CS 1217

File Systems, File System Data Structures

Logistics

- End semester exam on 3rd May, 3 – 5 pm, Takhshila
- Final Lecture on Tuesday, 2nd May 3 - 5 pm, venue TBD
- Zines are due on May 6th, Saturday 5 pm
  - To be done in groups
  - 1 submission / group
- No synchronization assignment; zines are now 5% (again)

Percent Distribution

Assignments	25%
Midterm Exam	25%
Endterm Exam	36%
Quizzes	9%
~~Synchronization Assignment~~	~~0%~~
Zines	5%
Total	100%

Aside

How to send and reply to email

[article index] [email me] [@mattmight] [rss]

The problem with email is that people think it's electronic mail.

Email is *not* mail in electronic form. You are not writing a letter.

https://matt.might.net/articles/how-to-email/

Ryan Martin Ph.D.

All the Rage

Avoiding the Angry Email

Why We Should Think Twice Before Firing Off that Rage Filled Message

Posted February 27, 2013

I am sure most of us have been there at some point. We notice a mistake one of our coworkers made, learn about a frustrating decision made by one of our elected officials, or even find out that a family member didn't do something they promised, and we fire off an angry email to the culprit without really thinking it through.

I've both sent such emails and received them. In fact, as a college professor, I get these sorts of emails from students more than I would like. The typical pattern is that a student gets a bad grade on something or doesn't agree with a decision I've made and quickly fires off an angry email to try and resolve the situation (or sometimes just to complain about it).

https://www.psychologytoday.com/intl/blog/all-the-rage/201302/avoiding-the-angry-email-0

Need for File Systems

- Low-level disk interface is very limited:
- Requires reading and writing entire 512-byte blocks. **Why**?
- No notion of files, directories, etc.

- File systems take what disks offer, and create the ***file system abstraction*** in software

Flash is Great, right?

- Not really
- The underlying technology has its own set of limitations

- Before writing, large chunks might need to be “erased”
- In-place writing is not possible!
- Cannot write to a cell more than a fixed number of times!

- Lesson: Every technology has its limitations, need to write s/w with that in mind

Concepts of Files

- The semantics of file have a variety of sources what are worth separating:
- **Just a file:** the minimum it takes to be a file.
- **About a file:** what other useful information do most file systems *typically* store about files?
- **Files and processes:** what additional properties does the UNIX file system interface introduce to allow user processes to manipulate files?
- **Files together:** given *multiple* files, how do we organize them in a useful way?

Expectations from Files

- What does a file have to do to be useful?
- **Reliably** store data : Disks fail **all** the time!
- Be located-able! Usually via a **name** : Humans organize information better via names.

Expectations from Files

- At minimum we expect that:
- file contents *should not* change unexpectedly.
- file contents *should* change when requested and as requested.

File Systems

- Low-level disk interface is very limited:
  - Requires reading and writing entire 512-byte blocks.
  - No notion of files, directories, etc.
- File systems take this limited block-level device and create the **file abstraction** almost *entirely in software*.
- Compared to the CPU and memory **more** of the file abstraction is implemented in software.
  - Led to a large number of file systems: ext2,3 and 4, reiserfs, NTFS, xfs, etc

File and Processes

- Many file systems provide an interface for establishing a **relationship** between a process and a file.
  - "I have the file open. I am using this file."
  - "I am finished using the file and will close it now."
- Why does the file system want to establish these process-file relationships?
  - Can improve **performance** if the OS knows what files are actively being used by using *caching* or r*ead-ahead*.
  - The file system may provide **guarantees** to processes based on this relationship, such as exclusive access.

File Location : Unix Semantics

- UNIX semantics simplify reads and writes to files by storing the file position for processes.

Unix File Interface

- Establishing relationships:
  - `open("foo")`
    - "I'd like to use the file named foo."
  - `close("foo")`
    - "I'm finished with foo."
- Reading and writing:
  - `read(2)`:
    - "I'd like to perform a read from file handle 2 at the current position."
  - `write(2)`:
    - "I'd like to perform a write from file handle 2 at the current position."
- Positioning:
  - `lseek(2, 100)`:
    - "Please move my saved position for file handle 2 to position 100

File Organization

- Each file has to have a unique name

- Letter to Mother : LetterToMother.txt
- Letter to Friend: LetterToFriend1.txt
- Letter to Another Friend: LetterToAnotherFriend.txt
- Another letter to mother: LetterToMother2.txt

- Flat name spaces were actually used by some early file systems

Hierarchical Organization

- Big idea: don’t look at **everything** all at **once**. Allows users to store and examine related files together.
- Letters/mother/1.txt
- Letters/Friends/Friend1/1.txt
- Letters/Cat/1.txt
- Letters/Dog/1.txt
- Letters/Friends/Friend2/1.txt
- Letters/mother/2.txt

- Each file is in **one** place.

