Operating System (OS)
CS232

Memory Virtualization: Address Translation

Dr. Muhammad Mobeen Movania

Outlines

- Limited Direct Execution (LDE)
- Address Translation
- Memory Virtualization Assumptions
- Memory Accesses and Virtualization
- Dynamic Relocation and Address Translation Examples
- Hardware Support and OS issues
- Dynamic Relocation Issues
- Summary

Limited Direct Execution (LDE)

- OS allows the program to run directly on the hardware
  - OS intervenes only when
    - something bad happens like an exception or
    - the process's time slice expires raising a timer interrupt
- Why LDE?
  - Efficiency (sharing of memory and resources among processes)
  - Control (restrict process to its own memory)

Address Translation

- OS has to ensure that the virtualization of memory through address translation must provide
  - Efficiency (using hardware support)
  - Control (for isolation of process memory)
- The hardware transforms each memory access (e.g., an instruction fetch, load, or store), changing **virtual** address to a **physical** address
- OS must keeping information of used and free space for allocating processes

Memory Virtualization Assumptions

- A process address space lies contiguously in memory
  - Helps in accessing data quicker
- Size of a process address space is smaller than physical memory
  - Simpler to deal with
- All process address spaces are of same size
  - Access to a process is easier and more **direct**

Example

```
void func() {
    int x = 3000;
    x = x + 3;
    ...
```

ebx register contains 15KB which is (address of x in stack memory)

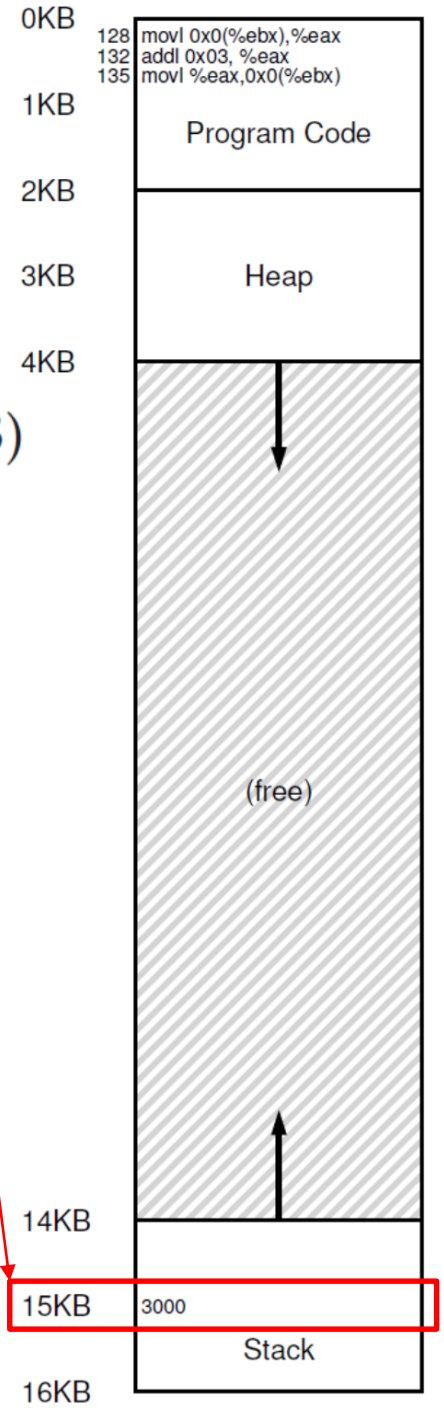
Read from address stored in ebx register

```
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
```

Write to address stored in ebx register

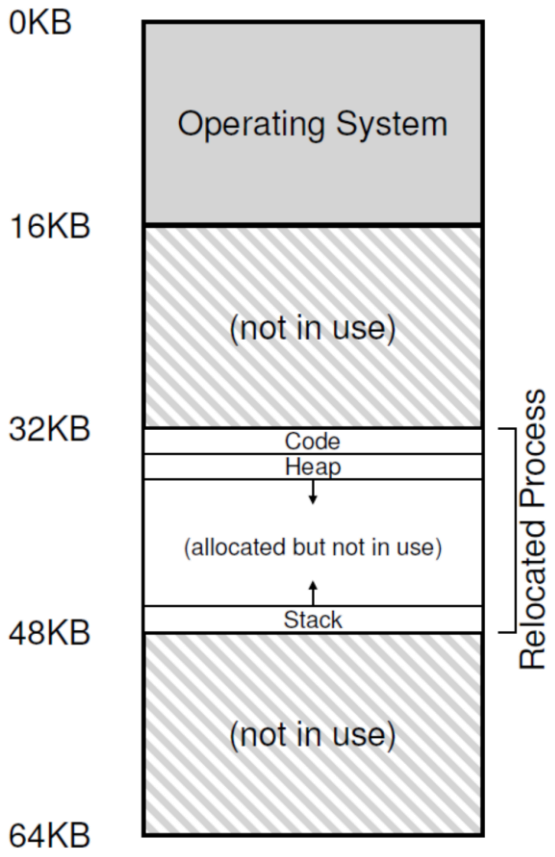
Memory Accesses

- Fetch instruction at address 128
- Execute this instruction (load from address 15 KB)
- Fetch instruction at address 132
- Execute this instruction (no memory reference)
- Fetch the instruction at address 135
- Execute this instruction (store to address 15 KB)



Virtualization

- From the process perspective, its address space starts at 0
- In reality the OS might not place the process image at that address
- How can the OS provide this illusion in a manner transparent to the process?



