Lecture Notes - Memory overcommitment

Debadatta Mishra
Indian Institute of Technology Kanpur

Memory over-provisioning/overcommitment

- Most operating systems support memory overcommitment. In overcommitted scenarios, the sum of promised memory to all applications exceeds memory capacity of the system. Memory overcommitment is particularly useful to increase degree of multiprogramming in a system.
- Application programmers are greedy with respect to their memory usage behavior. In general, applications overestimate memory requirement, do not perform just-in-time allocation, and more importantly, the average active memory of applications is significantly less than their worst case memory requirements.
- The OS should manage memory (and employ overcommitment techniques) in an application transparent manner. By implication, applications' view of memory and the memory related system calls should remain unchanged in overcommitted scenarios. Virtual memory abstraction detaches the applications from the physical memory and plays a vital role in supporting memory overcommitment.
- Ideally, memory access time of applications should not suffer because of memory overcommitment, especially when the total memory demand (also known as the working set) of all active applications fits in memory. However, if working set is more than memory capacity, sluggish performance is inevitable. In such a case, the OS should employ policies to efficiently utilize the memory and improve application performance as much as possible.

Memory overcommitment using paging

- Paging facilitates dynamic memory allocation by allowing the OS to modify the virtual to physical mapping through the page tables at run-time. Any particular virtual page can be provided with any free physical frame at run-time.
- Dynamic mapping facility can support the following overprovisioning primitives,
    - Lazy allocation: When applications request to allocate memory, only virtual address is allocated. Physical allocation takes place when the application accesses the allocated virtual page.
    - Swapping: The OS can backup (swap out) some physical pages mapped to applications into a backup storage (e.g., hard disk) to reduce physical memory contention. When the particular virtual address is accessed, the content can be read from the backup storage (swap in) into a page frame and mapping from the virtual page to physical page is updated in the page tables.
- There are two primary hardware features used by the OS to achieve dynamic page table adjustments,
    - Present bit (bit-0 in x86) of PTE reflects whether a virtual to physical mapping is valid.
    - If the hardware page table walker encounters a mapping with present bit set to 0, it will raise a page fault that can be handled by the OS.

- The page fault handler in OS should check the validity of the accessed virtual address to ascertain that the access is to a legitimate address. If the OS finds that the application is accessing memory beyond what it had allocated, the OS injects a error (a.k.a. SIGNAL). If the access is to a valid virtual address, then the OS finds a free page frame and updates the page table mapping for the faulting virtual address.
- The OS maintains a list of virtual address segments which are allocated by the applications. As mentioned in the previous point, this is necessary to demarcate legitimate virtual address ranges of the applications.
- In a overcommitted scenario, when the free memory in the system goes below a threshold, the OS employs swapping as the last resort. A victim page is selected using page eviction algorithm (e.g., LRU), the content of the page is written to a disk block, the corresponding PTE is updated with present bit = 0 and entry address = disk block address. When the same page (virtual address) is accessed, the hardware raises a page fault (as the present bit = 0) and the OS finds a free PFN to copy back the content from the disk and update the page table mapping (with present bit =1). After which the control returns to the application by re-executing the faulting instruction.
- To obtain PTE from a physical page, the OS needs to maintain a reverse mapping from PFN to PTE. This becomes even more challenging when a single page is shared (mapped to) by multiple virtual addresses. In such a case, a list of PTEs is required to be maintained.

Page replacement algorithms

- To select a victim page, there are several algorithms proposed by the community and still is an active area of research. Primary objective of page replacement algorithms is to minimize the number of page faults (swapping). Minimizing swapping is particularly important considering the performance implications due to orders of magnitude speed mismatch between the DRAM and hard disk.
- Another aspect that plays a vital role in using a page replacement algorithm is ease of implementation. Algorithms which require sophisticated information maintenance may require additional overheads. For example, implementation of LRU scheme requires maintaining a sorted list based on the last access time, which could be too much overhead since the list has to be updated on every memory access.
- Belady's MIN algorithm looks ahead into future and selects the page (in memory) which is accessed after the longest time. This algorithm is proven to be the most optimal but trivially impractical. However, MIN gives a nice comparison basis for all page replacement algorithms.
- Please refer the slides for more on page replacement algorithms.