Memory Managment

Page replacement algorithms:

- Page fault forces a choice
    - no roofer new page
    - which page must be removed to make room for an incoming page?
- How is a page removed from physical memory?
    - If the page is unmodified, simply overwrite it: a copy already exists on disk
    - if the page has been modified, it must be written back to disk: prefer unmodified pages
- better not to choose an often used page
    - it'll probably need to be brought back in soon

Optimal page replacement algorithm

- what's the best we can possibly do?
    - assume perfect knowledge of the future
    - not realized in practice
    - useful for comparison: if another algorithm is within 5% of optimal, not much more can be done...
- Algorithm: replace the page that will be used furthest in the future
    - only works we know the whole sequence
    - can be approximated by running the program twice
        - once be approximated by running the program twice
        - once to apply the optimal algorithm
- nice, but not achievable in real systems!
- offline algorithm

Not recently-used (NRU) algorithm:

- each page has reference bit and dirty bit
    - bits are set when page is referenced and/or modified
- pages are classified into four classes
    - 0: not referenced, not dirty
    - 1: not referenced, dirty
    - 2: referenced, not dirty
    - 3: referenced, dirty
- Clear reference but for all paged periodically
    - can't clear dirty bit: needed to indicate which pages need to be flushed to disk
    - Class 1 contains dirty pages where reference bit has been cleared

- algorithm: remove a page from the lowest non-empty class
    - select a page at random from that class
- easy to understand and implement
- performance adequate (though not optimal)

First-In, First-Out algorithm

- maintain a linked list of all pages
    - maintain the order in which they entered memory
- page at front of list replaced
- advantage: really easy to implement
- disadvantage: page in memory the longest may be often used
    - this algorithm forces pages out regardless of usage
    - usage may be helpful in determining which pages to keep

Second chance page replacement

- modify FIFO to avoid throwing out heavily used pages
    - if reference bit is 0, throw the page out
    - if reference bit is 1
        - reset the reference bit to 0
        - move page to the tail of the list
        - continue search for a free page
- Still easy to implement and better than plain FIFO

Clock Algorithm

- same functionality as second chance
- simpler implementation
    - clock hand points to next page to replace
    - if R = 0, replace page
    - if R=1, set R=0 and advance the clock hand
- Continue until page with R=0 is found
    - this may involve going all the way around the clock...