# File Systems


Tasks and Design of a File System

# Contents


- File Organizations and File Access Methods
- Directories
- File Attributes and File Sharing
- Storage Management
- File System Caches
- Disk Scheduling
- Journaling File Systems
- Additional File System Features

Acknowledgement: Figures are taken from William Stallings: „Operating Systems“

## What is a File?

- A file is a collection of related information, treated as a single entity.
- Users (programs) access files by name, and look at their contents in terms of records, or simply byte ranges.
- Files are stored on permanent media like hard disks, CD-ROMs, or tapes.
- Storage of files is in units of blocks, determined by the geometry of the storage medium.
- Files
  - have some internal organization,
  - can be accessed in different ways,
  - have additional attributes, e.g. a protection code, timestamps, etc.



## Common Operations on Files

| | |
|---|---|
| Retrieve_All | Retreive all records of a file. |
| Retrieve_One | Retreive a single record of a file. |
| Retrieve_Few | Retrieve several records which satisfy a certain set of criteria. |
| Retrieve_Next | Retreive the “next” record (in some sense). |
| Retrieve_Previous | Retreive the “previous” record (in some sense). |
| Insert_One | Insert a new record into the file. |
| Delete_One | Delete an existing record. |
| Update_One | Retrieve a record, update part or all of the record, and rewrite the updated record into the file. |

The necessity to perform these operations influences

- the way a file is organized,
- the way a file is stored.

## What is a File System?

- A File System
  - allows users (programs) to access files by name,
  - offers the users (programs) a view of the file contents in units of records or byte ranges,
  - offers the users (programs) methods for performing file operations, which are independent of the actual storage device,
  - manages the available space on the storage devices,
  - manages the storage of files on different types of media.
- Some (few) file systems take care of the internal organization of the files by
  - offering different file organizations,
  - offering different file access methods.

  If this is done by the file system, many performance optimizations are possible.



## Possible File System Software Architecture (1)

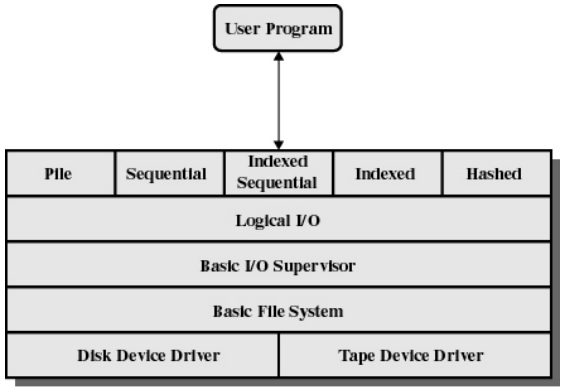



Figure 12.1 File System Software Architecture [GROS86]

## Possible File System Software Architecture (2)

- The Device Drivers (which are not considered part of the file system)
  - communicate with the physical devices,
  - are responsible for starting I/O operations on a device,
  - process the completion of an I/O request.
- The Basic File System
  - is concerned with the placement of blocks on the storage device,
  - is concerned with buffering blocks in main memory (file system cache).
- The Basic I/O Supervisor
  - is responsible for file I/O initiation and termination,
  - is concerned with scheduling I/O operations to optimize performance (disk scheduling).
- The Logical I/O layer
  - Allows locating and accessing files by name.
  - Provides general-purpose, record- or byte-range-oriented file I/O methods to applications,
  - Maintains basic data about the file (e.g. file size).
- The Access Method specifies the way a file is accessed.



## Elements of File Management

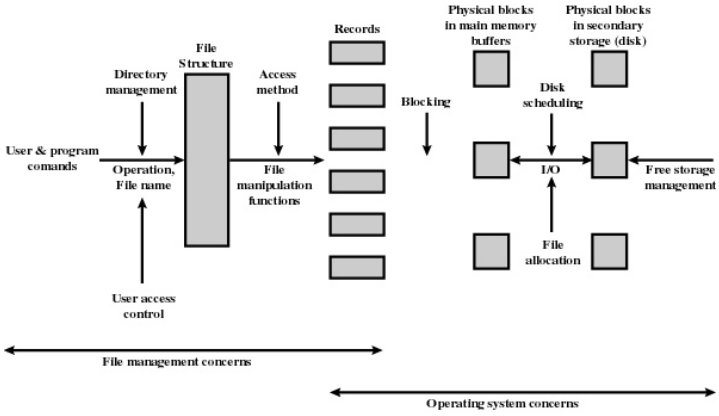



**Figure 12.2 Elements of File Management**

## File Organization

The two most common file organizations are:

