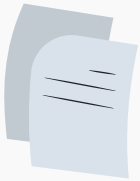
Pintos Project 4: File Systems

The last one!

May 23, 2025

Motivation



Pintos has a very limited file system implementation.

- No support for subdirectories
- Files cannot grow after they are created
- Each file must be allocated in one contiguous space
- Requires external synchronization

In project 4, you will remove these limitations!

Code Base Tour

- Top-level functions in filesys.h/c
- **`struct dir`** abstraction in directory.h/c
- **`struct file`** abstraction in file.h/c
- **`struct inode`** abstraction in inode.h/c
- Free map abstraction in free-map.h/c

These are thread-local

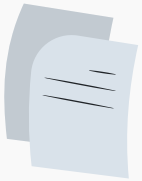
This is global

Current disk layout:

FREE MAP	ROOT DIR INODE	ROOT DIR DATA	FILE 1 INODE	FILE 1 DATA	FILE 2 INODE	FILE 2 DATA

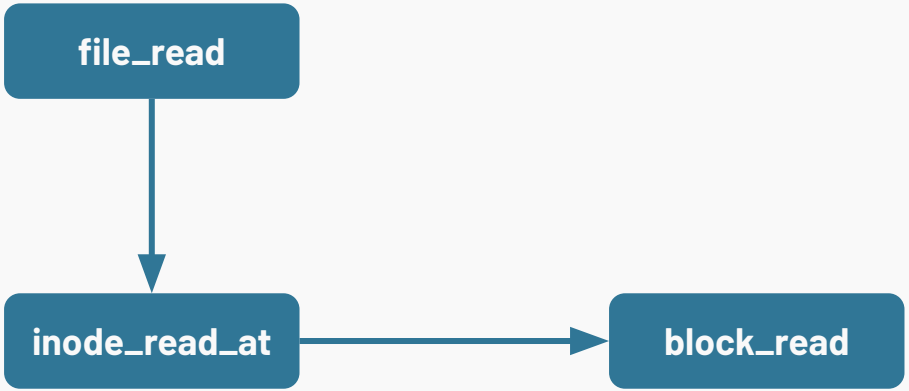
Project Components

Buffer Cache

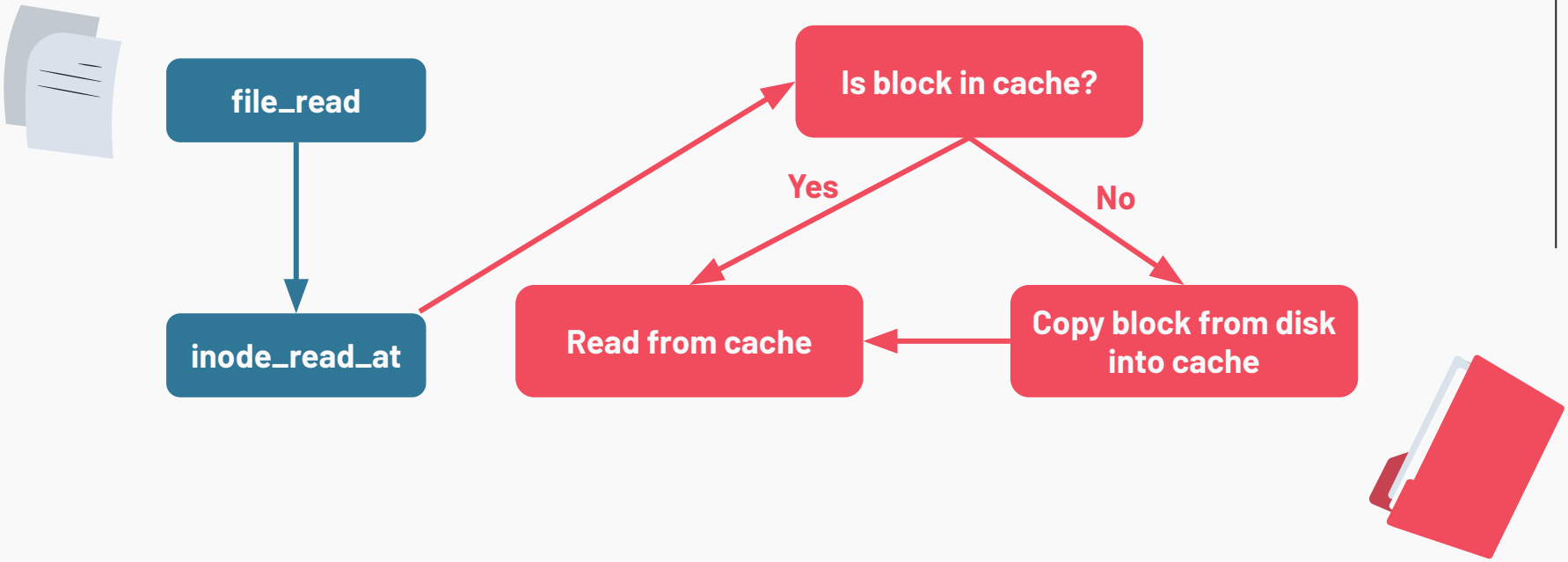


- Currently, all disk accesses use the **block_read()** and **block_write()** functions (check out **inode_read_at()**, for example)
- Instead, we want to create a cache of 64 blocks to store in memory
- Once this is established, all disk accesses should go through the cache
- **Do this first!**

