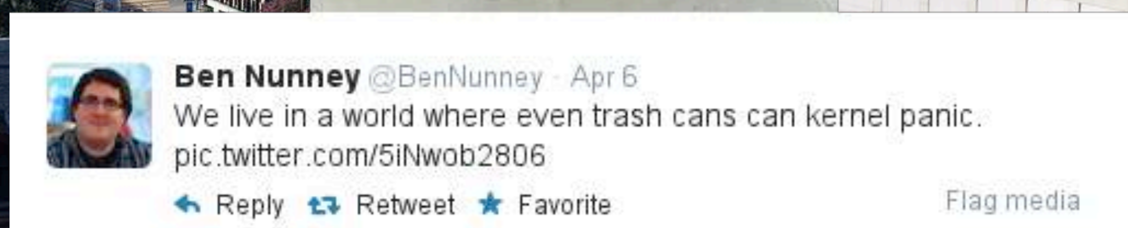
ETH zürich

ADRIAN PERRIG & TORSTEN HOEFLER

Networks and Operating Systems (252-0062-00)
Chapter 7: Filesystem Abstractions


Ben Nunney @BenNunney · Apr 6
We live in a world where even trash cans can kernel panic.
pic.twitter.com/5iNwob2806
Reply Retweet Favorite
Flag media

Paging OS back in …

- **Base + limit registers**
- **Segmentation**
- **Paging**
- **Page protection**
- **Page sharing**
- **Page table structures**
- **TLB shootdown**

- **Uses for virtual memory**
- **Copy-on-write**
- **Demand paging**
  - Page fault handling
  - Page replacement algorithms
  - …

Frame allocation policies

Thrashing

- **If a process does not have “enough” pages, the page-fault rate is very high. This leads to:**
  - low CPU utilization
  - operating system thinks that it needs to increase the degree of multiprogramming
  - another process added to the system

- **Thrashing ≡ a process is busy swapping pages in and out**

Source: wikipedia

Allocation of frames

- **Each process needs minimum number of pages**
- **Example: IBM 370 – 6 pages to handle SS MOVE instruction:**
  - instruction is 6 bytes, might span 2 pages
  - 2 pages to handle from
  - 2 pages to handle to
- **Two major allocation schemes**
  - fixed allocation
  - priority allocation

Fixed allocation

- **Equal allocation**
  - all processes get equal share
- **Proportional allocation**
  - allocate according to the size of process

$$s_i = \text{size of process } p_i$$

$$S = \sum s_i$$

$$m = \text{total number of frames}$$

$$a_i = \text{allocation for } p_i = \frac{s_i}{S} \times m$$

$$m = 64$$

$$s_1 = 10$$

$$s_2 = 127$$

$$a_1 = \frac{10}{137} \times 64 \approx 5$$

$$a_2 = \frac{127}{137} \times 64 \approx 59$$

Global vs. local allocation

- **Global replacement – process selects a replacement frame from the set of all frames; one process can take a frame from another**

- **Local replacement – each process selects from only its own set of allocated frames**

Priority allocation

- **Proportional allocation scheme**
- **Using priorities rather than size**

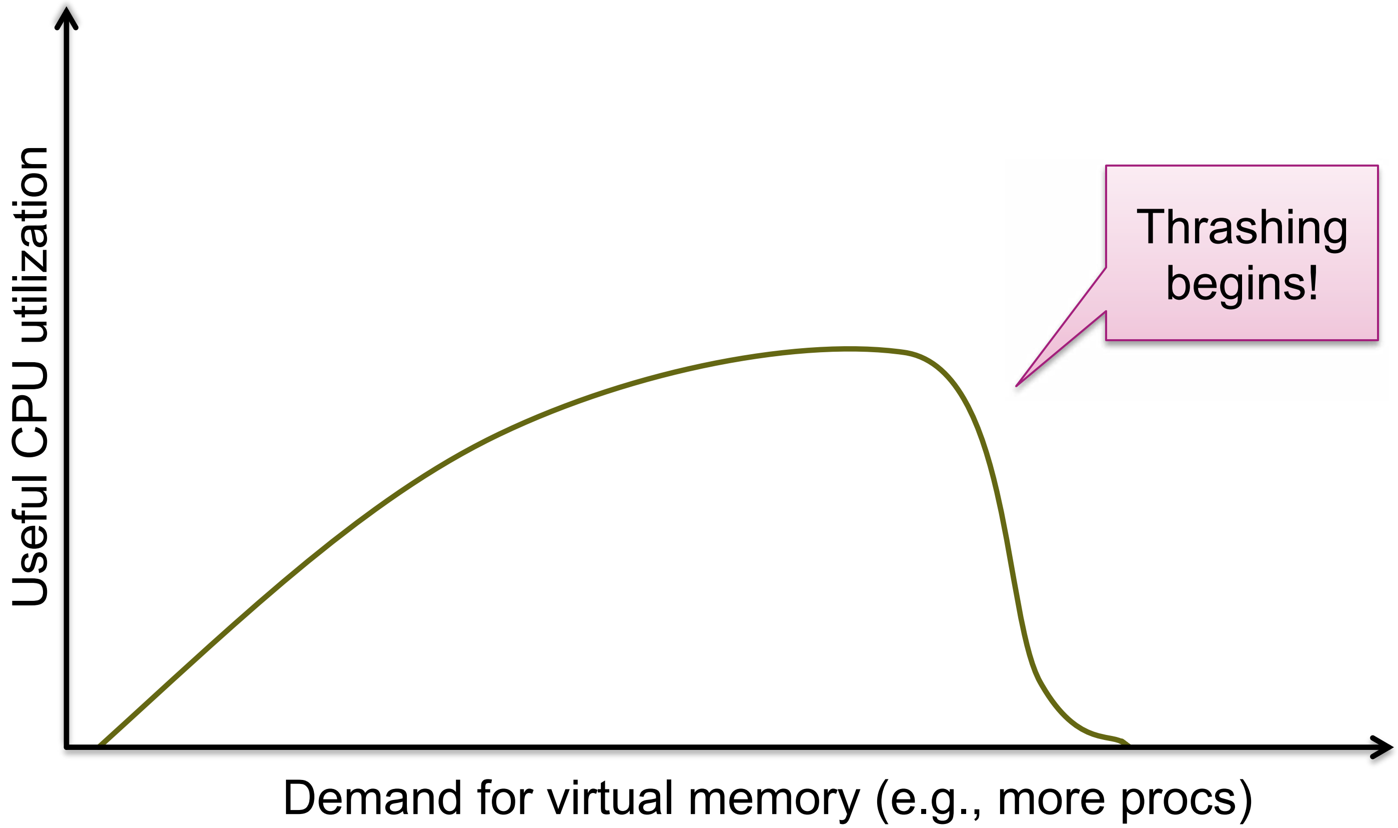
- **If process P_i generates a page fault, replace:**
  1. one of its frames, or
  2. frame from a process with lower priority

