6.033 Lecture 3: Naming

Systems

system = a bunch of resources, glued together with names
often the best way to view or build
often most interesting action is in the names
[cloud with components & connections]
what are the resources?
often can view them as just three basic types

* Fundamental abstractions
  storage
    mem, disk
    layered: data struct, FS, disk arrays
  interpreters
    cpu
    programming language e.g. java VM
  communication
    wire, ethernet
    layered: internet, ssh, unix pipe
we'll talk about all these abstractions during the course

Why these abstractions?
many components fit these molds
they work well together
hardware versions are cheap
for these abstractions, general techniques known
to increase performance, fault tolerance, functionality

Names
  all three abstractions rely on names
  memory stores named objects
  interpreter manipulates named objects
  communication link connects named entities
  much of sys design is naming --- glue

DNS Example
  what does web.mit.edu mean?
  you can look it up and turn it into an IP address
  18.7.22.69
  Internet knows how to send data to IP addresses
  how does the translation work?

The big picture
  my laptop O/S shipped with IP addr of a few "root name servers"
  these are hard-wired, in a file, about a dozen
  198.41.0.4 (virginia)
  128.63.2.53 (maryland)
  ...
  this list doesn't change much, otherwise old computers would break
  my laptop picks a root server, sends it a query,
  asking for IP address of the string "web.mit.edu"
  each root server has exactly the same table, the root "zone"
  edu NS 192.5.6.30
  edu NS 192.12.94.30
  com NS 192.48.79.30
finds entries in root zone that matches suffix of "web.mit.edu"
returns those records to my laptop
now my laptop knows who to ask about edu
re-sends request to one of the edu servers
each edu server has a copy of the edu zone
  nyu.edu NS 128.122.253.83
  mit.edu NS 18.72.0.3
  mit.edu NS 18.70.0.160
returns the two MIT records to my laptop
now my laptop forwards request to an MIT server
each MIT server has a copy of the mit.edu zone
  web.mit.edu A 18.7.22.69
  bitsy.mit.edu A 18.72.0.3
  csail.mit.edu NS 18.24.0.120
...  

DNS has a hierarchical name space
  diagram of a tree

Where do the zone files come from?
each zone centrally but independently administered
  root by IANA, an international non-profit org
  com by Verisign, a company
  MIT IS maintains its zone files, updates them on its servers

Why this design?
everyone sees same name space (at least for full names)
scalable in performance
  via simplicity, root servers can answer many requests
  via caching, not really described here
  via delegation (keeps table size down)
scalable in management
  MIT control own names by running its own servers
fault tolerant

There are problems:
who should control root zone? .com?
load on root servers
denial-of-service attacks
security problems: how does client know data is correct?
  how does verisign know i'm ibm.com when I ask it to change NS?

DNS has been very successful
  25 years old
  design still works after 1000x scaling
  you can learn more about DNS from Chapter 4 of the notes

How do systems use DNS?
  user-friendly naming (type a URL)
  web page tells browser what to do on button press
    i.e. connects one page to the next
  Adobe PDF Reader contains DNS name for where (later) to check for upgrades
  allows services to change IP address of servers (change hosting service)
  DNS servers can give different answers to different clients
Naming goals
let's step back
1. user friendly
2. sharing
   diagram: A and B both know N, refers to ... 
3. retrieval (sharing across time)
   A knows N, refers twice to ... over time
4. indirection
   diagram: A knows fixed N, refers to variable N', refers to ...
5. hiding
   hide implementation
   control access (access only if you know name, or hook to hand ACLs)

Naming systems are common, since so powerful
phone numbers
files / directories
e-mail addresses
function/variable names in programs

Example: virtual addressing
often called "virtual memory" or "paging"

Why this example?
   naming mechanism
   well designed
   solves many problems
   basis for many neat o/s features

Memory
this is review from 6.004
also Chapter 5 in notes
diagram: CPU, address/data wires, DRAM
DRAM an array of bytes
indexed with "physical address"
let us say 32-bit addresses
could have up to 4 GB of DRAM
example: LD R4, 0x2020

