Original Research Paper

Design of a Dynamic Boot Loader for Loading an Operating System

[1]**Alycia Sebastian and [2]Dr. K. Siva Sankar**

[1]*Research Scholar, Department of Computer Science and Engineering,*
Noorul Islam Centre for Higher Education, Tamil Nadu, India
[2]*Department of Information Technology, Noorul Islam Centre for Higher Education, Tamil Nadu, India*

Article history
Received: 26-08-2018
Revised: 22-09-2018
Accepted: 25-01-2019

Corresponding Author:
Alycia Sebastian
Department of Computer Science and Engineering, Noorul Islam Centre for Higher Education, Tamil Nadu, India
Email: alycia.sebastian@gmail.com

Abstract: Boot Loader is the crucial program that loads the operating system in memory and initializes the system. In today's world people are constantly on move and portable system are in demand specially the USB devices due to its portability and accessibility compared to CD/DVD drives. The purpose of this paper is to design a dynamic boot loader which removes the BIOS dependency and allow user to boot from USB without changing CMOS settings. The USB is devised as plug and play portable system with puppy Linux and newly developed dynamic boot loader. The device is experimented on a computer machine with 8 GB RAM, i5 processor, 64-bit Operating system and windows 7 and observed that nearly 50% reduction in booting time i.e., the time spent in changing the boot order is eliminated compared to the static boot loader. The time spent in the BIOS is dependent on the user knowledge in changing the boot priority. The portable system allows the user to work in ease in any environment with minimum requirement of Windows XP and USB 2.0 compatible system.

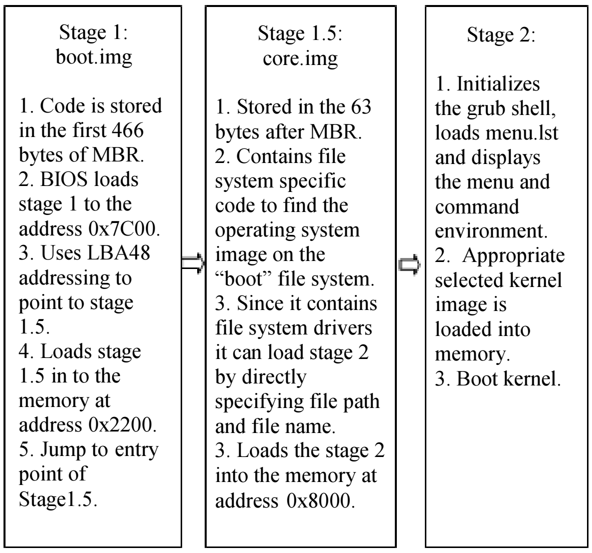
Keywords: Boot Loader, Dynamic, Operating System, USB, Open source OS

Introduction

Boot Loader is the first program that starts running after BIOS. Its task is to load the operating system in memory. The boot loader resides in the first sector of primary partition and the size is restricted to 512 bytes. Due to this size limitation, it becomes difficult to load and execute a 32 bit kernel. To accommodate more features the boot loader is divided into 3 stages. The first stage primary boot loader initializes the memory, hardware devices and loads the second stage boot loader. So there is no size restriction for secondary boot loader. Stage 1.5 which is 63 bytes is to interpret the file system. The boot process is as explained in Fig. 1.

The USB can be made bootable with different versions of Linux OS, Windows or Mac OS. Open source Operating systems are customizable, flexible, secure and freely available and so widely gaining popularity. Windows have restriction in installing to a drive other than the hard disk.

To boot from USB, the user needs to change the boot order and choose the USB to start the booting process from USB. The available boot process is static which is dependent on user changing the BIOS settings. In recent times, BIOS is being slowly replaced by Unified Extensible Firmware Interface (UEFI). UEFI secure boot allows only trusted OS to boot while locking out open source OS.



Stage 1:
boot.img

1. Code is stored in the first 466 bytes of MBR.
2. BIOS loads stage 1 to the address 0x7C00.
3. Uses LBA48 addressing to point to stage 1.5.
4. Loads stage 1.5 in to the memory at address 0x2200.
5. Jump to entry point of Stage1.5.

