File Systems - Part II

Amir H. Payberah
payberah@kth.se

2022

Motivation

- The file system resides permanently on secondary storage.

Motivation

- The file system resides permanently on secondary storage.

- How to

Motivation

- The file system resides permanently on secondary storage.
- How to
  - structure file use

Motivation

- The file system resides permanently on secondary storage.
- How to
  - structure file use
  - allocate disk space

Motivation

- The file system resides permanently on secondary storage.
- How to
  - structure file use
  - allocate disk space
  - recover free space

Motivation

- The file system resides permanently on secondary storage.
- How to
  - structure file use
  - allocate disk space
  - recover free space
  - track the locations of data

Motivation

- The file system resides permanently on secondary storage.
- How to
  - structure file use
  - allocate disk space
  - recover free space
  - track the locations of data
  - interface other parts of the OS to secondary storage

File System Structure

File System Structure

- Disk provides in-place rewrite and random access

File System Structure

- Disk provides in-place rewrite and random access
- File system resides on secondary storage
  - User interface to storage, mapping logical to physical
  - Efficient and convenient access to disk

File System Structure

- Disk provides in-place rewrite and random access
- File system resides on secondary storage
  - User interface to storage, mapping logical to physical
  - Efficient and convenient access to disk
- File structure
  - Logical storage unit
  - Collection of related information

File System Design Problems

- How the file system should look to the user?

File System Design Problems

- How the file system should look to the user?
  - Defining a file and its attributes
  - The operations allowed on a file
  - The directory structure for organizing files

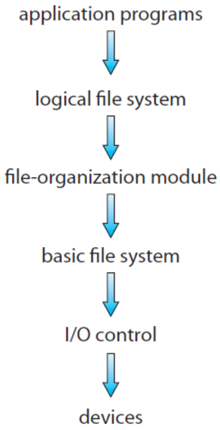
File System Design Problems

- How the file system should look to the user?
  - Defining a file and its attributes
  - The operations allowed on a file
  - The directory structure for organizing files

- Algorithms and data structures to map the logical file system onto the physical secondary-storage devices.

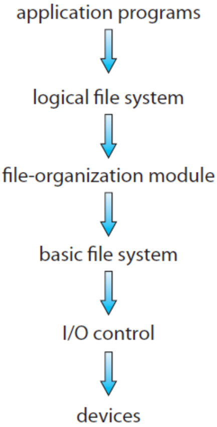
File System Layers (1/6)

- Different levels



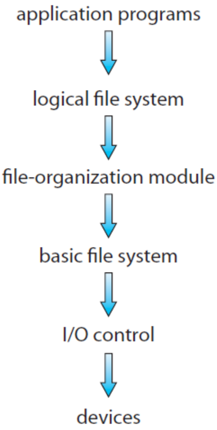
File System Layers (1/6)

- Different levels
- Each level uses the features of lower levels to create new features for use by higher levels.



File System Layers (1/6)

- Different levels
- Each level uses the features of lower levels to create new features for use by higher levels.
- Reducing complexity and redundancy, but adds overhead and can decrease performance.



application programs

logical file system

file-organization module

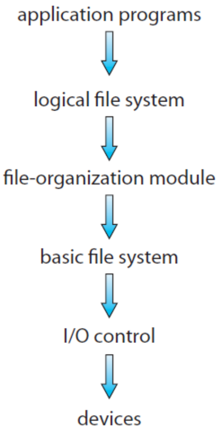
basic file system

I/O control

devices

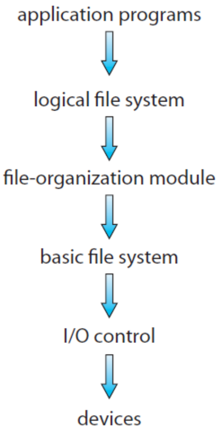
File System Layers (2/6)

- Device drivers manage I/O devices at the I/O control layer.



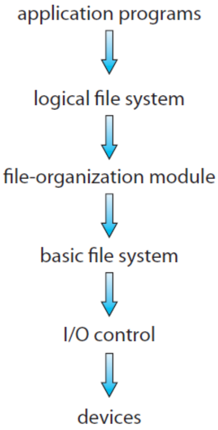
File System Layers (2/6)

- Device drivers manage I/O devices at the I/O control layer.
- Translates high-level commands to low-level hardware-specific instructions.



