Operating System (OS)
CS232

Memory Virtualization

Dr. Muhammad Mobeen Movania

Outlines

- Memory Abstraction using Address Spaces
- Historical Perspectives
  - Early systems
  - Multiprogramming and Time Sharing
- Address Space
- Memory Abstraction Goals
- Summary

Memory Abstraction using Address Space

- Key Idea
  - Provide memory to a process through an abstraction of address space
- Why
  - It makes access to memory easier
  - A diverse range of memory hardware may be easily supported

Early Systems

- OS was a bunch of libraries
- Only a single user program ran
- Not much abstraction needed because of the simple architecture

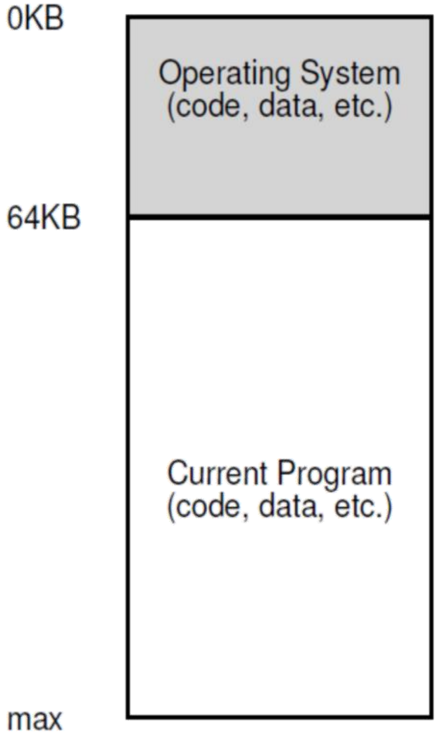


Figure 13.1: **Operating Systems: The Early Days**

Multiprogramming and Time Sharing

- Machines were expensive.
- Efficient use demanded sharing
  - b/w processes
  - b/w users
- With many users interactivity became important
- With time sharing, it is important to provide memory protection
  - ensures that a process cannot in anyway access some other process's memory

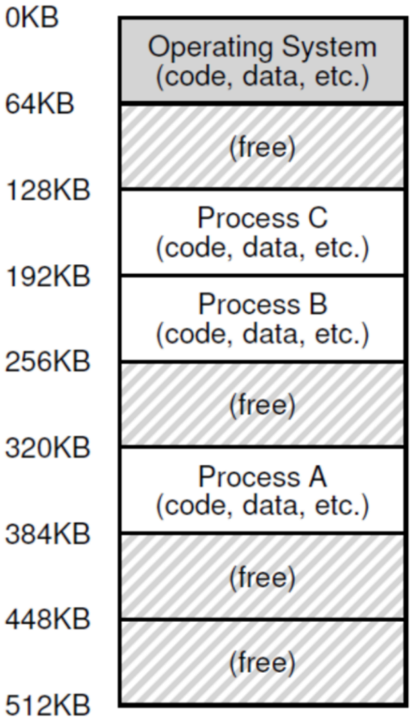

0KB
64KB
128KB
192KB
256KB
320KB
384KB
448KB
512KB
Operating System (code, data, etc.)
(free)
Process C (code, data, etc.)
Process B (code, data, etc.)
(free)
Process A (code, data, etc.)
(free)
(free)

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

Address Space

- It's an abstraction
  - a running program's view of memory
- The program thinks it's loaded at address 0
- Example of process A (Fig. 13.2), it's loaded at address 320KB

