A Novel Approach to Develop Dynamic Portable Instructional Operating System for Minimal Utilization

Siva Sankar Kanahasabapathy[1], Jebarajan Thanganadar[2], and Padmasuresh Lekshmikanthan[3]
[1]Department of Information Technology, Noorul Islam University, India
[2]Department of Computer Science and Engineering, Rajalekshmi Engineering College, India
[3]Department of Electrical and Electronics Engineering, Noorul Islam University, India

Abstract: *Most well-known instructional Operating Systems (OSs) are complex, particularly if their companion software is taken into account. It takes considerable time and effort to craft these systems and their complexity may introduce maintenance and evolution problems. The purpose of this paper is to develop a mini OS which is open source and Linux based. This OS is independent of hardware simulator and platform. It encompasses a simplified kernel occupying a low memory with minimal resource consumption. It also, includes a dynamic boot loader which ignores the BIOS priority, takes itself to be as a highest priority. It is designed to utilize low primary memory and minimal CPU utilization. This is developed mainly to satisfy the minimal and basic requirements for a normal desktop user.*

Keywords: *OSs, kernel, boot loader, linux, portable, opensource*.

Received July 23, 2012; accepted May 19, 2015; published online September 15, 2015

1. Introduction

In recent years the popularity of opensource Operating Systems (OSs) has increased dramatically as the computer users have discovered its many benefits. Commercial OS from Windows and Mac meets a severe competition with Linux OSs. Boot loader is a small program running before OS is invoked. The GRUB and LILO loaders have provided a strong background for the Linux family. These loader programs loads the kernel of the OS into the main memory for execution. A row-canonical innovation model is used in these bootstraps to locate their state [1]. Opensource has entered every business and normal activities of ordinary users. But when these OSs were taken to systems of different configurations some changes have to be produced [11, 14, 19]. This led to the development of portable OSs. The boot sectors in OSs are 512bytes long [19] and used to initiate the boot process. The kernel functions a major role here and designed to be the heart of these opensource OSs. Kernel is a program that resides in the memory, takes in user inputs, process those inputs and give out a suitable output. OSs relies basically on the BIOS priority [14]. It is true and continuing for the Linux OSs too. Here, occurs a dependency for OSs even they are well-built with much better performance. In this paper, we have proposed a solution to overcome this static boot process. The OS encompassed this dynamic boot is light-weight and satisfy the minimal user requirements.

2. Related Works

Ahmed and Sait [1] proposed a bootstrap algorithm and applied modification to self-tuning control for MIMO systems. Changes have to be induced for systems of different configurations. This seems to be a tedious process. In the same year, Rogers and Schulwitz [15] developed an architecture for distributed system to handle systems of various configurations (MULTIBUS II system). To overcome this Ke *et al.* [8] proposed portable OSs. Boot sectors in OSs are vulnerable to boot sector virus attacks. Tesauro *et al.* [18] from IBM T.J. of Watson Re-search Center used a neural networks scheme on boot sectors to prevent the attack of boot straps from boot-sector viruses. During these development stages OSs awaited for a code reuse and common framework. Scott and Davidson [16] introduced portable Safe Virtual Execution (SVE) framework, called Strata to target on SPARC/Solaris, x86/Linux and MIPS/IRIX machines. The data sets in this boot strap get corrupted easily. De Lacerda *et al.* [4] proposed a bootstrap technique that was implemented using genetic algorithms to detect true errors in bootstrap data sets. This boot strap techniques vary with the complexity of kernels in the OSs. Perez and Vila [14] developed a Real-Time Linux GPL (RT Linux) which is a small, deterministic, real-time kernel that handles time-critical tasks and makes use of Linux services in particular TCP/IP networking. Minnich [12] from Los Alumos National Lab used an own made bootstrap which reduced the risk of locating, verifying and loading a new OS image

because here OS boots and the other OS systems work. As netbooks and other portable devices evolved the concept of light weight OSs was introduced. Dunkels *et al.* [5] developed Contiki, a lightweight OS with support for dynamic loading and replacement of individual programs and services. Contiki is built around an event-driven kernel. Guo *et al*. [7] introduced the concept of network boot in which client is loaded from the server from one of the virtual disks, the client being loaded without any local memory. To ensure the integrity of files in OS Gu and Ji [6] proposed a secure bootstrap in which each time when the kernel is booted, the integrity of files and code are measured first, when these bootstraps were secured developers aimed for a secure light weight kernel to improve the efficiency of OSs. Nurnberger *et al*. [13] proposed a secure micro kernel from scratch called Ray keeping security features in mind. Lohr *et al.* [10] introduced a secure boot process in which the loader verified the integrity of the software before boot starts and access the resources. Zhang and Shao [21] implemented a boot loader module based on the new high-security OS with internal networking structure netOS-I.