Stage 1.5:
core.img

1. Stored in the 63 bytes after MBR.
2. Contains file system specific code to find the operating system image on the "boot" file system.
3. Since it contains file system drivers it can load stage 2 by directly specifying file path and file name.
3. Loads the stage 2 into the memory at address 0x8000.

Stage 2:

1. Initializes the grub shell, loads menu.lst and displays the menu and command environment.
2. Appropriate selected kernel image is loaded into memory.
3. Boot kernel.

Fig. 1: Stages of boot loader

This paper proposes a dynamic boot loader which addresses the static boot. A light weight open source operating system is installed in the USB. This portable system can act as a substitute for personal laptop particularly for users who are constantly on move. It's easier to carry and the user can work on any borrowed system without having to worry about the security of the data transaction.

Related Work

Universal Serial Bus (USB) flash drives are leading portable storage device for storage and easy transfer of data from one computer to another. Due to its significant growth in usage many researchers have experimented with USB as portable device and as installation resources. The boot loaders discussed below are dependent on the user knowledge in changing the boot order to boot from USB. The dynamic boot loader concept removes this dependency as explained in the next section.

With USB gaining popularity as replacement for CD/DVD drives as installation medium to multi boot different ISO images, comparison study has been made using two boot loaders SYSLINUX and GRUB2 for various Linux based operating system (Sivaiah *et al*., 2014). The success rate of both the boot loaders to boot load the different OS have been graphed and found that SYSLINUX have higher rate compared to GRUB2. For every booting of ISO images from USB, it is needed to change the boot order in the BIOS to USB. The boot loaders are static and dependent on BIOS.

The USB can be utilized as both as installation medium and also as a portable device (Karna, 2010; Karna and Chen, 2014). Different experiments were conducted with Windows OS, Linux distributions, Opensolaris and Fedora and the results are tabulated. Windows OS has limitation in installing to a USB drive.

Security is the major concern for any portable device. Trusted Platform Module (TPM) ensures data security from hardware layer. With the advancement of USB technology, USB is developed as portable TPM (Kushwaha, 2013) and for portability the EFI System Partition (ESP) which contains the boot loader program is installed in USB. For every boot, the hash value is calculated and checked with the previous stored value in USB. Here USB acts as portable TPM. So securing USB becomes highly critical.

To build USB as portable device (Jebarajan and Sankar, 2011; Sankar, 2010; Kanahasabapathy *et al*., 2015), a customized kernel and boot loader is developed and written into the drive. The lightweight OS reduces the complexity of compressing and loading the kernel image from USB to RAM.

Design of a Dynamic Boot Loader

The dynamic boot loader methodology was proposed in my previous paper (Sebastian and Sankar, 2015). This eliminates BIOS dependency, thus providing a user friendly system and significantly reduces the load on the user.

Live USB

A USB drive of 16 GB is used. Lesser size USB can also be used as it requires less space to make USB bootable. USB is formatted with FAT32 file system which is the compatible file system for a USB. The USB is made bootable with kernel puppy slacko 6.3.2 and SysLinux boot loader using Unetbootin tool. It is a simple and small Linux OS distribution of size around 240 MB.

To make USB portable, in the first boot, the Live USB session is saved as 'slackosave-s1.2fs'. This allows booting to the saved session on next reboot. More sessions can be saved with different customizations. If session not saved any changes made will not be saved and on reboot it starts from the scratch. The session s1.2fs is saved in the sdb1 i.e., USB with extension ext2 and file size of 512 MB. The USB can be made bootable with persistent storage of max size 4GB and the session can be saved in the persistent storage partition. The OS writes only once to the flash drive on session close thereby reducing the number of writes to the flash drive.

The file 'slackosave-s1.2fs' is password protected for security. Every time on shutdown the session is automatically saved to the file. Now the USB is portable and the same data is available for booting from different machines. Still USB boot is dependent on boot order priority. The next section shows how to remove the dependency.

Booting Process

The Fig. 2 depicts the functional design of the booting process of the proposed design.

