CMPSCI 377 Operating Systems **Fall 2005**

Lecture 13: October 25

Lecturer: Emery Berger *Scribe: Hall, Podrazhansky*

Today:

- Reading/Writing pages
- Swap Space
- Page eviction
- Cost of paging
- Page replacement algorithms

13.1 Demand Paging

13.1.1 General Information

- Virtual memory uses a technique called "paging", the virtual address space is divided into units called pages. The corresponding units in physical memory are called "page frames".
- MMU is responsible for mapping virtual addresses into physical adresses by means of the TLB.
- Demand Paging is the process of bringing pages into memory according to need.
- Allows system to treat RAM as a cache for the disk.
- Relies on locality of reference.
  - **Temporal locality** - one page used multiple times within a period of time.
  - **Spatial locality** - adjacent areas of memory used.
- "Working Set" of pages that have recently been referenced
  - Is calculated by the Operating System.
  - Must fit in RAM or else will cause page thrashing.

13.1.2 When to read/write a page to disk?

- Pages are loaded from disk on demand.

  r
  - Try to read a page, if the page isn't in TLB then fetch from HD and bring to memory (page fault).
  - May need to evict some pages, if there is not enough space in the physical memory.

13.1.3 Pre-paging

- OS guesses which pages will be needed in the near future, loads those pages in advance.
- OS may adjusts this behavior based on the success of its predictions.
- Predictions based on locality of reference.
- May avoid page faults in the future (hides disk latency).
- e.g. After page fault on page x, OS loads a contiguous block of pages, presuming that adjacent pages will be referenced soon after x.
- Smaller pages are good since the latency from the disk is lower.

13.1.4 Super Pages

- Since the TLB can only contain addresses of a limited number of pages, if the pages are made large then the TLB will cover a larger subset of physical memory.
- Helps avoid TLB misses, which are expensive.
- Used mainly in DB systems.

13.1.5 General procedure for demand paging

1. On each reference, check if page is in memory (check valid bit in page table).
2. If not there then trap to OS.
   (a) Select "victim page" to evict (daemon process such as paged does this).
   (b) Invalidate that page in the page table, if it was modified then write to disk.
   (c) Begin loading new page from the hard disk.
   (d) Switch processes since the current one is blocking on I/O (since page fault).
   (e) Interrupt signals the arrival of the page. OS updates the page table, awakens the faulting process.
   (f) When control returns to that process, re-run the instruction that caused the page fault.

Thrashing would occur if all processes are causing page faults, and the maximum number of pages in RAM was smaller than the number of pages requested by all processes. This would happen for example, if the OS didn't signal the arrival of the requested page and re-awaken the faulting process. In which case pages would be evicted before the referencing process got a chance to access them.

13.1.6 Swap Space

The swap space is a subset of the disk that is reserved for storing evicted pages. Under unix-like operating systems the swap space is most often a separate partition running a special lightweight FS (such as tmpfs). Under windows, the swap space is a special file on disk.

- Victim pages can be dropped if they have not been modified, else they must be written into the swap space.

- In worst case (all pages modified) then the size of the swap file must equal the sixe of VM.
- Hence the size of the swap space limits the total amount of VM.
- VM pages can exist in:
  1. RAM
  2. Swap space
  3. File system (if a file is m-mapped)

13.1.7 Cost of Paging

- Worst case scenario: assume page fault on every reference. Not realistic since most programs tend to exhibit locality of reference.
- Average case is important since it describes the behavior of real applications.
- Cost of page faults are high, hence want to minimize number of page swaps over the execution of a process.

13.1.8 Effective Access Time

Let p be probability of page fault.

Let t_m be memory access time (≈200ns).

Let t_d be time to load page from disk (≈25000000ns).

Then effective access time e is defined as:

$$e = \underbrace{(1-p)\cdot t_m}_{time\ to\ load\ from\ memory} + \underbrace{p\cdot t_f}_{time\ to\ service\ page\ fault}$$

Hence $p \approx 0$ is required for good performance.

13.1.9 Competitive Ratio

An algorithm is described as n-competitive if it cannot take longer than n times the optimal case. In this case n is called the competitive ratio.

13.1.10 Replacement Algorithms

MIN/OPT Easy to describe but impossible to implement. Simply, evict the page that will be referenced farthest into the future. The reason we cannot implement this is that it requires knowledge of the future behavior of the program.

Random Evict a random page.

- Works quite well, but not used in practise.
- Takes no advantage of locality.

LRU Evict the least-recently used page.

- Works well if the past is identical to the future (in which case LRU = OPT).
- k-competitive, where k is the number of pages. It turns out this is the best possible for a non-deterministic algorithm.
- A variant of LRU is used in all operating systems.

MRU Evict the page that has been used most recently

- Works well in the worst case of LRU (in which the number of referenced pages is one greater than the total amount allowed in RAM, and processes access these pages in a sequential manner).
- In this case LRU will cause a page fault on every reference, whereas MRU will cause fewer page faults.

FIFO Evict pages in the same order that they were loaded.

- Suffers from Belady's Anomaly, where increasing the number of page frames leads to an increase in the frequency of page faults.

In general we want an adaptive algorithm that incorporates the advantages of these various algorithms.