CMPSCI 630 Systems **Spring 2018**

Lecture 6

Lecturer: Emery Berger *Scribe: Angela Upreti*

6.1 Interlude: Software Architecture

In a client-server architecture and other time sharing architectures, virtual memory(VMM) allows the illusion that every client has all of the address space. In most systems, memory is virtually tagged and caches are physically tagged. Cloud service providers such as amazon take advantage of VMM and VMs to provide a seemingly isolated environment to their clients. They also guarantee timely execution which means an appropriate scheduling is necessary to meet this guarantee. We talked about naive round-robin scheduling and why round-robin would not be enough.

6.2 Context Switch

Time-slicing allows efficient use of the processors. If a process running on a processor is I/O bound, the idle cpu time can be used by another process. Before switching to another process, the current processor state has to be saved and another process's state has to be restored.This has a lot of overhead because context switch involves crossing the kernel boundary.

Overheads of a Context Switch

1. Registers
2. Vector Registers
3. Program Counter
4. TLB : Page table maps virtual memory space to physical address. TLB is like a cache of the page table. It stores recently accessed virtual memory addresses. It is much faster than looking up the page table. x-86 page table is stored as a radix tree which is expensive to trace in case of a cache miss. Every core has its own TLB. Whenever, a page table is modified by one of the processes, the TLB entries storing that mapping need to be invalidated for all processors. This is expensive. While discussing the TLB, we talked about the possibility of a shared TLB and other shared resources. Isolating resources between processors would provide better security but resources might go under utilized.
5. Cross-kernel boundary

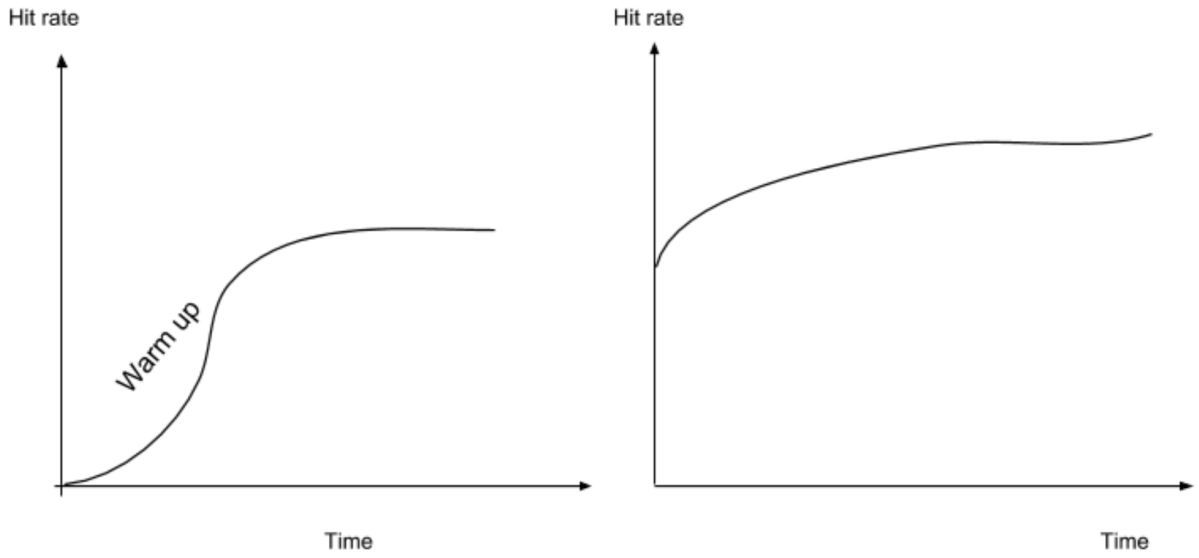
Timeline of a context switch

1. A process P_1 does a system call.
2. P_1 is waiting in the queue for execution.
3. P_1 joins the ready queue after it is signaled.

If we have two processes P_1 and P_2,

P_1——————- $t(q_1)$ ——————P_2——————

If q_1 is too long , memory is cold for P_2.



The image on the left shows the phase where the cache warms up at the beginning and achieves an optimal hit rate. The image on the left eliminates the warm-up rate. This can be achieved by sharing memory pages between processes.

Time Quantum length

Related to duration that a page stays in cache is the value of time quantum. A long time quantum provides higher throughput and a smaller quantum provides lower latency. Long time quantum will likely make better use of pages cache.

6.3 Scheduling

6.3.1 Priority-based Schedulers

: This is a multi-level preemptive scheduling policy. There are multiple ready queues and each queue has a priority level assigned to it. The queue with the highest priority is scheduled first. A round-robin scheduling is used within a queue. Priority-based scheduling can lead to starvation. If a process with a higher priority is always present, a low-priority process might never run. Real-time applications cannot use priority based scheduling.

6.3.2 Proportional Share Schedulers

These are based on lottery tickets. All processes get some lottery tickets. The number of tickets a process gets is based on the proportion of cpu time we want to allot it. A lottery ticket would be randomly sampled. This randomness in scheduling provides better security.
Independence: Between two sampled elements s_i and $s_j \in S$, $corr(s_i, s_j) = 0$. **Cryptographically secure** : Given samples $s_0, s_1, s_2, ..., s_n$, one cannot figure out s_{n+1}.

6.3.3 Scheduling-bases on Deadlines

Earliest deadline first would be an example of this.

6.3.4 Boosting

Processes with higher CPU utilization get higher priority. I/O bound processes get a lower priority. This leads to really good CPU utilization.

6.3.5 QoS Guarantee

An example would be '10 percent of the time every window of 100ms with a probability of 0.9.'