The Kernel Abstraction

CSE 006
12:30 Today
Physical Address Layout

0x00000000

DRAM

0x7FFFFFFF

Audio Controller
0xC0001000

Keyboard Controller
0xC0002000

Disk Controller
0xC0003000

0xFFFFFFFF = 2³²-1

Physical Address Ranges
Device I/O

- Polling vs. interrupts
- Programmed I/O vs. DMA
- One operation at a time vs. queue of operations
Question

- What (hardware, software) do you need to be able to run an untrustworthy application?

Memory range limit
System calls limit
which user/apps have which permissions
halt app - timers interrupt
limit apps to change permissions
Main Points

• Process concept
  – A process is the OS abstraction for executing a program with limited privileges

• Dual-mode operation: user vs. kernel
  – Kernel-mode: execute with complete privileges
  – User-mode: execute with fewer privileges

• Safe control transfer
  – How do we switch from one mode to the other?
Bootstrapping:

1. BIOS copies bootloader.
2. Bootloader copies OS kernel.
3. OS kernel copies login application.

Physical Memory:

- BIOS
  - Bootloader instructions and data
- OS kernel
  - Instructions and data
- Login app
  - Instructions and data
Challenge: Protection

• How do we execute code with restricted privileges?
  – Either because the code is buggy or if it might be malicious

• Some examples:
  – A script running in a web browser
  – A program you just downloaded off the Internet
  – A program you just wrote that you haven’t tested yet
Process Abstraction

• Process: an *instance* of a program, running with limited rights
  – Thread: a sequence of instructions within a process
    • Potentially many threads per process (for now 1:1)
  – Address space: set of rights of a process
    • Memory that the process can access
    • Other permissions the process has (e.g., which system calls it can make, what files it can access)
Thought Experiment

• How can we implement execution with limited privilege?
  – Execute each program instruction in a simulator
  – If the instruction is permitted, do the instruction
  – Otherwise, stop the process
  – Basic model in Javascript and other interpreted languages

• How do we go faster?
  – Run the unprivileged code directly on the CPU!
Dual-Mode Operation

• Kernel mode
  – Execution with the full privileges of the hardware
  – Read/write to any memory, access any I/O device,
    read/write any disk sector, send/read any packet

• User mode
  – Limited privileges
  – Only those granted by the operating system kernel

• On the x86, mode stored in EFLAGS register
• On the MIPS, mode in the status register
A Model of a CPU

- `add r1, r2, r3`
- `adds %eax, %ebx, %ecx`
- `jmp 0x1000`
- `bne 0x1000`
- ` Orr2`

**CPU**

Branch Address

**CPU** Instructions Fetch and Execute

- `opcode`
- `New PC`
- `Program Counter`
- `Select PC`
- `interrupt`
- `int`
- `type`

**Model of a CPU**
A CPU with Dual-Mode Operation

Branch Address

Handler PC

New PC

Program Counter

CPU Instructions Fetch and Execute

Select PC

New Mode

Select Mode

Mode

opcode
Hardware Support for Dual-Mode Operation

- Privileged instructions
  - Available to kernel
  - Not available to user code
- Limits on memory accesses
  - To prevent user code from overwriting the kernel
- Timer
  - To regain control from a user program in a loop
- Safe way to switch from user mode to kernel mode, and vice versa
Question

• For a “Hello world” program, the kernel must copy the string from the user program memory into the screen memory.

• Why not allow the application to write directly to the screen’s buffer memory?
Privileged instructions

• Examples?
  
  change elflags
  change page table
  
  quit

• What should happen if a user program attempts to execute a privileged instruction?
  
  kill it
  error flag
  exception -