PRINCIPLES OF COMPUTER SYSTEMS DESIGN

CSE130

Winter 2020

File Systems III - Allocation & Free Space

Baskin Engineering
UC SANTA CRUZ

Notices

- **Assignment 5** will be available **Sunday March 8**
- **Week 10 Lectures** (Final Prep) will ***not*** be webcast
- Assignments 1 to 4 - Grades available this week
- Late submissions are ignored by the automated grading system
  - If you e-mailed me an extension request and it was granted, see me in Week 10 for a manually assigned grade

Today's Lecture

- Open File Tables
- Space Management
- Allocation Algorithms
- The Network File System

- Introduction to Assignment 5

Everything is a File (descriptor)

- **Everything** in Unix and Unix-Like Operating Systems is exposed via the **file system name space**
- File type examples
  - File
  - Directory
  - Symbolic Link
  - Named Pipe
  - Network Socket
  - Device
- An elegant, simple, unified approach; access rights are easy to understand and implement
- When a file is opened by name (via the `open` system call), a **file descriptor** is returned to the user program
- All subsequent access to the file is done by passing the file descriptor to system calls like `read`, `write`, and `close`

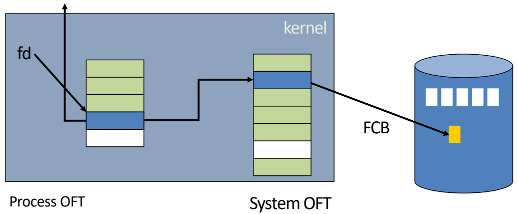
System Call : `create` (in UNIX "`creat`")

- Application program calls the logical file system
- Logical file system:
  - Allocates a new **File Control Block** (FCB)
  - Reads appropriate directory into memory
  - Updates it with new file name and FCB
  - Writes directory back to disk
- **Open File Table** (OFT)
  - One for whole system
  - One per process (not per thread)

File Control Block : Example Fields
Permissions: owner, group, ACL
Dates (creation, last access, last write)
Size
Disk location(s)
etc.
etc.

System Call : `open`

Program: `fd = open(fname, access_mode);`

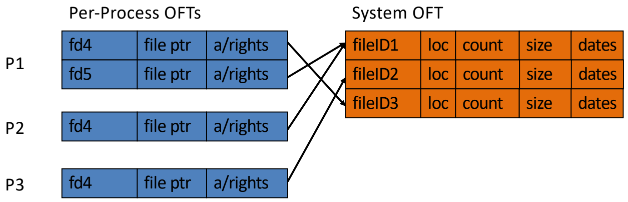


Process OFT
- System wide OFT pointer
- File Pointer
- Access Mode

System OFT
- FCB
- Times
- Buffers
- Open Count
- Lock(s)

Open File Tables

- Repeated searches of directory expensive
- Cache entry to reduce cost
- **Multiple Processes can have the same file open**



Per-Process OFTs

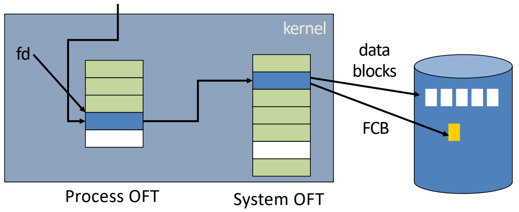
P1	fd4	file ptr	a/rights
	fd5	file ptr	a/rights
P2	fd4	file ptr	a/rights
P3	fd4	file ptr	a/rights

System OFT

fileID1	loc	count	size	dates
fileID2	loc	count	size	dates
fileID3	loc	count	size	dates

System Call : `read`

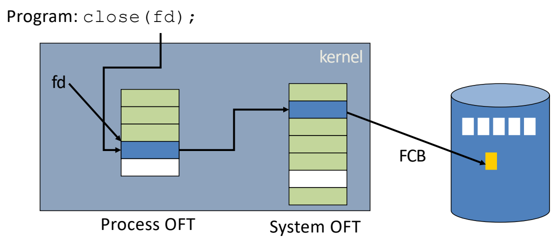
Program: `read(fd, …);`



Process OFT

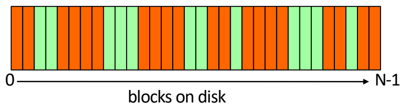
System OFT

System Call : `close`



Space Management

- How do we allocate chunks of disk space to files so that we make best use of all available space?
- How do we provide good **performance** and **reliability**?
- Allocation methods refer to how disk blocks are allocated for files:
  - Contiguous?
  - Linked?
  - Indexed?
- **Free Space Management**

