Next few lectures are going to cover memory management

- **Goals of memory management**
  - To provide a convenient abstraction for programming
  - To allocate scarce memory resources among competing processes to maximize performance with minimal overhead

- **Mechanisms**
  - Physical and virtual addressing (1)
  - Techniques: Partitioning, paging, segmentation (1)
  - Page table management, TLBs, VM tricks (2)

- **Policies**
  - Page replacement algorithms (3)
Lecture Overview

- Virtual memory warm-and-fuzzy
- Survey techniques for implementing virtual memory
  - Fixed and variable partitioning
  - Paging
  - Segmentation
- Focus on hardware support and lookup procedure
  - Next lecture we’ll go into sharing, protection, efficient implementations, and other VM tricks and features
Virtual Memory

- OS provides Virtual Memory (VM) as the abstraction for managing memory
  - Indirection allows moving programs around in memory
  - Allows processes to address more or less memory than physically installed in the machine
    - Virtual memory enables a program to execute with less than its complete data in physical memory
    - Many programs do not need all of their code and data at once (or ever) – no need to allocate memory for it
    - OS adjusts amount of memory allocated based upon behavior
- Requires hardware support for efficient implementation
- Let’s go back to the beginning…
In the beginning...

- Rewind to the days of batch programming
  - Programs use **physical addresses** directly
  - OS loads job, runs it, unloads it

- Multiprogramming changes all of this
  - Want multiple processes in memory at once
    - Overlap I/O and CPU of multiple jobs
  - Can do it a number of ways
    - Fixed and variable partitioning, paging, segmentation
  - Requirements
    - Need protection – restrict which addresses jobs can use
    - Fast translation – lookups need to be fast
    - Fast change – updating memory hardware on context switch
Virtual Addresses

- To make it easier to manage the memory of processes running in the system, we’re going to make them use virtual addresses (logical addresses)
  - Virtual addresses are independent of the actual physical location of the data referenced
  - OS determines location of data in physical memory
  - Instructions executed by the CPU issue virtual addresses
  - Virtual addresses are translated by hardware into physical addresses (with help from OS)
  - The set of virtual addresses that can be used by a process comprises its virtual address space

- Many ways to do this translation…
  - Start with old, simple ways, progress to current techniques
Fixed Partitions

- Physical memory is broken up into fixed partitions
  - Hardware requirements: base register
  - Physical address = virtual address + base register
  - Base register loaded by OS when it switches to a process
  - Size of each partition is the same and fixed
  - How do we provide protection?

- Advantages
  - Easy to implement, fast context switch

- Problems
  - Internal fragmentation: memory in a partition not used by a process is not available to other processes
  - Partition size: one size does not fit all (very large processes?)
Fixed Partitions

Base Register
P4’s Base

Virtual Address
Offset

+  

Physical Memory

P1
P2
P3
P4
P5
Variable Partitions

- Natural extension -- physical memory is broken up into variable sized partitions
  - Hardware requirements: base register and limit register
  - Physical address = virtual address + base register
  - Why do we need the limit register? Protection
  - If (physical address > base + limit) then exception fault

- Advantages
  - No internal fragmentation: allocate just enough for process

- Problems
  - External fragmentation: job loading and unloading produces empty holes scattered throughout memory
Variable Partitions

Virtual Address

Offset

<

Yes?

P3’s Base

Limit Register

P3’s Limit

P1

P2

P3

Protection Fault

No?

Base Register

P3’s Base

Limit Register

P3’s Limit
Paging

- Paging solves the external fragmentation problem by using fixed sized units in both physical and virtual memory.
Users (and processes) view memory as one contiguous address space from 0 through N
- Virtual address space (VAS)

In reality, pages are scattered throughout physical storage

The mapping is invisible to the program

Protection is provided because a program cannot reference memory outside of its VAS
- The address “0x1000” maps to different physical addresses in different processes
Translating addresses
- Virtual address has two parts: virtual page number and offset
- Virtual page number (VPN) is an index into a page table
- Page table determines page frame number (PFN)
- Physical address isPFN::offset

Page tables
- Map virtual page number (VPN) to page frame number (PFN)
  - VPN is the index into the table that determines PFN
- One page table entry (PTE) per page in virtual address space
  - Or, one PTE per VPN
Page Lookups

Virtual Address
Page number Offset

Page Table
Page frame

Physical Address
Page frame Offset

Physical Memory
Paging Example

- Pages are 4K
  - VPN is 20 bits ($2^{20}$ VPNs), offset is 12 bits
- Virtual address is 0x7468
  - Virtual page is 0x7, offset is 0x468
- Page table entry 0x7 contains 0x2
  - Page frame base is $0x2 \times 0x1000$ (4K) = $0x2000$
  - Seventh virtual page is at address 0x2000 (3rd physical page)
- Physical address = $0x2000 + 0x468 = 0x2468$
Next time…

- Read Chapter 9