- Sequential Files
  - Can have fixed length records, or variable length records.
  - Can only be accessed sequentially.
    (With fixed length records, direct access by record number is possible.)
- Indexed Files
  - In addition to the data in the file, one or more indexes are kept and maintained, which match the value of some part (field) of the records with the position of the record in the file.
  - Indexes can be partial or exhaustive (one entry per record in the file).
  - Direct access to records by specifying a value for one of the index fields.
  - A file with only one index, where the records are stored sequentially by the value of the index field is sometimes called an index-sequential file. Such a file can also be accessed sequentially by key value.



## Accessing File Records

- Sequential Access
  - Accessing the records in their normal order, from first to last, or
  - Accessing the records in the order of a key value.
  - Is the only possible access on tapes.
  - Makes performance optimizations possible, e.g. by sequential storage, or read-ahead.
- Direct Access
  - by record number (indexed file, or sequential file with fixed-size records), or
  - by key value (only for indexed files).
  - Possible on magnetic disks, and optical media (CD-ROMs, etc.).
  - Performance optimizations are more difficult (index organization and storage).

Often, the file system does not know, how a particular file will be accessed.

# Directories

- A directory is itself a file, which is owned by the file system. A directory
  - provides a mapping between file names and the files itself,
  - may contain additional attributes of the files (owner, protection code, access times etc.)
- Most file systems use hierarchical (tree-structured) directories, where each diretory can contain both files and subdirectories:

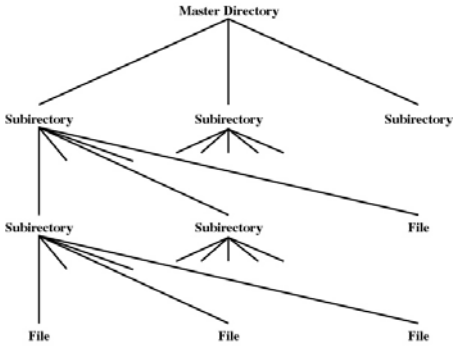



Figure 12.4 Tree-Structured Directory



# Common Operations on Directories

| | |
|---|---|
| Search | Find a file name in the directory file. |
| Create File | Add an entry for the new file to the directory. |
| Delete File | Remove the entry for the file from the directory. |
| List Directory | List the contents of the directory: file names, and selected attributes of the files. |
| Update Entry | Update the value of some attribute (e.g. access time). |

## Common File Attributes

- File name, file type, and file organization.
- Address information:
  - Location on the device on which the file is stored.
  - Size used, and size allocated on the device.
- File protection information:
  - Owner of the file.
  - Access rights.
- Usage information:
  - Date and time when the file was created.
  - Date and time when the file was last accessed, and/or modified.
  - Date and time of the last backup.

- File attributes may be stored in the directory, or in a separate data structure (e.g. the File Allocation Table in FAT, or the Master File Table in NTFS).



## File Sharing

There are two aspects to consider when sharing files:

- Access Rights
  A file protection mechanism allows to specify who (e.g. owner, group, world) is allowed which type of access (e.g. read, write, execute, delete) to a file.

- Management of simultaneous access.
  Most file systems only provide a way to specify
  - exclusive access
  - shared access

  for the whole file. A few file systems
  - offer more share modes (e.g. a reader-writer lock, which distinguishes read and write access),
  - allow to specify the share mode with a finer granularity (e.g. at record level).

## Timing of a Disk I/O Transfer

- To read or write, the disk head must be positioned at the desired track and at the beginning of the desired sector.
- Seek time is the time it takes to position the head at the desired track.
- Rotational delay is the time it takes for the beginning of the sector to reach the head.

