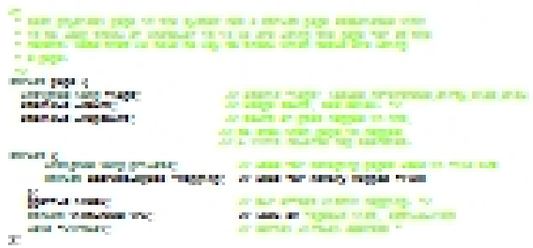
CS 537
Lecture 9
Paging and Page Replacement

Michael Swift

Hardware and Kernel structures for paging

- Hardware:
  - Page table base register
  - TLB
- Software:
  - Page table
    - Virtual --> physical or virtual --> disk mapping
  - Page frame database
    - One entry per physical page
    - Information on page, owning process
  - Swap file

Page Frame Database



Shared memory

- Exploit level of indirection between VA and PA
  - regions of two separate processes' address spaces map to the same physical frames
    - read/write: access to share data
    - execute: shared libraries!
  - will have separate PTEs per process, so can give different processes different access privileges
  - must the shared region map to the same VA in each process?

Saving memory to disk

- When there is not enough memory for all our processes, the OS can copy data to disk and re-use the memory for something else
  - Copying a whole process is called "swapping"
  - Copying a single page is called "paging"
- Where does data go?
  - If it came from a file and was read only, it says in the file
    - E.g. executable code
  - Unix: a swap partition
    - A region of the disk reserved for "backing store"
  - Windows: a swap file
    - A designated file in the regular file system
- When does data move?
  - Swapping: in advance of running a process
  - Paging: when a virtual page is accessed

Demand Paging

- We've hinted that pages can be moved between memory and disk
  - this process is called demand paging
  - OS uses main memory as a (page) cache of all of the data allocated by processes in the system
    - initially, pages are allocated from physical memory frames
    - when physical memory fills up, allocating a page requires some other page to be evicted from its physical memory frame
  - evicted pages go to disk (only need to write if they are dirty)
    - to a swap file
    - movement of pages between memory / disk is done by the OS
    - is transparent to the application
      - except for performance...

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

Why is this "demand" paging?

- 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...

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
      - [illegible]
    - 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

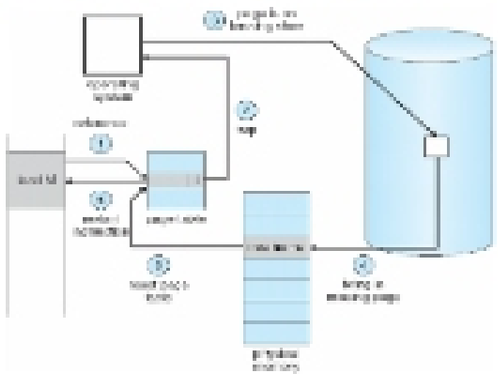
Page Fault

- If there is a reference to a page, first reference to that page will trap to operating system:

  page fault

1. Operating system looks at another table to decide:
   - Invalid reference ⇒ abort
   - Just not in memory
2. Get empty frame
3. Swap page into frame
4. Reset tables
5. Set validation bit = v
6. Restart the instruction that caused the page fault

Steps in Handling a Page Fault



Copy-on-Write

- copy-on-write (COW), e.g. on fork( )
  - instead of copying all pages, create shared mappings of parent pages in child address space
    - make shared mappings read-only in child space
    - when child does a write, a protection fault occurs, OS notes and can then copy the page and resume client
- Copy-on-Write (COW) allows both parent and child processes to initially share the same pages in memory
- If either process modifies a shared page, only then is the page copied
- COW allows more efficient process creation as only modified pages are copied
- Free pages are allocated from a pool of zeroed-out pages