Hardware Lab Project Documentation

UEFI Weather Application

Instructor: Dr. Alireza Ejlali

Teaching Assistants: Mr. Bahreyni, Ms. Farahani

Mohammad Sadegh Majidi Yazdi	**98106004**
Solale Mohammadi	**98106015**
Mahdi Jafari Raviz	**98109612**

Fall 2023

Introduction and Project Description

UEFI, or Unified Extensible Firmware Interface, has emerged as a fundamental component in modern computing environments, playing a key role in system preparation and configuration. This advanced hardware standard replaces the older Basic Input Output System (BIOS) and provides a more powerful and versatile environment for booting and managing computer systems. UEFI is crucial for enhancing system security, enabling boot-time diagnostics, and supporting a variety of software applications. Serving as a vital interface between hardware and the operating system, UEFI's extensive capabilities facilitate the development of diverse software solutions at the firmware level for various applications.

The proposed project, "UEFI Weather Reporting Application", represents a significant venture in harnessing UEFI capabilities to create a weather application. This project not only emphasizes the importance of this firmware environment but also delivers a practical utility to users. The goal of this project is to leverage the capabilities and high-level access provided by UEFI to retrieve online weather data from an external API and display it directly to users during the system boot process. This innovation will provide users with a unique tool for quickly accessing essential weather information, demonstrating that UEFI-based applications can have comparable, if not greater, power than applications running in traditional operating system environments. Additionally, team members will gain deeper insights into UEFI concepts, system programming, and programming for this environment, establishing direct connections with other components such as network cards throughout the project.

Project Execution Process and Challenges

Week One

During the first week, we initially delved deeper into the concept and use of the UEFI hardware interface. Most of this exploration was conducted through online resources and reports available on the internet. Subsequently, to kick off the project and gain more familiarity with developing a simple "Hello World" program for the output, we followed the steps provided in this link and successfully built and executed this straightforward program.

Week Two and Three

Throughout the second and third weeks, we completed our studies related to UEFI and EDK II to a suitable extent. Following the documentation available in the EDK II repository, we created directories and files (.c, .inf, .uni, and .dsc) specific to our project and placed them in a personal GitHub repository. We also attempted to standardize running QEMU with the UEFI shell and then execute our program through the shell (instead of directly executing our program with QEMU), but we were unsuccessful in achieving this.

Week Four

In the fourth week, the main focus was on understanding the example provided in the specification file related to UEFI. We aimed to derive inspiration and utilize it to write a program with a simple HTTP GET request. Through studying this example, we learned about the initial steps required to enable the HTTP protocol in our program. Firstly, we created a new handler through the Service Binding protocol provided to us, binding our program to the HTTP service and protocol. Subsequently, using this handler, we located and loaded tools related to the HTTP protocol. Configuration steps, such as DHCP usage, HTTP version selection, and IPv4 incorporation, were then performed. Using the structs provided by the protocol, we supplied various components of an HTTP request, such as headers, body, method, and address. Finally, we created an Event for this request using the specified characteristics and placed the event in the queue of HTTP requests using the Request. We also implemented mechanisms to ensure the reliability of sending requests using appropriate callbacks and followed similar steps for receiving a response from the external server. We rewrote these steps in a simplified manner for an HTTP GET request in our project. It was also necessary to use the relevant libraries and protocols in the project's .inf and .dsc files to ensure the correctness of the build and functionality of the program.

Week Five

In the fifth week, the initial code for our HTTP request was completed, but during execution, we encountered a “No Mapping” error. As no requests were observed in Wireshark, it seemed that the issue was related to the network settings of the qemu (UEFI simulator). Therefore, to resolve it, we had two potential solutions. The first solution was to examine and adjust the qemu settings and correctly add the network. The second solution was to execute our program directly with the hardware of our systems instead of running it with the simulator. The entire week was dedicated to debugging this error.

Week Six

During the sixth week, we successfully connected the Qemu simulator to the internet by making appropriate configurations. We ensured this connectivity by executing the `ping` command. The commands used for these configurations were as follows:

```
connect
ifconfig -r eth0
ifconfig -s eth0 dhcp
ifconfig -l eth0
```

Additionally, we managed to run Qemu in a different way. Initially, we gained access to the shell and then executed our program. With these advancements, the previous error “no mapping” was resolved. However, a new error, “invalid parameter”, occurred at a point beyond the previous issue, specifically after calling the `Request` function. Investigations revealed that the process of obtaining the address from the domain by DNS was performed correctly. Still, after that, during the TCP connection with the main server, the connection was quickly dropped after some message exchange and handshaking. The reason for this was unclear to us, and direct debugging was not feasible. Another issue was that, after an initial failed execution, subsequent executions of the program did not trigger any events.

Week Seven and Eight

The focus in these weeks was primarily on identifying the code problem and debugging. However, since we were unable to compile and successfully run our customized OVMF with prints for debugging purposes, progress was limited. The root cause of this problem was that the new versions of EDK II were not executable with old versions of Qemu. Unfortunately, no errors were provided to trace the origin of the problem. To avoid falling behind in other tasks, we designed and initially implemented a textual menu and user interface for the program in parallel during these two weeks.

Week Nine

In the ninth week, we succeeded in building an older version of EDK II and ran our OVMF image with prints in suspicious locations to trace errors by Qemu. With this approach, we were able to debug our code. It became evident that the main issue causing the "Invalid Parameter" error was the absence of sending any headers for our request. The HTTP implementation in EDK II required at least one header in the request to prevent it from being dropped. Additionally, the reason that no events occurred in subsequent program executions was due to improper cleanup, and the child handle related to HTTP was not properly removed. We swiftly addressed the bugs and successfully sent a simple HTTP request and received its response.

Week Ten

Finally, during this week, we implemented and deployed a simple and suitable weather API for our program using Python. We completed the user interface code, modified the HTTP request appropriately, and added logic for processing the API response and displaying the weather. By the end of this week, our program had 100% correct and complete functionality, and the code was well-organized. At present, we are engaged in writing the report and documentation for this project.

Summary of Challenges

In summary, our main challenge in this project was the absence of a suitable tool or method for debugging the program. Additionally, it can be said that there were almost no suitable and comprehensive resources on the internet for working with HTTP in UEFI. Consequently, finding the correct path in this project proved to be very challenging. Unexpected issues, such as the lack of compatibility between new versions of EDK II and Qemu, were among the other significant and time-consuming challenges in this project. Another noteworthy aspect was configuring the network for Qemu, which required significant time and effort. Throughout this project, most of our time was dedicated to debugging, searching, and studying related topics. We adhered largely to the proposed timeline in the project proposal, but minor adjustments were made due to encountered issues.

Code Explanation

For detailed explanations of all the code implementations, please refer to the comprehensive documentation provided in this readme.

Run & Results

To review the project's results and find instructions for building and running, please consult the information available in this readme.