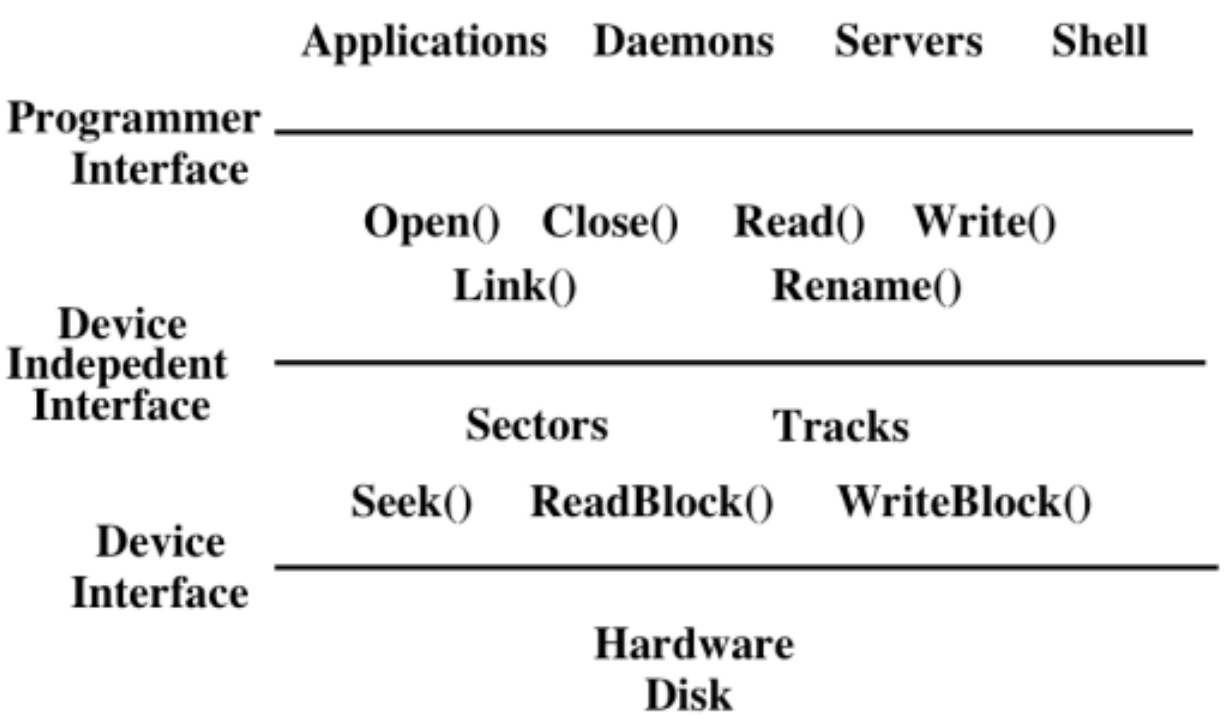
CSC139 Operating System Principles

Fall 2019, Part 4-4

Instructor: Dr. Yuan Cheng

Last Class: File System Abstraction

- Naming
- Protection
- Persistence
- Access Methods



Protection

- The OS must allow users to control sharing of their files => control access to files
- Grant or deny access to file operations depending on protection information
- Access lists and groups (Windows NT)
  - Keep an access list for each file with user name and type of access
  - Lists can become large and tedious to maintain
- Access control bits (UNIX)
  - Three categories of users (owner, group, world)
  - Three types of access privileges (read, write, execute)
  - Maintain a bit for each combination (111101000 = rwxr-x---)

Today: File System Implementation

- Disk management
  - Brief review of how disks work (already discussed two lectures ago)
  - How to organize data on disks

File Organization on Disk

- The information we need:

  fileID 0, Block 0 → Platter 0, cylinder 0, sector 0

  fileID 0, Block 1 → Platter 4, cylinder 3, sector 8

  …

- Key performance issues:

1. We need to support sequential and random access
2. What is the right data structure in which to maintain file location information?
3. How do we lay out the files on the physical disk?

