Memory Management So Far

- An application's view of memory is its virtual address space
  - OS's view of memory is the physical address space
  - A MMU (hardware) is used to implement segmentation, paging, or a combination of the two, by providing address translation

- Limitation until now — all segments / pages of a process must be in main (physical) memory for it to run
  - When process isn’t running, the entire process can be swapped out to disk

- Insight — at a given time, we probably only need to access some small subset of process’s virtual memory
  - Load pages / segments on demand

Demand Paging (Virtual Memory)

- At a given time, a virtual memory page will be stored either:
  - In a frame in physical memory
  - On disk (backing store, or swap space)

- A process can run with only part of its virtual address space in main memory
  - Provide illusion of almost infinite memory

Starting a New Process

- Processes are started with 0 or more of their virtual pages in physical memory, and the rest on the disk

- Page selection — when are new pages brought into physical memory?
  - Prepaging — pre-load enough to get started: code, static data, one stack page (DEC ULTRIX)
  - Demand paging — start with 0 pages, load each page on demand (when a page fault occurs) (most common approach)
    - Disadvantage: many (slow) page faults when program starts running

- Demand paging works due to the principle of locality of reference
  - Knuth estimated that 90% of a program’s time is spent in 10% of the code

Page Faults

- An attempts to access a page that’s not in physical memory causes a page fault
  - Page table must include a present bit (sometimes called valid bit) for each page
  - An attempt to access a page without the present bit set results in a page fault, an exception which causes a trap to the OS
  - When a page fault occurs:
    - OS must page in the page — bring it from disk into a free frame in physical memory
    - OS must update page table & present bit
    - Faulting process continues execution

- Unlike interrupts, a page fault can occur any time there’s a memory reference
  - Even in the middle of an instruction! (how? and why not with interrupts??)
  - However, handling the page fault must be invisible to the process that caused it
Handling Page Faults

- The page fault handler must be able to recover enough of the machine state (at the time of the fault) to continue executing the program.

- The PC is usually incremented at the beginning of the instruction cycle.
  - If OS / hardware doesn’t do anything special, faulting process will execute the next instruction (skipping faulting one).

- With hardware support:
  - Test for faults before executing instruction (IBM 370).
  - Instruction completion — continue where you left off (Intel 386…).
  - Restart instruction, undoing (if necessary) whatever the instruction has already done (PDP-11, MIPS R3000, DEC Alpha, most modern architectures).

Performance of Demand Paging

- Effective access time for demand-paged memory can be computed as:
  
  \[
  e_{acc} = (1-p)(m_{acc}) + (p)(p_{fault})
  \]

  where:
  - \( p \) = probability that page fault will occur.
  - \( m_{acc} \) = memory access time.
  - \( p_{fault} \) = time needed to service page fault.

- With typical numbers:
  - \( e_{acc} = (1-p)(100) + (p)(25,000,000) = 100 + (p)(24,999,800) \)
  - If \( p \) is 1 in 1000,
    
    \[
    e_{acc} = 25,099 \text{ ns} \quad (250 \text{ times slower!})
    \]
  - To keep overhead under 10%,
    
    \[
    110 > 100 + (p)(24,999,800)
    \]
  - \( p \) must be less than 0.0000004.
  - Less than 1 in 2,500,000 memory accesses must page fault!

Page Replacement

- When the OS needs a frame to allocate to a process, and all frames are busy, it must evict (copy to backing store) a page from its frame to make room in memory.
  - Reduce overhead by having CPU set a modified / dirty bit to indicate that a page has been modified.
    - Only copy data back to disk for dirty pages.
    - For non-dirty pages, just update the page table to refer to copy on disk.

- Which page to we choose to replace?
  - Some page replacement policies:
    - Random
      - Pick any page to evict.
    - FIFO
      - Evict the page that has been in memory the longest (use a queue to keep track).
      - Idea is to give all pages “fair” (equal) use of memory.