File Systems

CSE 410 - Computer Systems
December 7, 2001

Readings and References

- Reading
  - Chapter 11, Chapter 12 through 12.6, *Operating System Concepts*, Silberschatz, Galvin, and Gagne
- Other References
  - Chapter 10, *Inside Microsoft Windows 2000*, Third Edition, Solomon and Russinovich

Files

- A user-level abstraction for "a collection of bytes in (non-volatile) storage"
- Files have:
  - Name
  - Type (implicit or explicit)
  - Location - which device, where on that device
  - Size (and possibly maximum size)
  - Protection - who may read and write this?
  - Time, date, and user identification

Disk File Structure

- Disk block is fixed-size contiguous group of disk sectors
- Think of a file as simply a sequence of disk blocks
  - may not be contiguous
- Directory is a file that points to other files or directories
- File system issues
  - how many sectors per block?
  - how do you keep track of which blocks a file is using?
  - how do you keep track of which blocks are free?
  - most files are small, but most I/O is to big files. Must optimize both

File Operations

- File creation
  - make room for the file
  - enter the new file into the directory
- Writing a file
  - specify the file and the data to write to the file
  - OS keeps track of your location in the file
  - successive writes are placed one after the other in the file

More File Operations

- Reading a file
  - specify the file and the buffer into which the data should be read
  - OS keeps track of your location in the file
  - Location pointer is often shared between read and write operations
- Repositioning within a file
  - Changes the location pointer
  - Often called "seeking"
  - No actual I/O

Yet More File Operations

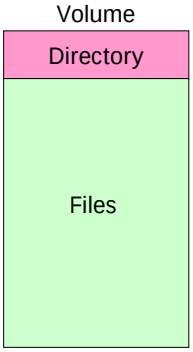
- Deleting a file
  - › Find the directory entry and delete it
  - › Mark the space used by the file as free
  - › Don't actually "erase" the file
- Truncating a file
  - › Throw away all the data in the file
  - › Keep the attribute information

Opening and Closing Files

- The above six operations are sufficient
- But we also have the notion of the ***open file***
- The open system call tells the OS that the specified file will be used by several operations
  - › user need not specify name each time
  - › OS need not search directories each time
  - › Location pointers, etc. need only be maintained for open files

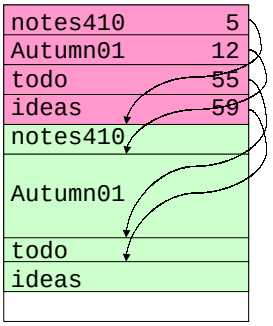
Volumes and Directories

- A volume is a logical disk
  - › there may be more than one volume per physical disk
  - › there may be more than one physical disk per volume
- The *directory* lists all of the files in the *volume*



Single-Level Directories

- In a single-level directory structure, the directory lists all files and their offsets
- Like a table of contents



Two-Level Directories

- Single-level directories suffer from *name collision*
  - › If you and I both name a file "prog1.c" then one file will overwrite the other
- Split up the space into top-level directories for each user
- Keep a directory for each user's files, and a directory of the user directories

Tree-Structured Directories

- Let directories contain subdirectories
- Arrange files in a tree
- To name a file, specify a list of directories from the top down, plus the name of the file itself
  - › This is called a *path name*
- A path beginning at the root is an *absolute path*; if part of the path is implied, it's a *relative path*

The Current Directory

- Set the *current directory* with setcwd() system call
- All future open() calls interpret path names relative to the current directory
  - › Saves on directory lookups
- Initial current directory is often set at login time, to the user's *home directory*

File Protection

- Protection allows the owner of a file or directory to define *who* may do *what* to that file or directory
  - › The *who* is restricted by user or group
    - usually use Access Control Lists (ACLs)
  - › The *what* is restricted by type of access:
    - read, write, execute

Disk Block Allocation

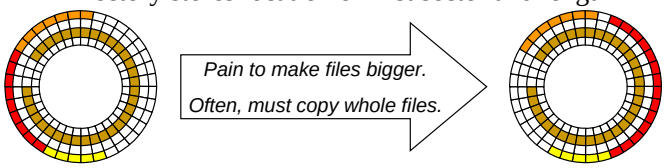
- The basic unit of storage on a disk is a *block*
  - › One or more disk sectors (which are usually 512 bytes)
- Each file is stored in one or more blocks
- For simplicity, blocks are not split between files; leftover space at the end of a block is wasted
  - › internal fragmentation
- Allocation strategy: When creating or enlarging a file, which disk block(s) should be allocated?

Contiguous Allocation

- In contiguous allocation, a file gets blocks b, b+1, b+2, ...
- Dir entry stores starting location, length
- Two blocks with sequential numbers are very likely to be in the same track, so no head movement is required
- What's the problem?

