Operating Systems ECE344

Lecture 11: Page Replacement

Ding Yuan

Review

- For a memory access instruction
  - Does it use a virtual address or physical address?
  - What can happen?
    - Best case
    - What if you are unlucky?
- Demand paging
  - What is it?
- Page fault
  - What is it?
  - Why does it happen?
  - Who handles it?
  - How costly is it?

Demand Paging Algorithm

- Algorithm NEVER brings a page into main memory until it is needed
  1. Page fault
  2. Check if a valid virtual memory addr. Kill proc. if not.
  3. If valid address, check if it's cached in memory already (perhaps by other processes). If so, skip to 7.
     - How can this be possible?
  4. Find a free page frame. If no free page available, choose one to evict (which one? focus of this lecture)
     - If the victim page is dirty, write it out to disk first
  5. Suspend user process, map address into disk block and fetch disk block into page frame
  6. When disk read finished, add vm mapping for page frame
  7. If necessary, restart process.

Demand Paging (detail)

- Some
  - Pages are evicted to disk when memory is full
  - Pages loaded from disk when referenced again
  - References to evicted pages cause a TLB miss
    - PTE was invalid, causes fault
  - OS allocates a page frame, reads page from disk
  - When I/O completes, the OS fills in PTE, marks it valid, and restarts faulting process
- Dirty vs. clean pages
  - Actually, only dirty pages (modified) need to be written to disk
  - Clean pages do not – but you need to know where on disk to read them from again

Issue: Eviction

- Hopefully, kick out a less-useful page
- Goal: kick out the page that's least useful
- Problem: how do you determine utility?
  - Kick out pages that aren't likely to be used again
  - Heuristic: temporal locality exists

Page Replacement Strategies

- The Principle of Optimality
  - Replace the page that will not be used again the farthest time in the future
- Random replacement
  - Choose a page randomly
- FIFO – First In First Out
  - Replace the page that has been in memory the longest
- LRU – Least Recently Used
  - Replace the page that has not been used for the longest time
- NRU – Not Recently Used
  - An approximation to LRU

Belady's Algorithm

- Known as the optimal page replacement algorithm because it has the lowest fault rate for any page reference sequence
  - Idea: Replace the page that will not be used for the longest time in the future
  - Problem: Have to predict the future!
- Why is Belady's useful then? Use it as a yardstick
  - Compare implementations of page replacement algorithms with the optimal to gauge room for improvement
  - If optimal is not much better, then algorithm is pretty good
  - If optimal is much better, then algorithm could use some work
    - Random replacement is often the lower bound

Optimal Example

12 references, 7 faults

Miss rate: 7/12
Hit rate: 5/12

Page Refs	3 Page Frames			
	Fault?	Page Contents		
A	yes	A		
B	yes	B	A	
C	yes	C	B	A
D	yes	D	B	A
A	no	D	B	A
B	no	D	B	A
E	yes	E	B	A
A	no	E	B	A
B	no	E	B	A
C	yes	C	E	B
D	yes	D	C	E
E	no	D	C	E

First-In First-Out (FIFO)

- FIFO is an obvious algorithm and simple to implement
  - Maintain a list of pages in order in which they were paged in
  - On replacement, evict the one brought in longest time ago
- Why might this be good?
  - Maybe the one brought in the longest ago is not being used
- Why might this be bad?
  - Then again, maybe it's not
  - We don't have any info to say one way or the other
- FIFO suffers from "Belady's Anomaly"
  - The fault rate might actually increase when the algorithm is given more memory (very bad)

