Storage Management

Talked about Threads, Processes, and Memory

Now for "slower memory", e.g. Disks (Chapter 11 Mass Storage Structures)

"Disks" come in a variety of flavors ...

- RAM disk -- dedicate some ram to look like disk
- SSDs (Solid State Disks)
- Hard Disk -- Many Flavors
- Optical Disks
- Network "Disks"
- Magnetic Tape?
- Thumb drives
- Communications with these devices over a "BUS" (From Chapter 12)
  - Parallel or Serial
  - Advanced Technology Attachment (ATA, parallel)
  - serial ATA (SATA, serial)
  - universal serial bus (USB)
  - fibre channel (FC)
  - various flavors of "SCSI" (Small Computer System Interface)
  - others ....

Typical disks:

- Sector -- a single unit of read/write
- Head -- a device to physical read/write on the disk
- Platter -- a side of a disk on which a head reads/writes
- Arm -- device on which heads are mounted, moves in and out
- Cylinder -- all sectors addressable without moving the arm
- Rotational speed in RPM (rotations per minute)
  - 5400, 7200, 10000, 15000 RPM
- Interesting times:
  - Transfer rate
  - seek time
  - rotational latency

Typical operations:

- Seek to a cylinder
- Read/Write a sector, select platter and sector on the platter
- More recent (LBA): Read/Write logical disk sector, no seek involved

Mass Storage

Solid-State Disks

- nonvolatile memory used like a hard drive
  - flash-memory (typically NAND semiconductors)
  - Advantages?
    - no moving parts, faster transfer, no seek time, less power
  - Disadvantages?
    - reads (fast) vs writes (slower), standard bus tech limits speed
    - max of 100,000 writes (erase, rewrite), lifespan measured in "Drive Writes Per Day"
    - (May use "wear leveling" algorithms, often implemented by the NVM controller.)
- LBA addressing is used, sometimes OS is told tracks/heads/... but they don’t exist
- SSD is starting to make rotating disks obsolete
  - 2TB ssd about $60 - $120 (10/2023)
  - 2TB rotating about $30 - $80 (10/2023)
  - 8TB ssd $500 - $1,000 (10/2023)
  - 8TB rotating $70 - $430 (Many in the $90-$180 range) (10/2023)

Tape -- still used in some places

- Sequential structure, no random access
- Transfer speeds similar to disk when ready

Mass Storage (page 2)

Disk Structure for most modern disks:

- addressed as a large one-dimensional array of logical blocks
- logical block size some power of 2, 512 usually the smallest
- bad block mapping makes it hard to map logical block to disk geometry
- recent disks -- use same linear size per sector
  - longer tracks have more sectors
  - drive speed changes as head moves in/out

Disk Attachment -- Where is the disk

- Host-Attached storage
  - "same box"
  - High-end, Fibre channel (FC)
    - multiple disks, multiple hosts
- Network-Attached storage
  - NFS, CIFS, Andrew -- network based file systems (later)
  - iSCSI -- SCSI over IP
- Cloud Storage
  - Storage on someone else’s computer
  - API based, WAN based access
- Storage Networks -- private networks not connected to internet

Mass Storage (page 2)

Host Attached Storage vs Network Attached Storage

- Network issues -- storage on network causes network traffic
- Storage Area Network -- e.g. storage devices on one NIC, LAN on another

Disk Scheduling

- Idea that you have a "queue" of disk requests
- How to best schedule them
- Light load ... no issue
- Heavy load ... how to best schedule them
  - FCFS scheduling
  - Shortest seek time scheduling
    - may cause starvation
  - Scan algorithm
    - AKA elevator algorithm
    - Circular scan
    - LOOK scheduling, look before moving the arm

Other topics

Things to read about

- disk formatting -- partitions, volumes
- bad block management
- Swap space management
- RAID (Redundant arrays of independent disks)
  - making larger virtual disks by striping (RAID 0)
    - Performance gains by parallelism
    - No redundant bits
  - making error correction/recovery by redundant disks
    - RAID 1: mirrored disks
    - RAID 2: Memory-style error-correcting codes (ECC)
    - RAID 3, 4, 5, 6: other techniques ...
- Stable-Storage -- Information is never lost
  - How to implement it?
    - multiple storage devices
    - NVRAM as a cache

I/O Hardware (Chapter 12)

OS is a hardware manager ... talked about CPU, Memory, Disks ...