Thrashing

- **If a process does not have "enough" pages, the page-fault rate is very high. This leads to:**
  - low CPU utilization
  - operating system thinks that it needs to increase the degree of multiprogramming
  - another process added to the system

- **Thrashing ≡ a process is busy swapping pages in and out**

Thrashing

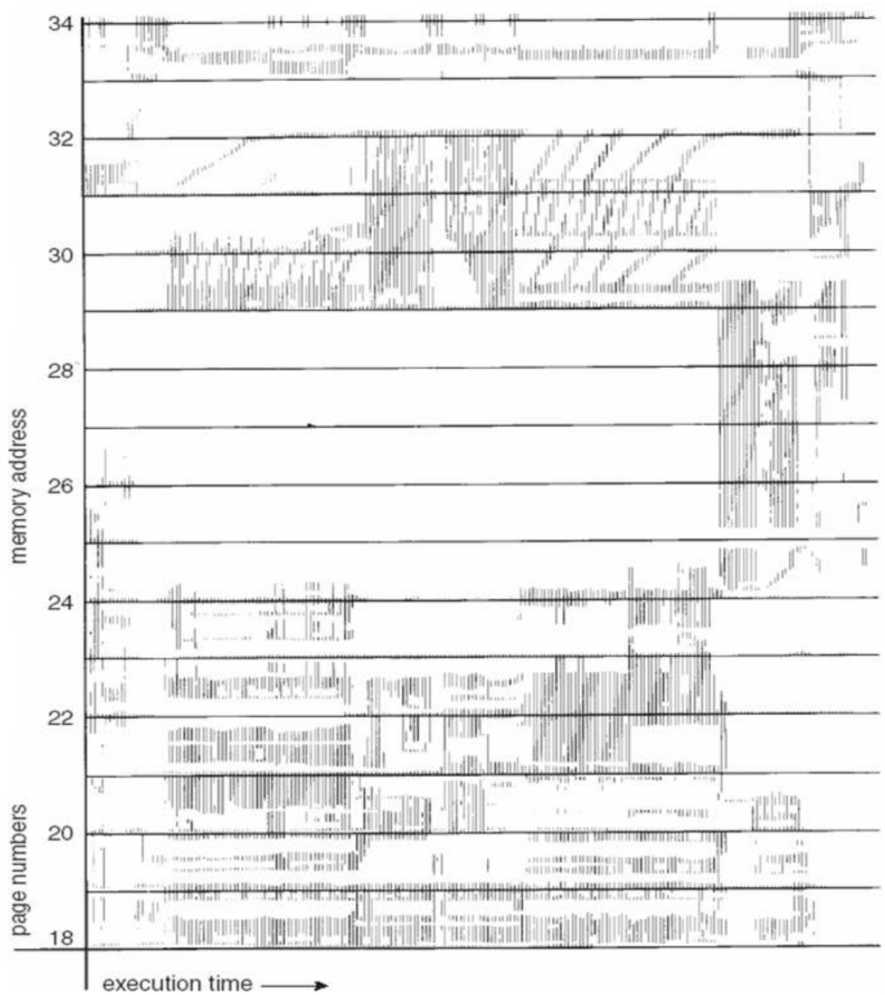

Useful CPU utilization
Thrashing begins!
Demand for virtual memory (e.g., more procs)

Demand paging and thrashing

- **Why does demand paging work?**
  Locality model
  - Process migrates from one locality to another
  - Localities may overlap

- **Why does thrashing occur?**
  Σ size of localities > total memory size

Locality in a memory reference pattern



Working-set model

- **Δ ≡ working-set window**
  ≡ a fixed number of page references
  - Example: 10,000 instructions

- **WSS_i (working set of process P_i) = total number of different pages referenced in the most recent Δ (varies in time)**
  - Δ too small $\Rightarrow$ will not encompass entire locality
  - Δ too large $\Rightarrow$ will encompass several localities
  - $\Delta = \infty \Rightarrow$ will encompass entire program

Allocate *demand frames*

- **D = Σ WSS_i ≡ total demand frames**
  - Intuition: how much space is really needed

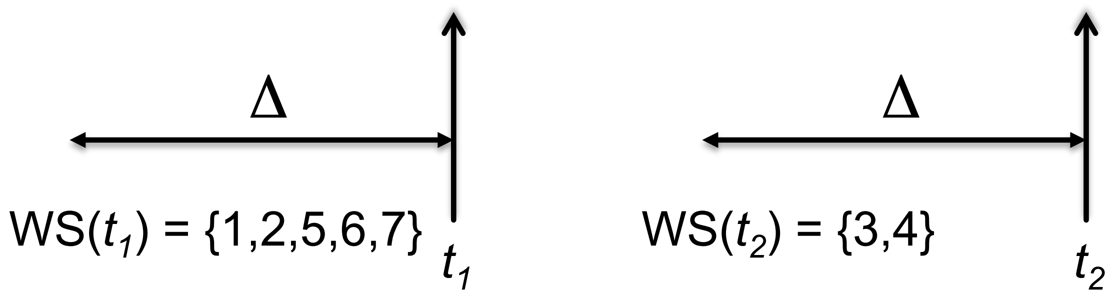
- **D > m ⇒ Thrashing**

- **Policy: if D > m, suspend some processes**

Working-set model

Page reference string:

. . . 2 6 1 5 7 7 7 7 5 1 6 2 3 4 1 2 3 4 4 4 3 4 3 4 4 4 1 3 2 3 4 4 4 3 4 4 4 . . .