Every works and researches maintained security and improved the booting performance of OSs to a large extent. All the research works focused only on a static booting via BIOS priorities. But this may be a difficult task for the users as they have to depend mostly on BIOS priorities. So, we are proposing a light weight OS embedded with a dynamic loader which would be more convenient for the users to have an independent work against this BIOS priority. The light weight OS is developed to satisfy the minimal requirement for a normal desktop user.

3. Design of Boot Loader

The proposed system presented in this paper is a design of boot loader which can be divided into different logical modules [12] they are boot loader, temporary file system and initial ram disk.

3.1. Boot Loader

Boot loader program loads the kernel of the OS into the main memory for execution [11]. The boot loader must be of size 512bytes and should reside in the first sector of the disk drive. The procedure of boot loader [6] is as follows: Check the boot signature 0AA55h at 10,511thbytes of the first sector of boot disk. If boot signature is present, it loads the code present in the first sector (512bytes) to memory address 07c00h. Next the code at 07c00h is executed. This code then tries to find the available physical memory and divides it into 64KB pages. After that, 2KB boot stack is allocated at (A0000-512)h and stack pointer is setup. Then space for Interrupt Vector Table (IVT) and BIOS routines are reserved and kernel is loaded at 00600h. The kernel that is to be loaded can be an EXE, BIN or COM file. Search for this kernel file will be conducted in the root directory (19th sector of the boot disk). On getting the file, it is allocated properly with all needed segments and memory pointers. If the kernel is in BIN or COM format it will have a single segment with all Data Segment (DS), Code Segment (CS), Extra Segment (ES), Stack Segment (SS) integrated. If the kernel is in EXE format, it will have separate code, data, extra and stack segments. In such cases, the exe header will be ripped off and proper relocation factors are added as needed. Virtually, boot loader follows two stages of loader they are primary and secondary boot loaders. After this, the loaded kernel is executed.

3.2. Stage 1 Boot Loader

The primary boot loader that resides in the MBR is a 512byte image containing both program code and a small partition table. The first 446bytes are the primary boot loader, which contains both executable code and error message text. The next 64bytes are the partition table, which contains a record for each of four partitions (16bytes each).

The MBR ends with two bytes that are defined as the magic number (0xAA55). The magic number serves as a validation check of the MBR. Figure 1 shows the three logical section of a MBR. After BIOS finishes initializing platform hardware devices [3], it will load OSloader Bootstrap Measurement Module (OSBMM) into host memory and then the measurement code part of OSBMM will analyze the measurement information to check the integrity of MBR, boot sectors and OS Loaders. If the verifications pass, OSBMM will load MBR into 0000H: 7C00H and give controls to MBR to perform the normal bootstrap. If the verification fails, it will read backup from hard disk and recover the tampered files and code.

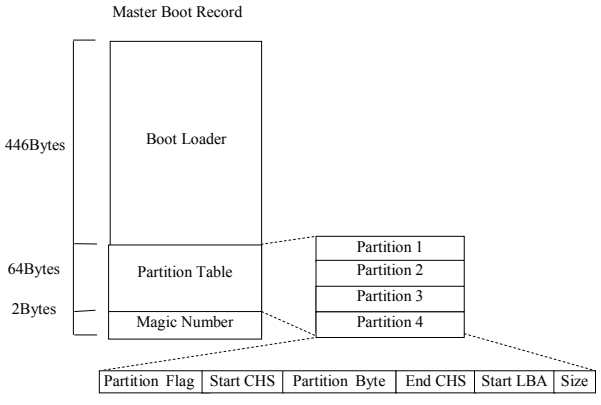

Figure 1. Logical sections of an MBR.

