Operating Systems
ECE344
Demand Paging

Ashvin Goel
ECE
University of Toronto
Outline

- Demand paging
- Virtual memory hierarchy
Motivation for Demand Paging

- Paging h/w simplifies memory management by allowing processes to access non-contiguous memory
- But, until now, we have required the entire process to be in memory, which causes several issues
  - Number of running programs limited by physical memory
  - Maximum size of a program limited by physical memory
Demand Paging

- **Demand paging** allow running programs that reside in memory **partially**

- **Memory becomes a cache for data on disk**
  - Cache hit: program accesses page in memory
  - Cache miss: page is loaded from disk into memory on demand
  - Cache eviction: pages in memory that are infrequently used are moved to disk (in an area called **swap**)
Memory as a Cache of Data on Disk

- CPU
- MMU
- TLB
- L1 Instr
- L1 Data
- L2 Cache
- L3 Cache
- Memory
- Disks (files)
Demand Paging

- **Demand paging** allow running programs that reside in memory partially

- Memory becomes a cache for data on disk
  - Cache hit: program accesses page in memory
  - Cache miss: page is loaded from disk into memory on demand, as needed
    - Entire program doesn’t need to be in memory
  - Cache eviction: pages in memory that are infrequently used are moved to disk (in an area called **swap**)
    - If page is not modified, it can be discarded
    - Otherwise, page needs to be saved to disk
Understanding Demand Paging

- How can we detect a cache miss, i.e., determine when a page should be loaded into memory?

- Recall that a page table entry (PTE) has a valid bit
  - When a page is not associated with a frame, the valid bit in the corresponding PTE is unset (invalid page)

- A cache miss occurs when a program accesses an address in an invalid page
  - Then MMU generates an exception, called a page fault, giving control to OS page fault handler code

- OS page fault handler performs miss handling
  - Allocates frame to page, updates PTE with frame number, sets valid bit in PTE, and restarts instruction
Understanding Demand Paging

- Demand paging is transparent
  - Program is not aware of demand paging
    - Similar to interrupt handling
  - Program is not aware of how much memory is available

- This approach works efficiently because programs tend to access recently accessed pages
  - Cache hit rate is high
Benefits of Demand Paging

- Allows running programs whose total memory requirements exceed available memory
- Allows running a program larger than physical memory
- Allows faster startup programs
- All these benefits were important when memory was very expensive (and small)

- Next, we see how demand paging works in more detail
Virtual Memory Hierarchy

- **Virtual Address**
  - **Lookup TLB**
    - Hit
  - **Page Table**
    - Hit
    - **Load TLB entry.**
    - If dirty TLB entry evicted, update PTE.
    - **Frame available**
      - Yes
      - Allocate frame.
      - Load page contents from disk.
      - Update PTE.
      - **Swap a frame**
    - Page fault
      - Yes
      - Allocate frame.
      - Load page contents from disk.
      - Update PTE.
      - **Swap a frame**
  - **Cache, Memory**
    - **Cache, Memory**
  - **Hardware**
    - **Hardware or OS Software**
  - **OS Software**
Common Case in Hardware

- Thread generates virtual address
- TLB does a lookup using page number of the address
- On a TLB hit (page number matches, TLB entry is valid, has consistent protection bits)
  - Get the frame number, combine with page offset to generate physical address
  - MMU reads physical address from cache/memory, returns value to CPU
- Otherwise, generate TLB miss fault
On TLB miss fault, CPU invokes TLB miss handler with **faulting virtual address**

Handler needs to load PTE into TLB
- Uses page nr to look up page table
- If PTE is valid:
  - Loads PTE into TLB
  - If a TLB entry is evicted and is dirty, synchronize corresponding PTE
- Else PTE is invalid:
  - Generate a page fault
Page Fault Handler

- On page fault, CPU invokes page fault handler with faulting virtual address
- Handler needs to allocate frame for page, map page to frame
  - Checks faulting address is in mapped region
  - If memory available:
    - Allocates frame
    - Loads page contents from disk into frame
      - Page contents in executable, swap
    - Allocates PTE, maps page to frame, makes PTE valid
    - Restarts faulting instruction
  - Else memory not available:
    - Invoke swap handler, try again
Swap Handler

- When no frames available, OS invokes swap handler
- Handler needs to evict a page to disk, return freed frame
  - Chooses a page to evict using page replacement algorithm
  - If page is modified, writes it to a free location on swap
    - For unmodified page, the up-to-date version is on disk
  - Find page(s) that map to the frame being evicted
    - Need mapping from frame to page (coremap, discussed later)
    - Change all corresponding PTEs to invalid
    - Keep track of where frame is located in swap in PTE (discussed later)
  - Return newly freed frame to page fault handler
Summary

- Paging hardware maps pages to frames at run time
  - Allows contiguous virtual address space
  - Allows non-contiguous memory allocation

- This indirection (mapping) enables demand paging
  - Physical memory becomes a cache for disk
  - Paging h/w enables detecting when a page is not in memory
  - Pages are loaded in memory from disk when they are needed
  - Programs are not limited by physical memory available

- Implementing demand paging requires both hardware and OS support
Think Time

- Why and when does hardware access page tables?
- Why and when does OS software access page tables?
- What does a page fault handler do?
- How does virtual memory allow running programs larger than physical memory?
- If programs can be larger than physical memory size, are there any limitations on their size?
Think Time Answers

- **Why and when does hardware access page tables?**
  - With h/w managed TLB, h/w access page tables on TLB-miss fault
  - With s/w managed TLB, h/w doesn’t access page tables

- **Why and when does OS software access page tables?**
  - OS accesses page tables for the following reasons:
    - For s/w managed TLB, on TLB-miss fault, it reads the page table
    - On page fault, it allocates frame and updates page table
    - Whenever address space of current process is modified, (frame is allocated/deallocated, protection changes to pages, context switch), it updates page table
Think Time Answers

- **What does a page fault handler do?**
  - Checks for errors, such as seg fault, protection fault
  - Allocates frame, pte
  - Loads data from disk into frame
  - Maps page to frame by updating pte

- **How does virtual memory allow running programs larger than physical memory?**
  - Using demand paging, only recently accessed pages are kept in physical memory

- **If programs can be larger than physical memory size, are there any limitations on their size?**
  - Programs are limited by their virtual address space (processor architecture) or by the amount of disk space