CS 347M (Operating Systems Minor)

Spring 2022

# Lecture 20: Filesystem Implementation

Mythili Vutukuru

CSE, IIT Bombay

# Recap: filesystems

- On disk and in-memory data structures
- Open, read, write system calls
- Link, unlink system calls
- Crash consistency and logging

# Virtual File System (VFS)

- Different filesystems can have different implementations of system calls
  - A filesystem using logging/journaling may write to log first
  - A different directory implementation (fixed size records vs linked list) will lead to a different lookup function
- How to write filesystem code in a modular manner?
  - Should be easy to change system call implementations and switch filesystems
- Solution: Virtual File System (VFS)
  - Defines a set of objects (files, directories, inodes) and operations to be performed on these objects (open a file, lookup filename in directory, ..) for various system calls
  - A specific filesystem implements these functions on VFS objects, provides pointers to the functions to be invoked by OS
- OS filesystem code is built in layers for modularity: VFS, filesystem implementation, disk buffer cache, device driver

# Memory mapping a file

- Alternate way of accessing a file, instead of using file descriptors and read/write syscalls
- mmap() allocates a page in the virtual address space of a process
  - "Anonymous" page: used to store program data
  - File-backed page: contains data of file (filename provided as arg to mmap)
- When file is mmaped, file data copied into one or more pages in memory, can be accessed like any other memory location in program

# Memory mapping a file

```
fd = open("/home/foo/a.txt")
char *buf = mmap(fd, size, ..)
buf[0] = …
buf[1] = …
munmap(..)
```

- Alternate ways to read/write a file is via memory mapping
  - mmap system call takes the file descriptor, size of data to memory map, and other arguments, returns the starting virtual address of mmap region
  - File data is read into one or more physical memory frames, which are mapped at free addresses in the process virtual address space (new page table entries)
  - Can access memory mapped file data like any other memory region
  - With demand paging, physical frames can be assigned on-demand only when mmap region accessed
- File can be memory mapped in private or shared mode
  - Shared mode: changes to file are written to disk immediately, seen by others
  - Private mode: changes to file are written to disk when memory unmapped

# mmap vs. read/write syscalls

- mmap can be used for file-backed as well as anonymous pages
  - Physical frame mapped into address space can be empty frame or with file data
- Memory mapping a file is an easy way to read file data
  - Executable code, shared library code are memory mapped into virtual address space
- Memory mapping a file avoids extra data copies
  - Read/write system calls read data first into memory (disk buffer cache), then copy from disk buffer cache into user provided buffer
  - Memory mapping a file copies file data into free physical frames, which are directly accessed by user using virtual addresses
- Memory mapping allows reading disk data in large page-sized chunks
  - Useful when reading/writing large amounts of data from file
  - Not very efficient when reading files in small chunks