Normally, the problems with the MBR of a system may prevent the system from booting. The MBR may be affected by malicious code, become corrupted by disk errors or being overwritten by other boot loaders when experimenting with multiple OSs on a host. This recipe describes one method of repairing the MBR using the recovery console. The master boot record can be taken as a backup and can restore it later.

The job of the primary boot loader is to find and load the secondary boot loader (stage 2). It does this by

looking through the partition table for an active partition.

When it finds an active partition, it scans the remaining partitions in the table to ensure that they are all inactive. When this is verified, the active partition's boot record is read from the device into RAM and executed.

3.3. Stage 2 Boot Loader

The second stage boot loader loads the compressed kernel image into memory and the kernel then setups the environment and starts to manage the resources, and then the boot loader places the appropriate root file-system image into memory. Once, the kernel and the root file-system images are loaded into memory, the boot loader holds the whole control of machine to the kernel.

It does this by making the two-stage boot loader into a three-stage boot loader. Stage 1 (MBR) boots a stage 1.5 boot loader that understands the particular file system containing the Linux kernel image. Examples include reiserfs_stage1_5 (to load from a reiser journaling file system). When the stage 1.5 boot loader is loaded and running, the stage 2 boot loader can be loaded.

In stage 2, it will copy itself into SDRAM. In our implementation, after the initialization in stage 1, kernel of USB drive will be copied from flash to offset 0x0000, 0000 of memory. This is the real boot program. It contains the user interface and the kernel loader.

With stage 2 loaded, it can display a list of available kernels defined in the boot menu. Normally, you can select a kernel and even amend it with additional kernel parameters. Here, by default it loads the live USB automatically. Optionally, with the end user interaction it can able to boot the remaining kernel system. With the second-stage boot loader in memory, the file system is consulted, and the default kernel image and initrd image are loaded into memory. With the images ready, the stage 2 boot loader invokes the kernel image.

4. Operating System

The "kernel" of the opearating system doesn't have to be a Linux kernel; it can be a boot sector or a COMBOOT file. Chain loading requires the boot sector of the foreign OS to be stored in a file in the root directory of the file system. Because neither Linux kernel boot sector images, nor COMBOOT files have reliable magic numbers, Syslinux will look at the file extension.

4.1. SysLinux

Syslinux is the boot loader for the OS that is developed. It is intended to simplify first-time installation of Linux and for creation of rescue and other special purpose boot disks. SysLinux eliminates the need for distribution of raw boot floppy images. SysLinux will alter the boot sector on the disk and copy a file named ldlinux.sys into the root directory. The extensions that are recognized can be described below.

4.2. Kernel Compilation

Kernel is a nothing but a program that resides in the memory, takes in user inputs, process those user inputs and give out a suitable response. Kernel can be EXE, BIN or COM file. To reduce the kernel size and have more flexibility it is very important to have a detailed knowledge of kernel modules at the time of kernel compilation. Kernel compilation involves configuring the kernel, building the kernel and installing the new kernel.

4.3. Configuring the Kernel

Configuring the kernel means we are making the kernel to suit to our hardware requirements. Configuration makes use of 'make menuconfig' or 'make xconfig' commands. In order to have a default configuration specified by the OS we can use 'make defconfig' command.

4.4. Building the Kernel

Now, there is a need to check the dependency relations between various modules using 'make dep' command. Since, we are developing an OS from scratch there is a need to use 'make clean' and 'make proper' commands for deleting the temporary loaded object files and other build. Finally, we can obtain a compressed kernel using 'make bzimage' ie a vmlinux.

4.5. Installing the Kernel

In this process, we select the various modules for our system and the necessary drivers for the system. For selecting the required modules use 'make module' which prompts for the necessary modules to install and use make modules_install to install the necessary modules in the system.

4.6. Temporary File System

Temporary file system (tmpfs) is one of the ramdisk technologies in Linux. A RAM disk is a portion of RAM which is being used as if it were a disk drive. Access time is much faster for a RAM disk than for a real, physical disk. Thus, putting the files into memory will increase the performance of computer. tmpfs distinguishes itself from the Linux ramdisk device by allocating memory dynamically and by allowing less-used pages to be moved onto swap space. These characteristics make tmpfs more flexible than RAMFS, MFS and some older versions of ramfs. In this study, tmpfs is adopted to create a RAM disk for storing the memory running system during system boot process.

tmpfs is supported by the Linux kernel from version 2.4 and we are using the kernel version 2.6 [2].

