CSCI [4|6]730 Operating Systems

File System

Chapter 10-11: File System

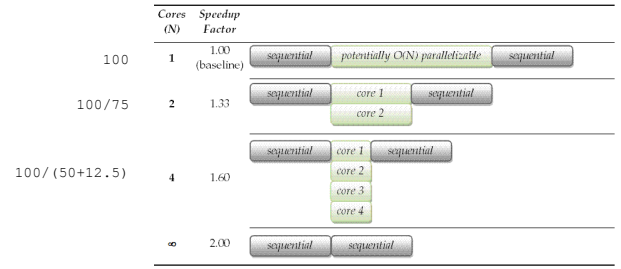
- What are files? What is file meta-data?
- How are directories organized?
- What operations can be performed on files?
- How are directories organized?
- What is the difference between hard & soft links?
- How are files protected?

Motivation: I/O is Important

- How could we get applications to run faster?
  - 'speedup' applications by running them on multiple processors:
    - 1 processor runs in 10 seconds.
    - 4 processors runs in 5 seconds.
    - Speedup = T(1)/ T(n) = 2.
- Applications have two essential components:
  - Processing
  - Input/Output (I/O)
- I/O performance is the bottleneck and therefore it predicts application performance

I/O performance predicts application performance

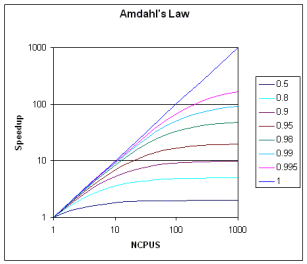
- Amdahl's Law: (Speedup is limited by the slowest component) *For a fixed problem size* - if continually improve only part of application (e.g., processing), then achieve diminishing returns in speedup.



  - Another Way to Look at IT: infinite speedup and affect only 15% of the overall task roughly: 1/(1-0.15) =1.18 times faster is max!

I/O performance predicts application performance

- Amdahl's Law: If continually improve only part of application (e.g., processing), then achieve diminishing returns in speedup
  - Example: infinite speedup and affect only 15% of the overall task roughly: 1/(1-0.15) = 1.18 times faster is max!
- `f`: portion of application that is improved (e.g., processing)
- $speedup_f$: speedup of a portion of application
- $Speedup_{Application} = 1/ ((1-f) + (f/speedup_f))$
  - Examples:
    - f = .15, $speedup_f$ = 2, $speedup_{app}$ = 1.08
    - f = 1/3, $speedup_f$ = 2, $speedup_{app}$ = 1.20
    - f = 1/2, $speedup_f$ = 2, $speedup_{app}$ = 1.33




```
Example: When only 10% of the application is
sequential the maximum speedup using infinite
number of processors is 10.

1/(1-9/10) = 10.
```


10% (.1)	10	5.26
10% (.1)	20	6.90
10% (.1)	100	9.17
10% (.1)	100,000	9.99 (~10)
25% (.25)	5	2.50
25% (.25)	10	3.08
25% (.25)	20	3.48
25% (.25)	100	3.88
25% (.25)	100,000	3.99 (~4)
40% (.40)	5	1.92
40% (.40)	10	2.17

Gene Ahmdahl

Role of OS for I/O

- Standard library
  - Provide abstractions, consistent interface
  - Simplify access to hardware devices
- Resource coordination
  - Provide protection across users/processes
  - Provide fair and efficient performance
    - Requires understanding of underlying device characteristics
- User processes do not have direct access to devices
  - Could crash entire system
  - Could read/write data without appropriate permissions
  - Could hog device unfairly
- OS exports higher-level functions
  - File system: Provides file and directory abstractions
  - File system operations: `mkdir`, `create`, `read`, `write`

Abstraction: *File*

- User view
  - Named collection of bytes (defined by user)
    - Untyped or typed
    - Examples: text, source, object, executables, application-specific
  - Permanently and conveniently available
- Operating system view
  - Map bytes as collection of blocks on physical non-volatile storage device
    - Magnetic disks, tapes, NVRAM, battery-backed RAM
    - Persistent across reboots and power failure

Files Attributes: *Meta-Data*

System information associated with each file:

- *Name* – only information kept in human-readable form.
- Type – needed for systems that support different types.
- Location – pointer to file location on device/disk.
- Size – current file size.
- Protection bits – controls who can do reading, writing, executing.
- Time, date, and user identification – data for protection, security, and usage monitoring.
- Special file?
  - Directory, Symbolic link…

