[Quintin Lopez-Scarim]
[February 2nd, 2024]
[23533621]

Report – [Bootloader Development]

Introduction

The goal of this bootloader development is to create a bootloader to run on the i386 (486 CPU) platform, and another one that runs on the akita platform – with both programs simply sending a UART message. This basic program lays the groundwork for a typical bootloader development flow and gives experience with a method of debugging in the form of UART messages.

Design

To develop the bootloaders an emulator in the form of qemu was utilized, and two separate toolchains were utilized to assemble, and link the code. One toolchain was targeted at the x64 architecture and the other aimed at ARMv3. Most of the information such as addresses to write to were found through internal documentation, along with the provided documentation packaged with the toolchains and emulator. From there initial development processes revolved around getting UART to print a letter in either platform, and then focusing on reading the string loaded in memory until the null character, and then combining both of those parts together. Before writing to the UART I first ensured the buffer was ready for data, however to prevent corrupting the buffer or data.

LINK FOR VIDEO:
https://youtu.be/49fdf5jm2Hc
(initial directory is implied in terminal, accidentally covered it in explorer)

Investigation:

Experiment: x86

1). Utilizing the following command:

```
qemu-system-x86_64 -cpu qemu64 -nographic -drive file=xxxxx,format=raw -device
loader,file=lemon_log.bin,addr=0x500 -s -S
```

I tested the bootloader for the i386 platform on the x86_64 emulator, and did indeed have the machine complete the task identically to the i386 platform.

2). The reasoning for the program working on both platforms is the standardization of the memory mapping and the backwards compatibility caused by the standardized boot process, without either of these variable results could have occurred when trying to map the program over to the x86_64 system.

<u>Experiment: Arm</u>

3.) Utilizing the following command:

```
qemu-system-arm -M raspi2b -nographic -device loader,file=xxxxx,addr=0x00 -device
loader,file=lemon_log.bin,addr=0xA0000000 -s -S
```

I tested the bootloader for the akita platform on the raspi2 platform and it did not complete the task correctly. It remained running seemingly forever, with no output.

4.) This result likely occurred due to the uniqueness of arm - the memory layout is not standardized like the x86 or x64 systems. This means that the program may have not been loaded into the right place to run, and that UART likely was also mapped elsewhere.

5.)The lack of cross-compatibility for arm is what differs itself from x86 and x64 systems. Due to standardization most if not all of the bootloader process (simple ones at least) can be transferred between x86 and x64, whereas even the most basic of programs are typically not transferrable for arm due to unique memory mapping.

Conclusion

This lab serves as an introduction to emulation and bootloaders and provides a real-life example of the effects of standardization vs unique architectures. It provides a foundation to expand on for unique bootloader programs, and also the resources to emulate a vast amount of different patterns.