Chapter 13 The Abstraction: Address Spaces

13.1 Early Systems

Only one process in memory at a time.

Needed to load one program.

Run it until finished.

And load and run another…

Problem:

- Takes a long time to load program into memory (from disk / tape) when switching
- Especially when size of programs grows large.

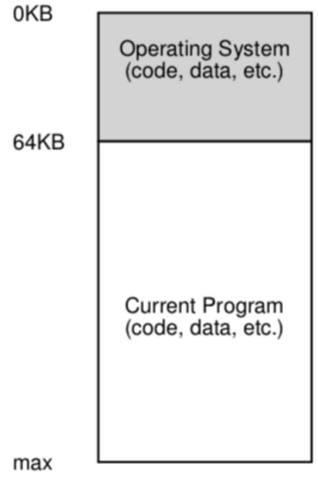

Figure 13.1: **Operating Systems: The Early Days**

13.2 Multiprogramming and Time Sharing

Idea:

- Leave the non-running process in memory
- Let each process have a part of the memory

Challenge: Protection

- Sharing memory of computer with other process
- Don't want another process to read or write our process's memory… (or the OS's memory)…

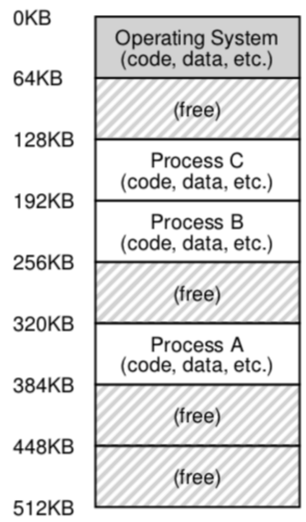

Figure 13.2: **Three Processes: Sharing Memory**

13.3 The Address Space

Abstraction:

Def. The Address Space:
The Running Program's view of the memory.

Q: What's the address space?

Code: Program's instructions

-

Stack: Keep track of

- Where the process is in the function call chain
- Allocates local variables of functions
- Pass parameters to functions
- Return values from functions
- Return address: where to continue execution after a func call

Heap: Dynamically allocated memory

- System call: malloc() in C
- Java/C++/Go:
  var j *job.Job
  id := j.ID()

  j := new(job.Job)
  id := j.ID()
  j = nil
  j = j2
  &cpu.CPU{} / &job.Job{}

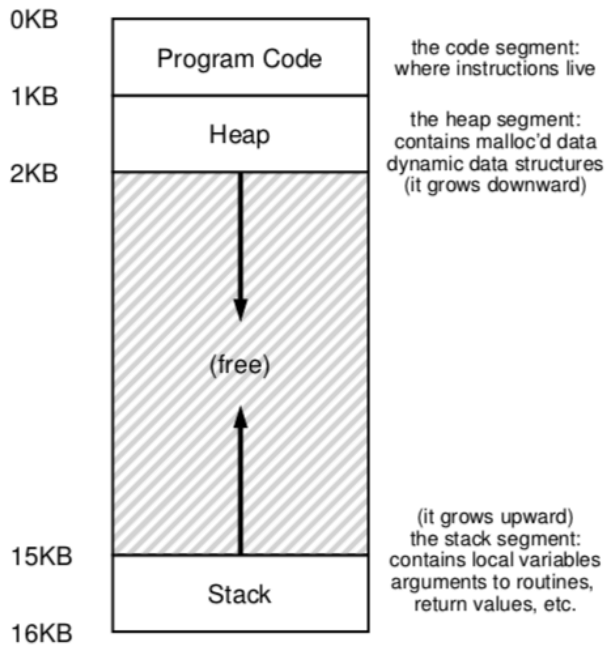

Figure 13.3: **An Example Address Space**

Code is static so won’t need more space as it runs…

The real program is loaded into an arbitrary physical address (location).

Q: How can the OS build the address space abstraction?

- Private for memory for each process
- Large address spaces (typically 32-bit or 48-bit)
- For multiple running programs
- Sharing a single physical memory

Ex. Process A (Fig 13.2)

- When process A tries to load address 0x0000 (virtual address)
- OS and HW: translating virtual address 0x0000 —> physical address 320KB.

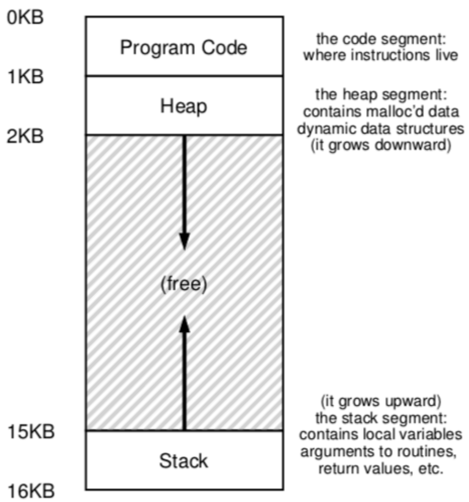

Figure 13.3: **An Example Address Space**

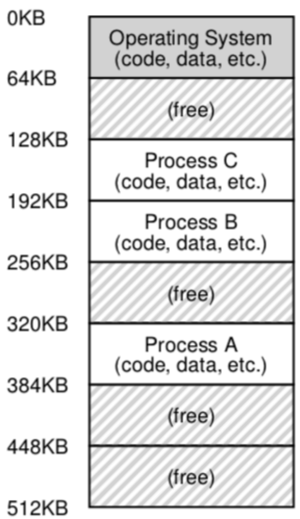

Figure 13.2: **Three Processes: Sharing Memory**

13.4 Goals

Major goal of **virtual memory (VM) system**:

- **Transparency** — the VM system should be invisible to the running program.
  - Invisible: Program shouldn't be aware that its memory is being virtualized
  - Easy: Program should behave as if it has its own private physical memory
- **Efficiency**
  - Time: should not make programs run much slower
  - Space: should not use much memory structures to support virtualization
- **Protection** — isolation property
  - Protect the processes from each other
  - Protect the OS from processes

In the following chapters:

- Mechanisms for virtualizing memory
- Policies for managing free space
- And policies for when to kick stuff out of memory when running low…
-