

# Eigenes Betriebssystem : Dokumentation

Maximilian Trautwein

18. Januar 2023



**FERDINAND  
BRAUN SCHULE**

Technische Schulen der Stadt Fulda

# Inhaltsverzeichnis

<b>1</b>	<b>UEFI / BIOS</b>	<b>3</b>
1.1	BIOS . . . . .	3
1.2	UEFI . . . . .	3
<b>2</b>	<b>Bootloader</b>	<b>3</b>
<b>3</b>	<b>Entwicklungsumgebung</b>	<b>4</b>
<b>4</b>	<b>Allgemeine Konzepte und Begriffe</b>	<b>5</b>
4.1	Kernel . . . . .	5
4.2	GOP / General Output Protocol . . . . .	5
4.3	Paging . . . . .	5
4.4	Interrupts . . . . .	6

# 1 UEFI / BIOS

Die erste Grundfrage, die ich mir gestellt habe, ist, ob ich einen BIOS oder UEFI Bootloader programmiere. Ich habe mich für einen UEFI-Bootloader entschieden, da das originale BIOS nicht sehr viele Möglichkeiten zur Interaktion mit der Hardware bietet und ich mit dem UEFI auf der sicheren und fortschrittlicheren Seite bin.

## 1.1 BIOS

Das BIOS (Basic Input Output System) ist ein von IBM in 1981 zuerst implementiertes und in 1975 von Gary Kildall erdachtes System, welches Entwicklern helfen soll, einfacher und unabhängiger auf die Hardware zuzugreifen und somit die Betriebssystementwicklung normierter und weniger komplex zu gestalten. Das BIOS stellt Funktionen und Felder zur Verfügung, mit denen man z.B. Text ausgeben oder einfachen Zugriff auf Speicher zum Starten eines Betriebssystems haben kann.

## 1.2 UEFI

Das UEFI (Unified Extensible Firmware Interface) ist eine Firmwaredefinition für x86-64, ARM und Itanium Architekturen. Diese definiert eine Schnittstelle zwischen BIOS und dem Betriebssystem. Ursprünglich hat Intel diese Firmware für ihre Itanium-Plattform und hat 2005 eine Gruppe (Unified EFI Forum) gebildet, welche aus allen groSSen Firmen besteht (AMD, Intel, Microsoft und Apple).

# 2 Bootloader

Der Bootloader ist ein kleines Programm, welches den Kernel aus einer Partition einliest und die Eingangsfunktion dieses Kernels aufruft und dem Kernel die Kontrolle über die Hardware erteilt. AuSSerdem wechselt er zum kernelspezifischen Betriebsmodus. Das BIOS/UEFI bestimmt, wie der Bootloader programmiert wird und wie dieser die Kontrolle übergibt. Die zweite Grundfrage, die ich mir gestellt habe, ist, ob ich einen eigenen Bootloader programmieren sollte oder lieber auf schon vorhandene Bootloader zurückgreife. Ich habe mich für einen eigenen Bootloader entschieden, da ich somit mehr lerne

und auch mehr Kontrolle darüber habe, wie und was etwas ausgeführt wird. Für die Entwicklung von UEFI-Applikationen gibt es eine Entwicklungsumgebung aus dem GNU-Projekt, das gnu-efi heisst. Diese Umgebung bietet einen effizienten Zugriff auf Funktionen und Informationen aus dem UEFI.

### **3 Entwicklungsumgebung**

Da ich plattformunabhängige Software programmiere, brauche ich einen Cross-Compiler. Als C/C++ Compiler nutze ich GCC/G++, da diese auch ohne spezifische Plattform ausführbare Dateien kompilieren kann. Als Code-Editor habe ich VS-Code verwendet, da ich auf Windows mit WSL mein Betriebssystem kompiliere. VS-Code unterstützt das Verbinden von dem Editor und WSL. Das ermöglicht einem die richtigen Vervollständigungen und Einfärbungen des Codes, damit die Programmierung angenehmer und effizienter verläuft. Als Testumgebung habe ich QEMU, ein Systememulator und virtuelle Maschine, welche mit der OVMF (Open Virtual Machine Firmware) ausgeführt wird, damit auch UEFI-Systeme emuliert werden können. Der Cross-Compiler erschafft eine Image-Datei, welche QEMU dann ausführt.

## 4 Allgemeine Konzepte und Begriffe

### 4.1 Kernel

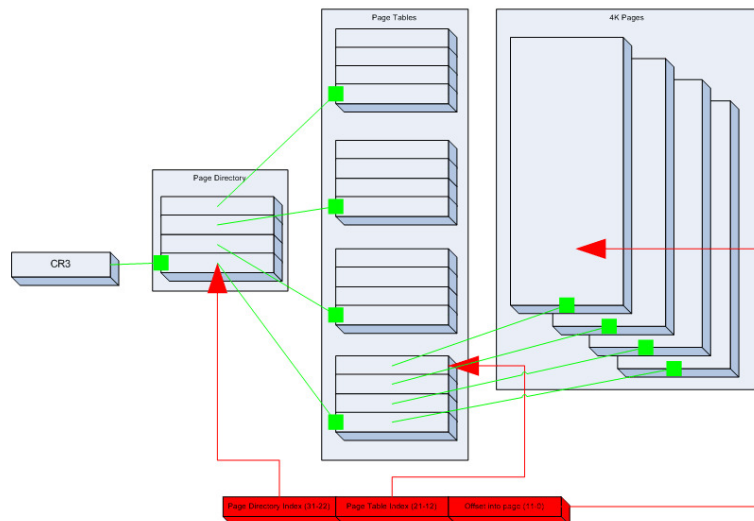
Der Kernel ist die zentrale Einheit eines Betriebssystems und verwaltet Speicher, Geräte und bietet eine Schnittstelle für Programme von Nutzern.

### 4.2 GOP / General Output Protocol

Ein UEFI Protokoll für den einfachen Zugriff auf den Bildschirm, um Text auszugeben. Das Buffer kann man an den Kernel weitergeben, damit dieser einfach die Pixel auf dem Bildschirm setzen kann.

### 4.3 Paging

Paging ist ein Speicherverwaltungssystem, das virtuelle Speicheradressen zu physischen Adressen setzt. Dadurch kann man Speicherbereiche für jede Applikation auf dem Betriebssystem erstellen und somit diese isolieren. Somit kann eine Applikation nicht den reservierten Speicher einer anderen korrumpieren. Das umadressieren erledigt die MMU(Memory Management Unit), welche sich auf der CPU befindet.



## **4.4 Interrupts**

Interrupts sind Anfragen von Geräte an die CPU, dass sie ihre derzeitige Aufgabe stoppen und mit etwas anderem, z.B. das Ausführen einer anderen Funktion, beginnen soll. Nach dem Sprung zur anderen Funktion kann die CPU wieder bei der letzten ausgeführten Anweisung fortfahren.

## **4.5 PCI**

PCI (Peripheral Component Interconnect)