Lecture 8: Mechanism of Address Translation

Mythili Vutukuru
IIT Bombay
A simple example

• Consider a simple C function

```c
void func() {
    int x = 3000; //
    x = x + 3;
}
```

• It is compiled as follows

```
128: movl 0x0(%ebx), %eax ; load 0+ebx into eax
132: addl $0x03, %eax ; add 3 to eax register
135: movl %eax, 0x0(%ebx) ; store eax back to mem
```

• Virtual address space is setup by OS during process creation
Address Translation

- Simplified OS: places entire memory image in one chunk
- Need the following translation from VA to PA
  - 128 to 32896 (32KB + 128)
  - 1KB to 33 KB
  - 20KB? Error!
Who performs address translation?

- In this simple example, OS tells the hardware the base (starting address) and bound (total size of process) values.
- Memory hardware Memory Management Unit (MMU) calculates PA from VA
  \[
  \text{physical address} = \text{virtual address} + \text{base}
  \]
- MMU also checks if address is beyond bound.
- OS is not involved in every translation.
Role of hardware in translation

• CPU provides privileged mode of execution
• Instruction set has privileged instructions to set translation information (e.g., base, bound)
• Hardware (MMU) uses this information to perform translation on every memory access
• MMU generates faults and traps to OS when access is illegal (e.g., VA is out of bound)
Role of OS in translation

• OS maintains free list of memory
• Allocates space to process during creation (and when asked) and cleans up when done
• Maintains information of where space is allocated to each process (in PCB)
• Sets address translation information (e.g., base & bound) in hardware
• Updates this information upon context switch
• Handles traps due to illegal memory access
Segmentation

- Generalized base and bounds
- Each segment of memory image placed separately
- Multiple (base, bound) values stored in MMU
- Good for sparse address spaces
- But variable sized allocation leads to external fragmentation
  – Small holes in memory left between segments