Introduction to Naming

COS 316: Principles of Computer System Design

Amit Levy & Ravi Netravali

It has been said that the principal function of an operating system is to define a number of different names for the same object, so that it can busy itself keeping track of the relationship between all of the different names.

-David Clark, 1982 RFC 814

Application: "I would like to send data to the Internet host, please."
System: "Which host?"
Application:"Oh... uh... `cos316.princeton.edu`"

Application: "I would like to send data to the Internet host, please."
System: "Which host?"
Application:"Oh… uh… `cos316.princeton.edu`"

`cos316.princeton.edu` is the *name* for an IP address!

Application: "Can I please get the data?"
System: "You're gonna have to be more specific."
Application:"The data in '/home/alevy/world-domination-todo.txt"

Application: "Can I please get the data?"

System: "You're gonna have to be more specific."

Application:"The data in '/home/alevy/world-domination-todo.txt"

`/home/alevy/world-domination.txt` is the *name* for a bunch of sectors on disk!

Application: "What is the sum of two numbers?"
System: "I really need to know which numbers..."
Application: "Fine, fine, fine: the ones in registers `r1` and `r2`."

Application: "What is the sum of two numbers?"
System: "I really need to know which numbers..."
Application: "Fine, fine, fine: the ones in registers `r1` and `r2`."

`r1` and `r2` are *names* for words of memory residing in CPU registers!

Whenever an application uses a resource, it must somehow *name* it.

Why does it matter?

Principle: minimize abstraction at each layer

An intellectual framework for naming

Next, naming in...

- UNIX File System
- Git
- Networking

Why does naming matter?

Naming is the most central design choice in systems

- Me, just now

Why does naming matter?

Recall: Systems provide an interface to underlying resources

- Mediate access to shared resources
- Isolate applications
- Abstract complexity
- Abstract differences in implementation

We always need *some* way for applications (or other clients) to *name* those resources.

Why does naming matter?

The names systems use to expose underlying resources affects every other aspect of the system:

- Performance of system implementation
- Application performance and flexibility
- Security
- Effectiveness of caching
- Resource sharing and concurrency

Performance Example: Naming Memory

Applications use Physical Addresses

- Does not require any translation hardware.
- Many small microcontrollers use this to save die space, energy, cost, cycles

Performance Example: Naming Memory

Applications use Physical Addresses

- Does not require any translation hardware.
- Many small microcontrollers use this to save die space, energy, cost, cycles

Applications use Virtual Addresses

- Gives the system more flexibility in allocating memory
  - Can deduplicate, colocate, overprovision, swap, etc
  - Ultimately makes applications faster and simpler
- Requires additional hardware and more complex memory management

Security Example: Naming Files

Filesystem Namespaces (chroot, jails, ...)

- Impossible for an application to express reading a file outside its sandbox
- No dynamic checks required from system on file access

Security Example: Naming Files

Filesystem Namespaces (chroot, jails, ...)

- Impossible for an application to express reading a file outside its sandbox
- No dynamic checks required from system on file access

Explicit Access Control for Global Namespace

- Allows much more flexible security policies
- Might be easier to audit, since policy is specified explicitly
  - But also might be harder if harder to divide policy into independent components
- Each file access must be guarded to ensure security
  - Lots of places there could be bugs
  - Might degrade performance

Other examples

Virtualization with subdivisible names

Subdivisible names let virtualization layers subdivide resources easily:

"You get disk sectors 1-100, while you get 101-200, ..."

Conversely, can't subdivide "any input from the keyboard," so system is responsible for implementing some multiplexing policy.

E.g., window managers forward all key strokes to the focused application.

Caching granular and coarse names

Naming big resources allows caches to easily "prefetch" related data, but is ineffective when data isn't specially local.

For example, huge memory pages vs. regular memory pages on x86.

Why are names important?

- The primary way applications interact with a system
- Often the first step in designing a system
  - A central design decision
  - Can make implementation easier or obvious
- A good place to start understanding systems
- Choice of naming scheme can dictate system trade-offs

Principle

> *Names in a system should minimally abstract underlying resources to achieve goal (portability, security, sharing, etc).*

Provide higher-level abstraction with more "layers".