File Organization: On-Disk Data Structures

- The structure used to describe where the file is on the disk and the attributes of the file is the *file descriptor* (*FileDesc*). File descriptors have to be stored on disks just like files

- Most systems fit the following profile:

1. Most files are small
2. Most disk space is taken up by large files
3. I/O operations target both small and large files

==> The per-file cost must be low, but large files must also have good performance

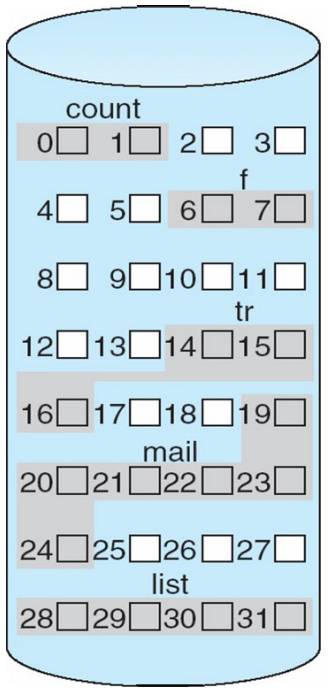
Directory Implementation

- **Linear list** of file names with pointer to the data blocks
  - Simple to program
  - Time-consuming to execute
    - Linear search time
    - Could keep ordered alphabetically via linked list or use B+ tree
- **Hash Table** – linear list with hash data structure
  - Decreases directory search time
  - Collisions – situations where two file names hash to the same location
  - Only good if entries are fixed size, or use chained-overflow method

Contiguous Allocation

- OS maintains an ordered list of free disk blocks
- OS allocates a contiguous chunk of free blocks when it creates a file
- Need to store only the start location and size in the file descriptor
- Advantages
  - Simple – only starting location (block #) and length (number of blocks) are required
  - Access time? Number of seeks? (sequential and random access)
- Disadvantages
  - Changing file sizes
  - Fragmentation? Disk management?
    - External fragmentation, need for compaction
- Examples: IBM OS/360, write-once disks, early personal computers

Contiguous Allocation (cont.)



directory

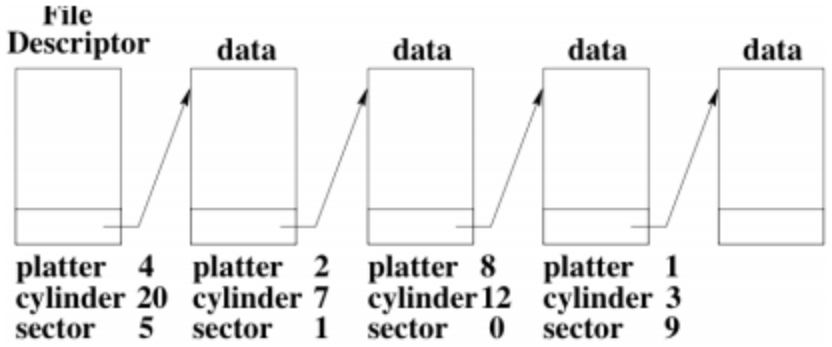
file	start	length
count	0	2
tr	14	3
mail	19	6
list	28	4
f	6	2

Extent-Based Systems

- Many newer file systems (i.e., Veritas File System) use a modified contiguous allocation scheme
- Extent-based file systems allocate disk blocks in extents
- An extent is a contiguous block of disks
  - Extents are allocated for file allocation
  - A file consists of one or more extents