Keeping track of the working set

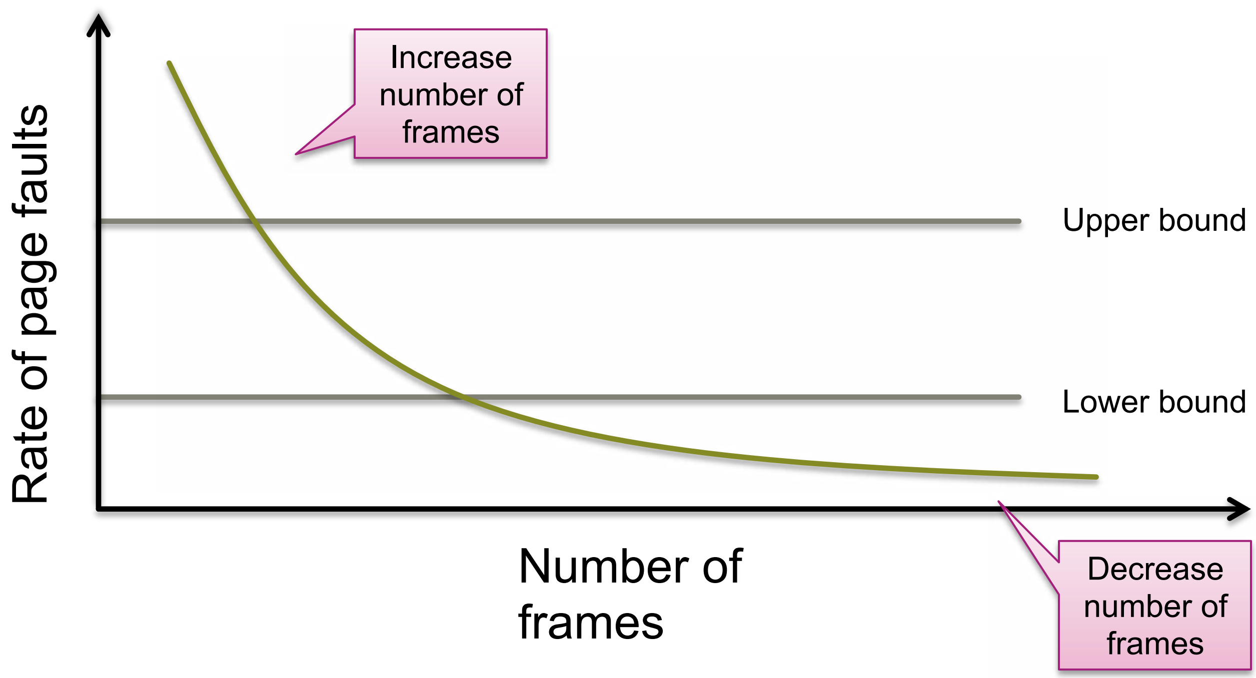
- **Approximate with interval timer + a reference bit**
- **Example: Δ = 10,000**
  - Timer interrupts after every 5000 time units
  - Keep in memory 2 bits for each page
  - Whenever a timer interrupts shift+copy and sets the values of all reference bits to 0
  - If one of the bits in memory = 1 $\Rightarrow$ page in working set
- **Why is this not completely accurate?**
  - Hint: Nyquist-Shannon!

Keeping track of the working set

- **Approximate with interval timer + a reference bit**
- **Example: Δ = 10,000**
  - Timer interrupts after every 5000 time units
  - Keep in memory 2 bits for each page
  - Whenever a timer interrupts shift+copy and sets the values of all reference bits to 0
  - If one of the bits in memory = 1 $\Rightarrow$ page in working set
- **Why is this not completely accurate?**
  - Cannot tell (within 5000 units) where the reference occurred
- **Improvement = 10 bits and interrupt every 1000 time units**

Page-fault frequency scheme

- **Establish "acceptable" page-fault rate**
  - If actual rate too low, process loses frame
  - If actual rate too high, process gains frame



Our Small Quiz

- **True or false (raise hand)**
  - Copy-on-write can be used to communicate between processes
  - Copy-on-write leads to faster process creation (with fork)
  - Copy-on-write saves memory
  - Paging can be seen as a cache for memory on disk
  - Paging supports an address space larger than main memory
  - It’s always optimal to replace the least recently used (LRU) page
  - The “second chance” (clock) algorithm approximates LRU
  - Thrashing can bring the system to a complete halt
  - Thrashing occurs only when a single process allocates too much memory
  - The working set model allows to select processes to suspend
  - Paging requires no memory management unit
  - Page-faults are handled by the disk
  - A priority allocation scheme for memory frames may suffer from priority inversion

Filesystem Abstractions

What is the filing system?

- **Virtualizes the disk**
- **Between disk (blocks) and programmer abstractions (files)**
- **Combination of multiplexing and emulation**

