CSE 451: Operating Systems

Section 7

File Systems; Project 3

Project 2

- Done!

Project 3

- Due: Thursday, December 9, 11:59pm

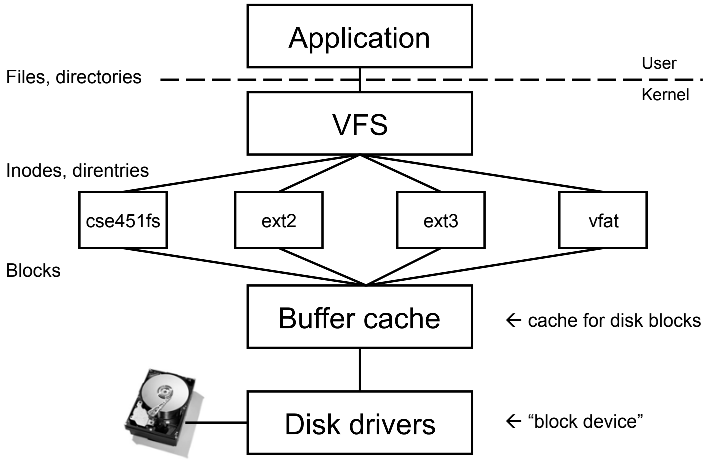
Project 3

- Work with a real file system
  - **cse451fs**: simple file system for Linux
- Goals:
  - Understand how it works
  - Modify implementation to:
    - Increase maximum size of files (currently 10KB)
    - Allow for longer file names (currently 30 chars)
    - Allow for more files (currently ~8000)

Project 3 procedure

- Build a kernel module for cse451fs
- On a virtual machine running Linux kernel:
  - Load the cse451fs module
  - Format the file system using (modified) *mkfs* tool
  - Mount the file system
  - Test using tools like ls, cat, etc.
- Try this procedure with given code first
  - Then carefully read writeup again, and go!

Linux file system layers

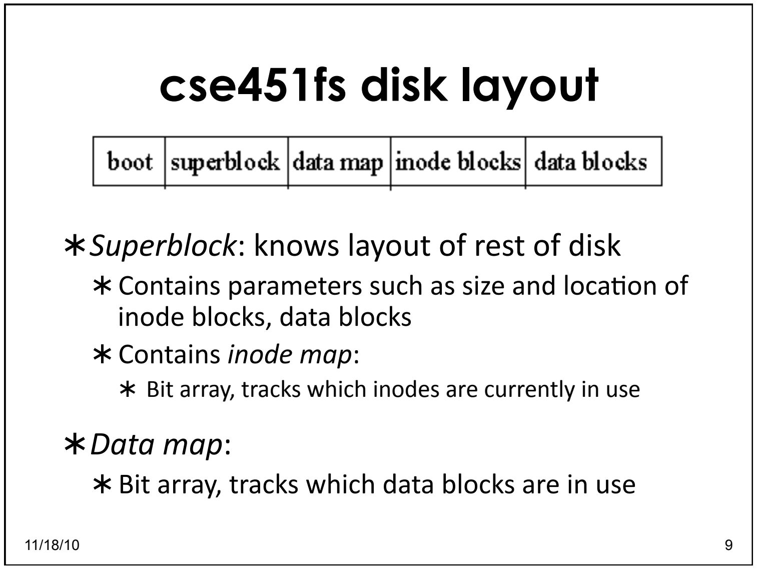


Inodes

- *Inode*: a structure maintaining all metadata about a file (or directory)
  - Inode number (unique ID of inode)
  - Permissions, timestamps
  - Pointers to *data blocks*
- Inode does *not* contain: name of file
  - One or more file names can point (link) to the same inode

