CS 3204
Operating Systems

Lecture 28

Godmar Back

Announcements

- Project 3 page table design document
  - Vijay will send feedback today
- Project 3 due April 13

VM Design Issues

Continued

VM Access Time & Page Fault Rate

access time = p * memory access time
+ (1-p) * (memory access time + page fault service time)

- Consider expected access time in terms of fraction p of page accesses that don't cause page faults.
- Then 1-p is page fault frequency
- Assume p = 0.99, assume memory is 100ns fast, and page fault servicing takes 10ms – how much slower is your VM system compared to physical memory?
- access time = 99ns + 0.01*(10000100) ns ≈ 100,000ns or 0.1ms
  - Compare to 100ns or 0.0001ms speed ≈ about 1000x slowdown
- Conclusion: even low page fault rates lead to huge slowdown

What is Thrashing

- System accesses a page, evicts another page from its frame, and next access goes to just-evicted page which must be brought in
- Worst case a phenomenon called Thrashing
  - leads to constant swap-out/swap-in
  - 100% disk utilization, but no process makes progress
    - CPU most idle, memory mostly idle

How to avoid Thrashing

- Or contain its effects
- Define: "working set" (1968, Denning)
- Set of pages that a process accessed during some window/period of length T in the past
  - Hope that it'll match the set accessed in the future
- Idea: if we can manage to keep working set in physical memory, thrashing will not occur

Working Set

- Suppose we know or can estimate working set – how could we use it?
- Idea 1: give each process as much memory as determined by size of its WS
- Idea 2: preferably evict frames that hold pages that don't seem to be part of WS
- Idea 3: if WS cannot be allocated, swap out entire process (and exclude from scheduling for a while)
  - "medium term scheduling", "swap-out scheduling"
  - Inactive vs active processes
  - Or don't admit in the first place ("long term scheduling")

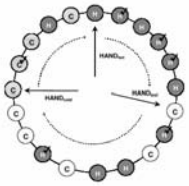
Estimating Working Set

- Compute "idle time" for each page
  - Amount of CPU time process received since last access to page
- On page fault, scan resident pages
  - If referenced, set idle time to 0
  - If not referenced, idle_time += time since last scan
  - If idle_time > T, consider to not be part of working set
- This is known as working set replacement algorithm
  - Variation is WSClock that treats working set a circular list like global clock does, and updates "time of last use" – evicting those where T_last < T_current - T

Page Fault Frequency

- Alternative method of working set estimation
  - PFF: # page faults/instructions executed
  - Pure CPU perspective vs memory perspective provided by WSClock
- Below threshold – can take frames away from process
- Above threshold – assign more frames
- Far above threshold – suspect thrashing & swap out
- Potential drawback: can be slow to adopt to periods of transition

Clock-PRO

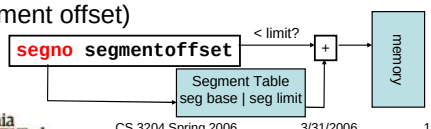


- Clock and algorithms like it try to approximate LRU:
  - When does LRU not work:
  - Sequential scans, large loops
- Alternative:
  - Reuse distance: should replace page with large reuse distance
- Clock-PRO: Idea – extend our reasoning capability by remembering information about pages that were evicted from frames previously
- See [Jiang 2005]

Segmentation

Segmentation

- Historical alternative to paging
- Instead of dividing virtual address space in many small, equal-sized pages, divide into a few, large segments
- Virtual address is then (segment number, segment offset)



Segmentation (2)

- Advantages:
  - little internal fragmentation “segments can be sized just right”
  - easy sharing – can share entire code segment
  - easy protection – only have to set access privileges for segment
  - small number of segments means small segment table sizes
- Disadvantages:
  - external fragmentation (segment requires physically contiguous addresses!)
  - if segment is partially idle, can’t swap out

Segmentation (3)

- Pure segmentation is no longer used
  - (Most) RISC architectures don’t support segmentation at all
  - Other architectures combine segmentation & paging
- Intel x86 started out with segmentation, then added paging
  - Segment number is carried in special set of registers (GS, ES, FS, SS), point to “selectors” kept in descriptor tables
  - Instruction opcode determines with segment is used
  - Today: segmentation unit is practically unused (in most 32-bit OS, including Pintos): all segments start at 0x00000000 and end at 0xFFFFFFFF
  - Do not confuse with Pintos’s code/data segments, which are linear subregions of virtual addresses spanning multiple virtual pages
- Note: superpages are somewhat of a return to segmentation

Combining Segmentation & Paging