Meta-data is stored on disk:

- Conceptually: meta-data can be stored as an array on disk (e.g., directory)

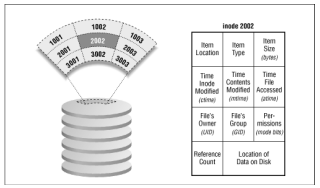
```
{atlas:maria:143} ls -lig ch11.ppt
231343 -rw-r--r--   1 profs     815616 Nov  4  2002 ch11.ppt
```

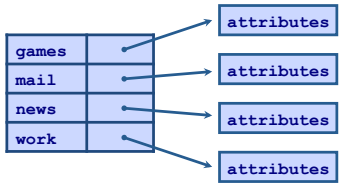

Directory Implementation

- Directory system function: Maps ASCII names onto what is needed to locate the data
- Where do we store the files' attributes?
  - A simple directory: fixed sized entries attributes stored with the entry

games	attributes
mail	attributes
news	attributes
work	attributes

  - Directory in each entry just refers to an i-node (UNIX implementation)





Directory Structure

- A directory "file" is a sequence of lines; each line holds an i-node number (index-node) and a file name

895690	"."
288767	".."
287243	"maria.html"
287259	"gunnar.txt"

- The data is stored as binary so we cannot simply `cat` to view it:
  - but some UNIXs allow an "octal dump" (other formats also available) :

```
{atlas:maria:187} od -c .
0000000  \0  \r 252 312  \0  \f  \0 001   .  \0  \0  \0  \0 004   g 377
0000020  \0  \f  \0 002   .   .  \0  \0  \0 004   b 013  \0 024  \0  \n
0000040   m   a   r   i   a   .   h   t   m   l  \0  \0  \0 004   b 033
0000060  \0 024  \0  \n   g   u   n   n   a   r   .   t   x   t  \0  \0
```

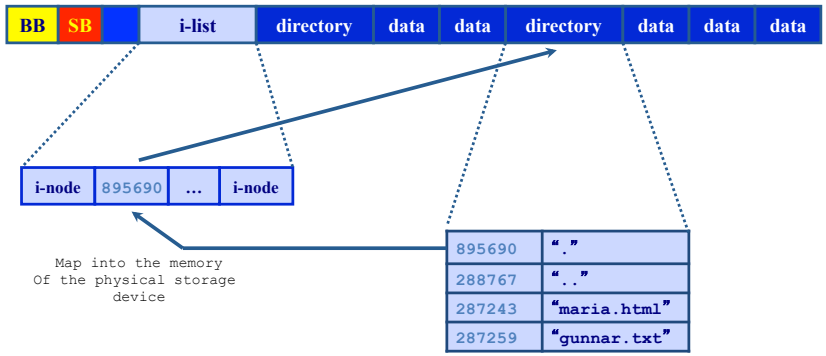
Directory Organization

- Organization technique: Map file name to blocks of file data on disk
  - Actually (indirectly), map file name to file meta-data (which enables one to find data on disk)
- Simplest approach: Single-level directory
  - Each file has unique name
  - Special part of disk holds directory listing
    - Contains <file name, meta-data index> pairs
    - How should this data structure be organized???
- Two-level directory
  - Directory for each user
  - Specify file with user name and file name
  - Disadvantage: Each user cannot organize files

Directories: *Tree-Structured*

- Directory listing contains <name, index>, but name can be directory
  - Directory is stored and treated like a file
  - Special bit set in meta-data for directories
    - User programs can read directories
    - Only system programs can write directories
  - Specify full pathname by separating directories and files with special characters (e.g., \ or /)
- Special directories
  - Root '/' : Fixed index for meta-data (e.g., 2)
  - This directory: .
  - Parent directory: ..
- Example: `mkdir /a/b/c`
  - Read meta-data 2 '/' (by default 2 is root in linux), look for "a": find <"a", 5>
  - Read 5, look for "b": find <"b", 9>
  - Read 9, verify no "c" exists; allocate c and add "c" to directory

