Caches and Virtual Memory

CS 333
Fall 2006

Topics

- Cache Operations
  - Placement strategy
  - Replacement strategy
  - Read and write policy
- Virtual Memory
  - Why?
  - General overview
  - Lots of terminology

Cache Operations

Cache Operations

- Placement strategy
  - Where to place an incoming block in the cache
- Read and write policies
  - How to handle reads and writes on cache hits and misses
- Replacement strategy
  - Which block to replace on cache miss

Read and Write Policies

Write Policies on Cache Hit

- Write-through
  - Update both cache and main memory on a write
    - Advantageous when:
      - Few writes
    - Disadvantageous:
      - Many writes (takes time)
- Write-back
  - Write to cache only. Update main memory when block is replaced (needs dirty bit per cache block)

Read Miss Policies

- Read block in from main memory.
  - Two approaches:
    - Forward desired word first (faster, but more hardware needed)
    - Delay until entire block has been moved to cache

Write Miss Policies

- Write allocate
  - Bring block to cache from memory, then update
- Write-no allocate
  - Only update main memory (don't bring into cache)

Write through caches usually use write-no allocate
Write-back caches usually use write allocate

Block Replacement Strategies

Block Replacement Policies

- If the cache is full
  - Least recently used (LRU)
    - Uses counters
  - Random replacement

What do you need to implement LRU?

Virtual Memory

Why Have A Memory Hierarchy?

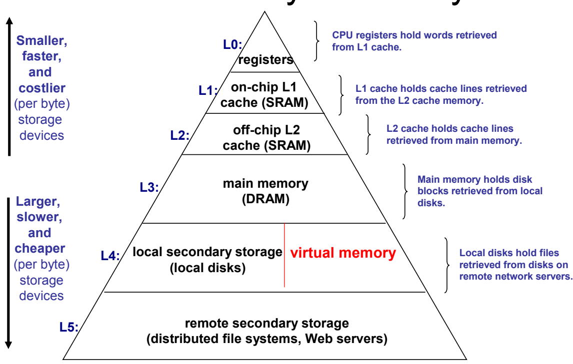
Want to create the illusion of large memory at small memory speeds

- Infinite storage
- Fast
- Cheap
- Compact
- Cold
- Non-volatile (can remember w/o electricity)

Can't have everything:

- **Bigger → Slower (speed of light)**
- **Faster, denser → hotter**
- **Faster → More expensive**

The Memory Hierarchy



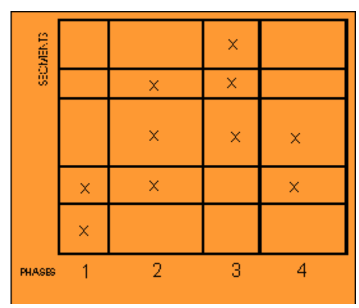
Why Use Disks?

- Multiprogramming
  - More than one application running at one time
    - May need more than available main memory to store all needed programs and data
    - May want to share data between applications
- Cheap, large (but, slow)

What was there before Virtual Memory?

- **Overlays**
  - Programmers had to use the principal of locality to choose segments of a program to keep in memory during program execution
    - 80/20 rule (80% of the time executing 20% of the code)
    - Use program phase behavior
- **Tedious** to do by hand
- **Difficult to determine overlay set**, especially when considering multiprogramming

Program Overlays



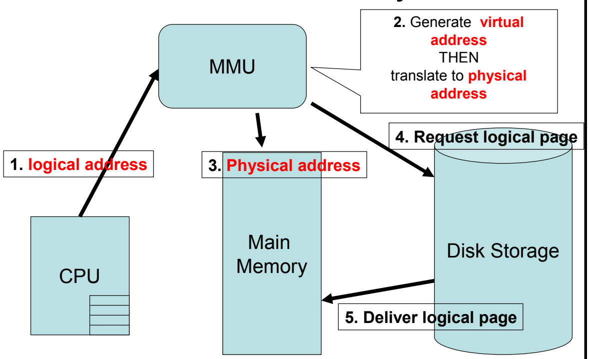
What is **Virtual Memory**?

