CS111, Lecture 3

Filesystem Design

😷 masks required

Reminder: PollEverywhere

- Participation recorded starting today - visit pollev.stanford.edu to log in (or use the PollEverywhere app) and sign in with your **@stanford.edu email – NOT your personal email!**
- Responses not anonymized, but we don't look at specific responses, just aggregated results and participation totals
- You can use any device with a web browser, or download the PollEverywhere app, or respond via text – **however, to respond via text you must first log in via a web browser and add your phone number to your profile.**
- Whenever we reach a poll question in the slides, it will automatically activate the poll and allow you to respond.

Announcements

- Remember to input your section preferences by 5PM Sat! Link is on the course website (under "Sections").
- CS Dept. social gathering Thurs. 10/6 4-6PM on the AT&T patio behind Gates building! RSVP if you want to attend so they know how much food to order.
- Reminder: masks required in classrooms (including lecture + section)

Topic 1: Filesystems - How can we design filesystems to manage files on disk, and what are the tradeoffs inherent in designing them? How can we interact with the filesystem in our programs?

CS111 Topic 1: Filesystems

Filesystems introduction and design → Case study: Unix V6 Filesystem → Filesystem System calls and file descriptors → Crash recovery

Filesystems introduction and design	Case study: Unix V6 Filesystem	Filesystem System calls and file descriptors	Crash recovery
Lecture 1/ lecture 2	**Lecture 2 / This lecture / Lecture 4**	**Lecture 5**	**Lectures 6-7**

assign1: implement portions of the Unix v6 filesystem!

Learning Goals

- Explore the design of the Unix V6 filesystem
- Understand how we can use inodes to store and access file data
- Learn about how inodes can accommodate small and large files

Plan For Today

- **Recap**: filesystems so far
- The Unix V6 Filesystem and Inodes
- **Practice**: reading file data
- Large files and Singly-Indirect Addressing
- **Practice**: singly-indirect addressing
- Large files and Doubly-Indirect Addressing

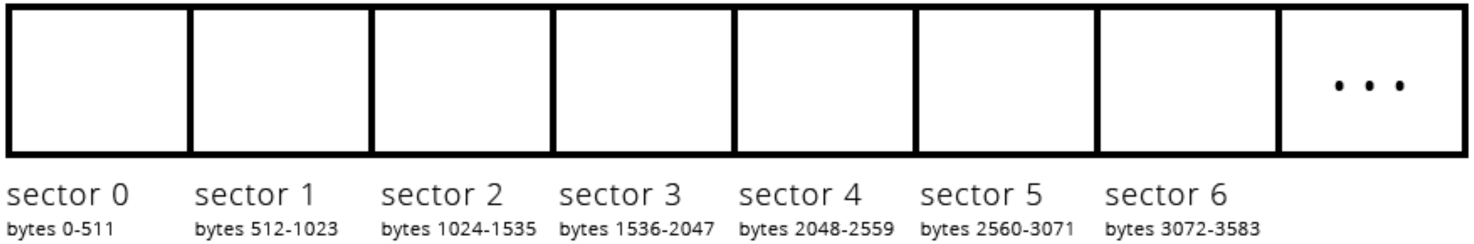
Plan For Today

- **<u>Recap</u>: filesystems so far**
- The Unix V6 Filesystem and Inodes
- **<u>Practice</u>**: reading file data
- Large files and Singly-Indirect Addressing
- **<u>Practice</u>**: singly-indirect addressing
- Large files and Doubly-Indirect Addressing

Recap: Filesystems

We are imagining that we are filesystem implementers. A **filesystem** is the portion of the OS that manages the disk.

