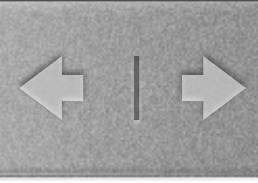



CS 415
OS Practicum

Project 4: Filesystems

sub: Oliver Kennedy (okennedy@cs)

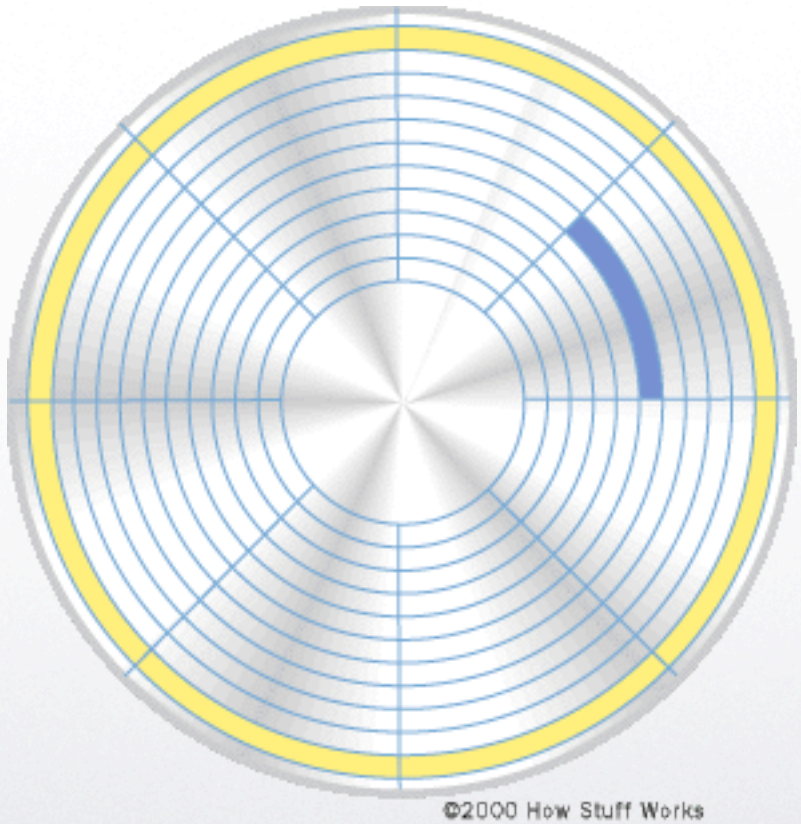
Project 4: Filesystems

- What are filesystems and why do we use them?
- How do we design our filesystem?
- What do you need to know to implement your filesystem?

The Hard Disk

- A hard disk is composed of one or more platters.
- Each platter is divided up into concentric rings called tracks.
- Each track is broken up into chunks called sectors, or blocks.
- Typically sectors on multiple tracks overlap.

A Hard Disk



What kind of speed properties do hard disks have?

- high bandwidth, high latency

Clustered accesses are better

Why do we transfer entire sectors?


The Hard Disk

- Hard drives are high latency.
  - ... but also high bandwidth.
- Reading individual bytes is slow.
  - We read a sector at a time (4k).

The Virtual Disk

- disk_create(*disk, name, size, flags)
  - Creates a virtual disk, backed to a file.
  - size is in 4k blocks.
- disk_startup(*disk, name)
  - Opens a virtual disk created with disk_create.
- disk_read_block(*disk, blocknum, buffer)
- disk_write_block(*disk, blocknum, buffer)

read and write return before the operation is complete

how do we know an operation is done?

The Virtual Disk

- install_disk_handler(handler)
- disk_send_request(...)
  - The raw request interface; you don't need this.
- The first parameter is passed by reference.
- How do you identify a request in the handler?

Identify the request with the buffer pointer! This MUST be unique for all accesses!

