

Custom Linux Distro Performance Analysis Using LFS

Submitted in partial fulfillment of the requirements for the award of Bachelor of
Engineering degree in Computer Science and Engineering
BY

B.Monish (37110075)

CHADALAVADA KARTHIK (39110117)



**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING SCHOOL
OF COMPUTING**

SATHYABAMA

INSTITUTE OF SCIENCE AND TECHNOLOGY (DEEMED TO BE UNIVERSITY)

Accredited with Grade "A" by NAAC I 12B Status by

UGC I Approved by AICTE

JEPPIAAR NAGAR, RAJIV GANDHI SALAI,

CHENNAI - 600 119

March – 2021



SATHYABAMA

INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)

Accredited with "A" grade by NAAC | 12B Status by UGC | Approved by AICTE

Jeppiaar Nagar, Rajiv Gandhi Salai, Chennai – 600 119

www.sathyabama.ac.in

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

BONAFIDE CERTIFICATE

This is to certify that this Project Report is the bonafide work of **B.Monish(37110075)** and **CHADALAVADA KARTHIK (39110117)** who carried out the project entitled "Custom Linux Distro Performance Analysis Using LFS" under my supervision from November 2020 to March 2021.

Internal Guide

A.YOVAN FELIX ., M.E.,(Ph.D)

Head of the Department

Submitted for Viva Voice examination held on _____

Internal Examiner

External Examiner

DECLARATION

I **BANDAPALLI MONISH(Reg. No. 37110075)** hereby declare that the Project Report entitled “**Custom Linux Distro Performance Analysis Using LFS**” done by me under the guidance of **A.YOVAN FELIX.**, is submitted in partial fulfilment of the requirements for the award of Bachelor of Engineering degree in Computer Science and Engineering.

DATE :

PLACE :

SIGNATURE OF THE CANDIDATE

ACKNOWLEDGEMENT

I am pleased to acknowledge my sincere thanks to **Board of Management** of **SATHYABAMA** for their kind encouragement in doing this project and for completing it successfully. I am grateful to them.

I convey my thanks to **Dr. T.Sasikala M.E., Ph.D .**, Dean, School of Computing and **Dr.L.Lakshmanan M.E., Ph.D.**, and **Dr.S.Vigneshwari M.E., Ph.D.**, Heads of the Department of Computer Science and Engineering for providing me necessary support and details at the right time during the progressive reviews.

I would like to express my sincere and deep sense of gratitude to my Project Guide **Mr.Yovan Felix M.E.,Ph.D** for his valuable guidance, suggestions and constant encouragement paved way for the successful completion of my project work.

I wish to express my thanks to all Teaching and Non-teaching staff members of the **Department of Computer Science and Engineering** who were helpful in many ways for the completion of the project.

ABSTRACT

Linux, most of us have heard of this term since we are learning the computer technology. What is Linux? Linux is an open-source UNIX-like operating system developed by Linus Torvalds in the early 90's. Most of us know about Linux, but what about the terms Fedora, Red-hat, Ubuntu, Centos etc., All these are the distributions of Linux. Before going in detail, let us know what is the term distribution mean? A distribution is as same as Linux operating system but with not all the utilities that Linux provide, this type of distributions used generally to reduce the load and increase the performance by using only necessary utilities for our purpose. Fedora, Red-hat, Ubuntu etc., are well known distributions with their own specifications. If all of them can create their own distributions meeting their privileges, why can't we do the same by creating our own distribution with only the necessary packages installed. We can achieve this by many means like Yocto, LFS etc., So the main thing is we have to decide which will be better for us to proceed.

In my project I am using Yocto to create my own custom distro with minimal packages which will meet my needs. Not only creating our own distro is the target of us, but also analysing its performance to get a clear understanding about how its working and how its utilizing CPU, Memory etc., For Performance Analysis I am going to use a Daemon called Collectd. Now we came to a question What is Daemon? Daemon is a program that will be running continuously in the background collecting the information of the performance of our distro. Even Collectd can also be used to analyse the performance of any distribution.

TABLE OF CONTENTS

CHAPTER NO.	TITLE	PAGE NO.
	ABSTRACT	
1.	INTRODUCTION	
	1.1. GENERAL	
	1.1.1COMPONENTS OF LINUX SYSTEM	
	1.1.2 BASIC FEATURES:	
	1.1.3 LINUX DISTRIBUTIONS	
	1.1.4 ARCHITECTURE OF LINUX KERNEL	
	1.1.5 BOOTLOADER	
	1.1.6 ROOT FILE SYSTEM	
	1.1.7 BUILD SYSTEM	
	1.2 OBJECTIVE	
	1.2.1 REQUIREMENTS	

1.2.2 SOFTWARE AND HARDWARE USED

1.3 PROPOSED METHOD

1.4 SCOPE OF THE PROJECT

1.5 ADVANTAGES OF PROPOSED SYSTEM

2 LITERATURE SURVEY

3 METHODOLOGIES

3.1 GENERAL

3.1.1 LINUX FROM SCRATCH

3.1.2 YOCTO

3.2 COLLECTD

3.3 RRD TOOL

3.3.1 WORKING OF RRD TOOL

3.4 GRAPH GENERATION

4 RESULT

5 CONCLUSION

5.1 REFERENCES

CHAPTER 1

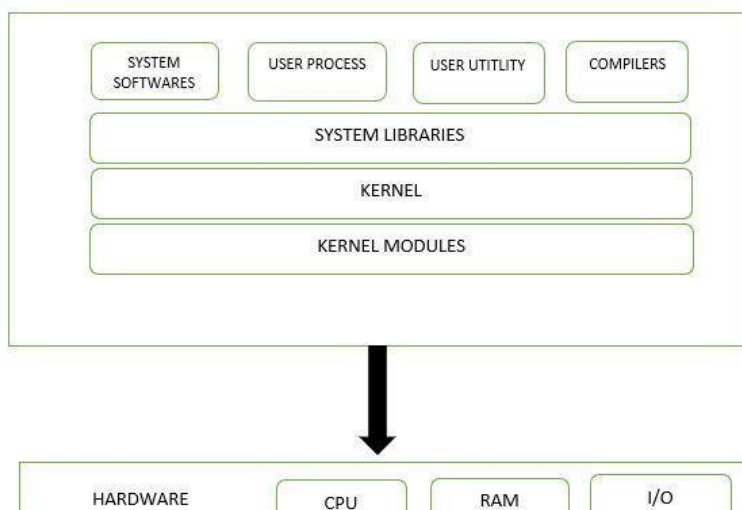
INTRODUCTION

1.1 GENERAL

Linux is an open source operating system (OS). An operating system is the software that directly manages a system's hardware and resources like CPU, memory, and storage. The OS sits between applications and hardware and makes the connections between all of your software and the physical resources that do the work.

