Getting Started in OS Development

It seems that most people really want to write an OS but they don't know where to start and they don't know what programming lanuages they need to know.

Well, when I started my OS I was pretty much the same way, so I'm writting this tutorial on getting started in OS development.

I've written this in Q&A format for easier understanding.

Q: So what languages do I need to know to make my own OS?

A: You should at least know assembly(ASM) language because you will need it to write your bootsector(more on this in the next question) and quite a few other things. I suggest that you also learn C/C++, because it's a much more understandable language compared to ASM. You can then use both C/C++ and ASM in your OS.
You might consider writting your OS in Pascal and ASM instead of C/C++ and ASM, but I suggest C/C++ and ASM.(note that there are some special problems with C++ and OS dev)

Q: What is a *bootsector*?

A: A bootsector is a small program that is stored at the very start of a floppy disk and/or hard drive. It is responsible for finding your kernel(the MAIN part of you OS that does most everything), loading it into memory, and running it.

Q: Do I already have to have an OS to make an OS?

A: Yes. Most people use Windows and/or Linux for OS development.

Q: What compiler(s) should I use?

A: For assembly lanuage, I suggest NASM because it's free, has good documentation, outputs to many file formats, is availible for most OS'es, source code for it is availible, and most assembly example found on OS dev sites need it. For C/C++ I suggest gcc under Linux and DJGPP under Windows(both are compatible with the other).

Q: How do I make my OS bootsector bootable?

A: You will need to output it to a flat binary file. For NASM use `-f bin`. If you are using C/C++ you will need to use a linker to do this(see next question).

Q: What's a linker and how do use it for OS development?

A: A linker takes the output file(s) that a compiler(s) generates and links it into one file. For OS dev, you will normally want to link your files into a flat binary file. For LD(the linker that is most often used with DJGPP and gcc) use the option `--oformat binary`. If you use LD you should use a linker script for better control of linking. See My suggestions for making your OS.

Q: What is Real Mode?

A: *Real Mode* is the state that a x86(386, 486, Pentium, etc) is in when you first turn on your PC. Real Mode has several disadvantiges:

1. It lacks *Protection*, the ability to keep programs from messing around and crashing each other.
2. You may only have one task.
3. You are limited to 1Meg of address space.
4. You must use *segmentation* to access memory.

Q: What is Protected Mode?

A: *Protected Mode*(also known as PMode) is a step up from Real Mode. Protected Mode has several advanteges over Real Mode:

1. Protected Mode has *Protection*, the ability to keep programs from messing around and crashing each other(this is how Proctected Mode got its name).
2. You may only have several tasks(like Windows and Linux).
3. You have 4Gig, yes 4 gigabytes, of address space(once the A20-Gate is enabled, see next question).
4. You can use *paging* to access memory.

If it isn't already obvious, your OS should run in Protected Mode unless you have a very specific reason to use Real Mode.

Q: Q: What is that A20-Gate?

A: The 8086 microprocessor(the processor that came before the 286, 386, 486, Pentium, etc) could only access 1MB of address space. For that address range, 20 address bits(A0 - A19) are sufficent. Then, Intel invented the 80286(often called just the "286"). The 80286 could access up to 16MB of address space, but the 80286 had to be compatible with the 8086(programs that were made for the 8086 had to be able to run on the 80286). What was the solution? To give the 80286 more address gates for the address range, but some programs depended on the fact, that the addresses wrapped around at 1MB:

Let's see it in the way the 8086 saw it: You have the address 11111111111111111111 in dual system and add 1 to it. The result is 0 as you don't have a 21th bit (A20!). But the 80286 actually had more bits.

To solve the problem, Intel made the 80286 disable the A20-Gate, and the limit the address range to 20 bits.

So, to access more than 1MB memory, we need to enable the A20-Gate.

Q: What is a selector?

A: A selector(sometimes also called a descriptor) is a definition of a segment. It contains a description of basic segment attributes, like the segment base (it's starting address) and the segments limit (it's length). Each selector is 64 bits long. More about them in the description of a GDT and LDT.

Q: What is a GDT?

A: A gdt is a table containing selectors. Basically it provides a starting address to address all selectors.

Q: What is an interrupt?

A: An interrupt is a signal that interrupts the current task that the CPU is doing. It tells the CPU that an important event has arrived, for example hardware interrupts, and that the CPU needs to stop the current task and run the proper interrupt handler. The interrupt handler is a special function which is designed to handle a specific interrupt(like a key press from the keyboard). As interrupts have special properties, for example they can be used to switch from ring 3 to ring 0, they might be used in operating systems as a way to do system calls. Linux system calls are handled via interrupts.

Q: What is PIC?

A: The PIC, the Programmable Interrupt Controller, is a device which collects hardware interrupt requests and signals the cpu that an interrupt has occured. When this happens, both the PIC and CPU communicate about which request has occured and the cpu will stop the current task and execute the the interrupt handler for that request.

Q: What is PIT?

A: The PIT, Programmable Interrupt Timer, is a programmable(you can change the speed of it) hardware timer that fires off an interrupt when it goes off(IRQ 0). It is mainly used for task schedualing.

Q: In what order should I make my OS in?

A: Well, you could start just about anywhere, but you would probably end up with dozens of bugs. I suggest that you start by making a bootsector, then making a basic kernel, and then add on to your kernel.

Q: What are the basic steps for setting up PMode at boot time?

A:

1. Disable interrupts so nothing gets messed up.
2. Setup a GDT.
3. Enable the A20.
4. Set the first bit of CR0 to 1.
5. Load the segment registers with correct values from the GDT.
6. Load CS and EIP with correct values by doing a far jump.
7. You are now in PMode

This is covered in more detail in Protected Mode by Chris Giese.

Q: My question isn't answered here. What do I do?

A: You can email it to me and I'll try to answer it and add it on to this(this tutorial is far from done).

This tutorial is written by K.J. and Joachim Nock. Updated 2002.11.20 by K.J.