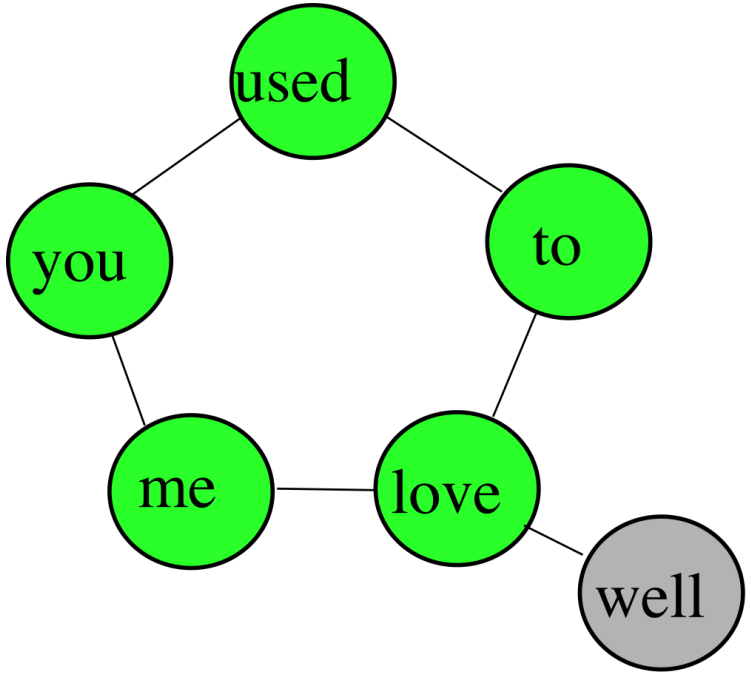
Hierarchical Organization Implications

- Location requires navigation
  - *relative* navigation is useful
  - locations (**directories**) can include pointers to other locations (**other directories**).
- Location is only meaningful if it is tied to a file's **name**
  - hierarchical file systems implement **name spaces**
  - require that a file's name map to a single unique location within the file system

Navigation using Data Structures

- File systems usually require that files be organized into an ***acyclic*** graph with a single ***root***, also known as a ***tree***.

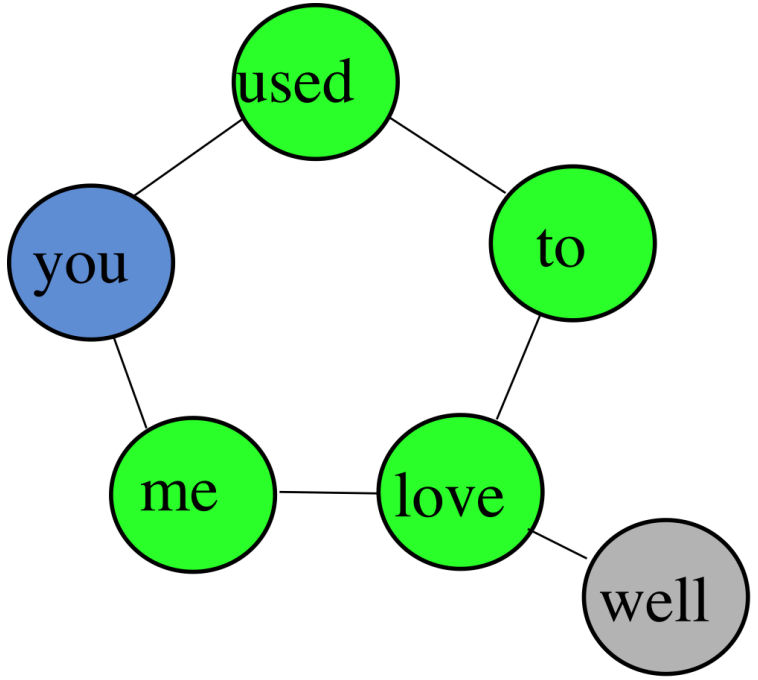
- Why?



Navigation using Data Structures

- File systems usually require that files be organized into an *acyclic* graph with a single *root*, also known as a *tree*.