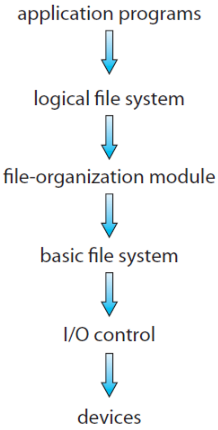
File System Layers (3/6)

- Basic file system translates given command like retrieve block 123 to device driver.



File System Layers (3/6)

- Basic file system translates given command like retrieve block 123 to device driver.
- Also manages memory buffers and caches.



application programs

logical file system

file-organization module

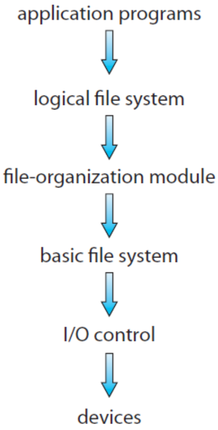
basic file system

I/O control

devices

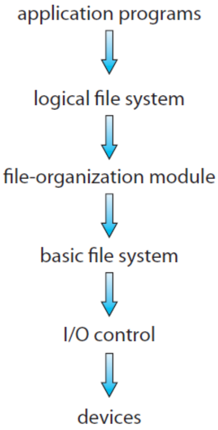
File System Layers (3/6)

- Basic file system translates given command like retrieve block 123 to device driver.
- Also manages memory buffers and caches.
  - Buffers hold data in transit



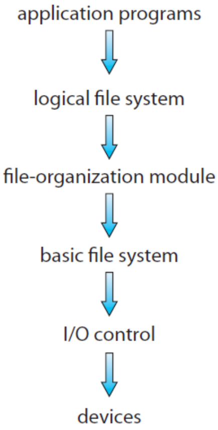
File System Layers (3/6)

- Basic file system translates given command like retrieve block 123 to device driver.
- Also manages memory buffers and caches.
  - Buffers hold data in transit
  - Caches hold frequently used data



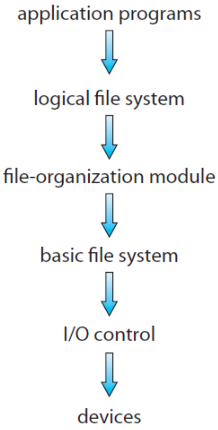
File System Layers (4/6)

- File organization understands files, logical address, and physical blocks.



File System Layers (4/6)

- File organization understands files, logical address, and physical blocks.
- Translates logical block number to physical block number.

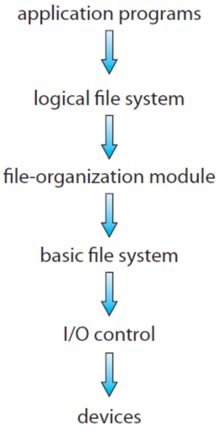


File System Layers (4/6)

- File organization understands files, logical address, and physical blocks.

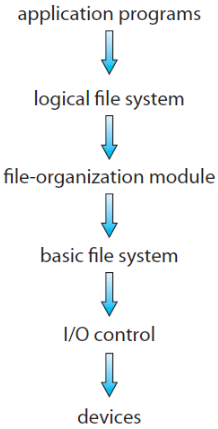
- Translates logical block number to physical block number.

- Manages free space and disk allocation.



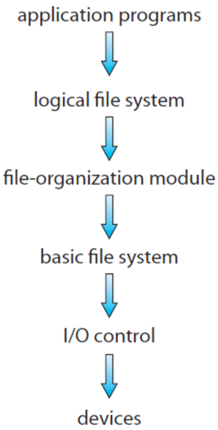
File System Layers (5/6)

- Logical file system manages metadata information.