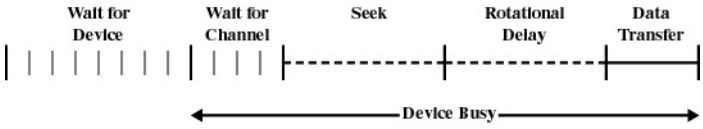



Figure 11.7 Timing of a Disk I/O Transfer



## Numerical Example for File Access Times

- A typical average seek time today is 5 to 10 ms.
- At 10000 rpm, one revolution takes 6ms, so the average rotational delay is 3 ms.
- Assume we want to read 1600 sectors (800 KB) from a disk which has 320 sectors per track, and a rotation speed of 10000 rpm.

| | All 1600 blocks are contiguous | 1600 random blocks on disk |
|---|---|---|
| Average seek: | 10 ms (at most once per track) | 10 ms per block |
| Rotational delay: | 3 ms (at most once per track) | 3 ms per block |
| Read time: | 6/320 ms per block | 6/320 ms per block |
| Total time: | 5*10ms + 5*3ms + 30ms = 0,095 s | 1600*(10+3)ms + 30ms = 20,83 s |

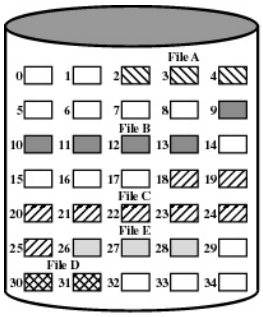
# Storage Management

- Disks are physically organized in sectors (or: blocks). Commonly, a sector is 512 bytes.
- A cluster is a fixed number of blocks. The cluster size for a volume is specified when the volume is formatted.
- Bigger clusters
  - reduce the amount of data needed for storage management,
  - waste more space due to internal fragmentation.
- File systems use a cluster as the smallest allocation unit.
- File systems need to allocate clusters to files.
- File systems need to keep track of the free space on a volume.



# Contiguous Allocation

- Each file is stored in contiguous clusters.
- The (maximum) size of a file must be declared when the file is created, and the space for it is preallocated.
- Extending a file is difficult: Allocate space for the new size, copy the file, deallocate the previously allocated space.
- External fragmentation occurs, and may require occasional compaction (defrag-mentation).
- Both sequential and direct access are possible.
- Very good performance for sequential access.




File Allocation Table

| File Name | Start Block | Length |
|---|---|---|
| File A | 2 | 3 |
| File B | 9 | 5 |
| File C | 18 | 8 |
| File D | 30 | 2 |
| File E | 26 | 3 |

**Figure 12.7 Contiguous File Allocation**

# Chained Allocation

- Allocation occurs on the basis of arbitrary single clusters.
- Each cluster contains a pointer to the next cluster in the file.
- The address information for the file only consists of the start location and the length of the file
- No external fragmentation.
- Only sequential access is possible.
- Performance for sequential access is worse than with contiguous allocation.

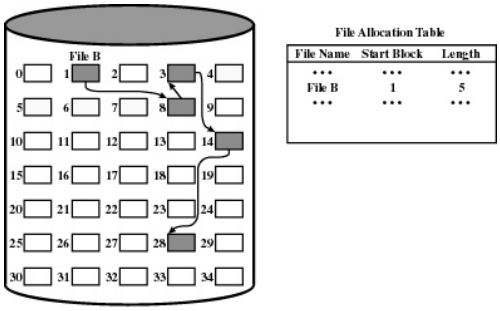



**Figure 12.9 Chained Allocation**



# Indexed Allocation of Single Clusters

- Arbitrary single clusters are used for the storage of a file.
- A table (index) contains the numbers of all clusters assigned to a file.
- The index is not stored in the directory, but in a separate data structure (FAT, MFT, ...).
- No external fragmentation.
- Both sequential and direct access are possible.
- Performance for sequential access is worse than with contiguous allocation.

