Project 4: File Systems

Josh Cho

2023-03-03 Tue

Outline

Motivation

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

- No subdirectories
- Files can't grow, fixed file size
- File data allocated contiguously, leading to fragmentation
- Requires external synchronization

We want to remove these limitations.

Starting Point

Build on top of Project 2 or 3.

- All project 2 (or 3) functionality must still work.
- If you build on project 3, edit `filesys/Make.vars` to enable VM.
- Enabling VM gives you up to 5% extra credit.

Overview

Reference solution builds on top of Project 3

```
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(-)
```

Tips

- Reuse existing systems. Look into `fsutil.c/h` and previous projects.
- Make sure to review your grading reports for Project 1 and Project 2.
- Design and style matters. At this point you should have a sense of what is elegant & inelegant design of systems (and how the different parts work together).
- Do a lot of design work beforehand

Requirements

- Buffer cache (tip: integrate the cache into design early)
- Indexed & Extensible files
- Subdirectories
- Synchronization

Buffer cache

Implement cache for file blocks. When a file block is read or written, check the cache:

- If present, use the cache.
- If not present, fetch blocks from disk.

Cache size is $\leq$ 64 sectors (`BLOCK_SECTOR_SIZE`: 512 bytes).

- This cache size includes inode and file metadata

Buffer cache

- To get started, remove the "bounce buffer" in `inode_{read, write}_at()`.
- Cache replacement policy must be at least as good as the clock algorithm.

Buffer cache

We want a cache that is:

- Write-behind
  - keep dirty blocks in cache and write to disk upon cache eviction
  - write to disk (flush) periodically
  - also flush on `filesys_done`
- Read-ahead
  - automatically fetch the next block of file
  - do this asynchronously

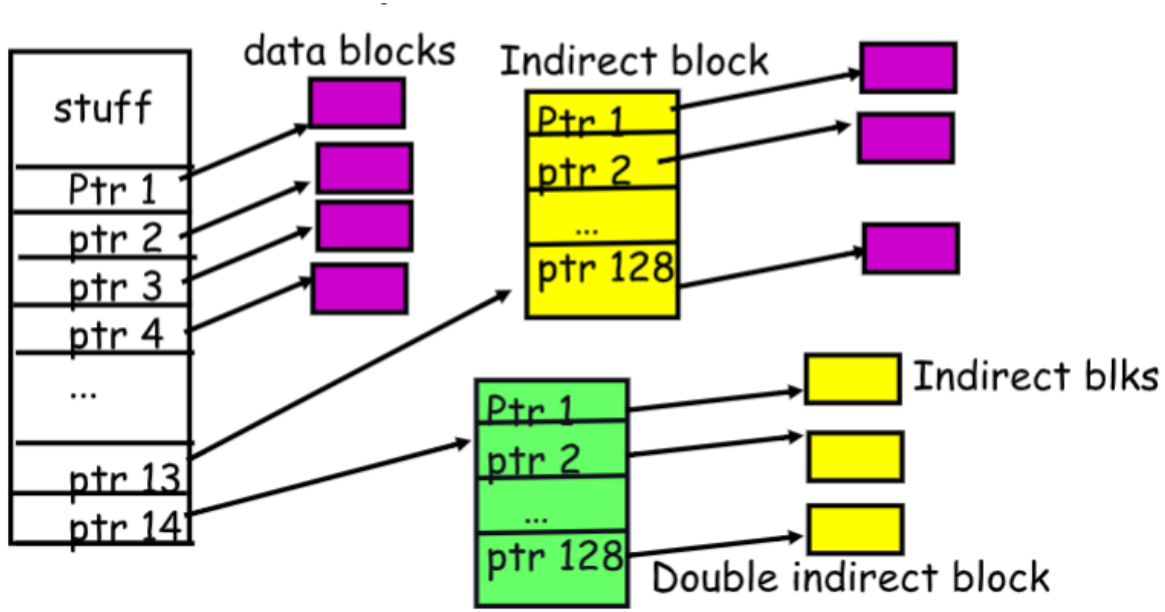
Indexed & Extensible files

Stored contiguously currently. Modify this struct to use index structure.

```
/* On-disk inode.
   Must be exactly BLOCK_SECTOR_SIZE bytes long. */
struct inode_disk
  {
    block_sector_t sectors[SECTOR_CNT];
    enum inode_type type;
    off_t length; /* File size in bytes. */
    unsigned magic;
  };
```

Indexed files

Implement direct/indirect/doubly indirect indexing. This should enable files of size up to entire file partition (8MB).



Extensible files

Implement stack growth.

- Files start with size 0
- When write is made past EOF, grow. Files can grow to size up to entire file partition.
- Zero out all bytes between old EOF and new write.
- (Optional) Support "sparse" files where zero blocks are allocated lazily.

Subdirectories

Implement hierarchical namespace.

- e.g. `/foo/bar/foobar.txt`
- Directory entries should point to files or other directories
- Maintain current directory for each process
  - Set to root on startup
  - Child processes via `exec` inherit current directory of parent

Subdirectories: Syscalls

1. Path resolution: support both absolute and relative paths
   - Support "." and ".."
   - No limit on path length, but optional 14-character limit on filenames
2. Update existing system calls to support directories
   - `open()` opens a directory
   - `close()` closes a directory
   - `remove()` deletes any empty directory (except root)
3. Implement new system calls
   - `chdir`, `mkdir`, `readdir`, `isdir`, `inumber`
   - See 5.3.3 Subdirectories for more details

Synchronization

Remove need for external synchronization.

- No more global filesys lock

Implement finer-grained synchronization strategy.

- Operations on independent entities should be independent

Synchronization

1. Operations on different cache blocks should be independent.
2. Multiple processes must be able to access a single file at once.
   - Multiple reads should not wait on each other
   - Multiple writes without file extension should not wait on each other (data may be interleaved)
   - A read of a file by one process when the file is being written by another is allowed to show none, all, or part of the write
   - Writes that extend file must be atomic.

Synchronization

Operations on different directories must be independent.

- Operations on same directory may wait for one another
  - Note this does NOT mean operations on the *files* of the directory!

Suggested Order of Implementation

1. Buffer cache
   - After implementation, all Project 2 (or if enabled, 3) tests should pass
2. Indexed & Extensible files
   - After implementation, all file growth tests (`grow-`) should pass
3. Subdirectories
   - After implementation, all directory tests (`dir-`) should pass

Debugging Tips

1. Isolate the problem
2. Don't be afraid to change different parts of the system
3. Return with a fresh perspective

Questions

Any questions?