Announcements

- Project 2 due Fri March 24
  - 2 day extension to stay in synch with other CS 3204 sections
- Midterm Fri March 24
  - We start at 10:00am!
- Project 3 page table design document due Wed March 29
  - See project page

Virtual Memory

Paging Techniques

Demand paging

- Idea: only keep data in memory that’s being used
  - Needed for virtualization – don’t use up physical memory for data processes don’t access
- Requires that actual allocation of physical page frames be delayed until first access
- Many variations
  - Lazy loading of text & data, mmapped pages & newly allocated heap pages
  - Copy-on-write

Lazy Loading & Prefetching

- Typically want to do some prefetching when faulting in page
  - Reduces latency on subsequent faults
- Q.: how many pages? which pages?
  - Too much: waste time & space fetching unused pages
  - Too little: pay (relatively large) page fault latency too often
- Predict which pages the program will access next (how?)
- Let applications give hints to OS
  - If applications knows
    - Example: madvise(2)
  - Usual conflict: what’s best for application vs what’s best for system as a whole

Copy-On-Write

- Sometimes, want to create a copy of a page:
  - Example: Unix fork() creates copies of all parent’s pages in the child
- Optimization:
  - Don’t copy pages, copy PTEs – now have 2 PTEs pointing to frame
  - Set all PTEs read-only
  - Read accesses succeed
  - On Write access, copy the page into new frame, update PTEs to point to new & old frame
- Looks like each have their own copy, but postpone actual copying until one is writing the data
  - Hope is at most one will ever touch the data – never have to make actual copy
Page Eviction

- Suppose page fault occurs, but no free physical frame is there to allocate
- Must evict frame
  - Find victim frame (how – next lecture)
    - If data in it isn’t already somewhere on disk, write to special area on disk (“swap space”)
    - Find & change old page table entry pointing to the victim frame
    - Install in new page table entry
    - Resume
- Requires check on page fault if page has been swapped out – fault in if so
- Lots of subtleties with locking & order of things
  - Process A might be faulting, but victim page may be mapped in process B - but A&B run concurrently

Page Eviction Example

Process A needs a frame decides it wants this frame Q.: how will it find the PTE, if any, that points to it?

Linux uses a so-called “map” for that that links frames to PTE

Managing Swap Space

- Contiguous region on disk
  - Preferably on separate disk, but typically a partition on same disk
- Different allocation strategies are possible
  - Simplest: when page must be evicted, allocate swap space for page; deallocate when page is paged back in
  - Or: allocate swap space upfront
  - Should page’s position in swap space change? What if same page is paged out multiple times?
- Can be managed via bitmap 0100100000001
  - Free/used bits for each page that can be stored
  - Pintos: note 1 page == 8 sectors

Locking Frames

- Aka “pinned” or “wired” pages or frames
- If another device outside the CPU (e.g., DMA by network controller) accesses a frame, it cannot be paged out
  - Device driver must tell VM subsystem about this

Accessing User Pointers & Paging

- Kernel must check that user pointers are valid
  - P2: easy, just check range & page table
- Harder when swapping:
  - Validity of a pointer may change between check & access (if another process sneaks in and selects frame mapped to an already checked page for eviction)
- Possible solution:
  - verify & lock, then access, then unlock

if (verify_user(addr))
    process_terminate();
  // what if addr’s frame is just now swapped out by another process?
  *addr = value;