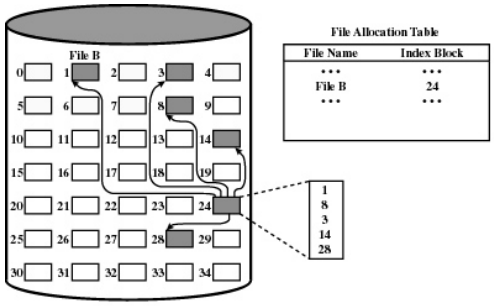



**Figure 12.11 Indexed Allocation with Block Portions**

# Indexed Allocation with Variable-Length Extents

- Instead of assigning individual clusters, portions of several contiguous clusters, called extents, are used.
- The index contains one entry per extent.
- The file system may use a best-try-contiguous approach when assigning clusters to a file.
- The performance of sequential access is improved, depending on the number and sizes of the extents.
- Occasional defragmentation can be performed to reduce the number of extents.

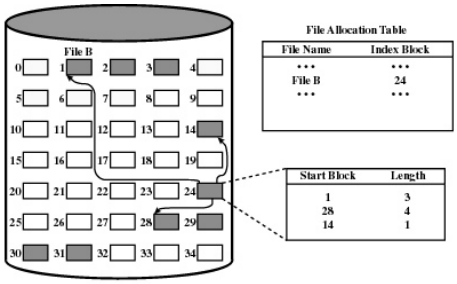



**Figure 12.12 Indexed Allocation with Variable-Length Portions**



# Free Space Management

- Most commonly, a bitmap is used to keep track of the free space on the disk.
  - The bitmap contains one bit per cluster.
  - In many file systems, the bitmap is a normal file on the disk, which is created when the disk is formatted.
- Searching for a free cluster, or a certain number of contiguous free clusters, requires searching (parts of) the bit map. This search must be accelerated
  - by keeping (parts of) the bitmap in memory, or
  - by additional data structures in main memory, describing the free space, e.g. a linked list of free portions on disk.

## Defragmentation

- With all kinds of storage management, defragmentation can be necessary, or can at least improve the performance.
- Files should be made contiguous (or nearly contiguous), by combining several extents into a single one.
- Free space should be consolidated into few, big areas by moving files.
- The placement of files on the disk (beginning, middle, end of disk) also has an influence on the performance of file accesses.
- Defragmentation can be done
  - occasionally at specified times,
  - continously in a background process.



## File System Caches

- To reduce the number of I/Os, and thus improve the performance, as much data as possible is kept in main memory.
- Caching of file system metadata:
  - Retrieval pointers for the extents of open files.
  - (Parts of) the information about free space on the device.
- Caching of file data:
  - Data read from the device is kept in memory for the case it's referenced again soon.
  - Read-ahead: Data is read from the device before being requested by the program.
  - Data written to disk is only written to memory, and will be flushed to disk later.
  - Can be particularly important for heavily used directories.

## Write-Behind vs. Write-Through Caching

- With write-behind (or write-back) caching
  - a write I/O is considered complete when the data has been written to the cache,
  - the data will be flushed to disk later.
- With write-through caching
  - data will be written both to the cache and to the disk,
  - a write I/O is only considered complete after the data has been written to disk.
- Most file systems use write-behind caching by default, because of its performance advantages.
- Depending on the particular file system, the kind of caching can be specified per volume, or per directory, or per file, or per file open.



## What is Disk Scheduling?

- Seek time and rotational delay are the reasons why I/Os can take varying amounts of time.
- Very often there will be a number of I/O requests for a single disk.
- Selecting the requests randomly will result in the worst possible performance.
- Disk scheduling means re-ordering the requests, taking the physical location on disk into account.
- Many different scheduling policies can be used.

## First In First Out (FIFO) Disk Scheduling Policy

- First in first out (FIFO) scheduling processes requests in the order in which they arrive.
- FIFO is very easy to implement.
- FIFO is fair, because it does not make a request wait in favor of some other request.
- FIFO is acceptable, if the number of concurrent requests is small.
- FIFO approaches random scheduling in performance, if there are many processes competing for the disk.



## Shortest Service Time First (SSTF) Disk Scheduling

- Shortest Service Time First (SSTF) scheduling selects the request that requires the least movement of the disk arm from its current position.
- For each request, SSTF minimizes the seek time.
- SSTF does not necessarily minimize the average seek time over a number of requests.
- A request may have to wait a long time, or even remain unfulfilled until the request queue is empty.