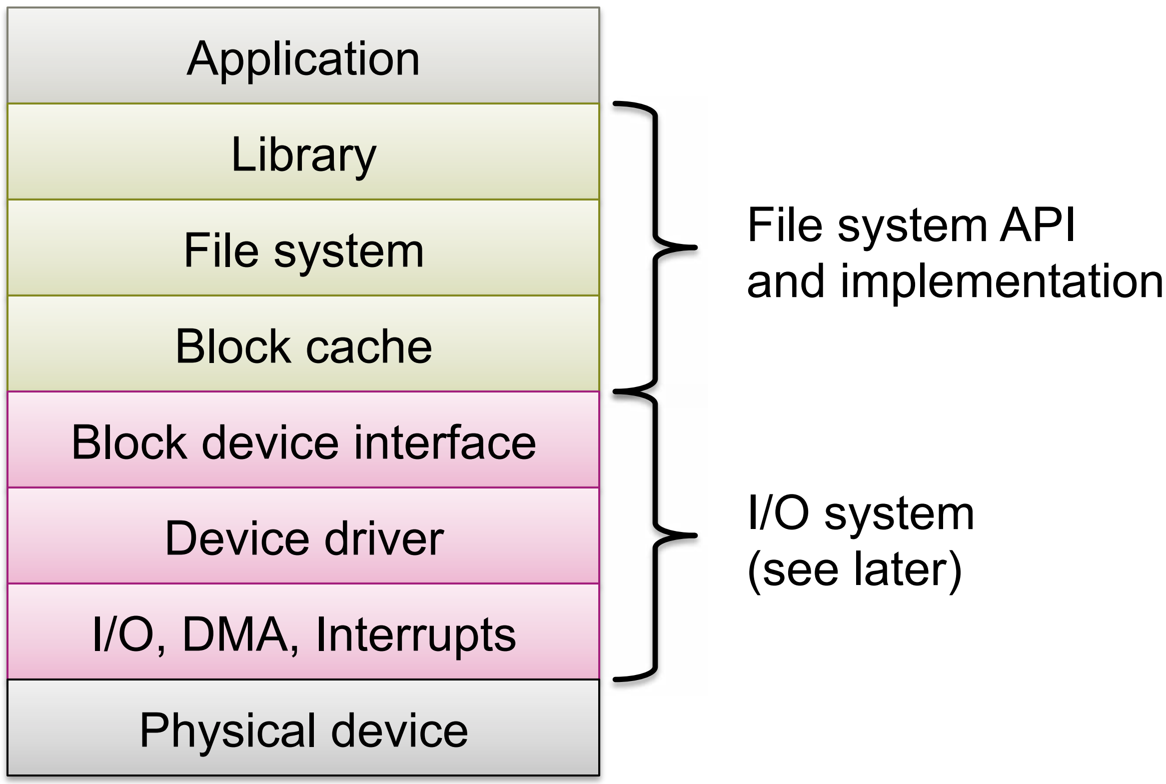
- **Generally part of the core OS**
- **Other utilities come extra:**
  - Mostly administrative

- **Book: OSPP Sections 11+13**

What does the file system need to provide?

Goal	Physical characteristic	Design implication
High performance	High cost of I/O access	Organize placement: access data in large, sequential units Use caching to reduce I/O
Named data	Large capacity, persistent across crashes, shared between programs	Support files and directories with meaningful names
Controlled sharing	Device stores many users' data	Include access control metadata with files
Reliable storage	Crashes occur during update	Transactions to make set of updates atomic
	Storage devices fail	Redundancy to detect and correct failures
	Flash memory wears out	Wear-levelling to prolong life

What the file system builds on



Filing System Interface

What is a file, to the filing system?

- **Some data**
- **A size (how many bytes or records)**
- **One or more names for the file**
- **Other metadata and attributes**
- **The type of the file**
- **Some structure (how the data is organized)**
- **Where on (disk) etc. the data is stored**
  - Next week's topic

File metadata

- **Metadata: important concept!**
  - Data *about* an object, not the object *itself*
- **File metadata examples:**
  - Name
  - Location on disk (next lecture)
  - Times of creation, last change, last access
  - Ownership, access control rights (perhaps)
  - File type, file structure (later)
  - Arbitrary descriptive data (used for searching)

Naming

Background

- **Good place to introduce Naming in general**

- **Naming in computer systems is:**
  - Complex
  - Fundamental

- **Computer systems are composed of many, many layers of different name systems.**
  - E.g., virtual memory, file systems, Internet, …

Basics: We need to name objects

```
Socket clientSocket = new Socket("hostname", 6789);
```

Give it a name

Create a new object

Naming provides *indirection*

```
DataOutputStream outToServer = new
  DataOutputStream(clientSocket.getOutputStream());
```

Could be any socket we have now

Indirection

- **Well-known quote by David Wheeler:**

"All problems in computer science can be solved by another level of indirection"

- **Might be less elegantly paraphrased as:**

"Any problem in computer science can be recast as a sufficiently complex naming problem"

Binding

- **The association between a name and a value is called a *binding*.**

- **In most cases, the binding isn't immediately visible**
  - Most people miss it, or don't know it exists
  - Often conflated with creating the value itself

