6.828 2012 Lecture 3: O/S Organization

plan:
  O/S organization
  processes
  isolation

topic: overall o/s design
  what should the main components be?
  what should the interfaces look like?

why have an o/s at all?
  why not just a library?
  then apps are free to use it, or not -- flexible
  some tiny O/Ss for embedded processors work this way

key requirement: support multiple activities
  multiplexing
  isolation
  interaction

helpful approach:
  abstract services rather than raw hardware
  file system, not raw disk
  TCP, not raw ethernet
  processes, not raw CPU/memory
  abstractions often ease multiplexing and interaction
    and more convenient and portable

note:
  i'm going to focus on mainstream designs (xv6, Linux, &c)
  for *every* aspect, someone has done it a different way!
  example: exokernel and VMM does *not* abstract anything!

xv6 has only a few abstractions / services
  processes (cpu, mem)
  I/O (file descriptors)
  file system

i'm going to focus on xv6 processes today
  a process is a running program
  it has its own memory, share of CPU, FDs, parent, children, &c
  it uses system calls to interact outside itself
    to get at kernel services
  xv6 basic design here very traditional (UNIX/Linux/&c)

xv6 user/kernel organization
  h/w, kernel, user
  kernel is a big program
    services: process, FS, net
    low-level: devices, VM
  all of kernel runs w/ full hardware privilege (very convenient)
  system calls switch between user and kernel

good: easy for sub-systems to cooperate (e.g. paging and file system)
bad: interactions => complex, bugs are easy, no isolation within o/s
called "monolithic"; traditional and successful
worth thinking about what *has* to be in the kernel
Q: could FS be a user-level library? why / why not?
note: you could have a small kernel, most functionality at user-level
microkernel, exokernel

isolation is the most constraining consideration!
isolation determines much of the basic design
it's much of the reason why we need the notion of process at all
isolation will come up again and again

what is isolation?
the process is the unit of isolation
prevent process X from wrecking or spying on process Y
memory, cpu, FDs, resource exhaustion
prevent a process from wrecking the operating system itself
i.e. from preventing kernel from enforcing isolation
in the face of bugs or malice
e.g. a bad process may try to trick the h/w or kernel

what are all the mechanisms that keep processes isolated?
user/kernel mode flag
address spaces
timeslicing
system call interface

the foundation of xv6's isolation: user/kernel mode flag
controls whether instructions can access privileged h/w
called CPL on the x86, bottom two bits of %cs
CPL=0 -- kernel mode -- privileged
CPL=3 -- user mode -- no privilege
x86 CPL protects everything relevant to isolation
writes to %cs (to defend CPL)
every memory read/write
I/O port accesses
control register accesses (eflags, %cs4, ...)
every serious microprocessor has something similar

user/kernel mode flag is not enough
protects only against direct attacks on the hardware
kernel must configure control regs, page tables, &c to protect other stuff
e.g. kernel memory

how to do a system call -- switching CPL
Q: would this be an OK design for user programs to make a system call:
set CPL=0
jmp sys_open
bad: user-specified instructions with CPL=0
Q: how about a combined instruction that sets CPL=0,
but *requires* an immediate jump to someplace in the kernel?
bad: user might jump somewhere awkward in the kernel
the x86 answer:

there are only a few permissible kernel entry points
INT instruction sets CPL=0 and jumps to an entry point
but user code can't otherwise modify CPL or jump anywhere else in kernel
system call return sets CPL=3 before returning to user code
also a combined instruction (can't separately set CPL and jmp)
but kernel is allowed to jump anywhere in user code

the result: well-defined notion of user vs kernel
either CPL=3 and executing user code
or CPL=0 and executing from entry point in kernel code
not:
CPL=0 and executing user
CPL=0 and executing anywhere in kernel the user pleases

Q: could one have process isolation WITHOUT h/w-supported kernel/user mode?
yes!
see Singularity O/S, later in semester
but h/w user/kernel mode is the most popular plan

how to isolate process memory?
idea: "address space"
give each process some memory it can access
for its code, variables, heap, stack
prevent it from accessing other memory (kernel or other processes)

how to create isolated address spaces?
xv6 uses x86 "paging hardware"
MMU translates (or "maps") every address issued by program
VA -> PA
instruction fetch, data load/store
for kernel and user
there's no way for any instruction to directly use a PA
MMU array w/ entry for each 4k range of "virtual" address space
refers to phy address for that "page"
this is the page table
o/s tells h/w to switch page table when switching process
why isolated?
each page table entry (PTE) has a bit saying if user-mode instructions can use
kernel only sets the bit for the memory in current process's address space
paging h/w used in many ways, not just isolation
e.g. copy-on-write fork(), see Lab 4
note: you don't need paging to isolate memory
type safety, JVM, Singularity
but paging is the most popular plan

how to isolate CPU?
prevent a process from hogging the CPU, e.g. buggy infinite loop
how to force uncooperative process to yield
h/w provides a periodic "clock interrupt"
forcefully suspends current process
jumps into kernel
which can switch to a different process
kernel must save/restore process state (registers)
totally transparent, even to cooperative processes

called "pre-emptive context switch"
note: traditional, but maybe not perfect; see exokernel paper

back to system calls
i've talked a lot about how o/s isolates processes
but need user/kernel to cooperate! user needs kernel services.
what should user/kernel interaction look like?
can't let user r/w kernel mem (well, you can, later...)
kernel can r/w user mem
  but don't want to do this too much!
so style of system call interface is pretty simple
  integers, strings (copying only), user-allocated buffers
  no objects, data structures, &c
  never any doubt about who owns memory
let's illustrate by tracing sys calls in xv6

on-screen:
xterm -fn 10x20
illustrate sh.c exercise
draw parent/child diagram
echo hi
echo hi > x
echo hi | wc

6.828 Operating System Engineering

Fall 2012