Directories

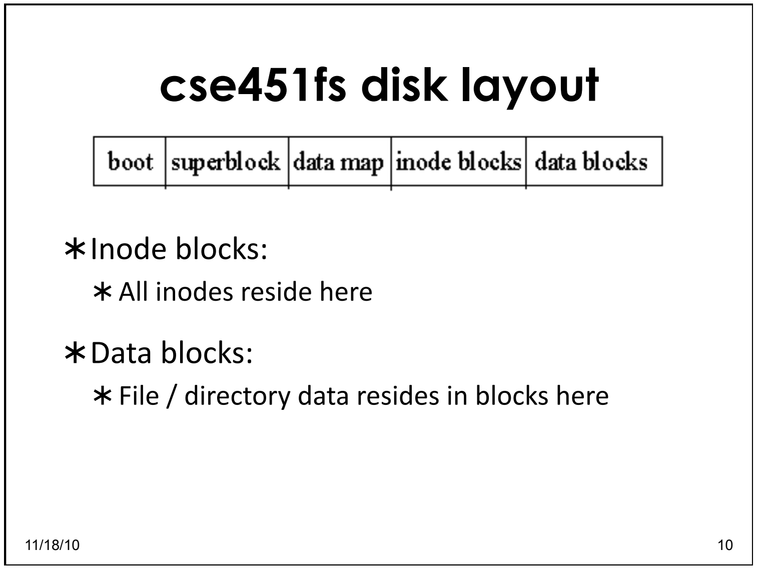
- *Directory entry* ("dirent"): a name + inode number pair
- *Directory*: a file that contains a list of directory entries



cse451fs disk layout

boot	superblock	data map	inode blocks	data blocks

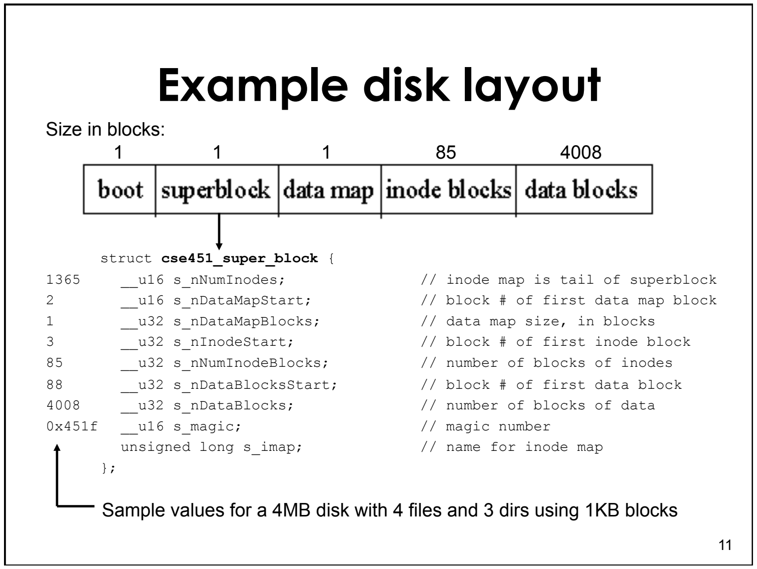
- *Superblock*: knows layout of rest of disk
  - Contains parameters such as size and location of inode blocks, data blocks
  - Contains *inode map*:
    - Bit array, tracks which inodes are currently in use
- *Data map*:
  - Bit array, tracks which data blocks are in use



cse451fs disk layout

boot	superblock	data map	inode blocks	data blocks

- Inode blocks:
  - All inodes reside here
- Data blocks:
  - File / directory data resides in blocks here



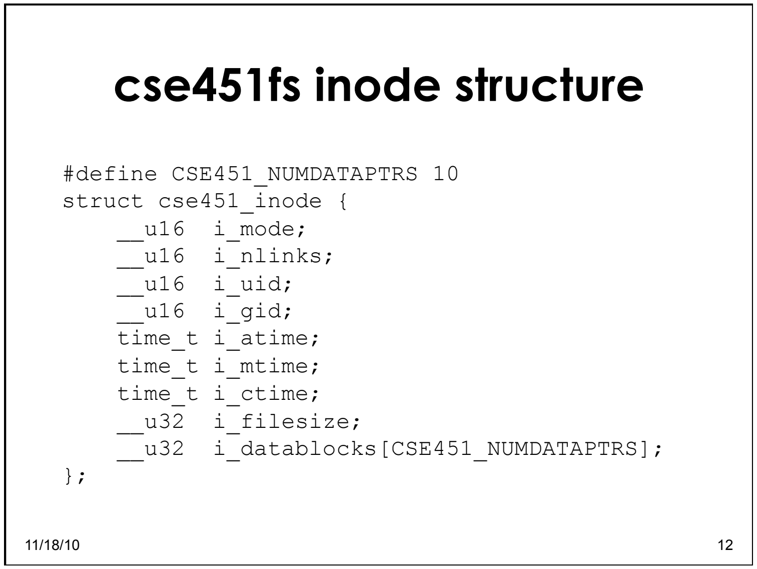
Example disk layout

Size in blocks:

1	1	1	85	4008
boot	superblock	data map	inode blocks	data blocks

```
      struct cse451_super_block {
1365    __u16 s_nNumInodes;             // inode map is tail of superblock
2       __u16 s_nDataMapStart;          // block # of first data map block
1       __u32 s_nDataMapBlocks;         // data map size, in blocks
3       __u32 s_nInodeStart;            // block # of first inode block
85      __u32 s_nNumInodeBlocks;        // number of blocks of inodes
88      __u32 s_nDataBlocksStart;       // block # of first data block
4008    __u32 s_nDataBlocks;            // number of blocks of data
0x451f  __u16 s_magic;                  // magic number
        unsigned long s_imap;           // name for inode map
      };
```

Sample values for a 4MB disk with 4 files and 3 dirs using 1KB blocks



cse451fs inode structure

```
#define CSE451_NUMDATAPTRS 10
struct cse451_inode {
    __u16  i_mode;
    __u16  i_nlinks;
    __u16  i_uid;
    __u16  i_gid;
    time_t i_atime;
    time_t i_mtime;
    time_t i_ctime;
    __u32  i_filesize;
    __u32  i_datablocks[CSE451_NUMDATAPTRS];
};
```

cse451fs inode structure

- Remember, inodes themselves are stored in blocks
  - What's the size of the inode struct?
  - So how many inside a 1K block?
- Max number of inodes (max number of files) usually decided when file system is formatted
  - mkfs heuristic: create an inode for every three data blocks

Data blocks

- Blocks for regular files contain file data
- Blocks for directories contain directory entries:

```
#define MAXDIRNAMELENGTH 30
struct cse451_dir_entry {
   __u16 inode;
   char name
      [MAXDIRNAMELENGTH];
};
```

Data block for /

Dir. entry	Field	Value
0	Inode	1
	Name	"."
1	Inode	1
	Name	".."
2	Inode	2
	Name	"etc"
3	Inode	3
	Name	"bin"
4	Inode	0
	Name	0

Example data block usage

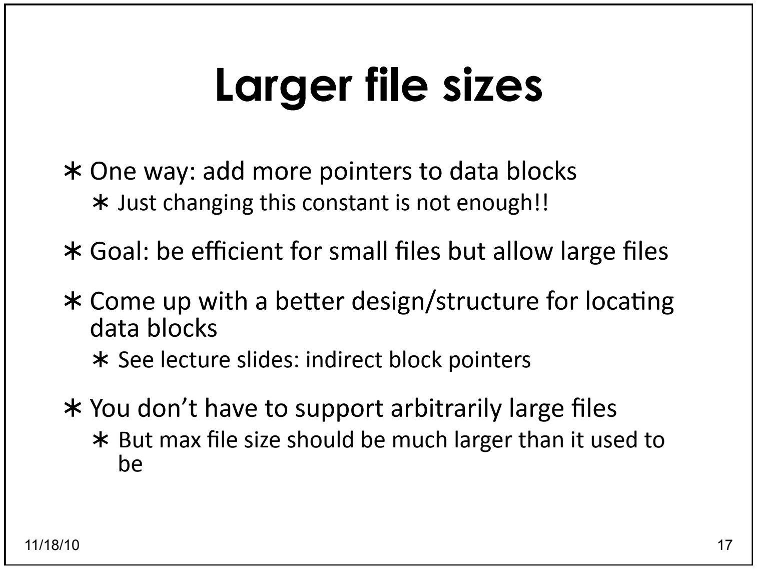
- For a 4MB file system with 1KB blocks, with hierarchy:

```
/
   etc
      passwd
      fstab
   bin
      sh
      date
```

File/Directory	Size	Data Blocks
/	4 entries + 1 null entry	1
/etc	4 entries + 1 null entry	1
/bin	4 entries + 1 null entry	1
/etc/passwd	1024 bytes	1
/etc/fstab	100 bytes	1
/bin/sh	10,000 bytes	10
/bin/date	5,000 bytes	5
	Total:	**20**

