Startup Time Optimization Techniques for Embedded Linux

PramodKumar Singh, Prof Swanili Karmore

Abstract— Embedded system performance and utilization has increased over years, these can be observed most obviously in the electronic consumer market once a mobile phone are now replaced by smart phones and internet tablets, once a car radios are now replaced by by In-Vehicle Infotainment Systems. More and more functionality is introduced into the once single-purpose system to utilize the increasing computational power, driven by the system's main target of providing improved services to the user. That implies an even faster growing complexity to be handled by the embedded systems and availability on demand. Operating system based on the Linux kernel are used in most of these consumer electronic devices, the user of these devices except these devices to be available for use very soon after being turned on. This leads to optimization of startup time for Linux. In this paper we represent various techniques available for optimization of the startup time for Linux for various different scenarios ranging from consumer level device like set-top boxes, to real time and mission critical system like industrial automation and medical instruments.

Index Terms— embedded, Linux, boot, optimization, kernel, system, embedded platform.

1 INTRODUCTION

Linux, developed by Linus Torvalds was specifically targeted to desktop PCs running an Intel 80x86 or compatible microprocessor. Today Linux has been ported to many different microprocessors and runs on all sorts of platforms, these devices are not even general purpose computer systems and include things such as network routers, heart monitors, and data collection units for tracking different weather pattern sensors from isolated unattended remote locations. It is these type of systems that have collectively come to be labeled as "Embedded Linux". Linux meets the requirements of everyone in all fields such as embedded, real time, personal computer in terms of functionality, scalability and cost. Linux supports all these features because of its configurable nature and hugely supported open source community of developer all across the world. Linux meets the requirements of everyone in all fields such as embedded, real time, personal computer in terms of functionality, scalability and cost. Various mission critical systems such as an aircraft control system or an application platform running on a communication node require system to be started up in few seconds to a minute. To achieve the fast startup in these system it is necessary to examine the boot process of the operating system used which is Linux.

Optimizing the Linux Bootup process should not override or change the existing functionality, nor the normal operation. On optimization we extend the system to a level where it reduced the time of the booting process, but fail to satisfy normal operation then whole process will be lost. Boot optimization should not affect the system functionality, but in turn help system to enhance its booting process for faster system upgrade.

• *Pramodkumar Singh is currently pursuing masters degree program in Embdded System and Computing in GHRCE, Nagpur, India, E-mail: pramod73@gmail.com*
• *Prof Swapnili Karmore, Department of Computer Science & Engineering, G.H.Raisoni College of Engineering,Nagpur, India E-mail: swapnili.karmore@raisoni.net*

2 EMBEDDED LINUX

Embedded system is defined to be an application-oriented special computer system designed to perform one or few dedicated functions (e.g. consumer electronics, in-vehicle system, industrial system)

Embedded Linux is defined to be an operating system which is built around the Linux kernel and is designed to be used in an embedded system

2.1 Embedded System and Desktop/Server system

Embedded System differ from the general purpose desktop/ Server system in applicability and hardware choices, below are some of the characteristic that differentiate and embedded system from any other general purpose platforms/ systems.

1. Limited hardware/ performance capabilities
2. Usually relatively static environment (target HW know at design time)
3. Large variety of different target hardware's (e.g. different processor architectures)
4. Restricted access to the system, users are generally can't install new software

2.2 Linux for embedded system

The open source Linux kernel based operating system has dominated the computing world and have now shown the same domination in the embedded system space, some of the advantage of Linux systems are

1. Low cost/ free (open source)
2. High performance (consistent in time)
3. Security (one of the most secure OS)
4. Stability (generally runs until hardware fails)
5. Scalability (handles small and large systems)
6. Flexibility (easily customized and optimized)
7. Compatibility/ Portability

8. Maintainability (regular updates)
9. Multitasking (true multitasking)
10. Multiuser (true multiuser)
11. Machine independent software environment
12. Supports almost all programming languages and a wide range of IDE's
13. Excellent technical support (community)

The flexibility and other Linux advantage are very much appreciated in the embedded system world, Linux kernel as embedded system allows the developer to utilize the vast amount of knowledge and components already available to embedded system world. Some of the advantages of using Linux for embedded systems are as follow