- A hard drive (or, more commonly these days, flash storage) is persistent storage – it can store data between power-offs.
- We have a hard disk that supports only two operations (reading a sector and writing a sector), and need to layer complex filesystem operations (like reading/writing/locating entire files) on top.
- A “block” is a filesystem storage unit (1 or more sectors)
- Both file payload data and metadata must be stored on disk



Some Possible Filesystem Designs

- *Contiguous allocation* allocates a file in one contiguous space
- *Linked files* allocates files by splitting them into blocks and having each block store the location of the next block.
- *Windows FAT* is like linked files but stores the links in a “file allocation table” in memory for faster access.
- *Multi-level indexes* store all block numbers for a file so we can quickly jump to any point in the file (but how?). Example: Unix v6 Filesystem
- Many other designs possible – many use a tree-like structure

Filesystem Designs

- A file may not take up all the space in the blocks it's using. E.g. block = 512 bytes, but file is only 300 bytes. *Internal fragmentation!* (you could share blocks between multiple files, but this gets complex)
- Wait, how do we look up / find files? (we'll talk more about this!)

Plan For Today

- **Recap**: filesystems so far
- **The Unix V6 Filesystem and Inodes**
- **Practice**: reading file data
- Large files and Singly-Indirect Addressing
- **Practice**: singly-indirect addressing
- Large files and Doubly-Indirect Addressing

Unix V6 Inodes

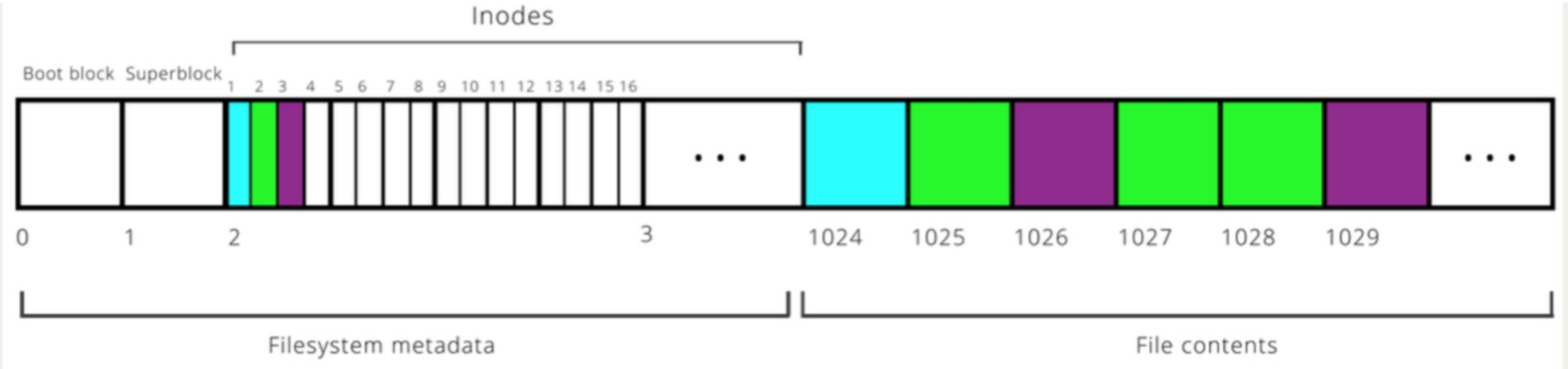
The Unix v6 filesystem stores inodes on disk together in the **inode table** for quick access. An **inode** ("index node") is a grouping of data about a single file. It’s stored on disk, but we can read it into memory when the file is open.

- inodes are stored in a reserved region starting at block 2 (block 0 is "boot block" containing hard drive info, block 1 is "superblock" containing filesystem info). Typically, at most 10% of the drive stores metadata.
- Inodes are 32 bytes big, and 1 block = 1 sector = 512 bytes, so 16 inodes/block.
- Filesystem goes from **filename** to **inode number** ("inumber") to **file data**.