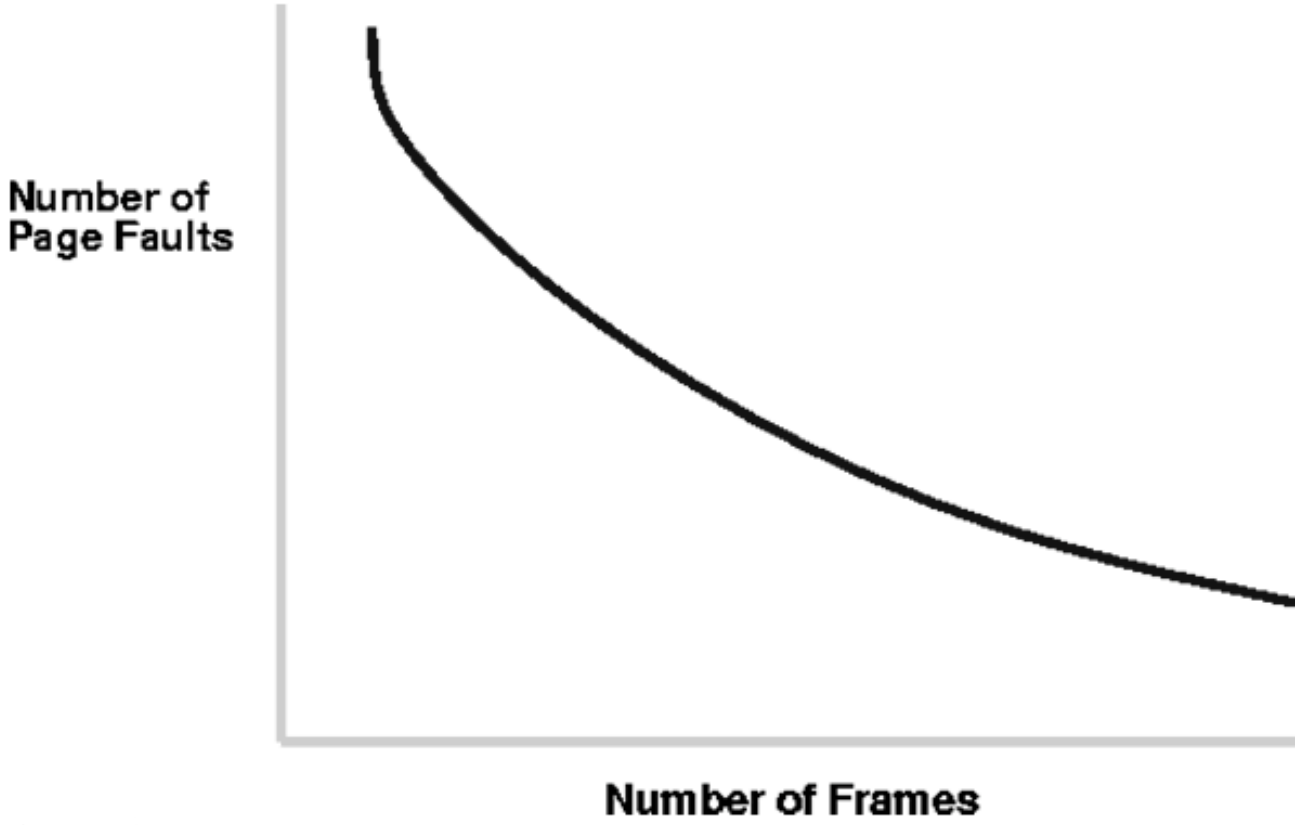
FIFO

12 references, 9 faults

Miss rate: 9/12
Hit rate: 3/12

Page Refs	3 Page Frames			
	Fault?	Page Contents		
A	yes	A		
B	yes	B	A	
C	yes	C	B	A
D	yes	D	C	B
A	yes	A	D	C
B	yes	B	A	D
E	yes	E	B	A
A	no	E	B	A
B	no	E	B	A
C	yes	C	E	B
D	yes	D	C	E
E	no	D	C	E

Intuitive Paging Behavior with Increasing Number of Page Frames



Belady's Anomaly (for FIFO)

12 references, 10 faults

	Page Refs	4 Page Frames Fault?	Page Contents			
1	A	yes	A			
2	B	yes	B	A		
3	C	yes	C	B	A	
4	D	yes	D	C	B	A
5	A	no	D	C	B	A
6	B	no	D	C	B	A
7	E	yes	E	D	C	B
8	A	yes	A	E	D	C
9	B	yes	B	A	E	D
10	C	yes	C	B	A	E
11	D	yes	D	C	B	A
12	E	yes	E	D	C	B

Least Recently Used (LRU)

- LRU uses reference information to make a more informed replacement decision
  - Idea: We can't predict the future, but we can make a guess based upon past experience
  - On replacement, evict the page that has not been used for the longest time in the past (Belady's: future)
  - When does LRU do well? When does LRU do poorly?
- Implementation
  - To be perfect, need to time stamp every reference (or maintain a stack) – much too costly
  - So we need to approximate it

LRU

12 references,
10 faults

No Belady's anomaly
• why?

