Virtual memory

By
Dr. L.Ranathunga

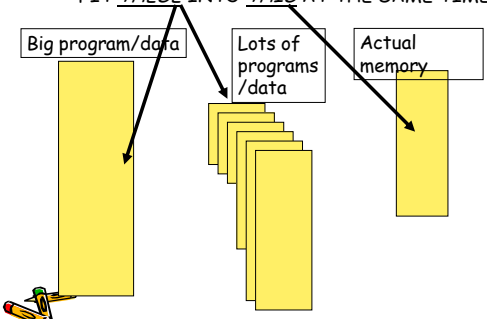
Lecture 4

Virtual memory

- Cache was seen to be used to speed up access to slower memory
- main "semiconductor" memory has finite extent. VIRTUAL MEMORY is introduced to:
  - allow efficient sharing of main memory between multiple processes
  - make the small memory appear larger, the storage used being slower, less costly magnetic media like disk
  - Allows a computer to act as its main memory were much larger than the actual size
    - Programmers can think that they have unlimited memory space

The virtual memory concept

FIT *THESE* INTO *THIS* AT THE SAME TIME



Motivations for virtual memory

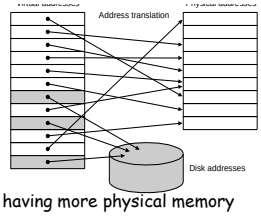
- Collection of programs and data running at once on machine
- Total memory required by all programs might be larger than physical memory available
- Only a fraction of the program/data for a process is required at any time
- Main memory need only contain the active portions (just as with cache)
- program size not known beforehand
- program size can vary dynamically

Virtual memory preparations

- Load each program into its own address space (instructions and data)
- Instead of breaking a single program into parts (called *overlays*) to enable it to fit into physical memory, load it into address space possibly bigger than physical memory
- VIRTUAL MEMORY translates PROGRAM ADDRESS SPACE into PHYSICAL ADDRESS SPACE

Virtual Memory

- Main memory can act as a cache for the secondary storage (disk): used for multiprogramming, not for direct disk access



- Goals:
  - ***Illusion*** of having more physical memory
  - ***Program relocation*** support (relieves programmer burden)
  - ***Protection:*** one program does not read/write data of another

Virtual memory implementation

- Based on two techniques
  - Paged
    - Overall program resides on larger memory
    - Address space divided into virtual pages with equal size
    - Main memory divided in to *page frames* of same size as pages in low level memory
    - Map *virtual page* to *physical page* by using page table
    - TLB (Translation look-aside buffer) is used to keep most recently used page numbers

Virtual memory implementation

  - Segmentation
    - Program is **not** viewed as a single sequence of instruction and data
    - Arranged in to several modules of code, data, and stacks
    - Each module called segment – *segment sector*
    - Different sizes
    - Associated with segment registers
      - Ex: Stack, Data, Program segment registers
  - Or combined

Virtual memory terminology

- ***Page:*** equivalent of "***block***." Fixed size.
- ***Page faults:*** equivalent of "***misses***"
- ***Virtual address:*** equivalent of ***"tag."***
  - ***No cache index equivalent:*** fully associative. VM table index appears because VM uses a different (***page-table***) implementation of fully-associative.
- ***Physical address:*** translated value of virtual address. Can be smaller than virtual address. No equivalent in caches.
- ***Memory mapping (address translation):*** converting virtual to physical addresses. No equivalent in caches.
- ***Valid bit:*** same as in caches
- ***Referenced bit:*** Used to approximate LRU algorithm
- ***Dirty bit:*** Used to optimize write-back.

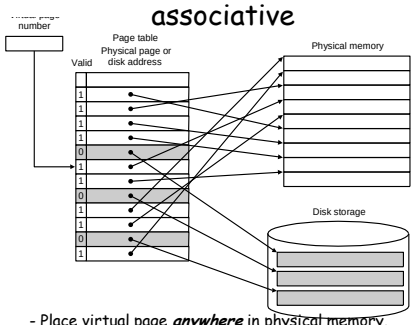
Virtual memory features

- Identical program addresses between different programs are separated
- Program can be *relocated* in physical memory
- Fixed size page blocks (simplifies the job of allocating physical memory)
- Virtual memory separated into virtual *page* and *offset*

Virtual memory design issues

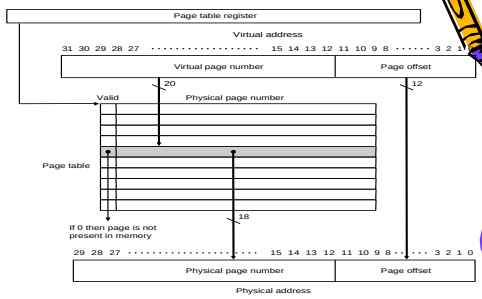
- ***Miss penalty huge:*** Access time of disk = millions of cycles
  - Highest priority: ***minimize misses*** ("page faults")
  - Use ***write-back*** instead of write-through. This is called ***"copy-back"*** in VM. Optimize: disk access only if the page has actually been written (dirty bit).
  - Page fault => Operating system schedules another process!
- ***Protection support:***
  - Break up program's code and data into pages. Add ***"process id"*** to the cache index; use separate tables for different pgms.
  - OS is called via an exception: handles page faults,

Page Tables: implements fully-associative



- Place virtual page ***anywhere*** in physical memory.
- Index page table by the virtual page number to find physical page address (or disk address)

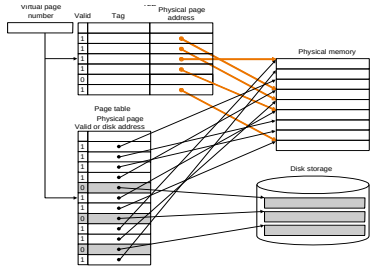
Using page tables to access a word



- Separate page tables per program
- Page table + register values = state of program: called ***"process"***

Making Address Translation Fast

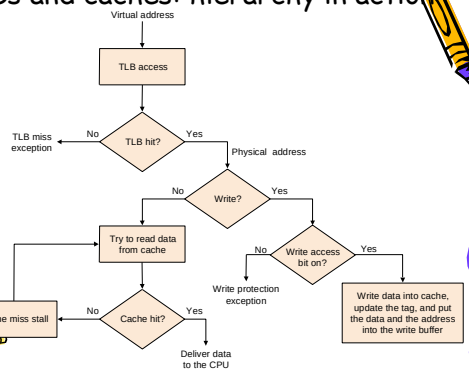
- Why ? Table (memory) access required for every memory access (100% overhead) !
- A hardware cache for address translations: ***translation lookaside buffer (TLB)***. Caches entries from the page table.



Translation Lookaside Buffer: Making address translation fast

- **Page tables are in memory**
- **each memory access requires access to page table entry plus access to desired address => doubles the memory access time**
- **Implement a special address location cache in fast memory, the *translation lookaside buffer***
- **Typical size:32-4096 entries**
- **miss rate 0.01% to 1%**

TLBs and caches: hierarchy in action



- Questions?
- Read Schaum's Outline series Com-Acrhi- Virtual Memory chapter