- **Sometimes bindings are explicit, and are objects themselves.**

A General Naming Model

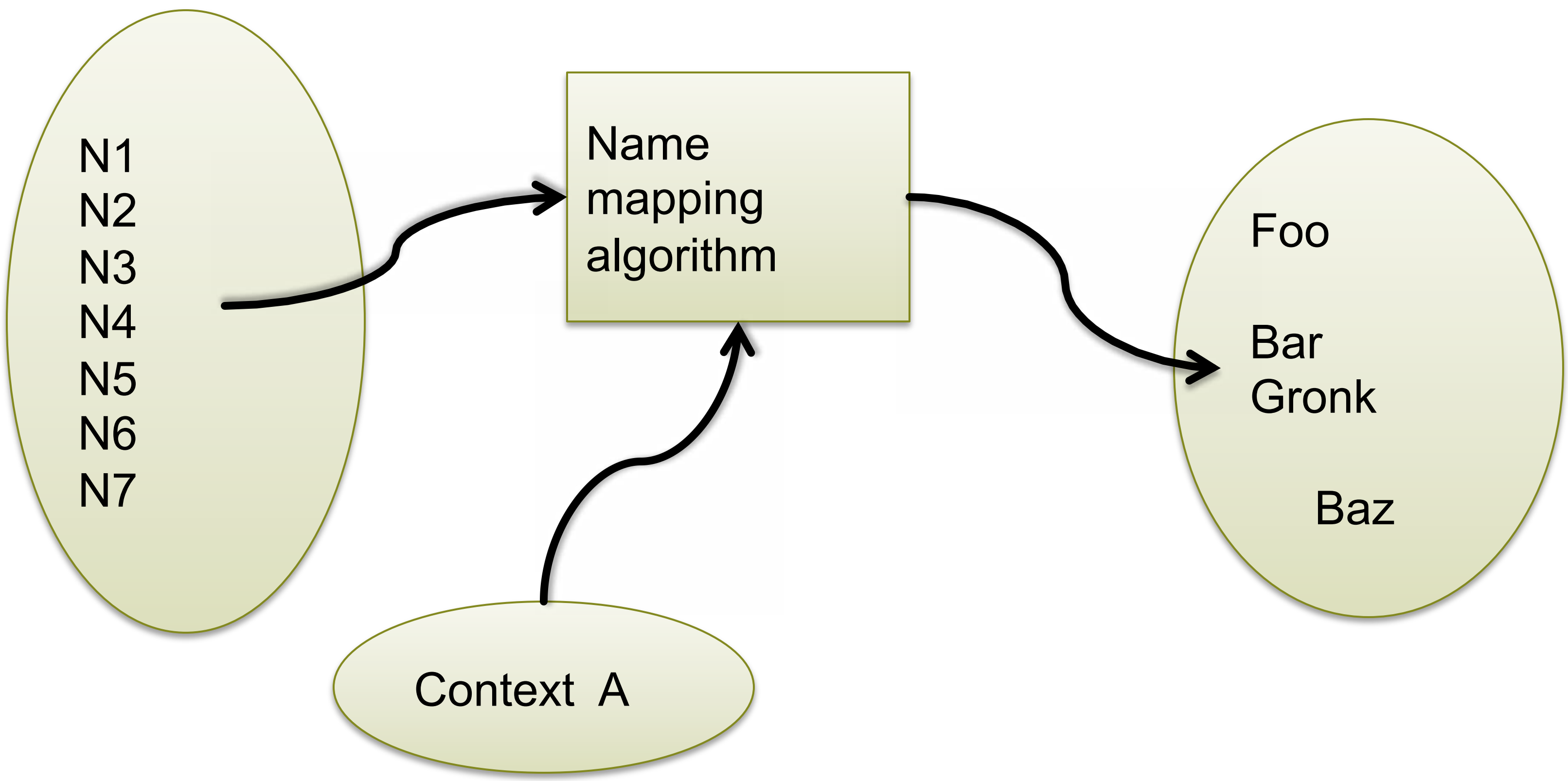
A general model of naming

- **Designer creates a naming scheme.**
  1. Name space: what names are valid?
  2. Universe of values: what values are valid?
  3. Name mapping algorithm: what is the association of names to values?

- **Mapping algorithm also known as a resolver**

- **Requires a *context***

General model



Context

- "you", "here", "Ueli Maurer" are names that require a context to be useful
- Any naming scheme must have ≥ 1 context
- Context may not be stated: always look for it!

Example naming scheme: Virtual address space

- **Name space:**
  - Virtual memory addresses (e.g., 64-bit numbers)
- **Universe of values:**
  - Physical memory addresses (e.g., 64-bit numbers)
- **Mapping algorithm:**
  - Translation via a page table
- **Context:**
  - Page table root

Single vs. multiple contexts

- **IPv4 addresses:**
  - E.g., 129.132.102.54
  - Single (global) context: routable from anywhere
  - Well, sort of…

- **ATM virtual circuit/path identifiers**
  - E.g., 43:4435
  - Local context: only valid on a particular link/port
  - Many contexts!

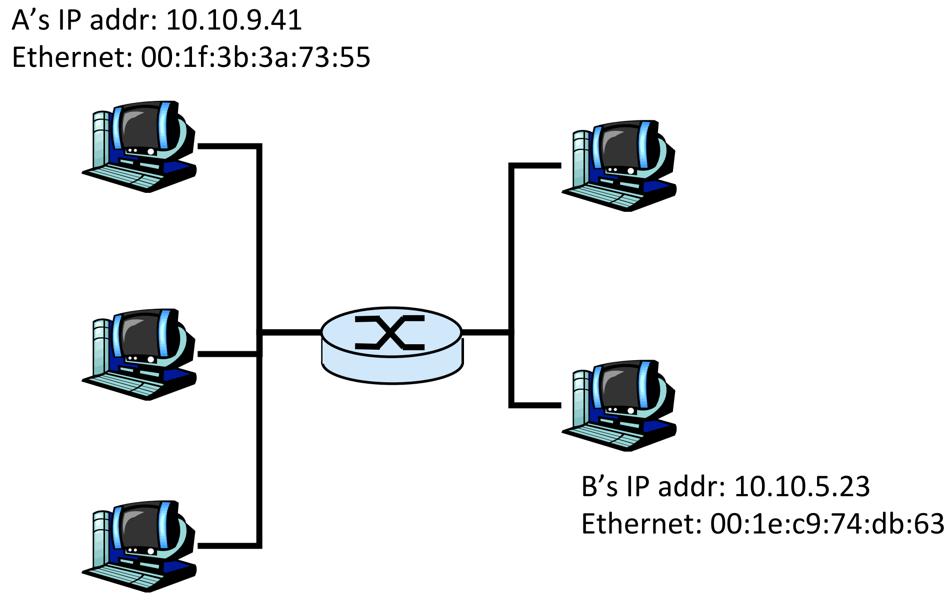
Naming operations

Resolution

- **Basic operation:**
  - *value* ← RESOLVE(*name, context*)
- **In practice, resolution mechanism depends on context:**
  - *value* ← *context.*RESOLVE(*name*)

Resolution example

- **Problem:**
  - How does A determine B's MAC address given its IP address?
- **Name space:**
  - IP addresses
- **Universe of values:**
  - Ethernet MAC addresses
- **Mapping algorithm:**
  - ARP: the Address Resolution protocol



Managing bindings

- **Typical operations:**
  - *status* ← BIND(*name, value, context*)
  - *status* ← UNBIND(*name, context*)
- **May fail according to naming scheme rules**
- **Unbind may need a value**

Example

- Unix file system (more on this later):

```
$ ln target new_link
```

- Binds "new_link" to value obtained by resolving "target" in the current context (working directory)

```
$ rm new_link
```

- Removes binding of "new_link" in cwd

- Actually called `unlink` at the system call level!

Enumeration

- **Not always available:**
  - *list* ← ENUMERATE(*context*)
- **Return all the bindings (or names) in a context**

Example enumeration

$ ls

or

C:/> dir

Comparing names

- *result* ← COMPARE(*name1, name2*)

- **But what does this mean?**
  - Are the names themselves the same?
  - Are they bound to the same object?
  - Do they refer to identical copies of one thing?
- **All these are different!**
- **Requires a definition of "equality" on objects**
- **In general, impossible…**

Examples

- **Different names, same referent:**

```
/home/htor/bio.txt
~/bio.txt
```

- **Different names, same content:**

```
htor.inf.ethz.ch://home/htor/hg/personal/websites/eth/bio.txt
free.inf.ethz.ch://home/htor/public_html/bio.txt
```

Naming policy alternatives

How many values for a name? (in a single context)

- **If 1, mapping is *injective* or "1-1"**
  - Car number plates
  - Virtual memory addresses

- **Otherwise: multiple values for a name**
  - Phone book (people have more than 1 number)
  - DNS names (can return multiple 'A' records)

How many names for a value?