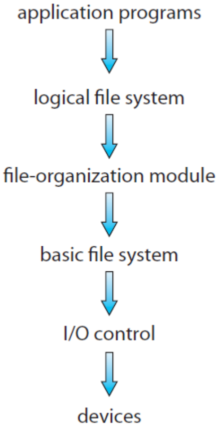
File System Layers (5/6)

- Logical file system manages metadata information.
- Translates file name into file number, file handle, location by maintaining file control blocks (inodes in Unix)



File System Layers (5/6)

- Logical file system manages metadata information.
- Translates file name into file number, file handle, location by maintaining file control blocks (inodes in Unix)
- Directory management and protection



File System Layers (6/6)

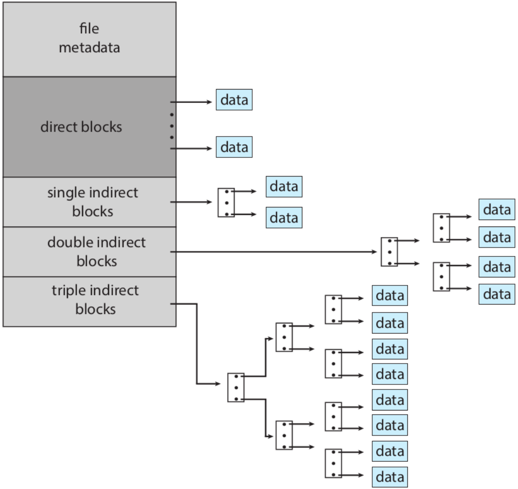
- Many file systems, sometimes many within an OS

File System Layers (6/6)

- Many file systems, sometimes many within an OS
- Each with its own format
  - CD-ROM: ISO 9660
  - Unix: UFS, FFS
  - Windows: FAT, FAT32, NTFS
  - Linux: more than 40 types, with extended file system (ext2, ext3, ext4)

File System Implementation

The Unix inode



File System Implementation

- Based on several on-disk and in-memory structures.

File System Implementation

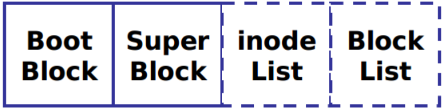
- Based on several on-disk and in-memory structures.
- On-disk
  - Boot control block (per volume)
  - Volume control block (per volume)
  - Directory structure (per file system)
  - File control block (per file)

File System Implementation

- Based on several on-disk and in-memory structures.
- On-disk
  - Boot control block (per volume)
  - Volume control block (per volume)
  - Directory structure (per file system)
  - File control block (per file)
- In-memory
  - Mount table
  - Directory structure cache
  - The open-file table (system-wide and per process)
  - Buffers of the file-system blocks

On-Disk File System Structures (1/2)

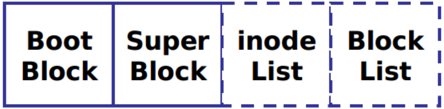
- Boot control block contains information needed by system to boot OS from that volume.



UFS on-disk structures

On-Disk File System Structures (1/2)

- Boot control block contains information needed by system to boot OS from that volume.
  - Needed if volume contains OS, usually first block of volume.
  - In UFS, it is called boot block, and in NTFS partition boot sector.

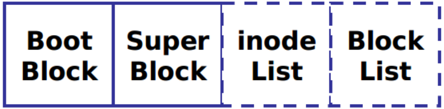


UFS on-disk structures

On-Disk File System Structures (1/2)

- Boot control block contains information needed by system to boot OS from that volume.
  - Needed if volume contains OS, usually first block of volume.
  - In UFS, it is called boot block, and in NTFS partition boot sector.

- Volume control block contains volume details.

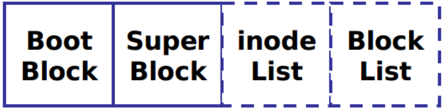


UFS on-disk structures

On-Disk File System Structures (1/2)

- Boot control block contains information needed by system to boot OS from that volume.
  - Needed if volume contains OS, usually first block of volume.
  - In UFS, it is called boot block, and in NTFS partition boot sector.

- Volume control block contains volume details.
  - Total num. of blocks, num. of free blocks, block size, free block pointers or array
  - In UFS, it is called super block, and in NTFS master file table.



Boot Block | Super Block | inode List | Block List

UFS on-disk structures

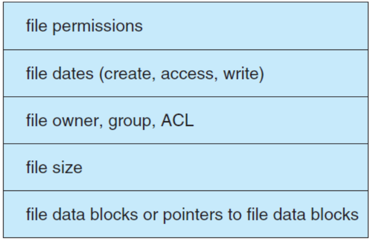
On-Disk File System Structures (2/2)

- Directory structure organizes the files.
  - In UFS, this includes file names and associated inode numbers.
  - In NTFS, it is stored in the master file table.

On-Disk File System Structures (2/2)

- Directory structure organizes the files.
  - In UFS, this includes file names and associated inode numbers.
  - In NTFS, it is stored in the master file table.