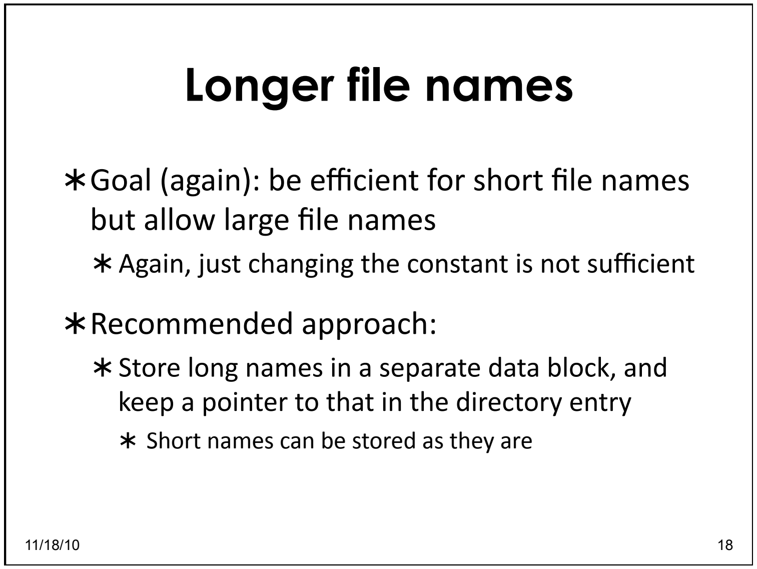
Project 3 requirements

- Increase the maximum file size
- Increase the maximum file name length
- Increase the maximum number of files



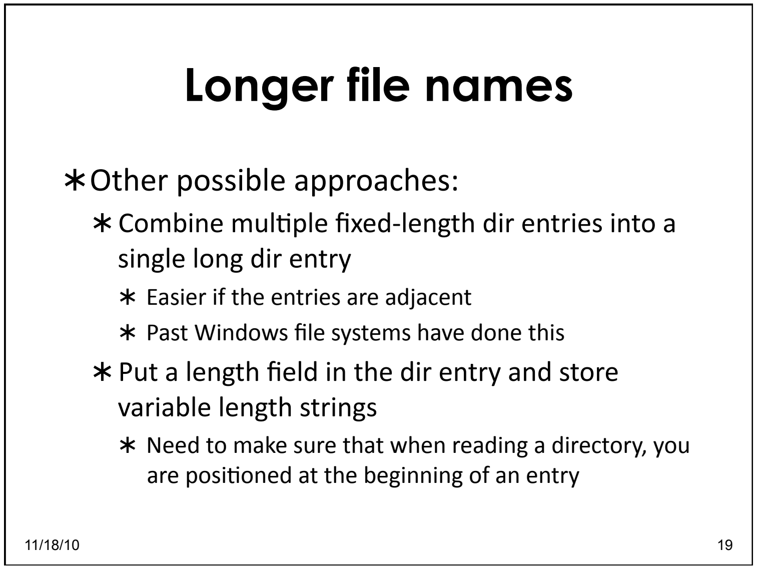
Larger file sizes

- ∗ One way: add more pointers to data blocks
  - ∗ Just changing this constant is not enough!!
- ∗ Goal: be efficient for small files but allow large files
- ∗ Come up with a better design/structure for locating data blocks
  - ∗ See lecture slides: indirect block pointers
- ∗ You don't have to support arbitrarily large files
  - ∗ But max file size should be much larger than it used to be



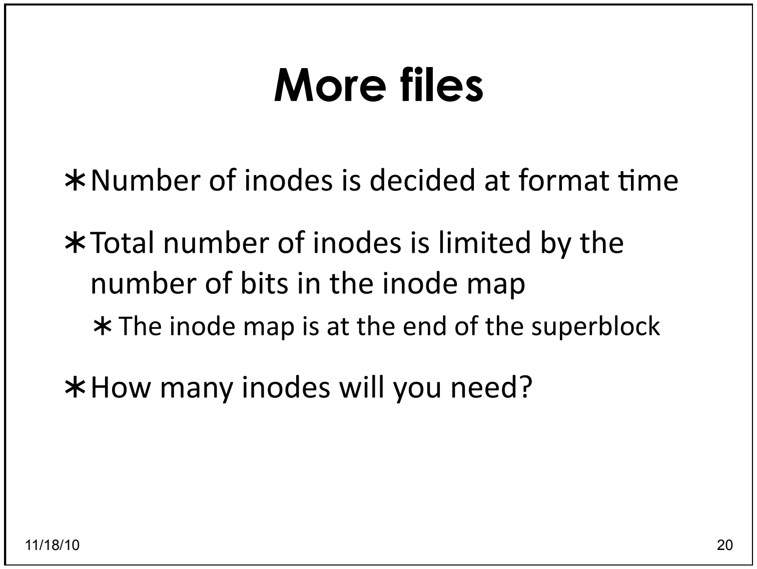
Longer file names

- ∗Goal (again): be efficient for short file names but allow large file names
  - ∗Again, just changing the constant is not sufficient
- ∗Recommended approach:
  - ∗Store long names in a separate data block, and keep a pointer to that in the directory entry
    - ∗ Short names can be stored as they are