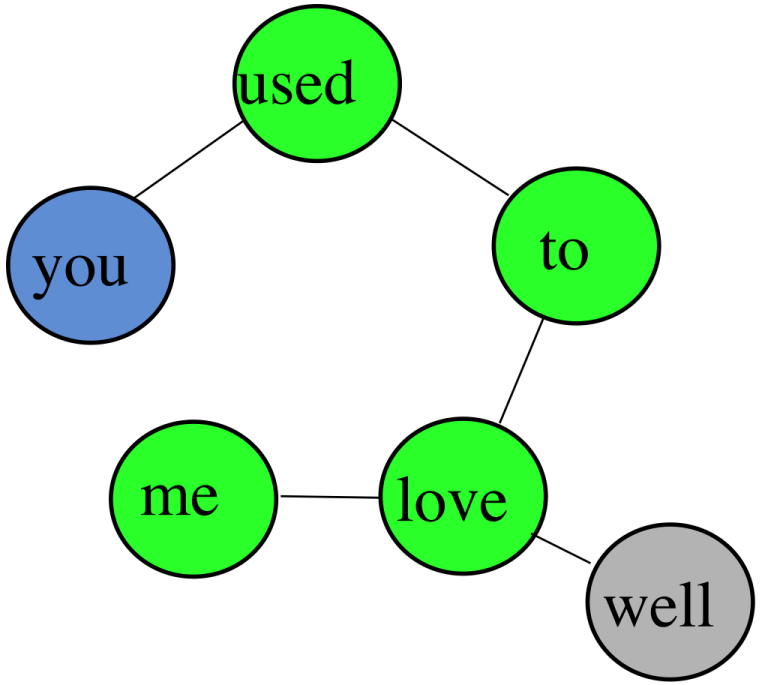
- Pick a root



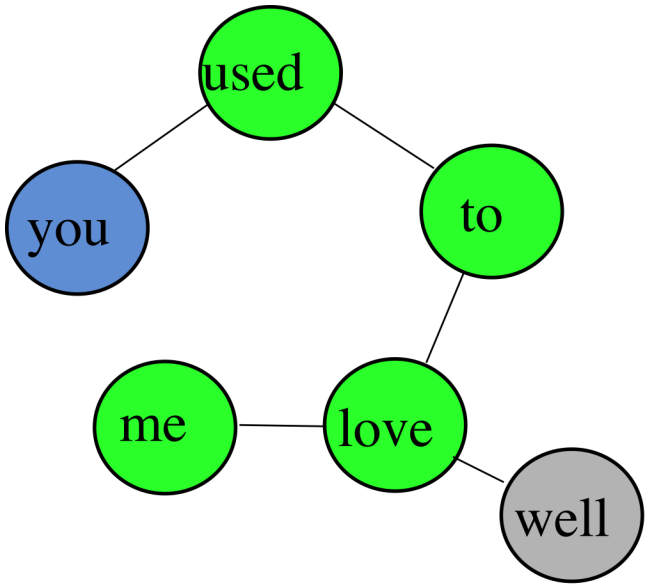
Navigation using Data Structures

- File systems usually require that files be organized into an *acyclic* graph with a single *root*, also known as a *tree*.

- Eliminate cycles



Tree Naming



- Trees produce a single canonical name for a file system
- What happens to relative names?
- **Canonical name:** /you/used/to/love/well
- **Relative name:** /you/used/to/love/me/../well
- **Relative name:** love/me/../../love/me/../well

File System Design Goals

- Efficiently **translate** file *names* to file *contents*.
- Allow files to **move**, **grow**, **shrink** and otherwise **change**.
- Optimize access to **single** files.
- Optimize access to **multiple** files, particularly related files.
- **Survive** failures and maintain a consistent view of file names and contents.

File System Features

- The file systems in our discussions all support the following features:
- **Files**, including some number of file attributes and permissions.
- **Names**, organized into a **hierarchical** name space.

- The interface and feature set remains consistent (mostly)
- The changes happen mostly in implementation / optimization

File System Implementations

- What makes file systems different?
- **On-disk layout**
  - How does the file system decide where to put data and metadata blocks in order to optimize file access?
- **Data structures**
  - What data structures does the file system use to translate names and locate file data?
- **Crash recovery**
  - How does the file system prepare for and recover from crashes?

Why is this hard: write example

- A process wants to write data to the end of a file. What does the file system have to do?
  - Find empty disk blocks to use and mark them as in use.
  - Associate those blocks with the file that is being written to.
  - Adjust the size of the file that is being written to.
  - Actually copy the data to the disk blocks being used.
- From the perspective of a process all of these things need to happen **synchronously**.
- In reality, **many different asynchronous** operations are involved touching many different disk blocks.
- This creates both a consistency and a performance problem!

On Disk Happenings

- Let's consider the **on-disk structures** used by modern file systems.
- Specifically we are going to investigate how file systems:
  - **translate** paths to file *index nodes* or inodes.
  - **find** data blocks associated with a given inode (file).
  - **allocate and free** inodes and data blocks.

- We're going to try and keep this at a relatively high level, but draw from ext4 file system