Design and Engineering of Computer Systems

Lecture 16: Filesystem Datastructures

Mythili Vutukuru

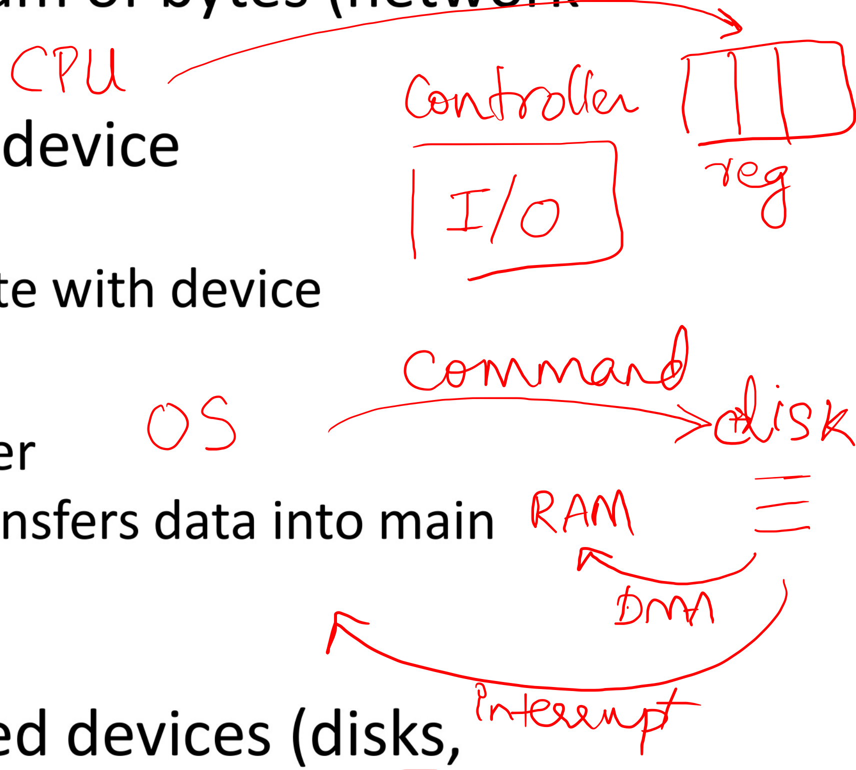
IIT Bombay

Recap of the course so far

- What are computer systems?
- Principles of designing computer systems
  - Modularity, abstraction, virtualization, fixed size allocation, caching, ..
- Overview of system hardware: CPU, memory, I/O devices
- How OS runs processes on CPU, how VMM runs VMs
- How OS manages memory, what happens on a memory access
- This week: how OS manages I/O devices
  - How OS stores files on hard disk or other secondary storage (filesystem)
  - How data is sent and received through the network (network stack)

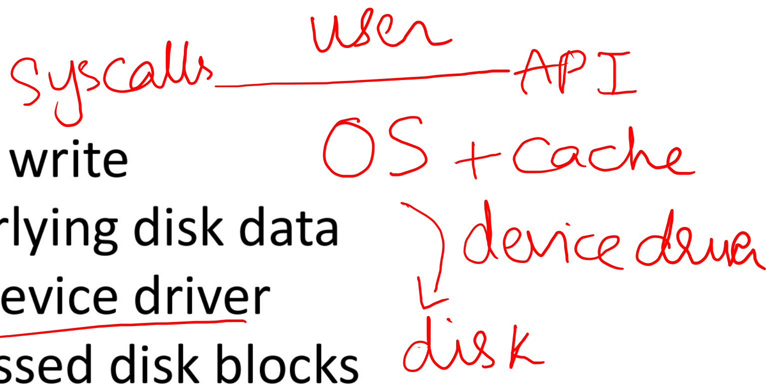
Recap of I/O devices

- I/O devices: expose block storage (hard disk) or stream of bytes (network card, keyboard, …)
- Device controller: microcontroller that manages I/O device
  - Exposes various registers: command, status, data
  - Device drivers reads/writes these registers to communicate with device
- Example: reading data from disk
  - Device driver gives command to read via command register
  - Device controller executes disk read on disk hardware, transfers data into main memory via DMA, raises interrupt
  - Device driver handles interrupt, processes received data
- DMA reduces I/O overheads, especially for high speed devices (disks, network cards)