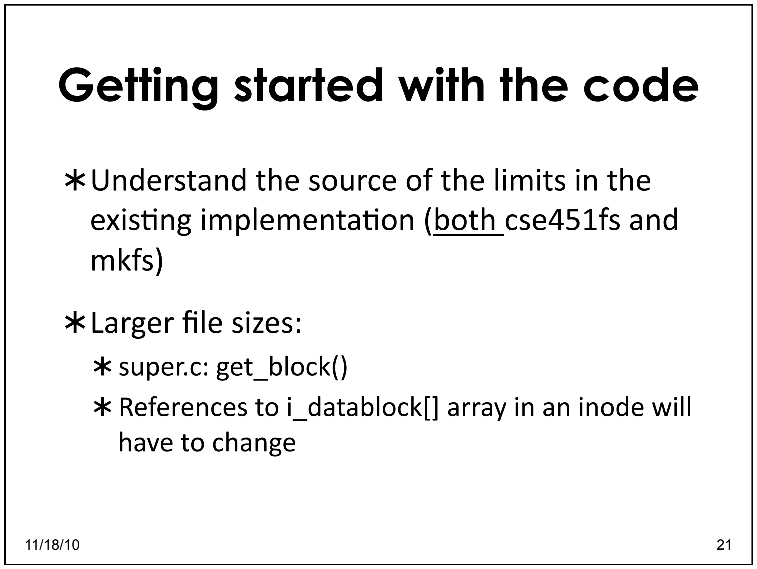
Longer file names

- ∗Other possible approaches:
  - ∗Combine multiple fixed-length dir entries into a single long dir entry
    - ∗ Easier if the entries are adjacent
    - ∗ Past Windows file systems have done this
  - ∗Put a length field in the dir entry and store variable length strings
    - ∗ Need to make sure that when reading a directory, you are positioned at the beginning of an entry



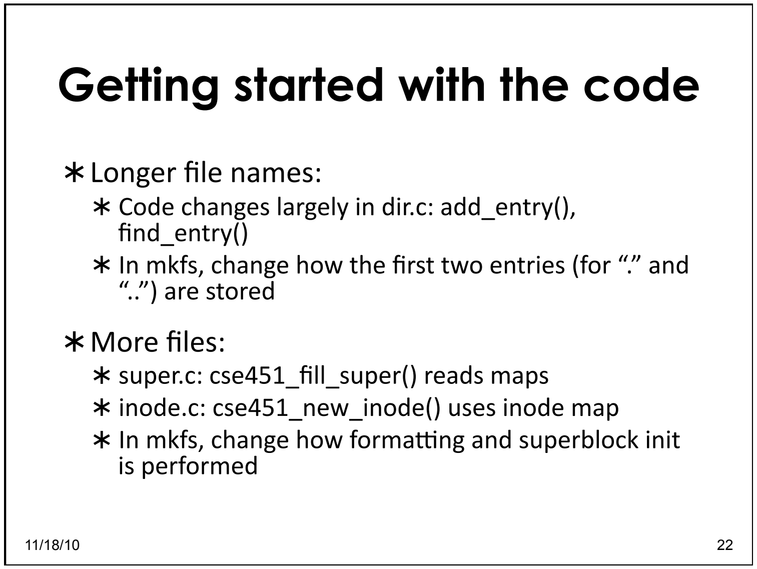
More files

- ∗Number of inodes is decided at format time
- ∗Total number of inodes is limited by the number of bits in the inode map
  - ∗The inode map is at the end of the superblock
- ∗How many inodes will you need?



