LECTURE 2.4
MEMORY ADDRESSING

COP4600

Dr. Matthew Gerber

3/16/2015

Memory Addressing

- Recall the types of memory addressing we’ve discussed
- The most interesting ones involve run-time binding with dynamic relocation
- Let’s start with the simple ones
  - Relocation Registers
  - Segmentation

Relocation Registers – Review

- Under ordinary relocation by registers, the processor has two memory management registers: the *relocation* (or *base*) register and the *limit* register
  - The base register is added to each logical address to obtain the corresponding physical address
  - The limit register is the size of the memory range that can be accessed
    - **IMPORTANT: The limit register <u>is not</u> the last physical <u>or</u> logical address that can be accessed!**
- Each process has a copy of these registers in its PCB
  - Are they in TCBs? Why or why not?

Relocation Registers – Example

Process	Base	Limit
P1	16384	2048
P2	4096	8192
P3	32768	1024

Process	Laddress	PAddress
P1	0	
P1	2000	
P1	2050	
P1	16384	
P2	128	
P2	2050	
P2	8100	
P2	8192	
P3	0	
P3	232	
P3	1020	

Relocation Registers – Example

Process	Base	Limit
P1	16384	2048
P2	4096	8192
P3	32768	1024

Process	Laddress	PAddress
P1	0	16384
P1	2000	18384
P1	2050	Error
P1	16384	Error
P2	128	4224
P2	2050	6146
P2	8100	12196
P2	8192	Error
P3	0	32768
P3	232	33000
P3	1020	33788

Segmentation - Review

- Like ordinary relocation except it allows each process to have multiple *segments*
    - Each segment has its own base and limit register
    - Used to separate “areas” of a process; typically code, data, heap and stack
- Each logical address has a segment and an *offset*
    - To obtain the physical address, the corresponding segment’s base is added to the offset
    - The corresponding limit register works the same way as before

Segmentation – Example

Process	Segment	Base	Limit
P1	1	16384	1024
	2	17408	256
	3	17664	512
	4	18176	256
P2	1	4096	2048
	2	20480	1024
	3	1024	2048
	4	512	512
P3	1	32768	1024
	2	0	128
	3	33792	1024
	4	128	256

Process	Segment	Offset	PAddress
P1	1	512	
P1	4	384	
P1	3	384	
P1	2	0	
P2	3	543	
P2	1	265	
P2	4	123	
P2	2	54	
P3	2	324	
P3	4	423	
P3	1	589	
P3	3	675	

Segmentation – Example

Process	Segment	Base	Limit
P1	1	16384	1024
	2	17408	256
	3	17664	512
	4	18176	256
P2	1	4096	2048
	2	20480	1024
	3	1024	2048
	4	512	512
P3	1	32768	1024
	2	0	128
	3	33792	1024
	4	128	256

Process	Segment	Offset	PAddress
P1	1	512	16896
P1	4	384	Error
P1	3	384	18048
P1	2	0	17408
P2	3	543	1567
P2	1	265	4361
P2	4	123	635
P2	2	54	20534
P3	2	324	Error
P3	4	423	Error
P3	1	589	33357
P3	3	675	34467

Paging - Review

- The next step in segmentation
- Break all of physical memory into *frames*
- Break all of a process's logical memory space into *pages*
- Processes' pages are loaded into frames
- Logical pages are mapped to physical frames with each and every memory access
- Physical frames therefore don't have to be contiguous in any way for logical pages to appear contiguous to their respective processes

Paged Addressing

- Segments use explicit tuples as their addresses; pages use implicit tuples
- The logical address is a single integer encoding a *page number* and an *offset*
- Recall integer division with remainder: For any integer a with integer divisor d, there are integers q and r so that

$$a = qd + r$$

- For paged addressing purposes, we decompose logical address a with the divisor as the page size, the quotient as the page number and the remainder as the offset:

$$a = (\text{Page Number})(\text{Page Size}) + (\text{Offset})$$

Paged Addressing

- The size of a page is always 2^n for some *n*
- This allows the computer to decompose paged addresses using bitwise operations instead of actual integer arithmetic
- The rightmost *n* bits encode the offset, and the rest of the bits on the left encode the page number
- With page size 512 and logical address 22,194, we find...

