UNIVERSITY of WISCONSIN-MADISON
Computer Sciences Department

CS 537
Introduction to Operating Systems

Andrea C. Arpaci-Dusseau
Remzi H. Arpaci-Dusseau

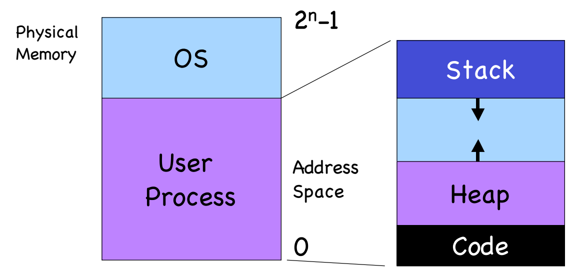
Memory Management

Questions answered in this lecture:

- How do processes share memory?
- What is static relocation?
- What is dynamic relocation?
- What is segmentation?

Motivation for Multiprogramming

Uniprogramming: One process runs at a time



Disadvantages:

- Only one process runs at a time
- Process can destroy OS

Multiprogramming Goals

Sharing
- Several processes coexist in main memory
- Cooperating processes can share portions of address space

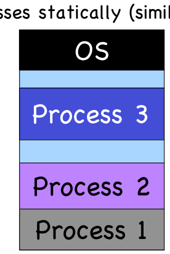
Transparency
- Processes are not aware that memory is shared
- Works regardless of number and/or location of processes

Protection
- Cannot corrupt OS or other processes
- Privacy: Cannot read data of other processes

Efficiency
- Do not waste CPU or memory resources
- Keep fragmentation low

Static Relocation

Goal: Allow transparent sharing - Each address space may be placed anywhere in memory
- OS finds free space for new process
- Modify addresses statically (similar to linker) when load process



Advantages
- Requires no hardware support

Discussion of Static Relocation

Disadvantages

- No protection
  - Process can destroy OS or other processes
  - No privacy
- Address space must be allocated contiguously
  - Allocate space for worst-case stack and heap
  - What type of fragmentation?
- Cannot move address space after it has been placed
  - May not be able to allocate new process
  - What type of fragmentation?

Dynamic Relocation

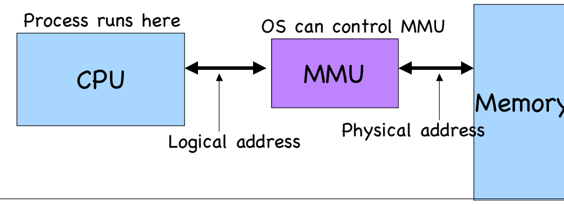
Goal: Protect processes from one another

Requires hardware support

- Memory Management Unit (MMU)

MMU dynamically changes process address at every memory reference

- Process generates logical or virtual addresses
- Memory hardware uses physical or real addresses



Hardware Support for Dynamic Relocation

Two operating modes

- Privileged (protected, kernel) mode: OS runs
  - When enter OS (trap, system calls, interrupts, exceptions)
  - Allows certain instructions to be executed
    - Can manipulate contents of MMU
  - Allows OS to access all of physical memory
- User mode: User processes run
  - Perform translation of logical address to physical address

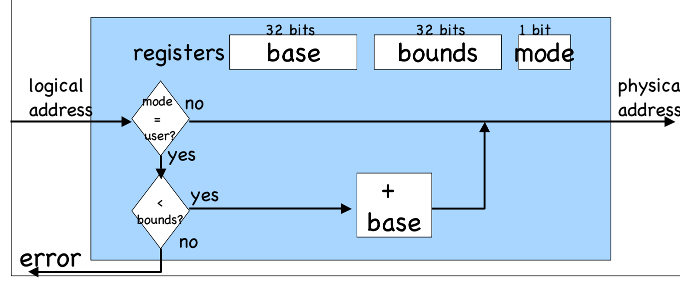
MMU contains base and bounds registers

- base: start location for address space
- bounds: size limit of address space

Implementation of Dynamic Relocation

Translation on every memory access of user process

- MMU compares logical address to bounds register
  - if logical address is greater, then generate error
- MMU adds base register to logical address to form physical address



Example of Dynamic Relocation

What are the physical addresses for the following 16-bit logical addresses?

Process 1: base: `0x4320`, bounds: `0x2220`

- `0x0000:`
- `0x1110:`
- `0x3000:`

Process 2: base: `0x8540`, bounds: `0x3330`

- `0x0000:`
- `0x1110:`
- `0x3000:`

Operating System

- `0x0000:`

Managing Processes with Base and Bounds

Context-switch

- Add base and bounds registers to PCB
- Steps
  - Change to privileged mode
  - Save base and bounds registers of old process
  - Load base and bounds registers of new process
  - Change to user mode and jump to new process

What if don't change base and bounds registers when switch?

Protection requirement

- User process cannot change base and bounds registers
- User process cannot change to privileged mode

Base and Bounds Discussion

Advantages

- Provides protection (both read and write) across address spaces
- Supports dynamic relocation
  - Can move address spaces
  - Why might you want to do this???
- Simple, inexpensive implementation
  - Few registers, little logic in MMU
- Fast
  - Add and compare can be done in parallel

Disadvantages

- Each process must be allocated contiguously in physical memory
  - Must allocate memory that may not be used by process
- No partial sharing: Cannot share limited parts of address space

Segmentation

Divide address space into logical segments

- Each segment corresponds to logical entity in address space
  - code, stack, heap

Each segment can independently:

- be placed separately in physical memory
- grow and shrink
- be protected (separate read/write/execute protection bits)

Segmented Addressing

How does process designate a particular segment?

- Use part of logical address
  - Top bits of logical address select segment
  - Low bits of logical address select offset within segment

What if small address space, not enough bits?

- Implicitly by type of memory reference
- Special registers

Segmentation Implementation

MMU contains Segment Table (per process)

- Each segment has own base and bounds, protection bits
- Example: 14 bit logical address, 4 segments

Segment	Base	Bounds	R W
0	0x2000	0x06ff	1 0
1	0x0000	0x04ff	1 0
2	0x3000	0x0fff	1 1
3	0x0000	0xffff	0 0

Translate logical addresses to physical addresses:

0x0240:
0x1108:
0x265c:
0x3002:

Discussion of Segmentation

Advantages

- Enables sparse allocation of address space
  - Stack and heap can grow independently
  - Heap: If no data on free list, dynamic memory allocator requests more from OS (e.g., UNIX: malloc calls sbrk())
  - Stack: OS recognizes reference outside legal segment, extends stack implicitly
- Different protection for different segments
  - Read-only status for code
- Enables sharing of selected segments
- Supports dynamic relocation of each segment

Disadvantages

- Each segment must be allocated contiguously
  - May not have sufficient physical memory for large segments