CSCE 410

kernel types

- unikernel, microkernel, exokernel
- TODO

interrupts/exceptions

- TODO

virtual memory

- TODO

paging

- cost of page fault
- locality of reference
- TODO

page replacement policies

- when you need more memory, how do you decide which pages to evict (and possibly writ to disk)?
- FIFO
  - just evict the page that has been in memory the longest
  - pro: simple
  - con: does not exploit principle of locality of reference (assumes that pages resident in memory longer are more likely to continue to be referenced)
- ideal
  - evict page that will be next used farthest in the future (if at all)
  - pro: proven lowest number of page faults
  - con: impossible to implement in real life because we cannot see the future
- least recently used (LRU)

    - evict the page that has not been accessed in the longest period of time
    - pro: good performance
    - con: difficult to implement (must keep track of all references)
    - must keep chronological history of page references
        - software: use a stack
        - hardware: charge a capacitor and let it slowly drain
- 2nd chance
    - approximation to LRU
    - TODO
- 2nd chance enhanced
    - TODO

working set

- AKA resident set? TODO I think it's a little different
- ideal resident set size changes dynamically as program runs
    - but we assume working set is constant
- all pages referenced within a specified time delta
    - because the process doesn't need all of it's pages at once
- rule
    1. at each reference, working set is determined, and only pages in virtual set are kept in memory
    2. a program can only run if it's entire current resident set is in memory
- if all you reference is one page, working set eventually becomes only that page
- removing frames
    - victims are not overwritten immediately, instead are put into one of two lists:
    - free frame list
        - for clean frames (non-dirty) (frames that are ok to overwrite, as they are in sync with the swap)
        - OS can freely pick frames from here and use them
    - modified frame list
        - dirty frames
        - periodically write these frames to disk, and then move them to the free frame list
- when a frame not in the current working set is referenced, first check the free frame list and modified frame list
    - if the frame exists in either of those, you can simply reclaim it
- victims:
    - when looking for a victim, first get from free frame list
        - if free frame list is non-empty, just pick one and use it
    - if free frame list is empty

        - pick from modified list, **but write it to disk first**
- TODO: solaris page buffering

recursive paging

- TODO