- Technique (an abstraction) for using disks to extend the apparent size of **physical memory** beyond it's actual physical size
  - Automatic storage allocation
- **Logical memory** – memory as seen by the process
- **Physical memory** – memory as seen by the processor

Virtual Memory Overview

- Components
  - **Memory Management Unit** (MMU)
    - mapping function between logical addresses and physical addresses
  - **Operating System**
    - controls the MMU
  - **Mapping tables**
    - guide the translation

Virtual Memory



Terminology

- **Effective address** – address computed by processor (as viewed inside CPU)
- **Logical address** – same as effective address (as viewed outside CPU)
- **Virtual address** – generated by MMU
- **Physical address** – address in physical memory (main memory)

Why Have Virtual Addresses?

Virtual address can be **larger** than the logical address, allowing program units to be mapped to a much larger virtual address space

Example: PowerPC 601

logical address: 32 bits
virtual address: 52 bits
physical address: depends on how much memory

Virtual Memory Advantages

- Simpler addressing
  - Programs can be compiled with their own address space
    - No need for compiler to generate addresses that are unique from addresses for other programs
    - Don't need to break program into fragments (overlays) to accommodate memory limitations
  - Operating system
- Disks are cheaper
- Access control (read, write, execute)
  - "bus error" – invalid virtual address
  - "segmentation fault" – improper permissions for the type of access

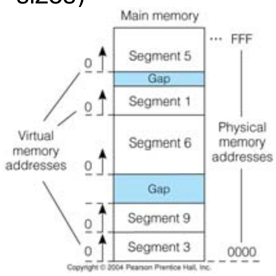
Approaches to Implementing Virtual Memory

Memory Management Approaches

- Segmentation
- Paging

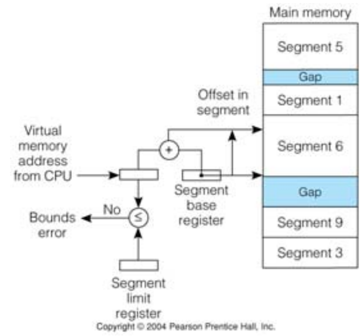
Segmentation

- Divide memory into segments (varying sizes)



Problem:
External fragmentation

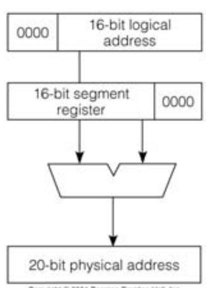
Address Translation in Segmented Memory



Q: How can OS switch processes?

Intel 8086

- 4 segment bases initialized by operating system
  - code
  - data
  - stack
  - extra (programmer use)
- 16-bit logical address
- 20-bit physical address



Paging

- Fixed size pages
- Demand paging
  - Pages brought in as needed
- Components
  - **Page table**
    - mapping of virtual pages to physical pages
    - generally one page table per process in the system
  - Virtual address

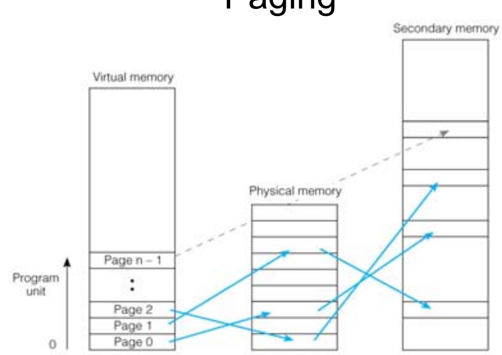
Page Number	Offset in Page

Page Table Entry

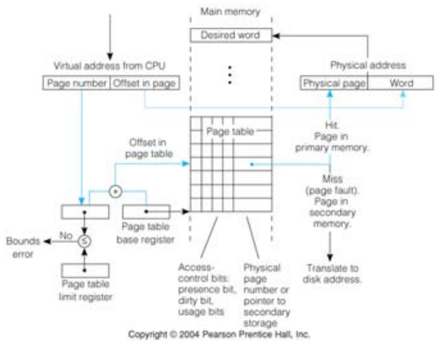
- Access control bits
  - Read, write, execute, etc.
- Presence bit
  - present in main memory (or not)
- Dirty bit
  - If the page has been modified
- Use bit
  - Recently used?
- Page number
  - physical page number OR pointer to secondary storage

Access	Present	Dirty	Use	Page Number

Paging



Address Translation in a Paged MMU



Paging

- Problem
  - **Internal fragmentation**
    - Last page is unlikely to be full
- Page Placement
  - Page table is direct-mapped
- Page Replacement
  - Use bits

Summary

- Cache Operations
  - Placement strategy
  - Replacement strategy
  - Read and write policy
- Virtual Memory
  - Why?
  - General overview
  - Lots of terminology