1 Announcements

Midterm on Tuesday (GEO Museum 100)
Assignment 3 Posted

2 Process

The term process has two useful definitions:

1. book’s definition: an instance of a program in execution (i.e. echo)
   - if you run a program more than once, it’s the same program, but it runs in a different context
   - the memory that the programs occupy is different

2. an abstract computer: all of the hardware is abstracted
   - a process gives a program the illusion that all hardware resources of the computer are at its disposal
   - it thinks that it owns them “all” (register and CPU)

3 Process Isolation

- concept: a process affects other processes only as explicitly permitted
  - different operating systems have different ideas of what permitted means
  - on a server there are many individual user processes running at the same time and those processes are not allowed to interact (i.e. cannot kill each other)
  - one of the great ideas of computer science: before process isolation, any mistake you made could ruin the computer you were running on

- Violations
  - P modifies Q’s registers
- P modifies Q’s memory
- P prevents Q from running

- Implementation: How do we implement process isolation?
  - requires hardware support
    * process code must not have full rights and privileges over the machine’s
      privileges (otherwise it might violate process isolation)
  - need hardware support for different machine levels of code
    * **Privileged Code**: full access to all of the machine’s capabilities
      - Kernel: software on your machine that has full access
    * **Unprivileged Code**: partial access
  - Once we have instructions that a process is not allowed to execute, what is
    the correct response to an invalid instruction from a process?
    * CPU has a default response (reboot)
    * Is a reboot a violation of process isolation? Yes
    * because we have dangerous instructions, we have to have responses
    * it’s the kernel’s job to manage and monitor instances of process isolation
      violation (Kernel is isolation police!)
      - if process causes an abort, kill the process
      - How does the kernel get control? CPU safely transfers control to
        kernel when it observes a violation using exceptional control flow
        (interrupts). The CPU observes dangerous instructions and needs to
        transfer control to kernel. Need to have exceptional control flow
        (aka interrupts)
        - can also use exceptional control flow to safely run kernel at the pro-
          cess’s request (system call)

4 Example: Process Isolation

coding example
  * sys_getpid
  * code is data: can write code into the kernel in order to activate an attack
  * insertion of while attack in kernel causes an infinite loop that is never interrupted

previous lecture: we introduced a timer interrupt (interrupts CPU periodically)
  * instruction that activates timer interrupt: programmed I/O instruction using ‘out’
if process can execute program I/O instruction, it can turn off timer interrupt
(which violates process isolation: concrete example of why we need hardware support)

Question: What counts as hardware?
    anything except CPU or memory: special instructions for interacting with them
    (disk, timer, printer, mouse, keyboard, etc)

other dangerous instructions:
    • CLI: turns off timed interrupts
    • changing privilege levels

5 Problems

problem: memory is NOT isolated
    • making mov a dangerous instruction would be far too expensive

solution: virtual memory

What does virtual memory do?
    • virtual memory maps addresses used by the processes to physical memory on the machine
    • memory is an array of bytes that is scrambled by virtual memory
    • think of virtual memory as a function

virtual memory = ADDRESS SPACE function AS
    AS(ptr, privilege level) is either
        \[\begin{align*}
        &\text{physical memory address} \\
        &\text{fault}
        \end{align*}\]

This layer of indirection (virtual memory) ensures that a process cannot access kernel memory. This prevents attacks like the while(1) attack.
It allows us to say that a process cannot access a specific space in memory. We initialize our virtual machine so every address maps to a physical address. However, there are functions that let you change that mapping.
Process can only access memory that is above the reserved memory (0x100000 or higher). When there is a memory protection problem, the computer can change to a page fault exception and kill the process.

What else can we do beside kill the process?

- take a snapshot of all the machine’s registers
- skip over instructions

What happens if we ignore invalid instructions instead of killing the process? It’s like skipping a direction in mapquest and expecting to get to the correct location.

- try to fix the problem and rerun instruction (patch the fault)
- start another process

The kernel has full control over the abstract machine that implements every process.

7 How is Virtual Memory Implemented?

x86 Virtual Memory: implemented using a TWO-LEVEL PAGE TABLE

- basically a tree with depth 2
- PAGE DIRECTORY (root node of the tree): array of 1024 pointers represent as pageentry_t (32 bits long)
  ex. pageentry_t kernel_pagedir[1024];
- PAGE TABLE PAGES: level one nodes (also contains an array of 1024 pointers)
- each entry in page directory is an address of a page table page
• can change intermediate page table locations without changing the output
• How big is a page table page? 4096 locations
• when a CPU encounters an empty page table page then a page fault occurs
• input: virtual address (32 bits long)
• output: physical address (32 bits long)

8 Example: Virtual Memory

input values are broken down into three parts

1. page directory index (bits 22 - 31): index into page directory array
2. page table index (bits 12 - 21): index into page table page array
3. offset (bits 0 - 11): added to physical address that we look up

example: virtual address 0x00403005
• in binary: 0000 0000 0100 0000 0011 0000 0000 0101
  – value of top ten bits (page directory index): 1
  – value of next ten bits (page table index): 3
  – value of bottom twelve bits (offset): 5
• corresponds to physical address 0x10000005