CSIS 429 Operating Systems

Lecture 6: Segmentation

September 23rd 2020

Textbook chapters

Read “Address Translation” and “Segmentation”

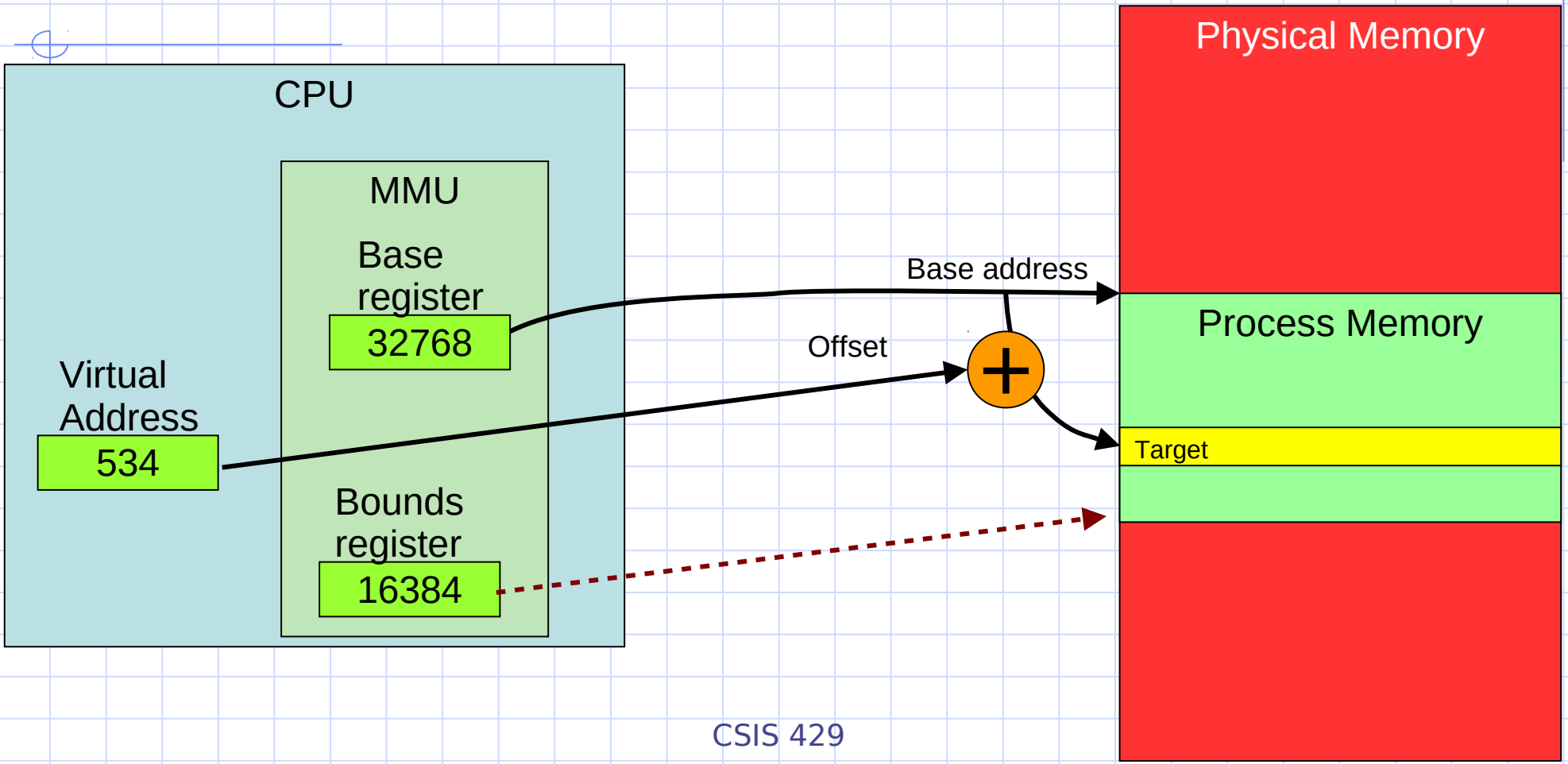
Intro	Virtualization		Concurrency	Persistence	Appendices
Preface	3 *Dialogue*	12 *Dialogue*	25 *Dialogue*	35 *Dialogue*	*Dialogue*
TOC	4 Processes	13 Address Spaces	26 Concurrency and Threads code	36 I/O Devices	Virtual Machines
1 *Dialogue*	5 Process API code	14 Memory API	27 Thread API	37 Hard Disk Drives	*Dialogue*
2 Introduction code	6 Direct Execution	15 Address Translation	28 Locks	38 Redundant Disk Arrays (RAID)	Monitors
	7 CPU Scheduling	16 Segmentation	29 Locked Data Structures	39 Files and Directories	*Dialogue*
	8 Multi-level Feedback	17 Free Space Management	30 Condition Variables	40 File System Implementation	Lab Tutorial
	9 Lottery Scheduling code	18 Introduction to Paging	31 Semaphores	41 Fast File System (FFS)	Systems Labs
	10 Multi-CPU Scheduling	19 Translation Lookaside Buffers	32 Concurrency Bugs	42 FSCK and Journaling	xv6 Labs
	11 *Summary*	20 Advanced Page Tables	33 Event-based Concurrency	43 Log-structured File System (LFS)	Flash-based SSDs
		21 Swapping: Mechanisms	34 *Summary*	44 Data Integrity and Protection	
		22 Swapping: Policies		45 *Summary*	
		23 Case Study: VAX/VMS		46 *Dialogue*	
		24 *Summary*		47 Distributed Systems	
				48 Network File System (NFS)	
				49 Andrew File System (AFS)	
				50 *Summary*	

Review: Virtual Addresses

- Every address generated by a process is treated as a “virtual address” by the OS
- OS translates each virtual address to a DRAM address.