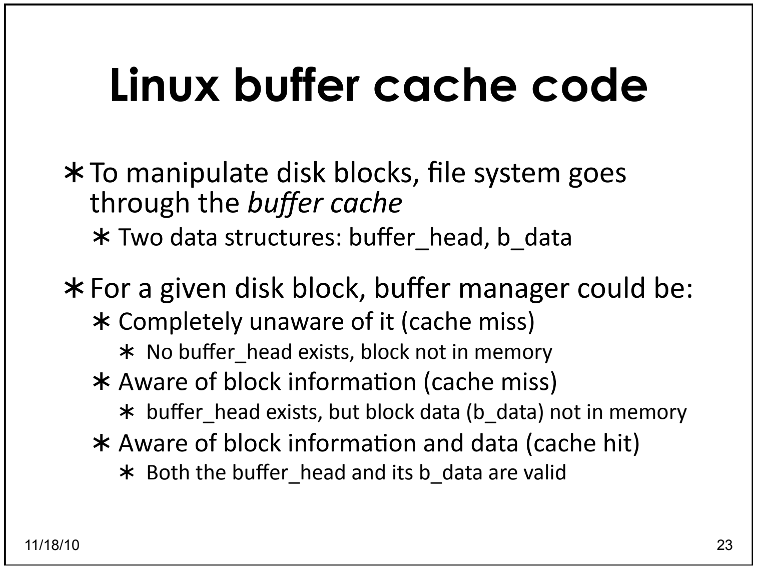
Getting started with the code

- Understand the source of the limits in the existing implementation (both cse451fs and mkfs)
- Larger file sizes:
  - super.c: get_block()
  - References to i_datablock[] array in an inode will have to change



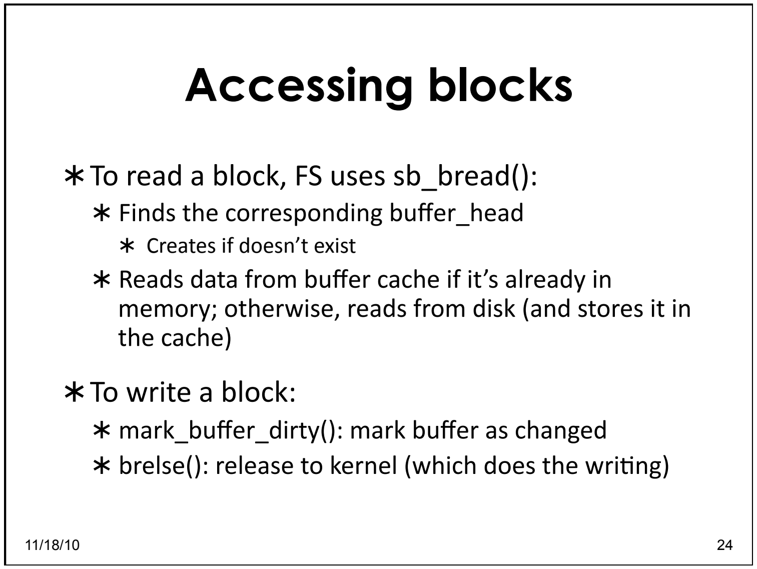
Getting started with the code

- Longer file names:
  - Code changes largely in dir.c: add_entry(), find_entry()
  - In mkfs, change how the first two entries (for "." and "..") are stored
- More files:
  - super.c: cse451_fill_super() reads maps
  - inode.c: cse451_new_inode() uses inode map
  - In mkfs, change how formatting and superblock init is performed



Linux buffer cache code

- To manipulate disk blocks, file system goes through the *buffer cache*
  - Two data structures: buffer_head, b_data
- For a given disk block, buffer manager could be:
  - Completely unaware of it (cache miss)
    - No buffer_head exists, block not in memory
  - Aware of block information (cache miss)
    - buffer_head exists, but block data (b_data) not in memory
  - Aware of block information and data (cache hit)
    - Both the buffer_head and its b_data are valid



Accessing blocks

- To read a block, FS uses sb_bread():
  - Finds the corresponding buffer_head
    - Creates if doesn't exist
  - Reads data from buffer cache if it's already in memory; otherwise, reads from disk (and stores it in the cache)
- To write a block:
  - mark_buffer_dirty(): mark buffer as changed
  - brelse(): release to kernel (which does the writing)

Tool limitation warning

- Some items in Linux kernel are limited to 256 chars
  - e.g. VFS, ls
  - Be careful when testing long filenames!
- dd is useful for creating large test files

```
dd if=/dev/zero of=200k bs=1024 count=200
```

- df is useful to check that you're freeing everything correctly

gcc warning

- gcc might insert extra space into structs
  - How big do you think this is?

```
struct foo { char a; int b; }
```

  - Why is this a problem?
    - What if foo represents something you want on disk (i.e. directory entries)?
    - Discrepancy between the disk layout and memory layout

gcc warning

- Fix:

```
struct bar {
  char a;
  int b;
} __attribute__((packed));
```

  - sizeof(bar) is now 5
- Real fix: don't make any assumptions in your code about size of structs!