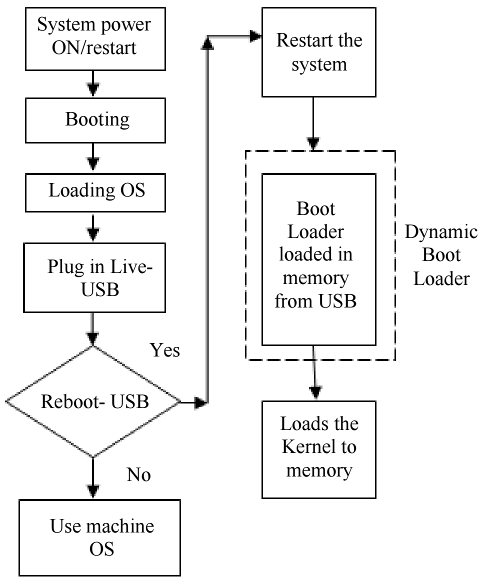
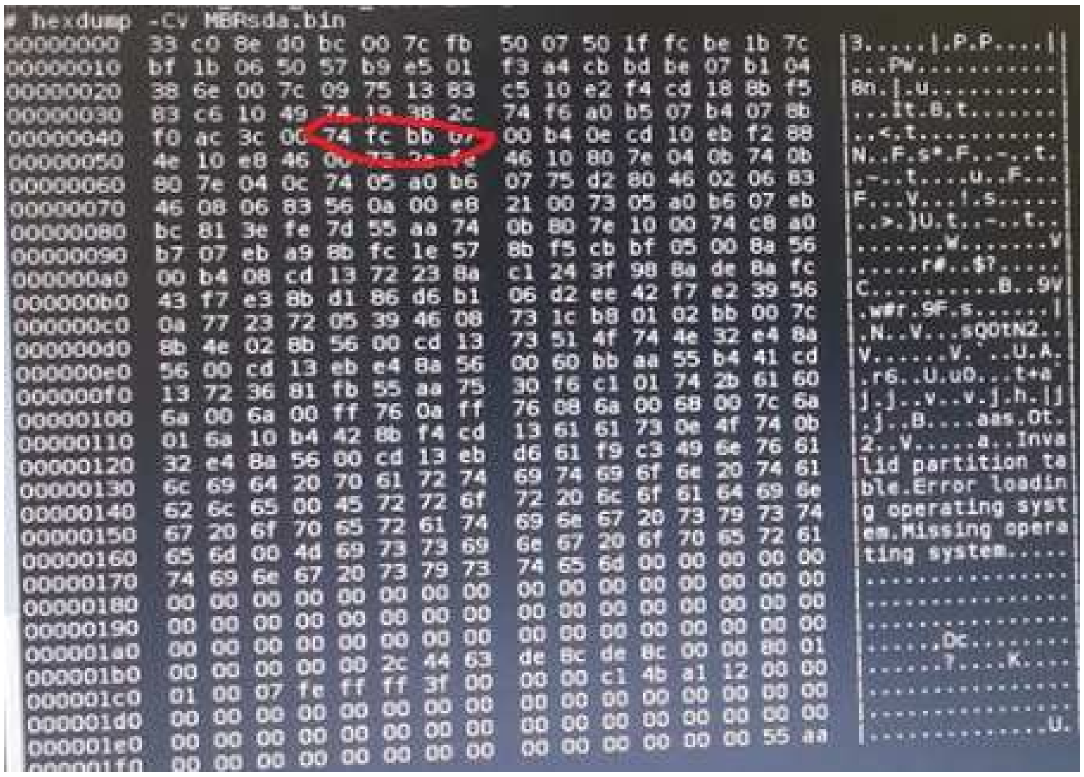


