Project 4:
File Systems

CS 212, Winter 2022

Section Outline

- Project Background
- Project Requirements
  - Buffer Cache
  - Indexed & Extensible Files
  - Subdirectories
  - Synchronization
- Getting Started

Background

Motivation

So far, Pintos has operated with a basic file system, with severe limitations:

- No subdirectories
- Files cannot grow (size fixed at creation time)
- File data allocated contiguously (leads to fragmentation)
- Requires external synchronization

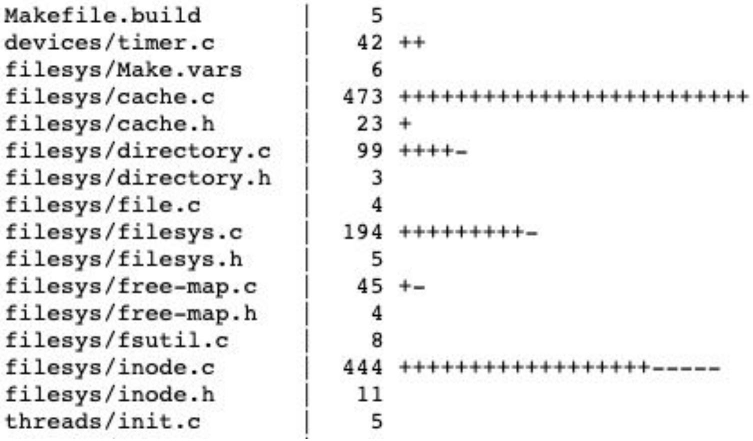
Motivation

So far, Pintos has operated with a basic file system, with severe limitations:

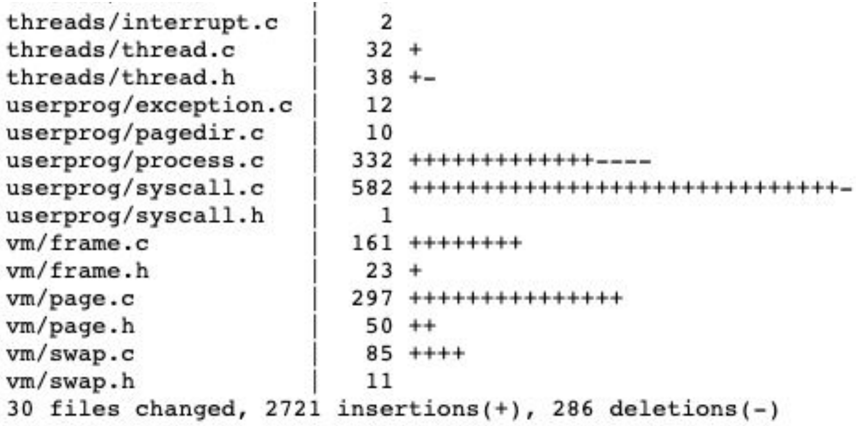
- No subdirectories
- Files cannot grow (size fixed at creation time)
- File data allocated contiguously (leads to fragmentation)
- Requires external synchronization

The goal of project four is to remove these limitations on the file system.

Reference Implementation



```
Makefile.build       |    5
devices/timer.c      |   42 ++
filesys/Make.vars    |    6
filesys/cache.c      |  473 +++++++++++++++++++++++++
filesys/cache.h      |   23 +
filesys/directory.c  |   99 ++++-
filesys/directory.h  |    3
filesys/file.c       |    4
filesys/filesys.c    |  194 +++++++++-
filesys/filesys.h    |    5
filesys/free-map.c   |   45 +-
filesys/free-map.h   |    4
filesys/fsutil.c     |    8
filesys/inode.c      |  444 ++++++++++++++++++-----
filesys/inode.h      |   11
threads/init.c       |    5
```



```
threads/interrupt.c  |    2
threads/thread.c     |   32 +
threads/thread.h     |   38 +-
userprog/exception.c |   12
userprog/pagedir.c   |   10
userprog/process.c   |  332 +++++++++++++----
userprog/syscall.c   |  582 ++++++++++++++++++++++++++++++-
userprog/syscall.h   |    1
vm/frame.c           |  161 ++++++++
vm/frame.h           |   23 +
vm/page.c            |  297 +++++++++++++++
vm/page.h            |   50 ++
vm/swap.c            |   85 ++++
vm/swap.h            |   11
30 files changed, 2721 insertions(+), 286 deletions(-)
```

(Reference solution chose to build on top of project 3.)