1. Allows to re-use components and can be freely duplicated on other devices(flexibility/ compatibility/ Portability)
2. Allows to have full control over the software part of the system (e.g. complete source codes available; debugging, changes, and optimizations easily possible)
3. Eliminates dependencies on third-party vendors (open-source kernel/ components)
4. Reduces licensing and development tools costs, possibly down to zero
5. Reduces development time (widely used and known, fast and active community support, plenty of ready-made components, large number of development tools available)
6. Offers high-quality components, widely used on millions of systems
7. Eases and speeds up testing of old and new features

3 BOOT PROCESS

The boot time for an embedded system is of paramount importance. To optimize the boot time of the Linux, let's understand the process of booting a system running Linux operating system, the Linux booting process consists of multiple stages as shown in the figure below

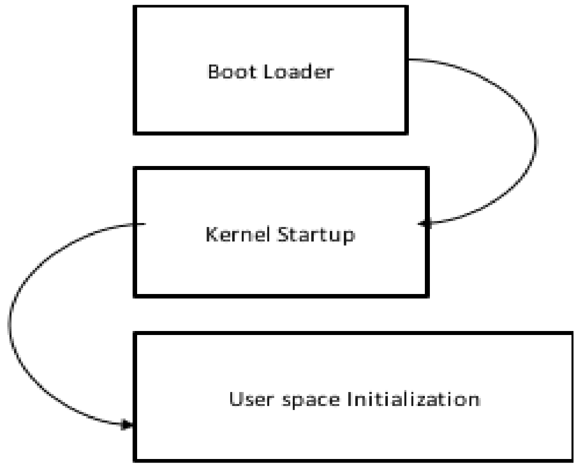

Fig. 1. High level linux boot process

3.1 Boot loader process

This process includes early hardware initiation and interaction and load the kernel from flash to RAM. The time take during this process can be described as

1. Power/ Clock Stabilization — usually negligible but should be considered
2. Low Level CPU Initialization - ~ 100 ms — Boot-loader (often multi-stage, ie secure boot)

3.2 Kernel Startup

This process does the following activities

1. Loading images (kernel, u-boot, rootfs, dtb)
   a. Usually from NOR or NAND Flash
   b. Compressed kernel
2. Subsystem (Driver) initialization
3. Mounting a root file system

3.3 User space

The user space process covers

1. Init scripts
2. System processes
3. Applications

The user space process and configuration are very user and applications dependent; the user space can have a display terminal or may not have a display terminal. The best example of the user space is the user interface of the Android operating system which is working over the Linux kernel.

4 OPTIMIZATION PROCESS

Linux boot optimizations methods are very platform and application dependent; the optimization need to consider the whole system architecture for selecting the boot optimization strategies e.g.

1. What software update methods will be used (affect features needed in boot loaders)
2. What are the essential functions of the device that must be running first?
3. When are network features needed
4. The difference between the production and development system images can also pose a different optimization need e.g. during the development more feature rich image might be needed and based on the development and production need the kernel configuration might be different (to enable easier debugging) for different version of the images.

Any optimization on the Linux booting process would fall into either of these 2 broad categories:

4.1 Size

The size dictates what would be kernel image size based on the available hardware or application, the size optimization process including

1. Reducing the size of binaries for each successive component loaded.
2. Removing features that are not required to reduce the size and complexity of the image.

4.2 Speed

The speed optimization process includes

1. Optimize for target processor
2. Use faster medium for loading primary, secondary boot loaders and kernel.
3. Reduce number of tasks leading to the boot.

The size and speed optimization process is dependent on the application need of the embedded system, if the system is designed to be working as an unattended remote sensor for weather collection then both the size and speed are of paramount the optimization process would require the identification of usability and then target time requirement, based on these criteria the optimization process for Size and Speed can be utilized.

A generic optimization process for the embedded system is depicted in the below diagram.

