Reminders

- Project 4 due Dec 10
  - turnin only (include writeup)
- If you used your late pass on HW5
  - Turn in to me, by Tue, Dec 7 at the latest.
- Today:
  - Project 4 and file systems
  - HW4 + Project 2 back (finally)
    - HW4 average: 64.8/80
    - Project 2 average: 74/85

1

Project 4 first steps

- Go through the mechanics
  - Compile the kernel & file system
  - Boot VMWare with new kernel
  - Install ramdisk, install cse451fs, mount, test
- Read the code
  - Start with cse451fs.h
  - Follow with mkfs & dir.c
- Start with increasing filename length

2

cse451fs disk structure

boot	superblock	data map	inode blocks	data blocks

- **Superblock**: tells where all other things are
  - Contains **inode map**:
    - Bit array, tracks which inodes are currently in use
    - E.g. for 3 dirs + 4 files, need 7 inodes
- **Data map**:
  - Bit array, tracks which data blocks are in use

3

cse451fs structure

Sample Size:	1	1	1	85	4008
	boot	superblock	data map	inode blocks	data blocks

```
struct cse451_super_block {
   __u16 s_nNumInodes;        // inode map is tail of superblock
   __u16 s_nDataMapStart;     // block # of first data map block
   __u32 s_nDataMapBlocks;    // data map size, in blocks
   __u32 s_nInodeStart;       // block # of first inode block
   __u32 s_nNumInodeBlocks;   // number of blocks of inodes
   __u32 s_nDataBlocksStart;  // block # of first data block
   __u32 s_nDataBlocks;       // number of blocks of data
   __u32 s_nBusyInodes;       // number of inodes in use
   __u16 s_magic;             // magic number
   char s_imap[0];            // name for inode map
};
```

4

Superblock values

- For a 4mb disk:

Field	Value
s_nNumInodes	1365
s_nDataMapStart	2
s_nDataMapBlocks	1
s_nInodeStart	3
s_nNumInodeBlocks	85
s_nDataBlocksStart	88
s_nDataBlocks	4008
s_nBusyInodes	7
s_magic	CSE451_SUPER_MAGIC (0x451F)

5

Inode structure

```
#define CSE451_NUMDATAPTRS 13

struct cse451_inode {
   __u16 i_mode;          ß determines if file or dir
   __u16 i_nlinks;          (+ protection)
   __u16 i_uid;
   __u16 i_gid;
   __u32 i_filesize;
   __u32 i_datablocks[CSE451_NUMDATAPTRS];
};
```

- Inode size?
- Multiple inodes per block!
  - How many for 1K block?
- mkfs decides how many inodes to create
  - mkfs.cse451fs.c : create an inode for every three data blocks

6

Data blocks

- Blocks for files contain file data
- Blocks for directories contain:

```
#define CSE451_MAXDIRNAMELENGTH   30
struct cse451_dir_entry {
   __u16 inode;
   char name[CSE451_MAXDIRNAMELENGTH];
};
```

- Data block for / directory containing:
  . .. etc bin
  - What's this dir's inode number?
  - What is the "file size" in this dir's inode?

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

7

Sample data block usage

For a 4MB file system with 1KB blocks

- /
  - 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**

8

Project 4 requirements

- Increasing maximum size of files
  - Be efficient for small files but allow large files
  - Changing constant (=13) is **not enough**.
  - Come up with a better design/structure for locating data blocks.
  - Don't have to support arbitrarily large files
    - Fine to have constant new_max (but new_max >> old_max)
- Allow for longer file names
  - Be efficient for short files names but allow large file names
  - Again, don't just change the constant

9

Approaches for longer file names

- Combine multiple fixed-length dir entries into a single long dir entry (win95)
  - It is easier if the entries are adjacent.
- 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.
- Put a length field in the dir entry and store variable length strings
  - need to make sure that when reading a directory, that you are positioned at the beginning of an entry.

10

Getting started with the code

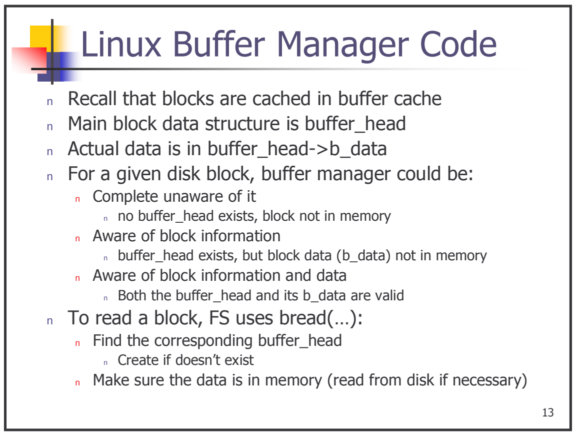
- Understand the source of the limits in the existing implementation
  - Look at the code that manipulates dir entries
    - mkfs code
    - dir.c in the file system source code
- Longer file names:
  - The code for will largely be in dir.c: add_entry() and find_entry()
  - In mkfs, change how the first two entries (for "." and "..") are stored
- Bigger files:
  - super.c:get_block()
  - References to i_datablock[] array in an inode will have to change