Starting Point

Build on top of project 2, or project 3.

- All project 2 functionality must still work.
- If you build on project 3, all project 3 functionality must still work.
  - Must edit `filesys/Make.vars` to enable VM functionality.
- Up to 5% extra credit if you enable with VMs.

Requirements

Buffer Cache

Modify the file system to keep a cache of file blocks.

When a file block is read/written, check cache.

If present, use cache without going to disk.

Otherwise, fetch blocks from disk into cache.

Cache size <= 64 Sectors (including inode/file metadata)

Buffer Cache

Get rid of “bounce buffer” in `inode_{read, write}_at()`

Implement cache replacement policy that is at least as good as the “clock” algorithm.

Buffer Cache

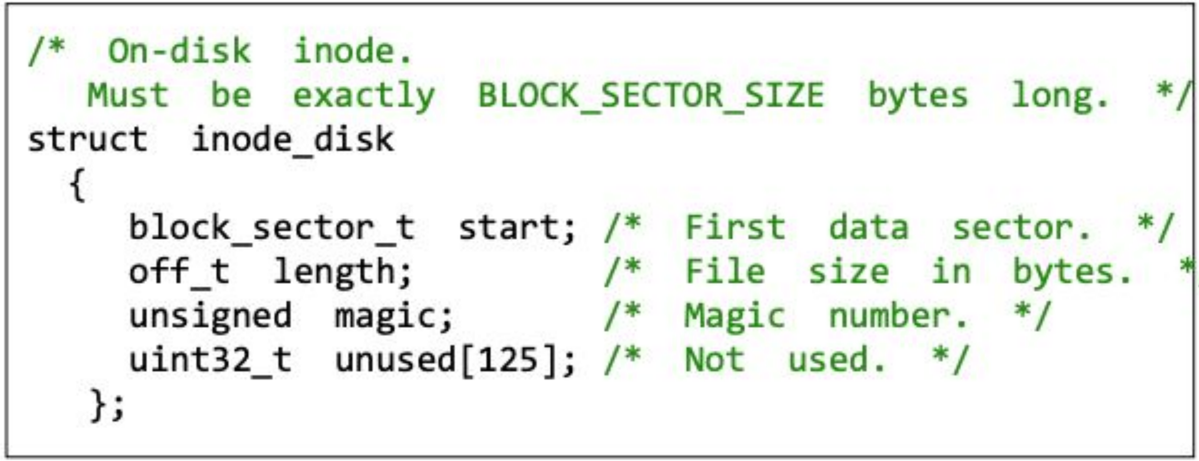
Cache should be:

- **write-behind**
  - Keep dirty blocks in cache
  - Write to disk upon cache eviction
  - Flush all dirty blocks to disk periodically
  - Flush when Pintos halts (in `filesys_done()`)
- **read-ahead**
  - Pre-fetch the next block of a file when the prior block is loaded into the cache.
  - Must be done asynchronously, in the background

Details in **5.3.4 Buffer Cache**

Indexed & Extensible Files

Currently, files are stored in contiguous memory, leading to fragmentation.

