Virtual Memory

Lecture 19

March 28 2017
Agenda

- Demand Paging
- Page Fault
- Page Replacement
Virtual Memory: Main Idea

- Processes use a virtual (logical) address space.
- Every process has its own address space.
- The virtual address space can be larger than physical memory.
  - Only part of the virtual address space is mapped to physical memory at any time.
- Parts of processes' memory content is on disk.
- Hardware & OS collaborate to move memory contents to and from disk (swapping).
Demand Paging

- Bring a page into memory only when it is needed
  - Why?
    - Less I/O needed
      - If a process of 10 pages actually uses only half of them, then demand paging saves the I/O necessary to load the 5 pages not used.
    - Less memory needed
    - Faster response
    - More multiprogramming is possible
Demand Paging

- We need hardware support to distinguish between pages that are in memory and the pages that are on disk.
- A valid-invalid bit is part of each page entry.
  - When the bit is set to “valid” the associated page is in memory.
  - If the bit is set to “invalid” the page is on the disk.
Demand Paging

- The valid-invalid bit for 1 is set to “i” since the page is not in the physical memory.
- The valid-invalid bit for 0 is “v” since the page is in memory.
Page Fault

- What happens if a process tries to access a page that was not brought into memory?
- Access to a page marked invalid causes a page fault
- The paging hardware, in translating the address through the page table, will notice that the invalid bit is set causing a trap to the operating system.
Steps in Handling a Page Fault

1. Reference
2. Trap
3. Page is on backing store
4. Bring in missing page
5. Reset page table
6. Restart instruction
Challenge: Performance

- Page Fault Rate $0 \leq p \leq 1.0$
  - if $p = 0$ no page faults
  - if $p = 1$, every reference is a fault

- Let $p$ be the probability of page fault:
  - Access time = $(1-p) \times$ memory time + $p \times$ page fault time
  - Assuming: memory time = 200ns, page fault time = 8 millisecond, $p = 0.1$
    - Access time = $99.9\% \times 200 + 0.1\% \times 8000000 = 8200$
    - Access time is directly proportional to the probability of a page fault
  - If one access out of 1000 causes a page fault the effective access time is 8.2 microseconds

- Need to keep the page fault rate small!
Page Replacement

- So why allow demand paging?
- If a process of 10 pages actually uses only half of them, the demand paging saves the I/O necessary to load the 5 pages that are never used.
- This allows us to increase the level of multiprogramming.
Page Replacement

- Let’s assume that our physical memory consists of 40 frames.
- We have 8 processes with 10 pages. That is 80 pages.
  - Obviously 80 pages is more than 40 frames.
  - But if a process is only using half of its pages is this really a problem.
- But there is a reason why there are 10 pages.
  - The process may need them.
Page Replacement

- What do we do when a process needs a frame and there isn't one free?
- Essentially we choose a frame and free it of the page that is currently residing on it
Page Replacement

- A page replacement algorithm describes which frame becomes a victim.
- Designing an appropriate algorithm is important since disk I/O is expensive
- Slight improvements in algorithms yield large gains in system performance
We will discuss several algorithms

The examples assume:

- 3 frames
- Reference string:
  \[7,0,1,2,0,3,0,4,2,3,0,3,2,1,2,0,1,7,0,1\]
- Each of the numbers above refers to a specific page number
Page Replacement Algorithms

- Optimal Page Replacement Algorithm
- Least Recently Used (LRU)
- Least frequently used (LFU)
- Most frequently used (MFU)

Most OS’s use LRU
Optimal Page Replacement Algorithm

- Replace page needed at the farthest point in future i.e. replace the page that will not be used for the longest period of time

- This should have the lowest page fault rate
Optimal Page Replacement

- Optimal is easy to describe but impossible to implement
- At the time of the page fault, the OS has no way of knowing when each of the pages will be referenced next
FIFO Page Replacement Algorithm

- Maintain a linked list of all pages
  - Each page is associated with the time when that page was brought into memory
- Page chosen to be replaced is the oldest page
- Implementation: FIFO queue
  - A variable head points to the oldest page
  - A variable tail points to the newest page brought in
FIFO Page Replacement

reference string

7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

page frames

Note: The read arrow is pointing to the oldest page
FIFO Page Replacement

- **Advantages**
  - Easy to understand and program

- **Disadvantage**
  - Performance is not always good
  - The page replaced may be an initialization module that was used a long time ago and is no longer needed, but on the other hand...
    - The page may contain a heavily used variable that was initialized early and is in constant use
LRU Page Replacement

- Can we use the recent past as an approximation of the near future?
  - This means replace the page that has not been used for the longest period of time.
  - This approach is the Least-Recently-Used (LRU) algorithm.
LRU Replacement Algorithm

- LRU replacement associates with each page the time of that page’s last use
- When a page must be replaced, LRU chooses the page that has not been used for the longest period of time.
LRU Page Replacement

reference string

7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

7 → 7 → 7 → 2 → 2 → 4 → 4 → 4 → 0 → 1 → 1 → 1

0 → 0 → 0 → 0 → 0 → 3 → 3 → 3 → 3 → 0 → 0

1 → 1 → 1 → 1 → 1 → 1

page frames

7 2 3 3 2 2 2 7
LRU Page Replacement

- LRU is often used and is considered to be good

- **Challenge: Implementing LRU**
  - Lots of overhead
  - Operating systems often use an approximation algorithm
Other Algorithms

- Least frequently used (LFU)
- Most frequently used (MFU)

- Most OS's use LRU
Summary

- We have studied the need for page replacement algorithms
- Several algorithms have been discussed including:
  - Optimal
  - FIFO
  - LRU