- File Control Block (FCB) contains many details about the file.
  - In UFS, inode number, permissions, size, dates.
  - In NFTS stores into in master file table.



file permissions

file dates (create, access, write)

file owner, group, ACL

file size

file data blocks or pointers to file data blocks

File Control Block (FCB)

In-Memory File System Structures

- Mount table contains information about each mounted volume.

In-Memory File System Structures

- Mount table contains information about each mounted volume.
- Directory structure cache holds the directory information of recently accessed directories.

In-Memory File System Structures

- Mount table contains information about each mounted volume.
- Directory structure cache holds the directory information of recently accessed directories.
- System-wide open-file table contains a copy of the FCB of each open file.

In-Memory File System Structures

- Mount table contains information about each mounted volume.
- Directory structure cache holds the directory information of recently accessed directories.
- System-wide open-file table contains a copy of the FCB of each open file.
- Per-process open-file table contains a pointer to the appropriate entry in the system-wide open-file table.

In-Memory File System Structures

- Mount table contains information about each mounted volume.
- Directory structure cache holds the directory information of recently accessed directories.
- System-wide open-file table contains a copy of the FCB of each open file.
- Per-process open-file table contains a pointer to the appropriate entry in the system-wide open-file table.
- Buffers hold file-system blocks when they are being read from disk or written to disk.

Create a File

- A program calls the logical file system.

Create a File

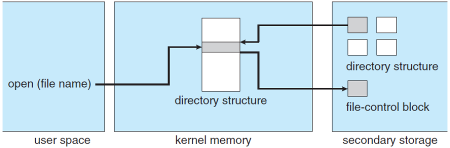
- A program calls the logical file system.
- The logical file system knows the format of the directory structures, and allocates a new FCB.

Create a File

- A program calls the logical file system.
- The logical file system knows the format of the directory structures, and allocates a new FCB.
- The system, then, reads the appropriate directory into memory, updates it with the new file name and FCB, and writes it back to the disk.

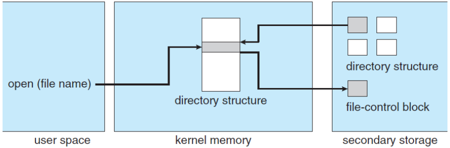
Open a File

- The `open()` passes a file name to the logical file system.



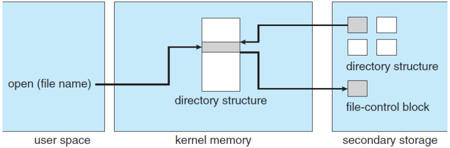
Open a File

- The `open()` passes a file name to the logical file system.
- The `open()` first searches the system-wide open-file: if the file is already in use by another process.



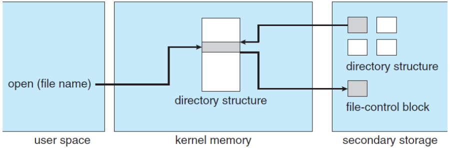
Open a File

- The open() passes a file name to the logical file system.
- The open() first searches the system-wide open-file: if the file is already in use by another process.
  - If yes: a per-process open-file table entry is created.



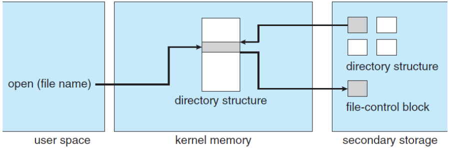
Open a File

- The open() passes a file name to the logical file system.
- The open() first searches the system-wide open-file: if the file is already in use by another process.
  - If yes: a per-process open-file table entry is created.
  - If no: the directory structure is searched for the given file name: once the file is found, the FCB is copied into a system-wide open-file table in memory.



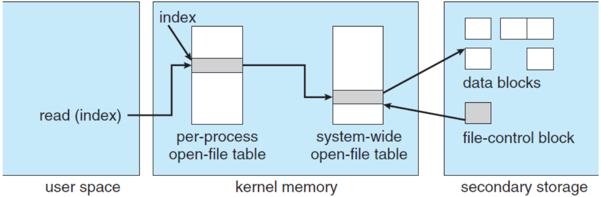
Open a File

- The open() passes a file name to the logical file system.
- The open() first searches the system-wide open-file: if the file is already in use by another process.
  - If yes: a per-process open-file table entry is created.
  - If no: the directory structure is searched for the given file name: once the file is found, the FCB is copied into a system-wide open-file table in memory.
- This table stores the FCB as well as the number of processes that have the file open.