- Other I/O Devices
  - transmission device (network, bluetooth,...)
  - human-interface devices (screen, keyboard mouse, audio, joystick)
  - specialized: sensor and control, ... (large variety)

- Memory mapped I/O
  - Address range communicates to devices, not real memory
  - Device Control register
  - Device Data Register
  - Device Memory -- could be large

- Techniques for I/O
  - Polling -- (assignment 1)
  - Interrupt driven -
    - Start operation, return to other stuff
    - Interrupt from I/O device
    - Interrupt processing needs to be fast
  - DMA and interrupts

Application I/O Interface

- Need an API for standard treatment of I/O devices
- Low level -- Device driver
  - Interface between Kernel and device driver
  - complete to deal with all devices
- Higher level -- user view may look like a "file"
  - UNIX - device file, (/dev/...)
  - Windows -- a device object ... that can be opened by file name
- Device characteristics
  - character-stream vs block
  - sequential vs random access
  - synchronous vs asynchronous
  - sharable vs dedicated
  - speed of operation
  - read/write properties
  - no direct user interaction ... e.g. clocks and timers
- Unix: Block and Character Devices
  - All devices look like a character device, some also look like a block device
  - Interface is slightly different between the two
- Other devices: clocks, network, ...

Ignore the rest of chapter 12, may come back later

File Systems Interface (Chapter 13)

- File System -- an abstraction on top of storage
- Typical Services
  - File abstraction
  - File manipulation
  - File protection
- Most visible service of OS
- Large code base in most OSes

File abstraction

- Bag of bits?
- known content? (e.g. is .txt for OS or users?)
  - By the OS?
    - executable files
  - By user land Tools?
    - required

File System Basics

- Standard attributes
  - Name: (symbolic, human readable)
  - Identifier: unique tag
  - Type: system tag
  - Location: where it is located on the storage
  - Size: both logical and physical size (if different)
  - Protection: who has what kind of access
  - Time, date, user identification, ...

File Operations

- Creation: Adding information
- Writing: adding information, file position pointer
- Reading: file position pointer also
- Deleting: removing information
- Truncating a file: removing information

May be many other file management routines

- renaming, moving, status, ...

Management of files in the kernel

- Open syscall: looking up information ... look up file only once
- Kernel keeps an "Open File Table" in the kernel
- Open syscall:
  - lookup file in file system (could be expensive)
  - "cache" information in the open file table
  - return a "handle", some data to uniquely represent file
- Close syscall:
  - done using the file, allow file to reclaim space
- Open and Close with shared files
  - multiple applications may open file at the same time
  - in systems with fork(), both processes have access to files
  - Typically ... two levels of tables in this case
    - Kernel wide "open file table"
    - Per process "local file table" that points to open file table
  - Kernel global open file table
    - File pointer -- offset into file
    - File-open count -- how many local file entries point here
    - Information for file location on disk
    - Access rights
  - Local table: Open flags, pointer to global open file table

Locks and File types

Locks -- shared or exclusive

- shared read locks
- exclusive locks
- mandatory or advisory
- deadlock issues here

File Types

- Kinds of data in files
  - executable, text, scripts, DataBase, ...
- How does OS know what is in the file?
  - file name ... extension (DOS, Windows)
    - .cpp -- file type?
      - C pre-processor input?
    - .app ?
      - OS X, extension on a directory!
  - extra information?
    - Mac: creator -- program that created a file

File Types (page 2)

- know how to rebuild executable files? (TOPS 20)
  - Used time information with source to executable
  - Source changed, recompile before running
- UNIX?
  - "magic" numbers to start off files
  - file(1) command

File Structure

- Executable ... OS needs to know structure to load file
  - Toy OS: OpenFile::LoadExecutable, elf.h
- Other files?
  - VMS -- knew structure of system files
  - Problem?
    - what if your app doesn’t want to use a known structure
  - Text vs Data?
  - Bag of bits?
- Mac -- Resource and Data "fork"
- Windows -- Multiple "streams" per file

- Internal structure
  - Any kind of packing?
  - Standard encoding?
    - Line in a text file? NL, CR/NL, CR
    - MPE/3000: text file, 80 character lines, all chars present

Access Methods

User level access to the file:

- Sequential (UNIX: read/write)
  - "tape model"
  - Sequential access
  - Possibly do "skip +/-n records" (seek)
  - Rewind
  - Go to end
    - (Tape model, multiple files per tape, double EOF => EOT)

- Direct (relative access) (UNIX: pread/pwrite)
  - Each read/write includes "record" number
  - Each number is a "relative record" number to start of file

- Should an OS provide both?
  - How about sequential access using direct files?
    - like UNIX: keep a file pointer
  - How about direct access using sequential files?
    - very bad!