Page Refs	3 Page Frames			
	Fault?	Page Contents		
A	yes	A		
B	yes	B	A	
C	yes	C	B	A
D	yes	D	C	B
A	yes	A	D	C
B	yes	B	A	D
E	yes	E	B	A
A	no	A	E	B
B	no	B	A	E
C	yes	C	B	A
D	yes	D	C	B
E	yes	E	D	C

Evict A (Least Recent)
Evict B (Least Recent)

Approximating LRU: NRU

- NRU: Evict a page that is NOT recently used;
- LRU: evict a page that is LEAST recently used
- NRU Implementation: simpler than LRU
  - uses reference bit
  - a counter is kept per bit
  - At regular intervals, for every page do:
    - if ref bit = 0, increment counter
    - if ref bit = 1, zero the counter
    - zero the reference bit
  - The counter will contain the number of intervals since the last reference to the page
  - The page with the largest counter is the least recently used

Review of last lecture

- Page replacement policy
  - What is the problem it tries to solve?
    - Similar problem in cache replacement policy you learnt before
  - Belady's algorithm
  - FIFO
    - doesn't make much sense
  - LRU
    - Approximation: NRU

LRU Clock (Not Recently Used)

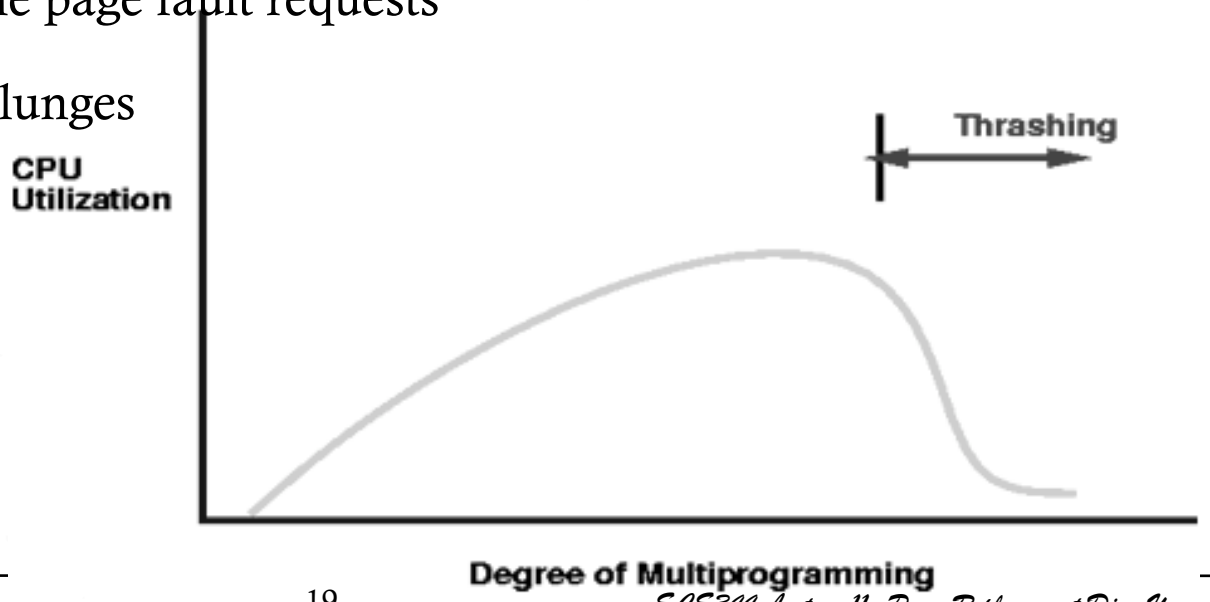
- Not Recently Used (NRU) – Used by Unix
  - Replace page that is "old enough"
  - Arrange all of physical page frames in a big circle (clock)
  - A clock hand is used to select a good LRU candidate
    - Sweep through the pages in circular order like a clock
    - If the ref bit is off, it hasn't been used recently
      - What is the 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 when plenty of memory
  - If memory is large, "accuracy" of information degrades
    - What does it degrade to?

Switching Gear

- So far, all we have talked about is memory management for a single process
- What about multiple processes?
  - If we just use "demand paging" for each process, why do we care?