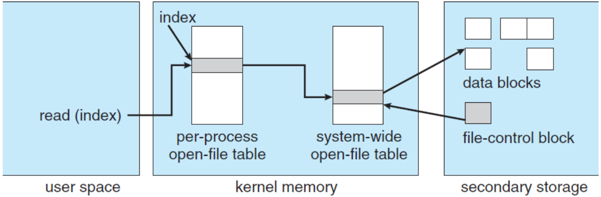
Read From a File

- The `open()` returns a pointer to the appropriate entry in the per-process file-system table.



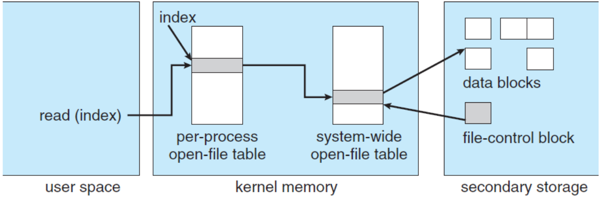
Read From a File

- The `open()` returns a pointer to the appropriate entry in the per-process file-system table.
- All file operations are then performed via this pointer.



Read From a File

- The `open()` returns a pointer to the appropriate entry in the per-process file-system table.
- All file operations are then performed via this pointer.
- This pointer is called file descriptor in Unix and file handle in Windows.



Close a File

- When a process closes the file:
  - The per-process table entry is removed.
  - The system-wide entry's open count is decremented.

Close a File

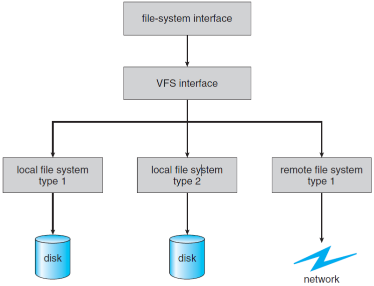
- When a process closes the file:
  - The per-process table entry is removed.
  - The system-wide entry's open count is decremented.

- When all users that have opened the file close it, any updated metadata is copied back to the disk-based directory structure, and the system-wide open-file table entry is removed.

Virtual File Systems

Virtual File Systems (1/2)

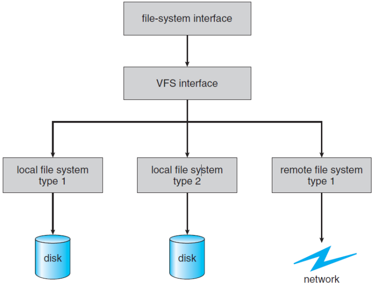
- Virtual File Systems (VFS) on Unix provide an object-oriented way of implementing file systems.



Virtual File Systems (1/2)

- Virtual File Systems (VFS) on Unix provide an object-oriented way of implementing file systems.

- VFS allows the same system call interface (the API) to be used for different types of file systems.



Virtual File Systems (2/2)

- VFS layer serves two important functions:

Virtual File Systems (2/2)

- VFS layer serves two important functions:
  1. It separates file-system-generic operations from their implementation, and allows transparent access to different types of file systems mounted locally.

Virtual File Systems (2/2)

- VFS layer serves two important functions:
  1. It separates file-system-generic operations from their implementation, and allows transparent access to different types of file systems mounted locally.
  2. It provides a mechanism for uniquely representing a file throughout a network.

Virtual File Systems (2/2)

- VFS layer serves two important functions:
  1. It separates file-system-generic operations from their implementation, and allows transparent access to different types of file systems mounted locally.
  2. It provides a mechanism for uniquely representing a file throughout a network.

- The VFS is based on a structure, called a vnode.

Virtual File Systems (2/2)

- VFS layer serves two important functions:
  1. It separates file-system-generic operations from their implementation, and allows transparent access to different types of file systems mounted locally.
  2. It provides a mechanism for uniquely representing a file throughout a network.

- The VFS is based on a structure, called a vnode.
  - Contains a numerical designator for a network-wide unique file.

Virtual File Systems (2/2)

- VFS layer serves two important functions:
  1. It separates file-system-generic operations from their implementation, and allows transparent access to different types of file systems mounted locally.
  2. It provides a mechanism for uniquely representing a file throughout a network.

