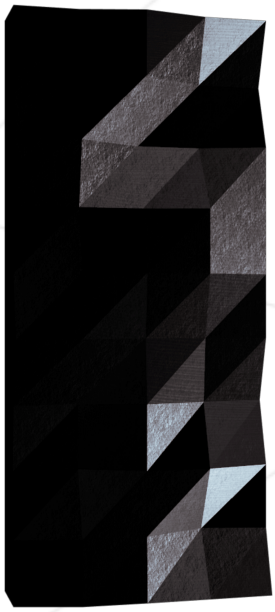

KFS_1

Grub, boot and screen

Louis Solofrizzo louis@ne02ptzero.me
42 Staff pedago@42.fr

Summary: The real code !

Version: 1

Contents

Chapter I

Introduction

Welcome to the first `Kernel from Scratch` project.

Finally some real coding, I'm pretty sure you were annoyed by the Linux Kernel project. In the `Kernel from Scratch` subjects you are going to write a kernel from scratch (no way ?), without any existing software, API, or such.

Once you will be done with those projects and standing on the shoulder of your non-revolutionnary-OS, you will be in right to laugh at those ordinary Javascript developers, who obviously, don't know anything about coding.

On a more serious note, those kernel programming skills are not quite spread in the IT world, so **take your time** to understand each and every different points in those projects. One does not simply consider himself a 'Kernel developer' if one only knows things about drivers or syscalls, like a sysadmin only knows things about iptables or softwares installation... It's a package of skills.

A word about the code itself : the `Kernel From Scratch` is divided into many projects, each one dealing with a specific aspect of kernel programming. All of those projects are linked together. So when you'll code those amazing features, keep in mind that your kernel must be flexible and that functions must easily fit in. Half the time you'll spend on these projects will be adding links between different aspects of your Kernel.

For instance, you must write memory code before processus & execution code. But processus must use memory, right ? So, you gotta link those two ! So keep your code *clean*, and your internal API simple.

Good luck, and most of all, have fun :)

Chapter II

Goals

At the end of this subject, you will have:

- A kernel you can boot via `GRUB`
- An `ASM` bootable base
- A basic kernel library, with basics functions and types
- Some basic code to print some stuff on the screen
- A basic "Hello world" kernel

Actually, that's not that much code. This subject is an introduction to kernel development. A (wo)man must learn.

Chapter III

General instructions

III.1 Code and Execution

III.1.1 Emulation

The following part is not mandatory, you are free to use any virtual manager you want to, however, i suggest you to use `KVM`. It's a `Kernel Virtual Manager`, and have advanced execution and debugs functions. All of the examples below will use `KVM`.

III.1.2 Language

All of `Kernel from Scratch` subjects have no constraints on the language you want to use.
C language is not mandatory at all, you can use any language. However, keep in mind that all language are not kernel friendly. So... Yes, you could code a Kernel in javascript, but are you sure it's a good idea ?
Also, a lot of example on the documentations are in C, so remember that you will do 'code translation' all the time if you choose a different language.
Furthermore, all of the features of a language cannot be used in a basic kernel.
Let's take C++ for example:
C++ use 'new' to make allocation, class and structures declaration. But in your kernel you do not have a memory interface (yet), so you cannot use those features in the beginning.
A lot of language can be used instead of `C`, like `C++`, `Rust`, `Go`, etc. You could even code your entire kernel in `ASM` if you want !
So yeah, choose a language, but choose wisely.

III.2 Compilation

III.2.1 Compilers

You can choose any compilers you want. I personnaly use `gcc` and `nasm`. A Makefile must be turned in.

III.2.2 Flags

In order to boot your kernel without any dependencies, you must compile your code with the following flags (Adapt the flags for your language, those ones are a `C++` example):

- `-fno-builtin`
- `-fno-exception`
- `-fno-stack-protector`
- `-fno-rtti`
- `-nostdlib`
- `-nodefaultlibs`

Pay attention to `-nodefaultlibs` and `-nostdlib`. Your Kernel will be compiled on a host system, yes, but cannot be linked to any existing library on that host, otherwise it will not be executed.

III.3 Linking

You cannot use an existing linker in order to link your kernel. As written above, your kernel will not boot. So, you must create a linker for your kernel.
Be carefull, you `CAN` use the 'ld' binary available on your host, but you `CANNOT` use the .ld file of your host.

III.4 Architecture

The i386 (x86) architecture is `mandatory` (you can thank me later).

III.5 Documentation

There is a lot of documentation available, good and bad. I personnaly think the `OSDev` wiki is one of the best.

Chapter IV

Mandatory part

IV.0.1 Base

You must make a kernel, bootable with `GRUB`, who can write characters on screen.
In order to do that, you have to:

- Install `GRUB` on an virtual image
- Write an `ASM` boot code that handles multiboot header, and use `GRUB` to init and call main function of the kernel itself.
- Write basic kernel code of the choosen language.
- Compile it with correct flags, and link it to make it bootable.
- Once all of those steps above are done, you can write some helpers like kernel types or basic functions (strlen, strcmp, ...)
- Your work must not exceed 10 MB.
- Code the interface between your kernel and the screen.
- Display `"42"` on the screen.

For the link part, you must create a linker file with the `GNU linker (ld)`. Some docs here.

IV.0.2 Makefile

Your makefile must compile all your source files with the right flags and the right compiler. Keep in mind that your kernel will use at least two different languages (ASM and whatever-you-choose), so make (<- joke) your Makefile's rules correctly.

After compilation, all the objects must be linked together in order to create the final Kernel binary (Cf. Linker part).

Chapter V

Bonus part

Bonus will be evaluated if and only if your mandatory part is PERFECT. By PERFECT, I obvioulsy mean that it is complete and stable, and free of the tinyest and creepiest error. Actually, if your mandatory part is not 100% on the scale, your bonus will be discarded in their ENTIRETY.

- Add scroll and cursor support to your I/O interface.
- Add colors support to your I/O interface.
- Add helpers like printf / printk in order to print information / debug easily.
- Handle keyboard entries and print them.
- Handle different screens, and keyboard shortcuts to switch easily between then.

Chapter VI

Turn-in and peer-evaluation

Turn your work into your `GiT` repository, as usual. Only the work present on your repository will be graded in defense.

Your must turn in your code, a Makefile and a basic virtual image for your kernel. Carefull with that image, your kernel does nothing with it yet, so there is no need for it to be sized like an elephant. 10 MB is the upper bound, deal with it.