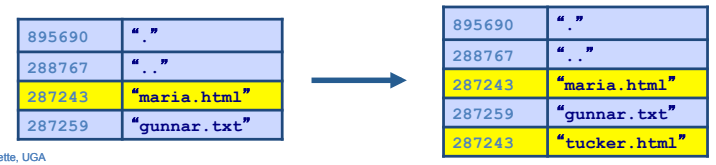
File System Expanded



Acyclic-Graph Directories

- More general than tree structure
  - Add connections across the tree (no cycles)
  - Create links from one file (or directory) to another
- Hard link: "`ln a b`" ("a" must exist already)
  - Idea: Can use name "a" or "b" to get to same file data
  - Implementation: Multiple directory entries point to same meta-data

```
link( "maria.html", "tucker.html" );
```



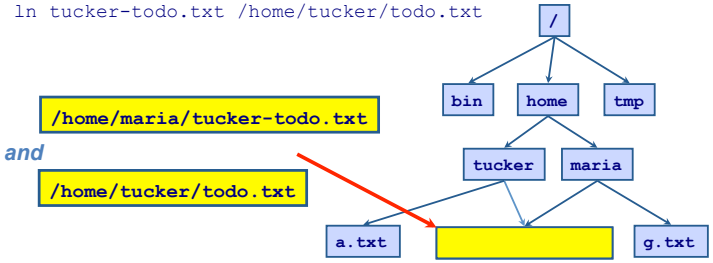
Why Links?

- A link is a pointer to a file
- Useful for sharing files;
  - A file can be shared by giving each person their own link (pointer to it)

```
ln <existing-file> <new-pointer>
```

- Maria types, in directory: ~/maria

```
ln tucker-todo.txt /home/tucker/todo.txt
```



Creating Links

- Changes to a file affect every link:

```
{atlas} cat file_a
This is file A.
{atlas} ln file_a file_b
{atlas} cat file_b
This is file A
{atlas} echo "appending this to b" >> file_b
{atlas} cat file_b
This is file A.
appending this to b
{atlas} cat file_a
This is file A.
appending this to b
```


Seeing Links

- Compare status information :

```
{saffron:maria:104} ls -l file_a file_b file_c
-rw-r--r-- 2 maria 36 May 24 10:52 file_a
-rw-r--r-- 2 maria 36 May 24 10:52 file_b
-rw-r--r-- 1 maria 16 May 24 10:55 file_c
File mode, # links, owners name, group name, #bytes, date, pathname
```

- Look at i-node number:

```
{saffron:maria:105} ls -i file_a file_b file_c
3534 file_a   3534 file_b
5800 file_c
```

- Directories may appear to have more links:

```
{saffron:maria:106} ls -ld dir
drwxr-xr-x  2 maria   users      68 Apr  7 17:57 dir/
{saffron:maria:107} mkdir dir/hello
{saffron:maria:108} ls -ld dir
drwxr-xr-x  3 maria   users      68 Apr  7 17:58 dir/
```

- This is because subdirectories (e.g. directories inside `dir/`) have a link back to their parent.

Removing a Link

- Removing or deleting a link does not necessarily remove the file (why?)
- Only when the file *and* every link is gone will the file be removed

Symbolic Links

- The links described so far are *hard links*
  - A hard link is a pointer to a file which must be on the same file system
- A symbolic link is an *indirect pointer* to a file
  - Stores the pathname of the file that it points to
  - Symbolic links can link across file systems
- Symbolic links are listed differently:

```
{saffron:ingrid:62} ln -s dir ~/unix/d/Sdir
{saffron:ingrid:62} ls -lFd dir ~/unix/d/Sdir
lrwxr-xr-x 1 ingrid  staff   3 1 Apr 21:51 /home/ingrid/unix/d/Sdir@ -> dir
drwxr-xr-x 3 ingrid  staff 102 1 Apr 21:39 dir/
```


Review: File Operations

- Create file with given pathname /a/b/file
  - Traverse pathname, allocate meta-data and directory entry
- Read from (or write to) offset in file
  - Find (or allocate) blocks of file on disk; update meta-data
- Delete
  - Remove directory entry, free disk space allocated to file
- Truncate file (set size to 0, keep other attributes)
  - Free disk space allocated to file
- Rename file
  - Change directory entry
- Copy file
  - Allocate new directory entry, find space on disk and copy
- Change access permissions
  - Change permissions in meta-data

Hard Linking Directories?

- Question for thought Should it be permitted?