- The VFS is based on a structure, called a vnode.
  - Contains a numerical designator for a network-wide unique file.
  - Unix inodes are unique within only a single file system.

Virtual File Systems (2/2)

- VFS layer serves two important functions:
  1. It separates file-system-generic operations from their implementation, and allows transparent access to different types of file systems mounted locally.
  2. It provides a mechanism for uniquely representing a file throughout a network.

- The VFS is based on a structure, called a vnode.
  - Contains a numerical designator for a network-wide unique file.
  - Unix inodes are unique within only a single file system.
  - The kernel maintains one vnode structure for each active node.

Allocation Methods

Allocation Methods

- How disk blocks are allocated to files?

Allocation Methods

- How disk blocks are allocated to files?

- Methods:
  - Contiguous allocation
  - Linked allocation
  - Indexed allocation

Contiguous Allocation

Contiguous Allocation (1/2)

- Contiguous allocation: each file occupies set of contiguous blocks.

Contiguous Allocation (1/2)

- Contiguous allocation: each file occupies set of contiguous blocks.
  - Best performance in most cases
  - Simple: only starting location (block number) and length (number of blocks) are required.

Contiguous Allocation (1/2)

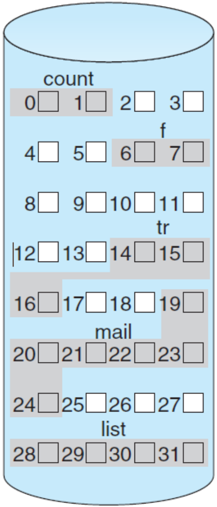
- Contiguous allocation: each file occupies set of contiguous blocks.
  - Best performance in most cases
  - Simple: only starting location (block number) and length (number of blocks) are required.
  - Supports both sequential and direct access.

Contiguous Allocation (1/2)

- Contiguous allocation: each file occupies set of contiguous blocks.
  - Best performance in most cases
  - Simple: only starting location (block number) and length (number of blocks) are required.
  - Supports both sequential and direct access.

- Allocation strategies like contiguous memory allocation:
  - First fit
  - Best fit
  - Worst fit

Contiguous Allocation (2/2)



directory

file	start	length
count	0	2
tr	14	3
mail	19	6
list	28	4
f	6	2

Contiguous Allocation Problems

- Finding space for file

Contiguous Allocation Problems

- Finding space for file
- External fragmentation

Contiguous Allocation Problems

- Finding space for file
- External fragmentation
- Need for compaction (fragmentation) off-line or on-line: lose of performance

Contiguous Allocation Problems

- Finding space for file
- External fragmentation
- Need for compaction (fragmentation) off-line or on-line: lose of performance
- Knowing file size

Linked Allocation

Linked Allocation (1/2)

- Linked allocation: each file is a linked list of blocks.
  - Each block contains pointer to next block.
  - File ends at null pointer.

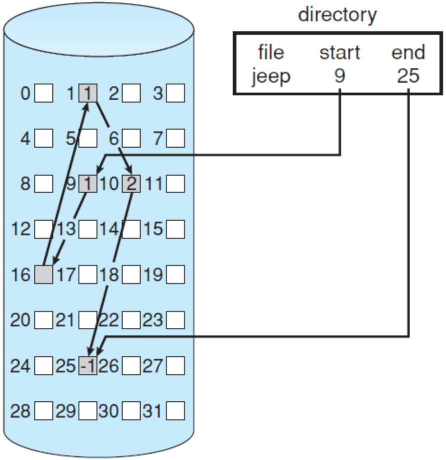
Linked Allocation (1/2)

- Linked allocation: each file is a linked list of blocks.
  - Each block contains pointer to next block.
  - File ends at null pointer.
- No external fragmentation, no compaction.

Linked Allocation (1/2)

- Linked allocation: each file is a linked list of blocks.
  - Each block contains pointer to next block.
  - File ends at null pointer.
- No external fragmentation, no compaction.
- Free space management system called when new block needed.

Linked Allocation (2/2)



Linked Allocation Problems

- Locating a block can take many I/Os and disk seeks.

Linked Allocation Problems

- Locating a block can take many I/Os and disk seeks.
- Reliability can be a problem.