## SCAN and C-SCAN Disk Scheduling Policies

- With the SCAN scheduling policy
  - the disk arm moves in one direction, satisfying all outstanding requests en route, until it reaches the last request in this direction,
  - then the arm direction is reversed, and the scan proceeds in the opposite direction.
- SCAN scheduling
  - causes unpredictable delays for the requests (from immediate processing to long wait times),
  - favors newly arriving requests over already waiting ones, if the new request is for a track closer to the current position of the disk arm,
  - is biased against the areas most recently traversed.
- The C-SCAN (circular scan) scheduling policy
  - restricts scanning to one direction only,
  - when the last track in this direction has been visited, the arm is returned to the opposite end of the disk, and the scan begins again.
  - reduces the delay for requests for tracks near „the other end“ of the disk.



## N-step-SCAN and FSCAN Disk Scheduling Policies

- The N-step-SCAN scheduling policy
  - segments the disk request queue into subqueues of length N,
  - processes the subqueues one at a time using SCAN,
  - places new requests into a queue different from the one being currently processed.
- The FSCAN scheduling policy
  - uses two subqueues of arbitrary length,
  - alternately processes the subqueues,
  - places new requests into the queue which is not currently processed.
- Both N-step-SCAN and FSCAN avoid that requests have to wait a very long time because newly arriving requests are being processed first.

# Example for the Different Disk Scheduling Policies

The following pages show an example for the use of different disk scheduling policies from the book „Operating Systems“ by William Stallings.

The example assumes a disk with 200 tracks, and a disk request queue containing requests for the tracks 55, 58, 39, 18, 90, 160, 150, 38, and 184, in this order.