- **Only one name for each value**
  - Names of models of car
  - IP protocol identifiers
- **Multiple names for the same value**
  - Phone book again (people sharing a home phone)
  - URLs (multiple links to same page)

Unique identifier spaces and stable bindings

- **At most one value bound to a name**

- **Once created, bindings can never be changed**

- **Useful: can always determine identity of two objects**
  - Social security numbers
  - Ethernet MAC addresses
    E8:92:A4::*:* → LG corporation*
    E8:92:A4:F2:0B:97 → Torsten's phone's WiFi interface

Types of lookup

Name mapping algorithms

1. **Table lookup**
   - Simplest scheme
   - Analogy: phone book

Faculty

name	phone	email	address
Alonso, Gustavo	+41 44 632 7306	alonso@inf.ethz.ch	CAB F 77 Universitätstrasse 6 CH-8092 Zürich
Kossmann, Donald	+41 44 632 2940	donaldk@inf.ethz.ch	CAB F 73 Universitätstrasse 6 CH-8092 Zürich
Roscoe, Timothy	+41 44 632 8840	timothy.roscoe@inf.ethz.ch	CAB F 79 Universitätstrasse 6 CH-8092 Zürich
Tatbul, Nesime	+41 44 632 8920	tatbul@inf.ethz.ch	CAB F 75 Universitätstrasse 6 CH-8092 Zürich

2. **Recursive lookup (pathnames)**
3. **Multiple lookup (search paths)**

Table lookup: other examples

- Processor registers are named by small integers.
- Memory cells are named by numbers.
- Ethernet interfaces are named by MAC addresses
- Unix accounts are named by small (16bit) numbers (userids)
- Unix userids are named by short strings
- Unix sockets are named by small integers

Default and explicit contexts, qualified names

Where is the context?

1. **Default (implicit): supplied by the resolver**
   1. Constant: built in to the resolver
   2. Variable: from current environment (state)

2. **Explicit: supplied by the object**
   1. Per object
   2. Per name (qualified name)

Constant default context

- **Universal name space:**
  e.g. DNS
- **Short answer:**
  - context is the DNS root server
- **Longer answer:**
  - /etc/hosts, plus DNS root server
- **Even longer answer:**
  - /etc/nsswitch.conf, WINS resolver, domain search path, … ☹

spcl.inf.ethz.ch

Variable default context

- **Example: current working directory**

```
$ pwd
/home/htor/svn
$ ls
osnet/
$ cd osnet
$ ls
archive/       lecture/ organisation/         svnadmin/
assignments/   legis/   recitation sessions/   svn-commit.tmp
$ ls lecture
chapter1/    chapter2/   chapter5/   chapter8/   template.pptx
chapter10/   chapter3/   chapter6/   chapter9/
chapter11/   chapter4/   chapter7/   dates.xls
$
```

Explicit per-object context

- **Note: context reference is a name!**
  - Sometimes called a base name
- **Examples:**

```
$ ssh –l htor spcl.inf.ethz.ch
$ dig @8.8.8.8 -q a spcl.inf.ethz.ch
$ dig @google-public-dns-a.google.com -q a spcl
```

Explicit per-name context

- **Each name comes with its context**
  - Actually, the *name* of the context
  - (context,name) = qualified name

- **Recursive resolution process:**
  - Resolve *context* to a context object
  - Resolve *name* relative to resulting context

