Chapter 15: Address Translation

Adam Disney

THE UNIVERSITY OF
TENNESSEE
KNOXVILLE

Crux: How To Efficiently And Flexibly Virtualize Memory

THE CRUX:
HOW TO EFFICIENTLY AND FLEXIBLY VIRTUALIZE MEMORY

How can we build an efficient virtualization of memory? How do we provide the flexibility needed by applications? How do we maintain control over which memory locations an application can access, and thus ensure that application memory accesses are properly restricted? How do we do all of this efficiently?

Initial Assumptions

- User's address space is placed contiguously in physical memory
- User's address space fits in physical memory
- Users' address spaces are of equal size
- These are completely unrealistic but simplifies our model

Need for Translation

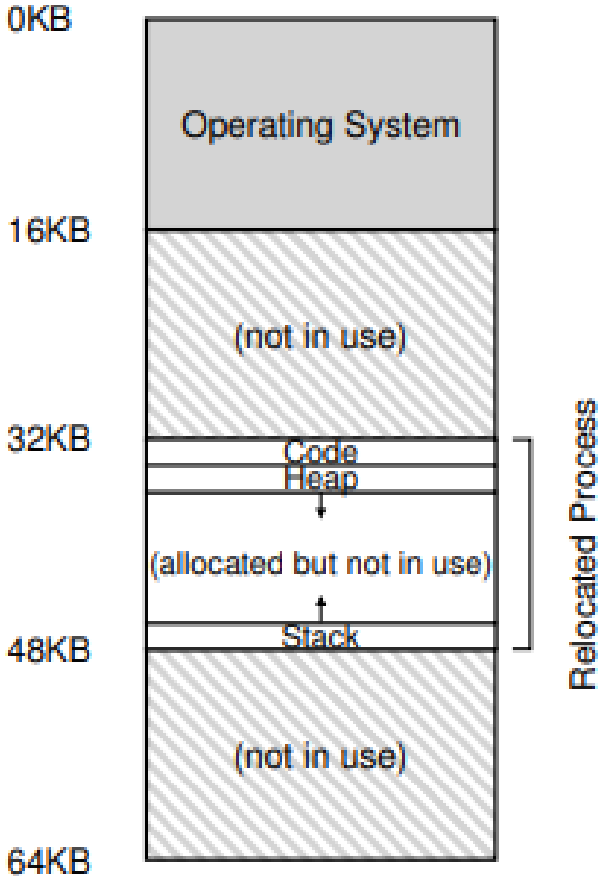

Figure 15.2: **Physical Memory with a Single Relocated Process**

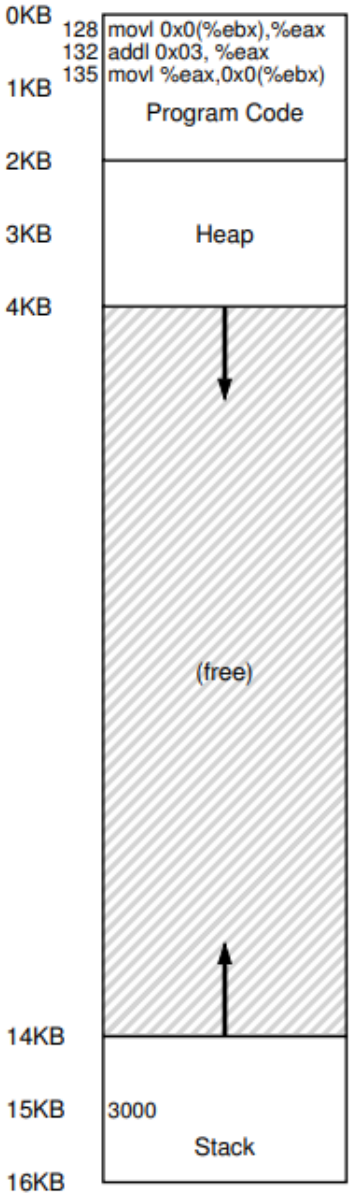

Figure 15.1: **A Process And Its Address Space**

Dynamic (Hardware-based) Relocation

- Base and bounds (dynamic relocation)
  - Two hardware registers, base and bounds (beginnings of MMU)
- Programs assume they're loaded at address 0x0
- When the OS places the program in memory, it sets the base/bounds for the program.
- Any virtual address provided is translated to physical with a simple formula.
  - Physical address = virtual address + base
- Bounds register prevents process from accessing outside its range