Fig. 2: Booting from USB



```
# hexdump -Cv MBRsda.bin
00000000  33 c0 8e d0 bc 00 7c fb  50 07 50 1f fc be 1b 7c  |3.....|.P.P....||
00000010  bf 1b 06 50 57 b9 e5 01  f3 a4 cb bd be 07 b1 04  |...PW...........|
00000020  38 6e 00 7c 09 75 13 83  c5 10 e2 f4 cd 18 8b f5  |8n.|.u..........|
00000030  83 c6 10 49 74 19 38 2c  74 f6 a0 b5 07 b4 07 8b  |...It.8,t.......|
00000040  f0 ac 3c 00 74 fc bb 07  00 b4 0e cd 10 eb f2 88  |..<.t...........|
00000050  4e 10 e8 46 00 73 2a fe  46 10 80 7e 04 0b 74 0b  |N..F.s*.F..~..t.|
00000060  80 7e 04 0c 74 05 a0 b6  07 75 d2 80 46 02 06 83  |.~..t....u..F...|
00000070  46 08 06 83 56 0a 00 e8  21 00 73 05 a0 b6 07 eb  |F...V...!.s.....|
00000080  bc 81 3e fe 7d 55 aa 74  0b 80 7e 10 00 74 c8 a0  |..>.}U.t..~..t..|
00000090  b7 07 eb a9 8b fc 1e 57  8b f5 cb bf 05 00 8a 56  |.......W.......V|
000000a0  00 b4 08 cd 13 72 23 8a  c1 24 3f 98 8a de 8a fc  |.....r#..$?.....|
000000b0  43 f7 e3 8b d1 86 d6 b1  06 d2 ee 42 f7 e2 39 56  |C..........B..9V|
000000c0  0a 77 23 72 05 39 46 08  73 1c b8 01 02 bb 00 7c  |.w#r.9F.s......||
000000d0  8b 4e 02 8b 56 00 cd 13  73 51 4f 74 4e 32 e4 8a  |.N..V...sQOtN2..|
000000e0  56 00 cd 13 eb e4 8a 56  00 60 bb aa 55 b4 41 cd  |V......V.`..U.A.|
000000f0  13 72 36 81 fb 55 aa 75  30 f6 c1 01 74 2b 61 60  |.r6..U.u0...t+a`|
00000100  6a 00 6a 00 ff 76 0a ff  76 08 6a 00 68 00 7c 6a  |j.j..v..v.j.h.|j|
00000110  01 6a 10 b4 42 8b f4 cd  13 61 61 73 0e 4f 74 0b  |.j..B....aas.Ot.|
00000120  32 e4 8a 56 00 cd 13 eb  d6 61 f9 c3 49 6e 76 61  |2..V.....a..Inva|
00000130  6c 69 64 20 70 61 72 74  69 74 69 6f 6e 20 74 61  |lid partition ta|
00000140  62 6c 65 00 45 72 72 6f  72 20 6c 6f 61 64 69 6e  |ble.Error loadin|
00000150  67 20 6f 70 65 72 61 74  69 6e 67 20 73 79 73 74  |g operating syst|
00000160  65 6d 00 4d 69 73 73 69  6e 67 20 6f 70 65 72 61  |em.Missing opera|
00000170  74 69 6e 67 20 73 79 73  74 65 6d 00 00 00 00 00  |ting system.....|
00000180  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000190  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
000001a0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
000001a0  00 00 00 00 00 00 2c 44  63 de 8c de 8c 00 00 80 01  |......,Dc.......|
000001b0  00 00 00 00 00 2c 44 63  de 8c de 8c 00 00 80 01  |......,Dc.......|
000001c0  01 00 07 fe ff ff 3f 00  00 00 c1 4b a1 12 00 00  |......?....K....|
000001d0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
000001e0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 55 aa  |..............U.|
000001f0  00 00 00 00 00 00 00 00
```

Fig. 3: Stage 1 memory hexdump

Dynamic Boot Loader

The dynamic boot loader consists of stage1, stage2 and w32grub and runs from USB on reboot.

W32Grub

W32grub is a windows version of grub boot loader to boot Linux based OS from windows. On yes to reboot from USB, the w32grub installs the grub in the windows boot sector, calculates the stage2 address and stores sector address in the stage 1 at address 0044-0047. Stage 1 loads stage 2 using this sector number. W32grub displays the below information:

- stage1< 2075182704>
- stage2_address = 0×8000
- stage2_sector= 2075182704
- stage2_segment = 0×800
- stage2< 2075182704+223>

The stage 2 sector absolute address is stored as 0×7bb0c74f in stage 1 at address 0044-0047. The following commands display the 512 bytes of the USB MBR.

