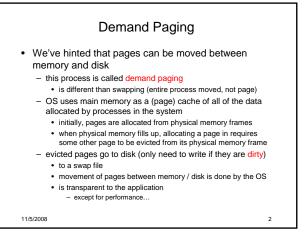
CSE 451: Operating Systems Autumn 2008 **Demand Paging and Page Replacement**

Hank Levv



Page Faults

- What happens to a process that references a VA in a page that has been evicted?
 - when the page was evicted, the OS sets the PTE as invalid and stores (in PTE) the location of the page in the swap file
 - when a process accesses the page, the invalid PTE will cause an exception (page fault) to be thrown

 - the OS will run the page fault handler in response · handler uses invalid PTE to locate page in swap file
 - · handler reads page into a physical frame, updates PTE to point to it and to be valid
 - · handler restarts the faulted process
- But: where does the page that's read in go?
 - have to evict something else (page replacement algorithm) OS typically tries to keep a pool of free pages around so that allocations don't inevitably cause evictions

11/5/2008

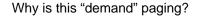
Why does this work?

- · Locality!
 - temporal locality
 - · locations referenced recently tend to be referenced again soon spatial locality
 - locations near recently references locations are likely to be referenced soon (think about why)
- · Locality means paging can be infrequent
 - once you've paged something in, it will be used many times
- on average, you use things that are paged in
- but, this depends on many things:
 - · degree of locality in application
 - page replacement policy and application reference pattern
 - amount of physical memory and application footprint

11/5/2008

3

Λ



• Think about when a process first starts up:

- it has a brand new page table, with all PTE valid bits 'false'
- no pages are yet mapped to physical memory
- when process starts executing:
 - instructions immediately fault on both code and data pages
 - · faults stop when all necessary code/data pages are in memory
 - only the code/data that is needed (demanded!) by process needs to be loaded
 - what is needed changes over time, of course...

11/5/2008

Evicting the best page

- · The goal of the page replacement algorithm:
 - reduce fault rate by selecting best victim page to remove
 - the best page to evict is one that will never be touched again
 as process will never again fault on it
 - "never" is a long time
 - Belady's proof: evicting the page that won't be used for the longest period of time minimizes page fault rate

6

Rest of this lecture:

- survey a bunch of replacement algorithms

11/5/2008

#1: Belady's Algorithm

- Provably optimal lowest fault rate (remember SJF?)
 pick the page that won't be used for longest time in future
 problem: impossible to predict future
- Why is Belady's algorithm useful?
 - as a yardstick to compare other algorithms to optimal
 - if Belady's isn't much better than yours, yours is pretty good
- Is there a lower bound?
 - unfortunately, lower bound depends on workload
 but, random replacement is pretty bad

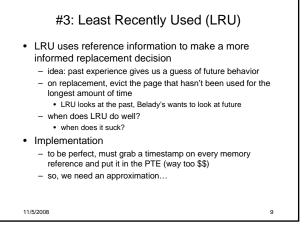
```
.
```

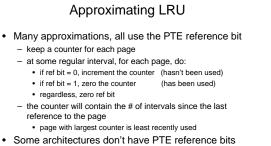
11/5/2008

7

5

#2: FIFO	
 FIFO is obvious, and simple to implement when you page in something, put in on tail of list on eviction, throw away page on head of 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 FIFO suffers from Belady's Anomaly fault rate might increase when algorithm is given more physical memory a very bad property 	
11/5/2008	8





- can simulate reference bit using the valid bit to induce faults hack, hack, hack

11/5/2008

#4: LRU Clock

- AKA Not Recently Used (NRU) or Second Chance - replace page that is "old enough"
 - 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

11/5/2008

11

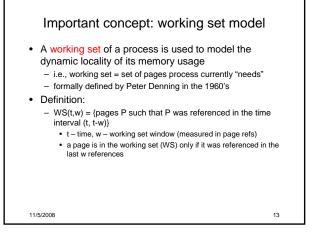
Another Problem: allocation of frames

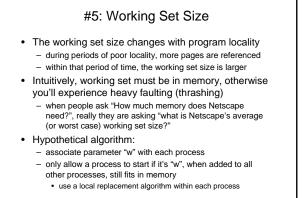
- In a multiprogramming system, we need a way to allocate physical memory to competing processes - what if a victim page belongs to another process?
 - family of replacement algorithms that takes this into account
- Fixed space algorithms
 - each process is given a limit of pages it can use
 - when it reaches its limit, it replaces from its own pages
 - local replacement: some process may do well, others suffer
- Variable space algorithms
 - processes' set of pages grows and shrinks dynamically
 - global replacement: one process can ruin it for the rest
 - · linux uses global replacement

11/5/2008

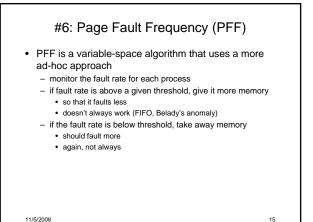
12

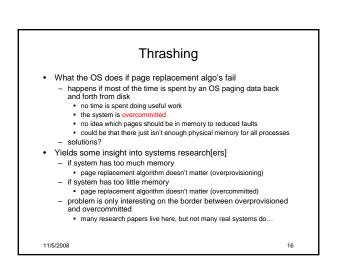
10





11/5/2008





Summary

demand paging

- start with no physical pages mapped, load them in on demand
- page replacement algorithms
 - #1: Belady's optimal, but unrealizable

 - #1: Belady's optimal, but unrealizable
 #2: Fifo replace page loaded furthest in past
 #3: LRU replace page reference d furthest in past
 approximate using PTE reference bit
 #4: LRU Clock replace page that is "old enough"
 #5: working set keep set of pages in memory that induces the minimal fault rate
 - #6: page full frequency grow/shrink page set as a function of fault rate
- · local vs. global replacement
 - should processes be allowed to evict each other's pages?

11/5/2008

17