Page Replacement Algorithms

ECE 595
Feb 20

Y. Charlie Hu
Ning Ding

Virtual Memory review

- What is *Virtual Memory*?
- Virtual address vs. virtual memory?
- How is Virtual Memory typically implemented?
  - What is *demand paging?*
  - It can also be implemented via *demand segmentation*
    - Double drawbacks?
- In Virtual Memory, what will the virtual address space be translated to?
- What is a page fault?
- Page fault vs. TLB miss?
- Does a TLB hit guarantee no page fault?

[week6] Virtual Memory

- Definition: *Virtual memory* permits a process to run with only some of its virtual address space loaded into physical memory
- Virtual address space translated to either
  - Physical memory (small, fast) or
  - Disk (backing store), large but slow
- Objective:
  - To produce the illusion of memory as big as necessary

Virtual Memory

- "To produce the illusion of memory as big as necessary" (yet as fast as physical memory)
  - What makes this possible?

[week6] Separating Policy from Mechanism

A fundamental design principle in Computer Science

Mechanism – tool/implementation to achieve some effect

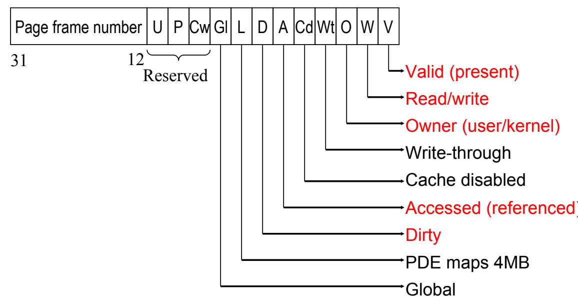
Policy – decisions on what effect should be achieved

Examples:

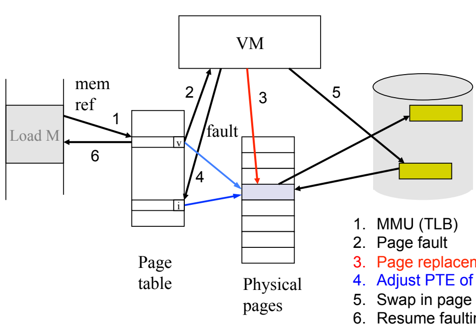
- All users treated equally
- All program instances treated equally
- Preferred users treated better

Separation leads to flexibility

The Mechanism: x86 Page Table Entry



The Mechanism: Page Fault Handling in demand paging



1. MMU (TLB)
2. Page fault
3. Page replacement
4. Adjust PTE of victim pg
5. Swap in page from swap
6. Resume faulting intr

Page out on critical path?

- Page in has to wait till page out is finished
  - Page fault handling time = proc. overhead + ~~2~~ * I/Os

    1

- Solution: page buffering
  - System maintains a pool of free pages
  - When a page fault occurs, victim page chosen as before
  - But desired page paged into a free page right away before victim page paged out
  - Victim page written out when disk is idle

The Policy Issue: Page selection and replacement

- Once the hardware has provided basic capabilities for VM, the OS must make two kinds of decisions
  - Page selection: *when* to bring pages into memory
  - Page replacement: *which* page(s) should be thrown out
    - Try to kick out the least useful page

Page selection (when)

- Prepaging
  - Bring a page into memory before it is referenced
  - Hard to do without a "prophet"
- Request paging
  - Let user say which pages are needed when
  - Users don't always know best
  - And aren't always impartial
- Demand paging
  - Start up process with no pages loaded
  - Load a page when a page fault occurs, i.e., wait till it MUST be in memory
  - Almost all paging systems are demand paging

Page replacement problem

Definition:

- Expect to run with all physical pages in use
- Every "page-in" requires an eviction to swap space
- How to select the victim page?

