hhu.

towboot

a Multiboot-compatible bootloader for EFI-based x86 systems

Niklas Sombert

2023-07-06

Contents

Introduction: Contents

Heinrich Heine
Universität Düsseldorf

Introduction

Heinrich Heine
Universität Düsseldorf

Goals

- UEFI application
- Multiboot-compatible bootloader

Motivation

- firmware of current devices follows the UEFI standard
- Multiboot as the interface between kernel and bootloader
- hhuOS
  - Multiboot-compatible
  - licensing reasons
  - whole operating system

Background: Contents

Technologies used: UEFI

- Unified Extensible Firmware Interface[1]
- Tianocore[2]
- Open Virtual Machine Firmware[3]

[1] *Unified Extensible Firmware Interface (UEFI) Specification, Version 2.8 Errata B, May 2020*. UEFI Forum, 2020. URL: https://uefi.org/sites/default/files/resources/UEFI%20Spec%202.8B%20May%202020.pdf (visited on 10/13/2020).

[2] *tianocore/edk2*. URL: https://github.com/tianocore/edk2 (visited on 01/01/2021).

[3] *OVMF*. tianocore/tianocore.github.io. URL: https://github.com/tianocore/tianocore.github.io/wiki/OVMF (visited on 01/01/2021).

UEFI: Boot Protocol

(without a Compatibility Support Module)

- relocatable PE[4] files
- loaded by the firmware (or another application)
- same CPU mode as the firmware
- `BootXXXX` variables
  - EFI System Partition
- removable media
  - `\EFI\BOOT\BOOTarch.EFI`

[4] Portable Executable

UEFI: API

Heinrich Heine
Universität Düsseldorf

- `efi_main` function
- System Table
- Boot and Runtime Services
- opaque handles
- protocols with UUIDs

UEFI: Protocols

- Loaded Image Protocol
- Simple File System Protocol / File Protocol
- Simple Text Input Protocol / Simple Text Output Protocol
- Graphics Output Protocol[5]
- many more possible

[5] *Replacing VGA, GOP implementation for UEFI*. UEFI Summer Plugfest. 2011. URL: https://uefi.org/sites/default/files/resources/UPFS11_P4_UEFI_GOP_AMD.pdf (visited on 10/25/2020).

UEFI: example

Heinrich Heine Universität Düsseldorf

```
#![no_std]
#![no_main]
use uefi::prelude::*;
use uefi::proto::loaded_image::LoadedImage;
#[entry]
fn efi_main(image: Handle, mut systab: SystemTable<Boot>) -> Status {
    uefi_services::init(&mut systab).expect("Failed to initialize utilit
    let loaded_image = systab.boot_services()
            .open_protocol_exclusive::<LoadedImage>(image)
            .expect("Failed to open loaded image protocol");
    let load_options = loaded_image.load_options_as_cstr16()
            .expect("Failed to get load options");
    writeln!(systab.stdout(), "got load options: {}", load_options);
    Status::SUCCESS
}
```

Technologies used: Rust[6]

- system programming language
- low-level code, high-level abstractions
- type- and memory-safety (excluding `unsafe`)
- `cargo`

[6] *Rust Programming Language*. URL: https://www.rust-lang.org/ (visited on 01/01/2021).

Rust and UEFI

- `i686-unknown-uefi` and `x86_64-unknown-uefi` targets[7]
  - Tier 2
  - `no_std`
- `uefi` crate[8]

[7] *The rustc book*. Chap. Platform Support. URL: https://doc.rust-lang.org/nightly/rustc/platform-support.html (visited on 06/23/2023).

[8] Gabriel Majeri. *uefi*. crates.io. URL: https://crates.io/crates/uefi (visited on 12/04/2020).

Technologies used: Multiboot[12][13]

Heinrich Heine Universität Düsseldorf

- standard for kernels and bootloaders
- from the GNU GRUB[9] project
  - GNU kernel: gnumach[10] (for GNU HURD[11])
- two major versions

[9] *GNU GRUB*. GNU. URL: https://www.gnu.org/software/grub/ (visited on 10/13/2020).

