

## CS 414 : Operating Systems

UNIVERSITY OF VIRGINIA  
Department of Computer Science

Spring 2008

### Topic 13: Sharing Main Memory -- Paging

- Readings for this topic: Ch.8 (8.4 & 8.5)
- Paging: goal is to make allocation and swapping easier.
  - Make all chunks of memory the same size, call them *pages*. Typical sizes range from 512-8k bytes.
  - For each process, a *page table* defines the base address of each of that process' pages along with read-only and existence bits.
  - Translation process: page number always comes directly from the address. Since page size is a power of two, no comparison or addition is necessary. Just do table lookup and bit substitution.
  - Easy to allocate: keep a free list of available pages and grab the first one. Easy to swap since everything is the same size, which is usually the same size as disk blocks to and from which pages are swapped.
- Problems of paging:
  - Internal fragmentation: The larger the page, the worse this is.
  - Efficiency of access: even small page tables are generally too large to load into fast memory in the relocation box. Instead, page tables are kept in main memory and the relocation box only has the page table's base address. It thus takes one overhead reference for every real memory reference.
  - Table space: Page tables are big. How big? Consider a 32-bit addresss space with 4k pages as in Windows XP.

- Paging with segmentation: use two levels of mapping to make tables manageable.
  - Each segment contains one or more pages.
  - Segments correspond to logical units: code, data, stack. Segments vary in size and are often large. Pages are for the use of the OS; they are fixed-size to make it easy to manage memory.
  - Going from paging to P+S is like going from single segment to multiple segments, except at a higher level. Instead of having a single page table, have many page tables with a base and bound for each. Call the stuff associated with each page table a segment.
- We can share at two levels: single page, or single segment (whole page table).
- Pages eliminate external fragmentation, and make it possible for segments to grow without any reshuffling.
- If page size is small compared to most segments, then internal fragmentation is not too bad.
- The user is not given access to the paging tables.
- Problem with segmentation and paging: extra memory references to access translation tables can slow programs down by a factor of two or three. Too many entries in translation tables to keep them all loaded in fast processor memory.
- Page table structures
  - Multi-level (hierarchical) paging: paging the page table
    - For big address space, two-level paging may not be enough.
  - Hashed page tables: page number is the virtual address is hashed into the hash table.
    - Each entry in the hash table contains linked list of elements consisting of 1) virtual page number, 2) physical page number, 3) pointer to the next element in the link list.
  - Inverted page tables: page table contains one entry for each physical page frame; each entry consists of process id and virtual page number.
    - Logical address contains the process id, virtual page number, and offset.

- The notion of locality: at any given time a process is only using a few pages or segments.
- Idea: Translation Look-aside Buffer (TLB). A translation buffer is used to store a few of the translation table entries. It's very fast, but only remembers a small number of entries. On each memory reference:
  - First ask TB if it knows about the page. If so, the reference proceeds fast.
  - If TB has no info for page, must go through page and segment tables to get info. Reference takes a long time, but give the info for this page to TB so it will know it for next reference (TB must forget one of its current entries in order to record new one).
- TB Organization: Virtual page number goes in, physical page location comes out. Similar to a cache, usually direct-mapped.
- TB is just a memory with some comparators. Typical size of memory: 2k entries. Each entry holds a virtual page number and the corresponding physical page number. How can memory be organized to find an entry quickly?
  - One possibility: search whole table from start on every reference.
  - A better possibility: restrict the info for any given virtual page to fall in exactly one location in the memory. Then only need to check that one location. E.g. use the low-order bits of the virtual page number as the index into the memory. This is the way real TB's work. Why low-order (instead of high-order) bits?
- Disadvantage of TB scheme: if two pages use the same entry of the memory, only one of them can be remembered at once. If process is referencing both pages at same time, TB doesn't work very well.
- Example: TB with 64 entries. Suppose the following virtual pages are referenced (octal): 621, 2145, 621, 2145, ... 321, 2145, 321, 621.
- In practice, TB's have been extremely successful: 98% hit ratio is typical for 128 entries.

- TBs are a lot like hash tables except simpler (must be to be implemented in hardware). Some hash functions are better than others.
  - Is it better to use low page number bits than high ones?
- Another approach: let any given virtual page use either of *two* slots in the TB. Make memory wider, use two comparators to check both slots at once.
  - This is about as fast as the simple scheme, but a bit more expensive (two comparators instead of one, also have to decide which old entry to replace when bringing in a new entry).
  - Advantage: less likely that there will be conflicts that degrade performance (takes three pages falling in the same place, instead of two).
- Cache terminology
  - Direct mapped.
  - Set associative.
  - Fully associative.
- Must be careful to flush TB during each context swap. Why?
  - Valid bit
- Problem: how does the operating system get information from user memory? E.g. I/O buffers, parameter blocks. Note that the user passes the OS a *virtual address*.
  - In some cases the OS just runs unmapped. Then all it has to do is read the tables and translate user addresses in software. However, addresses that are contiguous in the virtual address space may not be contiguous physically. Thus I/O operations may have to be split up into multiple blocks.