1.1.1 COMPONENTS OF LINUX SYSTEM

- **Kernel** – Kernel is the core part of Linux. It is responsible for all major activities of this operating system. It consists of various modules and it interacts directly with the underlying hardware. Kernel provides the required abstraction to hide low level hardware details to system or application programs.
- **System Library** – System libraries are special functions or programs using which application programs or system utilities accesses Kernel's features. These libraries implement most of the functionalities of the operating system and do not requires kernel module's code access rights.
- **System Utility** – System Utility programs are responsible to do specialized, individual level tasks.



1.1.2 BASIC FEATURES:

- **Portable** – Portability means software can work on different types of hardware in the same way. Linux kernel and application programs support their installation on any kind of hardware platform.
- **Open Source** – Linux source code is freely available and it is a community based development project. Multiple teams work in collaboration to enhance the capability of Linux operating system and it is continuously evolving.
- **Multi-User** – Linux is a multiuser system means multiple users can access system resources like memory/ ram/ application programs at the same time.
- **Multiprogramming** – Linux is a multiprogramming system means multiple applications can run at the same time.
- **Hierarchical File System** – Linux provides a standard file structure in which system files/ user files are arranged.
- **Shell** – Linux provides a special interpreter program which can be used to execute commands of the operating system. It can be

used to do various types of operations, call application programs. etc.

- **Security** – Linux provides user security using authentication features like password protection/ controlled access to specific files/ encryption of data.

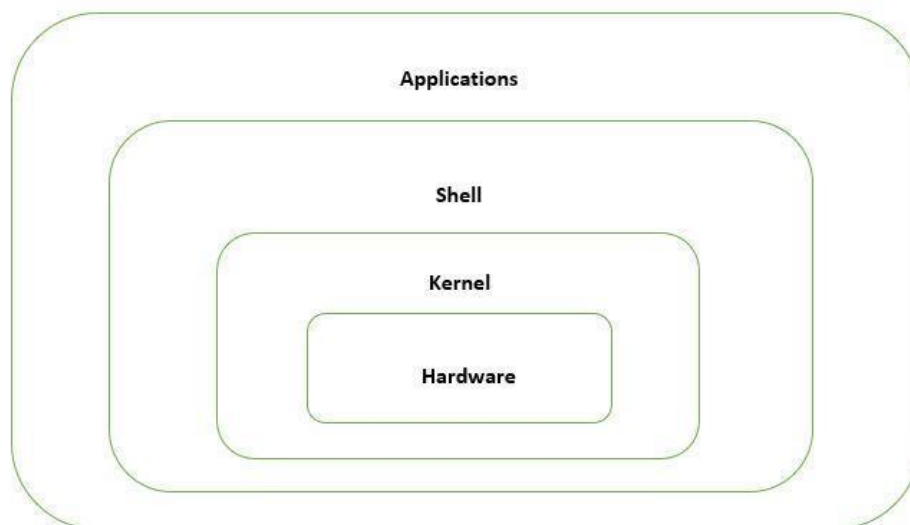
1.1.3 LINUX DISTRIBUTIONS

There are three major distribution families: Debian Family Systems (such as Ubuntu) SUSE Family Systems (such as openSUSE) Fedora Family Systems (such as CentOS)

1.1.4 ARCHITECTURE OF LINUX KERNEL

It consists of three main components:-

- **System call interface**- this is the topmost and undertakes the basic actions such as read and write.
- **The Kernel code [architecture independent kernel code]**- it is common to all of the processor architectures supported by Linux.
- **The architecture-dependent code**- it is referred to as a Board Support Package(BSP). BSP contains a small program called bootloader that places the operating system and device drivers.



User Space

This is where the user applications are executed. There is also the GNU C Library (glibc). This provides the system call interface that connects to the kernel and provides the mechanism to transition between the user-space application and the kernel.

Kernel Space

Here, the Linux Kernel exists which can be further divided into three levels. At the top is the system call interface, which implements the basic functions such as read and write. Below the system call interface is the kernel code, which can be more accurately defined as the architecture-independent kernel code. This code is common to all of the processor architectures supported by Linux. Below this is the architecture-dependent code, which forms what is more commonly called a BSP (Board Support Package). This code serves as the processor and platform-specific code for the given architecture.

CORE SUBSYSTEMS OF THE LINUX KERNEL

- **Process scheduler**

This kernel subsystem is responsible for fairly distributing the CPU time among all the processes running on the system simultaneously.

- ***Memory management unit***

This kernel sub-unit is responsible for proper distribution of the memory resources among the various processes running on the system. The MMU does more than just simply provide separate virtual address spaces for each of the processes.

