index at each level must be searched before the computation of the CBA can be done. The entry point to this type of data file is usually through the index file.

Chapter 6

Levels in a File Management System

Chapter 7

Access Control Verification Module

Chapter 8

Data Compression

Data compression algorithms consists of two types:

Lossless Compression Algorithms -

- Typically used for text/arithmetic files
- Retains all the data in the file throughout the compression-decompression process

Lossy Compression Algorithms -

- Used for image/audio/video files
- Permanently remove data without compromising the quality of the file.

8.1 Text Compression

There are many methods to compress text, with the most common being:

- Records with repeating characters
- Repeated terms
- Front-end compression

8.1.1 Records with Repeating Characters

Data in a fixed-length field might include a short name followed by many blank characters. This can be replaced with a variable-length field and a special code to indicate how many blanks were truncated.

For example the original string MADIBA, looks like this when it's stored uncompressed in a field that's 15 characters wide, where b stands for a blank space:

```
MADIBAbbbbbbbbb
```

When it's compressed it would look like this:

```
MADIBAb10
```

Likewise, numbers with many zeros can shortened with a code, in this case a #, to indicate how many zeros must be added to recreate the original number. For instance, if the original entry is this number:

```
5000000000
```

the compressed entry is:

```
5#9
```

8.1.2 Repeated Terms

Repeated terms can be compressed by using symbols to represent each of the most commonly used words in the database. For example the word "the" could be represented by the symbol `%1`, and the word "and" by `%2`. Therefore the sentence:

```
The cat and the dog
```

could be compressed to:

```
%1 cat %2 %1 dog
```

8.1.3 Front-end Compression

Front-end compression builds on the previous data element. For example the database that stores student's names in alphabetical order could be compressed by retaining the first name in full, then each entry after that takes a given number of characters from the previous entry that they have in common, shown as an integer, and then adds the characters that make it unique. For example

Original Name	Compressed Name
Anderson, Mary	Anderson, Mary
Anderson, Mark	8k
Anderson, Martha	8tha
Brown, James	Brown, James
Brown, Jamie	6ie

The trade-off with this is that with storage space saved, the overhead increases and processing time is lost.

8.2 Image and Sound Compression

Lossy compression allows a loss of data from the original image file to allow significant compression. This means the compression process is irreversible, as the original file cannot be reconstructed exactly.