Why physical addressing not sufficient?
many reasons, as we will see
here's one: what if program too big to fit in DRAM?
   but still fits in 2^32 bytes
   e.g. two-megabyte program, only one megabyte of DRAM

Demand Paging
diagram: cpu, dram, disk
want some data to be in memory, some on disk
   when program refers to data that's not in memory,
   move something else from memory to disk,
   load in the desired memory from disk
   called "demand paging"
   that's the original motivation, not so relevant any more

Virtual Addressing Idea
diagram: MMU, two arrays of memory cells
set of arrows from left to right
"virtual addresses" vs "physical addresses"
  s/w ONLY loads/stores using virtual addresses
  phys addr ONLY show up in the mapping table
conceptually, translations for all 2\(^{32}\)
  some refer to physical memory
  others to disk locations
how to implement?
  if per-byte mappings, 16 GB!
  not practical to have a translation table with entry per byte addr

Page Tables
  how to make the mapping table small?
  divide virtual addresses into contiguous aligned "pages"
  upper (say) 20 bits are "virtual page number"
  MMU only maps page number \(\rightarrow\) upper 20 bits of a phys address
  then use lower 12 va bits as lower 12 pa bits

Example:
  page table:
    0: 3 (i.e. phys page 3)
    1: 0
    2: 4
    3: 5
  register R3 contains 0x2020
  LD R4, (R3)
  CPU sends (R3)=0x2020 to MMU
  virtual page number is 2
  offset is 0x20
  thus pa = 4*4096 + 0x20 = 0x4020
  MMU sends 0x4020 on wires to DRAM system

Page table small enough to fit in DRAM
  has 2\(^{20}\) entries, probably 4 bytes each
  now page table is only 4 MB, not so big

Where is the page table?
  in memory
  so can be modified with ordinary instructions
  CPU Page Table Base Register points to base
  so a memory reference conceptually requires *two* phys mem fetches
    one to get the page table entry
    one to get the actual data
  allows us to keep multiple page tables, quickly switch
  will come in handy later...

Flags
  "Page Table Entry"
  20 bits, plus some flags
  Valid/Invalid, Writeable/ReadOnly
  MMU only does translation if valid and correct read/write
  otherwise forces a "page fault":
    save state
    transfer to known "handler" function in operating system
Demand paging
page table:
0: 1, V
1: -, I
2: 0, V

and only two pages of physical memory!
if program uses va 0x0xxx or 0x2xxx, MMU translates
fault if program uses va 0x1xxx
handler(vpn):
   (ppn, vpn') = pick a resident page to evict
   write dram[ppn] -> disk[vpn']
   pagetable[vpn'] = -, I
   read dram[ppn] <- disk[vpn]
   pagetable[vpn] = ppn, V

Demand paging discussion
picking the page to evict requires cleverness
e.g. least recently used
we are treating DRAM as a cache for "real" memory on disk
can do that b/c programs use "names" (va) for memory
we have full control over what the names mean
map to DRAM
or arbitrary action in page fault handler

What do operating systems use page tables for?
Demand paging
Lazy loading of big programs
Zero fill
Address spaces for modularity, defend against bugs
   one page table per running program
   one program cannot even name another program's memory!
Shared memory for fast communication
   two different names for same physical page
   tight control over what's shared
Copy-on-write fork
Distributed shared memory

Closing
keep your eyes out for naming systems
be aware in your own work
   when introducing naming might help

****

finish up at start of next day

using virtual addressing as an example of naming
   has turned out to be powerful, many uses of one underlying mechanism
diagram: cpu, mmu, ram, simplified mapping table
programs "name" data with virtual addresses
mapping is valid (refers to dram), or causes "page fault" to o/s
   o/s fills in entry, returns
   program never the wiser
what can you do with this naming mechanism?
   demand page from disk: support big programs
   lazy start from disk
address spaces: different mappings for different programs
  o/s switches depending on which program is currently running
  cannot even *name* each other's memory
  contains bugs
shared memory, e.g. X client and server (on same machine)
distributed shared memory
  build parallel programs on a cluster of separate machines
  but with shared memory, all one big program
diagram: draw other machine, its ram and map
page fault: fetch, mark him invalid