Lecture 24: Virtual Memory

CSE 29: Systems Programming and Software Tools

Aaron Schulman (Shalev)

UCSDCSE

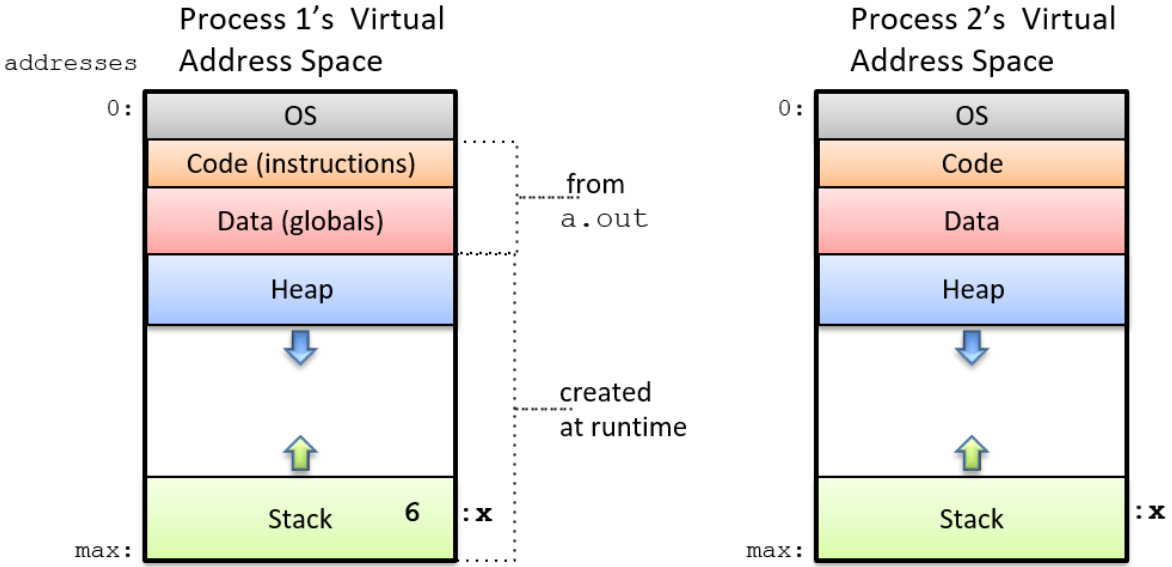
Computer Science and Engineering

Virtual Memory: Isolating process memory

- Each process has its own “Virtual Memory” space
- Every process thinks it has the full memory space (address range $0\text{-}2^{64}$)
- A process’s virtual memory state includes all of the **context** of the program
  - Stack
  - Heap
  - Code
  - Data (Globals)
- The Operating System and CPU work together to make Virtual Memory possible

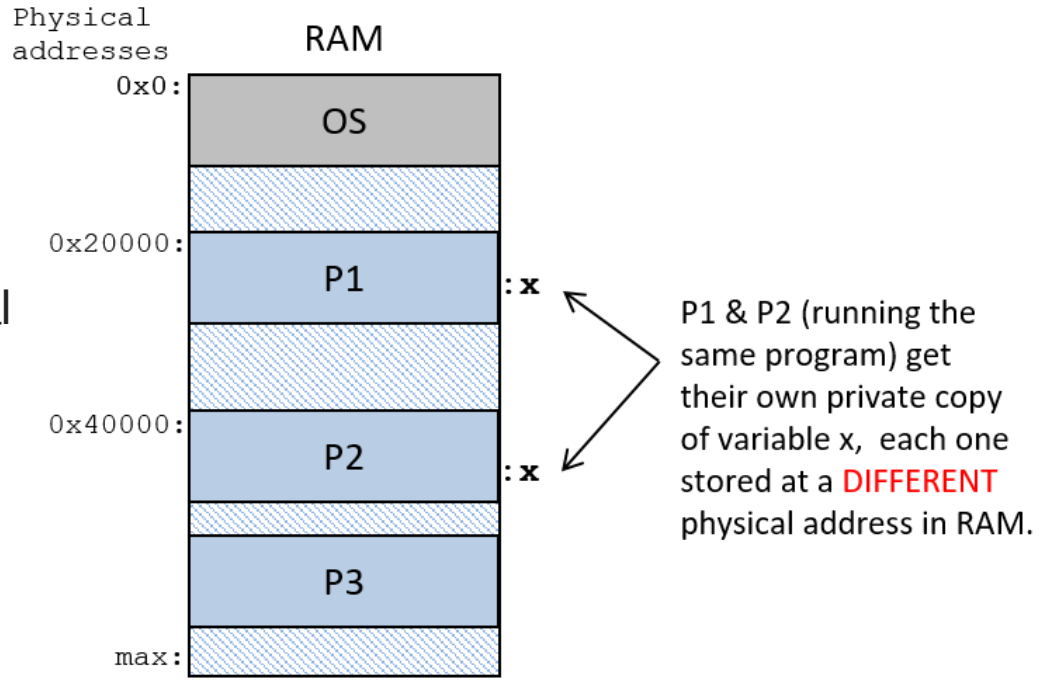
Two procs running same code are isolated

```c
/* a simple program */
#include <stdio.h>

int main(int argc, char* argv[]) {
    int x, y;

    printf("enter a value: ");
    scanf("%d", &y);

    if (y > 10) {
        x = y;
    } else {
        x = 6;
    }
    printf("x is %d\n", x);

    return 0;
}
```



How does virtual map to physical RAM?

- The OS and CPU work together to map *Virtual* to *Physical* addresses
- They automatically remap the Virtual Addresses to physical addresses
- Remapping transparently at runtime
  - You don't know this happens!



How does virtual memory make your life easier as a programmer?

- The pointers you use in your C code can be anywhere in the memory space
  - With some limitations, like you can not write into the code memory space
- If your program has a pointer with a hardcoded pointer address, it is portable
  - unsigned long* int_ptr = 0xFFFF0000;
  - Code will work on any computer, regardless of how big their physical memory is
- The translation is done transparently
  - You do not have to write any code to make it happen the OS and CPU just does it for you!

How does virtual memory work?

- MMU hardware (in CPU) translates Virtual to Physical addresses
- This happens for every *read* and *write* instruction
- The MMU has a table of mappings *per process*
- The table is called a “page table"