VM NOTES

1. Overview

1.1. Motivation. .

- Error handling code rarely used, need not be loaded.
- Heap overallocation.
- Unused features.

1.2. Benefits. .

- Size of RAM no longer a constraint.
- Higher parallelism, since each process takes less memory.
- Less IO
- Memory sharing
  - Dynamically linked libraries: Part of each process' virtual address space, but only one physical (read-only) copy.
  - IPC
  - Forking ?

2. Demand Paging

Demand paging: Pages are loaded only when the process tries to access them.

Procedure:

- When a process tries to access a page not in memory, the paging hardware will notice that the invalid bit is set, causing a trap to the OS, viz. a page fault.
- Since this interrupt can be caused both when the reference is actually invalid and when the reference is valid but the page is not in memory, we check a process-internal table to determine the same.
- If the reference is indeed invalid, the process is terminated. Otherwise, a free frame is procured and the referenced page is copied into memory from disk. The page table and the process-internal table are modified to indicate that the page is now in memory, and the instruction that caused the fault is restarted.

Issues:

- Performance degradation if multiple pages accessed by a single instruction (i.e. multiple page faults) : Empirically shown to be exceedingly infrequent and therefore negligible.
- Irrecoverable states when multiple locations modified:
  - IBM System 360/370 contains the MVC instruction, which can move upto 256 bytes from one location to another (possibly overlapping) location. If either block straddles a page boundary, a page fault may occur after the move is partially done.
  - In addition, if these locations overlap, then the source block may have been modified, in which case we cannot simply restart the instruction.
  - We can solve this by initially accessing both ends of both blocks. If any page fault occurs, it will occur here and thus the pages are guaranteed to be in memory during the move operation.
  - Alternatively, the old values of the overwritten portions of the blocks can be stored in temporary registers, which are restored in the event of a page fault - thus allowing us to restart the instruction.

3. COW

Mainly for fork calls. Child process shares all of its pages with parent process. These pages are marked as COW, and whenever either process attempts to modify it, the OS will acquire a free frame, copy the page onto it and map it to the address space of the modifying process. The process will then modify the copied page and not the original.

4. Page Replacement

Reference String: We evaluate an algorithm by running it on a particular string of memory references and computing the number of page faults. The string of memory references is called a reference string.

4.1. **FIFO.** :

End up enqueueing and dequeueing pages. Weird shit ensues:

Belady's Anomaly: # faults increases with # frames for some reference string.

4.2. **Optimal.** :

Granted perfect prescience, we simply replace the page that will not be used for the longest period of time.

Obviously, such an algorithm cannot exist. It is used only to evaluate the degree of deviation of an algorithm from optimality.

4.3. **LRU.** :

Stack algorithm: Algos for which the set of pages in memory given # frames = n is always a subset of the pages in memory given # frames = n + 1

- Counters: A time-of-last-access is associated with each page, which necessitates a logical clock/counter in the CPU for that express purpose. Each page fault results in a search
- Stack: Linked list, really. When page accessed, place at top. Victim is page at bottom.

LRU policy requires substantial hardware support, with each memory reference needing to update the counters of each page or modify the stack. Without it, we would need to raise an interrupt for each memory access to update the data structures, the overhead would exceed the execution time by a factor of ten. Clearly, this is prohibitive.

4.4. **LRU approximations - Reference bits.** :

Additional Reference Bit algo:

- Maintain a history byte (or two, or three, or five . . .).
- Whenever a process makes a reference to a page, it sets the reference bit to 1.
- In regular intervals (say 100ms), a timer interrupt transfers control to the OS, which copies the reference bit into the highest bit of the history byte, and right-shifts it by 1.
- If we interpret this byte as an unsigned integer, then the page with the least reference number is the LRU page, and it can be replaced.

Second chance / Clock Algorithm:

- No history byte.
- If ref-bit set, reset.
- If ref-bit reset, replace.
- Implemented using a circular queue. When a frame is required, the pointer advances until a page has ref-bit 0, clearing the ref-bits. This page is replaced.
- Worst case, the pointer cycles through the entire queue, clearing all ref-bits. First page in the second cycle replaced.

Enhanced Clock algo: Consider ref-bit and modify/dirty bit as an ordered pair.