Cache Types

There are a few ways to organize the cache past the basic organization:

- Fully associative → one row
  - The cached item can go anywhere in the cache row
- Direct mapped → one column
  - Cached item can only go in a row slot/bin corresponding to its memory address
- N-way Set associative → N columns and M rows
  - N items per bin, M bins

Cache Writes

There are generally 2 write behaviors for a cache

- Write-through: changes in caches go immediately to memory
- Write-back: changes set a “dirty bit” and are written on replacement

On top of this, we have to figure out what to do when you try and write to something that isn’t in the cache:

- Write-allocate: load into cache, update in cache write-back style
- No-write-allocate: write immediately to memory

—

Memory management -

MMU

The MMU is responsible for 2 things:

- Address translation: translate logical to physical addresses
- Bounds checking: make sure the requested memory address is within the upper and lower limits of the address space.

Fragmentation

Variable partitioning often causes fragmentation, where holes aren’t big enough, but the total memory you need is still available.

- Holes get consistently smaller and more scattered
- “External Fragmentation” happens when there is enough total memory space to satisfy a request, but not enough contiguous memory space
- Given N allocated blocks, .5N blocks will be lost to fragmentation (50% rule)
- Internal fragmentation is when the memory allocated to the process is bigger than requested (unused memory within a partition)
- Can “compact” everything, but only if the memory is allocated on execution and you can modify the page tables

Paging

- Break physical memory into fixed size boxes called “frames”
- Break logical memory into same sized boxes called “pages”
- Every address is split into 2 parts, page number (p) and a page offset (t)
- Page number is the index in the page table, which contains the base address of that page
- The offset points to a more specific byte.
- Page table is kept in main memory, and a page table base register (PTBR) points to its location in memory
    - This is slow (2 memory accesses are needed to look at 1 byte), but we speed that up using a “translation lookaside buffer” (TLB) which acts as a cache for the page table
    - If it isn’t in the TLB (a “TLB miss”), a memory reference to the page table must be made.
    - Most TLBs allow entries to be “wired down” which prevents them from exiting the TLB
    - TLBs sometimes add address-space identifiers (ASIDs) to the TLB entry, which allows extra security and prevents programs from accessing each other’s address spaces
- Extra protection is added to a page table in the form of a “valid-invalid” bit, which tells us whether a process is allowed to access a given page.
- A page table length register (PTLR) is sometimes included and used to double check that the correct amount of memory is allocated at a given time

Page-fault

On a page fault, the needed page should be loaded into RAM. The following is the process:

1. MMU detects a page is not in memory and sends a page-fault
2. OS saves process state and jumps to the page fault handler
3. Fault handler
    a. If the reference to the page is not in the logical address space, seg fault
    b. If the reference is not in RAM, load the page
    c. Free a frame/find a free frame
    d. Schedule a disk read
4. Returns with interrupt when done reading desired page, OS writes page to frame
5. OS updates page table
6. Restart interrupted instruction

—

Rings of Privilege (Intel)

Commonly, there are 4 rings:

- OS runs in ring 0 (kernel mode)
- rings 1-2 are unused, or sometimes house drivers
- Applications run in ring 3

Another configuration targeted towards VM's

- Hypervisor (which runs the VMs) runs in ring 0
- VMs OS's run in ring 1 or 2
- Applications run in ring 3

—

Thrashing

Repeated page faulting, usually occurs when a process' working set is bigger than the memory it has.

- Under local replacement, P1 thrashes, P2-PN only suffer in disk I/O
- Under global: a process needs more frames, page faults and takes them away from another process which pages, faults..etc etc

Solution
Working set: Find a set of recently accessed pages that is repeated often, allocate to the process the size of the working set. You can set a reference bit and use a timer to evict pages out of the working set.

OR

Page fault measuring: If the page fault frequency is higher than some threshold for a process, allocate it more memory.

Linux Replacement

- Global page replacement
- Clock-like LRU (global)
- Thrashing can only occur if the sum of all working sets exceed the size of physical memory
    - If this happens, terminate a process

Basic Replacement Strategy

There is one page replacement strategy that is worth mentioning, and that is using a dirty/modify bit. Essentially, a dirty bit is set when a page is written to. If the dirty bit is set, the page needs to be written to the disk. If it isn't set, you can just evict it without writing to disk.

On top of this, the following replacement strategies are good to know

- LRU
  - "Virtual time" based
  - Queue
- FIFO
  - If frequently accessed, this is bad
  - Belady's anomaly: more pages → more page faults (contrary to logic)
- OPT
  - "OPTimal": replace the page that will not be referenced for the longest time. Hard to implement
- Reference bit
  - Set a reference bit every time a page is referenced
  - LRU approximation
    - Can record last 8 bits and the LRU is the lowest valued record
    - Second-chance/Clock: Go around in a circle, reset the reference bit to 0, then continue. If a reference bit is still 0, you have a page to evict.
    - Enhanced Clock: Use dirty bit as part of the selection, if reference=0 and dirty=0 you have the best page to replace
- Non-LRU counter replacement
  - Least frequently used
    - Lowest counter
  - Most frequently used
    - Highest counter: in theory the page with the smallest count was probably just brought in and has yet to be used.
  - Expensive to implement

Improving Replacement

A couple of improvement strategies:

- Use a dirty/modify bit to reduce disk writes
- Choose a good page replacement algorithm

- Make the search for the least important page fast
- Page buffering: select a page to evict after the page you want is already in
- Keep a pool of free frames and remember their content (“free” them, but don’t erase them)