**Bootloader Technical Specifications**

Autor(s): Oliver Ladwig

date: 07/24/2023

**General Tasks**

The bootloader must fulfill a variety of requirements which I want to list here.

The obvious things at first: The bootloader must load its second stage and the kernel into memory to given addresses (specified later). For this it must be assured that the bootloader is given the correct number and position of sectors to read from the disk. This must be configured manually as the second stage of the bootloader and kernel are growing.

Another apparent task is the switch into 32bit protected mode. This process includes setting up the GDT, correctly setting the segment registers and performing a long jump to the new code in protected mode.

The bootloader is also responsible for setting up the stack and setting the correct VESA mode (specified later).

**Tasks Stage 1**

Since the size of stage 1 is very limited its number of tasks are very minimal.

Mainly is stage 1 responsible for loading stage 2 into memory to completely setup the environment. Additionally, it sets up the stack.

This will be the only tasks to leave enough space for detailed debug and error message and handling. Mostly it will write to the “serial0” monitor but also output basic updates using BIOS interrupts.

**Tasks Stage 2**

While stage 1 is only required to load stage 2 into memory the range of tasks for stage 2 is much bigger.

First everything about hardware configuration must be done, since later the access to BIOS interrupts will be gone. Stage 2 uses VBE functions to get informations about the different modes and switches to the later specified VESA mode. Additionally, the second stage must save certain data only available at this point like the position in memory of the Linear Frame Buffer or the Segment. These are values acquired through running the VBE functions as the return values. They will be saved to specific locations in memory (specified later) and later be retrieved by the Kernel.

Now the Kernel gets loaded into memory since we later don’t have access to BIOS interrupts to read from the disk. It gets loaded to a later specified location in memory.

After all of this was set up the switch to 32bit Protected Mode can be made. This involves setting up the GDT, setting the segment registers and performing a long jump to 32bit code.

In the end the environment is setup for the Kernel to take over control and perform more complex tasks. For that a jump to the specified location of the Kernel is performed. That is the end of the bootloader.

Additionally, stage 2 provides debug and status information over the “serial0” monitor. It’s not decided yet if output will be provided to the VGA or VESA screen since it would consume large amounts of time and memory for a library that will only be used in a short time frame in the start.

**Data Specifications**

I will now specify things like the memory locations of certain data in memory.

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Address** | **Value** | **Description** |
| BOOT\_DRIVE | 0x1000 |  | Value acquired through the BIOS containing the boot drive |
| STAGE\_2\_OFFSET |  | 0x1100 | Position in memory the second stage of the bootloader gets loaded to |
| KERNEL\_OFFSET |  | 0x2000 | Position in memory the Kernel gets loaded to |
| VESA\_LFB\_START | 0x400 |  | Adress of the Linear Frame buffer |
| VESA\_LFB\_SEGMENT | 0x500 |  | Value of the “esi” register after switching VESA mode |