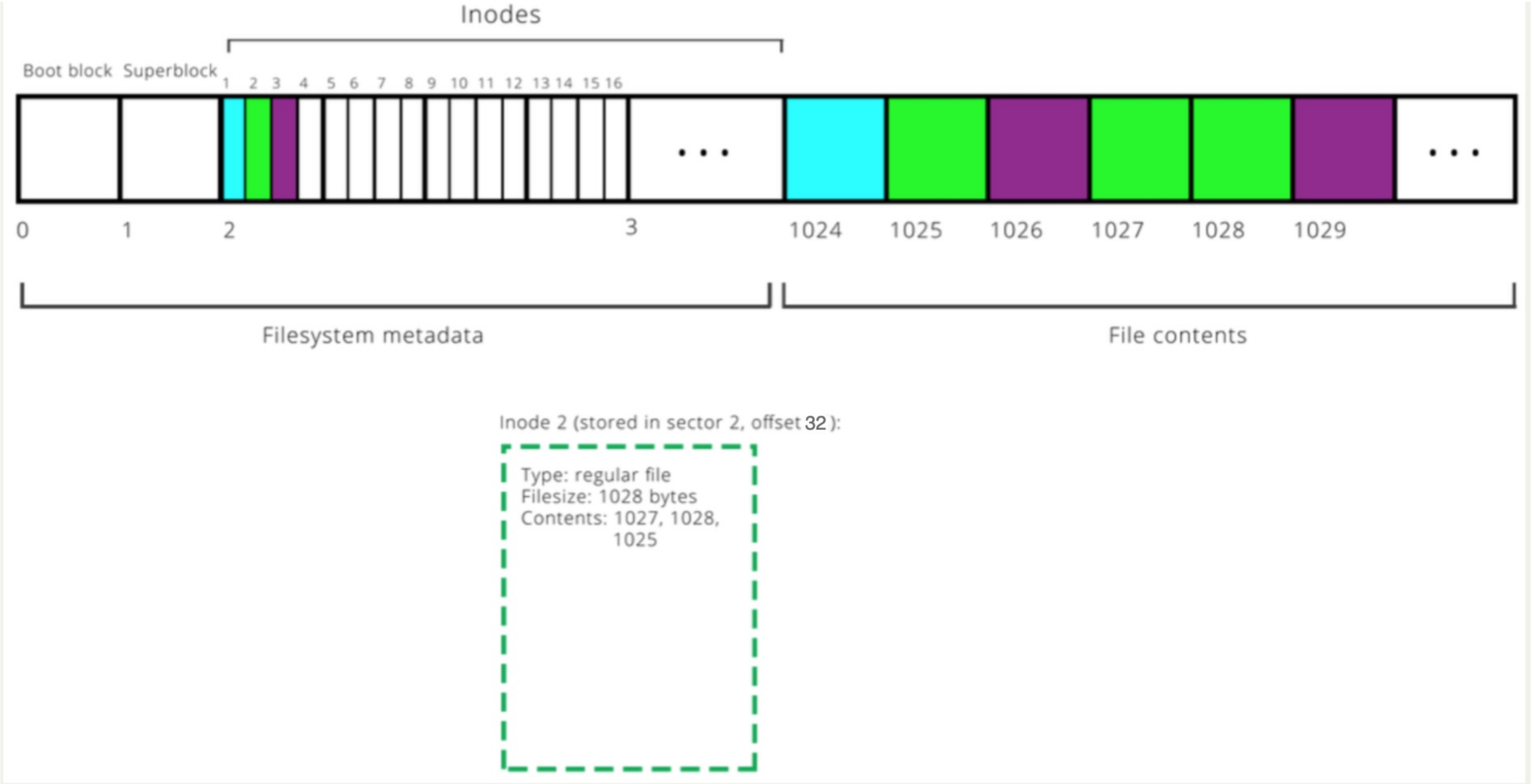
Unix V6 Inodes

The Unix v6 filesystem stores inodes on disk together in the **inode table** for quick access. An **inode** ("index node") is a grouping of data about a single file. It's stored on disk, but we can read it into memory when the file is open.

- inodes are stored in a reserved region starting at block 2 (block 0 is "boot block" containing hard drive info, block 1 is "superblock" containing filesystem info). Typically, at most 10% of the drive stores metadata.
- Each Unix v6 inode has space for 8 block numbers



Unix V6 Inodes



Unix V6 Inodes

Let's imagine that the hard disk creators provide software to let us interface with the disk.

```
void readSector(size_t sectorNumber, void *data);
void writeSector(size_t sectorNumber, const void *data);
```

*(Refresher: size_t is an unsigned number, void * is a generic pointer)*

Let’s look at how we might access inodes in filesystem code.