Thrashing and CPU utilization

- As the page fault rate goes up, processes get suspended on page queues for the disk
- The system may try to optimize performance by starting new jobs
  - But is it always good?
- Starting new jobs will reduce the number of page frames available to each process, increasing the page fault requests
- System throughput plunges

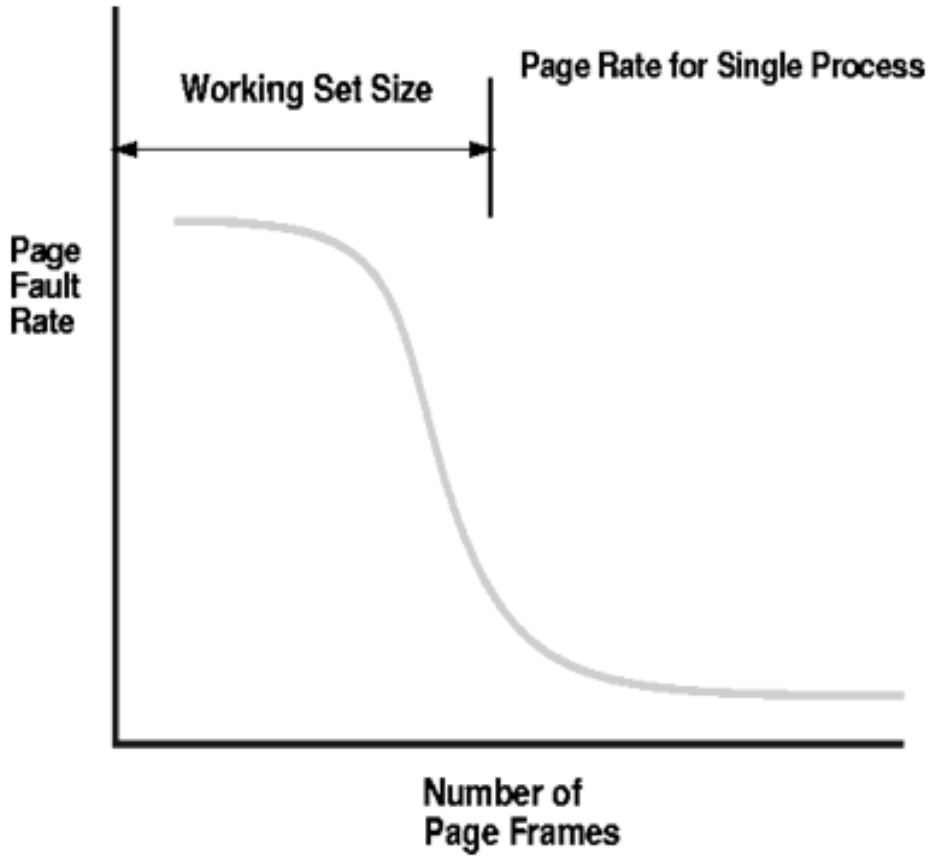


Fixed vs. Variable Space

- In a multiprogramming system, we need a way to allocate memory to competing processes
- Problem: How to determine how much memory to give to each process?
  - Fixed space algorithms
    - Each process is given a limit of pages it can use
    - When it reaches the limit, it replaces from its own pages
    - Local replacement
      - Some processes may do well while others suffer
  - Variable space algorithms
    - Process' set of pages grows and shrinks dynamically
    - Global replacement
      - One process can ruin it for the rest

