Virtualization, Concurrency, and Persistence

01/19/2023

Professor Amanda Bienz

Review : OS Responsibilities

- Make it easy to **run** programs
- Allow programs to **share** the system
- Enable programs to **interact** with devices
- Providing programs a **portable** system interface

Review : System Call

- **Allows user to tell the OS what to do**
- Mode bit flips to kernel mode, system call is executed, and then back to user mode to continue application program
- OS provides an interface for these system calls (APIs, standard library)
- Include running programs, accessing memory, accessing devices

Review : Resource Manager

- OS manages resources (i.e. CPU, memory, disk)
- OS allows:
  - Many programs to run (sharing the CPU)
  - Many programs to concurrently access their own instructions and data (sharing memory)
  - Many programs to access devices (sharing disks)

Themes of Course

1. **Virtualization**

    - Provide a high-level interface to hardware for each program
    - Each program runs in it’s own semi-contained environment
    - CPU, Memory, and I/O is all virtualized

Themes of Course

2. **Concurrency**

- Multiple activities going on at once, sometimes not transparently
- What kinds of bugs and performance issues come up?
- How to manage, control, and program concurrency

Themes of Course

3. **Persistence**

   - Storage virtualization is particularly difficult
   - Wide range of hardware technologies with different tradeoffs
   - How to manage state to deal with crashes?

Virtualization

- The operating system takes a physical resource and transforms it into a virtual form of itself
    - **Physical resource :** Processor, Memory, Disk, …
- ***The virtual form is more general, powerful, and easy to use***

Virtualizing the CPU

- System has very large number of virtual CPUs
    - Turns a single CPU into a seemingly infinite number of CPUs
    - Allows many programs to seemingly run at one time
    - Let’s step through a code example (virtual_cpu.c)

Virtualizing Memory

- Remember, physical memory is an array of bytes
- Instructions and data for each program are held in memory
- **Read memory** (load):
  - Specify an address to be able to access the data
- **Write memory** (store):
  - Specify the data to be written to the given address
- Let's write a program that updates memory (virtual_mem.c)

Virtualizing Memory

- **Each process accesses its own private virtual address space**
  - The OS maps address space onto physical memory
  - A memory reference within one running program does not affect the address space of other processes
  - Physical memory is a shared resource, managed by the OS

Concurrency

- The OS is juggling many things to do at once
  - First running one process
  - Then another
  - And so forth
- Modern multi-threaded programs also exhibit the concurrency problem
- Let’s write a simple multithreaded program that updates a variable (thread.c)

Multithreading Issues

- Increment a shared counter (three instructions)
  1. Load the value of the counter from memory into register
  2. Increment value
  3. Store it back in memory
- These instructions do not execute atomically
- OS needs to avoid similar problems with concurrency

Persistence

- Devices such as DRAM store values in volatile memory (requires power)
- Hardware and software need to store data persistently
  - **Hardware**: I/O device such as hard drive, solid-state drives (SSDs)
  - **Software:** File system manages the disk and is responsible for storing any files that the user creates
- Let’s write a program that creates /tmp/file and writes the string “hello world” to the file (persist.c)
  - What are the system calls in this program?

Persistence

- What does OS do in order to write to disk?
  - Figure out where on disk the new data will reside
  - Issue I/O requests to underlying storage device
- File system handles system crashes during write
  - Journaling or copy-on-write
  - Carefully ordering writes to disk