4.7. Initial Ram Disk

Initrd, the initial ramdisk, is a temporary file system commonly used in the boot process of the Linux based kernel. It is typically used for making preparations before the real root file system can be mounted [20]. Initrd provides the capability to load a RAM disk by the boot loader. This RAM disk can then be mounted as the root file system and programs can be executed. Next a new root file system can be mounted from a different device. The previous root (from initrd) is then moved to a directory and can be subsequently unmounted [9]. Linux based OS is booted from a external USB drive and initrd makes preparations for shifting to real root file system resides in a USB drive.

In order to, shift to real root file system resides in a RAM disk which is created with tmpfs, initrd need to be customized to perform some tasks as follows:

- Create a tmpfs_based RAM disk.
- Release the USB memory running system which stores in the USB flash disk to a tmpfs_based RAM disk.
- Shift to the USB running system which resides in a tmpfs based RAM disk.

Normally, kernel loads along with their modules and it is shown in the Figure 2. It describes the initiation of different stages of boot loader and the OS loading stages. In the next chapter, we concluded the result.

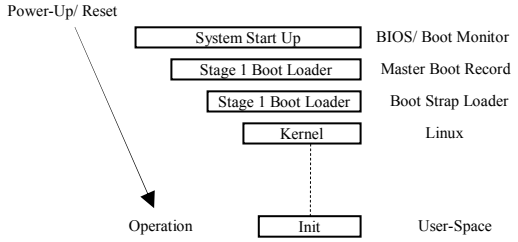

Figure 2. Process involved in booting of live-USB.

5. Conclusions

The role of boot loader is significant in the world of OSs. While taking the existing OSs into account, they are usually static in nature. They could load the OSs only when the BIOS priority is provided for the device that contains the required OSs. A serious disadvantage here is when two or more OSs reside on various storage devices we can't achieve the OSs to load until the high priority is given to the storage device that contains the OSs. This threat was overcome by the implementation of a dynamic boot loader. The dynamic boot loader displays a menu listing live USB OS and the list of existing OSs installed. In case of normal OSs, consume more secondary, primary memories and much CPU resource. This was greatly reduced in this light-weight OS and could satisfy all the minimal requirement of a normal desktop user.

References

[1] Ahmed M. and Sait N., "State-Space Adaptive Control through a Modified Bootstrap Algorithm for Parameter and State Estimation," *IEEE Proceedings D, Control Theory and Applications*, vol. 136, no. 5, pp. 215-224, 1989.

[2] Albazaz D., "Design Mini-Operating System for Mobile Phone," *the International Arab Journal of Information Technology*, vol. 9, no.1, pp. 56-65, 2012.

[3] Bovet D. and Cesati M., *Understanding the Linux Kernel*, O'Reilly press, 2005.

[4] De Lacerda E., de Carvalho A., and Ludermir T., "A Study of Cross-validation and Bootstrap as Objective Functions for Genetic Algorithms," *in Proceedings of the 7th Brazilian Symposium on Neural Networks*, Pernambuco, Brazil, pp. 118-123, 2002.

[5] Dunkels A., Gronvall B., and Voigt T., "Contiki-A Lightweight and Flexible Operating System for Tiny Networked Sensors," *in Proceedings of the 29th Annual IEEE International Conference on Local Computer Networks*, FL, USA, pp. 455-462, 2004.

[6] Gu J. and Ji W., "A Secure Bootstrap based on Trusted Computing," *in Proceedings of International Conference on New Trends in Information and Service Science*, Beijing, China, pp. 502-504, 2009.

[7] Guo G., Zhang Y., Zhou Y., and Yang L., "Performance Modeling and Analysis of the Booting Process in a Transparent Computing Environment," *in Proceedings of the 2nd International Conference on Future Generation Communication and Networking*, Hainan Island, China, pp. 83-88, 2008.

[8] Ke P., Gang Z., and Fu-jiang L., "Design of Boot Loader with Multiple Communication Port," *in Proceedings of the 2nd International Conference on Computer Science and Software Engineering*, Hubei, China, pp. 169-175, 2008.