Linked Files

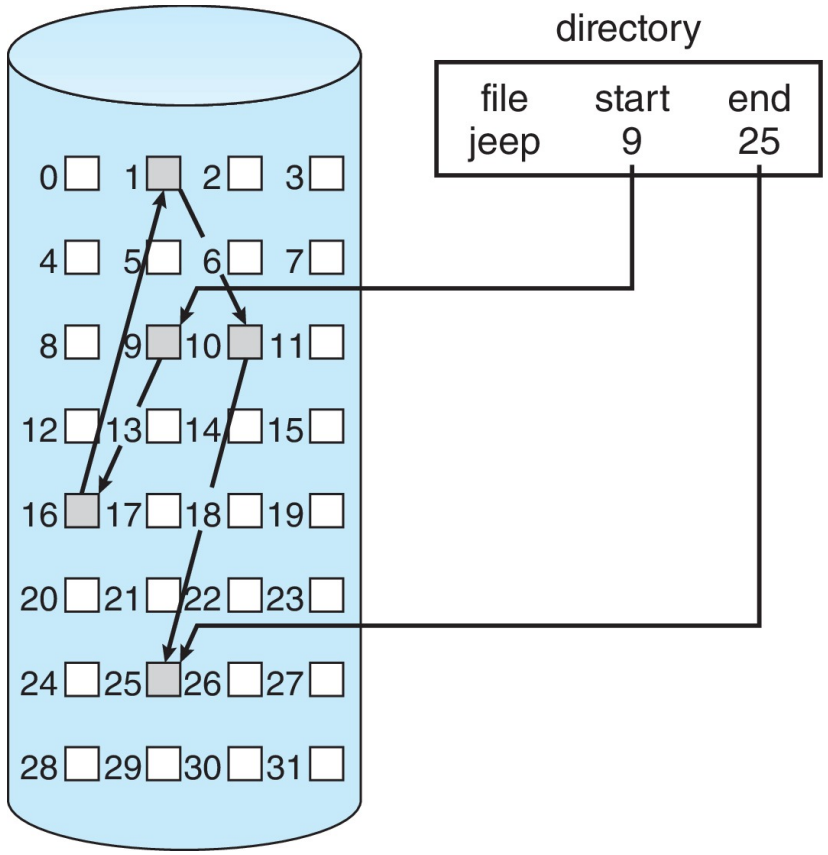
- Keep a list of all the free sectors/blocks
- In the file descriptor, keep a pointer to the first sector/block
- In each sector, keep a pointer to the next sector



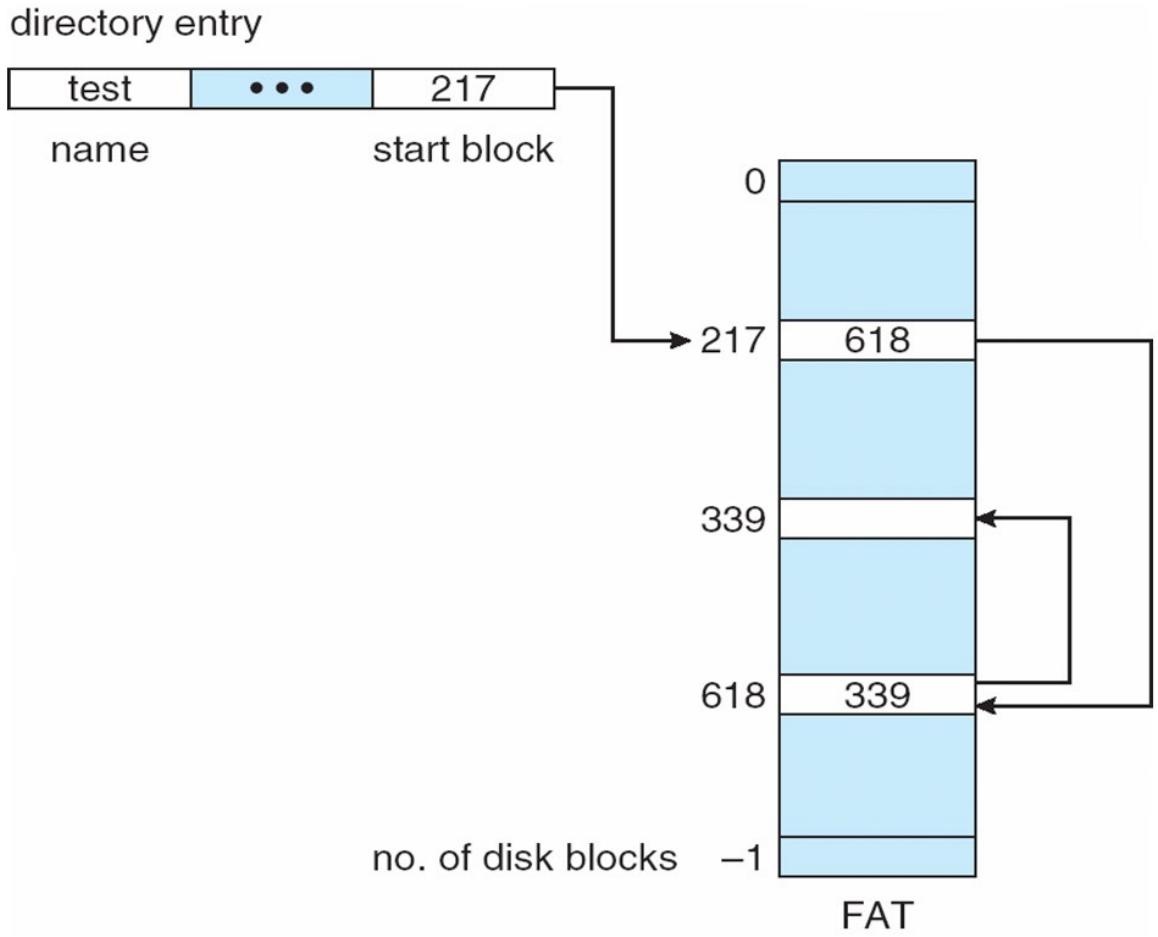
Linked Files (cont.)