Working Set

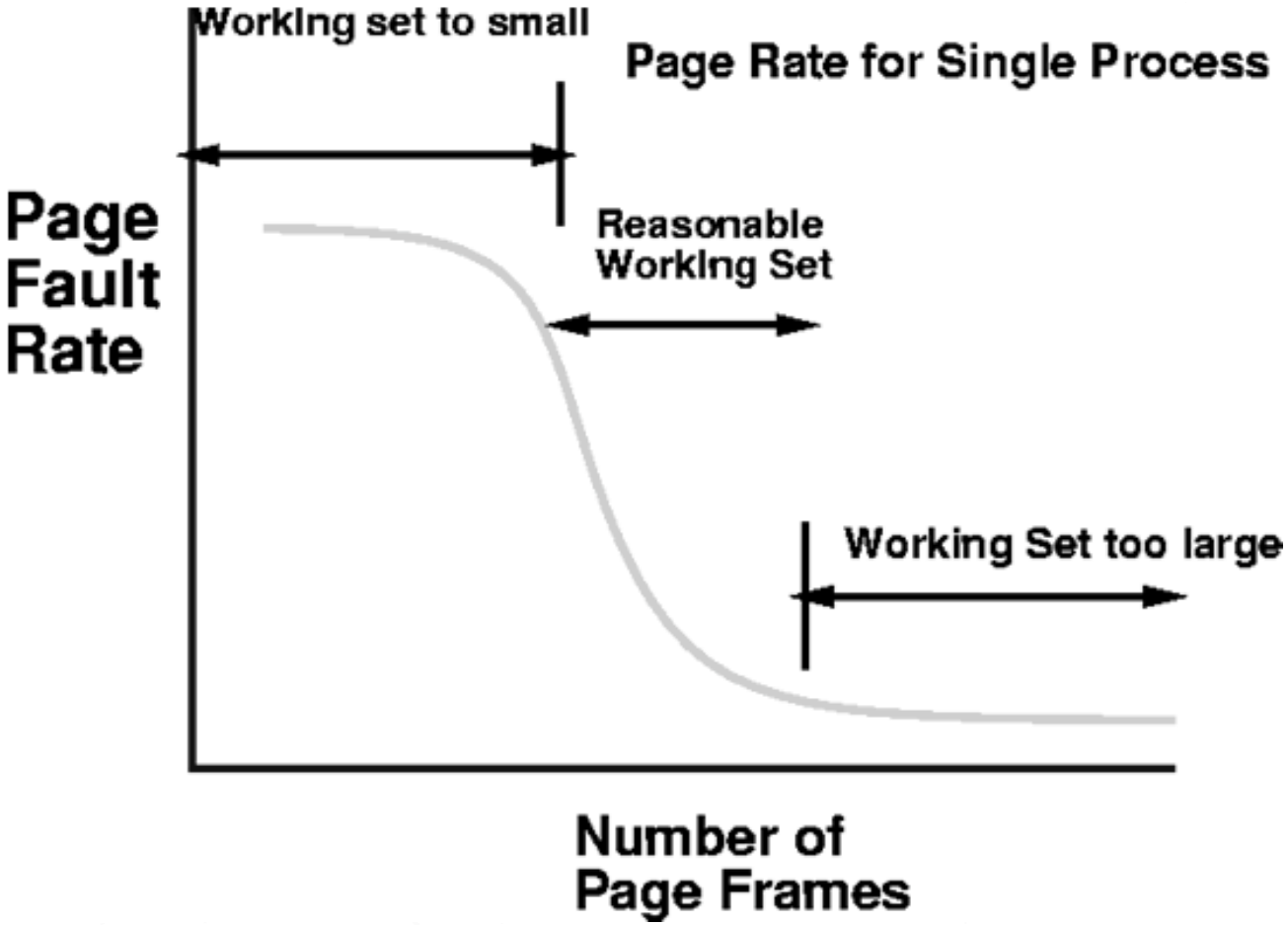
- The working set model assumes locality
- The principle of locality states that a program clusters its access to data and text temporarily
- As the number of page frames increases above some threshold, the page fault rate will drop dramatically



Working Set Model

- A working set of a process is used to model the dynamic locality of its memory usage
  - Defined by Peter Denning in 60s
- Definition
  - WS(t,w) = {pages P such that P was referenced in the time interval (t, t-w)}
  - t – time, w – working set window (measured in page refs)
- A page is in the working set (WS) only if it was referenced in the last w references

Working Set Size vs. Page Faults



Working Set Size

- The working set size is the number of pages in the working set
  - The number of pages referenced in the interval (t, t-w)
- The working set size changes with program locality
  - During periods of poor locality, you reference more pages
  - Within that period of time, the working set size is larger
- Intuitively, want the working set to be the set of pages a process needs in memory to prevent heavy faulting
  - Each process has a parameter w that determines a working set with few faults
  - Denning: Don't run a process unless working set is in memory

Working Set Problems

- Problems
  - How do we determine w?
  - How do we know when the working set changes?
- Too hard to answer
  - So, working set is not used in practice as a page replacement algorithm
- However, it is still used as an abstraction
  - The intuition is still valid
  - When people ask, "How much memory does Firefox need?", they are in effect asking for the size of Firefox's working set

Page Fault Frequency (PFF)

- Page Fault Frequency (PFF) is a variable space algorithm that uses a more ad-hoc approach
  - Monitor the fault rate for each process
  - If the fault rate is above a high threshold, give it more memory
    - So that it faults less
    - But not always (FIFO, Belady's Anomaly)
  - If the fault rate is below a low threshold, take away memory
    - Should fault more
    - But not always
- Hard to use PFF to distinguish between changes in locality and changes in size of working set

Summary

- Page replacement algorithms
  - Belady's – optimal replacement (minimum # of faults)
  - FIFO – replace page loaded furthest in past
  - LRU – replace page referenced furthest in past
    - Approximate using PTE reference bit
  - LRU Clock – replace page that is "old enough"
  - Working Set – keep the set of pages in memory that has minimal fault rate (the "working set")
  - Page Fault Frequency – grow/shrink page set as a function of fault rate
- Multiprogramming
  - Should a process replace its own page, or that of another?