[9] Liu J., "The Design of Booting Program Based on ARM Linux Embedded System [J]," *Control and Automation Publication Group*, vol. 22, no. 2, pp. 123-125, 2006.

[10] Lohr H., Sadeghi A., and Winandy M., "Patterns for Secure Boot and Secure Storage in Computer Systems," *in Proceedings of International Conference on Availability, Reliability and Security*, Krakow, Poland, pp. 569-573, 2010.

[11] Miedlar M., Bauer S., Powers P., "The Portable Operating System," in Proceedings of National Aerospace and Electronics Conference, Ohio, USA, pp. 693-698, 1995.

[12] Minnich R., "Give Your Bootstrap the Boot: using the Operating System to Boot the Operating System," *in Proceedings of International Conference on Cluster Computing*, California, USA, pp. 439-448, 2004.
[13] Nurnberger S., Feller T., and Huss S., "Ray-A Secure Micro Kernel Architecture" *in Proceedings of the* 8th *Annual International Conference on Privacy*, *Security and Trust*, Ottawa, Canada, pp. 3-6, 2010.
[14] Perez S. and Vila J., "Building Distributed Embedded Systems with RTLinux-GPL," *in Proceedings of Conference on Emerging Technologies and Factory Automation*, Lisbon, Portugal pp. 161-168, 2003.
[15] Rogers S. and Schulwitz L., "Reconfiguration Architecture for Distributed Processing," *in Proceedings of the 34th International Conference on Intellectual Leverage*, *Digest of Papers*, CA, USA, pp. 545-551, 1989.
[16] Scott K. and Davidson J., "Safe Virtual Execution using Software Dynamic Translation," *in Proceedings of the 18th Annual Computer Security Applications Conference*, Nevada, USA, pp. 209-218, 2002.
[17] Stein L., "Stupid File Systems Are Better," available at: https://www.usenix.org/legacy/event/hotos05/final_papers/full_papers/stein/stein.pdf, last visited 2012.
[18] Tesauro G., Kephart J., and Sorkin G., "Neural Networks for Computer Virus Recognition," *IEEE expert*, vol. 11, no. 4, pp. 5-6, 1996.
[19] Wikipedia, "initrd," available at: http://en.wikipedia.org/wiki/Initrd, last visited 2012.
[20] Wikipedia, "TMPFS," available at: http://en.wikipedia. Org/wiki/TMPFS.
[21] Zhang J. and Shao F., "A Bootloader Module is Designed and Implemented based on a New Computer Architecture," *in Proceedings of the 2nd International Conference on Future Networks*, Hainan, China, pp. 424-427, 2010.

Siva Sankar Kanahasabapathy obtained his doctorate from M S University. Currently, he is an Assistant Professor in Department of Information Technology, Noorul Islam University, India. He is well known for his contributions to the field in both research and education contributing over 26 research articles in Journal and Conferences. He also, served in many committees as Convener, Chair and Advisory member for various external agencies. Currently, his research is focused on system software and embedded systems.

Jebarajan Thanganadar obtained his doctorate from MS University. Currently, he is a Professor and Head in Department of Computer Science and Engineering, Rajalekshmi Engineering College, India. He is well known for his contributions to the field in both research and education contributing over 67 research articles in Journal and Conferences. He is the editorial member and also served as reviewer for various reputed journals. He has been a life member of the Computer Society of India. He also, served in many committees as Convener, Chair and Advisory member for various external agencies. Currently, his research is focused on networking and image processing.

Padmasuresh Lekshmikanthan obtained his doctorate from MS University and Dr. MGR University, respectively. He is presently working as a Professor and Head in Department of Electrical and Electronics Engineering, Noorul Islam University, India. he is well known for his contributions to the field in both research and education contributing over 50 research articles in journal and conferences. He is the editorial member of International Journal of Advanced Electrical and Computer Engineering and also served as reviewer for various reputed journals. He has been a life member of the Indian Society for Technical Education. He also served in many committees as Convener, Chair and Advisory member for various external agencies. Currently, his research is focused on artificial intelligence, power electronics, evolutionary algorithms, image processing and control systems.