Access Methods (page 2)

Other Access Methods?

- Hash table?
  - e.g. Key/Data pairs as basic storage element
  - Also can be stored by trees
- Index file -- keep keys, pointer to data
- IBM ISAM -- indexed sequential-access method
  - two level of indexes to access file

General Disk Structure

File system may depend on storage

- RAM disk -- short life, temp file systems, simple structures
- Collection of disks -- long life, reliable, error protection, hot swapping
- Large disk, subdisks (minidisks, partitions, slices)
  - Allows multiple kinds of file systems on one disk
- Special kinds of file systems?
  - procfs -- a file system interface to "process manager"
  - ZFS -- a "pool" based "general file system"
  - coda, smb, afs, nfs ... -- network file systems
- Volume -- contains a FS.
  - May be anywhere from part of a disk to multiple disks

Directory overview

- Directory Operations
  - lookup (search)
  - add (create)
  - delete
  - list
  - rename
  - traverse the file system

Directory Structures

Single level directory

- RT-11, small disk

Two level directory

- user/file -- top level contains no files
- Or volume:/user/file

Tree structured directories

- current directory, absolute path, relative path

Acyclic Graph structured

- Directory have just "links" to files or directories
- single file can appear in many directories

General Graph structured

- Acyclic?
- Livermore Timesharing System ... full graph
  - traversal algorithms had to detect cycles

Data stored in Directory Entry

- Full information: e.g. DOS
- Pointer to full information: e.g. UNIX UFS

Volume access

Each file system is placed on a "volume"

Multiple volumes to access, How?

- DOS/Windows (in USER space)
  - volume ID
  - path within that volume
  - User needs to see the volume
- UNIX -- File System "mount"
  - Associates a directory on one file system with the root of another
  - System mounts one file system as "Root"
  - Other file systems are mounted on directories of Root
  - User does not need to see mounts
  - User does not need to know file system types
  - Automounting ...
    - to the desktop (Mac)
  - Windows?
    - internally does mounts
    - exposes volume via special "mounts"
    - now allows full mounts

File Sharing

On the same OS with multiple users

- need protection and sharing to be considered
- what kinds of sharing
  - read only sharing?
  - read/write sharing?

Remote file systems

- NFS, DFS, SMB, FTP -- different kinds of files
  - (Some systems can "mount" remote files via ftp.)
- sshfs -- an integrated solution for ssh access to files
- Lots of issues in remote file systems -- not much here yet
- client-server fs peer-to-peer
- authentication systems ... distributed naming services ...
- larger number of failure modes

File consistency

How are files shared ... how do reads and writes interact

- Immutable-Shared-Files semantics
  - Once shared, a file can never change
- Session Semantics
  - File gets a "snapshot" at open
  - Changes are not committed until close
  - Changes are not visible unless opened after a close
- UNIX Semantics
  - writes are visible immediately to any process with an open file
  - allow processes to interfere with each other.

- Network file systems have done all 3.
- NFS -- UNIX
- AFS, Coda -- mostly session semantics
  - (process on the same machine get UNIX semantics)
- SPRITE (Berkeley, very old) -- read only shared

Protection

reliability -- safe from physical damage

protection -- safe from improper access

Protection may depend of use of file system

- Operations to control: read, write, execute, append, delete, list, change attributes ..
  - Possibly others ... rename, copy, create
- Special directories ...
  - take and give directories at LLNL

Approaches to access control

- Access Control Lists
  - each file has a list of users and allowed operations
  - not on the list? no access
  - Drawback?
    - Long lists

Protection (page 2)