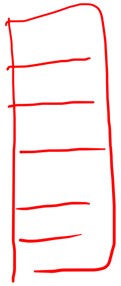
Filesystems

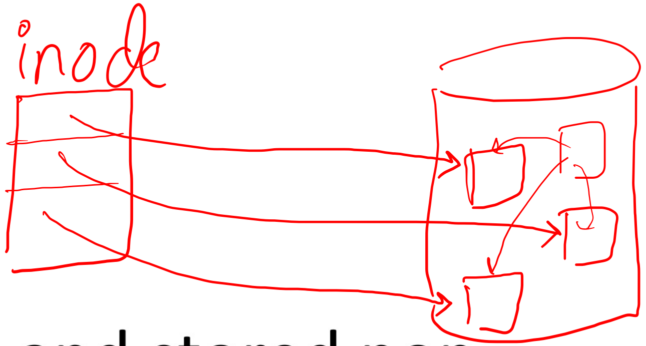
- File: persistent storage of user data on secondary storage (hard disks, ..)
  - Traditional hard disk stores file data in fixed size blocks
- Filesystem: OS code that manages files on disk
  - User programs invokes system calls to access files: open, read, write
  - OS implements system calls and performs operations on underlying disk data
  - OS system call implementation accesses blocks on device via device driver
  - OS also manages disk buffer cache which caches recently accessed disk blocks



- A filesystem is a specific way of storing and organizing file data on disk
  - Many ways to organize data, many filesystems exist (e.g., ext3, ext4, FAT)
- This lecture: Data structures used in a simple filesystem
- Next lecture: Implementation of filesystem-related system calls

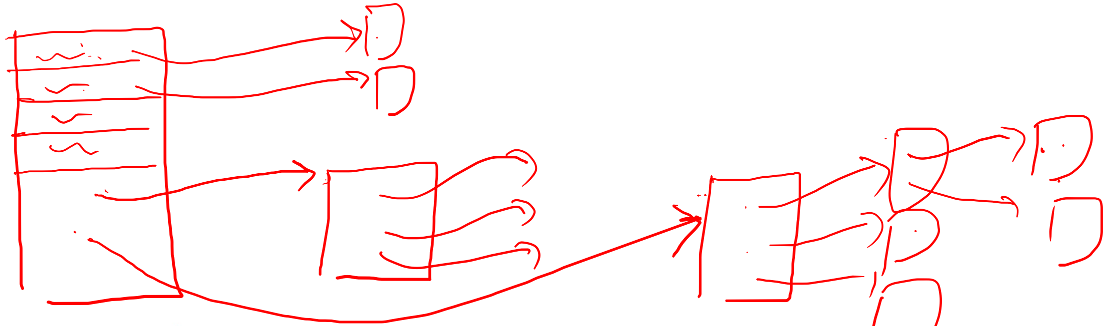
Index node (inode)





- Files are variable sized, split into fixed size blocks and stored non-contiguously on disk
  - Much like how memory image of process is split into fixed size pages
  - Fixed size blocks avoids external fragmentation of disk storage
- For every file, index node (inode) keeps track of all the block numbers (locations on disk) where the file data is stored
  - Equivalent to a page table which keeps track of physical frame numbers
- Inode of a file is also stored in one or more disk blocks
  - Much like how page table is stored in one or more pages hierarchically
- Inode stores all metadata about file (size, permissions, time of last access/modification, disk block numbers of file data, ..)

Structure of inode



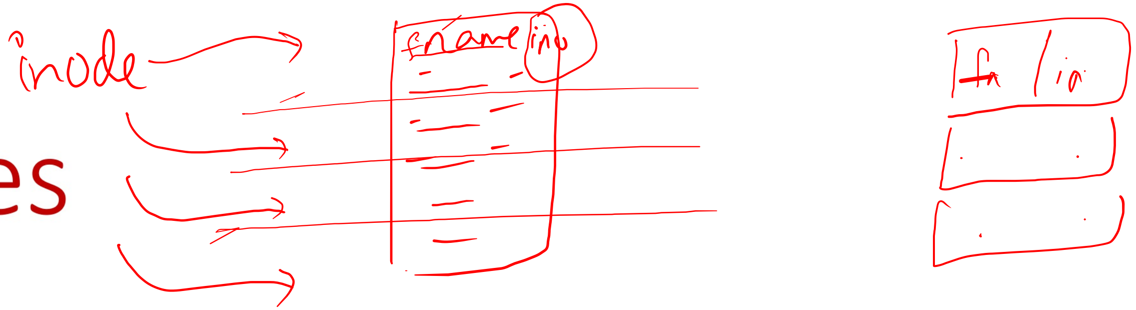
- A file is uniquely identified by its inode number on disk
  - An inode number uniquely identifies disk block containing inode on disk
- How are block numbers of file data blocks stored in inode?
  - All block numbers of a file may not fit in one inode disk block
- Hierarchical method of storing block numbers in inode
  - Inode contains the block numbers of first few blocks of a file (direct blocks)
  - If direct blocks are full, inode contains block number of single indirect block, which contains block numbers of next few blocks of a file
  - If single indirect block is full, inode contains block number of double indirect block, which contains block numbers of more single indirect blocks
  - Triple indirect block can also be used for large files
- Not a symmetric hierarchical structure like page table
  - Most files are small, so first few block numbers of a file are made available easily without accessing multiple levels of inode
- Accessing a file from disk may require multiple disk accesses for inode

