CSE321 Theory Assignment 01

NAME: MD. SABBIR AKON

ID: 22301242

SECTION: 07

Bootloaders are the unsung heros of computing, they are the bridge between the hardware of our computer and the operating system that runs on it. They are the earliest software to be executing on our machine upon booting, with their main responsibility being to set up the hardware prior to loading the operating system kernel into memory to take over.

BIOS-based Bootloaders

BIOS, the old firmware in PC helped to drive PCs over a period that spanned more than three decades. Its boot process is a procession-like relay race in a highly-defined order in which the successive stages pass control to each other.

Workflow and Components

In a system with a BIOS, the path between the power-on state and a usable desktop can be understood as running according to a general hardware design.

The system boots with Power-On Self-Test (POST). This is a Hardware system check which involves the BIOS asking questions such as, "Is the memory OK?" Does it have a keyboard? In case something important malfunctions, we may hear a set of beep codes, diagnostic messages about the problem.

In the event of a successful POST, the BIOS accesses the configured boot order list. It is a mere priority list, such as the following: 1. USB Drive, 2. CD/DVD drive, 3. Hard Drive. The BIOS searches the first machine on boot instructions. When it does not find one, it moves on to the next.

BIOS locates its first 512-byte sector whenever it locates a bootable drive. This is the small, very important miscellaneous bit of the disk: the Master Boot Record (MBR). The MBR resembles an ancient rolodex directory and a master key to the whole drive. It carries two principal

components, the very small program that serves as an initial bootloader and a partition table that specifies how the drive is partitioned (up to 4 primary partitions).

The BIOS then transfers power to the miniature programm of the MBR. The role of this program is to scan through the partition table to identify active one. This is the partition the system should boot in. In general, this is most likely our C: drive, in the case of Windows.

The last step the MBR code takes is loading yet another 512byte sector, in this case it is at the start of that active partition. This is the volume boot record, or VBR. The VBR is followed by the next stage of the bootloader, which is operating system-specific depending on the installed operating system on a given partition. An example of this would be VBR on a more senior Windows system loading up the file NTLDR, which then in turn loads the remaining Windows. This is the standard way the BIOS boot chain works: BIOS loads to MBR, which loads into VBR, which loads into OS loader.

UEFI-based Bootloaders

Unified Extensible Firmware Interface (UEFI) is the new system that is the successor to the BIOS, which was built with the limitation in mind. It is not more of a simple set of hardware instructions, rather a mini-OS that runs ahead of our main OS.

Workflow and Components

The UEFI boot process is much more file-based, and is less rigid than the traditional process.

When a UEFI system is powered-on, the UEFI firmware does a much more thorough initialization than what was done on a BIOS system during POST. It then searches our storage drive (usually through the drive letter assigned to it) to find a particular partition entitled the EFI System Partition (ESP). The ESP is a game-changer; a tiny space (typically 100-500 MB) formatted in the easy to read FAT32 file-system where the UEFI firmware can read directly.

This ESP is a typical, central, directory of all bootloaders. Rather than a concealed MBR, each operating system adds its bootloader as a normal file with an.efi extension into this partition. As an example, a computer containing both Windows and Ubuntu would be placed with a single ESP containing directories such as \EFI\Microsoft\ (with the Windows Boot Manager) and \EFI\Ubuntu\ (with the GRUB bootloader).

The EFI firmware itself contains a boot manager with a list of things to boot, typically .efi files. When booting the system, it first checks this list (modifiable in the firmware configuration) and remembers to step through the default or user-specified.efi file next. As an example, it may be set to run \EFI\Microsoft\Boot\bootmgfw.efi to boot into Windows.

This.efi file is a fully functional application and not some small piece of code. It replaces the UEFI firmware and then loads the remainder of its operating system into the memory, as well as

executing it. This pure-file, file-based mechanism is a lot cleaner and solid compared to the earlier MBR based technique.

Architectural Difference between BIOS and UEFI

The shift in technology, moving BIOS to UEFI, is a profound step forward in technology, and opens up a 30-year gap in limitations on computing architecture.