- Domain based access:
  - Owner, Group, Universe
  - Each file has protection for each domain
  - Access checks user’s domain membership
  - Drawback?
    - Hard to select a single user

Typical implementations

- Primary protection by domain
- Secondary protection by ACLs

Examples:

- UNIX: primary protections: read, write, execute
- NT: full control, modify, read&execute, read, write, ...
  - ACL "who" can be a domain or a user
- DOS: nothing!
- Variety of ways to set these:
  - NT: typically a GUI
  - Solaris: has both UNIX and ACL
    - getfacl(1) and setfacl(1)

Read 13.5 Memory-Mapped files ... we talked about them earlier

File System Implementation (Chapt 14)

Typically file systems are stored on disks of some kind ...

They provide:

- rewrite: read data, modify, write back to same location (Not ZFS)
- random access to any block of data ... may take time

Basic File Systems -- Typical hardware components

- Disk
- Device Driver -- knows how to control disk
- Basic File System -- uses Device Driver to operate, manages buffers, caches
- File-organization module -- knows about file structure
- Logical file system -- manages meta-data information
  - meta-data -- data about the file, size, date, ...
- Management of open files ...
  - Idea of a Virtual File system ...
  - One interface to ALL file systems implemented by OS
    - UNIX V-node
    - All file systems implement same API for OS to use
    - Core OS knows nothing about actual FS detail
- Best if implemented as a layers of "independent" subsystems

File System Implemenation (page 2)

FUSE -- more recent Abstraction ...

- Implementation of a file system in user space
- OS passes API calls to user space
- User space program (daemon) implements FS

On Disk Structures Vs In Memory Structures

- On Disk:
  - Total information to access all data
- In Memory:
  - Caches of On Disk information
  - Dynamic information:
    - Mount information
    - Open files and file pointers
    - per-process information (file handle, file descriptor)
- Issue:
  - Keeping data in memory in sync with disk
  - partial writes to disk in case of OS failure

Typical Disk Structures:

- Boot control "block" -- information needed by ROM/OS for boot
- Volume control "block" -- core information on FS
  - UFS: superblock, NTFS: master file table
- Directory Formats
- FS block management structures
- File/Directory block management

Directory Implementation

Directory:

- Keeps names of files with method to lookup meta-data
- Simple Method: linear
  - Fixed or variable sized entries
  - Entry data depends on kind of FS
  - Search time O(n), n number of entries
  - Insert/Delete time?
- Hash table:
  - O(1) search time, insert, delete time
  - collision techniques?
  - base hash table size
  - dynamic issues hash tables
- Some kind of tree storage:
  - trees in a linear file?

Allocation methods

Allocation of data blocks (sectors) for files

- Simple: Contiguous Allocation
  - Define a linear ordering of sectors
  - File starts at LBA (logical block address) X
  - data contained in next Y blocks
  - Issues?
    - random access -- easy
    - sequential access -- easy
    - dynamic file size -- hard
    - creating a new file, unknown space needs
      - Start in largest block
    - extending a file -- hard
    - ends up with external fragmentation
    - may need a de-fragmentation function
      - Live or offline?
- Used by RT-11, PDP-11 computers

Allocation methods (page 2)

- Linked Allocation
  - directory/meta-data has first block address
  - each block has a "next block" address in the block
  - Issues?
    - creating -- easy
    - writing/extending -- easy
    - sequential access -- easy, may take longer than contiguous
    - random access -- hard
    - ends up with internal fragmentation
    - dynamic file size -- easy
    - data in each sector is less than sector size
    - reliability?
      - data corrupted (link) => lose the remainder of file
      - Doubly linked list?
      - Store filename, block number?

Allocation methods (page 3)

- FAT -- File allocation table (MSDOS, OS-2)
  - array of block numbers, one for each data block on FS
  - links are in the FAT, no loss of data on disk
  - not allocated: 0 entry or on a free list
  - Disk reads for FAT and file

Allocation methods (page 3)

Indexed allocation

- Block of "pointers to data blocks"
- Each file has its own index block
- Directory has pointer to index block
- Issues?
  - Create, read, write, append, random access .... easy
    - Run out of space in index block?
  - Small files ... lots of wasted space in index block
  - Small index blocks ... small files
  - linked scheme, last entry in index block is to next index block
  - multi-level index scheme, top level points to index blocks ...
  - UNIX UFS combined method
    - small index block, one level regular index block, 2 & 3 level ...