[10] *gnumach*. GNU. URL: https://www.gnu.org/software/hurd/microkernel/mach/gnumach.html (visited on 12/05/2020).

[11] *GNU Hurd*. GNU. URL: https://www.gnu.org/software/hurd/ (visited on 01/01/2021).

[12] Bryan Ford and Erich Stefan Boleyn. *Multiboot Specification version 0.6.96*. Free Software Foundation, Inc. URL: https://www.gnu.org/software/grub/manual/multiboot/multiboot.html (visited on 10/13/2020).

[13] Bryan Ford and Erich Stefan Boleyn. *Multiboot2 Specification version 2.0*. GNU. Free Software Foundation, Inc. URL: https://www.gnu.org/software/grub/manual/multiboot2/multiboot.html (visited on 10/13/2020).

Technologies used: Multiboot

- load kernel and modules (ELF or flat binary)
  - ELF or flat binary
  - header: static information
    - module alignment
    - require passing memory or video information
    - preferred video mode
    - binary loading and entry
    - additional flags

Technologies used: Multiboot

- pass information about the system
  - struct, `EAX` and `EBX` registers
    - memory (upper, lower, map)
    - kernel command line
    - modules (and command line)
    - symbol
    - bootloader name
    - framebuffer
    - legacy stuff (boot device, drives, config table, APM, VBE)
    - on Multiboot 2: SMBIOS, ACPI, UEFI, network
  - machine state
    - 32-bit Protected Mode
    - no interrupts
    - no paging
    - no PAE (in practice)
    - Multiboot 2: also native UEFI mode

Technologies used: Multiboot

- `multiboot` crate[14], PR[15]
- `multiboot2` crate[16], PR[17]

[14] Gerd Zellweger. *multiboot*. crates.io. URL: https://crates.io/crates/multiboot (visited on 01/04/2021).

[15] Niklas Sombert. *Add some functionality for using this in bootloaders (Pull Request 8)*. gz/rust-multiboot. URL: https://github.com/gz/rust-multiboot/pull/8 (visited on 01/04/2021).

[16] Philipp Oppermann. *multiboot2*. crates.io. URL: https://crates.io/crates/multiboot2 (visited on 06/23/2023).

[17] Niklas Sombert. *Add a builder to multiboot2 (Pull Request 133)*. rust-osdev/multiboot2. URL: https://github.com/rust-osdev/multiboot2/pull/133 (visited on 06/23/2023).

Technologies used: x86

- `i686` (32-bit) and `x86_64` (64-bit)
- multiple modes[18]
  - Long Mode
    - 64-Bit Mode
    - Compatibility Mode (32-Bit)
  - Legacy Mode
    - Protected Mode (32-Bit or 16-Bit)
    - Virtual-8086 Mode (16-Bit)
    - Real Mode (16-Bit)
- Alternatives:
  - MIPS (Multiboot 2)
  - Itanium, ARM, RISC-V (UEFI)

[18] *AMD64 Architecture Programmer's Manual*. 2020. Chap. 1.2 Modes of Operation. URL: https://www.amd.com/system/files/TechDocs/40332.pdf (visited on 01/04/2021).

Technologies used: x86

Heinrich Heine Universität Düsseldorf

- Multiboot machine state requires many x86-specifics
  - native mode to 32-bit Protected Mode
  - may have to switch through Compatibility Mode
- memory
  - Real Mode (`8086`): 16-bit pointers, 24-bit address bus, 1MB, memory hole, lower 640KB
  - `286`: 24-bit address bus: 1MB to approx 10MB
  - `i686`: 4GB virtually, physically (more with PAE)
  - `x86_64`: physically 256TB memory
  - *segmentation*

Technologies used: x86: segmentation on the 8086

Heinrich Heine Universität Düsseldorf

- in Real Mode, segment registers are offsets

```
1       mov [ax], 0xab ; writes to *(ds * 16 + ax)
2       mov es:[ax], 0xab ; similar, but with es
3       jmp 0xabc ; actually jumps to cs * 16 + 0xabc ("near-jump")
```

- they're settable, usually

```
1       mov ds, 0x12
2       jmp 0xde:0xabc ; cs can only be set with a "far-jump"
```

Technologies used: x86: segmentation on the 286

- in Protected Mode, segment registers are indices into the Global Descriptor Table
- for a very simple OS with just one application it might look like this:

index	content
0	NULL
1	system code
2	system data
3	application code
4	application data

- information contained in these Segment Descriptors:
  - base, limit, access, flags
  - not layed out sequentially!

