UF UNIVERSITY of FLORIDA
Herbert Wertheim College of Engineering

Computer Engineering
CEN3907C

Bootloader Development

Overview

In this activity, students will write two bootloaders – one for x86 processors, and another for ARM processors. To complete this activity, students will emulate the Intel i386 and AMD x64 platforms – specifically, the **486** and **qemu64** virtual CPUs – for the x86 bootloader. They will also emulate the "Akita" platform (ARM).

The purpose of this lab assignment is to help students understand the basics of boot routines, the services they provide, and how to develop them. It will also highlight the differences between ARM and x64 boot routines and their respective levels of standardization.

Specification

Students will develop a raw binary bootloader image for the targeted platform and may use any of the target tools. The bootloader will read a message from a memory mapped block device at a designated memory address with a null terminator (**`\0`**) representing the end of the string. This message should be sent to the UART (serial) device, and then the bootloader should halt.

This process mimics two common themes in bootloaders – 1) loading the bootloader from a device, and 2) reading logging data and sending it to a serial output stream (often used for debugging.)

Memory & Devices

On many systems – including x86- and ARM-based ones – memory devices (or portions thereof) are directly mapped into the memory space. For example, in the 1980s, some systems had a ROM chip that stored a lightweight Disk Operating System (DOS). This system could then be directly mapped into memory at a specific address, allowing the system to boot without any storage or disk dependency. Such devices can be accessed directly by memory address at boot – they do not need to be "loaded". Such memory devices can also be mapped to memory for IO purposes.

In this assignment, we will directly map an image file to represent the memory of the logging device. For the x86 target, we will use a standard i386 virtual machine and map the logging device to main memory at the beginning of conventional memory (RAM), which by standard lies at address **`0x500`**.

As ARM memory layout is not standardized, we provide the Akita's key addresses here.

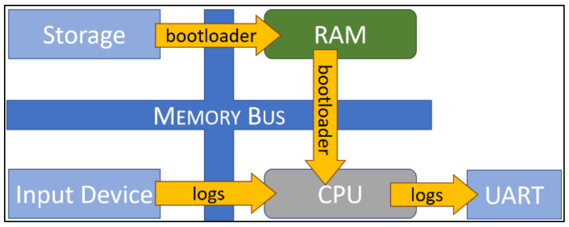

Figure 1. Virtual Machine diagram.

Akita Mapping	
Device	***Address***
Flash	`0x00000000`
UART	`0x40100000`
RAM	`0xA0000000`
Log Dev.	`0xA3F00000`

Source

Students will write source code and build files for the GCC toolchain. The build files must include at least one **Makefile** that will generate the binary bootloader images from the source.

Note that you will need to use the cross-compile version of the compiler toolchain for ARM bootloader development! Make sure your builds target the correct instruction set.

Investigation

After building the bootloader images, students should verify that they run on the target platforms via QEMU. Following this, students should investigate per these instructions.

Experiment: x86

After successfully running he bootloader on the i386 platform (486 CPU), try using the same image, but use the x86_64 emulator (qemu64 CPU).

Questions

1) Describe the result; does the machine complete the task successfully?
2) Why do you think this was the result you observed?

Experiment: ARM

After successfully running he bootloader on the "**akita**" platform, try using the same image, but us the "**raspi2**" platform.

Questions

3) Describe the result; does the machine complete the task successfully?
4) Why do you think this was the result you observed?
5) How does this differ from the way the x86 bootloader worked on x86 and x64, if different?

Development Suggestions

Bootloaders differ from typical software in that they do not have the usual benefits of an operating system or even resident monitor. To help students progress, we suggest tackling the problems in the following order.

0) **Read the documentation**. Yes, it is dense – but everything you need is in it!
1) Start with a "Hello World" bootloader for x86; the BIOS does a lot of "heavy lifting" for you!
2) Once you have your "Hello World", use GDB to trace through it to get a feel for the debugger.
3) Investigate the "x86-mini" example; it has source and binaries you can use as a reference.
4) You should be able to rebuild the x86-mini binary from its source and match it byte-for-byte.
5) Finally, tackle the UART systems on each platform.

Submission

Your submission will be composed of the following elements:

- Screencast of your bootloaders running as an unlisted Internet resource (e.g., YouTube)
- Report (**BootloaderReport.pdf**) on Canvas, *including the link to your screencast*
- Source and build files as a gzipped tar file (**bootloader.tar.gz**) archive on Canvas

Screencast

Your video demonstration will be submitted on Canvas as a list in your report, must be no more than 2 minutes, and should include the following, at a minimum:

- Display the file directory/directories for the project, including the Makefiles
- Show and explain the contents of all Makefiles in your build
- Build the project to completion to show the build works
- Run the bootloaders generated by the build process

Report

Your report will explain the process you used to develop the bootloaders, including what tools you used. It will describe how testing was performed and any known bugs. In addition to all other template elements, your report should include an explanation of the investigative element of the assignment and answers to the posed questions. It should be 500 words or fewer, cover all relevant aspects of the project, and be organized and formatted professionally – ***this is not a memo!***

Compressed Archive (bootloader.tgz)

Your compressed tar file should have the following directory/file structure:

```
bootloader.tgz
   └→bootloader.tar
         └→bootloader (directory)
               ├→Makefile
               └→(Other sources / folders)
```

To build the library, we will execute these commands:

```
$ tar zxvf bootloader.tgz
$ cd bootloader
$ make
```

The build process should generate two files in the **bootloader** directory:

```
bootloader (directory)
   ├→i386_image.bin
   └→akita_image.bin
```

Additional Resources

Here is some additional reading / resources that you may find helpful as you complete this project:

- https://gcc.gnu.org/onlinedocs/gcc/x86-Options.html
- https://gcc.gnu.org/onlinedocs/gcc/ARM-Options.html
- http://web.mit.edu/gnu/doc/html/gdb_5.html
- https://en.wikipedia.org/wiki/X86_instruction_listings
- https://en.wikibooks.org/wiki/X86_Assembly/GNU_assembly_syntax
- https://www.eecg.utoronto.ca/~amza/www.mindsec.com/files/x86regs.html