BIOS Architecture

BIOS architecture is a creation of its time. It is a real mode program, 16-bit one, so it can only address approximately 1 MB of system RAM. That is why the BIOS set up screens are simple text based. This inherent limitation has some significant repercussions

The most prominent is that it is based on MBR partition scheme. The MBR format sets aside 32-bit counts to define disk sectors, and this provides mathematically a maximum limit in disk size of 2.2 terabytes (TB). With any hard drive larger than that, a BIOS system just could not use that extra space to boot. Moreover, the MBR has a limited number of entries in its partition table: four entries to use as primary partitions. To add more, we must make one of those an extended partition to contain more logical partitions--a hacky workaround.

UEFI Architecture

UEFI, in its turn, is located in 64-bit or 32-bit protected mode, which makes it possible to access the entire available memory of the system it runs as well as to process much more sophisticated programs. That is why the modern UEFI firmware options appeal usually to the complete graphics user interfaces (GUI) with mouse support, comprehensive diagnostics of the hardware, and even networking support.

Also architecturally, UEFI is intended to employ GUID Partition Table (GPT). GPT takes place of the limitations of the MBR It offers 64-bit addressing of disk sectors which extends the maximum disk size to 9.4 zettabytes which is essentially inexhaustible to current technology. A maximum of 128 primary partitions is allowed in Windows by default, thus eliminating the use of extended partitions.

Of most importance, UEFI implements Secure Boot. This is one of the major security components because the firmware will have a database of the trusted digital certificates. Prior to execution of a bootloader, the firmware verifies its digital signature. In case of a signature that corresponds to an authentic signature in its database, the bootloader is allowed to execute. It prevents execution of an unsigned or signed executable image with an origin that is unknown by the firmware (as it would be in the case of a rootkit virus that tries to steal the boot process).

This provides a kind of chain of trust at which there is no point in pressing the power button when the system is already secured even before it boots the main OS.

Multi-Boot Systems

Multi-boot: This is a configuration where the user not only installs more than one operating system into a machine, but they specifically select which operating system they want to run when they startup the machine. A master bootloader is set up to coordinate all multiple systems installed.

Managing and Detecting Multiple OSeS

A multi-boot loader, such as commonly used GRUB (GRand Unified Bootloader), takes control of the initial booting process. When installing a Linux distribution such as Ubuntu alongside of Windows, the installer will normally install GRUB as the first boot manager. With a UEFI system, it does this by informing the firmware that its grubx64.efi file must be the initial one to be loaded on startup.

After that, GRUB gets loaded first, when we start up our computer. Rather than loading an OS itself, it shows us a menu. This menu is created by a utility script (such as os-prober on Linux) that searches all of our drives to uncover other operating systems. Such as--as soon as it notices that Windows Boot Manager has an .efi file on the ESP, it will automatically add a Windows entry to the menu list of options in addition to adding the entry for Ubuntu.

The strength of Chainloading

GRUB has trouble knowing about the system-specific complex procedures required to boot by any of a virtually unlimited range of potentially available operating systems. The simple elegant remedy of this issue is chainloading.
chainloading One bootloader (or another stage in the chain of bootloaders) passes control to another. It is an approach to delegation. We will run through a typical scenario of using Windows grub menu:

1. We power up our PC. GRUB is run in its UEFI firmware.
2. GRub shows its menu: Ubuntu and Windows Boot Manager.
3. We press the arrow keys to select " Windows Boot Manager" and press Enter.
4. The task belonging to RUB now is so simple It does not make any attempt to load the Windows kernel or even know the windows file system. It simply searches and loads the official windows bootloader file (bootmgfw.efi) on ESP and runs it.

5. At this point, the work of GRUB has ended. The Windows Boot Manager assumes total control over the screen, and Windows booting commences as in the case of the sole existence of the Windows operating system on the machine.

That makes it mutually more compatible since each operating system has the expertise of the use of its sole custom-written bootloader. RUB merely serves as an ascerating doorman inviting us to choose the operating system they wants to see us in and sending us through the appropriate door.