## **The kernel**

Our C kernel will just print an '*X*' on the top left corner of the screen. Go ahead and open **kernel.c**.

You will notice a dummy function that does nothing. That function will force us to create a kernel entry routine which does not point to byte 0x0 in our kernel, but to an actual label which we know that launches it. In our case, function main().

**i386-elf-gcc -ffreestanding -c kernel.c -o kernel.o**

That routine is coded on kernel\_entry.asm. Read it and you will learn how to use [extern] declarations in assembly. To compile this file, instead of generating a binary, we will generate an elf format file which will be linked with kernel.o

**nasm kernel\_entry.asm -f elf -o kernel\_entry.o**

## **The linker**

A linker is a very powerful tool and we only started to benefit from it.

To link both object files into a single binary kernel and resolve label references, run:

**i386-elf-ld -o kernel.bin -Ttext 0x1000 kernel\_entry.o kernel.o --oformat binary**

Notice how our kernel will be placed not at 0x0 in memory, but at 0x1000. The bootsector will need to know this address too.

## **The bootsector**

It is very similar to the one in lesson 10. Open bootsect.asm and examine the code. Actually, if you remove all the lines used to print messages on the screen, it accounts to a couple dozen lines.

Compile it with **nasm bootsect.asm -f bin -o bootsect.bin**

## **Putting it all together**

Now what? We have two separate files for the bootsector and the kernel?

Can't we just "link" them together into a single file? Yes, we can, and it's easy, just concatenate them:

**cat bootsect.bin kernel.bin > os-image.bin**

## **Run!**

You can now run **os-image.bin** with qemu.

Remember that if you find disk load errors you may need to play with the disk numbers or qemu parameters (floppy = 0x0, hdd = 0x80). I usually use **qemu-system-i386 -fda os-image.bin**

You will see four messages:

* "Started in 16-bit Real Mode"
* "Loading kernel into memory"
* (Top left) "Landed in 32-bit Protected Mode"
* (Top left, overwriting previous message) "X"

Congratulations!

## **Makefile**

As a last step, we will tidy up the compilation process with a Makefile. Open the Makefile script and examine its contents. If you don't know what a Makefile is, now is a good time to Google and learn it, as this will save us a lot of time in the future.