Need for Translation

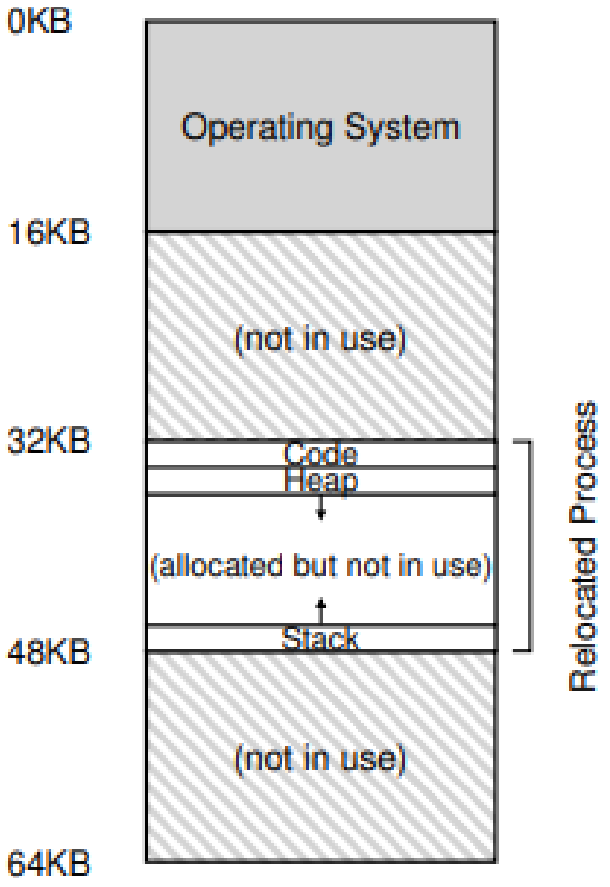

Figure 15.2: **Physical Memory with a Single Relocated Process**

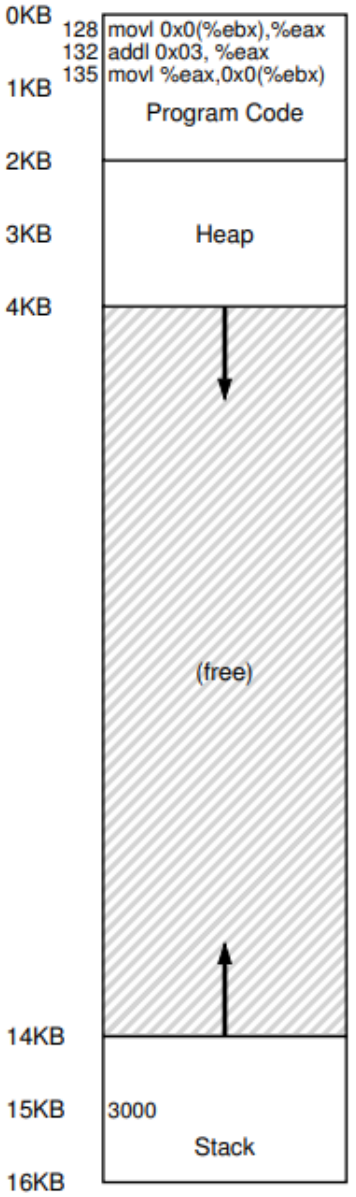

Figure 15.1: **A Process And Its Address Space**

?

Hardware Support So Far

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*

Figure 15.3: **Dynamic Relocation: Hardware Requirements**

Operating System Issues

- Needs to find space for new processes
  - Given our current constraints, this is easy
  - Variable sized address spaces would make this more complicated
- Needs to track free space
  - When a process is created/terminated, remove/add free space
- Extra steps in context switch
  - Now we must save and restore base/bounds
- Exception handlers
  - MMU says program tried to access out of bounds. What do we do?