- # sudo dd if = /dev/sdb1 of = /mnt/linux.bin bs = 512 count = 1
- #hexdump –Cv /mnt/linux.bin

The Fig. 3 shows the hexdump of stage1 indicating the stage2 address.

Stage 1

The windows boot loader stage 1 is in the Master Boot Record (MBR) which is the first sector of hard disk. The MBR is of size 512 bytes of which 446 bytes is the stage 1 boot loader, 64 bytes is the partition table which is 4 primary partitions of 16 bytes entries each and remaining 2 bytes which is 0xAA55 is boot record signature.

The above hexdump shows the stage 1 with relocatable address code which during the boot process gets loaded at address 0×7C00. On selecting the GRUB boot loader, the stage 2 is loaded in memory at address 0×8000.

Stage 2

Stage2 initializes the grub shell and loads the configuration file menu.lst from the directory boot/grub. The configuration file must contain the directory path and filename of the active partition where the kernel is stored and partition containing intird. Below are the configuration files where menu.lst is used for grub and grub.cfg file for grub2 for booting from USB.

<u>menu.lst</u>

timeout 60

```
default 1
# For booting Windows
title Windows
unhide (hd0,0)
rootnoverify (hd0,0)
chainloader +1
# For booting Linux
title Puppy Linux
rootnoverify (hd1,0)
kernel (hd1,0)/vmlinuz root=/dev/sdb1
initrd (hd1,0)/initrd.gz
```

<u>grub.cfg</u>

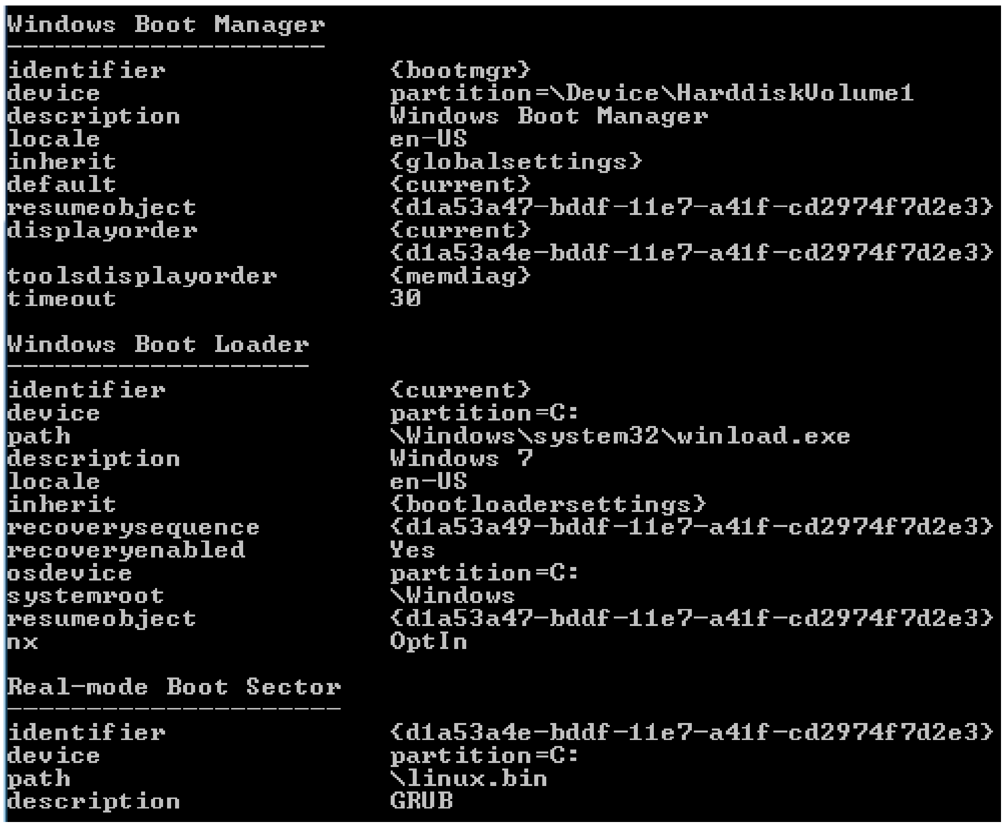
```
set timeout 60
set default = 0
# For booting Windows
menuentry "Windows "
{
set root = (hd0,1)
chainloader +1
}
# For booting Linux
menuentry "Puppy Linux"
{
rootnoverify (hd1,1)
linux (hd1,1)/vmlinuz root=/dev/sdb1
initrd (hd1,1)/initrd.gz
}
```