Dynamic relocation

- Two special registers in CPU
  - Base
  - Bound (limit)
- Every address generated by the program is translated by the hardware:

```
physical address = virtual address + base
```

- **Memory Management Unit (MMU)**: part of CPU that helps with address translation

Memory Management Unit (MMU)

- Base register
  - Contains the start of process address space in physical memory
  - Used in address translation
- Bound register
  - Can contain either:
    - The size of process address space, or
    - The final physical address of the process address space
  - Used in checking for illegal memory accesses

Address Translation Example

- Process with an address space of 4KB
  - Min Virtual Address: 0 bytes
  - Max Virtual Address: 4*1024=4096 bytes

Virtual address	**Physical Address**
0	16KB= 16384 bytes
1KB	17KB
3000 bytes	16KB+3000=19384 bytes
4400 bytes	16KB+4400=20784 bytes Fault(Out of bounds memory access)

H/W support we need for MM

Hardware Requirements	Notes
Privileged mode	*Needed to prevent user-mode processes from executing privileged operations*
Base/bounds registers	*Need pair of registers per CPU to support address translation and bounds checks*
Ability to translate virtual addresses and check if within bounds	*Circuitry to do translations and check limits; in this case, quite simple*
Privileged instruction(s) to update base/bounds	*OS must be able to set these values before letting a user program run*
Privileged instruction(s) to register exception handlers	*OS must be able to tell hardware what code to run if exception occurs*
Ability to raise exceptions	*When processes try to access privileged instructions or out-of-bounds memory*

OS support we need for MM

OS Requirements	Notes
Memory management	*Need to allocate memory for new processes;* *Reclaim memory from terminated processes;* *Generally manage memory via* ***free list***
Base/bounds management	*Must set base/bounds properly upon context switch*
Exception handling	*Code to run when exceptions arise;* *likely action is to terminate offending process*

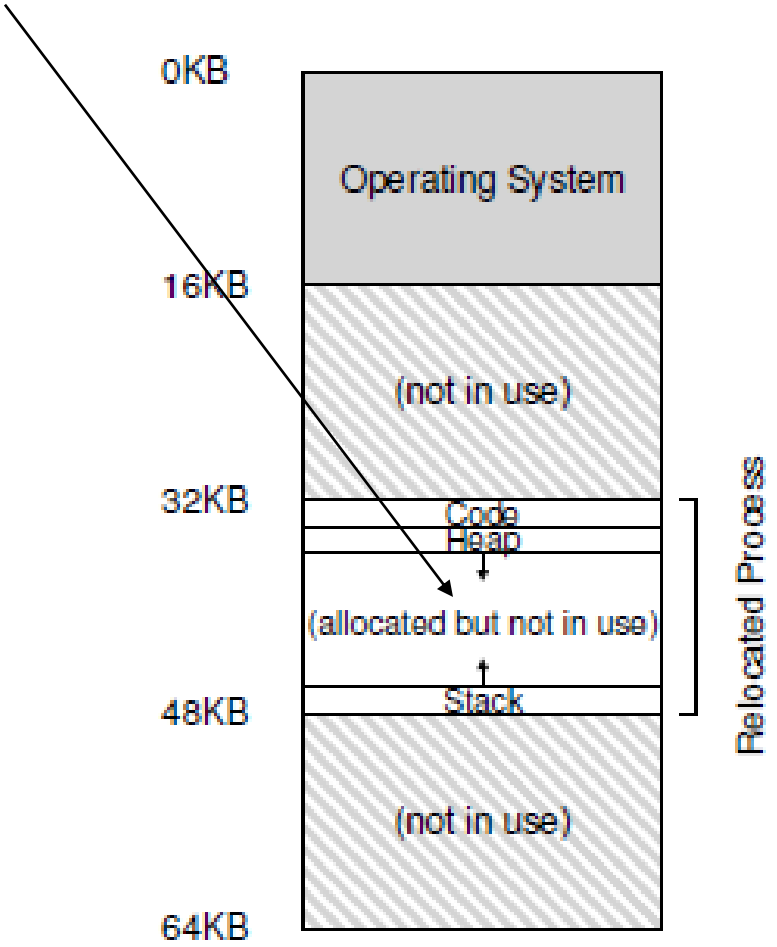
Can the OS move a process image in physical memory?

Operating System Responsibilites

- OS must
  - *keep track of free space* in a data structure called **free list**.
  - do *termination housekeeping* when a process is terminated (reclaim process's memory)
  - *save and restore* the base-and-bounds pair in **PCB** when it switches between processes.
  - provide **exception handlers** to handle exceptions
- OS just sets up the hardware and lets the process run directly on the CPU
  - only when the process **misbehaves** does the OS have to become involved.

Dynamic Relocation Issues

- Internal fragmentation



Summary

- We have extended the concept of limited direct execution with **address translation**
- OS can control each and every memory access ensuring the accesses stay within the bounds of the address space
- Efficiency depends on hardware support
- Address translation remains transparent to a process