11

VFS vs cse451fs

- Don't conflate VFS structures and cse451fs structures!
  - inodes, superblocks
- E.g., there are "two" inodes:
  - VFS `struct inode`
    - Generic inode used in Linux source (works for any FS)
    - Lives in memory
  - cse451 `struct cse451_inode`
    - Actual inode representation on disk
  - **`inode.c`**:`cse451_read_inode` converts from `cse451_inode` to `struct inode`!
    - Copies over mode, size, etc
    - Copies over i_datablocks[] to `struct inode`'s `generic_ip` field (which will now be used as type `cse451_inode_info`)
  - **`inode.c`**:`cse451_write_inode` converts the other way

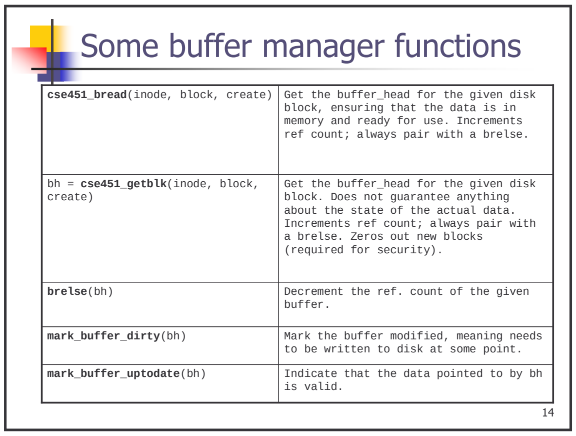
12



Linux Buffer Manager Code

- Recall that blocks are cached in buffer cache
- Main block data structure is buffer_head
- Actual data is in buffer_head->b_data
- For a given disk block, buffer manager could be:
  - Complete unaware of it
    - no buffer_head exists, block not in memory
  - Aware of block information
    - buffer_head exists, but block data (b_data) not in memory
  - Aware of block information and data
    - Both the buffer_head and its b_data are valid
- To read a block, FS uses bread(…):
  - Find the corresponding buffer_head
    - Create if doesn't exist
  - Make sure the data is in memory (read from disk if necessary)

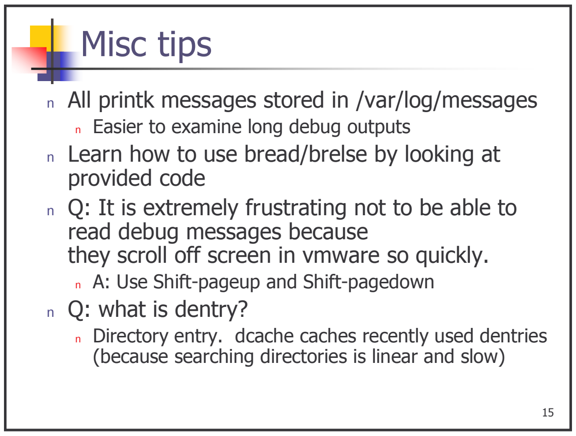
13



Some buffer manager functions

cse451_bread(inode, block, create)	Get the buffer_head for the given disk block, ensuring that the data is in memory and ready for use. Increments ref count; always pair with a brelse.
bh = **cse451_getblk**(inode, block, create)	Get the buffer_head for the given disk block. Does not guarantee anything about the state of the actual data. Increments ref count; always pair with a brelse. Zeros out new blocks (required for security).
brelse(bh)	Decrement the ref. count of the given buffer.
mark_buffer_dirty(bh)	Mark the buffer modified, meaning needs to be written to disk at some point.
mark_buffer_uptodate(bh)	Indicate that the data pointed to by bh is valid.

14



Misc tips

- All printk messages stored in /var/log/messages
  - Easier to examine long debug outputs
- Learn how to use bread/brelse by looking at provided code
- Q: It is extremely frustrating not to be able to read debug messages because they scroll off screen in vmware so quickly.
  - A: Use Shift-pageup and Shift-pagedown
- Q: what is dentry?
  - Directory entry. dcache caches recently used dentries (because searching directories is linear and slow)

15