Adding Boot Entries

The Boot Configuration Data (BCD) is the data store that represents windows boot manager, boot loader and other boot applications as objects (GUIDs) (https://docs.microsoft.com/en-us/previous-versions/windows/it-pro/windows-server-2008-R2-and-2008/ee221031(v=ws.10)). Using BCD WMI (Windows Management Instrumentation) or bcdedit, the menu is edited to add GRUB as new entry.

The below bcdedit commands creates a GRUB entry in the boot menu as shown in Fig. 4, assigns the first 512 bytes, Stage 1 which is the linux.bin to GRUB.

- bcdedit/create/d "GRUB" /Application Bootsector
- bcdedit/set {d1a53a4e-bddf-11e7-a41f-cd2974fd2e3} device partition = C:
- bcdedit/set {d1a53a4e-bddf-11e7-a41f-cd2974fd2e3} path\linux.bin
- bcdedit/displayorder {d1a53a4e-bddf-11e7-a41f-cd2974fd2e3}/addlast
- bcdedit/timeout = 30



```
Windows Boot Manager
--------------------
identifier              {bootmgr}
device                  partition=\Device\HarddiskVolume1
description             Windows Boot Manager
locale                  en-US
inherit                 {globalsettings}
default                 {current}
resumeobject            {d1a53a47-bddf-11e7-a41f-cd2974f7d2e3}
displayorder            {current}
                        {d1a53a4e-bddf-11e7-a41f-cd2974f7d2e3}
toolsdisplayorder       {memdiag}
timeout                 30

Windows Boot Loader
-------------------
identifier              {current}
device                  partition=C:
path                    \Windows\system32\winload.exe
description             Windows 7
locale                  en-US
inherit                 {bootloadersettings}
recoverysequence        {d1a53a49-bddf-11e7-a41f-cd2974f7d2e3}
recoveryenabled         Yes
osdevice                partition=C:
systemroot              \Windows
resumeobject            {d1a53a47-bddf-11e7-a41f-cd2974f7d2e3}
nx                      OptIn

Real-mode Boot Sector
---------------------
identifier              {d1a53a4e-bddf-11e7-a41f-cd2974f7d2e3}
device                  partition=C:
path                    \linux.bin
description             GRUB
```

Fig. 4: Windows boot menu

The above commands creates an entry with GRUB and when selected provides options for both windows and Puppy Linux booting.

Kernel

Kernel is a self extracting compressed image. The kernel vmlinz gets loaded from /dev/sdb1 which is the compressed bzImage at address 0x3a00 and loads the temporary root file system initrd at address 0x37eaa000. The bzImage is then decompressed to ramdisk and kernel is booted.

Init

The init in/sbin/init is the first process to run. The last step is the user space initialization function. The initrd is unmounted and real file system is loaded and system starts working as per the system runlevel under/etc/inittab.

WMI

Windows Management Instrumentation (WMI) is Microsoft technology to retrieve information about disk drive, network configurations, Operating system and other internal state of computer systems. It uses query language to query and update any section of the WMI repository (https://msdn.microsoft.com/en-us/library/aa384642(v=vs.85).aspx).