- Advantages:
  - Fragmentation?
  - File size changes?
  - Efficiently supports which type of access?
- Disadvantages:
  - Does not support which type of access? Why?
  - Number of seeks?
- Examples: FAT, MS-DOS

Linked Allocation



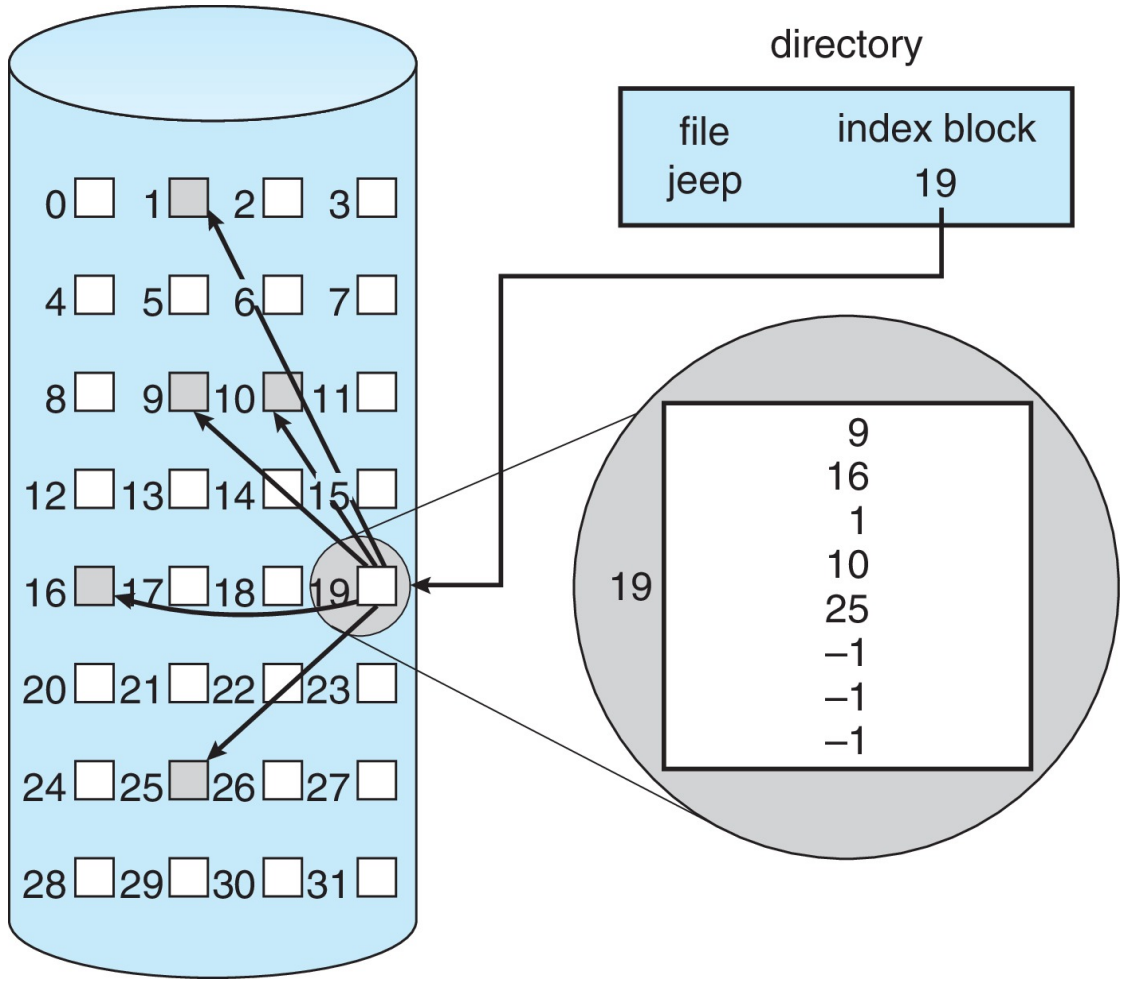
File-Allocation Table



Indexed Files

- OS keeps an array of block pointers for each file
- The user or OS must declare the maximum length of the file when it is created
- OS allocates an array to hold the pointers to all the blocks when it creates the file, but allocates the blocks only on demand
- OS fills in the pointers as it allocates blocks