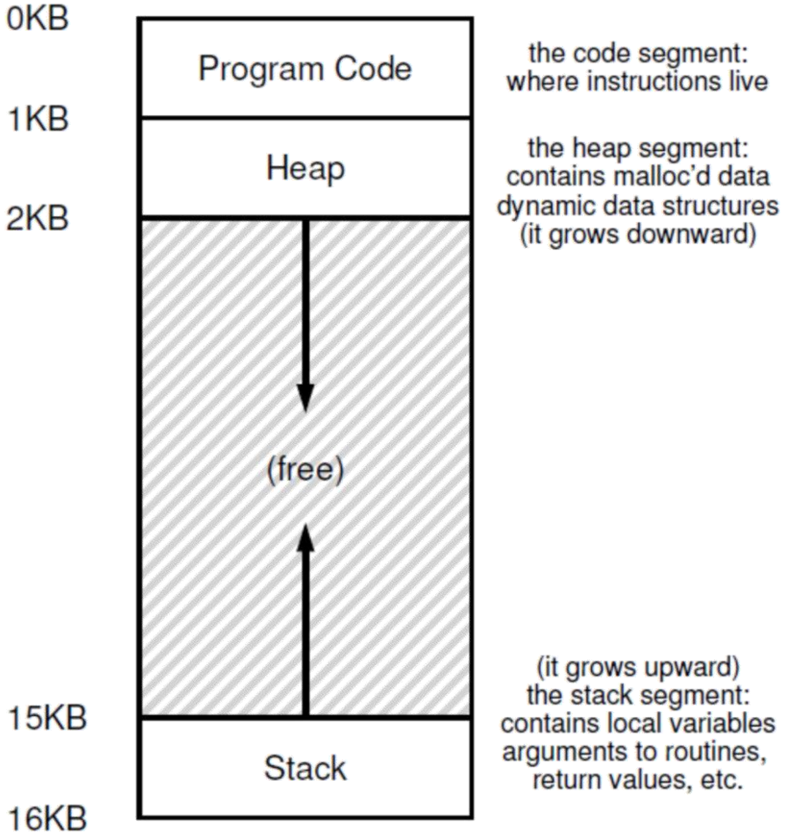

0KB
1KB
2KB
15KB
16KB
Program Code
Heap
(free)
Stack
the code segment: where instructions live
the heap segment: contains malloc'd data dynamic data structures (it grows downward)
(it grows upward) the stack segment: contains local variables arguments to routines, return values, etc.

