Lecture 7: Memory Management

CSE 120: Principles of Operating Systems

UC San Diego: Summer Session I, 2009
Frank Uyeda
Announcements

• PeerWise questions due tomorrow.
• Project 2 is due on Friday.
  – Milestone on Tuesday night.
• Homework 3 is due next Monday.
Goals for Today

• Motivation for Memory Management
• Understand Paging
  – How to translate from virtual to physical address
  – Determine size, structure of page tables
Memory Management

• Goals of memory management
  – Provide a convenient abstraction for programming
  – Allocate scarce memory resources among competing processes
  – Maximize performance with minimal overhead

• Mechanisms
  – Physical and virtual addressing
  – Techniques: Partitioning, paging, segmentation
  – Page table management, TLBs, VM tricks

• Policies
  – Page replacement algorithms
In the beginning.....

• Batch programmed systems
  – Programs use physical addresses directly
  – OS loads job, runs it, unloads it
  – Similar to what nachos does right now
    (you’ll change this in Project 2)
Let there be Multiprogramming

• 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 changes – updating memory hardware on context switch
Virtual Memory

• The basic abstraction provided by the OS memory management is virtual memory
  – A process’s address space in memory is not necessarily the same as the physical memory (RAM) address in which it resides
  – When a process requests a memory address, the OS will translate the address from a virtual address to a physical address.
Virtual Addresses

- Processes access memory using a **virtual address**
  - The virtual address is not the same as the physical RAM address in which it resides
  - The OS (hardware MMU) translates the virtual address into the physical RAM address
  - Who determines the mapping for the translation?
Virtual Addresses

- Processes access memory using a **virtual address**
  - The virtual address is not the same as the physical RAM address in which it resides
  - The OS (hardware MMU) translates the virtual address into the physical RAM address
  - Who determines the mapping for the translation?
Virtual Addresses

- Processes access memory using a **virtual address**
  - The virtual address is not the same as the physical RAM address in which it resides
  - The OS (**hardware MMU**) translates the virtual address into the physical RAM address
  - Who determines the mapping for the translation?
Virtual Memory

• Virtual memory enables programs to execute without requiring their entire address space reside in physical memory
  – Saves space
    • Many programs do not need all of their code and data at once (or ever), so there is no need to allocate memory for it
  – Allows flexibility for application and OS
    • Indirection allows moving programs around in memory; OS can adjust amount of memory allocated based upon its run-time behavior
    • Allows processes to address more or less memory than physically installed in the machine
  – Isolation and protection
    • One process cannot access memory addresses in others
    • Exception: shared memory, which we’ve already covered
Memory Management Requirements

• Protection
  – Restrict which physical addresses processes can use, so they can’t stomp on each other

• Fast translation
  – Accessing memory must be fast, regardless of the protection scheme

• Fast context switching
  – Overhead of updating memory hardware on a context switch must be low

• Requires hardware support for efficient implementation
MMU and TLB

- Memory Management Unit (MMU)
  - Hardware unit that translates a virtual address to a physical address
  - Each memory reference is passed through the MMU
  - Translate a virtual address to a physical address

- Translation Lookaside Buffer (TLB)
  - Essentially a cache for the MMU’s virtual-to-physical translations table
  - Not needed for correctness but source of significant performance gain
Memory Allocation

• How should we allocate memory to processes?
Fixed Partitions

- Physical memory is broken up into fixed partitions
  - Hardware requirements: base register
  - Physical address = virtual address + base register
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?)
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 protection fault
Variable Partitions

Base Register
P4’s Base

Limit Register
P3’s Base

Virtual Address
Offset

<

Yes?

No?

Protection Fault

Physical Memory

P1

P2

P3
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? Protections
    • 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
Paging

• Paging solves the external fragmentation problem by using fixed sized units in both physical and virtual memory
User/Process Perspective

• Users (and processes) view memory as one contiguous address space from 0 to N
  – Virtual address space

• 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 virtual address space
  – The address “0x1000” maps to different physical addresses in different processes
### Paging

- **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 is PFN::offset

\[
0x{\text{BAADF00D}} = \begin{array}{c}
0x{\text{BAADF}} \\
0x{\text{00D}}
\end{array}
\]

\[
0x{\text{BAADF}} \xrightarrow{\text{Translation Table}} 0x{\text{900DF}}
\]

- virtual address
- virtual page number
- offset
- virtual page number
- page table
- physical page number
- (page frame number)
Page Lookups

Virtual Address

Page number
0xBAADF
Offset
0xFOOD

Physical Address

Page frame
0x900DF
Offset
0xFOOD

Page Table

Page frame

Physical Memory

0x00000000
Page 1
Page 2
Page 3
Page N

0x900DF00D
0xFFFF0000

Page Lookup Diagram
Paging

• 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
Paging Example

• Memory address is **32 bits**
• Pages are **4K**
  – VPN is____ bits (___ VPNs), offset is____ bits
• Virtual address is **0x7468**
  – Virtual page is ____, offset is ______
• Page table entry ____ contains **0x2**
  – Page frame base is 0x2 << _______ = _______
  – ______th virtual page is address ________ (3rd physical page)
• Physical address = ______ + ______ = ______
Paging Example

• Memory address is 32 bits
• Pages are 4K
  – VPN is 20 bits (1M 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 << 12 bits = 0x2000
  – 7_th virtual page is address 0x2000 (3rd physical page)
• Physical address = 0x2000 + 0x468 = 0x2468
Next Time

• Read Chapter 8.3-8.8
• Peerwise questions due tomorrow at midnight.
• Check Web site for course announcements
  – http://www.cs.ucsd.edu/classes/su09/cse120