Limitations on file size

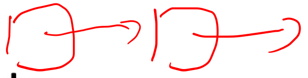
- Filesystem metadata imposes limits on maximum size of file that can be stored on filesystem, maximum number of files, maximum disk size that can be managed, and so on..
- Example: limit on file size imposed by inode structure
  - Suppose inode can store K direct blocks, one single, double, triple indirect block each
  - Suppose single indirect block can store N block numbers, double indirect block can store block numbers of N single indirect blocks, triple indirect block can store block numbers of N double indirect blocks
  - Maximum file size = K + N + N^2 + N^3 blocks

- Different file systems differ in these limits

Directories

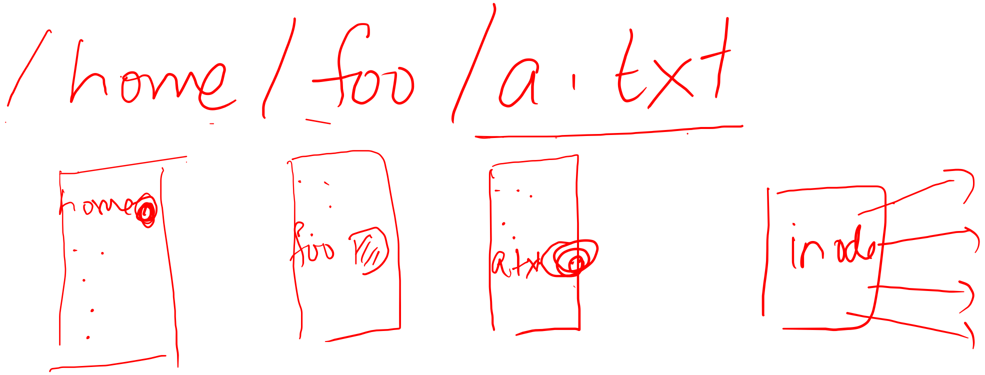


- Directory is also a special kind of file in Linux-like operating systems
  - File type in inode identifies if regular file or “directory” file
- Directory is a “file” which contains special data: names of files or sub-directories located within it, and their inode numbers
  - Data blocks of directory store these mappings between names and inode numbers
  - Inode of directory keeps track of these data blocks of directory
- How are filename→inode number mappings stored in directory data blocks?
  - Can store fixed size records containing name and inode number



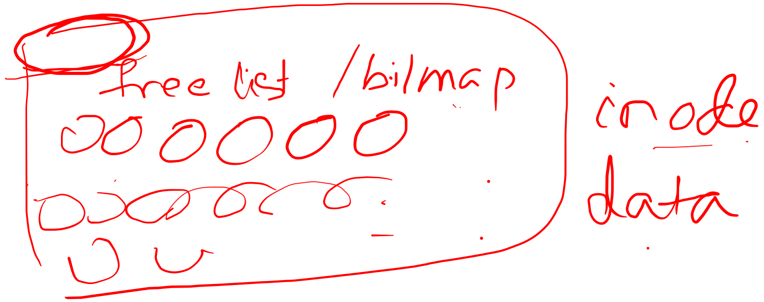
  - Linked list, binary search tree, or other datastructures can be used to
- How to lookup a file in a directory?
  - Fetch inode of directory, locate its data blocks, read data blocks
  - Search for filename in data blocks of directory (traverse array/linked list/binary search tree) and retrieve inode number of file

Pathnames

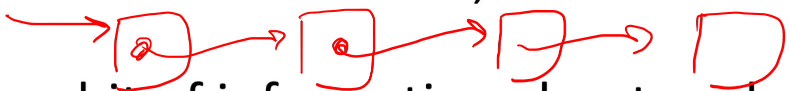


- File identified in filesystem by its pathname: series of directories, starting at root dir, leading to a file in the root filesystem
- When we want to open/read/write file, we need to find its inode number (from which we can retrieve file data) using pathname
- Given a pathname of file, how to locate its inode number?
  - Start with root directory inode (well known)
  - For every element (directory) in pathname, read directory data blocks, lookup next element filename in directory, retrieve inode number of next element
  - Repeat above process recursively, until entire path name is traversed and we find inode number of the desired file in its parent directory