- `x86` crate[19] has a builder

[19] Gerd Zellweger. *x86*. crates.io. URL: https://crates.io/crates/x86 (visited on 06/30/2023).

Technologies used: x86: code (1/3)

Heinrich Heine Universität Düsseldorf

```
let code_segment_builder = SegmentDescriptorBuilder::code_descriptor(
    0, u32::MAX, CodeSegmentType::ExecuteRead,
);
let code_segment: Descriptor = code_segment_builder
    .present().limit_granularity_4kb().db().finish();
let data_segment_builder = SegmentDescriptorBuilder::data_descriptor(
    0, u32::MAX, DataSegmentType::ReadWrite,
);
let data_segment: Descriptor = data_segment_builder
    .present().limit_granularity_4kb().db().finish();
let gdt = DescriptorTablePointer::new_from_slice(
    &[Descriptor::NULL, code_segment, data_segment]
);
```

Technologies used: x86: code (2/3)

Heinrich Heine Universität Düsseldorf

```
unsafe { x86::irq::disable();
    x86::dtables::lgdt(&gdt);
    x86::dtables::lidt(&DescriptorTablePointer::default()); // invalid
    asm!("push 0x08", // code segment
        "lea rbx, [2f]",
        "push rbx",
        "retfq", // overwrite CS
        "2:", // compatibility mode, yay.
        ".code32",
        "mov eax, 0x10", // data segment
        "mov ds, eax",
        "mov es, eax",
        "mov fs, eax",
        "mov gs, eax",
        "mov ss, eax",
```

Technologies used: x86: code (3/3)

```
1           "mov ecx, cr0",
2           "and ecx, ~(1<<31)",  // disable paging
3           "or ecx, 1",  // enable protected mode
4           "mov cr0, ecx",
5
6           "mov ecx, cr4",
7           "and ecx, ~(1<<5)",  // disable PAE
8           "mov cr4, ecx",
9
10          "mov ecx, 0xC0000080",
11          "rdmsr",
12          "and eax, ~(1<<8)",  // disable long mode
13          "wrmsr",
14      );
15  }
```

Technologies used: ELF

- Executable and Linkable Format[20]
- support needed to be Multiboot-compliant
- header
- Program Header
- Section Header
- `goblin` crate[21]

[20] Wikipedia contributors. *Executable and Linkable Format — Wikipedia, The Free Encyclopedia*. 2020. URL: https://en.wikipedia.org/w/index.php?title=Executable_and_Linkable_Format&oldid=992446183 (visited on 01/04/2021).

[21] m4b et al. *goblin*. crates.io. URL: https://crates.io/crates/goblin (visited on 01/04/2021).

Technologies used: TOML

- Tom's Obvious Minimal Language[22]
- format for configuration files
- simple, yet type-safe and well-defined
- `toml` crate[23], PR[24]

[22] Tom Preston-Werner. *TOML: Tom's Obvious Minimal Language*. URL: https://toml.io/ (visited on 10/14/2020).

[23] Eric Huss and Alex Crichton. *toml*. crates.io. URL: https://crates.io/crates/toml (visited on 01/04/2021).

[24] Thom Chiovoloni. *Added nostd support (Pull Request 429)*. alexcrichton/toml-rs. URL: https://github.com/alexcrichton/toml-rs/pull/429 (visited on 04/14/2022).

Alternatives

- GNU GRUB[25]
  - widely used bootloader
  - supports various platforms and operating systems
  - menu, can edit entries interactively
  - complex scripting language
  - de-facto reference implementation of Multiboot
- Syslinux[26]
  - very lightweight bootloader
  - supports multiple platforms and some operating systems
  - current release 2014
- Limine[27]
  - supports various platforms and operating systems
  - also has an own boot protocol

[25] *GNU GRUB*.
[26] *The Syslinux Project*. URL: https://www.syslinux.org/ (visited on 01/02/2021).
[27] *Limine*. URL: https://limine-bootloader.org/ (visited on 06/08/2023).