- **Virtual file system**

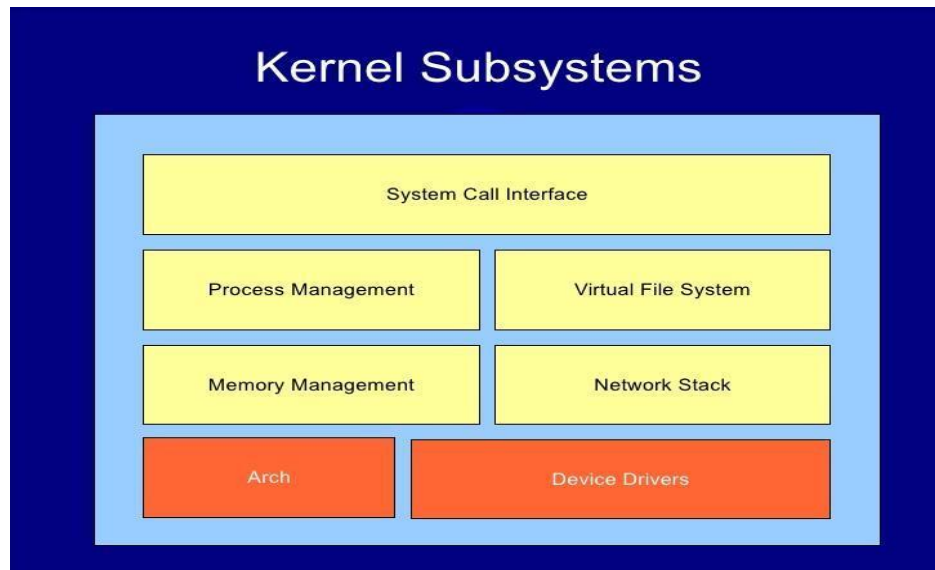
This subsystem is responsible for providing a unified interface to access stored data across different filesystems and physical storage media.

- **Networking unit**

This allows Linux systems to connect to other systems over a network. Networking is a connection between two hosts. On each end, an application gets a socket, makes the transport layer connection, and then sends or receives packets. In Linux, a socket is actually composed of two socket structures.

- **Inter process communication**

Inter-process communication is used for exchanging data between multiple threads in one or more processes or programs.



LINUX KERNEL INTERFACES

There are two API's:-

- Kernel user space- It is Linux API. It give access to the programs in the user space into the system resources and services of kernel.

1.1.5 BOOTLOADER

Bootloader is a program responsible for loading the kernel with optional kernel parameters and the Linux initial RAM disk known as initrd. Bootloaders usually contain several ways to boot the OS kernel and also contain commands for debugging and modifying the kernel environment. Kernel is the core of the Linux OS. It starts the init process such as system immediately after being loaded. RAM stores temporary storage space for loading critical files into the memory before real root system can be mounted.

Older computers with BIOS, bootloader resides in Master Boot Record(MBR) where the memory is of 512 bytes. Newer computers with UEFI(Unified Extensible Firmware Interface) store it in a special partition called EFI System Partition(EFS).

Bootloader is loaded by BIOS or UEFI after successful POST(Power On Self Test)

LINUX BOOT AND START UP PROCESS

Linux boot and start up process comprised of following steps:-

- BIOS POST
- Boot loader (Example:- GRUB 2)
- Kernel initialisation
- Start system

BIOS POST

POST is the self-test process performed when computer is switched ON. It is the first step of the Linux boot process. It is the hardware portion of the boot process and same for any OS. When power is applied to the computer, it runs the POST and it is the part of BIOS.

POST will ensure the working of hardware, if it fails computer may not work and boot process will be stopped. If the POST is success then it issues a BIOS interrupt and locates the boot sectors on any attached bootable devices.

BOOT sector is the first stage of the boot loader which finds that contains a valid boot record is loaded into RAM. Thus the control is transferred to the code that was loaded from the boot sector.

BOOTLOADER(GRUB 2)

GRUB 2 is the primary boot loader for most of the current Linux distributions. It makes the computer to find OS kernel and load it into memory. It has been designed to compatible with the multiboot specification and allows to boot many versions of Linux and other free OS.

It affords the ability to boot to a previous kernel version if an updated one fails or incompatible with an important piece of software.

KERNEL INTIALISATION

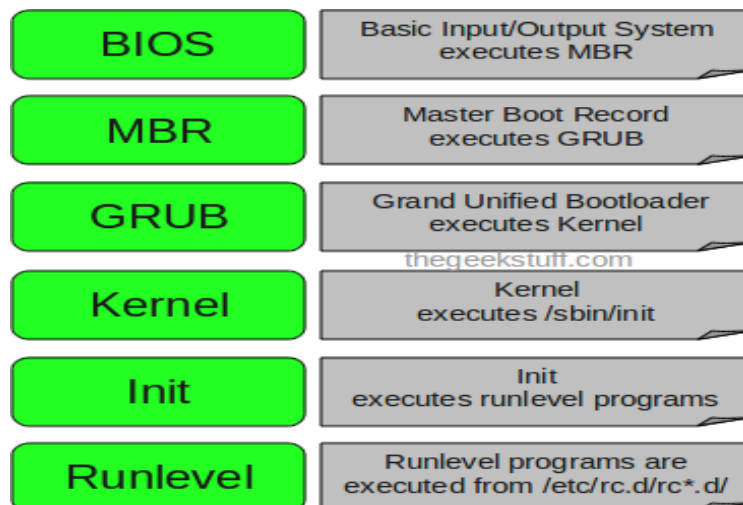
All the kernels are in a self-extracting, compressed format to save space. The kernels are located in a /boot directory along with an initial RAM disk image and device maps of the hard drives.

After it is loaded, to perform any useful task it should first extract itself from compressed version.

Then it will be extracted and loads system and turn control over it. This is the end of boot process.

STARTUP PROCESS

Systemd is the mother of all process.



TYPES OF BOOTLOADER

GRUB (Grand Unified Bootloader)

It is the most popular one, most feature-packed bootloader for Linux operating system.

Modern GRUB is more powerful and cleaner.

LIFO(Linux Loader):-

It was once popular but gradually fallen from grace because it lacked support for multi-boot environment.

SYSLINUX

It is collection of several light weight boot loaded at support major file systems.

It is commonly used for booting live distributions of Linux OS and can boot from floppy disks and USBs.

1.1.6 ROOT FILE SYSTEM

The root file system is the most basic component of Linux. A root file system contains everything needed to support a full Linux system. It contains all the applications, configurations, devices, data, and more. Without the root file system, Linux system cannot run.