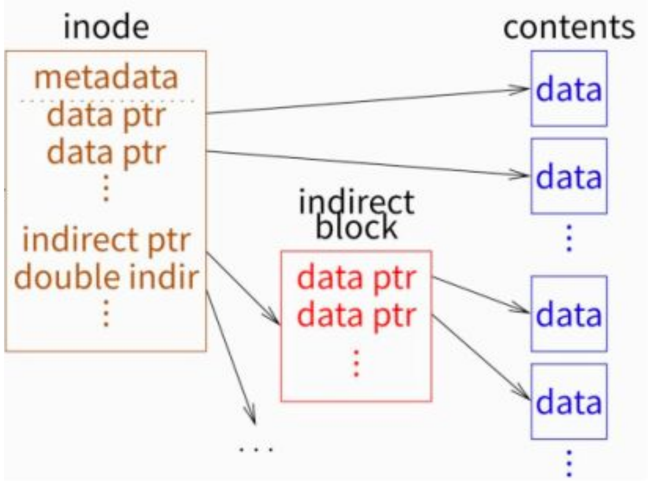


```
/* On-disk inode.
   Must be exactly BLOCK_SECTOR_SIZE bytes long. */
struct inode_disk
  {
    block_sector_t start; /* First data sector. */
    off_t length;         /* File size in bytes. */
    unsigned magic;       /* Magic number. */
    uint32_t unused[125]; /* Not used. */
  };
```

Indexed & Extensible Files

Modify `struct inode_disk` to use index structure, rather than contiguous memory.

In practice, this likely means using direct, indirect, and doubly indirect blocks.



Indexed & Extensible Files

Requirements:

- Must support files of size up to entire file partition (minus metadata).
  - Partition is up to 8 MB
  - Each inode is stored in one disk sector → doubly indirect blocks are needed.
- Implement file growth
  - Files start with size 0.
  - File grows whenever a write is made past EOF.
  - Directory files can also grow up to size of entire file partition.
  - Writing past EOF extends the file to byte being written. All bytes between old EOF and new write are zeroed.
  - Optional: support “sparse” files where entirely zero blocks are allocated lazily.

Details in **5.3.2 Indexed and Extensible Files**

Subdirectories

Implement hierarchical name space. (e.g. “/foo/bar/foobar.txt”)

Directory entries (`struct dir_entry`) point to files or other directories

Maintain “current directory” for every process.

Set to root at startup, inherited by child processes from parent.

Subdirectories: System Calls

- Path resolution: Update every system call that takes filenames to also accept absolute and relative paths.
  - Support filenames "." and ".."
  - No limit on path length. Optional 14-character limit on filenames.
- Update existing system calls:
  - `open()` - can open directories
  - `close()` - can close directories
  - `remove()` - can delete empty directories
- New system calls:
  - `chdir, mkdir, readdir, isdir, inumber`

Details in **5.3.3 Subdirectories**

Synchronization

Eliminate need for external synchronization.

- No more global file system lock.
- Operations on independent entities should be independent.

Synchronization

Details:

- Operations on different cache blocks must be independent.
- Multiple processes must be able to access the same file at once.
  - Multiple reads must not wait on each other.
  - Multiple writes must not wait on each other, if file is not growing.
    - Data may be interleaved.
  - Reading during write may show that all, some, or none of the data has been written
  - Writes that extend a file must be atomic.
- Operations on different directories must be independent.
  - Operations on same directory may be serialized.
  - “Operations on directory” does not include writing/reading from file within a directory.

Details in **5.3.5 Synchronization**

Getting Started

Suggested Order of Implementation

1. Buffer cache
   a. After implementation, all proj2 (and proj 3, if enabled) tests should still pass.
2. Indexed & Extensible Files
   a. After implementation, file growth tests should pass.
3. Subdirectories
   a. After implementation, directory tests should pass.
   b. Can be done mostly in parallel with extensible files, if you temporarily make the number of entries in directories fixed.

Think about synchronization throughout implementation.

Advice

- Start early!
- Design first!
  - Decide how you will design each aspect of the project before you start implementing.
  - Focus on Buffer Cache early in the design process.
- Be willing to change your design
  - If things are getting very complicated during implementation, take a step back. Is there a simpler way to accomplish your goal?
- Pay attention to synchronization while designing and implementing
  - Make sure to avoid deadlock by avoiding circularity in graph of synchronization requests.
  - Organize synchronization mechanisms hierarchically.
- Focus on general code quality throughout implementation

Questions?