Outline

- Announcements
  - Write-up 2 has been graded (finally!)
  - Lab 4 due today, e-mail directory information to the TA
  - Out today
    - Lab 5 (I/O): due April 10th
    - Write-up 3 (Virtual Machines): due April 12th
  - Questions?

- Virtual memory
  - Page replacement algorithms (contd.)
  - Frame allocation algorithms
  - Thrashing
  - Process working sets

[Silberschatz/Galvin: Sections 9.4 - 9.10]

---

Page Replacement: FIFO

- Evict the page that brought in the earliest

- **Pro:** Simple to implement
  - OS can maintain a FIFO queue and evict the one at the beginning

- **Con:** Assumes that a page brought in a long time ago has low utility
  - Obviously not true in general (e.g., much-used library routines)

How does FIFO perform?

- Consider reference string (length 12)
  1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

  (with 3 frames) 1 2 3 4 1 2 (9)
  (with 4 frames) 1 2 3 4 1 2 (10)

Belady’s anomaly

Algorithms that don’t exhibit this behavior are known as stack algorithms

---

Page Replacement: What is the Best Algorithm?

- For read-only pages (discounting clean-page preference issues), it can be proven that the optimal algorithm (OPT) is
  - Replace the page whose next use is the farthest

  1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

  (with 3 frames) 1 3 4 1 2 (7)
  (with 4 frames) 1 3 4 1 (6)

- Optimality stems from the fact that
  - The page replaced will cause a page fault far away
  - Any other page will cause a fault at least as quickly

- How do you prove that OPT does not suffer from Belady’s anomaly?
Page Replacement: LRU

- Problem with OPT: Clairvoyance is generally not possible
  - But sometimes possible to analyze deterministic algorithms
  - In any case, a good baseline to compare other policies against

- LRU (least recently used) is a good approximation of OPT
  - Assumes that recent past behavior is indicative of near future behavior
    - A phenomenon called locality which is exploited repeatedly in virtual memory

- Main idea: Evict the page that has not been used for the longest time

  1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5
  (with 3 frames) \(\uparrow\uparrow\uparrow\uparrow \underbrace{1}_2 \underbrace{3}_4 \underbrace{5}_1 \underbrace{1}_2\) (10) versus FIFO (9) and OPT (7)
  (with 4 frames) \(\uparrow\uparrow\uparrow\uparrow \underbrace{1}_2 \underbrace{3}_4 \underbrace{5}_1 \underbrace{1}_2\) (8) versus FIFO (10) and OPT (6)

Page Replacement: Implementing LRU

- Counters
  - Attach to each frame, a counter that serves as a logical clock
    - Updated by the hardware on every reference
  - Page replacement: choose page in frame with smallest counter value
    - Counter is reset when a new page is loaded
  - Problems: Elaborate hardware, Search time
    - Largely of theoretical value

- Stack
  - Maintain a stack of page numbers
    - On each access, hardware moves the page\# to the top of the stack
  - Page replacement: the LRU page is at the bottom of the stack
  - Typical implementation: microcoded doubly linked list
    - Used by one of the earlier CDC machines
  - Still too high a hardware cost

Page Replacement: LRU (contd.)

- LRU works reasonably well in simulations
  - “real” program traces exhibit locality
    - but, some pathological access patterns

  1, 2, 3, 4, 1, 2, 3, 4, 1, 2, 3, 4, ...  
  (with 3 frames) \(\uparrow\uparrow\uparrow\uparrow \underbrace{1}_2 \underbrace{3}_4 \underbrace{1}_2\)

- Main problem with LRU: How does one maintain an active “history” of page usage?
  - Counters
  - Stack

Page Replacement: LRU Approximations

- Page reference bit
  - Stored with the frame containing the page
    - Bit is set whenever the page is accessed
    - Periodically, the OS (or hardware) resets all reference bits
  - Page replacement: Choose an unreferenced page

- Additional reference bits
  - For each page \(p\), OS maintains an n-bit last-reference-time \(lrt[p]\)
    - Periodically, OS (or hardware)
      - Shifts right \(lrt[p]\), adds current reference bit as MSB, and resets reference bit
    - Note that the additional bits can be maintained in software
  - Page selected is the one with the lowest \(lrt\)
    \(lrt[p1] = 11000100\) has been used more recently than \(lrt[p2] = 01110111\)
Page Replacement: LRU Approximations (contd.)

- Second-chance Algorithm (also known as Clock)
- Only uses single-bit page reference information
- Maintains a list of frames as a circular list
- Maintains a pointer into the list
- Replacement: search for a page with reference bit zero
  - If there is a page with reference bit 1
    - Set the bit to 0, and continue searching
  - Each page gets a second chance before being evicted

- Enhanced second-chance algorithm
  - Make decision using two bits: page reference and page modify
    - (0, 0): neither recently used nor modified: best candidate
    - (0, 1): not recently used but modified
    - (1, 0): recently used, but not modified
    - (1, 1): recently used and modified: worst candidate
  - Used in the Macintosh

Page Replacement: Performance Enhancements

- Maintain a pool of free frames
  - Buffered (delayed) writes
    - Frame allocation precedes deallocation
    - Allocate immediately from pool, replace later
  - Rapid frame and page reclaim
    - Keep track of which page was in which frame
    - Reclaim pages from free pool if referenced before re-use
      - Can be used as an enhancement to FIFO schemes
      - Used in VAX/VMS
  - Background updates of writes to secondary store
    - Whenever the disk update mechanism is free
      - Write out a page whose modified bit is set and then reset the bit
  - Delayed write (copy-on-write)
    - Create a lazy copy (on the first write): defer allocation
    - Used to optimize Unix fork

