

File Systems: Fundamentals

1

Files

- ◆ What is a file?
 - A named collection of related information recorded on secondary storage (e.g., disks)
- ◆ File attributes
 - Name, type, location, size, protection, creator, creation time, last-modified-time, ...
- ◆ File operations
 - Create, Open, Read, Write, Seek, Delete, ...
- ◆ How does the OS allow users to use files?
 - "Open" a file before use
 - OS maintains an **open file table** per process
 - Allow sharing by maintaining a system-wide open file table

2

File System Functionality

- ◆ Key functions:
 - Allocate disk storage to files
 - Manage the collection of files (locate files and its content)
- ◆ Allocation of disk blocks to files
 - Data structures storing free block list
- ◆ Management of file blocks
 - Data structures storing names, locations, lengths, owner, etc. of all files on disk
 - ❖ a symbol table
 - A file header per file - indicating which disk sectors are associated with each file

3

Allocation of Storage Space to Files

- ◆ Represent the list of free blocks as a *bit vector*:
1111111111111111001110101011101111...
 - If bit $i = 0$ then block i is *free*, if $i = 1$ then it is *allocated*

Simple to use but this can be a big vector:
17.5 million elements for a 9 GB disk (2.2 MB worth of bits)

However, if free sectors are uniformly distributed across the disk then the expected number of bits that must be scanned before finding a "0" is

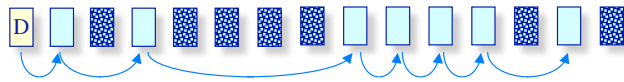
where n/r
 n = total number of blocks on the disk,
 r = number of free blocks

If a disk is 90% full, then the average number of bits to be scanned is 10, independent of the size of the disk

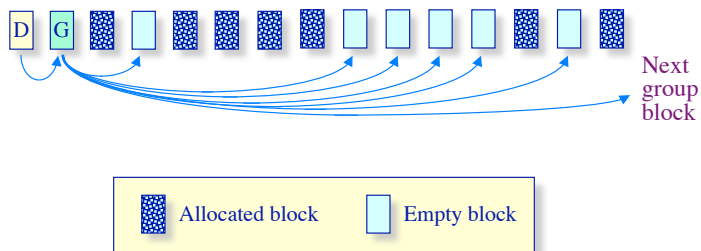
4

Other Free List Representations

◆ In-situ linked lists



◆ Grouped lists



5

File Allocation Methods

Contiguous allocation

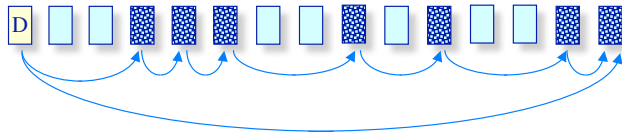


- ◆ File header specifies starting block & length
- ◆ Placement/Allocation policies
 - First-fit, best-fit, ...
- ◆ Pluses
 - Best file read performance
 - Efficient sequential & random access
- ◆ Minuses
 - Fragmentation!
 - Problems with file growth
 - ❖ Pre-allocation?
 - ❖ On-demand allocation?

6

File Allocation Methods

Linked allocation



- ◆ Files stored as a linked list of blocks
- ◆ File header contains a pointer to the first and last file blocks
- ◆ Pluses
 - Easy to create, grow & shrink files
 - No fragmentation
- ◆ Minuses
 - Impossible to do true random access
 - Reliability
 - ❖ Break one link in the chain and...

7

File Allocation Methods

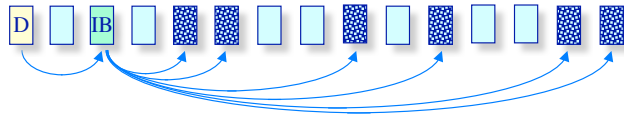
Linked allocation - File Allocation Table (FAT) (Win9x, OS2)

- ◆ Maintain linked list in a separate table
 - A table entry for each block on disk
 - Each table entry in a file has a pointer to the next entry in that file (with a special "eof" marker)
 - A "0" in the table entry → free block
- ◆ Comparison with linked allocation
 - If FAT is cached → better sequential and random access performance
 - ❖ How much memory is needed to cache entire FAT?
 - 20GB disk, 1KB/sector → 20M entries in FAT → 80MB

8

File Allocation Methods

Indexed allocation



- ◆ Create a non-data block for each file called the *index block*
 - A list of pointers to file blocks
- ◆ File header contains the index block

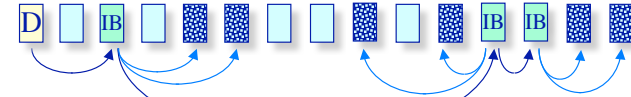
- ◆ Pluses
 - Easy to create, grow & shrink files
 - No fragmentation
 - Supports direct access
- ◆ Minuses
 - Overhead of storing index when files are small
 - How to handle large files?