Contiguous Allocation

- Allocating blocks on one track or adjacent tracks
  - › makes accessing the file fast
- Random access is easy because offsets are easy to calculate
- Directory stores location of first sector and length

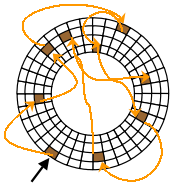


Linked allocation

- In linked allocation, a file gets a linked list of disk blocks
- Dir entry stores starting location
- Each block contains data and a pointer to the next block

Linked Allocation

- Each block contains a pointer to the next block in the file (the last block is NULL)
- Directory stores location of first and last sectors
- Advantages
  - › easy to grow files
- Disadvantages
  - › poor random access
  - › pay seek penalty many times
  - › link overhead

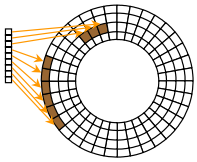


Indexed allocation

- In indexed allocation, the file gets a list of disk blocks
- An index block contains the block list

Indexed Allocation

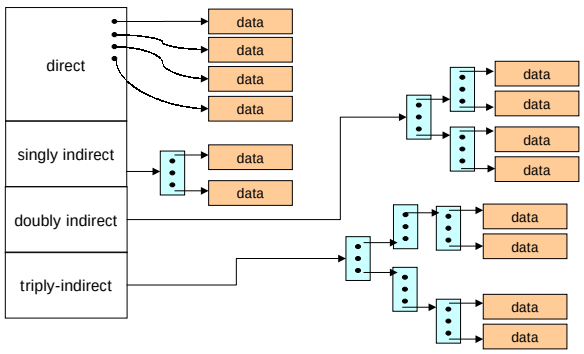
- An array lists where each block of the file is stored
- Try to allocate blocks contiguously
- But can allocate blocks anywhere
- Issues
  - › Where is this array list stored?
  - › Is the array fixed size?



Unix Inodes

- In Unix this list of blocks is stored in an **inode**
  - › for each file a directory stores the file name and an inode
- Some entries point directly to a file block
  - › these are sufficient for small files (up to 1KB)
- Some entries point to a list of block entries
  - › these are sufficient for medium sized files (up to 256KB)
- Some entries point to lists of lists of block entries
  - › these are sufficient for large files (up to 64MB)
- Some entries point to lists of lists of lists of block entries
  - › these are sufficient for humongous files (up to 16GB)

Inode Example



Free Space

- How do you find free disk blocks?
- Bitmap: One long string of bits represents the disk, one bit per block
- Linked list: each free block points to the next one (slow!)
- Grouping: list free blocks in the first free block
- Counting: keep a list of streaks of free blocks and their lengths

Sectors per Block

- What if there are many sectors per block
  - a file might fit in a single block (faster access)
  - internal fragmentation
- What if there is only one sector per block
  - increases access time because files are spread over multiple blocks

Win2K File System

- The root directory of a volume is stored at a fixed location so you always know where to start
- The MFT (master file table) stores information about each file
- Each entry is 1KB and stores
  - name, attribute, security info, data
  - a small file's data fits in the MFT entry (don't even need to allocate another block)
  - or data can be list of block ranges (similar to inodes)
- A directory is like any other file
  - it stores the MFT numbers of the files or subdirectories in that directory

Making Disks Faster

- What if a program reads just one value from a file and does some processing?
- What if a program writes results to a file in the same way?
- Ways to make disks faster
  - caching
  - minimize seeks

Disk Buffers

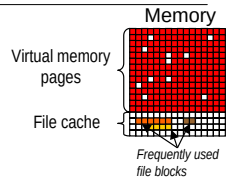
- Most files are read sequentially
- When one block is read, the disk reads the blocks that follow it because they will likely be read too
- These blocks are stored in a memory buffer on the disk
- Reads to the next blocks don't have to pay seek and rotational delay

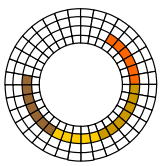
File Caches

- File accesses exhibit locality just like everything else
- Therefore cache frequently-used file blocks in main memory
  - modern file systems wouldn't work without this
- It's interesting that we use memory to store frequently-used disk blocks and disk to store infrequently used memory pages

File Cache

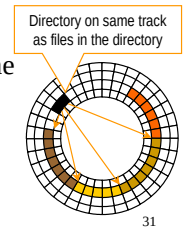
- A portion of memory is devoted to storing frequently used files
- The amount of memory changes based on the workload
  - if more files are being accessed then use more memory
- Virtual pages that are evicted from physical memory often go to the file cache before the page file
  - gives a virtual page another chance
  - doesn't require a copy because file cache can be stored anywhere in memory





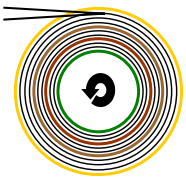
Disk Layout

- Prevent fragmentation
  - › allocate files to contiguous blocks
- Put directories and their files (and the files' inodes) near each other
  - › improves locality, reduce seek time
- Put commonly used directories in center track



Disk Scheduling

- The disk has requests to read tracks
  - › 0, 10, 4, 7 (0 is on the outside)
- If the disk head is at track 1, how should we order these reads to minimize how far the disk head moves?



Disk Scheduling

- FIFO--First In First Out
  - › lots of back and forth seeking
- SSTF--Shortest Seek Time First
  - › pick the request closest to the disk head
  - › starvation is an issue
- SCAN, C-SCAN
  - › also known as an elevator algorithm
  - › take the closest request in the direction of travel
  - › head moves back and forth from edge to edge