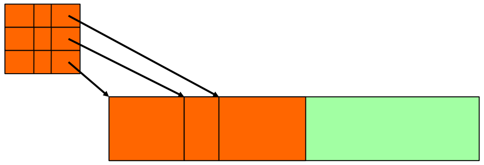


Bitmap

```
11001111
00011110
01101111
00011011
```

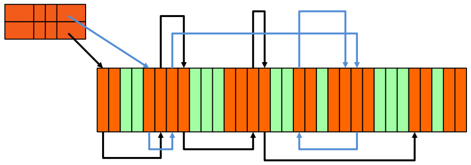
Contiguous Allocation

- File defined by base (start block) and length
- Sequential and direct access supported
- Difficult to allocate space or increase file size ☹



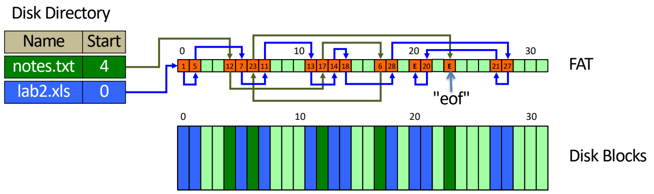
Linked Allocation

- File defined by first and last
- Solves storage problem - any free block will do
- Direct access not supported (efficiently) ☹



FAT : File Allocation Table (Windows)

- Variation of **Linked Allocation**
  - Small array of block numbers linked from start to “eof” (end of file)
- Space set aside at the start of each disk volume for the FAT
- Very efficient if FAT is cached (but synchronisation with disk copy is a problem ☹)
- Follow links in the table to find all the blocks allocated to a file

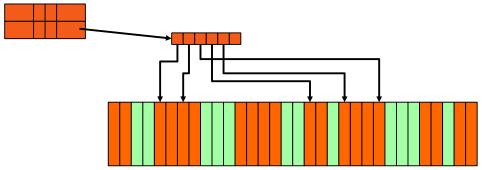


Disk Directory

Name	Start
notes.txt	4
lab2.xls	0

Indexed Allocation

- File defined by index
- Supports sequential and direct access
- Solves large file storage problem by introducing the overhead of index blocks ☺ ☹

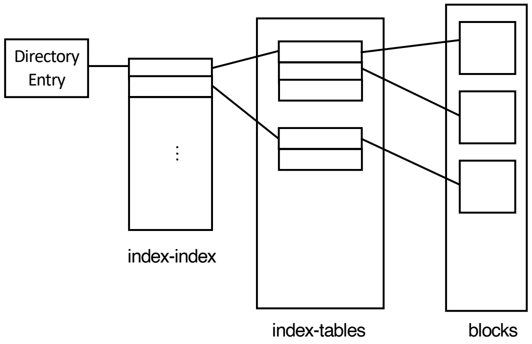


Problems?

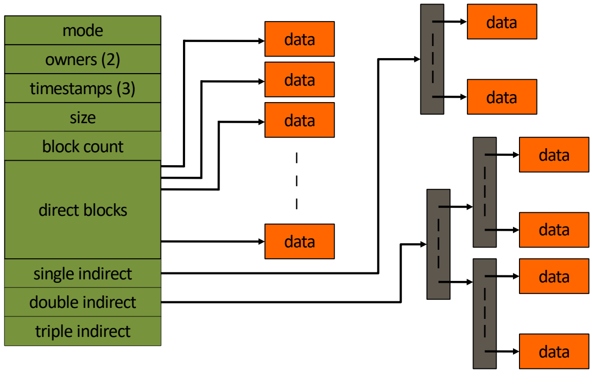
Indexed Allocation

- Need index table
  - What size should the index be?
  - Small is preferred to lower overhead, but
  - Cannot index large files
  - Can dynamically chain index blocks to extend the table
  - Requires following the last entry to next index table
- Could try multilevel index, or
- Combined scheme (index + links)

Multi-Level Allocation

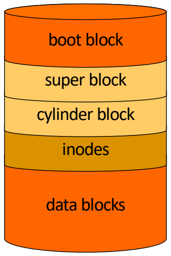


The i-node



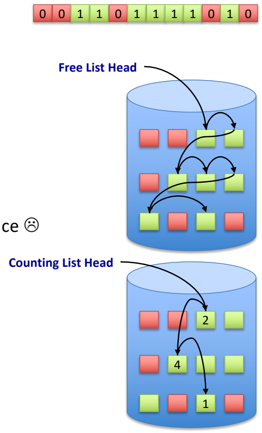
Example : The Unix File System

- Boot block
  - Operating System Bootstrap
- Super block
  - Static partition information
- Cylinder block
  - Free block bit map(s)
  - Allocation statistics
- i-nodes
  - Fixed number
    - **Run out of i-nodes and the file system is “full”, even if the disk is not ☹**
- Data blocks
  - Raw data of user files



