CMPSCI 577 Operating Systems Design and Implementation **Spring 2020**

Lecture 2: Jan 23

Lecturer: ***Prashant Shenoy*** *Scribe:* ***Hee Hwang (grad)***

Architecture is important because OS directly interacts with hardware.

2.1 Intro

Generic computer architecture includes CPU, I/O devices, memory, and system bus. To facilitate these devices, OS provides the following functionalities.

Process and Thread Management

Concurrency

I/O devices

Memory Manaagement

Files

Distributed systems & networks

From these functionalities, we want to focus on architectural features about OS services.
These are topics we are interested in lecture 2.

OS Service	**What is it?**
Protection	You don't want to have a process go over other process
System calls	OS procedure that executes privileged instructions using trap
Scheduling	Deciding when to execute
Synchronization	Atomic instructions e.g. lock, semaphore
Virtual memory	Memory seen by application, not necessarily be the actual one (It uses TLB)

2.2 Protection

There is a set of assembly instructions supported by CPU. x86, intel, arm have their own set of instructions. They are very similar and have privileged instructions. For example, IO addressing, memory allocation, and interrupts are critical to OS behavior. We essentially do not want a random application to mess up with these functionalities for fun. Generally, there is a protection bit that decides "modes" that allow these sensitive instructions or not. Different chipmaker supports a different number of modes. Some processor has more than two modes; for instance, the Intel processor has four modes.

2.3 System Call

We know that the CPU has sensitive instructions, and the user cannot use these directly. However, what if we need them? We need to "ask" OS to do it. These are system calls.
System Calls: Exposed OS functionalities to the application process Every OS has a set of system calls. First, a user executes the system call, and the OS raises the trap. This trap flips the mode bit and throws the process into kernel. The process stores current status as well. Now is the kernel's turn. It validates the user's call and does a sensitive job requested by the user. Finally, the kernel kicks the process out to the user area.

2.4 Memory Protection

You do not want an application to touch another process. How do we ensure this? We use registers. For example, register checks the boundary of a memory address. The memory area breaks down into four parts, stack, data, and text area. Heap grows up, and the stack grows down. In reality, these addresses are not contiguous.

2.5 Registers and caches

Registers are the fastest memories managed by CPU. Two kinds of registers are available. General-purpose registers such as AX, BX, and CX contains frequently accessed variables while special-purpose registers have program counter, stack pointer, and frame pointer. Accessing the main memory turns out to be very expensive, and registers cannot cut it. L1, L2, and L3 cache are small and fast memory for preventing us from accessing the main memory every time. One noticeable difference between RAM and cache is that RAM shares its address space while the cache does not.

2.6 Writing back and writing through

We have argued that accessing the main memory is costly and uses cache for it. The question is that, how do we propagate changes? These changes may alter the cache or not. There are two methods of applying these changes to the main memory from the cache. The first one is write-back, which delivers changes to the main memory only if the cache is modified. Write-through always maintain the main memory up-to-date.

2.7 Traps

Traps are software interrupt. For example, dividing by zero is illegal, and OS wants to let others know that something is wrong. Page fault, write to a read-only page, overflow, and system calls are special conditions that trigger traps. When this happens, program execution halts, and the hardware saves its state. The hardware transfers control to trap the handler by jumping to the address in the trap vector and executes it. After completion, the previous program continues execution.

2.8 I/O

There are three I/O methods. The first one is synchronous I/O, a.k.a Blocking I/O. When a user requests a certain task to the kernel, the user cannot do anything until it gets a message from the kernel. Asynchronous I/O, unlike the previous one, a user can do freely after requesting a task to kernel. This behavior explains its another name, non-blocking I/O. It will know the task completion by getting interrupted that signals job completion. However, this requires hardware support. Memory-mapped I/O takes device and gives fake address as if it is RAM. For example, hard disk, graphics card, and keyboard are examples of memory-mapped I/O, which enables direct access.

2.9 Virtual Memory and TLB

Virtual memory is an abstraction of having unlimited memory. In other words, we want to use disk as a back-up for RAM. Since accessing drive is much much slower than RAM, we use TLB that speeds up this. TLB(Translation Lookaside Buffer) is a special kind of cache that keeps where pages of memory are stored. TLB boosts up virtual memory performance.