Virtual Memory – Address Translation, Paging and Segmentation Schemes

Administrivia

- Timing of latest written assignments and projects
- Today: preparation for the midterm and the latest homework
- Homework emphasizes “thinking” questions, but there are also some that directly test “knowing”
    - There are rarely best or correct solutions - there are design tradeoffs to be made
    - State your assumptions and go from there
    - The stress is on practicality

Virtual Memory

- Why virtual memory? Revisiting the argument of “creating an illusion” from Section #2
    - Another reason: the need to keep several processes in memory
        - To maximize CPU utilization
        - Disk access is >1000 times slower than memory access (low ms vs. low µs range)
            - Memory is fast, (relatively) small, volatile
            - Disks are (relatively) slow, large, non-volatile

- New terms: paging, segmentation, swapping, fragmentation

Paging and Segmentation

- Schematics of how address translation works with both
    - Use of a page table / a segment table per process
- Important distinctions between the two schemes
    - Segments are of variable length (not necessarily a power of 2), while pages are of fixed length (a power of 2)
        - “Pages are fixed-size small segments.”
    - Addressing
        - Single address (with paging) vs. segment ID + offset (with segmentation)
        - User's view: one-dimensional (with paging) vs. two-dimensional (with segmentation)
    - Need for protection
        - Can't get a wrong address with paging since the offset is fixed length
        - With segmentation, one needs to check the condition (offset < limit)
        - Inherently finer-grained protection and sharing with segments (e.g., array sizes)
- How to avoid a huge page table (per process)
    - In 32-bit architectures with 4KB-sized pages, there are potentially 2^{20} ~ 1,000,000 pages
        - The naïve approach is to set aside 4MB of memory just to store the page indices
    - Hierarchical paging comes to the rescue (there are other schemes too)
        - But it requires at least two page table (i.e., memory) accesses for every address translated
            - … which goes against the desire to have address translation as efficient as possible
        - Idea: exploit the spatial locality of programs (and data)
            - Using TLB (translation look-aside buffer) to cache page index translations saves most of those memory accesses, because of the locality and the fact that the TLB is implemented in hardware (i.e., accessing it is faster than a memory access).