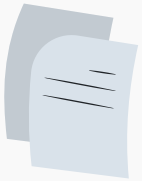
Current Flow



Buffer Cache Flow



Buffer Cache: Implementation



- **Create an interface to your buffer cache** to perform operations on sectors (read, write, evict, etc.)

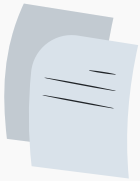
- **Replace direct disk accesses** in the existing code base with your buffer cache. For example...
  - Remove the data attribute from struct inode
  - Eliminate bounce buffer from inode_read_at and inode_write_at
  - Update byte_to_sector to use the buffer cache

- **Implement an eviction policy** like the clock algorithm

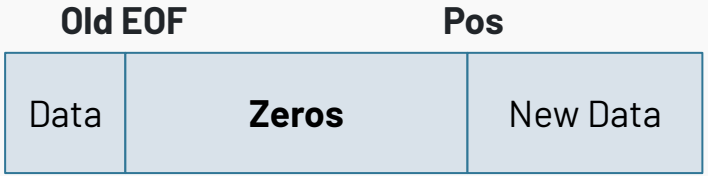
- Implement the **read-ahead** and **write-behind** background threads
  - Optionally, you can do this part later to reduce initial complexity

- To start, you can use a single “buffer cache lock”. Eventually, we’ll want more fine-grained synchronization

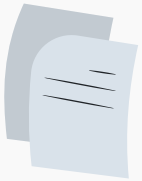
Indexed and Extensible Files



- Currently, files have a fixed length upon creation and must be allocated contiguously
- We will implement direct, singly-indirect, and doubly-indirect blocks within the inode to enable file data to span multiple sectors
  - Implementation tip: write and test these one at a time to avoid debugging lots of code at once!
- File size should start at 0 and grow each time there is a write past EOF. Bytes in between the current EOF and the write position should be filled with zeros.

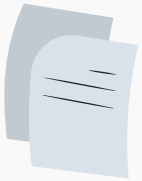


Indexed and Extensible: Implementation



- Modify struct inode_disk to support your indexing scheme (you can use the previously unused bytes)
- Update byte_to_sector, inode_read_at, inode_write_at, inode_create, and other relevant functions
- Should pass file growth tests!

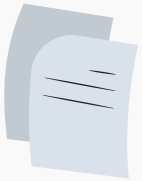
Subdirectories



Implement hierarchical namespace **(e.g. src/filesys/foobar.txt)**.

- Only have to support 14-character file names, but must allow much longer full path names.
- Track the current directory of a process (set to root at startup).
- Ensure child processes inherit the current directory of the parent.
- Path resolution requirements
  - Absolute and relative paths
  - “.” and “..”

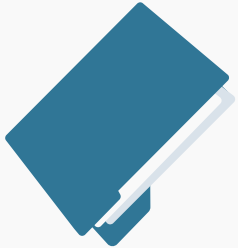
Subdirectories



Update system calls to support directories

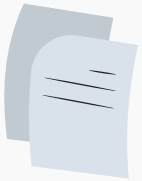
- open() can open directories
- close() can close directories
- remove() can delete empty directories except root

Implement new system calls: chdir, mkdir, readdir, isdir, inumber



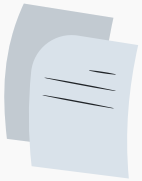
Synchronization

Remove Global Filesys Lock!



- Scary!
- Initially, you can leave the global file system lock, but eventually we want the file system to manage its own synchronization
- Operations on independent entities should not wait for each other

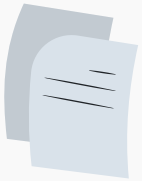
Buffer Cache Synch



- Independent operations on different cache blocks

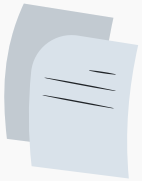
- Allow non-I/O operations while I/O is in progress on other blocks

File Access Synch



- Should allow multiple concurrent readers on same file
- Should allow multiple concurrent writers (when not extending file)
- File extension + write to newly extended region should be atomic

Directory Synch



- Should allow concurrent operations on different directories
- Ensure proper locking for same directory operations