Unix V6 Inodes

```
typedef struct inode {
  uint16_t  i_addr[8];  // block numbers
  ...
} inode;

// Loop over each inode in sector 2
inode inodes[512 / sizeof(inode)];

readSector(2, inodes);
for (size_t i = 0; i < sizeof(inodes) /
    sizeof(inodes[0]); i++) {
    printf("%\n", inodes[i].i_addr[0]); // first block num
}
```

Plan For Today

- Recap: filesystems so far
- The Unix V6 Filesystem and Inodes
- **Practice: reading file data**
- Large files and Singly-Indirect Addressing
- Practice: singly-indirect addressing
- Large files and Doubly-Indirect Addressing

Practice #1: Inodes

Let's say we have an inode with the following information (remember 1 block = 1 sector = 512 bytes):

file size: 600 bytes

block numbers: 56, 122

- How many bytes of block 56 store file payload data?
- How many bytes of block 122 store file payload data?

Bytes 0-511 (512 bytes) reside within block 56, bytes 512-599 (88 bytes) within block 122.

Practice #2: Inodes

Let's say we have an inode with the following information (remember 1 block = 1 sector = 512 bytes):

- **file size: 2000 bytes**
- **block numbers: 56, 122, 45, 22**

- Which block number stores the index-1500th byte of the file?

Bytes 0-511 reside within block 56, bytes 512-1023 within block 122, bytes 1024-1535 within block 45, and bytes 1536-1999 at the front of block 22.

Plan For Today

- **Recap**: filesystems so far
- The Unix V6 Filesystem and Inodes
- **Practice**: reading file data
- **Large files and Singly-Indirect Addressing**
- **Practice**: singly-indirect addressing
- Large files and Doubly-Indirect Addressing

File Size

Problem: with 8 block numbers per inode, the largest a file can be is 512 * 8 = 4096 bytes (~4KB). That definitely isn't realistic!

Let's say a file's payload is stored across 10 blocks:

45, 42, 15, 67, 125, 665, 467, 231, 162, 136

<u>Assuming that the size of an inode is fixed, where can we put these block numbers?</u>

Solution: let's store them *in a block*, and then store *that* block's number in the inode!

File Size

Let's say a file's payload is stored across 10 blocks:

451, 42, 15, 67, 125, 665, 467, 231, 162, 136

Solution: let's store them *in a block*, and then store *that* block's number in the inode! This approach is called *indirect addressing*.

inode

filesize: 5000
blocknums: 450
...

Block 450

451, 42, 15,
67, 125, 665,
467, 231,
162,136

Block 451

The quick
brown fox
jumped over
the...

Indirect Addressing

Design questions:

- Should we make *all* the block numbers in an inode use indirect addressing?
- Should we use this approach for all files, or just large ones?

Indirect addressing is useful but means that it takes more steps to get to the data, and uses more blocks.

inode

filesize: 5000
blocknums: 450
…

Block 450

451, 42, 15, 67, 125, 665, 467, 231, 162,136

Block 451

The quick brown fox jumped over the...

Indirect Addressing

The Unix V6 filesystem uses *singly-indirect addressing* (blocks that store payload block numbers) just for large files.

- check flag or size in inode to know whether it is a small file (direct addressing) or large one (indirect addressing)
  - If small, each block number in the inode stores payload data
  - If large, **first 7 block numbers** in the inode stores block numbers for payload data
  - 8th block number? we'll get to that :)
- Let's assume for now that an inode for a large file uses all 8 block numbers for singly-indirect addressing. What is the largest file size this supports? Each block number is 2 bytes big.

Indirect Addressing

Let's assume for now that an inode for a large file uses all 8 block numbers for singly-indirect addressing. What is the largest file size this supports? Each block number is 2 bytes big.