WMI query along with VB script is written to access information regarding USB drive number, host system details, BCD store edit and also to reboot the system. Below are few queries which retrieve the required windows information:

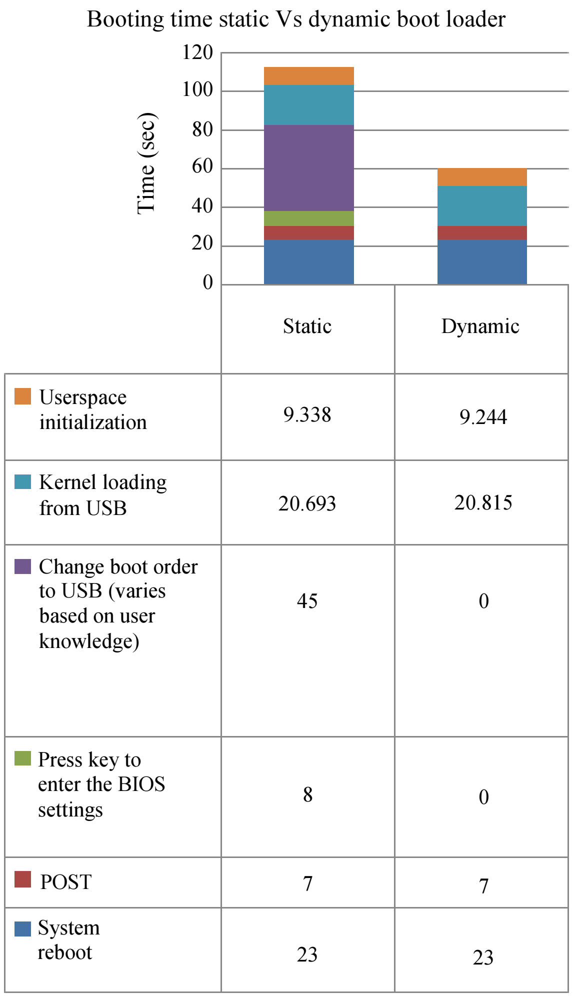
- "SELECT * FROM Win32_OperatingSystem where Primary = true"- To reboot the system using Reboot () method under the win32_operating system class.
- "SELECT * FROM Win32_LogicalDisk where DriveType = 3" – To get the properties of device drive.
- "SELECT * FROM Win32_ComputerSystemProduct" – To get the properties about the computer system like Name, Version, UUID, etc.

Results

The dynamic boot loader designed completely eliminates the dependency on boot loader and the need for user knowledge on boot order change. The dynamic boot loader is implemented using VC++, VB script and WMI and the experimental result shows the dynamic boot loader takes less booting time when compared with the static boot loader. The minimum requirement for USB is 2.0 with size of 8GB with 4 GB as persistent storage to make it portable.

The booting time is measured using a stopwatch from the time the host system is rebooted to boot from USB. The experiment was conducted on computer machine with 8 GB RAM, i5 processor, 64-bit Operating system and windows 7.

The booting time for both existing and dynamic boot loader is tabulated as shown in Fig. 5. The time in seconds is measured and the result shows that more than 50% booting time improvement is attained when using the dynamic boot loader to boot from USB. Maximum time is spent in entering the CMOS settings and changing the boot order. This time is dependent on the user knowledge of changing the boot priority. The dependency time can be eliminated using the dynamic boot loader. The booting time measured may differ depending on the processor speed and specification but in all cases the dynamic boot loader eliminates the time spent in boot order change.



	Static	Dynamic
🟧 Userspace initialization	9.338	9.244
🟦 Kernel loading from USB	20.693	20.815
🟪 Change boot order to USB (varies based on user knowledge)	45	0
🟩 Press key to enter the BIOS settings	8	0
🟥 POST	7	7
🟦 System reboot	23	23

Fig. 5: Boot time comparison chart

Table 1: List of OS experimented with dynamic boot loader

Operating system		Dynamic boot loader: Portable	Dynamic boot loader: Installation medium
Windows OS	Win XP	x	√
	Win Vista	x	√
	Win 7	x	√
	Win 8	x	√
	Win 10	x	√
GNU/Linux distributions	Ubuntu	√	√
	Puppy Linux	√	√
	Fedora	√	√
	Debian	√	√
	Linux Mint	√	√
	Kubuntu	√	√
	Lubuntu	√	√

The above graph shows that the Live USB along with dynamic boot loader greatly reduces the booting time and makes it an excellent replacement as a portable device.

The designed boot loader works with any version of Windows Operating system. The user needs to plug in and reboot the system. The advantage of dynamic boot loader is that it supports USB both as a portable system and as an installation media for different type of OS.

