Virtual Memory - II

Tevfik Koşar

University at Buffalo
October 25th, 2012

Roadmap

- Virtual Memory
  - Page Replacement Algorithms
    - Optimal Algorithm
    - Least Recently Used (LRU)
    - LRU Approximations
    - Counting Algorithms
  - Allocation Policies
  - Thrashing
  - Working Set Model
FIFO

- FIFO is obvious, and simple to implement
  - when you page in something, put it on the tail of a list
  - evict page at the head of the list
- Why might this be good?
  - maybe the one brought in longest ago is not being used
- Why might this be bad?
  - then again, maybe it is being used
  - have absolutely no information either way
- In fact, FIFO’s performance is typically lousy
- In addition, FIFO suffers from Belady’s Anomaly
  - there are reference strings for which the fault rate increases
    when the process is given more physical memory

Optimal Algorithm

- Replace page that will not be used for the longest time in future
- 4 frames example
  1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5
Optimal Algorithm

- Replace page that will not be used for longest period of time
- 4 frames example
  1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

  1 4
  2
  3
  4 5

  6 page faults

- How would you know this in advance?

Optimal (Belady’s) Algorithm

- Provably optimal: lowest fault rate (remember SJF?)
  - evict the page that won’t be used for the longest time in future
  - problem: impossible to predict the future
- Why is Belady’s Optimal algorithm useful?
  - as a yardstick to compare other algorithms to optimal
    - if Belady’s isn’t much better than yours, yours is pretty good
      - how could you do this comparison?
- Is there a best practical algorithm?
  - no; depends on workload
- Is there a worst algorithm?
  - no, but random replacement does pretty badly
    - there are some other situations where OS’s use near-random algorithms quite effectively!
Least Recently Used (LRU)

- Reference string: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5
Least Recently Used (LRU)

- LRU uses reference information to make a more informed replacement decision
  - idea: past experience gives us a guess of future behavior
  - on replacement, evict the page that hasn’t been used for the longest amount of time
    - LRU looks at the past, Belady’s wants to look at future
    - How is LRU different from FIFO?

- Implementation
  - to be perfect, must grab a timestamp on every memory reference, then order or search based on the timestamps …
  - way too costly in memory bandwidth, algorithm execution time, etc.
  - so, we need a cheap approximation …

LRU Implementations

- Stack implementation - keep a stack of page numbers in a double link form:
  - Page referenced:
    - move it to the top
    - requires 6 pointers to be changed
  - No search for replacement

```
reference string
  4 7 0 7 1 0 1 2 1 2 7 1 2

stack before
  a

stack after
  b
```
LRU Approximation Algorithms

- **Reference bit**
  - With each page associate a bit, initially = 0
  - When page is referenced bit set to 1
  - Replace the one which is 0 (if one exists). We do not know the order, however.

- **Additional Reference bits**
  - 1 byte for each page: eg. 00110011
  - Shift right at each time interval

LRU Clock Algorithm

- **AKA Not Recently Used (NRU) or Second Chance**
  - replace page that is “old enough”
  - logically, arrange all physical page frames in a big circle (clock)
    - just a circular linked list
  - a “clock hand” is used to select a good LRU candidate
    - sweep through the pages in circular order like a clock
    - if ref bit is off, it hasn't been used recently, we have a victim
      - so, what is minimum “age” if ref bit is off?
    - if the ref bit is on, turn it off and go to next page
  - arm moves quickly when pages are needed
  - low overhead if have plenty of memory
  - if memory is large, “accuracy” of information degrades
    - add more hands to fix
Second-Chance (clock) Page-Replacement Algorithm

- Keep a counter of the number of references that have been made to each page
- **LFU Algorithm**: replaces page with smallest count
- **MFU Algorithm**: based on the argument that the page with the smallest count was probably just brought in and has yet to be used
Allocation of Frames

- Each process needs *minimum* number of pages

- Two major allocation schemes
  - fixed allocation
  - priority allocation

Fixed Allocation

- **Equal allocation** - For example, if there are 100 frames and 5 processes, give each process 20 frames.
- **Proportional allocation** - Allocate according to the size of process

\[
\begin{align*}
    s_i &= \text{size of process } p_i \\
    S &= \sum s_i \\
    m &= \text{total number of frames} \\
    a_i &= \text{allocation for } p_i = \frac{s_i}{S} \times m
\end{align*}
\]

\[
\begin{align*}
    m &= 64 \\
    s_i &= 10 \\
    s_2 &= 127 \\
    a_1 &= \frac{10}{137} \times 64 \approx 5 \\
    a_2 &= \frac{127}{137} \times 64 \approx 59
\end{align*}
\]
Priority Allocation

• Use a proportional allocation scheme using priorities rather than size

• If process $P_i$ generates a page fault,
  - select for replacement one of its frames
  - select for replacement a frame from a process with lower priority number

Global vs. Local Allocation

• **Global replacement** - process selects a replacement frame from the set of all frames; one process can take a frame from another

• **Local replacement** - each process selects from only its own set of allocated frames
Thrashing

- If a process does not have “enough” frames, the page-fault rate is very high. This leads to:
  - Replacement of active pages which will be needed soon again
  
  ⇒ **Thrashing** = a process is busy swapping pages in and out

- Which will in turn cause:
  - low CPU utilization
  - operating system thinks that it needs to increase the degree of multiprogramming
  - another process added to the system

Thrashing (Cont.)
Locality in a Memory-Reference Pattern

\[ \Delta \equiv \text{working-set window} \equiv \text{a fixed number of page references} \]
Example: 10,000 instruction

\[ WSS_i (\text{working set of Process } P_i) = \text{total number of pages referenced in the most recent } \Delta \text{ (varies in time)} \]
- if \( \Delta \) too small will not encompass entire locality
- if \( \Delta \) too large will encompass several localities
- if \( \Delta = \infty \) \( \Rightarrow \) will encompass entire program

\[ D = \sum WSS_i \equiv \text{total demand frames} \]
- if \( D > m \) \( \Rightarrow \) Thrashing
- Policy if \( D > m \), then suspend one of the processes
Exercise

- Consider the following page-reference string:

1, 2, 3, 4, 4, 3, 2, 1, 5, 6, 2, 1, 2, 3, 7, 8, 3, 2, 1, 5

Assuming 4 memory frames and LFU, LRU, or Optimal page replacement algorithms, how many page faults, page hits, and page replacements would occur? Show your page assignments to frames.
Summary

• Virtual Memory
  - Page Replacement Algorithms
    - Optimal Algorithm
    - Least Recently Used (LRU)
    - LRU Approximations
    - Counting Algorithms
  - Allocation Policies
  - Thrashing
  - Working Set Model

• Next Lecture: Project 2 Discussion
• Reading Assignment: Chapter 9 from Silberschatz.

Acknowledgements

• “Operating Systems Concepts” book and supplementary material by A. Silberschatz, P. Galvin and G. Gagne

• “Operating Systems: Internals and Design Principles” book and supplementary material by W. Stallings

• “Modern Operating Systems” book and supplementary material by A. Tanenbaum

• R. Doursat and M. Yuksel from UNR
• Gribble, Lazowska, Levy, and Zahorjan from UW