- **Examples:**
  - htor@inf.ethz.ch
  - /var/log/syslog

Path names, naming networks, recursive resolution

Path names

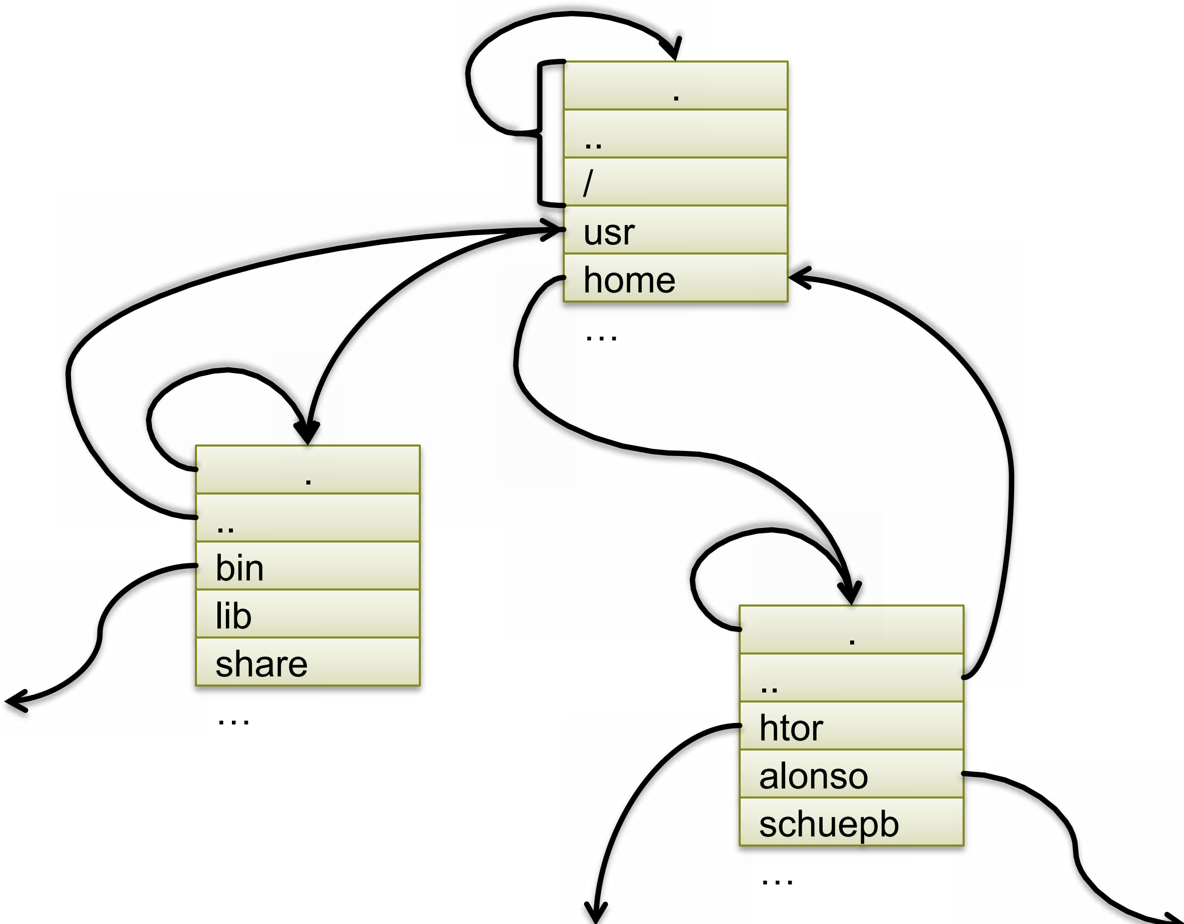
- **Recursive resolution ⇒ path names**

- **Name can be written forwards or backwards**
  – Examples: /var/log/messages or spcl.inf.ethz.ch

- **Recursion must terminate:**
  – Either at a fixed, known context reference
    - *(the root)*
  – Or at another name, naming a default context
    - *Example: relative pathnames*

- **Syntax gives clue (leading '/')**
  - Or trailing "." as in spcl.inf.ethz.ch.

Naming networks



"Soft links"

- **So far, names resolve to values**
  - Values may be names in a different naming scheme (usually are…)
- **Names can resolve to other names in the *same scheme:***
  - Unix symbolic links (`ln -s`), Windows "short cuts"
  - Forwarding addresses (Die Post vs. USPS, WWW, Email)

Multiple lookup

Sometimes, one context is not enough…

- **Multiple lookup, or "search path"**
  - try several contexts in order
- **Union mounts: overlay two or more contexts**
- **Examples:**
  - binary directories in Unix
  - resolving symbols in link libraries
- **Somewhat controversial…**

- **Note: "search", but not in the Google sense…**

"Search path" example

```
$ echo $PATH
/home/htor/bin:/local/bin:/usr/local/bin:/usr/bin:
/bin:/sbin:/usr/sbin:/etc:/usr/bin/X11:/etc/local:
/usr/local/sbin:/home/netos/tools/bin:/usr/bin:
/home/netos/tools/i686-pc-linux-gnu/bin
$ which bash
/bin/bash
$
```

Name Discovery

How to find a name in the first place?

- **Many options:**
  - Well-known.
  - Broadcast the name.
  - Query (google/bing search)
  - Broadcast the query.
  - Resolve some other name to a name space
  - Introduction
  - Physical rendezvous
- **Often reduces to another name lookup…**

Bad names

“The Hideous Name”, Rob Pike and P.J. Weinberger, AT&T Bell Labs

`research!ucbvax!@cmu-cs-pt.arpa:@CMU-ITC-LINUS:dave%CMU-ITC-LINUS@CMU-CS-PT`

(Attributed to the Carnegie-Mellon mailer)

Warning

- **Don't look too closely at names**
- **Almost *everything* can be viewed as naming**
  - This does not mean it *should* be.

"All problems in computer science can be solved by another level of indirection…"

"...except for the problem of too many layers of indirection."

- **A naming model is a good servant, but a poor master.**

Conclusion

- **Naming is everywhere in Computer Systems**
  - Name spaces
  - Contexts
  - Resolution mechanisms
- **When understanding a system, ask:**
  - What are the naming schemes?
  - What's the context?
  - What's the policy?
- **When designing a system, it *will* help stop you making (some) silly mistakes!**

File system operations

We've already seen the file system as a naming scheme.

Directory (name space) operations:

- **Link (bind a name)**
- **Unlink (unbind a name)**
- **Rename**
- **List entries**

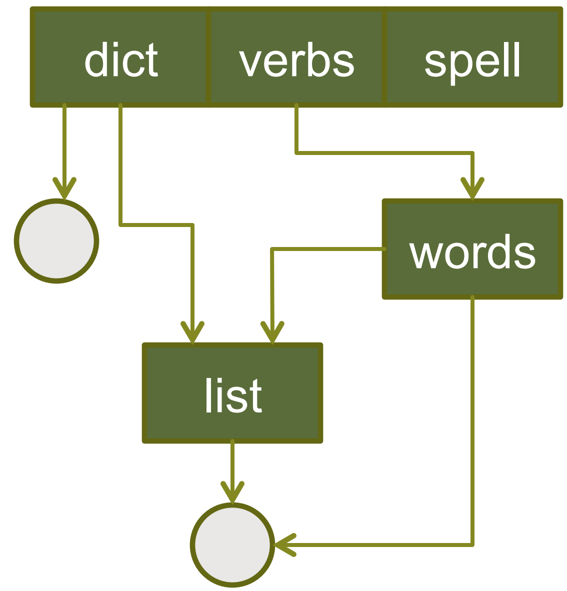
Acyclic-Graph Directories

- **Two different names (aliasing)**

- **If *dict* deletes *list* ⇒ dangling pointer**

  Solutions:
  - Backpointers, so we can delete all pointers
    Variable size records can be a problem
  - Backpointers using a daisy chain organization
  - Entry-hold-count solution

- **New directory entry type**
  - **Link** – another name (pointer) to an existing file
  - **Resolve the link** – follow pointer to locate the file