Filesystems

- This interface is not programmer-friendly.
  - Most programmers don't like asynchronous interfaces.
  - Restricting file sizes to 4k and filenames to block numbers is just bad.
- Filesystems place a friendly face on the raw disk interface.
- Let’s design our filesystem...
  - What do we need?

Disclaimer: The *basic* filesystem we’re asking you to write is considered to be one of the hardest projects in this course. We’ll be glad to discuss optimizations and the state of the art offline.

- A way to link blocks together for large files.
- A way to 'name' blocks.
  - Some sort of directory structure.
- A way of keeping track of free blocks
- A way to make this structure persistent.

We need...

- A way to link blocks together for large files.
- A way to 'name' blocks.
  - Some sort of directory structure.
- A way of keeping track of free blocks.
- A way to make this structure persistent.

A way to link blocks...

- Let's use some special indexing blocks; we can call them inodes.
- Use an inode as an array of data block numbers.
  - but there's still a problem...
- Use the 1024th block pointer as a pointer to the next inode.
- More advanced solutions:
  - Use indirection: Store pointers to 1024 inodes that point to blocks.
  - Hybrid approach: store 512 block pointers and 512 indirect pointers (or subdivide even further).

Note that inode pointer != memory pointer

What does an inode stand for? Well, a while back some Apple engineers wanted to create these pink designer databases....

Err... Well, it's not much more accurate, but according to Wikipedia:

When asked, Unix pioneer Dennis Ritchie replied:

'In truth, I don't know either. It was just a term that we started to use. "Index" is my best guess, because of the slightly unusual file system structure that stored the access information of files as a flat array on the disk, with all the hierarchical directory information living aside from this. Thus the i-number is an index in this array, the i-node is the selected element of the array. (The "i-" notation was used in the 1st edition manual; its hyphen became gradually dropped).'

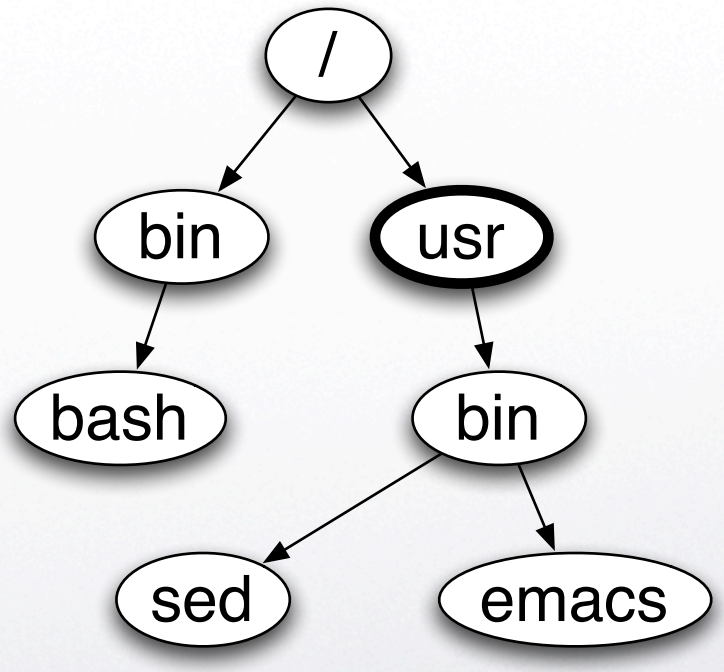
A way to name blocks...

- We want
  - A structure to index our files.
  - A way to identify a node.
  - A way to quickly list the children of a node.
  - A way to determine if a node is a leaf.

A flat filesystem (a list of files) is all well and good, but sometimes you want more organization
We want to introduce a hierarchy; a tree-like structure on the filesystem.

A way to name blocks...

- Hierarchical Filesystems
  - [/][bin][/][bash]
  - /usr/[bin/]emacs
  - []bin/sed



The user can specify a ‘path’ and this translates to a tree traversal.
Node names are separated by slashes
A leading slash indicates that the path starts from the ROOT
No leading slash indicates that the path starts from the WORKING DIRECTORY (more on that later)
.. as a node name indicates the PARENT directory
Files may only be leaf nodes.


