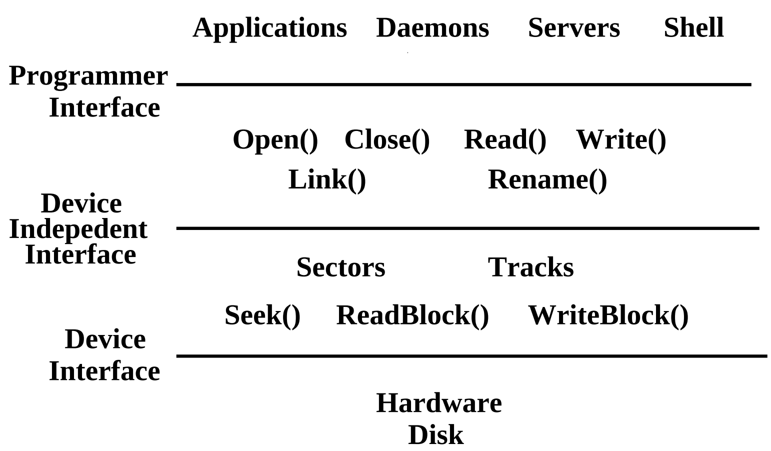
Last Class: File System Abstraction

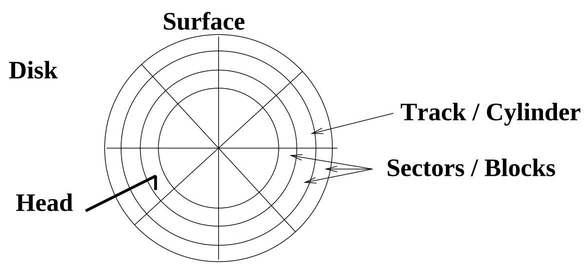


Today: File System Implementation

Disk management

- Brief review of how disks work.
- How to organize data on to disks.

How Disks Work



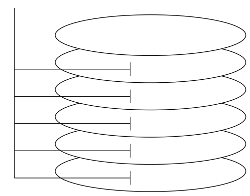
- The disk surface is circular and is coated with a magnetic material. The disk is always spinning (like a CD).
- Tracks are concentric rings on disk with bits laid out serially on tracks.
- Each track is split into *sectors* or *blocks*, the minimum unit of transfer from the disk

Disk Overheads

- **Overhead:** time the CPU takes to start a disk operation
- **Latency:** the time to initiate a disk transfer of 1 byte to memory.
  - **Seek time:** time to position the head over the correct cylinder
  - **Rotational time:** the time for the correct sector to rotate under the head
- **Bandwidth:** once a transfer is initiated, the rate of I/O transfer

How Disks Work

- CDs come individually, but disks come organized in *disk pack* consisting of a stack of platters.
- Disk packs use both sides of the platters, except on the ends
- Comb has 2 read/write head assemblies at the end of each arm.
- *Cylinders* are matching sectors on each surface.
- Disk operations are in terms of radial coordinates.
  1. Move arm to correct track, waiting for the disk to rotate under the head.
  2. Select and transfer the correct sector as it spins by



File Organization on Disk

The information we need:

fileID 0, Block 0 $\rightarrow$ Platter 0, cylinder 0, sector 0
fileID 0, Block 1 $\rightarrow$ 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.

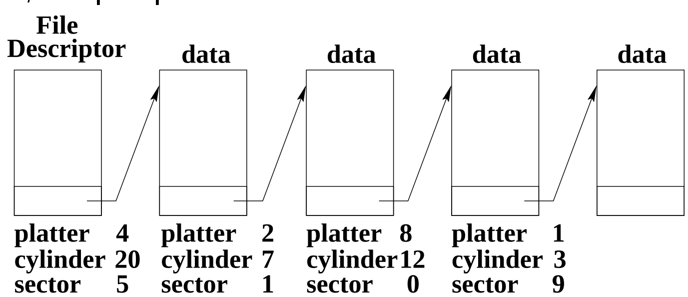
$\Longrightarrow$ The per-file cost must be low, but large files must also have good performance.

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,
  - Access time? Number of seeks?
- **Disadvantages**
  - Changing file sizes
  - Fragmentation? Disk management?
- **Examples:** IBM OS/360, write-only disks, early personal computers

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.

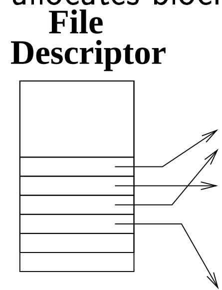


Linked files

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

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.