8 block numbers in an inode x

256 block numbers per singly-indirect block x

512 bytes per block

= ~1MB

Practice: Indirect Addressing

Let's say we have an inode with the following information (remember 1 block = 1 sector = 512 bytes, and block numbers are 2 bytes big):

file size: 200,000 bytes

block numbers: 56, 122

Which singly-indirect block stores the block number holding the index-150,000th byte of the file?

*Bytes 0-131,071 reside within blocks whose block numbers are in block 56. Bytes 131,072 (256*512) - 199,999 reside within blocks whose block numbers are in block 122.*

Even Larger Files

Problem: even with singly-indirect addressing, the largest a file can be is 8 * 256 * 512 = 1,048,576 bytes (~1MB). That still isn't realistic!

Solution: let's use *doubly-indirect addressing*; store a block number for a block that contains *singly-indirect block numbers*.

Plan For Today

- **Recap**: filesystems so far
- The Unix V6 Filesystem and Inodes
- **Practice**: reading file data
- Large files and Singly-Indirect Addressing
- **Practice**: singly-indirect addressing
- **Large files and Doubly-Indirect Addressing**

Even Larger Files

Solution: let's use *doubly-indirect addressing*; store a block number for a block that contains *singly-indirect block numbers*.

Allows even larger files, but data takes even more steps to access. How do we employ this idea?

inode

filesize: 5000
blocknums: 450
…

Block 450

451, 42, 15, 67, 125, 665, 467, 231, 162,136

Block 451

55, 34, 12, 44,…

Block 55

The quick brown fox jumped over the...

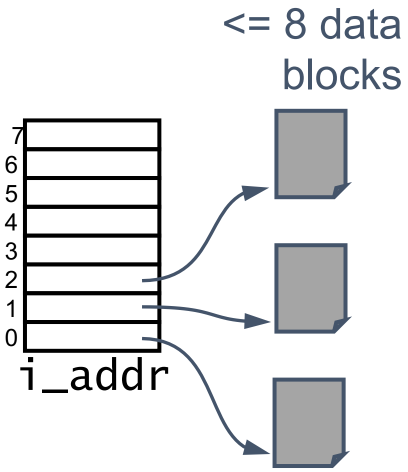
Indirect Addressing

The Unix V6 filesystem uses *indirect addressing* (blocks that store payload block numbers) just for large files.

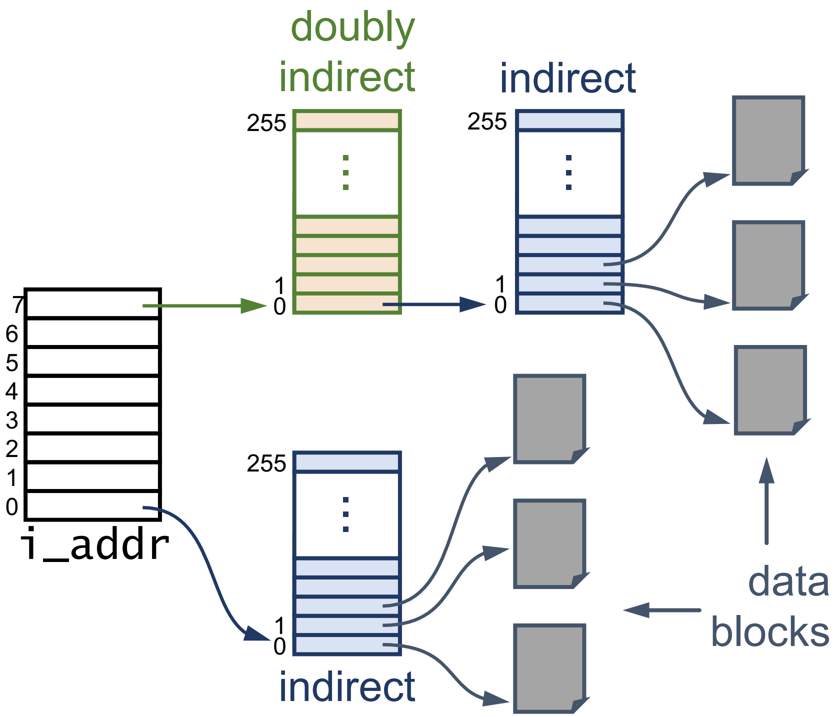
- check flag or size in inode to know whether it is a small file (direct addressing) or large one (indirect addressing)
  - If small, each block number in the inode stores payload data
  - If large, **first 7 block numbers** are singly-indirect
  - NEW: If large (and if needed), **8th block number** is doubly-indirect (it refers to a block that stores singly-indirect block numbers)