It must contain all of the *files* required to boot the *Linux system* before other *filesystems* are mounted. It contains the static bootloader and kernel executable and configuration *files* required to boot a *Linux* computer.

PURPOSE OF ROOT SYSTEM

The contents of the root filesystem must be suitable to boot, restore, recover, and to repair the system.

- To boot a system, enough software and data must be present on the root partition to mount other filesystems. This includes utilities, configuration, boot loader information,

and other essential start-up data. /usr, /opt, and /var are designed such that they may be located on other partitions or filesystems.

- To enable recovery and/or repair of a system, those utilities needed by an experienced maintainer to diagnose and reconstruct a damaged system must be present on the root filesystem.
- To restore a system, those utilities needed to restore from system backups (on floppy, tape, etc.) must be present on the root filesystem.

REQUIREMENTS

The following directories, or symbolic links to directories, are required in root file system;

Directory Description

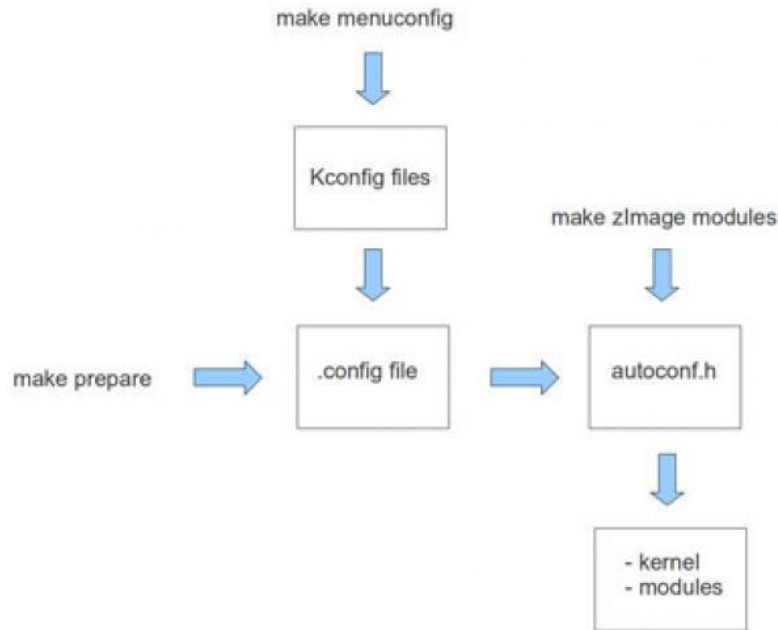
bin	Essential command binaries
boot	Static files of the boot loader
dev	Device files
etc	Host-specific system configuration
lib	Essential shared libraries and kernel modules
media	Mount point for removable media
mnt	Mount point for mounting a filesystem temporarily
opt	Add-on application software packages
run	Data relevant to running processes
sbin	Essential system binaries
srv	Data for services provided by this system
tmp	Temporary files
usr	Secondary hierarchy
var	Variable data

1.1.7 BUILD SYSTEM

The infrastructure to manage the building of the kernel image and its modules is known as the Kernel Build System (kbuild).

The Linux Kernel Build System has four main components:

- Config symbols: The compilation options that can be used to compile code conditionally in source files and to decide which objects to include in a kernel image or its modules.
- Kconfig files: It defines each config symbol and its attributes, such as its type, description and dependencies. Programs that generate an option menu tree (for example, `make menuconfig`) read the menu entries from these files.
- `.config` file: It stores each config symbol's selected value. We can edit this file manually or use one of the many make configuration targets, such as `menuconfig` and `xconfig`, that call specialized programs to build a tree-like menu and automatically update (and create) the `.config` file for you.
- Makefiles: Normal GNU makefiles that describe the relationship between source files and the commands needed to generate each make target, such as kernel images and modules.



1.2 OBJECTIVE

The main objective of my project is to demonstrate the performance analysis of beaglebone black and raspberry pi in Linux distro using Linux from scratch(LFS). Linux From Scratch is a way to install a working Linux system by building all components of it manually. This is naturally a longer process than installing a pre-compiled Linux distribution. The performance analysis will be displayed in graph.

1.2.1 REQUIREMENTS

- Performance Measurement Linux System (PMLS) shall be able to easily measure performance of critical parameters on different hardware on Linux.
- PMLS shall be made portable as easy as possible across different hardware
- PMLS shall provide easy way to configure the needed parameters to measure performance.
- PMLS shall be able to automatically start measuring the configured parameters.

- PMLS shall provide an UI (like web server on localhost) to publish the measurements.

1.2.2 SOFTWARE AND HARDWARE CONFIGURATION:

- Hardware used: Beaglebone black, Raspberry pi
- Operating system used: Fedora, ubuntu
- Software tool used(for generating graphs): collectd, rrdtool

1.3 PROPOSED METHOD

In this research work, we proposed a system to analysis the performance of any hardware once after the boot process of the operating system is completed. This helps to know about the behaviour of any hardware. For demonstrating purpose we used beaglebone black and raspberry pi. The performance analysis will be recorded by collectd and the outcome of it can be displayed through graph. The graph will be displayed using RRD tool.

1.4 SCOPE OF THE PROJECT

Developing our own Linux distro is having lots of advantages as we can set only the needed library files. The complete Linux distro will be the one we wanted. The need to develop our own Linux distro as in already available Linux distro, we can't eliminate the unnecessary libraries which occupy more space and if we did any changes we may even get some errors. By developing our own Linux distro and analysing the performance of hardware helps us to know about its behaviour. To make it easier, we are demonstrating the final result in graph with the help of rrd tool.

BLOCK DIAGRAM:



1.5 ADVANTAGES OF PROPOSED SYSTEM

- We can able to eliminate the unnecessary modules and components from it.
- Removing excess modules and components will reduce the size of the compiled **kernel** and helps it to boot quickly.