22,194 / 512 = 43 22,194 % 512 = 178

00000000000000000101011010110010

Page 43 Offset 178

Hint: If your calculator doesn't have a remainder function, multiply the fractional part by the page size

Paged Addressing

- Once we’ve obtained the page number and the offset, we’re ready to map the address
- We interrogate the *page table* for the physical frame corresponding to the logical page
- We switch out the frame for the page, and leave the offset
- So still considering logical address 22,194, and with page 43 mapped to frame 81, we reach physical address 41,650 by:

22,194 / 512 = 43 22,194 % 512 = 178

00000000000000000101011010110010

Page 43 Offset 178

Frame 81 Offset 178

00000000000000001010001010110010

81 * 512 = 41,472 41,472 + 178 = **41,650**

Paging – Example (Page size 512)

Proc	Page	Frame	Start
P1	0	93	
	1	123	
	2	Invalid	
P2	0	Invalid	
	1	934	
	2	14592	
	3	349	
P3	0	6	
	1	3	
	2	1285	
	20	Invalid	
	21	12	

Proc	LAddr	Page	Offset	PAddr
P1	0			
P1	511			
P1	512			
P1	1024			
P2	609			
P2	187			
P2	1512			
P2	2047			
P3	10751			
P3	10752			
P3	9730			
P3	0			

Paging – Example (Page size 512)

Proc	Page	Frame	Start
P1	0	93	47616
	1	123	62976
	2	Invalid	
P2	0	Invalid	
	1	934	478208
	2	14592	7471104
	3	349	178688
P3	0	6	3072
	1	3	1536
	2	1285	657920
	20	Invalid	
	21	12	6144

Proc	LAddr	Page	Offset	PAddr
P1	0	0	0	47616
P1	511	0	511	48127
P1	512	1	0	62976
P1	1024	2	0	Error
P2	609	1	97	478305
P2	187	0	187	Error
P2	1512	2	488	7471592
P2	2047	3	511	179199
P3	10751	20	511	Error
P3	10752	21	0	6144
P3	9730	19	2	Error
P3	0	0	0	3072

Virtual Memory - Review

- Adds swapping to paging
- Allows each process to have a truly arbitrary memory space
- Pages still have size 2^n
- The logical address is still single integer, with the rightmost *n* bits determining the offset and the rest of the bits on the left determining the page number
- To obtain the physical address, the *page table* is interrogated for the mapping from the logical page number to the physical frame, and the offset added to the result
- Attempting to access an unallocated or invalid page results in a *page fault*
- This can result in an error *or* in the need to swap in the page

Virtual Memory – Example (Page size 512)

Proc	Page	Frame	Start
P1	0	93	47616
	1	123	62976
	2	Invalid	DISK
P2	0	Invalid	ERROR
	1	934	478208
	2	14592	7471104
	3	349	178688
P3	0	6	3072
	1	3	1536
	2	1285	657920
	20	Invalid	ERROR
	21	12	6144

Proc	LAddr	Page	Offset	PAddr
P1	0	0	0	47616
P1	511	0	511	48127
P1	512	1	0	62976
P1	1024	2	0	
P2	609	1	97	478305
P2	187	0	187	
P2	1512	2	488	7471592
P2	2047	3	511	179199
P3	10751	20	511	
P3	10752	21	0	6144
P3	9730	19	2	
P3	0	0	0	3072

Virtual Memory – Example (Page size 512)

Proc	Page	Frame	Start
P1	0	93	47616
	1	123	62976
	2	Invalid	DISK
P2	0	Invalid	ERROR
	1	934	478208
	2	14592	7471104
	3	349	178688
P3	0	6	3072
	1	3	1536
	2	1285	657920
	20	Invalid	DISK
	21	12	6144

Proc	LAddr	Page	Offset	PAddr
P1	0	0	0	47616
P1	511	0	511	48127
P1	512	1	0	62976
P1	1024	2	0	SWAP
P2	609	1	97	478305
P2	187	0	187	ERROR
P2	1512	2	488	7471592
P2	2047	3	511	179199
P3	10751	20	511	SWAP
P3	10752	21	0	6144
P3	9730	19	2	ERROR
P3	0	0	0	3072

NEXT TIME: PAGE REPLACEMENT AND ACCESS TIME