Virtual Memory

H. Levy

Virtual Memory

- VM allows a program to run with less than its full VAS in primary memory.
- So, a program can run on a machine with less memory than it "needs".
- Many programs don't need all of their code and data at once (or ever), so this saves space and might permit more programs to shared primary memory.
- The amount of real memory that a program needs to execute can be adjusted dynamically to suit the programs behavior.
- VM requires hardware support, as we saw previously, plus OS management algorithms.

VM

- VM is completely invisible to the programmer.
- It's the responsibility of the OS to load those parts of the program that are needed, and to remove from memory parts that are not needed. How it does this is a *policy* decision.
- "Virtual Memory" is not free -- it must exist somewhere. Those program pages that are not stored in primary memory must be stored on disk.

Demand Paging

- *Demand Paging* refers to a technique where program pages are loaded from disk into memory as they are referenced.
- Each reference to a page not previously touched causes a page fault.
- The fault occurs because the reference found a page table entry with its valid bit off.
- As a result of the page fault, the OS allocates a new page frame and reads the faulted page from the disk.
- When the I/O completes, the OS fills in the PTE, sets its valid bit, and restarts the faulting process.

High Level

- Imagine that when a program starts, it has:
  - no pages in memory
  - a page table with all valid bits off
- The first instruction to be executed faults, loading the first page.
- Instructions fault until the program has enough pages to execute for a while.
- It continues until the next page fault.
- Faults are expensive, so once the program is running they should not occur frequently, assuming the program is "well behaved" (has good locality).

Page Replacement

- When a fault occurs, the OS loads the faulted page from disk into a page of memory.
- At some point, the process has used all of the page frames it is allowed to use.
- When this happens, the OS must *replace* a page for each page faulted in. That is, it must select a page to throw out of primary memory to make room.
- How it does this is determined by the *page replacement algorithm*.
- The goal of the replacement algorithm is to reduce the fault rate by selecting the best victim page to remove.

Optimal Algorithm

- **The optimal algorithm, called Belady's algorithm, has the lowest fault rate for any reference string.**
- **Basic idea: replace the page that will not be used for the longest time in the future.**
- **Basic problem: phone calls to psychics are expensive.**
- **Basic use: gives us an idea of how well any implementable algorithm is doing relative to the best possible algorithm.**

FIFO

- **FIFO is an obvious algorithm and simple to implement.**
- **Basic idea, maintain a list or queue of pages in the order in which they were paged into memory.**
- **On replacement, remove the one brought in the longest time ago.**
- **Why might it work? Maybe the one brought in the longest ago is one we're not using now.**
- **Why it might not work? Maybe it's not. We have no real information to tell us if it's being used or not.**
- **FIFO suffers from "Belady's anomaly" -- the fault rate might actually increase when the algorithm is given more memory -- a bad property.**

Least Recently Used (LRU)

- **Basic idea: we can't look into the future, but let's look at past experience to make a good guess.**
- **LRU: on replacement, remove the page that has not been used for the longest time in the past.**
- **Implementation: to really implement this, we would need to time stamp every reference, or maintain a stack that's updated on every reference. This would be too costly.**
- **So, we can't implement this exactly, but we can try to approximate it.**

Using the Reference Bit

- **Various LRU approximations use the PTE reference bit.**
  - keep a counter for each page
  - at regular intervals, do:
    for every page:
      if ref bit = 0, increment its counter
      if ref bit = 1, zero its counter
      zero the reference bit
  - the counter will thus contain the number of intervals since the last reference to the page.

Second Chance

- **Another scheme is called clock or second chance:**
  - there is a FIFO list of pages
  - to choose a victim, we examine the "oldest page"
    - if its reference bit is zero, it is the victim
    - else, zero the ref. bit, and look at next page
- **In this way, a page stays alive as long as it is referenced in the time it takes to cycle around through all the pages.**

Fixed Space vs. Variable Space

- **In a multiprogramming system, we need a way to allocate memory to the competing processes.**
- **In a fixed-space algorithm each process is given a limit of pages it can use; when it reaches its limit, it starts replacing new faults with its own pages. This is called *local* replacement.**
- **In variable-spaced algorithms, each process can grow or shrink dynamically, displacing other process' pages. This is *global* replacement.**

Working Set Model

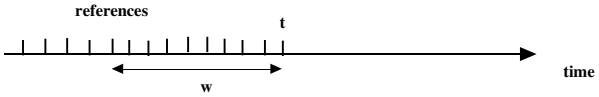
- **Peter Denning defined the working set of a program as a way to model the dynamic *locality* of a program in execution.**
- **Definition:**

  WS(t,w) = {pages i s.t. i was referenced in the interval (t,t-w)}

  t is a time, w is the working set window, a backward looking interval.
- **So, a page is in the WS only if it was referenced in the last w references.**

Working Set Model

- **The working set size is the *number* of pages in the working set; i.e., the number of pages touched in the interval (t, t-w).**
- **The working set size changes with program locality.**



- **For some parameter w, we could keep the working sets of each process in memory.**

WS

- **But, we have two problems:**
  - how do we select w?
  - how do we know when the working set changes?
- **So, working set is not used in practice.**

Page Fault Frequency

- **PFF is a variable space algorithm that uses a more ad hoc approach.**
- **Basic idea:**
  - monitor the fault rate for each process
  - if the fault rate is above a high threshold, give it more memory
  - if the rate is below a low threshold, take away memory