Linked Allocation Problems

- Locating a block can take many I/Os and disk seeks.
- Reliability can be a problem.
- The space required for the pointers.
    - Efficiency can be improved by clustering blocks into groups but increases internal fragmentation.

Indexed Allocation

Indexed Allocation (1/2)

- Indexed allocation: each file has its own index block(s) of pointers to its data blocks.

Indexed Allocation (1/2)

- Indexed allocation: each file has its own index block(s) of pointers to its data blocks.
- Need index table

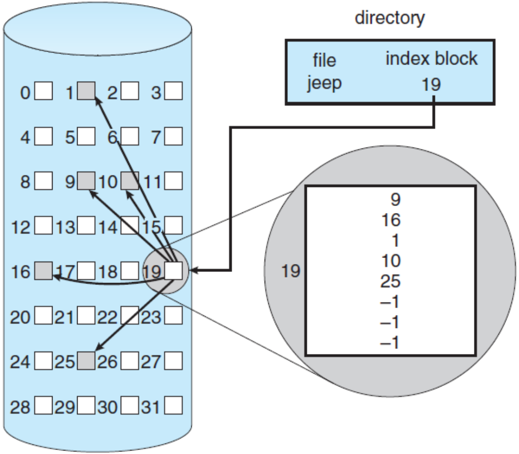
Indexed Allocation (1/2)

- Indexed allocation: each file has its own index block(s) of pointers to its data blocks.
- Need index table
- Random access

Indexed Allocation (1/2)

- Indexed allocation: each file has its own index block(s) of pointers to its data blocks.
- Need index table
- Random access
- Dynamic access without external fragmentation, but have overhead of index block

Indexed Allocation (2/2)



Indexed Allocation Problems

- Wasted space: overhead of the index blocks.
- For example, even with a file of only one or two blocks, we need an an entire index block.

Index Block Size

- How large the index block should be?

Index Block Size

- How large the index block should be?

- Keep the index block as small as possible.
  - We need a mechanism to hold pointers for large files.

Index Block Size

- How large the index block should be?

- Keep the index block as small as possible.
  - We need a mechanism to hold pointers for large files.

- Mechanisms for this purpose include the following:
  - Linked scheme
  - Multi-level index
  - Combined scheme

Linked Scheme

- Linked scheme: link blocks of index table (no limit on size)

Linked Scheme

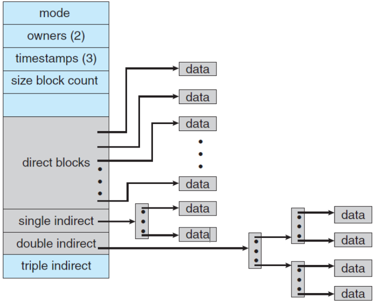
- Linked scheme: link blocks of index table (no limit on size)
- For example, an index block might contain a small header giving the name of the file and a set of the first 100 disk-block addresses.
- The next address is null or is a pointer to another index block.

Multi-Level Index

- Two-level index
- A first-level index block to point to a set of second-level index blocks, which in turn point to the file blocks.
- Could be continued to a third or fourth level.

Combined Scheme

- Combine scheme: used in Unix/Linux FS
- The first 12 pointers point to direct blocks
  - The data for small files do not need a separate index block.
- The next 3 pointers point to indirect blocks.
  - Single indirect
  - Double indirect
  - Triple indirect



Summary

Summary

- FS layers: device, I/O control, basic FS, file-organization, logical FS, application

Summary

- FS layers: device, I/O control, basic FS, file-organization, logical FS, application
- FS implementation:
  - On-disk structures: boot control block, volume control block, directory structure, and file control block
  - In-memory structures: mount table, directory structure, open-file tables, and buffers

Summary

- FS layers: device, I/O control, basic FS, file-organization, logical FS, application
- FS implementation:
  - On-disk structures: boot control block, volume control block, directory structure, and file control block
  - In-memory structures: mount table, directory structure, open-file tables, and buffers
- Virtual file system (VFS)

Summary

- FS layers: device, I/O control, basic FS, file-organization, logical FS, application

- FS implementation:
  - On-disk structures: boot control block, volume control block, directory structure, and file control block
  - In-memory structures: mount table, directory structure, open-file tables, and buffers

- Virtual file system (VFS)

- Allocation methods: contiguous allocation, linked allocation, and indexed allocation

Questions?

Acknowledgements

Some slides were derived from Avi Silberschatz slides.