Implementation and Usage: Contents

Heinrich Heine Universität Düsseldorf

Boot process

1. initialize the bootloader
2. determine what to boot
    1. parse the command line parameters
    2. parse the configuration file
    3. display a menu
3. boot the operating system
    1. load the kernel
    2. load the modules
    3. configure the graphics output
    4. prepare the information to pass to the kernel
    5. deinitialize most parts of the bootloader
    6. jump to the kernel's entry point

Configuration: NVRAM

- binary encoding: CBOR[28] or JSON[29] (for instance)
- EFI variable
- requires tooling
- bound to a system
- firmware might “forget” configuration

[28] Carsten Bormann. *Concise Binary Object Representation*. URL: https://cbor.io (visited on 10/14/2020).
[29] Douglas Crockford. *Introducing JSON*. URL: https://www.json.org/ (visited on 10/14/2020).

Configuration: Command line parameters

```
FS0:\> towboot.efi -help
Usage:
-config Load the specified configuration file instead of the default one.
-kernel Don't load a configuration file, instead boot the specified kernel.
-logLevel Set the log level.  (This only applies if `-kernel` is specified.)
-module Load a module with the given args.  Can be specified multiple times.
-quirk Enable a specific quirk.  (Only applies when loading a kernel.)
-help Displays all available options and how to use them.
-version Displays the version of towboot
```

- `towboot.efi -kernel "kernel.elf quiet" -module "module1.bin -verbose" -module "module2.bin -quiet"`
- `towboot.efi -config config.toml`
- file names or arguments may contain spaces or quotes
- `BootXXXX` variables
- shell or from another bootloader
- boot manager in the firmware usually not very user-friendly
- firmware might “forget” configuration

Configuration: Configuration file

```
default = "entry1"
timeout = 10
# log_level defaults to 'info'

[entries]
  [entries.entry1]
    name = "Operating System"
    image = "kernel.elf"
    argv = "quiet"
    quirks = ["KeepResolution"]


  [[entries.entry1.modules]]
    image = "module1.bin"
    argv = "--verbose"

  [[entries.entry1.modules]]
    image = "module2.bin"
    argv = "--quiet"
```

- easy to read and modify by hand
- most bootloaders do this
- `towboot.toml`, placed in top-level directory of the ESP

Memory Management: Stack

- local variables
- works always
- size needs to be known at compile time
- used for most runtime data, for example `MultibootInfo`
- tracked by `rustc` at compile time

Memory Management: Heap

Heinrich Heine
Universität Düsseldorf

- `alloc::alloc::alloc` is bound to `allocate_pool`
- tracked both by `rustc` at compile time and by the firmware at runtime
- used for everything with a dynamic size and no further requirements

Memory Management: Whole pages

Heinrich Heine
Universität Düsseldorf

- can be allocated to specific address (used for the kernel)
- custom `mem::Allocation` wrapper (`impl Drop`)
- tracked by the firmware at runtime
- also used for modules

Workarounds

- stack probes[30] broken with Rust 1.49 (fixed with 1.51, at least)
- initially no support for floating point operations (now just for x86_64)
- `hacks` module

[30] *probestack.rs*. rust-lang/compiler-builtins. URL: https://github.com/rust-lang/compiler-builtins/blob/63ccaf11f08fb5d0b39cc33884c5a1a63f547ace/src/probestack.rs (visited on 01/04/2021).

Changes made to other software: multiboot crates

- `multiboot` crate[31] was designed for use in kernels
- added support for parsing the header, for setting values in `MultibootInfo` and tests[32]
- most difficult part: integration with Rust's and UEFI's memory management
- the same goes for `multiboot2`[33][34]

[31] Zellweger, *multiboot*.
[32] Sombert, *Add some functionality for using this in bootloaders (Pull Request 8)*.
[33] Oppermann, *multiboot2*.
[34] Sombert, *Add a builder to multiboot2 (Pull Request 133)*.