- UNIX file system
- Git internals
- Network naming

Naming Scheme Framework

- **Values**: What is it that we're naming?
  - Disk sectors?
  - Network nodes?
  - Users?
- **Names**: What's the format of a name?
  - Alphanumeric strings up to 32 characters
  - Non-zero integers
  - 128-bit numbers
- **Allocation mechanism**: How does the system create new names and values?
- **Lookup mechanism**: How does the system map from names to values?

Naming and system trade-offs

Let's compare alternate naming systems from the physical world:

Building numbers in a city block

Naming and system trade-offs

Let's compare alternate naming systems from the physical world:

Building numbers in a city block

San Francisco -vs- Tokyo

Figure 1: How do we name each of these buildings?

San Francisco

How do buildings get their name?

San Francisco

How do buildings get their name?

Subdivide the block into N plots numbered `1-N`

Name buildings with the number of their plot

How do buildings get their name?

Subdivide the block into N plots numbered `1-N`

Name buildings with the number of their plot

But… what happens when we build the `N+1`th building? Or if we want to divide a plot in half?

San Francisco

How do buildings get their name?

Subdivide the block into N plots numbered `1-N`

Name buildings with the number of their plot

But… what happens when we build the `N+1th` building? Or if we want to divide a plot in half?

E.g., I used to live at `3477 1/2 17th St.`

Tokyo

1987

1993

2012

1964

How do buildings get their name?

Tokyo

How do buildings get their name?

Buildings numbered from oldest to newest.

Whenever a new building is built, give it the next highest number on the block

Tokyo

How do buildings get their name?

Buildings numbered from oldest to newest.

Whenever a new building is built, give it the next highest number on the block

Finding a building is hard! Need to check buildings until you find the right number.

Tokyo

How do buildings get their name?

Buildings numbered from oldest to newest.

Whenever a new building is built, give it the next highest number on the block

Finding a building is hard! Need to check buildings until you find the right number.

But... virtually infinite possibility for expansion!

Different naming and trade-offs

	San Francisco Addressing	**Tokyo Addressing**
Easy allocation	no :(	YES
Easy lookup	YES	no :(

Applying Framework to Building Numbers

	San Francisco Addressing	**Tokyo Addressing**
Values	Physical buildings	Physical buildings
Names	Numbers 1-N	Numbers >=1
Allocation	Pre-defined plot numbers	New building gets next highest number
Lookup	Search ordered set	Visit every building and check

Virtual Memory

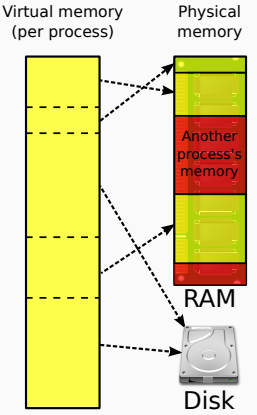

Figure 2: Virtual Memory

What does virtual memory give us?

- Isolation: `0xdeadbeef` maps to different values for different applications
- Abstraction over storage media: virtual addresses can name memory, storage, remote memory, HW register, ...
- Flexibility in provisioning
  - Resource efficiency: physical memory can be less sparse than virtual memory
  - Overcommit physical memory (e.g. swap to disk when necessary)

Virtual Memory as a Naming Scheme

- Values?
- Names?
- Allocation?
- Lookup?

Virtual Memory: Values

A pair, including the *type* of storage (memory, file, hardware register, etc), and:

- *Physical memory address (i.e. 32-bit or 64-bit address up to size of RAM)*

Virtual Memory: Values

A pair, including the *type* of storage (memory, file, hardware register, etc), and:

- *Physical memory address (i.e. 32-bit or 64-bit address up to size of RAM)*
- On-disk file and offset in file

Virtual Memory: Values

A pair, including the *type* of storage (memory, file, hardware register, etc), and:

- *Physical memory address (i.e. 32-bit or 64-bit address up to size of RAM)*
- On-disk file and offset in file
- Some hardware registers (e.g. a network card configuration registers)

Virtual Memory: Values

A pair, including the *type* of storage (memory, file, hardware register, etc), and:

- *Physical memory address (i.e. 32-bit or 64-bit address up to size of RAM)*
- On-disk file and offset in file
- Some hardware registers (e.g. a network card configuration registers)
- Remote memory

Virtual Memory: Names

Pointer-sized (e.g. 32-bit or 64-bit) addresses and process identifiers

$(PID, virtual\_address)$

e.g.

