Lecture Notes - Virtual Memory (II)

Debadatta Mishra
Indian Institute of Technology Kanpur

Design issues in paging .

- Number of levels in a hierarchical page table depends primarily on two factors---(i) width of virtual address, (ii) page size
- With increasing physical memory capacity in systems, virtual address width is bound to increase. With increased virtual address width, number of page table levels will increase if the page size is kept fixed. By implication, the translation overheads will also increase.
- Virtual memory systems with larger page size can reduce the memory access overhead but results in memory wastage due to fragmentation. For example, if an application requires 73KB memory, with page size 4KB, memory wastage is 3KB while with a page size of 16 KB, memory wastage is 7KB.
- Memory required to maintain page table levels in a hierarchical paging mechanism depends primarily on virtual memory layout of applications and page size.
- With a smaller page size, memory usage to maintain page tables for application using sparsely layed out virtual address spaces can be minimized.
- With a large page size, memory usage to maintain page tables in a sparsely used virtual address space is more. This is due to wasted memory within page table entries.
- OS can allocate contiguous virtual address space to minimize memory required to maintain page tables.

Translation efficiency

- In absence of caches, virtual address translation requires memory access at each level during page table walk. This can be a huge source of overhead even for applications with low memory usage footprints.
- CPU caches (L1, L2 etc.) can help improving the translation latency by caching the page table entries for each translation level.
- There are two main limitations related to effectiveness of caches in expediting the page table walk,
  - Translation of a given address, requires N cache entries and N cache accesses, where N is the number of page table levels.
  - Caches are used to cache recently accessed data, not dedicatedly used to cache page table entries
- A special cache, commonly known as Translation Lookaside Buffer (TLB) is used to store recent translations. For any virtual address originating from the processor,
  - Virtual page to PTE translation is looked up in the TLB
  - On a TLB hit, the physical address is determined from the PTE and memory access is performed

    - On miss, page table walk is performed to find out the last level PTE (pointing to the PFN), a new entry is made in the TLB and memory request is forwarded.
- TLB can be helpful in avoiding page table walks for frequently accessed memory locations. Especially, it comes really handy for code segments as their size is small and their accesses provide good temporal and spatial locality.
- In most architectures (including X86), two TLBs are maintained---instruction TLB is separated from the data TLB.
- Huge memory workloads with random data access patterns negates the benefits of TLB.
- How TLB is shared across multiple applications?
    - Why not share the TLB as it is? As the virtual address range of applications overlap, using virtual page as the key does not work.
    - One solution is every time the page table is changed (through a CR3 switch), flush the TLB entries. This results in higher overheads and negatively impacts the benefits of TLB.
    - Another approach is to extend the key (virtual page) by adding the process ID commonly known as Address Space Identifier (ASID). Using this approach, an address is looked up using <ASID, virtual page> as the key in the TLB. In this solution, flushing the TLB is not necessary on a CR3 switch.

OS virtual address space

- In protected mode (with paging enabled), all addresses are virtual. In a paging enabled system, OS must have its separate address space which is not accessible by the user space applications (from ring-3).
- One solution is to have a separate page table for the OS. For every entry into OS (through syscall, interrupts etc.), the CR3 is loaded with the level-1 page table address containing the translation hierarchy for OS used virtual addresses. On return from OS to user space process, the page table address for the process is loaded to the CR3 register.
    - This causes significant overheads due to CR3 switch and a possible TLB flush (if ASID is not used) for every entry into and exit from the OS.
    - Some OS code page frames are required to be mapped to all the application process page tables if hardware does not support CR3 switching during entry into the OS.
- The second solution is to partition the virtual address space into two parts---one part for the user and another for the OS. For example, for 32-bit virtual address, the OS virtual address space in each process can be 0x0 - 0x7FFFFFFF and the user virtual address space can be 0x80000000 - 0xFFFFFFFF.
- The OS virtual address is made accessible only in ring-0 by appropriately setting the access permission bit in the page table entries.
    - While this scheme is efficient it may lead to security issues. For example, if somehow one malicious application can trigger access to the OS virtual address space (e.g., through speculative execution), the OS memory location contents can be indirectly inferred. Interested readers can refer to Meltdown

(https://meltdownattack.com/meltdown.pdf and Spectre
(https://spectreattack.com/spectre.pdf) attacks.