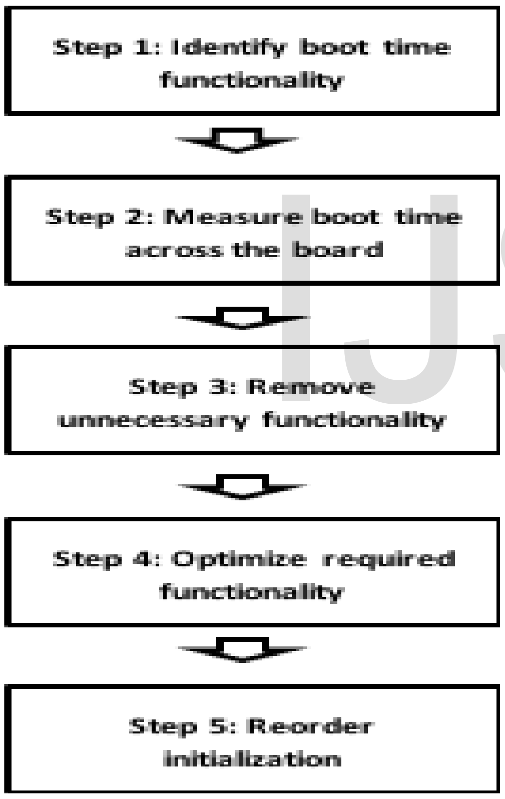
Step 1: Identify boot time functionality
Step 2: Measure boot time across the board
Step 3: Remove unnecessary functionality
Step 4: Optimize required functionality
Step 5: Reorder initialization

Fig. 2. Optimization Process

5 Various Optimization Technique

To achieve the optimal or fast startup time the Linux embedded system need to be optimized based on the optimization process mentioned in the above section, once the need and process is identified, the optimization can be categorized and done by below activities

5.1 Reducing kernel boot time

Kernel boot time can be reduced by performing some or all of these activities

1. Disable IP auto config
2. Reducing the number of PTYs
3. Disable console output
4. Preset loops_per_jiffy
5. kernel decompression
6. Reduce the kernel size
7. Faster rebooting
8. Copy kernel and initramfs from flash to RAM using DMA
9. Async initcall
10. Deferred initcalls

5.2 System startup time and application speed

System startup speed is dependent on multiple factors apart from the kernel, the filesystem, processor, IO and services, the optimization for startup time can be achieved by utilizing some or all the following activities.

1. Starting system services
2. Prefetching Reading ahead
3. Execute In Place (XIP)
4. Processor acceleration instructions
5. Use faster filesystems
6. Speed up applications with tmpfs
7. Boot from a hibernate image
8. Reducing disk footprint and RAM size of the Linux kernel
9. Replacing initrd by initramfs

5.3 Application size and RAM usage

The application size can have a detrimental impact on the whole embedded system, embedded system is generally starved for RAM and other hardware because of power consumption, size and other environmental constraints. The optimization of application size and RAM usage can be performed based on some or all of the following activities

1. Static or dynamic linking
2. Library Optimizer
3. Using a lighter C library
4. Compressing filesystems
5. Restartable applications
6. Merging duplicate files
7. Compiler space optimizations

5.4 Reducing power consumption

One of the major constraint of the embedded system is power consumption, embedded systems are used in some environments which are not suitable for general purpose computing, even human intervention are not possible (hazardous environment, factory automations) in this type of environment the embedded system sensors are supposed to work for longer duration without any need to power and other interventions. Power consumption optimization can be achieved by some or all of the activities

1. Tickless kernel
2. Cpufreq

3. Suspend hidden GUIs
4. Software suspend

6 Conclusion

Linux kernel is quite flexible and portable, even each application of embedded system need specific optimization of the Linux kernel based on the choice and availability of the embedded system hardware.

The startup time of the embedded Linux based are based on multiple factors and the embedded Linux boot time are

1. Highly dependent on choices made in HW
2. Hardware architecture and system architecture matter
3. Individual requirements vary based on the utility and application need

The reduction in startup time varies based of factors affecting different need and hardware of embedded system application.

Acknowledgment

The authors wish to thank all the Prof of Embedded System and Computing department of GHRCE, Nagpur, India for there support and encourgament.

