Some Review Material

Important metric time prefixes

When dealing with times in operating systems, it's important to know the difference between milliseconds, microseconds, and nanoseconds.

- A *millisecond* is 1/1000 of a second (.001 seconds). There are 1000 milliseconds in a second.
- A *microsecond* is 1/1000000 of a second (.000001 seconds). There are one million milliseconds in a second.
- A *nanosecond* is 1/1000000000 of a second (.000000001 seconds). There are one billion nanoseconds in a second.

Each of the prefixes above is a factor of a 1000 away from the previous one. For instance, there are 1000 microseconds in a millisecond and 1000 nanoseconds in a microsecond. Most processors run at a clock speed of a few gigahertz. That means that means that they can do a few billion simple instructions, like adding integers, in a second. In other words, each of those instructions runs in a few tenths of a nanosecond. Most timing things we will be interested in with operating systems will happen at the microsecond or millisecond range.

Important metric size prefixes

Storage size is typically measured in bytes. Here are important metric prefixes for storage:

- A *kilobyte* is 1000 bytes.
- A *megabyte* is 1,000,000 bytes.
- A *gigabyte* is 1,000,000,000 bytes.
- A *terabyte* is 1,000,000,000,000 bytes.

Each of the units above is a factor of a thousand greater than the previous. For instance, there are 1000 kilobytes in a megabyte and a 1000 megabytes in a gigabyte. In rare cases you may see even larger prefixes, specifically *peta* and *exa*, which are the next two after *tera*. These sometimes appear when talking about the amounts of data transferred over a busy network over a period of time.

CPUs

The *central processing unit* (CPU) is the key component of any computer. Without one, you really don't have much of a computer. Its job is to run instructions which are written in what's called *machine language*. *Assembly language* is a human-readable version of machine language. Most programs are written in a higher level programming language and *compiled* into machine language.[1] Below is a simple program written in the C programming language.

```
#include <stdio.h>
int main() {
    int x = 3;
    int y = 4;
    int z = x + y;
```

[1] For some languages, like Python and Java, there is an intermediate program that sits in between.

```
    printf("Sum is %d\n", z);
    return 0;
}
```

Here is the result of compiling it and running it through the Linux *objdump* command to spit out the machine and assembly language code. The machine and assembly language code shown here are for x86 processors. Different architectures have their own assembly and machine languages.

```
1139:       55                      push   rbp
113a:       48 89 e5                mov    rbp,rsp
113d:       48 83 ec 10             sub    rsp,0x10
1141:       c7 45 fc 03 00 00 00    mov    DWORD PTR [rbp-0x4],0x3
1148:       c7 45 f8 04 00 00 00    mov    DWORD PTR [rbp-0x8],0x4
114f:       8b 55 fc                mov    edx,DWORD PTR [rbp-0x4]
1152:       8b 45 f8                mov    eax,DWORD PTR [rbp-0x8]
1155:       01 d0                   add    eax,edx
1157:       89 45 f4                mov    DWORD PTR [rbp-0xc],eax
115a:       8b 45 f4                mov    eax,DWORD PTR [rbp-0xc]
115d:       89 c6                   mov    esi,eax
115f:       48 8d 05 9e 0e 00 00    lea    rax,[rip+0xe9e]        #
1166:       48 89 c7                mov    rdi,rax
1169:       b8 00 00 00 00          mov    eax,0x0
116e:       e8 bd fe ff ff          call   1030 <printf@plt>
1173:       b8 00 00 00 00          mov    eax,0x0
1178:       c9                      leave
1179:       c3                      ret
117a:       66 0f 1f 44 00 00       nop    WORD PTR [rax+rax*1+0x0]
```

The first column are line numbers. The middle column contains the machine language code. Note that it is a purely numeric language, shown here in hexadecimal. Each machine language command corresponds directly to an assembly language command shown in the right column. The instructions are things like *mov* (which moves a value into a register) and *add* (which adds two values and stores the result in a register).

Memory

Here is a little about types of memory on a system:

- RAM — Most of a program's data is stored in RAM.
- Cache — Cache memory used to hold frequently-accessed items. Cache memory is close to the processor core, and is considerably more limited in size than RAM. Part of this is because cache memory is expensive to manufacture, and part of it is that its speed depends on its size. The larger the cache, the longer it takes to locate an item in it. There are various algorithms run by a specific hardware controller that determine which items get kicked out of the cache when a new item goes in so as to keep the cache operating efficiently. Cache memory considerably speeds up programs, and various operating system choices as well as various programming techniques can affect it. Generally, anything that causes the cache to hold values that are not relevant to the currently running program can cause serious slowdowns.
- Registers — Registers are small units of memory that sit on the CPU. They are the fastest memory available, but they are few in number, typically a few dozen to maybe a few hundred, depending on the architecture. Most CPU instructions involve doing something with values in registers. Some registers are general purpose registers that can be used for many purposes. Others have very particular purposes. For instance the *program counter* (also known as the instruction pointer) holds the memory location of the current instruction being executed by the processor. Other registers hold flags that contain important information about the state of the system.