**The point of this document:**

I’m actually not so sure. I think the point is to explain *why* xv6 does stuff, in broad strokes, as a complement to the explanations of the actual *code* of xv6.

**Super duper important note:**

Often, I’ll refer to things that haven’t been explained yet. In order to make this clear, these terms will be marked.

**Important-ish note:**

These explanations are not full. They may not even be accurate. I’m just jotting these down during class, hoping it more or less gives a picture of what’s going on.

Anyways, here we go.

**Kernel**

When I say “kernel”, I mean the operating system. This is the mother of all programs that hogs the computer to itself and manages all the other programs and how they access the computer resources (such as memory, registers, time (that is, who runs when), etc.).

The kernel also keeps control over the “mode” bit (in the PS register), which marks whether current commands are running in user-mode (and therefore can’t do all kinds of privileged stuff) or kernel-mode (and therefore can do whatever it wants).

**Boot**

The processor (that is, the actual computer) does not know what operating system is installed, what it looks like, or how large it is.

So how can the processor load xv6 into the memory?

The processor PC register points - by default - to a certain memory place in the ROM, which contains a simple program that:

1. Copies the very first block (512 bytes. AKA the boot block) from the disc to the memory
2. Sets the PC to point to the beginning of the newly-copies data

So what?

Every operating system needs to make sure that its first 512 bytes are a small program (it has to be small; it’s only 512 bytes!) that loads the rest of the operating system to the memory and sets the PC to whatever place it needs to be in. If 512 bytes aren’t enough for this, the program can actually call a slightly larger program that can load and set up more.

In short:

1. PC points to hard-coded ROM code
2. ROM code loads beginning of OS code
3. Beginning of OS code loads the rest of the code
4. Now hear the word of the Lord!

**Processes**

A process is the running of a program, including the program’ state and data. The state includes such things as:

* Memory the program occupied
* Memory contents
* Register values
* Files
* Kernel structures

This is managed by the kernel. The kernel has a simple data structure for each process, organized in some list. The kernel juggles between the processes, using a context switch, which:

1. Saves state of old process to memory
2. Loads state of new process from memory

Context switch can be preemptive (i.e. the kernel decides “next guy’s turn!”, using the scheduler) or non-preemptive (i.e. the hardware itself decides). In non-preemptive, it is asked of programmer to make calls to the kernel once in a while, in order to let the kernel choose to let the next guy run.

Xv6 is preemptive.

Processes are created by the kernel, after another process asks it to. Therefore, the kernel needs to run the first process itself, in order to create someone who will ask for new processes to be created.

**Fork()**

Every process has a process ID (or **pid** for short).

A process can call the kernel to do fork(), which creates a new process, which is entirely identical to the parent process (registers, memory and everything). The only differences are:

* The pid
* The value returned from fork:
  + In the new process - 0
  + In the parent process - the new pid
  + In case of failure - some negative error code

**Exec()**

Fork() creates a new process, and leaves the parent running. Exec(), on the other hand, replaces the process’s program with a new program. It’s still the same process, but with new code (and variables, stack, etc.). Registers, pid, etc. remain the same.

It is common practice to have the child of fork call exec after making sure it is the child. So why not just make a single function that does both fork and exec together? There is a brilliant explanation for this, and I don’t remember what it is.

**Process termination**

Trigger warning: sad stuff ahead. And also zombies.

A process will be terminated if (and only if) one of the following happens:

1. The process invokes exit()
2. Some other process invokes kill() with its pid
3. The process generates some exception

Note that kill does not actually terminate the process. What it does is leave a mark of “You need to kill yourself” on the process, and it’ll be the process itself that commits suicide (after it starts running again, when the scheduler loads it).

Note also that not any process can kill any other process. The kernel makes sure of that.

Once killed, the process’s resources (memory, etc.) are not released yet, until its parent (that is, the process which called for its creation) allows this to happen. A process that killed itself but whose parent did not acknowledge this is called a zombie.

In order to a parent to “acknowledge” its child’s termination, it needs to call wait(). When it calls wait(), it will not continue until *one of* its children exists, and then it will continue. If there are a few children, the parent will need to call wait once for each child process.

Wait returns the pid of the exited process.

What happens if a parent process exits before its children? Its children become orphans, and the The First Process (whose pid is 1) will make them His children.

**System calls**

Many commands will only run if the “mode” bit is set to kernel-mode. However, all processes run on user-mode only; xv6 makes sure of that.

In order to run a privileged command, a process must ask the kernel (which runs in kernel-mode, of course) to carry out the command. This “asking” is called a system call.

Here’s an example of system call on Linux with x86 processor:

movl flags(%esp),%ecx

lea name(%esp),%ebx

movl $5,%eax // loads the value 5 into eax register,  
 // which is the command “open” in Linux

int $128 // invokes system call. Always 128!

// eax should now contain success code

The kernel has a vector with a bunch of pointers to functions (Yay, pointers!).