POINTERS: LESSON

Memory in C

2 min

For a program to execute, it must use some of the computer's resources such as CPU, RAM, IO, or other hardware. Deciding how and at what capacity to use these resources is the job of the underlying operating system. One of the most important of these resources is memory, more specifically: the temporary memory used for program execution which is called random access memory (RAM). When a program executes, the operating system reserves a section of the computer's physical RAM to be used exclusively by the program. The fundamental unit of this memory is a byte. As you learned previously, all variables are simply a collection of some number of bytes: an `int` is four bytes, a `double` is eight bytes, and so on. The allocated section of RAM is simply a block of however many bytes the program needs (if available, of course).

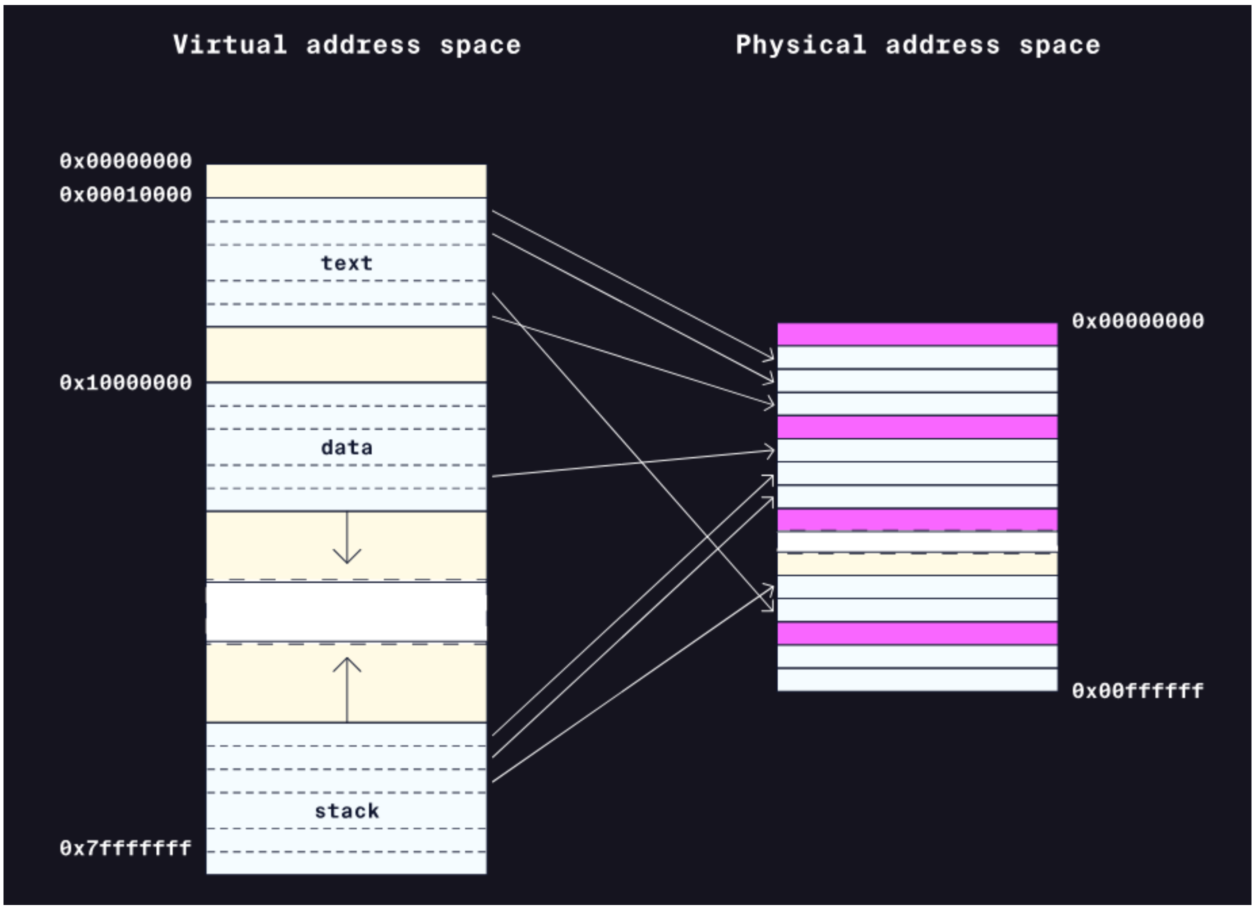
As you can see, in this block of memory to the right, every byte has an associated address numbered using the hexadecimal numbering system. For example, a byte of memory could be located at address 0x200 and the immediate byte next to it is located at address 0x201.

Every programming language has a different policy regarding the direct access and manipulation of a byte in memory; some allow it, some do not. C is one of the languages that allow such operations through the use of a *pointer*, and we will see how in this lesson.

At first glance, pointers may appear to be an overcomplicated way to work with variables, and for such simple examples meant to illustrate basic principles, this is true. However, the real value of pointers becomes apparent in more complex applications, such as working with data structures or embedded systems (think robots and microchips). Examples of these are well beyond the scope of this introductory tutorial. For now, just familiarize yourself with these ideas such that you may recall them when you need to!

Instructions

Move on when you're ready to learn more!


Virtual address space
Physical address space
0x00000000
0x00010000
text
0x10000000
data
stack
0x7fffffff
0x00000000
0x00ffffff