- The “crux” of the problem:
- How can the OS build the abstraction of a private address space while actually having many processes use physical memory?
- And doing so securely? And efficiently?

Dynamic Relocation with Base and Bounds



Pros/Cons of Base+bounds:

Advantages:

- ✔ Dynamic relocation is possible
- ✔ Processes are isolated – secure
- ✔ Inexpensive: 2 registers + some logic, ALU
- ✔ Fast: Compare, Add can be done in 2 clock cycles.

Disadvantages:

- ✘ Memory for each process has to be contiguous.
- ✘ Must allocate memory that may not be used.
- ✘ Sharing will need additional memory “segments” & logic
- ✘ Process address space may be smaller than necessary

Larger Address Spaces

Dynamic relocation with Base+Bounds works but it may be limiting the size of the address space and all that free space between the stack and the heap may be underutilized

The “crux” of the problem:

How can the OS support a large address space without “wasting” the free space between the stack and the heap?

Segmentation

Divide up a process address space into segments:

i. Code
ii. Stack
iii. Heap

Instead of having just one base+bounds pair for each process, use a pair of registers for each segment.

Place each segment in different areas of memory: no need for them to be contiguous!

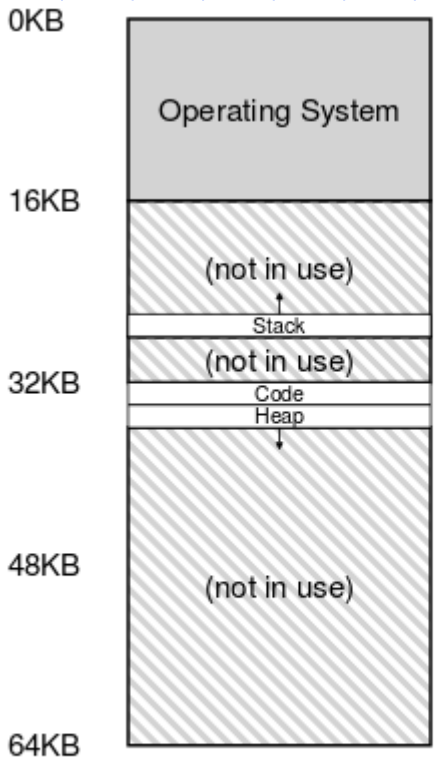
Segmentation

- An MMU with support for segmentation will need 3 pairs of base+bounds registers.

Segment Register Values

Segment	Base	Size
Code	32K	2K
Heap	34K	2K
Stack	28K	2K

The term “Segmentation Fault” or “Segmentation violation” comes from a memory access outside one of these segments to an illegal address.

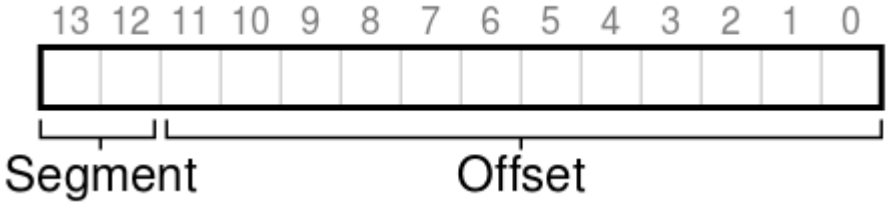


Placing Segments In Physical Memory

Segmented Address

With 3 segments, we will need 2 bits to choose.

Explicit approach: use the most significant 2 bits:



0 0 ==> Code segment

0 1 ==> Heap segment

1 0 ==> Stack segment

Rest of the virtual address (12 bits) → offset into segment.

Protection Bits and Shared Segments

By storing protection bits – i.e. information about whether a segment can be written to, the OS can figure out whether a segment can be safely shared.

Segment	Base	Size	Grows Positive?	Protection
Code	32K	2K	1	Read-Execute
Heap	34K	2K	1	Read-Write
Stack	28K	2K	0	Read-Write

Segment Register Values (with Protection)

The one physical Code segment may be safely shared between processes – executing the same program.

Fine-grained Segmentation

- So far we have considered having just a few segments in a process – a few coarse pieces
- Fine grained segmentation divides the address space into 100s or 1000s of segments, possibly a different number for each process

  → need a “Segment Table” full of the information in previous slide: Base + Bounds, protection bits, etc.

Q: What happens to a segmentation table during a context switch?

External Fragmentation

External fragmentation – physical memory becomes full of little “holes” of available memory

→ This is called “External Fragmentation”

We may have plenty of unused memory but it is all in small pieces and therefore unusable!