Lecture 13: Page Replacement

Adam Hawley

February 28, 2019

Contents

1 Introduction

1.1 When does Page Replacement Occur?

When a page fault occurs, we need to bring the desired page into memory. However, sometimes there are no free frames for the page to fill. This is when we must replace one of the pre-existing pages to with the new one.

1.2 Outline of Page Replacement

Page replacement is done by finding some page in memory, but not really in use and paging it out. We use an algorithm to decide which frame to free. The performance of this algorithm is based on trying to achieve the minimum number of page faults possible. When replacing pages it should be noted that the same page can be brought into memory several times.

1.3 Optimisation of Page Replacement

Use a **modify*(*dirty**)*bit* to reduce overhead of page transfers (then only modified pages are written back to the disk and we are not rewriting unchanged pages). Page replacement completes separation between logical memory and physical memory (large virtual memory can be provided on a smaller physical memory).

2 Replacement Mechanism

1. Find the location of the desired page on disk
2. Find a free frame
   - if there is a free frame, use it
   - if there is no free frame, use a page replacement algorithm
     - select a **victim frame**
     - write victim frame to disk if it is *dirty*
3. Bring the desired page into the (newly) free frame (i.e. update the page and frame tables).
4. Continue the process by restarting the instruction that caused the trap.

3 Page Replacement Algorithms

Remember: page-replacement algorithms aim for the lowest page-fault rate on both first access and re-access. Algorithms are evaluated by running it on a particular string of memory references and computing the number of page faults on that string. The string is just page numbers, not full addresses. Repeated access to the same page does not cause a page fault. Results depend on the number of frames available.

3.1 FIFO

Just use a FIFO queue to keep track ages of pages. One would expect that the more frames are allocated to a process, the fewer page faults. In FIFO this is not necessarily true (see Belady's anomaly).

3.2 Optimal

Best possible replacement policy: replace page that will not be used for longest period of time. If we can see into the future (like when we have a reference string) then we can see which page will not be used for the longest period of time. Obviously this is really only theoretical so it is often used mainly as an upper-bound in comparative evaluation.

3.3 Least Recently Used (LRU)

Use past knowledge rather than future: replace page that has not been used for the longest period of time. Obviously, this involves keeping track of time of last use for each page. Using the lecture reference string, it had 12 faults which was better than FIFO but worse than OPT.

It is generally a good approach and widely used but there are implementation issues.

3.4 LRU Approximation Algorithms

These are obviously similar to LRU but without complex timestamping.

- **Reference Bit** :
  - With each page associate a hardware-provided bit; initially 0
  - When a page is referenced the associated bit is set to 1
  - Replace any page with reference bit = 0 (if one exists). However, we do not know the order

- **Second-Chance Algorithm**:
  - FIFO scheme, plus hardware-provided reference bit
  - If page to be replaced has
    - Reference bit = 0 → Replace it
    - Reference bit = 1 then:
      - Set reference bit 0, leave page in memory
      - Replace next page, subject to same rules

4 Page and Frame Locking

The OS may wish some pages to remain in physical memory frames (e.g parts of the OS code itself or I/O buffers).

4.1 Frame Locking

If a frame is locked, it may not be replaced. It requires a lock bit with each frame (usually supported by the OS rather than hardware).

5 Frame Allocation

Frame allocation algorithms determine how many frames to give each process and which frames to replace in case of scarcity. Each process needs a **minimum** number of frames. The **maximum** is the total frames in the system. There are many allocation schemes such as *equal allocation*, *size-proportional allocation*, *priority allocation*, etc. It is important to keep a free frame buffer pool.

5.1 Global Replacement

Global Replacement Process selects a replacement frame from the set of all frames.

- One process can take a frame from another
- In priority based allocation of frames, this may enable a high priority process to increase its allocated frames by taking a frame froom a low priority process
- Page fault behaviour of a process becomes dependent on the behaviour of other processes
- Greater overall throughput, so more common (e.g. Linux)

5.2 Local Replacement

Local Replacement Each process selects from only its own set of allocated frames.

- More consistent per-process performance
- If a process does not have sufficient number of frames allocated to it, the process will suffer many page faults (thrashing)
- Possibly underutilised memory

6 Thrashing

Thrashing When a process is busier swapping pages in and out then executing itself.

If a process does not have *enough* pages, the page-fault rate is very high. Insufficient frames lead to page faults. Pages are swapped out and then needed again, so page fault so another page is swapped out which is needed again...

This can lead to:

- Low CPU utilisation
- Operating system thinking that it needs to increase the degree of multiprogramming
- Another process added to the system

6.1 Solving Thrashing

6.1.1 Locality Model

Demand paging works well using a **locality model**. This is when processes migrate from one locality to another (localities may overlap). Thrashing occurs when:

$$\sum \text{size of locality} > \text{total memory size} \tag{1}$$

The effects of thrashing can be limited by:

- Local page replacement
- Priority page replacement — replace a page from a process with the lowest priority.

6.1.2 Working-Set Model

Define Δ to be a working-set window. Then analyse the most recent Δ page references. If a page is in use, it is in the working set but if it is no longer used then it will drop from the working set Δ time units after its last reference.

$\mathrm{WSS_i}$(working set of Process $\mathrm{P_i}$) is defined to be the total number of pages referenced in the window Δ. $\mathrm{WSS_i}$ tries to approximate the size of the locality of process $\mathrm{P_i}$:

- If Δ is too small will not encompass entire locality
- If Δ is too large will encompass several localities
- If $\Delta = \infty \rightarrow$ Will encompass the whole program

$$D = \sum_{i=0}^{n} WSS_i \tag{2}$$

If $D > m \rightarrow$ Thrashing (where m is the total number of frames).

Policy: if $D > m$, then suspend or swap out one of the processes.

6.1.3 Page-Fault Frequency

This is a more direct approach than WSS. It is done by establishing an *acceptable* **page-fault frequency** (PFF) rate and use local replacement policy. If the actual rate is too low, process loses a frame and if the actual rate is too high, the process gains a frame.