Example of Indexed Allocation

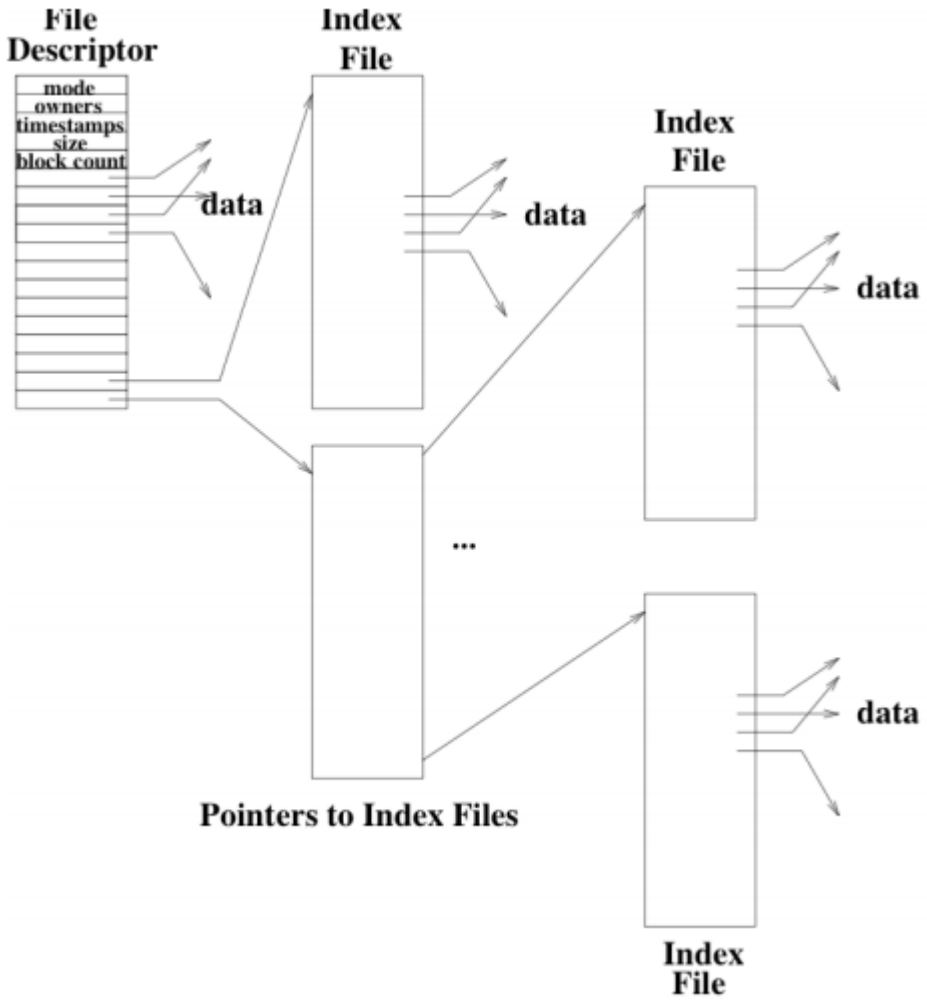


Indexed Files (cont.)

- Advantages:
  - Not much wasted space for files
  - Both sequential and random accesses are easy
- Disadvantages:
  - Wasted space in file descriptors
  - Sets a maximum file size
  - Lots of seeks because data is not contiguous

Multilevel Indexed Files

- Each file descriptor contains (for example) 14 block pointers
- First 12 pointers point to data blocks
- 13th pointer points to a block of 1024 pointers to 1024 more data blocks (One indirection)
- 14th pointer points to a block of pointers to indirect blocks (Two indirections)

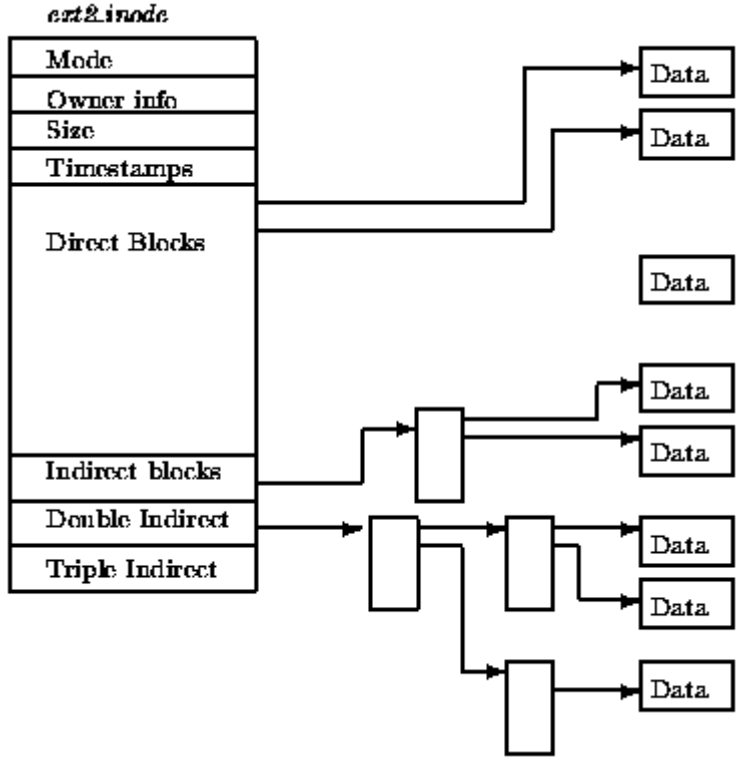


Multilevel Indexed Files

- Advantages
  - Supports incremental file growth
  - Small files?
- Disadvantages
  - Indirect access is inefficient for random access to very large files
  - Lots of seeks because data is not contiguous
- What could the OS do to get more contiguous access and fewer seeks?
- Examples: BSD Unix 4.3

The EXT2 inode

- The inode is the basic building block; every file and directory is described by one and only one inode
- Contains the following fields:
  - Mode
  - Owner info
  - Size
  - Timestamps
  - Datablocks
    - The first twelve are pointers to the physical blocks and the last three pointers contain more levels of indirections



Performance

- Best method depends on file access type
  - Contiguous great for sequential and random
- Linked good for sequential, not random
- Declare access type at creation -> select either contiguous or linked
- Indexed more complex
  - Single block access could require 2 index block reads then data block read
  - Clustering can help improve throughput, reduce CPU overhead

Free-Space Management

- Need a free-space list to keep track of which disk blocks are free (just as we need a free-space list for main memory)
- Need to be able to find free space quickly and release space quickly => use a bitmap
  - The bitmap has one bit for each block on the disk.
  - If the bit is 1, the block is free. If the bit is 0, the block is allocated.
- Can quickly determine if any page in the next 32 is free, by comparing the word to 0. If it is 0, all the pages are in use. Otherwise, you can use bit operations to find an empty block.