Frame Allocation

- So far: how OS can manage the frames allocated to a process
  Control is also possible in how we allocate frames to processes

- Naïve single-user system
  - Keep a list of free frames
  - Allocate from this list
  - Use eviction (replacement) algorithm when list exhausted

- Problem: Multiprogrammed systems
  - How many frames for each process?
  - Performance varies dramatically with the number of frames
    - E.g., matrix multiplication (A := B*C)
      - Square matrices of size 64x64, 4-byte words
      - A page size of 4096 bytes (each matrix can fit into 4 pages)
    - With 12 page frames: get only 12 page faults
    - With 5, get >4096
    - With 1, get >250000

Frame Allocation: Two Critical Questions

- How many frames to assign to each process?
  - Fixed
  - Variable (from a global pool)
  - Is there a minimum (critical) number of frames that must be allocated?

- How are they assigned?
  - When a new process needs more frames, do we
    - Take away uniformly from a given process
    - Or do we assign frames back and forth between processes?
Frame Allocation Algorithms: How Many?

- **Static approach**
  - Allocate once and stays fixed during the process’ lifetime

- **Uniform approach**
  - Given $m$ frames and $n$ processes, allocate $m/n$ per process
  - Very simple, but can lead to a lot of wasted frame usage since the size of the process’ virtual space is not considered

- **Proportional allocation**
  - Let $S$ be the sum of all the virtual memory “needs” across processes where $s_i$ is the virtual memory need of process $i$
    - Allocate $(s_i / S) * m$ frames to process $i$
  - Problems:
    - Does not distinguish between process priorities
    - Does not distinguish between process behaviors

Frame Allocation: Scope of Replacement

- How are additional requests for frame allocation satisfied?
  - **Local replacement**
    - New frames are allocated to pages from a fixed set associated with the process
    - Number does not change with time
  - **Global replacement**
    - New frames can be selected from a variable pool that is shared by the whole system
    - The performance due to page faults of any one process is dependent on the behavior and demands of others using this approach

Frame Allocation: Constraints on Number of Frames

- **Hardware**: Determined by page fault induced instruction restarts
  - Need frames to store all the needs of a single instruction
  - Could be more than one page
    - CISC instruction may straddle page boundary
    - Data may straddle page boundary
    - Indirect addressing may straddle page boundary

- **Software**: Clearly there is a constraint
  - If a process gets too few frames, it spends all its time demand paging
  - This phenomenon is called **thrashing**
  - Formally,
    - Over any time window and summed over all processes, let $T$ be the time spent by the process in computing and $P$ be the time spent in page faults
    - A characterization of thrashing in a time window is when $T < P$
  - We can define it, but can we do anything to reduce it?

Thrashing

- Not enough memory for all processes
  - Processes spend their time page-faulting
Dealing with Thrashing

- **The idea**
  - Exploit the fact that programs demonstrate temporally localized behavior in terms of their memory access
  - Over each "time window"
    - Monitor the behavior of active processes
    - Estimate how many pages each process needs
    - Adjust the frame allocation (and multiprogramming level) accordingly

- **The working set of a process** over time window $W$ is the set of pages it accesses within $W$
  - Use of the working set
    - Choose a parameter $W$
    - Over a time window of size $W$, estimate the size $|w_i|$ of the working set of each process $i$
    - Do not activate more processes if the current sum of the $|w_i|$ together with the set $|w_j|$ of the new process $j$ exceeds available memory

Working Set Model

- **Examine the most recent $\Delta$ page references**
  - This defines the process working set
    - If a page is in active use it will be in the process working set
    - If it is no longer being used, it will drop from the working set $\Delta$ units after its last reference

- Working set strategy prevents thrashing while keeping the degree of multiprogramming as high as possible

- **Difficulty: Keeping track of the working set**
  - Approximated using a fixed interval timer interrupt and a reference bit
    - Periodically, write out reference bits into a structure
    - 0: not referenced, else, was referenced in the last interval

Page-Fault Frequency

- More direct approach for controlling thrashing

- **Keep track of the page-fault rate of a process**
  - When too high: process needs more frames
  - When too low: process might have too many frames
  - Keep each process’ page-fault rate within a upper and a lower bound

![Graph showing the relationship between page-fault rate and number of frames](image)

Demand Paging: Other Issues

- **I/O interlocking**
  - Need to ensure that I/O does not refer to pages that are swapped out
  - Two common solutions
    - Use kernel buffers to receive I/O responses
    - "pin-down" (or lock) the concerned pages

- **Prepaging (warm start)**
  - Initial working set is brought in as a block
  - Advantageous when the cost of bringing in a block is lower than that of generating page faults to bring in the subset of the working set that is used

- **Choice of page size**
  - Large pages: smaller tables, smaller I/O costs, fewer page faults
  - Small pages: less external fragmentation, less overall I/O
  - Trend towards larger page sizes
    - Limiting factor is reducing the number of page faults (disks are slow)
Next Lecture

- File system interface
  - File concept
  - Access methods
  - Directory structure
  - Protection
  - Consistency semantics

- Reading
  - Silberschatz and Galvin, Chapter 10