How to organize data on disks.

Brief review of how disks work.

Disk management

Today: File System Implementation

Last Class: File System Abstraction
Bandwidth: once a transfer is initiated, the rate of I/O transfer

- Rotational time: the time for the correct sector to rotate under the head
- Seek time: time to position the head over the correct cylinder

Latency: the time to initiate a disk transfer of 1 byte to memory

Overhead: time the CPU takes to start a disk operation
3. How do we lay out the files on the physical disk?

Information?

2. What is the right data structure in which to maintain file location

I. We need to support sequential and random access.

Key performance issues:

... 

File ID 0, block 1
Platter 4, cylinder 3, sector 8

File ID 0, block 0
Platter 0, cylinder 0, sector 0

The information we need:

File Organization on Disk

**How Disks Work**

- Select and transfer the correct sector as it spins by
  - Move arm to correct track, waiting for the disk to rotate under the head.
- Disk operations are in terms of radial coordinates.
  - Cylinders are matching sectors on each surface
  - Comb has 2 read/write head assemblies at the end of each arm.
  - Disk packs use both sides of the platters, except on the ends.
  - Platters.
  - CDs come individually, but disks come organized in disk pack consisting of a stack of
Examples: IBM OS/360, write-only disks, early personal computers

- Fragmentation? Disk management?
- Changing file sizes

Disadvantages

- Access time? Number of seeks?
- Simple?

Advantages

- Need to store only the start location and size in the file descriptor
- OS allocates a contiguous chunk of free blocks when it creates a file.
- OS maintains an ordered list of free disk blocks

Continuous Allocation

Performance:

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

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

Most systems fit the following profile:

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

File Organization: On-Disk Data Structures
Examples: MS-DOS
- Number of seeks?
- Does not support which type of access? Why?

Disadvantages:
- Efficiency supports which type of access?
- File size changes?
- Fragmentation?

Advantages:

Linked files

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

- Lots of seeks because data is not contiguous.
- Sets a maximum file size.

Disadvantages

- Both sequential and random accesses are easy.
- Not much wasted space.

Advantages

Indexed F iles

Descriptor

File

OS fills in the pointers as it allocates blocks.

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

Indexed F iles
What could the OS do to get more continuous access and fewer seeks?

- Is the file size bounded?

  - Lots of seeks because data is not contiguous.
  - Indirect access is inefficient for random access to very large files.

Disadvantages

- Small files?
- Supports incremental file growth
- Simple to implement

Advantages

<table>
<thead>
<tr>
<th>Multi-level Indexed Files: BSD UNIX 4.3</th>
</tr>
</thead>
</table>

- Each file descriptor contains 14 block pointers.
- 14th pointer points to a block of pointers to indexed blocks (two indirections)
- 13th pointer points to a block of 1024 pointers to 1024 more data blocks.
- First 12 pointers point to data blocks.
Bitmaps are implemented in userProc/itmap.}"hc{

Bitmaps in Nachos

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

110001001000111111110...
operations to find an empty block.
word to 0. If it is 0, all the pages are in use. Otherwise, you can use bit
Can quickly determine if any page in the next 32 is free, by comparing the

- If the bit is 1, the block is free. If the bit is 0, the block is allocated.
- The bitmap has one bit for each block on the disk.

Need to be able to find free space quickly and release space quickly use
we need a free-space list for main memory

Need a free-space list to keep track of which disk blocks are free (just as

Free-Space Management
Free-Space Management

- How expensive is it to allocate contiguous blocks?
- How expensive is it to free a block?
- How expensive is it to allocate a block?
- The head of the list is cached in kernel memory. Each block contains a pointer to the next free block.

An alternate implementation is to link together the free blocks.

- If most of the disk is in use, it will be expensive to find free blocks with a bitmap.

Problem: Bitmap might be too big to keep in memory for a large disk.

2 GB disk with 512 byte sectors requires a bitmap with 4,000,000 entries (500,000 bytes).

A more efficient implementation of contiguous allocation could be accomplished by adding a method to Bitmap to find a contiguous chunk.

- If a consecutive block is found, use a second loop to set the blocks by calling Mark.
- Check each block.
- If so, loop to find a consecutive number of blocks meeting your need, using Test to exist.

To allocate a contiguous block, use NumClear to see if enough free blocks

and return its index. Returns -1 if none are available.

To allocate an item, call Find. It will search for a free item, set its bit,

to free an item, call Clear, passing the index for the item.

The items are cleared.

Create a Bitmap for the number of items you want to track. Initially all

Using Bitmaps in Nacos
2. Memory allocation strategies:

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

Topics you should understand:

Memory Management

Summary

- Page segmentation
- Segment allocation (first-fit and best-fit algorithms)
- Continuous allocation

Many of the concerns and implementations of the system
corresponding physical address.

Given a virtual address and the necessary tables, determine the

using contiguous allocation.

Given a request for memory, determine how the request can be satisfied.

Things you should be able to do:

**Memory Management (cont.)**

What needs to happen on a context switch to support memory management?

- Memory protection
- Ability to move processes
- Ability to share memory with other processes
- Ability to grow processes
- Compatibility with fragmentation
- Hardware support required
- Address translation

For each strategy, understand these concepts:

**Memory Management (cont.)**
What is a working set? 

What is temporal locality? What is spatial locality? What effect do these have on the performance of paging?

How do global and per-process (aka local) allocation differ?

Page replacement algorithms. For each understand how they work.

Page (cont.)
Topics you should understand:

File Systems

Page Faults.

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

Things you should be able to do:

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

Pageing (cont.)
... system, such as Unix, Ncurses, Windows NT, ... you will be asked detailed questions about any specific operating system, such as Unix, Ncurses, Windows NT, ... You will **not** be asked to read or write C++ code.

You will **not** be asked to read or write C++ code.

Changes in one part of the OS might impact another. You should have a good sense of how the pieces fit together and how...