References

[1] Heeseung Jo; Hwanju Kim; Jinkyu Jeong; Joonwon Lee; Seungryoul Maeng; **"Optimizing the startup time of embedded systems: a case study of digital TV"** , Consumer Electronics, IEEE Transaction on , Publication year 2009 Volume: 55 , Issue : 4 page(s): 2242 -2247

[2] Kyung Ho Chung; Myung Sil Choi; Kwang Seon Ahn; Kyungpook Nat. Univ., Daegu **"A Study on the Packaging for Fast Boot-up Time in the Embedded Linux"** Embedded and Real-Time Computing Systems and Applications, 2007, RTCSA 2007, 13th IEEE international conference, Publication Year: 2007, Page(s) 89-94

[3] K. H. Chung, H. Y. Cha, K.S. Ahn, "**A Study on the Effective File System Packaging in the Embedded Linux Systems**", Proceedings of the 2005 International conference on Embedded System and Applications, Las Vegas USA, 2005, pp252-258

[4] Joe, Inwhee; Lee, Sang Cheol; Division of Computer Science & Engineering, Hanyang University, Seoul, South Korea **"Bootup time improvement for embedded Linux using snapshot images created on boot time"** Next Generation Information Technology(ICNIT), 2011 The 2nd international Connference, Publication Year : 2011 Page(s) 193 – 196

[5] Kumar, L.; Kushwaha, R.; Prakash, R; Embedded Syst., Centre for Dev. of Adv. Comput., Noida, India "**Design & Development of Small Linux Operating System for Browser Based Digital Set Top Box"** Computational Intelligence, Communication Systems and Networks, 2009 CICSYN 09, First intenational Conference, Publication year: 2009, page(s): 277 – 281

[6] S. Dey and R. Dasgupta, "**Fast Boot User Experience Using Adaptive Storage Partitioning**", Future Computing, Service Computation, Cognitive, Adaptive, Content, Patterns, 2009. COMPUTATIONWORLD '09. Computation World, pp. 113-118, 2009

[7] Seltzer, M.I, "**Transaction support in a Log-Structured file system, Data Engineering**", Proceedings Ninth International Conference, IEEE, Vienna Austria, 1993, pp503-510

[8] M. Resenblum, J. Ousterhout, "**The design and Implementation of a Log-Structured File System**", ACM Trans On Computer Systems, ACM Press, 1992, Vol.10 No1, pp26-52

[9] R. Bryant, R. Forester, J. Hawkes, "**Filesystem Performance and Scalability in Linux 2.4.17**", 2002 USENIX Annual Technical Conference, USENIX Association, Berkeley USA, 2002, pp259-274

[10] M. K. McKusick, W. N. Joy, S. J. Leffler, R.S. Fabry, "**A Fast File System for UNIX";**, ACM Transactions on Computer Systems, ACM Press, 1984, Vol.2 No.3, pp181-197

[11] D. Roselli, J. R. Lorch, and T. E. Anderson, "**A Comparison of File System Workloads**" Proceedings of USENIX Annual 2000 Technical Conference, San Diego USA, 2000, pp41-54

[12] C. W. Rhodes, "**Interference to DTV Reception due to Non-Linearity of Receiver Front-Ends**", IEEE Transactions on Consumer Electronics, vol. 54, no. 1, 2008

[13] H. Jo , H. Kim , H.-G. Roh and J. Lee, "**Improving the Startup Time of Digital TV**", IEEE Transactions on Consumer Electronics, vol. 52, no. 2, pp. 485-493, 2009

[14] H. Kaminaga, "**Improving Linux Startup Time Using Software Resume**", Proceedings of the Linux Symposium, 2006

[15] C. Park , K. Kim , Y. Jang and K. Hyun, "**Linux Bootup Time Reduction for Digital Still Camera**",Proceedings of the Linux Symposium, 2006

[16] V. Wool, "**Optimizing boot time for Embedded Systems**", Proceedings of Free and Open Source Software Developerâ□™s European Meeting (FOSDEM), 2006

[17] L. Xing , J. Ma , X.-H. Sun and Y. Li, "**Dual-Mode Transmission Networks for DTV**", IEEE Transactions on Consumer Electronics, vol. 54, no. 2, 2008