FS Performance ... a major component of "system feels fast"

- FAT/NTFS systems -- De-fragmentation -> get files closer to contiguous
- Berkeley's changes to UFS for FFS
  - Allocate file in the same cylinder, not just contiguous
  - Other disk related tweaks .... of which many are not valid any more

Free-Space Management

Free disk space management needs to be done

- Keep track of unallocated blocks
- May use unallocated blocks to help keep track
- Bit Vectors
  - one bit per FS block
  - 0 allocated, 1 free
  - Advantage
    - compact
    - ffs (find first set) instructions
  - Disadvantage
    - large bit maps (e.g. 1TB file system)
    - ffs instructions need all bits in memory

- Linked List
  - Either in the Disk Blocks or the FAT
  - Advantage -- relative easy
  - Disadvantage -- May be hard to allocate from same cylinder ...

Free-Space Management (page 2)

Counting (aka run length encoding)

- Free blocks usually come in groups
- Linked list has first block, number of blocks free
  - Advantages
    - An empty disk has one entry in the list.
  - Disadvantages
    - Turns into simple linked list after much use
- Space Maps
  - Sun’s ZFS -- designed for a huge number of files
  - Can include multiple file systems
  - Meta Data I/O is of importance
  - Divides space into meta-slabs each with a spacemap
  - One spacemap easily fits into memory ... read, modify, write
  - ZFS also depends on transaction processing and log file systems
    - more later on log file systems
- TRIMing unused blocks
  - NVM flash-based, writing is very slow
  - Tell device a block is no longer in a file so it can be erased
  - Management of free "lists" when rewrite is expensive

Efficiency and Performance

Disk is the major bottleneck in OSes.

- name lookups can be expensive
- space allocation can be costly
- Size of pointers to files => space used to store them
  - 16, 32, 64 bit pointers
  - ZFS: 128 bit pointers
- reading and writing can cause system to slow down
  - e.g. write a block, now need it again
  - (page out, page fault is an example)
- Buffer cache
  - Cache of Disk blocks Read/Written
  - Page cache and FS cache VS Unified buffer cache
  - LRU replacement algorithm in cache
- Synchronous vs Asynchronous writes
- Read Ahead for buffer management of read files

File System Maintenance

File de-fragmentation

- Why needed?
- Which FSes need this?

File system consistency checker

- diskchk in DOS
- fsck in UNIX
- Make sure all structures are correct and complete
  - Free inodes and Used inodes add up to total
  - Free blocks and Used Blocks add up to total
  - File meta-data matches reality (e.g. nlinks)
  - All files (inodes) are reachable in directory tree
- (ToyFs needs a fsck program! or a check option to the toyfs program)

Log-Structured File Systems

- DB style transactions as applied to file systems
- Tries to make sure that we never need to repair much
- Basic Idea
  - Write to the "log" what will be done (e.g. metadata)
  - Do what you said
  - Write to log you have done it.
  - Log can be a circular buffer of appropriate size
  - At "recovery time" can see that a log entry was not finished
    - Abort or reply entry
  - Log writes are sequential and thus very fast

- Used in many file systems now, NTFS, LFS (BSD), ext3fs, FFS (BSD)

Other types of things have been used to improve speed and reliability

- ZFS -- snapshot, never overwrites blocks, no FSCK ...

Backups -- another way to preserve your FS data

- Full backups vs Incremental backups

Read 14.8 (WAFL)

File System Internals

Kinds of file systems

- general-purpose -- files, directories -- on long term storage
- tmpfs -- a file system in main memory
- objfs -- a "virtual" file system, access to kernel symbols
- ctfs -- a virtual file system, "contract information"
- lofs -- a "loop back" file system
- procfs -- a virtual file system with system information, process information
- ufs, ffs, extXfs, zfs -- general purpose file systems

File-System Mounting

- Toy Fs constructor -- read the first sector, get ready to use
- General term for that is mounting
- Mount point ... place to access the file system
- DOS/Windows:
  - drive letter:\path\to\file
- UNIX/Linux:
  - mount on a directory (usually empty, hides directory contents)
  - mount various kinds of file systems
  - Linux: gio allows users to mount smb file systems