CS153: Memory Management 1

Chengyu Song

Slides modified from Harsha Madhyvasta, Nael Abu-Ghazaleh, and Zhiyun Qian
Administrivia

- **Lab**
  - Lab2 has been released
  - Due Monday June 5th midnight
  - Design document due Monday May 22 midnight
Recap and preview

• So far we were focusing on CPU resources virtualization
  • Process, thread, scheduling, synchronization
• In the second half of this class we will cover
  • Memory virtualization
  • Persistent (file systems)
  • Communications
  • Security
Need for virtual address space

• Multiprogramming
  • Want multiple processes in memory at once
    • Overlap I/O and CPU of multiple jobs

• How to share physical memory across multiple processes?
  • Many programs do not need all of their code and data at once (or ever) – no need to allocate memory for it
  • A program can run on machine with less memory than it "needs"
Virtual addresses

- **Virtual addresses** (logical addresses)
  - Easier for processes to manage their memory
  - Independent of the actual physical location of the data referenced
  - Instructions executed by the CPU access virtual addresses
  - Virtual addresses are translated by hardware into physical addresses
    - OS determines location of data in physical memory
  - The set of virtual addresses that can be used by a process comprises its **virtual address space**
Virtual address translation

- Many ways to do this translation
- Requirements
  - **Protection** – restrict which addresses jobs can use
  - **Fast translation** – lookups need to be fast
  - **Fast change** – updating memory hardware on context switch
First try: fixed partitions

+----+----+----+----+----+
| P1 | P2 | P3 | P4 | P5 |
+----+----+----+----+----+

- Physical memory is broken up into fixed partitions
  - Size of each partition is the same and fixed
  - Hardware requirements: base register
  - Physical address = virtual address + base register
  - Base register loaded by OS when it switches to a process (part of PCB)
Fixed partitions address translation

Base Register
- P4’s Base

Virtual Address
- Offset

[0, MAX_PART_SIZE)

How do we provide protection?

Physical Memory
- P1
- P2
- P3
- P4
- P5
Fixed partitions

• Advantages
  • Easy to implement: only need base register, protection only needs to verify offset is less than the fixed partition size
  • 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?)
Second try: 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 (virtual address > limit) then fault
Variable partitions address translation

Virtual Address

Offset

[0, LIMIT)

<

Yes?

<

No?

Base Register

P3's Base

Limit Register

P3's Limit

Protection Fault

+
Variable Partitions

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

• Problems?
  • **External fragmentation**: job loading and unloading produces empty holes scattered throughout memory
Third try: segmentations

- Partitions memory into logically related data units
  - Code, stack, data, file, etc.
- Natural extension of variable-sized partitions
  - Variable-sized partitions = 1 segment/process
  - Segmentation = many segments/process
- Hardware support
  - Multiple base/limit pairs, one per segment (segment table)
  - Virtual addresses become segment#::offset
Segmentation address translation

Virtual Address

Segment Table

Physical Memory

Segment # Offset

<

limit base

Yes?

+

No?

Protection Fault
Segmentations

• Advantages
  • More flexible than variable partitions

• Problems?
  • External fragmentation still exists
State-of-the-Art: paging

- Paging solves the external fragmentation problem by using tiny and fixed sized units in both physical and virtual memory.
Process perspective

- Its virtual address space (VAS) is one contiguous address space from 0 through N (N = $2^{32}$ on 32-bit arch)
- In reality, pages are scattered throughout physical memory
- The mapping is invisible to the program
- Protection is provided by disallowing a process to reference *physical memory* outside of its VAS
Paging address translation

- 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**
Paging address translation (cont.)

Diagram showing the process of paging address translation:

- Virtual Address
  - Page number
  - Offset
- Page Table
  - Page frame
- Physical Address
  - Page frame
  - Offset
- Physical Memory
  - Register storing page table addr
Paging example

- Pages are 4KB
  - Offset is 12 bits (because 4KB = $2^{12}$ bytes)
  - VPN is 20 bits (32 bits is the length of every virtual address)
- Virtual address is \(0x7468\)
  - VPN is \(0x7\), offset is \(0x468\)
- Page table entry \(0x7\) contains \(0x2000\)
  - PFN is \(0x2000\)
- Physical address = \(0x2000 + 0x468 = 0x2468\)
Page table entries (PTEs)

+---------------------+-----+----+-+-+-+
|  Page Frame Number  | ... |PROT|R|M|V|
+---------------------+-----+----+-+-+-+
|<---- 20 bits ---->|
4b 1 1 1

- The page frame number (PFN) determines physical page
- The Protection bits say what operations are allowed on page
  - Read, write, execute, kernel (Why do we need these?)
- The Valid bit says whether or not the PTE can be used
  - It is checked each time the virtual address is used (Why?)
The Reference bit says whether the page has been accessed
  • It is set when a read or write to the page occurs
The Modify bit says whether or not the page has been written
  • It is set when a write to the page occurs
Virtual memory

- Memory not backed by physical memory
  - Also known as swap pages

Why?
- To serve a process that asks for more memory than we physically have

How?
- By transferring a page worth of physical memory to disks
Virtual memory (cont.)
Paging advantages

• Easy to allocate memory
  • Memory comes from a free list of fixed size chunks
  • Allocating a page is just removing it from the list
  • External fragmentation not a problem
• Easy to swap out chunks of a program
  • All chunks are the same size
  • Use valid bit to detect references to swapped pages
  • Pages are a convenient multiple of the disk block size
Problems of naive paging implementation

- Can still have internal fragmentation
- Memory reference overhead
  - 2 references per address lookup (page table, then memory)
- Memory required to hold page table can be significant
  - 32 bit address space w/ 4KB pages = $2^{20}$ PTEs
  - 4 bytes/PTE = 4MB/page table
  - 25 processes = 100MB just for page tables!
Summary

- Virtual address space
  - Processes use virtual addresses
  - OS + hardware translates virtual address into physical addresses
- Various techniques
  - Fixed partitions, variable partitions, segmentation, paging
- Virtual memory
  - Memory that is not backed by physical memory but swapped to disk
Next class ...

• More on paging
• Textbook
  • Module 17, 18, 19