Figure 13.3: **An Example Address Space**

Memory Abstraction Goals

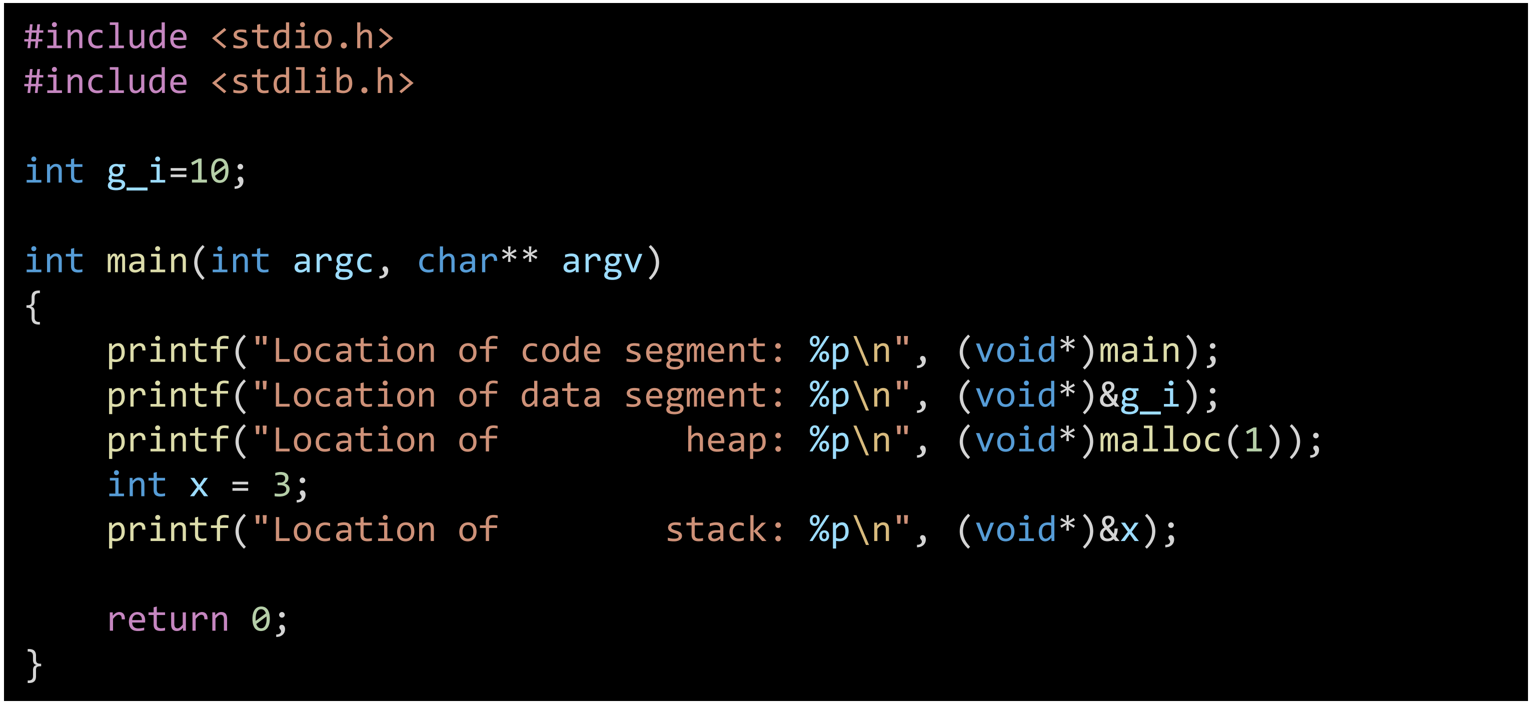
Transparency

Protection

Efficiency

Pointers store virtual addresses

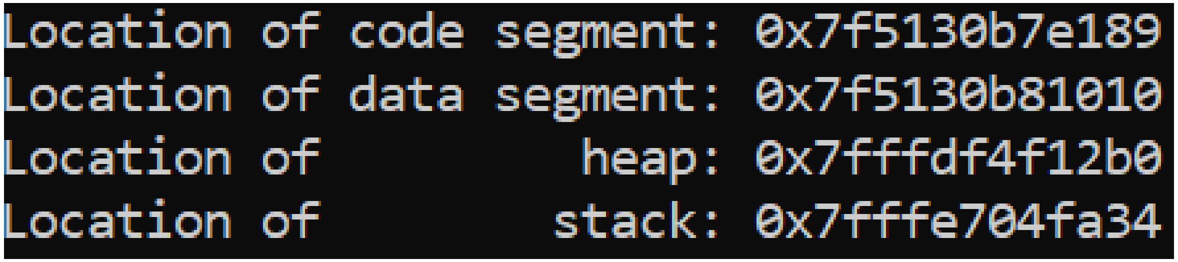
- All addresses in a program are virtual addresses



```c
#include <stdio.h>
#include <stdlib.h>

int g_i=10;

int main(int argc, char** argv)
{
    printf("Location of code segment: %p\n", (void*)main);
    printf("Location of data segment: %p\n", (void*)&g_i);
    printf("Location of         heap: %p\n", (void*)malloc(1));
    int x = 3;
    printf("Location of        stack: %p\n", (void*)&x);

    return 0;
}
```

Pointers store virtual addresses

- Program output on my laptop



```
Location of code segment: 0x7f5130b7e189
Location of data segment: 0x7f5130b81010
Location of         heap: 0x7fffdf4f12b0
Location of        stack: 0x7fffe704fa34
```

All of these are virtual addresses

Summary

- We introduced what memory abstraction means
- We talked about how OS uses the concept of address space
- We detailed the goals of memory virtualization which include
    - Transparency
    - Efficiency
    - Isolation
- All address in a user program are virtual addresses