- Performance goal:
  - Give a sequence of memory accesses, minimize the # of page faults
  - → [Equivalent problem] given a sequence of virtual page references, minimize the # of page faults

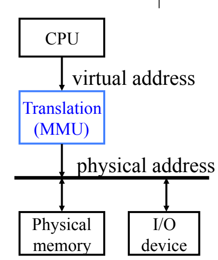
- Intuition: kick out the page that is least useful
- Challenge: how do you determine "least useful"?
  - Even if you know future?

What makes finding the least useful page hard?

- Don't know future
  - Using past to predict future is pretty accurate

- Getting perfect reference stream is hard (why?)

- Can we get imperfect information without high cost?



Optimal or MIN

- Algorithm:
  - Replace the page that won't be used for *the longest time*
- Pros
  - Minimal page faults (can you prove it?)
  - This is an ***off-line*** algorithm for performance analysis
- Cons
  - No ***on-line*** implementation
- Also called Belady's Algorithm

What can we do without extra hardware support?

First-In-First-Out (FIFO)

Recently loaded → | 5 | 3 | 4 | 7 | 9 | 11 | 2 | 1 | 15 | → Page out

- Algorithm
  - At page fault, throw out the oldest page
- Pros?
- Cons?

Predicting future based on past

- "Principle of locality"
  - Recency:
    - Page recently used are likely to be used again in the near future
  - Frequency:
    - Pages frequently used (recently) are likely to be used frequently again in the near future
- Is this temporal or spatial locality?
- Why not spatial locality?

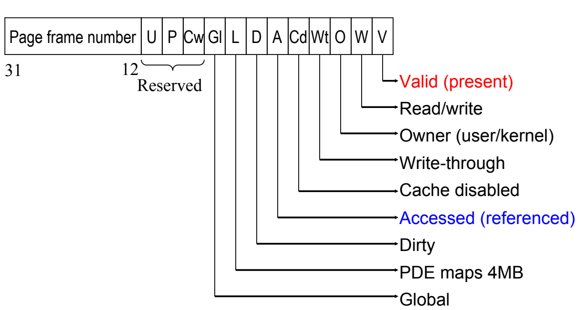
How to record locality in the past?

- Software solution?

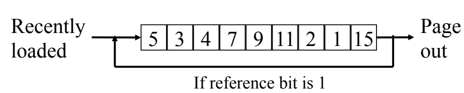
Exploiting locality fast needs hardware support

- Reference bit
  - A hardware bit that is set whenever the page is referenced (read or written)

x86 Page Table Entry

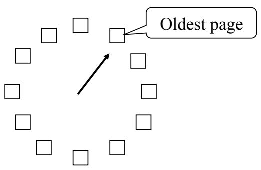


FIFO with 2nd Chance



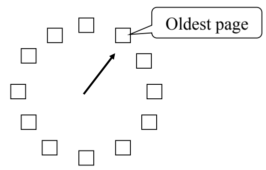
- Algorithm
  - Check the reference-bit of the oldest page
  - If it is 0, then replace it
  - If it is 1, clear the referent-bit, put it to the end of the list, and continue searching
- Pros
  - Potentially fast
  - Frequency → do not replace a “heavily” used page
- Cons
  - The worst case may take a long time

Clock: a simple FIFO with 2nd chance



- FIFO clock algorithm
  - Hand points to the oldest page
  - On a page fault, follow the hand to inspect pages
- Second chance
  - If the reference bit is 1, set it to 0 and advance the hand
  - If the reference bit is 0, use it for replacement
- What is the difference between Clock and the previous one?
  - Mechanism vs. policy?

Clock: a simple FIFO with 2nd chance



- What happens if all reference bits are 1?
- What does it suggest if observing clock hand is sweeping very fast?
- What does it suggest if clock hand is sweeping very slow?