CHAPTER 2

LITERATURE SURVEY

Abhimanyu Sharma, Bhuwan Mukhia, Neha Thapa, Ravi Kr Prasad, Arvind Lal(2016)

Open-source software (OSS) is computer software with its source code made available with a license under End User License. Agreement (EULA) in which the copyright holder provides the rights to study, changes, and distributes the software to anyone and for any purpose. Open-source software may be developed in a collaborative public manner. Open-source software is the most prominent example of open-source development. The open-source model, or collaborative competition development from multiple independent sources, generates an increasingly more diverse scope of design perspective than any one company is capable of developing and sustaining long term. The Linux kernel is an operating system kernel used by the Linux family of Unix-like operating systems. It is one of the most prominent examples of free and open source software. Linux is, in simplest terms, an operating system. It is the software on a computer that enables applications and the computer operator to access the devices on the computer to perform desired functions. The operating system (OS) relays instructions from an application to, for instance, the computer's processor.

Shahid H. Bokhari (1995) Linux is an operating system that runs on IBM PC-compatible machines with Intel-386/486/Pentium or equivalent processors. It was written by Linus Torvalds, a graduate student at the Helsinki University of Technology, Finland. Since he wrote it from scratch, it does not contain proprietary code. He holds the copyright but permits free distribution of the source code (see “The Linux software license” sidebar). In addition to Torvalds, an enthusiastic, worldwide group of volunteers collaborated in developing this operating system through the Internet. Linux has been under development for only a few years. As is to be expected, the initial versions were unstable and of interest only to diehard hackers. However, Linux has now matured and can be used for serious work. Many of the powerful compilers and utilities developed by the Free Software Foundation under its GNU (GNU is Not Unix) project

can run on Linux. The X Window System from MIT, the de facto standard for windowing under Unix, is also available for Linux in an Intel-386/486 version called XFree86. Linux is thus a full-fledged operating system that provides all the capabilities normally associated with commercial Unix systems. It has excellent compilers for C, C++ , Pascal, Modula-2 and -3, Oberon, Smalltalk, and Fortran. Editors such as Emacs and vi are available in several versions. Password security, file protection, multiple logins, virtual memory, and multitasking are fully supported. Networking for Linux has developed to the stage where large networks can support remote logins, remote shells, and electronic mail. A version of the Network File System (NFS) has been developed for Linux. This permits file-systems sharing among several machines, thus reducing hard disk use and minimizing system administration overhead. The TeX/LaTeX text-processing system has been ported to Linux along with previewing facilities (ghostview and xdvi) and drawing packages (xfig and idraw). All these capabilities are available free without licensing fees.

Marko Boras, Josip Balen, Kresimir Vdovjak, (2020), Performance often plays a very important role in the field of software evaluation, especially when it is necessary to guarantee the quality, reliability and performance of an operating system [4]. However, it is a complex and time-consuming task because it requires controlled environment and same conditions must be applied for all tested systems, so it remains a bit challenging task, especially in a field of operating systems. In our previous work, several different aspects of an operating systems performance were studied on personal computers. In [5] a comprehensive performance evaluation based on CPU scheduling, disk drive management, graphic subsystem management, memory management and networking performance of the three different versions of Windows operating system, namely Windows XP, Windows Vista and Windows 7, was performed. Also, measurements were conducted in two different environments. Performance measurement results showed better performance of Windows XP in the majority of tests on the low-end computer system when compared to Windows Vista and Windows 7. Furthermore, on the high-end computer system, newer operating systems showed improved performance in the area of graphics subsystem and memory management. Also, a performance evaluation model for Windows operating systems was developed and the similar will be used in this paper with some modifications to

accommodate Linux based operating systems. In [6], three popular virtual private server hosts based on Linux Debian operating system were evaluated. In [7], authors evaluated networking performance of three version of Windows operating systems, namely Windows XP, Windows Vista and Windows. Research in the field of networking performance was continued and expanded in [8] where three latest Windows operating systems, namely Windows 7, Windows 8.1 and Windows 10, were evaluated. Experimental results showed that Windows 10 introduced some improvements in network management, delays were lowered, however, CPU usage was increased.

Ricardo P. Jasinski, Maiko R. Moroz, Volnei A. Pedroni (2012), The use of an operating system (OS) is advocated as a means to simplify software development, freeing programmers from managing low-level hardware and providing a simpler programming interface for common tasks. The high complexity of modern desktop computers makes an OS indispensable; embedded systems, on the other hand, are limited architectures, usually severely cost- and power-constrained. Because of the additional demands imposed by an OS, embedded developers are faced with the crucial decision of whether to adopt an OS or not. In this paper, we present a case study in which a sample application (an embedded weather station) was developed under three different scenarios: without any OS, using the μ C/OS-II real-time OS, and using the uCLinux general-purpose OS. An FPGA and an SoPC were used to provide a flexible hardware platform able to accommodate all three configurations. The adoption of an OS provided a reduction of up to 48% in development time; on the other hand, it increased program memory requirements in at least 71%.

Hari Narayanan,¹ Vivek Radhakrishnan,² Shiju-Sathyadevan³ and Jayaraj Poroor⁴ (2017) Operating system security is a hot research area for the past several decades. Various security mechanisms have been introduced till now to secure the operating system. In this paper we are focusing on securing Linux operating system. Even though Linux is open source and large numbers of people are involved in developing kernel patches for security holes, there are still many malwares to exploit the existing vulnerabilities. Using our architecture we are trying to minimize the damage done by the malwares if not blocking them altogether. Our architecture is designed to ensure the principle of least privilege. Principle of least privilege guarantees that a

process will get the privileges just enough to perform its task. This ensures that even if the process is compromised it can do the least damage to the system as it is running in a sandbox. Major chunk of our system is running in the user level to make it portable across the distributions. Our system uses a specially structured security ticket to provide fine grained authorization to user processes which is not currently possible in the traditional Linux architecture. The security ticket is designed in such a way that it can be inherited by a child process, can be shared and is unforgeable. The core module in the system is called Secd (Secure Daemon) which authorizes all the requests and also manages the security tickets.