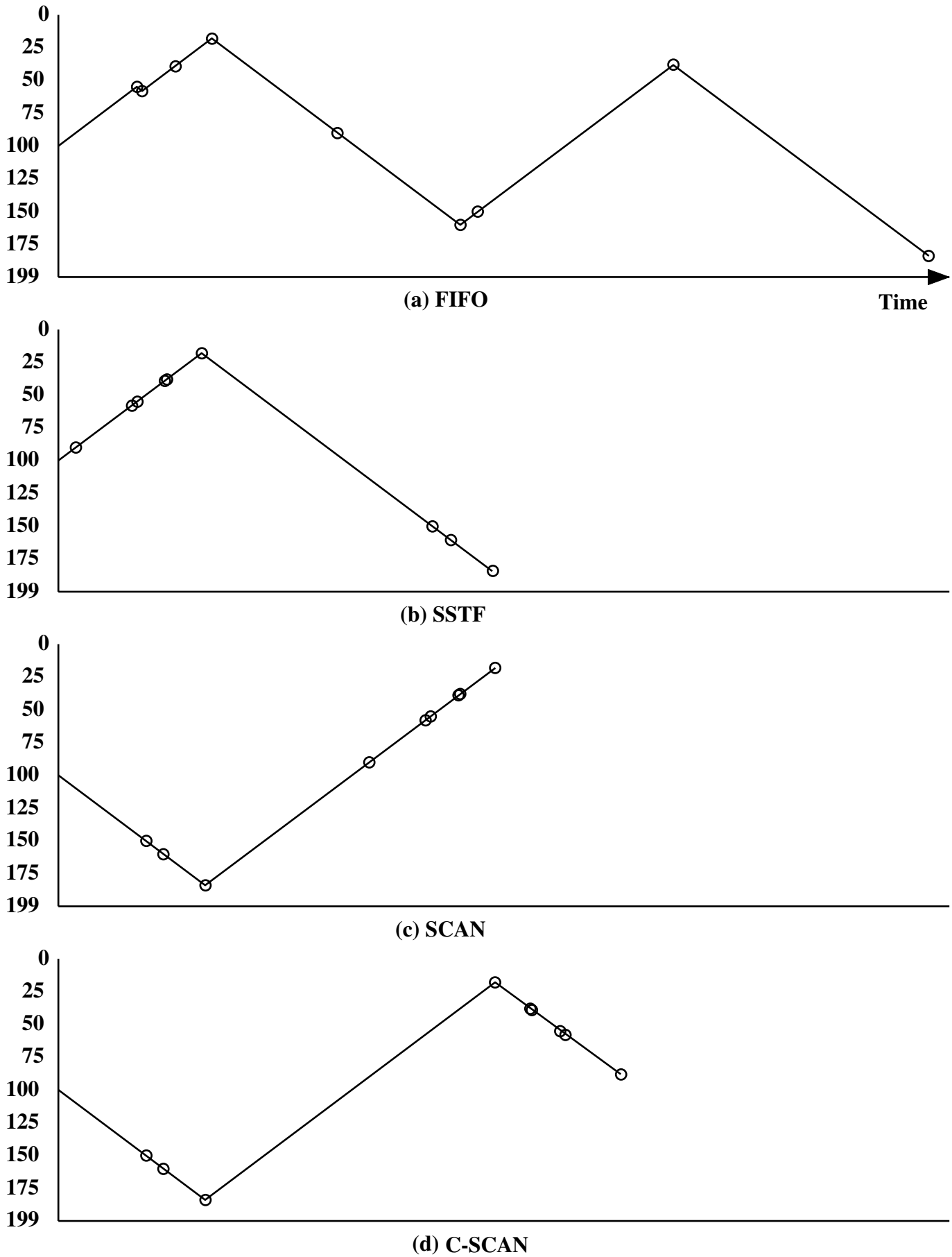



**Figure 11.8 Comparison of Disk Scheduling Algorithms (see Table 11.3)**

## Table 11.2 Comparison of Disk Scheduling Algorithms

| (a) FIFO (starting at track 100) | | (b) SSTF (starting at track 100) | | (c) SCAN (starting at track 100, in the direction of increasing track number) | | (d) C-SCAN (starting at track 100, in the direction of increasing track number) | |
|---|---|---|---|---|---|---|---|
| **Next track accessed** | **Number of tracks traversed** | **Next track accessed** | **Number of tracks traversed** | **Next track accessed** | **Number of tracks traversed** | **Next track accessed** | **Number of tracks traversed** |
| 55 | 45 | 90 | 10 | 150 | 50 | 150 | 50 |
| 58 | 3 | 58 | 32 | 160 | 10 | 160 | 10 |
| 39 | 19 | 55 | 3 | 184 | 24 | 184 | 24 |
| 18 | 21 | 39 | 16 | 90 | 94 | 18 | 166 |
| 90 | 72 | 38 | 1 | 58 | 32 | 38 | 20 |
| 160 | 70 | 18 | 20 | 55 | 3 | 39 | 1 |
| 150 | 10 | 150 | 132 | 39 | 16 | 55 | 16 |
| 38 | 112 | 160 | 10 | 38 | 1 | 58 | 3 |
| 184 | 146 | 184 | 24 | 18 | 20 | 90 | 32 |
| **Average seek length** | 55.3 | **Average seek length** | 27.5 | **Average seek length** | 27.8 | **Average seek length** | 35.8 |

## Journaling (or: Log-Based) File Systems

- Most file system operations require several disk writes. For example, creating a file requires
  - modifying the bitmap to allocate clusters to the new file,
  - possibly adding an entry for the new file in a data structure like the FAT or MFT.
  - modifying the directory file by adding an entry for the new file,
- If the system fails in between such writes
  - the file system may be left in an inconsistent state,
  - the file system may have to be checked and repaired (if possible) during the next reboot. This can be a very time-consuming operation.
- Journaling (or: log-based) file systems record (log) changes to the meta data of the file system, to enable (and speed up) the recovery of the file system after a system crash.
- Journaling for the data in the file is normally not done by file systems, but, for example, by database systems.



## Additional File System Features

- Compression
  - Data is compressed before writing it to disk.
  - Data is de-compressed after reading it from disk.
- Encryption
  - Data is encrypted before writing it to disk.
  - Data is decrypted after reading it from disk.
- Disk Quotas
  - The administrator can specify per-user limits for the amount of disk space used.
  - The file system keeps track of the disk space used by a specific user.
- Bad Block Replacement
  - When the controller notifies the file system of an unrecoverable error on a certain block
    - the cluster containing this block can be assigned to a bad block file,
    - the cluster can be replaced by another one (with a possible data loss on a read, but no data loss on a write operation).