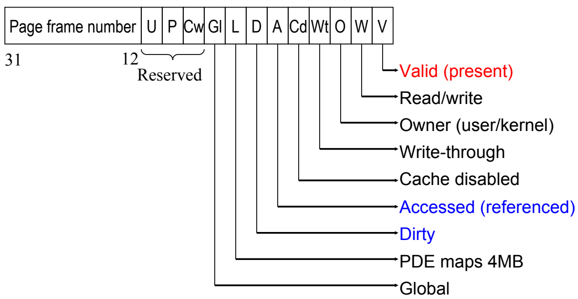
Can we refine the Clock Algorithm?

- Key observation: it is cheaper to pick a "clean" page over a "dirty" page
  - Clean page does not need to be swapped to disk
- Challenge:
  - How to get this info?

Refinement by adding extra hardware support

- Reference bit
  - A hardware bit that is set whenever the page is referenced (read or written)
- Modified bit (dirty bit)
  - A hardware bit that is set whenever the page is written into

x86 Page Table Entry



Enhanced FIFO with 2nd-Chance Algorithm (used in Macintosh VM)

- Same as the basic FIFO with 2nd chance, except that it considers both (reference bit, modified bit)
  - (0,0): neither recently used nor modified (good)
  - (0,1): not recently used but dirty (not as good)
  - (1,0): recently used but clean (not good)
  - (1,1): recently used and dirty (bad)
  - When giving second chance, only clear reference bit
- Pros
  - Avoid write back
- Cons
  - More complicated, worse case scans multiple rounds

Enhanced FIFO with 2nd-Chance Algorithm – implementation

- On page fault, follow hand to inspect pages:
  - Round 1:
    - If bits are (0,0), take it
    - if bits are (0,1), record 1st instance
    - Clear ref bit if (0,1) not found yet
  - At end of round 1, if (0,1) was found, take it
  - If round 1 does not succeed, try 1 more round

FIFO revisited

- Book-keeping when pages are loaded in or kicked out
- No idea or vague idea (2 bits) what happened in between two 2 faults
- Ideally, what more info would we like?

Least Recently Used (LRU)

- Algorithm
  - Replace page that hasn’t been used for the longest time
- Question
  - What hardware mechanisms are required to implement *exact* LRU?

(exact) LRU implemention

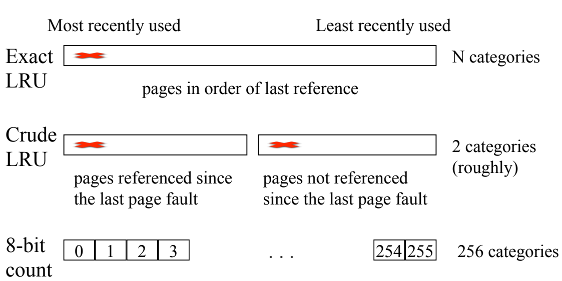
Mostly recently used

5	3	4	7	9	11	2	1	15

Least recently used

- ideally
  - Extend PTE with a time-of-use field
  - Add to CPU a clock, incremented & copied into the PTE for *every* mem reference
  - So we have the time of the last reference to each page
  - At TLB eviction?
  - At finding the LRU page?

Approximate LRU



Keep 8-bit counter for each page in a table in memory

Approximate LRU

Interval 1	Interval 2	Interval 3	Interval 4	Interval 5
00000000	00000000	10000000	01000000	10100000
00000000	10000000	01000000	10100000	01010000
10000000	11000000	11100000	01110000	00111000
00000000	00000000	00000000	10000000	01000000

- Algorithm
  - At regular interval, OS shifts reference bits (in PTE) into counters (and clear reference bits)
  - Replacement: Pick the page with the “smallest counter”
- How many bits are enough?
  - In practice 8 bits are quite good
- Pros: Require one reference bit, small counter/page
- Cons: Require looking at many counters (or sorting)

Page replacement algorithms: Summary

- Optimal
- FIFO
- Random
- FIFO with 2nd chance
- Clock: a simple FIFO with 2nd chance
- Enhanced FIFO with 2nd chance
- Approximate LRU

Reading

- Chapter 9