Zhongjiang Zhang, Shiqiao Qin, Xingshu Wang and Dejun Zhan (2015), Currently, Linux operating system has become a research focus in the embedded system field, due to its unique advantages such as open-source and security. In this case, this paper provides an idea to transplant the navigation program from VxWorks system to Linux operating system. Therefore, in order to achieve the objective mentioned above, this paper focuses on the analysis of startup process of Linux and kernel based on x86 platform. Then, the key techniques will be resolved to establish a complete development environment and construct an embedded Linux operating system to meet application requirements. In this paper, the method taking advantages of Clonezilla is recommended to backup and restore the embedded Linux.

CHAPTER 3

METHODOLOGIES

3.1 GENERAL

First thing we to do develop our own custom distribution is to do a tiny research. This research should involve the process that we are going to follow to develop the distribution, this helps us to follow a particular pattern of development, then the research should also include what are all the characteristics and specifications of the distribution. By choosing this, we can include it in the

development process and finally the output will be our custom distribution which will meet our needs.

In my research I have come across two development process:

1. LFS
2. Yocto

I found that LFS was introduced in the very early days, so it will maximum support the older versions. So I have chosen yocto for the development of my distribution. The Yocto Project is a Linux Foundation collaborative open source project which is useful for developing Linux distributions. The important thing regarding yocto developers is they will be updating yocto in the newer versions regularly. The steps are mentioned in detail in their official website. Next important step in every distribution after developing it to check its performance, there comes the use of collectd daemon.

3.1.1 LINUX FROM SCRATCH

Linux From Scratch is a way to install a working Linux system by building all components of it manually. This is naturally a longer process than installing a pre-compiled Linux distribution. According to the Linux From Scratch site, the advantages to this method are a compact, flexible and secure system and a greater understanding of the internal workings of the Linux-based operating systems. The Linux from Scratch project, like [BitBake](#), also supports cross-compiling Linux for ARM embedded systems such as the Raspberry Pi and [BeagleBone](#).

REQUIREMENTS AND PROCEDURE

A clean partition and a working Linux system with a compiler and some essential software libraries are required to build LFS. Instead of installing from an existing Linux system, one can also use a Live CD to build an LFS system. First, a toolchain must be compiled consisting of the tools used to compile LFS, like GCC, glibc, binutils and other necessary utilities. Then, the root directory must be changed, (using chroot), to the toolchain's partition to start building the final

system. One of the first packages to compile is glibc; after that, the toolchain's linker must be adjusted to link against the newly built glibc, so that all other packages that will make up the finished system can be linked against it as well. During the chroot phase, bash's hashing feature is turned off and the temporary toolchain's bin directory moved to the end of PATH. This way the newly compiled programs come first in PATH and the new system builds on its own new components.

STANDARD BUILT UNIT

A "standard build unit" ("SBU") is a term used during initial bootstrapping of the system, and represents the amount of time required to build the first package in LFS on a given computer. Its creation was prompted by the long time required to build an LFS system, and the desire of many users to know how long a source tarball will take to build ahead of time.

As of Linux From Scratch version 6.3, the first package built by the user is GNU binutils. When building it, users are encouraged to measure the build process using shell constructs and dub that time the system's "standard build unit". Once this number is known, an estimate of the time required to build later packages is expressed relative to the known SBU.

Several packages built during compilation take much longer to build than binutils, including the GNU C Library (rated at 4.1 SBUs) and the GNU Compiler Collection (rated at 8.4 SBUs). The unit must be interpreted as an approximation; various factors influence the actual time required to build a package.

KERNEL/DISTRO VERSION

We use Fedora Core 33 in this project. We can run any distribution which uses a 5.x kernel, but be aware that doing so may introduce inconsistencies into the build process. Some distributions may package utilities using older or incompatible versions than what is needed in this course.

CLEAN INSTALL

We should do clean install for the build system. Any virtualization environment will do, provided we have access to the console, as it may be necessary at various points. Actual hardware is also acceptable, if those resources are readily available to us.

DEVELOPMENT TOOLS

We'll need GCC, binutils, and other software packages installed with the "development tools" package group. You can select this during the installation process, or you can install using the group install option for yum or dnf. Note that the package group names may differ depending on distribution, but generally, we will need the "development tools" and "C development tools" groups installed.

3.1.2 YOCTO

The Yocto Project through the OpenEmbedded build system provides an open source development environment targeting the ARM, MIPS, PowerPC, and x86 architectures for a variety of platforms including x86-64 and emulated ones. You can use components from the Yocto Project to design, develop, build, debug, simulate, and test the complete software stack using Linux, the X Window System, GTK+ frameworks, and Qt frameworks.

- Provides a recent Linux kernel along with a set of system commands and libraries suitable for the embedded environment.
- Makes available system components such as X11, GTK+, Qt, Clutter, and SDL (among others) so you can create a rich user experience on devices that have display hardware. For devices that do not have a display or where you wish to use alternative UI frameworks, these components need not be installed.
- Creates a focused and stable core compatible with the OpenEmbedded project with which you can easily and reliably build and develop.
- Fully supports a wide range of hardware and device emulation through the Quick EMUlator (QEMU).

- Provides a layer mechanism that allows you to easily extend the system, make customizations, and keep them organized.

We can use the Yocto Project to generate images for many kinds of devices. As the Yocto Project supports creation of reference images, we can boot within and emulate using QEMU. The standard example machines target QEMU full-system emulation for 32-bit and 64-bit variants of x86, ARM, MIPS, and PowerPC architectures. Beyond emulation, we can use the layer mechanism to extend support to just about any platform that Linux can run on and that a toolchain can target.

PREREQUISITES FOR THE DEVELOPMENT MACHINE:

- At least 4 - 6 GB RAM.
- Recent Ubuntu OS (20.04 LTS in this case).
- At least 100 GB free space on the disk.
- Installation of following packages before creation of new Linux distro.
- Download latest Yocto (Poky which is minimal development environment) stable branch.

