

CSE 120 Discussion 5

Chengmo Yang

11/14/07

Topic of Today

- We will simply go through a famous paper by *Ozalp Babauglu* and *William Joy* in 1981:
 - Converting Swap-Based System to do Paging in an Architecture Lacking Page-Referenced Bits

Recap: Paging

- Paging: use fixed sized units in both physical and virtual memory
 - Easy to allocate memory
 - No external fragmentation but internal fragmentation
 - Overhead for holding and looking up page tables
- Page table entries contains following fields:
 - *Dirty* bit: whether or not the page has been written
 - *Reference* bit: whether the page has been accessed
 - *Valid* bit: whether or not the PTE can be used

Page Replacement

- OS usually keeps a pool of free pages so that allocations do not always cause evictions
- But, if no free physical page frames are available, an old page needs to be evicted
- Possible algorithms:
 - Random
 - FIFO: evict the one brought in longest time ago
 - Obvious, simple to implement, but bad performance
 - LRU: evict the page that has not been used for the longest time in the past
 - Good performance, but need to time stamp every reference
 - much too costly

Simulate the Reference Bit

- LRU relies on the reference bit, but the architecture (VAX-11/780) does not support
- How do we simulate reference bit?
 - Use the valid bit to induce page faults
 - Create a *reclaimable* state: valid but unreferenced
 - The fault handler detects this special state

Clock Algorithm

- Simulating reference bits imposes performance overhead
 - Need a replacement algorithm with fewer use of reference bits than LRU
- Arrange all physical page frames in a big circle (clock)
- A clock hand is used to select a good LRU candidate
 - Scan the pages in circular order like a clock
 - If the ref bit is off, it hasn't been used recently, select the page
 - If the ref bit is on, turn it off and go to next page
- Almost same performance as LRU, but quite lower overhead

Global vs. Local Replacement

- On replacement, evict a page of the same processes or other processes?
- They selected global replacement, resulting in a variable-size memory partition for each process
 - Easy to implement global replacement, especially for an architecture without reference bits
 - Processes have highly diverse memory requirement
 - Application has no requirement of guaranteed response times

Clock Triggering

- The clock page replacement policy is only engaged upon a page fault
 - Sometimes bad performance because memory requirement occasionally has high spikes
- Solution: adaptively select more than a single page to be replaced
 - Maintain a pool of free pages
 - Trigger the clock policy whenever the size of the pool drops below a threshold
 - Increase the hand's scan rate linearly as the free page pool size reduces
- The free page pool also keeps pages of processes that are swapped out for potential reuse

Implementation: virtual fork

- The system call *fork()* is expensive because of copying: use Copy on Write (CoW)
 - Create shared mappings of parent pages in child virtual address space
 - Defer copying until the first modification of a page by either process
- They observed that a majority (over 80%) of all forks only serve to establish the context for the new process
 - They creates a new primitive, virtual-fork, to allow the child to run in the address space of the parent
 - The creation of new address space delayed until the subsequent `exec()`

Controlling Multiprogramming Load

- Based on a desire to limit paging overhead
- When the system consumes 10% of CPU time to sample page utilization: a process is moved away from the ready queue
- Which process to be swapped out?
 - The oldest? The largest?
 - They selected the oldest amongst the n largest resident processes
- Processes swapped out are assigned priorities to return to the runnable set based on
 - Their size
 - The amount of time they have swapped out

Pre-paging vs. Page-on-Demand

- Page-on-Demand saves memory, but not efficient for paging intensive programs
- They implemented a simple form of pre-paging
 - Upon a page fault, the faulting page and the next several adjacent pages are read in as a single operation
 - Upon a page out decision, a clusters of dirty adjacent pages are written together
- Where to put pre-paged data (valid but unreferenced)?
 - Not in the clock loop, but at the bottom of free page pool