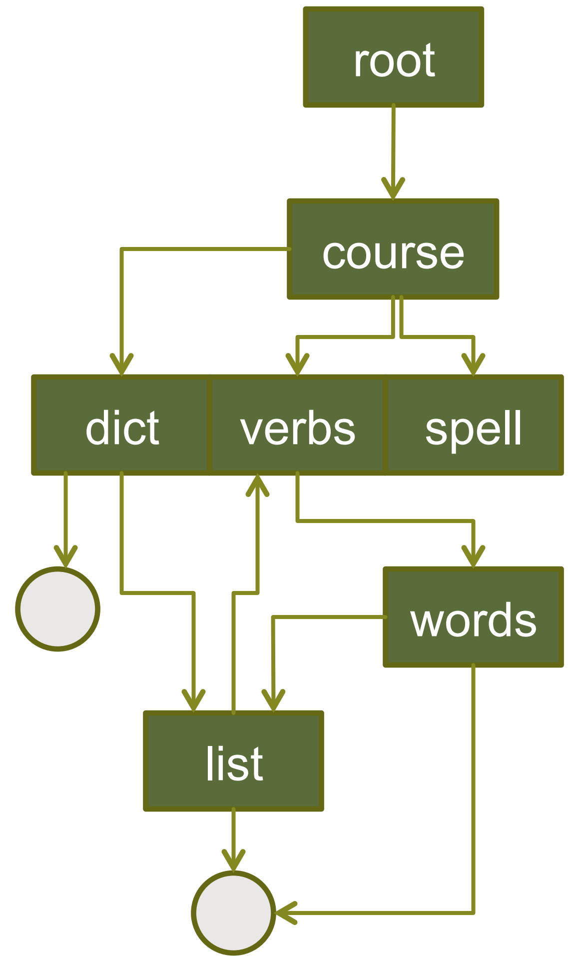


General Graph Directory

- **How do we guarantee no cycles? Options:**
  - Allow only links to files and not directories
  - Garbage collection (with cycle collector)
  - Check for cycles when every new link is added
  - Restrict directory links to parents

    E.g., "." and ".."

    All cycles are therefore trivial



Access Control

Protection

- **File owner/creator should be able to control:**
  - what can be done
  - by whom

- **Types of access**
  - **Read**
  - **Write**
  - **Execute**
  - **Append**
  - **Delete**
  - **List**

Access control matrix

For a single file or directory:

Principals

Rights

	A	B	C	D	E	F	G	H	J	…
Read	☑	☑	☑			☑	☑			
Write	☑	☑		☑			☑			
Append	☑				☑					
Execute	☑	☑	☑	☑						
Delete	☑									
List	☑				☑					
…										

Problem: how to scalably represent this matrix?

Row-wise: ACLs

- **Access Control Lists**
  - For each right, list the principals
  - Store with the file
- **Good:**
  - Easy to change rights quickly
  - Scales to large numbers of files
- **Bad:**
  - Doesn’t scale to large numbers of principals

Column-wise: Capabilities

- **Each principal with a right on a file holds a *capability* for that right**
  - Stored with principal, not object (file)
  - Cannot be forged or (sometimes) copied
- **Good:**
  - Very flexible, highly scalable in principals
  - Access control resources charged to principal
- **Bad:**
  - Revocation: hard to change access rights
    (need to keep track of who has what capabilities)

POSIX (Unix) Access Control

- **Simplifies ACLs: each file identifies 3 principals:**
  - Owner (a single user)
  - Group (a collection of users, defined elsewhere)
  - The World (everyone)
- **For each principal, file defines 3 rights:**
  - Read (or traverse, if a directory)
  - Write (or create a file, if a directory)
  - Execute (or list, if a directory)

Example

```
drwx--x--x  9 htor htor     4096 May  9 13:14 pagai
htor@lenny ~ > ls -l projekte/llvm/llvm-svn            < 09.05.13 19:08:49 >
total 860
drwx--x--x  3 htor htor   4096 Jan 29 15:58 autoconf
drwx--x--x  4 htor htor   4096 Dec 25 13:20 bindings
drwx--x--x  4 htor htor   4096 Jan 29 15:57 cmake
-rw-------  1 htor htor  16401 Dec 25 13:20 CMakeLists.txt
-rw-------  1 htor htor   2782 Jan 29 15:57 CODE_OWNERS.TXT
-rwx------  1 htor htor 658352 Jan 29 15:57 configure
-rw-------  1 htor htor  10048 Dec 25 13:20 CREDITS.TXT
drwxr-xr-x 11 htor htor   4096 Apr  4 11:13 Debug
drwx--x--x 10 htor htor   4096 Jan 29 15:57 docs
drwx--x--x 10 htor htor   4096 Dec 25 13:20 examples
drwx--x--x  4 htor htor   4096 Dec 25 13:20 include
drwx--x--x 18 htor htor   4096 Jan 29 15:58 lib
-rw-------  1 htor htor   3254 Jan 29 15:57 LICENSE.TXT
-rw-------  1 htor htor    752 Dec 25 13:20 LLVMBuild.txt
-rw-------  1 htor htor   1865 Dec 25 13:20 llvm.spec.in
-rw-------  1 htor htor   8618 Jan 29 15:58 Makefile
-rw-------  1 htor htor   2599 Dec 25 13:20 Makefile.common
-rw-------  1 htor htor  12068 Jan 29 15:57 Makefile.config.in
-rw-------  1 htor htor  79586 Jan 29 15:57 Makefile.rules
drwx--x--x  4 htor htor   4096 Dec 25 13:21 projects
-rw-------  1 htor htor    687 Jan 29 15:58 README.txt
drwx--x--x  3 htor htor   4096 Dec 25 13:20 runtime
drwx--x--x 27 htor htor   4096 Jan 29 15:57 test
drwx--x--x 35 htor htor   4096 Dec 25 13:21 tools
drwx--x--x 11 htor htor   4096 Jan 29 15:57 unittests
drwx--x--x 32 htor htor   4096 Jan 29 15:57 utils
```

Full ACLs

- **POSIX now supports full ACLs**
  - Rarely used, interestingly
  - setfacl, getfacl, …
- **Windows has very powerful ACL support**
  - Arbitrary groups as principals
  - Modification rights
  - Delegation rights

Concurrency

Concurrency

1. **Must ensure that, regardless of concurrent access, file system *integrity* is ensured**
   - Careful design of file system structures
   - Internal locking in the file system
   - Ordering of writes to disk to provide transactions

2. **Provide mechanisms for users to avoid conflicts themselves**
   - Advisory locks
   - Mandatory locks

Common locking facilities

- **Type:**
  - Advisory: separate locking facility
  - Mandatory: write/read operations will fail

- **Granularity:**
  - Whole-file
  - Byte ranges (or record ranges)
  - Write-protecting executing binaries

Compare with databases

- **Databases have a *way* better notions of:**
  - Locking between concurrent users
  - Durability in the event of crashes
- **Records and indexed files have largely disappeared in favor of databases**
- **File systems remain much easier to use**
  - And much, much faster
  - As long as it doesn't matter…