Storing the Names

- The Unix approach: Directory files.
  - Use an ordinary file to store filename->file_inode mappings.
  - Potential problem!
- Filesystems store other information with each mapping.
  - What else should you store?

Problem: How do you identify whether a file is a file or a directory (hint: see below)

Other things to store: File type (File or directory), Not required for project, but also Permissions, Owner/Group

Path Traversal

- 1) Set *ans* to the start of the search.
  - Root directory for now.
- 2) Read the 'file' at *ans*.
- 3) Find the mapping that corresponds to the next entry in the path.
- 4) Set *ans* to the inode the entry refers to.
- 5) Repeat 2-4 for all the path elements.

Some more block types

- Free-list:
  - What do we need?
    - Simple to maintain.
    - Fast: O(1) on all modifications.
- Root Block/Superblock
  - Stores pointers to all the inodes you need to maintain the system. (Which do you need?)
  - Stored in a well known location. (block 0)

A stack is the ideal approach to a free list
Thought questions:
How do you implement a stack of free blocks?
How do you initialize this stack?

What pointers do you need in the superblock?

Design Overview

- Root Block:
  - Unique Identifier.
  - Pointer to the root directory.
  - Pointer to the first free block.
- Freelist Block:
  - Pointer to the next free block.
- Inode:
  - 1023 child inodes.
  - 1 overflow node pointer.
- Data Block

Synchronization

- The hard disk controller may reorder requests as it sees fit.
  - Multithreaded writes should remain consistent.
- The hard disk may fail at any time.
  - It's ok to 'lose' free blocks after a crash.
  - It's not ok to lose user data.
  - All changes should be 'atomic'.

Why is it ok to lose free blocks?


You there, in the back wake up!

- How do you...
  - Create a directory?
  - Create a file?
  - Write a block to a file?


Project 3

minifile_t **minifile_creat**(filename)

minifile_t **minifile_open**(filename, mode) *//r, w, or rw*

minifile_read(file, data, len) *//should block until the read succeeds*

minifile_write(file, data, len) *//should block until the write succeeds*

minifile_close(file)

minifile_unlink(filename) *//delete a file*

minifile_mkdir(dirname)

minifile_rmdir(dirname) *//delete a directory*

int **minifile_stat**(path) *//filesize = file; -1 = doesn't exist; -2 = dir*

char ****minifile_ls**(path) *//assume the caller will free the array and each element*

your interrupt handler

What's an open file?

- Pointer to the file's inode
- Cursor
- Open (reference) count
- Read/Write lock
- What else?

Other things to put in:

Linked list pointers so you can keep all the open files in a linked list for easy lookups
Cache the file's inode(s), or at least the one which the cursor is in.
Cache file metadata: file size?

Working directories

- minifile_cd(path)
- char *minifile_pwd()
- Each thread has a separate working directory
  - It's ok to store this in minithread_t and access it via minithread_self()
  - Typically this would be stored per-process.

mkfs.exe

- disk_create()
- Create the superblock
- Create /
- Initialize the free list

mkfs is a separate application. You should either create a 2nd makefile which generates mkfs or modify the first makefile so it generates both mkfs.exe and minithread.exe


Some final words

- Writes may not be aligned to block boundaries
  - You may need to read one or more blocks before writing
- Synchronization, synchronization, synchronization
  - ... via semaphores/mutexes/interrupt disabling
  - ... via enforced ordering
- Don't assume a block has been written to disk until the write response returns.
  - You'll need a lookup table here!

More final words

- The disk must not enter an inconsistent state
  - 'lost' free blocks are ok
  - Inodes pointing to free blocks are not
  - Write data first, then propagate updates back up the directory tree.
- Synchronize your root block!
  - Only one root block write pending at any given time!

One last reminder...

- Design Docs due early the week after Spring Break (Week of the 26th)
- Start coding early.
- Think before you code.

fin