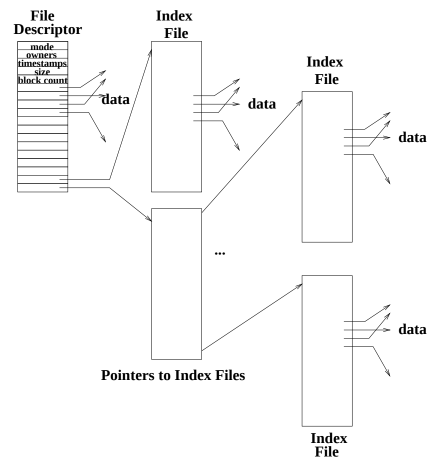


Indexed files

- **Advantages**
  - Not much wasted space.
  - Both sequential and random accesses are easy.
- **Disadvantages**
  - Sets a maximum file size.
  - Lots of seeks because data is not contiguous.
- **Examples:** Nachos

Multilevel indexed files

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



Multilevel indexed files: BSD UNIX 4.3

- **Advantages**
  - Simple to implement
  - 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.
- Is the file size bounded?
- What could the OS do to get more contiguous access and fewer seeks?

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.

  110000100100011111110...

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

BitMaps in Nachos

Bitmaps are implemented in userprog/bitmap.{h,cc}

```
class BitMap {
  public:
    BitMap(int nitems);
    void Mark(int which);
    void Clear(int which);
    bool Test(int which);
    int Find();
    int NumClear();
};
```

Using BitMaps in Nachos

- Create a BitMap for the number of items you want to track. Initially, all the items are cleared.
- To free an item, call Clear passing the index for the item.
- To allocate an item, call Find. It will search for a free item, set its bit, and return its index. Returns -1 if none are available.
- To allocate a contiguous block, use NumClear to see if enough free blocks exist.
  - If so, loop to find a consecutive number of blocks meeting your need, using Test to check each block.
  - If a consecutive block is found, use a second loop to set the blocks by calling Mark.
- A more efficient implementation of contiguous allocation could be accomplished by adding a method to BitMap to find a contiguous chunk.

Free-Space Management

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

Exam Review

Memory Management

Topics you should understand:

1. What is virtual memory and why do we use it?

2. Memory allocation strategies:
   - Contiguous allocation (first-fit and best-fit algorithms)
   - Paging
   - Segmentation
   - Paged segmentation

Memory Management (cont.)

For each strategy, understand these concepts:

- Address translation
- Hardware support required
- Coping with fragmentation
- Ability to grow processes
- Ability to share memory with other processes
- Ability to move processes
- Memory protection
- What needs to happen on a context switch to support memory management

Memory Management (cont.)

Things you should be able to do:

- Given a request for memory, determine how the request can be satisfied using contiguous allocation.
- Given a virtual address and the necessary tables, determine the corresponding physical address.

Paging

Topics you should understand:

- What is paging, a page, a page frame?
- What does the OS store in the page table?
- What is a TLB? How is one used?
- What is demand paging?
- What is a page fault, how does the OS know it needs to take one, and what does the OS do when a page fault occurs?

Paging (cont.)

- Page replacement algorithms. For each understand how they work, advantages, disadvantages, and hardware requirements.
    1. FIFO
    2. MIN
    3. LRU
    4. Second chance
    5. Enhanced second chance
- How do global and per-process (aka local) allocation differ?
- What is temporal locality? What is spatial locality? What effect do these have on the performance of paging?
- What is a working set?

Paging (cont.)

- What is thrashing and what are strategies to eliminate it?
- What considerations influence the page size that should be used?

Things you should be able to do:

1. Given a page reference string and a fixed number of page frames, determine how the different replacement algorithms would handle the page faults.

File Systems

Topics you should understand:

1. What is a file, a file type?
2. What types of access are typical for files?
3. What does the OS do on a file open, file close?
4. What is a directory?
5. What is a link?
6. What happens if the directory structure is a graph?
7. How does Unix support multiple users of shared files?
8. Strategies for laying files out on disk. Advantages and disadvantages.
   (a) Contiguous allocation
   (b) Linked
   (c) Indexed

General Skills

- You should have a good sense of how the pieces fit together and how changes in one part of the OS might impact another.
- You will **not** be asked to read or write C++ code.
- You will **not** be asked detailed questions about any specific operating system, such as Unix, Nachos, Windows NT, ...