Tuesday, December 4, 2018

CH 19, 20, 21 , 22

CH 19 – Paging Faster Translation

- With paging, the major challenge that needs to be overcome is its speed. This is overcome with the help of hardware help, down as the Translation Look-aside Buffer (**TLB**). This TLB would be holding the information that would be needed to translate a virtual address into a physical address. If the TLB has the information, it would be known as a **TLB Hit**, and the translation would take place, if it does not have the required information, it would be called a **TLB Miss**. In this case there would be the cost associated with paging that would be incurred. Once the Page Table Entry is calculated, it would be checked whether it is valid, or if it is accessible, if so TLB would cache the information and then the virtual addr would be translated into a physical address.
- **Spacial Locality** – It is taken into consideration when most of the entities are very close to each other, and while accessing one of them, increases the probability of a TLB hit for the other memory locations with the entities near them. **Temporal Locality** – It is taken into consideration, when a block of memory recently accessed is accessed again, which increases the probability of achieving all Hits, due to the relative indifference in time, due to TLB caching. Also, Page size is inversely proportional to TLB Misses
- The management of TLB misses was in olden days done by the Hardware, but now with modern systems, it is done entirely by the OS. In the case of a TLB miss, OS would simply raise an exception, which would then handled by the trap handler, however, there is a key difference. In case with TLB misses, the return from trap function would require the execution of the instruction that caused the trap in the first place, and not the execution of the next instruction as would be the normal case. Additionally, OS needs to ensure that it does not cause an infinite chain of TLB misses, which could be solved by having the TLB miss handlers in the physical memory.
- TLB issues with **context switches**. It is imminent that two different processes share the same virtual address, pointing to different physical frame. This would lead the TLB to get **confused** when context switch takes place. To solve this problem, one of the ways to go ahead would be to make the **TLB flush** every time on a context switch, **but** that would cause a lot of **overhead** because it would lead to TLB misses as it touches the code and the data pages. To solve this, instead of flushing, we can use a **Address space identifier** (*ASID*) which is akin to the process identifier (*PID*), with less bits (8 bits) to identify to which process a particular virtual address maps to. There could also be processes, that refer ti the same physical frame number, as in cases where different programs share the code segment, which is common and OK.

Tuesday, December 4, 2018

CH 20 – Smaller Tables

- Just ask kush about the final address translation
-

CH 21 — Beyond physical memory: Mechanism

- To allow for a larger virtual memory than what the physical memory allows, we would have to use the disk for the storage of the pages to be swapped in and out called **swap space**. To do this, some disk place would have to be received for such swapping to happen, and the OS also needs to remember this disk address for a given page.
- So there are two possibilities with the use of TLB — **TLB Hit** or **Miss**, in the case with miss, the OS would have to look into the swap space of the disk to determine whether or not the page is present in the disk. To enable this, we need additional machinery to support this, specifically a present bit that would tell the TLB, whether the page that it is currently searching for is in the physical memory or in the disk somewhere. The act of accessing a page that is not in the physical memory us commonly referred to as **page fault**.
- So when a page fault is encountered, the OS would then look into the PTE to find the address in the disk and then use this to issue and I/O to fetch the page back into memory. While this happens, the page would be in a *blocked* state, and the OS can run any other process. Once the I/O is complete, the OS would update the PFN field of the PTE to record the in-memory address of the page. When the TLB tries to access the page, it would get a TLB miss, and then it would be loaded in the TLB, which when the return from trap function happens would result in a Hit. If however the memory or the swap space is full, a page replacement policy would be used to load the desired page in memory by swapping the not-required pages.
- There are three scenarios in which TLB miss might occur. **First** is when the page is both valid and present, TLB miss handler would simply grab the PFN from the PTE. The **second** case is when a page fault occurs, that is, the page is not present in the physical memory. The **third** case is when the page being accessed in invalid.
- To keep a small amount of memory free, the OS would keep some kind of ***high watermark*** and ***low watermark***, which would be responsible for keeping the memory in check. If the memory available Is lower than the low watermark, then the eviction of pages start, until the free memory reaches the high watermark level. All this is done by a background thread called the swap daemon.

CH 22 – Beyond physical memory: Policies

- The main focus would be to minimize the number of cache misses. To calculate the Average Mean Access Time (**AMAT**) is $AMAT = T_M + (P_{miss} \,.\, T_D)$, where T_M is the cost of accessing memory, T_D is cost of accessing disk, and P_{miss} is the probability of missing a cache. The value of T_M << T_D, hence it is imperative to reduce the number of cache misses.
- Optimal cache replacement policy. In this case, the page that would be accessed the farthest would be replaced. Also, if the cache size is x, then there would be x number of misses, credits to **cold start** or **compulsory misses**.
- **FIFO Policy**: In this the page that was accessed the first would be evicted first. However, there is special case with *Belady's Anomaly*, in which increasing the cache size increases the miss rate.
- **Random Policy**: In this policy, the pages to be evicted are randomly chosen. In some cases, Random does better than the FIFO policy and as good as the Optimal policy, but it is entirely dependent on the luck of Draw.
- **Least Recently Used**: LRU and LFU are based on the *Principle of Locality*. LRU keeps in mind the the history of the pages being used, and would then evict the page that has been Least Recently Used. Where as LFU, would keep in mind the past frequency of usage of the pages and would then replace the page that has been Least Frequently Used.