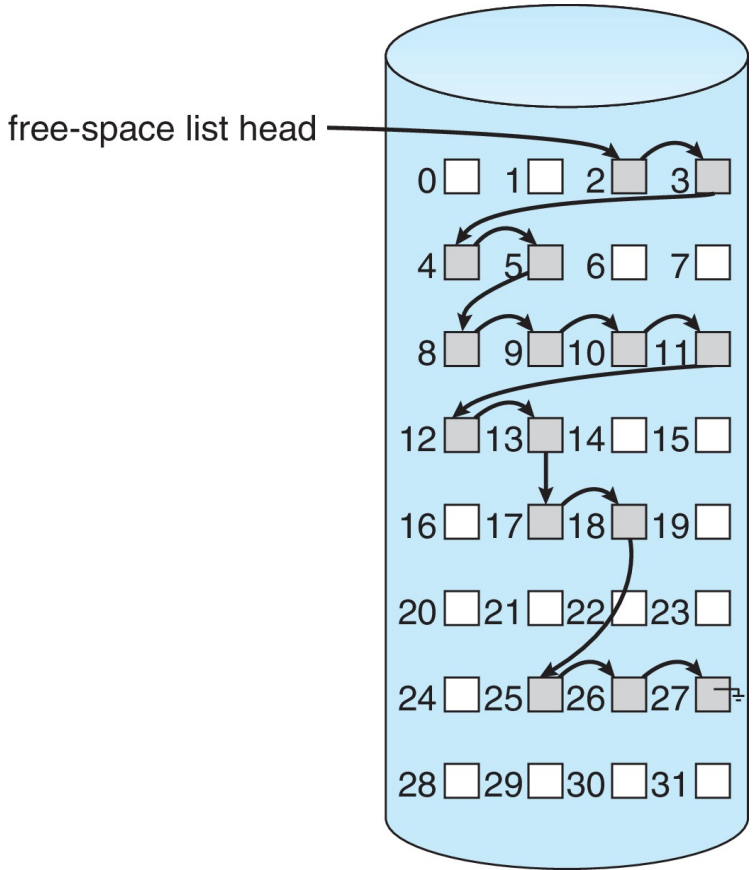
  11000010010001111110…
- Marking a block as freed is simple since the block number can be used to index into the bitmap to set a single bit.

Free-Space Management (cont.)

- Problem: Bitmap might be too big to keep in memory for a large disk. A 2 TB disk with 512 byte sectors requires a bitmap with 4,000,000,000 entries (500,000,000 bytes = 500 MB).
- If most of the disk is in use, it will be expensive to find free blocks with a bitmap.

Linked Free Space List on Disk

- An alternative implementation is to link together the free blocks.
  - The head of the list is cached in kernel memory. Each block contains a pointer to the next free block.
  - How expensive is it to allocate a block?
  - How expensive is it to free a block?
  - How expensive is it to allocate consecutive blocks?



Summary

- Many of the concerns and implementations of file system implementations are similar to those of virtual memory implementations.
  - Contiguous allocation is simple, but suffers from external fragmentation, the need for compaction, and the need to move files as they grow.
  - Indexed allocation is very similar to page tables. A table maps from logical file blocks to physical disk blocks.
  - Free space can be managed using a bitmap or a linked list.

Exit Slips

- Take 1-2 minutes to reflect on this lecture
- On a sheet of paper write:
  - One thing you learned in this lecture
  - One thing you didn’t understand

Next class

- Quiz 3
- We will briefly discuss:
  - Security and Privacy
- Reading assignment:
  - SGG: Ch. 16 & 17

Acknowledgment

- The slides are partially based on the ones from
  - The book site of *Operating System Concepts (Tenth Edition)*: http://os-book.com/