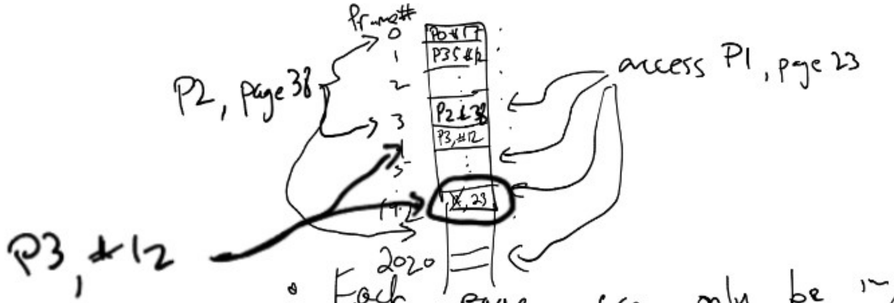
Lecture 14: Memory management

- Hashed paging
- Abusing permission bits
- Page replacement
- (time permitting) Thrashing

IPT (Hashed paging)

- table with entry per frame containing PID, Page# in that frame.
  - if log. addr. space huge → much smaller table.
- Need to search to find a page

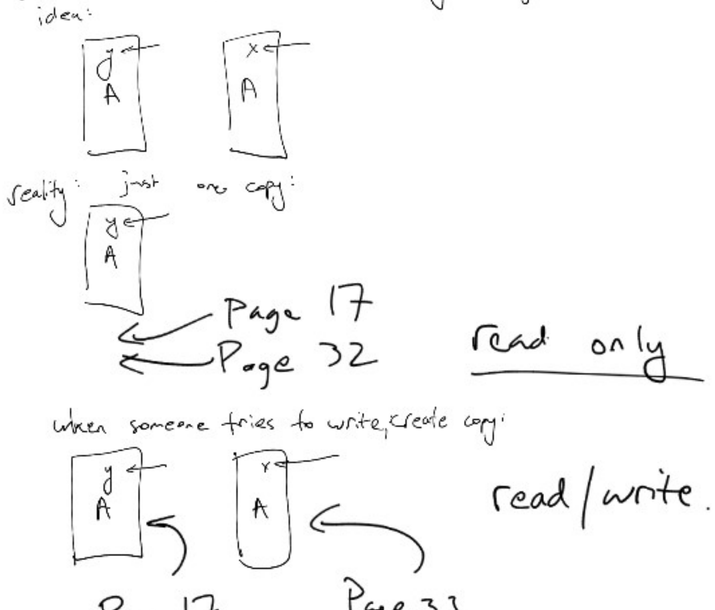


- Each page can only be in a small # of frames (e.g. 4) = hash (PID, page #, 0) = hash (PID, page #, 3)
  - only need to look in those places.
  - running
  - pointer we're accessing.
  - can have collisions
    - extremely unlikely all of the hashes for small # of different pages to all collide.
    - unlikely to have a lot of competition for small # of possible frame #s.

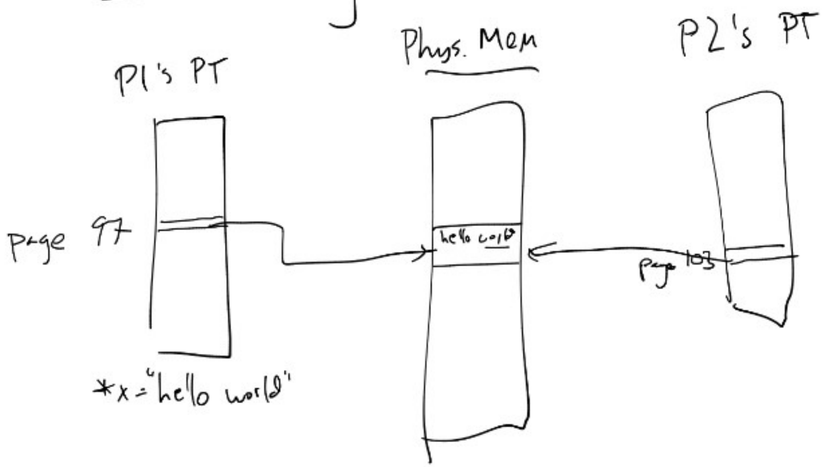
TLB protection bits

- Segmentation: try to write to code section, execute on stack, read NULL pointer } segmentation fault!
- lazily create pages full of NULLs (eg. for new process heap)
  - when created, mark not readable, mark as "zeroed" in PT.
  - if process tries to read → HW exception (segfault), at that point, zero out page. Mark it as R/W.

- Copy on write pages: want to copy a page.



Shared memory between processes



Debugger watchpoints

debugger

- Controlling process request a signal
  if controlled process updates a page } page marked read only.
  program being debugged.
- debugger & program share memory:
  debugger pause program & inspect new value of variable.

Page replacement:

- more logical pages than physical memory.
- extra pages need to be swapped out (to disk)
  [extremely slow!]

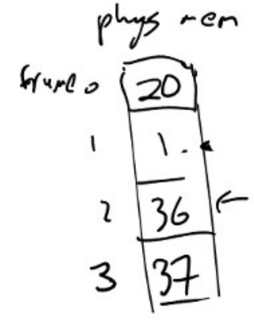
- When you access a page not in memory: need to kick out (swap out) a page that is → free up a frame for P.
  - Need to decide what to evict.

Possible algorithms to decide what to evict:

- RANDOM: randomly select a page to evict.
  Pro: fast, simple, no storage.
  Con: Can remove "popular" ↙ accessed frequently.
- OPT (Belady's algorithm): (evict) remove page that won't be accessed for the longest amount of time.
  Pro: minimizes # of page faults (can prove this)
  Con: Can't predict future!
- LRU (least recently used): keep track of when pages are accessed (timestamp). Evict page that was used least recently.
  Pro: recency is a good measure of future need. (temporal locality)
  Con: need space & time to keep track of usage.
  - can't do in OS (unless get exception for every access - terrible)
  - need HW support in TLB (put a timestamp in each TLB entry, TLB update timestamp)
- LFU (least frequently used):
  - use # of accesses as an estimation of future use
    - popular pages likely to be used soon.
  - just like LRU, but keep track of count of accesses instead of timestamp.
  - has same pros, cons as LRU
    - need space in PT,
    - need TLB to manage use counts.

which is better? depends on trace

- FIFO (first in, first out)
  keep rotating through frames, kick out next frame
  Pro: extremely simple.
  Con: when I first accessed a page may not be good predictor of future use.



P1, #73
P1, #42
#36
#37
#20 ←
#1 ←

Belady's anomaly:

- With some page replacement algorithms, can actually get worse performance with more memory
  - worse performance ← more page faults.