UNIX FS : Free Space Management

- **Free Block Bit Map**
  - Requires additional space. E.g.
    block size = 2^{12} bytes (4KB)
    disk size = 2^{30} bytes (1GB)
    no. of blocks = bitmap size =
    2^{30} / 2^{12} = 2^{18} bits = 32KB
- **Free List** (a linked list)
  - Cannot easily get a sense of contiguous space ☹
  - But less wasted space ☺
- **Counting List**
  - A disk block address and a count
  - Reduces the length of the free list ☺



UNIX FS : Efficiency & Performance

- Efficiency dependent on:
  - Disk allocation and directory algorithms
  - Types and size of data kept in file’s directory entry
- Enhanced Performance
  - **Disk Cache**
    - Separate section of main memory for frequently used blocks
  - **Read-Ahead** and **Free-Behind**
    - If it appears user code is reading a large file sequentially (a very common action)
      - **Read into cache the next few disk blocks, even if they haven’t been asked for yet**
      - **Also remove from cache the blocks that have already been dealt with, even if the user code has not explicitly let go of them by closing the file**
  - **Seek Scheduling**
    - Covered on later slides

File Systems : Self Study

- Tanenbaum & Bos : Section 10.6 - The Linux File System

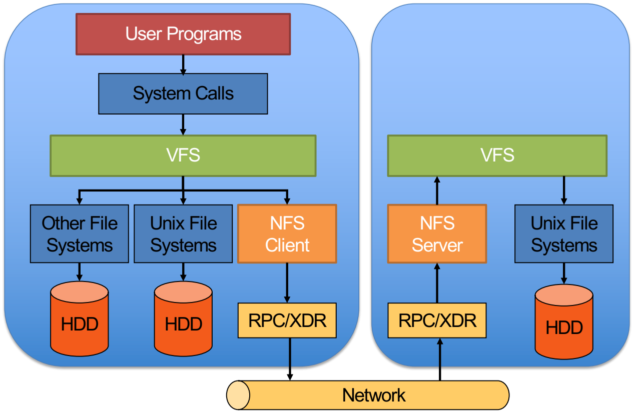
NFS : The Network File System

- Interconnected machines sharing file systems in a transparent manner (via UDP)
  - A **remote** directory is mounted **locally**
  - The mounted directory looks like an integral sub-tree of the local file system, replacing the sub-tree descending from the local directory
  - Specification of the remote directory for the mount operation is **nontransparent**; the host name of the remote directory has to be provided
  - Files in the remote directory can then be accessed in a transparent manner
  - **Subject to access-rights accreditation, potentially any file system (or directory within a file system) can be mounted remotely on top of any local directory**

NFS : The Network File System

- NFS is designed to operate in a **heterogeneous environment** of different machines, operating systems, and network architectures; the NFS specification is platform independent
- This independence is achieved through the use of RPC primitives built on top of an **External Data Representation (XDR)** protocol used between two implementation-independent interfaces
- The NFS specification includes a mount protocol and a separate network protocol for the remote-file-access services

NFS : The Network File System



NFS : Protocol Summary

- Provides a set of **remote procedure calls** for remote file operations:
  - Searching for a file within a directory
  - Reading a set of directory entries
  - Manipulating links and directories
  - Accessing file attributes
  - Reading and writing files
- Servers up to Version 3 are **stateless**
  - Each request must provide a full set of arguments
- Version 4 servers are **statefull**
- Modified data must be committed to the server's disk before results are returned to the client (lose advantages of caching)
- Does **not provide concurrency-control** mechanisms

NFS : Remote Operations

- **File-Block Cache**
  - On opening a remote file, kernel checks with the remote server whether to fetch or revalidate cached attributes
  - Cached file blocks used only if its cached attributes are up to date
- **File-Attribute Cache**
  - Attribute cache updated when new attributes arrive from the server
  - Clients do not free delayed-write blocks until the server confirms that the data has been written

File Systems : Summary

- Virtual File System
- Mount Points
- Device Directory Goals
- System Calls & File Descriptors
- Open File Tables
- Space Management
- File Allocation Table (FAT)
- i-nodes
- UNIX File System
- Linux File System (self study)
- Network File System

Assignment 5

Do NOT use `system()` system call

Do NOT use `exec()` family of system calls

Clearly credit any "borrowed" code

- **Basic** (30%)
  - Read & Write ASCII text files
- **Advanced** (30%)
  - Append to ASCII text files
  - Copy ASCII text files
- **Stretch** (30%)
  - Print a tree of the directory structure

```
vkhqmgwsgd
  ├── agmugje
  │   ├── fwbv
  │   │   ├── trx
  │   └── surxeb
  │       ├── dyjxfseur
  │       └── wy
  └── tcx
      └── jbjfwbv
          └── rnixyjz
```

Especially for Stretch requirement:
- Take your time
- Start simple
- Build up solution one-step-at-a-time

Next Lecture

- Swap Space
- Mass Storage
- Disk Scheduling