BUILDING IMAGES

- Build a qemu86 reference image and run it in the QEMU emulator.
- Easily change configurations so that you can quickly create a second image, which would be for MinnowBoard MAX-compatible boards.
- **Be Sure Your Build Host is Set Up:** The steps to build an image in this section depend on your build host being properly set up.
- **Check Out our Branch:** Be sure you are in the Source Directory (e.g. poky) and then check out the branch associated with the latest Yocto Project Release:

```
$ cd ~/poky
```

```
$ git checkout -b jethro origin/jethro
```

Git's checkout command checks out the current Yocto Project release into a local branch whose name matches the release (i.e. jethro). The local branch tracks the upstream branch of the same name. Creating your own branch based on the released branch ensures you are using the latest files for that release.

Initialize the Build Environment: Run the oe-init-build-env environment setup script to define the OpenEmbedded build environment on your build host.

```
$ source oe-init-build-env
```

Among other things, the script creates the Build Directory, which is build in this case and is located in the Source Directory. After the script runs, your current working directory is set to the Build Directory. Later, when the build completes, the Build Directory contains all the files created during the build.

Examine Your Local Configuration File: When you set up the build environment, a local configuration file named local.conf becomes available in a conf subdirectory of the Build Directory. Before using BitBake to start the build, you can look at this file and be sure your general configurations are how you want them:

- To help conserve disk space during builds, you can add the following statement to your project's configuration file, which for this example is poky/build/conf/local.conf. Adding this statement deletes the work directory used for building a recipe once the recipe is built.

```
INHERIT += "rm_work"
```

- By default, the target machine for the build is qemu86, which produces an image that can be used in the QEMU emulator and is targeted at an Intel® 32-bit based architecture. Further on in this example, this default is easily changed through

the MACHINE variable so that you can quickly build an image for a different machine.

- Another consideration before you build is the package manager used when creating the image. The default local.conf file selects the RPM package manager. You can control this configuration by using the PACKAGE_CLASSES variable.

Selection of the package manager is separate from whether package management is used at runtime in the target image.

For additional package manager selection information, see the "package*.bbclass" section in the Yocto Project Reference Manual.

Start the Build: Continue with the following command to build an OS image for the target, which is core-image-sato in this example:

```
$ bitbake core-image-sato
```

Simulate our Image Using QEMU: Once this particular image is built, you can start QEMU and run the image:

```
$ runqemu qemux86
```

Exit QEMU: Exit QEMU by either clicking on the shutdown icon or by opening a terminal, typing poweroff, and then pressing "Enter".

3.2 COLLECTD

collectd is a daemon which collects system and application performance metrics periodically and provides mechanisms to store the values in a variety of ways, for example in RRD files. collectd gathers metrics from various sources, e.g. the operating system, applications, logfiles and external devices, and stores this information or makes it available over the network. Those

statistics can be used to monitor systems, find performance bottlenecks (i.e. performance analysis) and predict future system load (i.e. capacity planning).

LIMITATIONS

- It does not generate graphs. It can write to RRD files, but it cannot generate graphs from these files. There's a tiny sample script included in contrib/, though. Take a look at kcollectd, an X frontend, and draw, a very generic solution, though. More utility programs are listed on the related projects page.
- Monitoring functionality has been added in version 4.3, but is so far limited to simple threshold checking. The document "Notifications and thresholds" describes collectd's monitoring concept and has some details on the limitations, too. Also, there's a plugin for Nagios, so it can use the values collected by collectd.

3.3 RRD TOOL

RRDtool is the Open Source industry standard, high performance data logging and graphing system for time series data. RRDtool can be easily integrated in shell scripts, perl, python, ruby, lua or tcl applications.

It is pretty easy to gather status information from all sorts of things, ranging from the temperature in your office to the number of octets which have passed through the FDDI interface of your router. But it is not so trivial to store this data in an efficient and systematic manner. This is where **RRDtool** comes in handy. It lets you log and analyze the data you gather from all kinds of data-sources (**DS**). The data analysis part of RRDtool is based on the ability to quickly generate graphical representations of the data values collected over a definable time period.

RRDtool provides the building blocks for near real-time aberrant behaviour detection. These components include:

- An algorithm for predicting the value of a time series one time step into the future.
- A measure of deviation between predicted and observed values.

- A mechanism to decide if and when an observed value or sequence of observed values is too deviant from the predicted value(s).

3.3.1 WORKING OF RRD TOOL

DATA ACQUISITION

When monitoring the state of a system, it is convenient to have the data available at a constant time interval. Unfortunately, you may not always be able to fetch data at exactly the time you want to. Therefore RRDtool lets us to update the log file at any time you want. It will automatically interpolate the value of the data-source (DS) at the latest official time-slot (interval) and write this interpolated value to the log. The original value you have supplied is stored as well and is also taken into account when interpolating the next log entry.

CONSOLIDATION

We may log data at a 1 minute interval, but you might also be interested to know the development of the data over the last year. You could do this by simply storing the data in 1 minute intervals for the whole year. While this would take considerable disk space it would also take a lot of time to analyse the data when you wanted to create a graph covering the whole year. RRDtool offers a solution to this problem through its data consolidation feature. When setting up a Round Robin Database (RRD), you can define at which interval this consolidation should occur, and what consolidation function (CF) (average, minimum, maximum, last) should be used to build the consolidated values (see `rrdcreate`). You can define any number of different consolidation setups within one RRD. They will all be maintained on the fly when new data is loaded into the RRD.

ROUND ROBIN ARCHIVES

Data values of the same consolidation setup are stored into Round Robin Archives (RRA). This is a very efficient manner to store data for a certain amount of time, while using a known and constant amount of storage space.

It works like this: If you want to store 1'000 values in 5 minute interval, RRDtool will allocate space for 1'000 data values and a header area. In the header it will store a pointer telling which slots (value) in the storage area was last written to. New values are written to the Round Robin

Archive in, you guessed it, a round robin manner. This automatically limits the history to the last 1'000 values (in our example). Because you can define several RRAs within a single RRD, you can setup another one, for storing 750 data values at a 2 hour interval, for example, and thus keep a log for the last two months at a lower resolution.