Changes made to other software: hhuOS

- kernel assumed the Multiboot structs to be before the kernel in memory
- PR[35] to copy them to BSS very early
- kernel also looks for the free memory after data passed by the bootloader

[35] Niklas Sombert. *Copy the Multiboot info struct recursively to BSS before enabling paging (Pull Request 19)*. hhuOS/hhuOS. URL: https://github.com/hhuOS/hhuOS/pull/19 (visited on 01/09/2021).

Changes made to other software: miniarg

- parses strings such as `program -foo "bar baz"`
- `no_std`-compatible
- released to crates.io[36]
- not specific to UEFI or Multiboot

[36] Niklas Sombert. *miniarg*. crates.io. URL: https://crates.io/crates/miniarg (visited on 12/05/2020).

Changes made to other software: multiboot12

- abstraction over `multiboot` and `multiboot2` crates
- currently just laying around on GitHub[37]

[37] Niklas Sombert. *multiboot12*. URL: https://github.com/YtvwlD/multiboot12 (visited on 06/30/2023).

Testing: Contents

Heinrich Heine
Universität Düsseldorf

Testing

Components

- unit tests for `multiboot`, `multiboot2` and `miniarg`
- not (yet?) for `multiboot12`

Bootloader

- manual end-to-end test, could be automated?
- Multiboot specs[38][39] contain example kernels
- had to be modified to print to serial
- booting a whole operating system
- virtual machine

[38] Ford and Boleyn, *Multiboot Specification version 0.6.96*.
[39] Ford and Boleyn, *Multiboot2 Specification version 2.0*.

Conclusion: Contents

Conclusion: Testing

- successfully tested with the test kernel, hhuOS[40], GNU HURD[41], Lemon OS[42], HelenOS[43] and NetBSD[44]
- tests with FlingOS[45] and OpenIndiana[46] failed

[40] *hhuOS*. URL: https://hhuos.github.io/ (visited on 01/01/2021).
[41] *GNU Hurd*.
[42] *Lemon OS*. URL: https://lemonos.org/ (visited on 06/30/2023).
[43] *HelenOS*. URL: http://www.helenos.org/ (visited on 06/30/2023).
[44] *NetBSD*. URL: https://netbsd.org/ (visited on 06/30/2023).
[45] Edward Nutting. *FlingOS*. URL: http://www.flingos.co.uk/ (visited on 01/01/2021).
[46] *OpenIndiana*. URL: https://www.openindiana.org/ (visited on 06/30/2023).

Conclusion: Summary

Heinrich Heine
Universität Düsseldorf

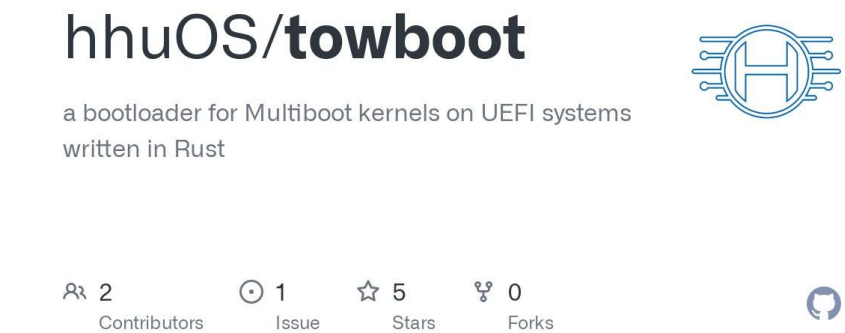
- UEFI's abstractions make writing a bootloader very straight-forward
- `uefi-rs` maps them to Rust
- Rust support is good, but could be better
- Multiboot requires x86-specifics

Conclusion: Further Improvements

- Secure Boot[47]
  - enabled by default on most modern systems
  - changes to kernels and the kernel file format needed to be secure
- 64-bit, UEFI-unaware kernels
- other CPU architectures?
- load improvements
- file paths

[47] Richard Wilkins and Brian Richardson. *UEFI Secure Boot In Modern Computer Security Solutions*. 2013. URL: https://uefi.org/sites/default/files/resources/UEFI_Secure_Boot_in_Modern_Computer_Security_Solutions_2019.pdf (visited on 10/25/2020).

Code



https://github.com/hhuOS/towboot