9

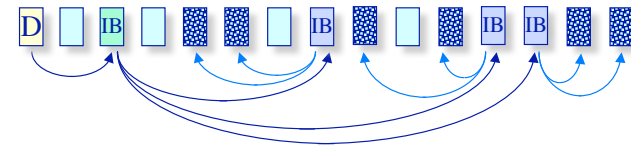
Indexed Allocation

Handling large files

- ◆ Linked index blocks ($|IB| + |IB| + \dots$)



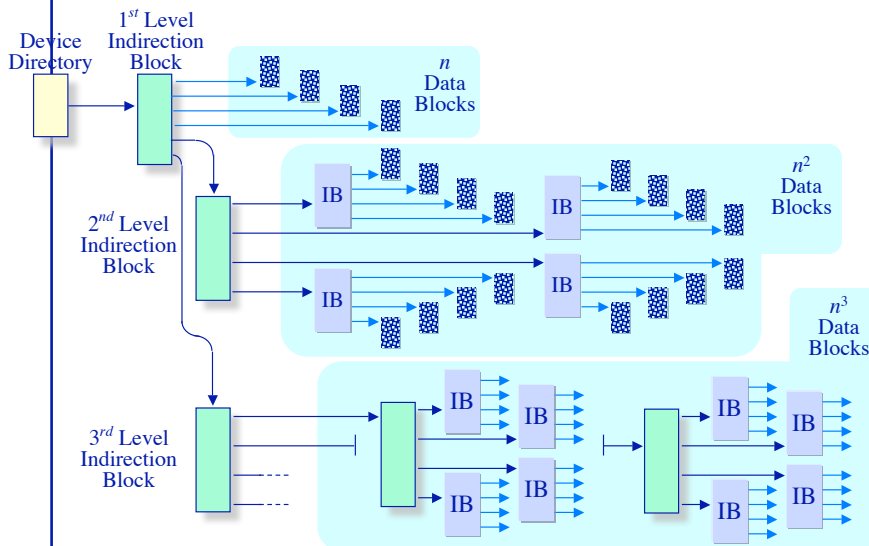
- ◆ Multilevel index blocks ($|IB| * |IB| * \dots$)



10

Indexed Allocation in UNIX

Multilevel, indirection, index blocks



Multi-level Indirection in Unix

- ◆ File header contains 13 pointers
 - 10 pointers to data blocks; 11th pointer → indirect block; 12th pointer → doubly-indirect block; and 13th pointer → triply-indirect block
- ◆ Implications
 - Upper limit on file size
 - Blocks are allocated dynamically (allocate indirect blocks only for large files)
- ◆ Features
 - Pros
 - ◆ Simple
 - ◆ Files can easily expand
 - ◆ Small files are cheap
 - Cons
 - ◆ Large files require a lot of seek to access indirect blocks

12

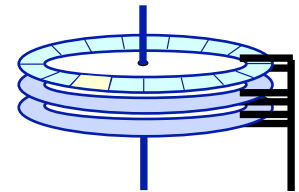
Naming and Directories

- ◆ Once you have the file header, you can access all blocks within a file
 - How to find the file header?
- ◆ Where are file headers stored on disk?
 - In early Unix:
 - ❖ Special reserved array of sectors
 - ❖ Files are referred to with an index into the array (I-node number)
 - ❖ Limitations: (1) Header is not near data; (2) fixed size of array → fixed number of files on disk (determined at the time of formatting the disk)
 - Recent Unix:
 - ❖ Distribute file header array across cylinders
- ◆ How do we find the I-node number for a file?
 - Solution: directories and name lookup

13

Naming and Directories

- ◆ Files are organized in directories
 - Directories are themselves files
 - Contain <name, pointer to file header> table
- ◆ Only OS can modify a directory
 - Ensure integrity of the mapping
 - Application programs can read directory (e.g., ls)
- ◆ Directory operations:
 - Search (find a file)
 - ❖ Linear search
 - ❖ Binary search
 - ❖ Hash table
 - Create a file
 - Delete a file
 - List contents of a directory
 - Backup



14

Directory Hierarchy and Traversal

- ◆ Directories are often organized in a hierarchy
- ◆ Directory traversal:
 - How do you find blocks of a file? Let's start at the bottom
 - ❖ Find file header - it contains pointers to file blocks
 - ❖ To find file header, we need its I-number
 - ❖ To find I-number, read the directory that contains the file
 - ❖ But wait, the directory itself is a file
 - ❖ Recursion !!
 - Example: Read file /A/B/C
 - ❖ C is a file
 - ❖ B/ is a directory that contains the I-number for file C
 - ❖ A/ is a directory that contains the I-number for file B
 - ❖ How do you find I-number for A?
 - "/" is a directory that contains the I-number for file A
 - What is the I-number for "/"? In Unix, it is 2

15

Directory Traversal (Cont'd.)

- ◆ How many disk accesses are needed to access file /A/B/C?
 1. Read I-node for "/" (root) from a fixed location
 2. Read the first data block for root
 3. Read the I-node for A
 4. Read the first data block of A
 5. Read the I-node for B
 6. Read the first data block of B
 7. Read I-node for C
 8. Read the first data block of C
- ◆ Optimization:
 - Maintain the notion of a current working directory (CWD)
 - Users can now specify relative file names
 - OS can cache the data blocks of CWD

16