Several experiments were conducted to test the working of the dynamic boot loader both as a portable medium and as an installation resource. The dynamic boot loader works efficiently with open source OS both for portable OS and installation medium whereas Windows OS only for installation medium. Both 32-bit and 64-bit OS's are supported. The results are tabulated as shown in Table 1.

The experiment conducted on few distributions of Linux OS as shown in the table and its scope is not limited to only the listed OS.

Conclusion

The Live USB increases the lifetimes of the USB to a great extent, since all writes are done in ram and saved only once in the USB on shutdown. The user can easily carry the portable USB anywhere and plug in and start running the customized session in less boot time. Implementing the dynamic boot loader concept allows any novice to use the portable USB system to run smoothly on any machine without worry about the knowledge of the BIOS settings. The designed system along with the fingerprint authentication will provide a highly secure and portable system.

Future Enhancements

The Live USB is vulnerable to attacks when used with other system. Biometric authentication is on high demand particularly for portable device. The Live USB with dynamic boot loader being portable can be integrated with fingerprint authentication to make the system a highly secure portable system.

The USB is divided into 2 partitions with the first partition made read only and second partition the secure one protected with fingerprint. The read only partition contains the dynamic boot loader and fingerprint program. The private partition contains the kernel file and storage space for storing secured data. On reboot, it prompts the user to scan the fingerprint. The authorization process matches the fingerprint pattern against each of the stored pattern. On successful authorization the secured partition opens and kernel starts loading into RAM of the host machine.

The USB drive when used in a host environment, the transfer of data between host system and USB makes the data vulnerable for attacks. Since the OS runs from the USB and not dependent on the operating system of the machine, the proposed design achieves the security for the data stored as well as make the environment secure. The kernel image can be customized as per the requirement of the user. This will greatly improve the loading time of the OS to the host machine.

Acknowledgment

I sincerely thank my guide Dr. K. Siva Sankar for his continuous guidance and support for the successful completion of this work.

Author's Contributions

Alycia Sebastian: Design, conduct experiments, data analysis and manuscript writing.

Dr. K. Siva Sankar: Design analysis, interpretation of data and review manuscript writing.

Ethics

As corresponding author, I assure whole or any part of the manuscript is not under consideration for publication in any journal.

References

https://docs.microsoft.com/en-us/previous-versions/windows/it-pro/windows-server-2008-R2-and-2008/ee221031(v=ws.10)

https://msdn.microsoft.com/en-us/library/aa384642(v=vs.85).aspx

Jebarajan, T. and K.S. Sankar, 2011. A method for developing an operating system for plug and play bootstrap loader for USB drive. Int. J. Comput. Sci., 8: 295-301.

Kanahasabapathy, S.S., J. Thanganadar and P. Lekshmikanthan, 2015. A novel approach to develop dynamic portable instructional operating system for minimal utilization. Int. Arab J. Inform. Technol., 12: 780-784.

Karna, A.K. and Y. Chen, 2014. Multi-operative USB HD: An all-in-one solution to IT supports and forensic experts. J. Software, 9: 847-858. DOI: 10.4304/jsw.9.4.847-858

Karna, A.K., 2010. Multipurpose USB hard disk: Your mini laptop. Proceedings of the International Conference on Information Technology and Computer Science, pp: 357-360. DOI: 10.1109/ITCS.2010.93

Kushwaha, A.S., 2013. A trusted bootstrapping scheme using USB key based on UEFI. Int. J. Comput. Commun. Eng., 2: 543-546. DOI: 10.7763/IJCCE.2013.V2.245

Sankar, K.S., 2010. A method for developing bootstrap loader and configuring network for live USB flash drive. Int. J. Decis. Mak. Supply Chain Logist., 1: 221-232.

Sebastian, A. and K.S. Sankar, 2015. Design of a boot loader for operating system. Austral. J. Basic Applied Sci., 9: 368-374.

Sivaiah, B., T.S.N. Murthy and T. Vandana Babu, 2014. Boot multiple operating systems from ISO images using USB disk. Proceedings of the International Conference on Electronics and Communication Systems, Feb. 13-14, IEEE Xplore Press, Coimbatore, India, pp: 1-5. DOI: 10.1109/ECS.2014.6892602