

1.5.2 Address Spaces

Every computer has some main memory that it uses to hold executing programs. In a very simple operating system, only one program at a time is in memory. To run a second program, the first one has to be removed and the second one placed in memory. This is known as swapping.

More sophisticated operating systems allow multiple programs to be in memory at the same time. To keep them from interfering with one another (and with the operating system), some kind of protection mechanism is needed. While the hardware must provide this mechanism, it is the operating system that controls it.

The above viewpoint is concerned with managing and protecting the computer's main memory. A different, but equally important, memory-related issue is managing the address space of the processes. Normally, each process has some set of addresses it can use, typically running from 0 up to some maximum. In the simplest case, the maximum amount of address space a process has is less than the main memory. In this way, a process can fill up its address space and there will be enough room in main memory to hold it all.

However, on many computers addresses are 32 or 64 bits, giving an address space of 2^{32} or 2^{64} bytes, respectively. What happens if a process has more address space than the computer has main memory and the process wants to use it all? In the first computers, such a process was just out of luck. Nowadays, a technique called virtual memory exists, as mentioned earlier, in which the operating system keeps part of the address space in main memory and part on SSD or disk and shuttles pieces back and forth between them as needed. In essence, the operating system creates the abstraction of an address space as the set of addresses a process may reference. The address space is decoupled from the machine's physical memory and may be either larger or smaller than the physical memory. Management of address spaces and physical memory forms an important part of what an operating system does, so all of Chap. 3 is devoted to this topic.

To avoid any confusion, it is worth stating explicitly that in this book, as in computer science in general, metric units are used instead of traditional English units (the furlong-stone-fortnight system). The principal metric prefixes are listed in Fig. 1-31. The prefixes are typically abbreviated by their first letters, with the units greater than 1 capitalized. Thus a 1-TB database occupies 10^{12} bytes of storage and a 100-psec (or 100-ps) clock ticks every 10^{-10} seconds. Since milli and micro both begin with the letter “m,” a choice had to be made. Normally, “m” is for milli and “ μ ” (the Greek letter mu) is for micro.

It is also worth pointing out that, in common industry practice, the units for measuring memory sizes have slightly different meanings. There kilo means 2^{10} (1024) rather than 10^3 (1000) because memories are always a power of two. Thus a 1-KB memory contains 1024 bytes, not 1000 bytes. Similarly, a 1-MB memory contains 2^{20} (1,048,576) bytes and a 1-GB memory contains 2^{30} (1,073,741,824) bytes. However, a 1-Kbps communication line transmits 1000 bits per second and a 1-Gbps LAN runs at 1,000,000,000 bits/sec because these speeds are not powers of two. Unfortunately, many people tend to mix up these two systems, especially for

Exp.	Explicit	Prefix	Exp.	Explicit	Prefix
10^{-3}	0.001	milli	10^3	1,000	Kilo
10^{-6}	0.000001	micro	10^6	1,000,000	Mega
10^{-9}	0.000000001	nano	10^9	1,000,000,000	Giga
10^{-12}	0.000000000001	pico	10^{12}	1,000,000,000,000	Tera
10^{-15}	0.000000000000001	femto	10^{15}	1,000,000,000,000,000	Peta
10^{-18}	0.000000000000000001	atto	10^{18}	1,000,000,000,000,000,000	Exa
10^{-21}	0.000000000000000000001	zepto	10^{21}	1,000,000,000,000,000,000,000	Zetta
10^{-24}	0.00000000000000000000001	yocto	10^{24}	1,000,000,000,000,000,000,000,000	Yotta

Figure 1-31. The principal metric prefixes.

Draw memory:

File is ADC.txt with 4 bytes of data.

Look at chart. Get the hex values.

Draw an address space of 16 bytes. The memory is byte-addressible.

List addressed low to high on left using 4 binary digits, 0000-1111.

Draw the same space for data on the right.

Put in the text.