$(3487, 0xdeadbeef)$

Virtual Memory: Names

Pointer-sized (e.g. 32-bit or 64-bit) addresses and process identifiers

$(PID, virtual\_address)$

e.g.

$(3487, 0xdeadbeef)$

Note: the processes identifier is generally implicitly whichever process performed the memory operation

Virtual Memory: Allocation

`void *mmap(void *addr, size_t length)` (simplified)

- Application chooses an unused name: an address not yet allocated for it
- Kernel keeps a list of unused physical 4KB memory pages
- Kernel allocates "value" by removing a physical page from the list
- Kernel adds new mapping between virtual address and physical to the application's "page table"
  - in-memory data structure understood by the virtual memory hardware that maps virtual addresses to physical addresses

Virtual Memory: Lookup

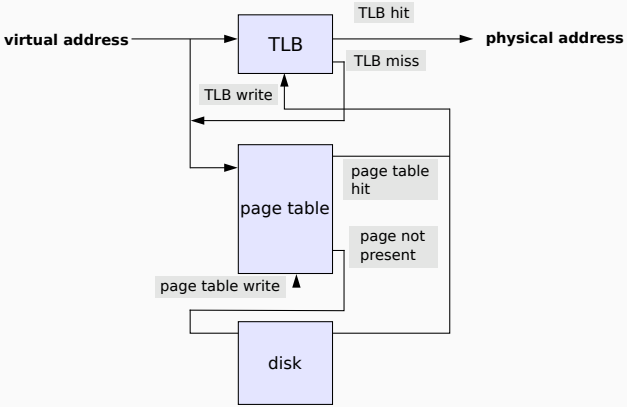

Figure 3: Virtual-to-physical translation

Virtual Memory: Lookup

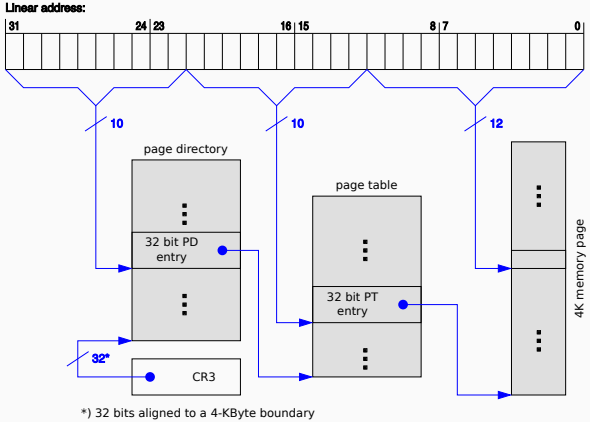

Figure 4: Two-level page table structure in x86

Virtual Memory: Lookup

For this to work, the OS needs to do some housework when context switching:

- Set CR3 register to point to process's page table
- Invalidate the TLB
  - Mark entire TLB as invalid—simple but can cause unnecessary slow down
  - Associate process IDs with each TLB entry

Virtual Memory: Trade-offs with Alternative Naming Schemes

- Single shared address space (identity mapping)
  - Better performance, since no translation hardware/software needed
  - But can't isolate using names
- Swap out all memory for one process at a time (original UNIX)
  - Simple and efficient to implement in hardware
  - Can't run applications in parallel
  - Expensive to switch between applications
- Segmentation: virtual address are low-order bits of physical address + segment register
  - Relatively simple hardware (just concatenate segment register and virtual address)
  - Isolates concurrent applications using names
  - Much coarser grain: all virtual memory must be contiguous in RAM
  - Can't share memory between applications

Summary

- Names are the way systems expose resources to applications
- Central to designing and understanding systems
  - performance
  - security
  - caching
  - resource sharing
- Framework for naming:
  - Values
  - Names
  - Allocation mechanism
  - Lookup mechanism