Disk layout of filesystem

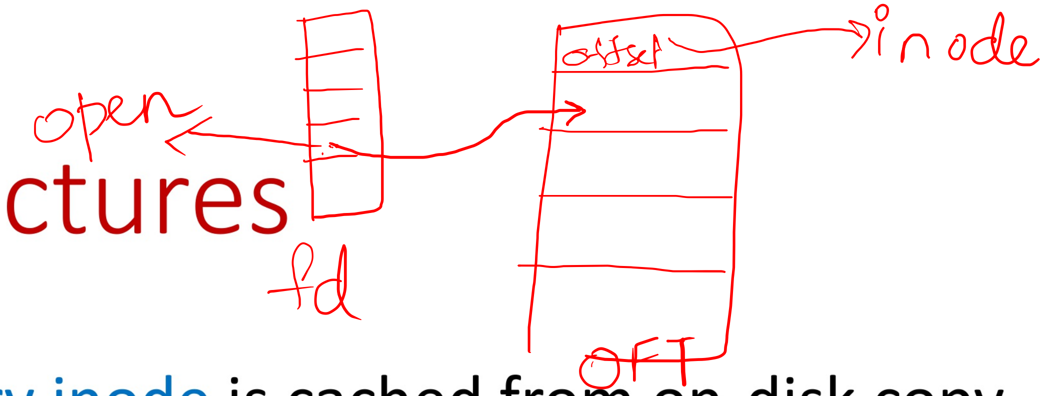


- What all does hard disk have?
  - Data blocks of files and directories
  - Inode (metadata) blocks of files and directories
  - Information about which data/metadata blocks are free and which are occupied
  - Overall master plan of disk is stored in the first block: superblock
- Free space management: how to know which blocks are free?
  - Free list: superblock contains disk block number of first free block, which contains block number of next free block, and so on..

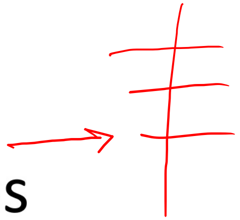


  - Bitmap: few blocks on disk contain bitmaps, one bit of information about each disk block, whether free or not
- All this layout is done when a hard disk is "formatted" with a filesystem
  - Different filesystems will have different layouts, formatted differently
  - Maximum number of files, maximum disk size etc. depend on this format

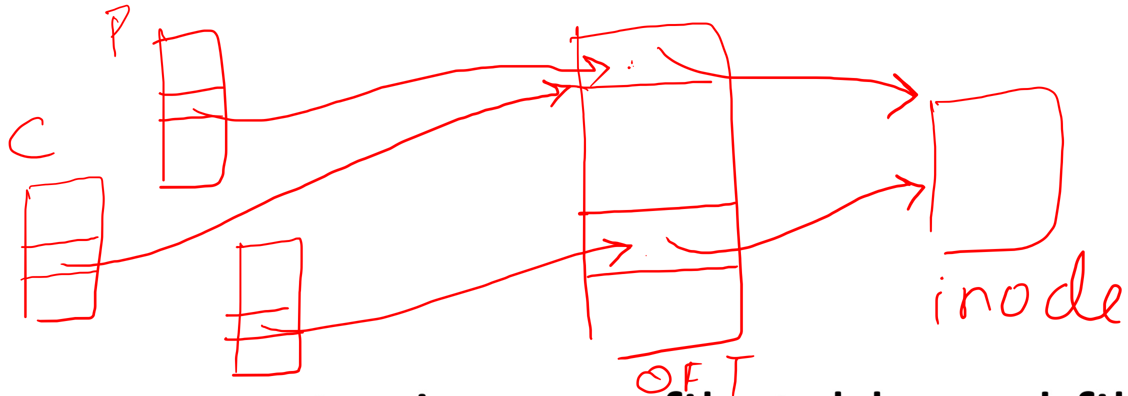
In-memory data structures



- When a file is opened, in-memory inode is cached from on-disk copy
  - Quick access of file data block numbers as long as file is in use
- Open file table: data structure used by kernel to keep track of open files
  - One open file table entry created for every open system call
  - Contains pointer to in-memory inode and other information about open file (e.g., offset at which the file is being read/written)
- File descriptor array: per-process array of open file table entries for files that are opened by the process (part of PCB)
  - When you open a file, open file table entry is created, its pointer is stored in file descriptor array and index in array is returned as file descriptor/handle
  - OS can locate file inode for reading/writing using file descriptor
- Disk buffer cache: LRU cache for recently read blocks from disk
- Next lecture: how all these data structures are used in implementing system calls



Open file table



- Every open system call creates new entry in open file table and file descriptor array
- Suppose same file opened by two separate "open" system calls
  - Will result in separate entries in open file table, and file descriptor array, because offset of reading/writing is different
  - Multiple open file table entries can store pointer to same inode of the file
- Exception: if parent forks child process, file descriptor array of parent is duplicated in child process
  - Parent and child file descriptor arrays point to same open file table entries
  - Offset of file reading/writing are shared between parent and child
  - Usually one of them should close the file for correct operation

Summary

- In this lecture:
  - Introduction to filesystem in OS
  - Inodes and disk layout of filesystem
  - In-memory datastructures
- Find out the name of the filesystem(s) in the computer that you are using. What are the most popular filesystems in use today?