In other words; a file can be represented using at most 256 + 7 = 263 singly-indirect blocks. The first seven are stored in the inode. The remaining 256 are stored in a block whose block number is stored in the inode.

“Multi-Level Indexes”



Small



Large

Indirect Addressing

An inode for a large file stores 7 singly-indirect block numbers and 1 doubly-indirect block number. What is the largest file size this supports? Each block number is 2 bytes big.

(7+256) singly-indirect block numbers total x

256 block numbers per singly-indirect block x

512 bytes per block

= ~34MB

Indirect Addressing

An inode for a large file stores 7 singly-indirect block numbers and 1 doubly-indirect block number. What is the largest file size this supports? Each block number is 2 bytes big.

OR:

*(7 * 256 * 512) + (256 * 256 * 512) ~ 34MB*

(singly indirect) + (doubly indirect)

Better! still not sufficient for today's standards, but perhaps in 1975. Moreover, since block numbers are 2 bytes, we can number at most 2^16 - 1 = 65,535 blocks, meaning the entire filesystem can be at most 65,535 * 512 ~ 32MB.

Plan For Today

- **Recap**: filesystems so far
- The Unix V6 Filesystem and Inodes
- **Practice**: reading file data
- Large files and Singly-Indirect Addressing
- **Practice**: singly-indirect addressing
- Large files and Doubly-Indirect Addressing
- **Assignment 1**

Assignment 1

- Assignment 1 released later today, due Mon. 10/10
- Implement core functions to read from a Unix v6 filesystem disk!
  - **inode_iget** -> fetch a specific inode
  - **inode_indexlookup** -> fetch a specific payload block number
  - **file_getblock** -> fetch a specified payload block
  - **directory_findname** -> fetch directory entry with the given name
  - **pathname_lookup** -> fetch inumber for the file with the given path

Assignment 1

- Assignment 1 released later today, due Mon. 10/10
- Implement core functions to read from a Unix v6 filesystem disk!
  - **inode_iget** -> fetch a specific inode
  - **inode_indexlookup** -> fetch a specific payload block number
  - **file_getblock** -> fetch a specified payload block
  - **directory_findname** -> fetch directory entry with the given name
  - **pathname_lookup** -> fetch inumber for the file with the given path

can start now

will discuss next time!

Assignment 1

- Assignment 1 released later today, due Mon. 10/10
- Implement core functions to read from a Unix v6 filesystem disk!
  - **inode_iget** -> fetch a specific inode
  - **inode_indexlookup** -> fetch a specific payload block number
  - **file_getblock** -> fetch a specified payload block
  - **directory_findname** -> fetch directory entry with the given name
  - **pathname_lookup** -> fetch inumber for the file with the given path
- **“YEAH” Hours** (Your Early Assignment Help Hours) Mon. 10/3 7-8PM, location TBD (recorded)

Plan For Today

- **Recap**: filesystems so far
- The Unix V6 Filesystem and Inodes
- **Practice**: reading file data
- Large files and Singly-Indirect Addressing
- **Practice**: singly-indirect addressing
- Large files and Doubly-Indirect Addressing

Lecture 3 takeaway: The Unix v6 filesystem represents small files by storing direct block numbers, and larger files by using indirect addressing - storing 7 singly-indirect and 1 doubly-indirect block number.

Next time: directories, file lookup and links