The use of RRAs guarantees that the RRD does not grow over time and that old data is automatically eliminated. By using the consolidation feature, you can still keep data for a very long time, while gradually reducing the resolution of the data along the time axis.

Using different consolidation functions (CF) allows you to store exactly the type of information that actually interests you: the maximum one minute traffic on the LAN, the minimum temperature of your wine cellar, ... etc.

UNKNOWN DATA

As mentioned earlier, the RRD stores data at a constant interval. Sometimes it may happen that no new data is available when a value has to be written to the RRD. Data acquisition may not be possible for one reason or other. With RRDtool you can handle these situations by storing an `*UNKNOWN*` value into the database. The value `'*UNKNOWN*'` is supported through all the functions of the tool. When consolidating a data set, the amount of `*UNKNOWN*` data values is accounted for and when a new consolidated value is ready to be written to its Round Robin Archive (RRA), a validity check is performed to make sure that the percentage of unknown values in the data point is above a configurable level. If not, an `*UNKNOWN*` value will be written to the RRA.

3.4 GRAPH GENERATION

RRDtool allows you to generate reports in numerical and graphical form based on the data stored in one or several RRDs. The graphing feature is fully configurable. Size, colour and contents of the graph can be defined freely.

`rrdtool graph` needs data to work with, so you must use one or more [data definition](#) statements to collect this data. We are not limited to one database, it's perfectly legal to collect data from two or more databases.

The data fetched from the **RRA** is then **consolidated** so that there is exactly one data point per pixel in the graph. If you do not take care yourself, **RRDtool** will expand the range slightly if necessary.

Sometimes data is not exactly in the format you would like to display it. For instance, we might be collecting **bytes** per second, but want to display **bits** per second. This is what the **data calculation** command is designed for. After **consolidating** the data, a copy is made and this copy is modified using a rather powerful **RPN** command set.

When you are done fetching and processing the data, it is time to graph it (or print it). This ends the **rrdtool graph** sequence.

CHAPTER 4

RESULT

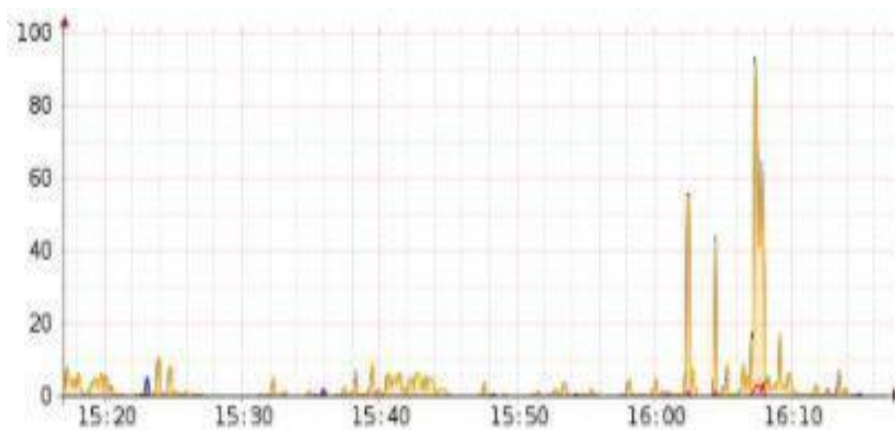
Below mentioned tables and graphs are the performance measurements of the Linux distro that we developed. The details will be in two sections

1. Epoch

2. Value

Epoch means the no. of seconds since the distribution is developed. The value refers to the value of the respected plugin that we enabled.

```
epoch,value
1614574041.673,2783895552.000000
1614574051.651,2782306304.000000
1614574061.657,2782081024.000000
1614574071.652,2782081024.000000
1614574081.652,2782081024.000000
1614574091.651,2782081024.000000
1614574101.651,2782081024.000000
1614574111.652,2781822976.000000
1614574121.652,2781822976.000000
1614574131.652,2781822976.000000
1614574141.652,2781822976.000000
1614574151.652,2781822976.000000
1614574161.651,2781822976.000000
```



CHAPTER 5

CONCLUSION

From my project I can conclude that where ever the distribution can be used, it doesn't concern us in checking the performance of the distribution. So checking the performance of any distribution is the most recommended action to check whether the distribution is meeting our expectations.

5.1 REFERENCES

1. James Sanders (1998) "Linux, Open Source, and Software's Future"
2. Shahid H. Bokhari (1995) "The Linux operating system" University of Engineering and Technology, Lahore, Pakistan.
3. Abhimanyu Sharma, Bhuwan Mukhia, Neha Thapa, Ravi Kr Prasad, Arvind Lal (2016) Chisopani Nandugao Namchi Sikkim, India.
4. Ricardo P. Jasinski, Maiko R. Moroz, Volnei A. Pedroni (2012) "The Impact of Operating System Adoption in an Embedded Project: a Case Study" Laboratory of Microelectronics, UTFPR - Federal University of Technology.
5. Marko Boras, Josip Balen, Kresimir Vdovjak (2020) "Performance evaluation of Linux Operating Systems" Faculty of Electrical Engineering, Computer Science and Information Technology Osijek, Osijek, Croatia.
6. "LFS News". *www.Linuxfromscratch.org*. Retrieved 3 September 2020.
7. *Beekmans, Gerard (2020). Linux From Scratch, Version 10.0(PDF).*
8. Preface:LFS Target Architectures, Linux From Scratch
9. What is Linux From Scratch?, LFS Project Homepage
10. Gerard Beekmans: Beyond Linux From Scratch, Version 6.3 (August 2008)
11. <https://www.yoctoproject.org/docs/2.0/yocto-project-qs/yocto-project-qs.html>

12. <https://oss.oetiker.ch/rrdtool/doc/rrdtool.en.html>

13. <http://www.buildyourownLinux.com/>

14. <https://collectd.org/>