

CUSTOM LINUX DISTRO FOR PERFORMANCE ANALYSIS USING LINUX FROM SCRATCH (LFS)

Submitted in partial fulfilment of the requirements for the award of Bachelor of
Engineering Degree in Electronics and Communication Engineering

by

S. VISHNU PREETHA (37130441)



**DEPARTMENT OF ELECTRONICS AND COMMUNICATION
ENGINEERING SCHOOL OF ELECTRICAL AND
ELECTRONICS ENGINEERING**

**SATHYABAMA INSTITUTE OF SCIENCE AND
TECHNOLOGY (DEEMED TO BE UNIVERSITY)**

Accredited with Grade "A" by NAAC

JEPPIAAR NAGAR, RAJIV GANDHI SALAI, CHENNAI - 600 119

APRIL 2021



**SATHYABAMA INSTITUTE
OF SCIENCE AND
TECHNOLOGY**



(DEEMED TO BE UNIVERSITY)

Accredited with Grade “A” by NAAC

JEPPIAAR NAGAR, RAJIV GANDHI SALAI, CHENNAI - 600 119

www.sathyabama.ac.in

**DEPARTMENT OF ELECTRONICS AND COMMUNICATION
ENGINEERING**

BONAFIDE CERTIFICATE

This is to certify that this Project Report is the bonafide work of **S.VISHNU PREETHA (37130441)** who carried out the project entitled “**CUSTOM LINUX DISTRO FOR PERFORMANCE ANALYSIS USING LINUX FROM SCRATCH (LFS)**” under our supervision from September 2020 to April 2021.

[Internal Guide](#)

Dr. P.CHITRA.,Ph.D.

Department of ECE

Head of the Department

Dr. T. RAVI, M.E., Ph.D.

Submitted for Viva voce examination held on _____

Internal Examiner

External Examiner

DECLARATION

I, **S. VISHNU PREETHA(37130441)** hereby declare that the Project Report entitled **“CUSTOM LINUX DISTRO FOR PERFORMANCE ANALYSIS USING LINUX FROM SCRATCH (LFS)”** done by us under the guidance of **Dr.P.CHITRA., Ph.D**, is submitted in partial fulfilment of the requirements for the award of Bachelor of Engineering degree in Electronics and Communication Engineering.

DATE:

PLACE:

SIGNATURE OF THE CANDIDATE

ACKNOWLEDGEMENT

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

I convey my thanks to **Dr. N. M. NANDHITHA, M.E., Ph.D. Dean, School of Electrical and Electronics Engineering** and **Dr. T. RAVI, M.E., Ph.D. Head of the Department, Department of Electronics and Communication Engineering** for providing me with 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 **Dr.P.CHITRA., Ph.D, Department of Electronics and Communication Engineering** for her valuable guidance, suggestions and constant encouragement paved way for the successful completion of the project work.

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

I express my gratitude to my parents for their constant encouragement and support for the completion of the project.

ABSTRACT

In the early 1990s, Linus Torvalds developed Linux as a UNIX-like open-source framework. Many Linux discs, including Fedora, Red Hat, Ubuntu, Centos and others, are available. A Linux-like distribution but without any of the Linux utilities. This type of delivery is usually used to decrease load and improve output only by providing the necessary utilities. Other distributions are well-known for their specifications such as Fedora, Red Hat, Ubuntu, and others. Why not distribute ourselves using the requested packages only if you can build distributions of your own to fulfil your privileges? This can be done by many Linux development distributions, including Yocto, LFS and others. The main thing is to find out what to do first. In my project, I use Yocto to build the very least bundle of my own personalised distro that meets my requirements. The goal is not just to create our own results, it is also to analyse how it functions and how its CPU, memory and other resources are used. I will use the Collected Daemon for performance analysis. A decision has been reached. Daemon is a programme that collects data and analyses our distro efficiency in a context. The collected data is then used to measure the consistency of any distribution.

TABLE OF CONTENTS

TITLE	PAGE NO.
ABSTRACT	v
LIST OF FIGURES	viii

CHAPTER NO.	TITLE	PAGE NO.
1.	INTRODUCTION	1
	1.1 GENERAL	1
	1.1.1 COMPONENTS OF LINUX SYSTEM	1
	1.1.2 BASIC FEATURES	2
	1.1.3 LINUX DISTRIBUTIONS	2
	1.1.4 ARCHITECTURE OF LINUX KERNEL	3
	1.1.5 BOOTLOADER	6
	1.1.6 ROOTFILE SYSTEM	8
	1.1.7 BUILD SYSTEM	10
	1.2 OBJECTIVE	11
	1.2.1 REQUIREMENTS	
	1.2.2 SOFTWARE AND HARDWARE CONFIGURATION	12
	1.2 PROPOSED METHOD	12
	1.3 SCOPE OF THE PROJECT	12
	1.4 ADVANTAGES OF PROPOSED SYSTEM	13
2.	LITERATURE SURVEY	14
3.	METHODOLOGIES	18
	3.1 GENERAL	18
	3.1.1 LINUX FROM SCRATCH	18
	3.1.2 YOCTO	20
	3.1.3 BUILDING IMAGES	21
	3.2 COLLECTD	23
	3.2.1 LIMITATIONS	23
	3.3 RRD TOOL	24

3.4	WORKING OF RRD TOOL	24
4.	RESULT	27
5.	CONCLUSION	31
6.	REFERENCES	32

LIST OF FIGURES

FIGURE NO.	TITLE	PAGE NO.
1.1	COMPONENTS OF LINUX SYSTEM	10
1.2	ARCHITECTURE OF LINUX KERNEL	12
1.3	KERNEL SUBSYSTEMS	13
1.4	BOOTLOADER PROCESS	16
1.5	ROOTFILE SYSTEM	18
1.6	BUILD SYSTEM	19
1.7	BLOCK DIAGRAM	21

CHAPTER 1

INTRODUCTION

1.1 GENERAL

Linux is an open source operating system (OS). An operating system is the software that manages a system's hardware and resources like CPU, memory, and storage. The OS lies 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** – The main component of Linux is the kernel. It is in charge of all major operating system operations. It consists of a number of modules and interfaces with the base hardware directly. It provides the necessary abstraction to hide device and application programmes with low-level hardware data.
- **System Library** – There are special features or programmes that control kernel features with applications or machine utilities. These system libraries implement much of the operating system functionality and do not include code access rights to the kernel module.
- **System Utility** – These programs are responsible to do specialized individual level tasks.

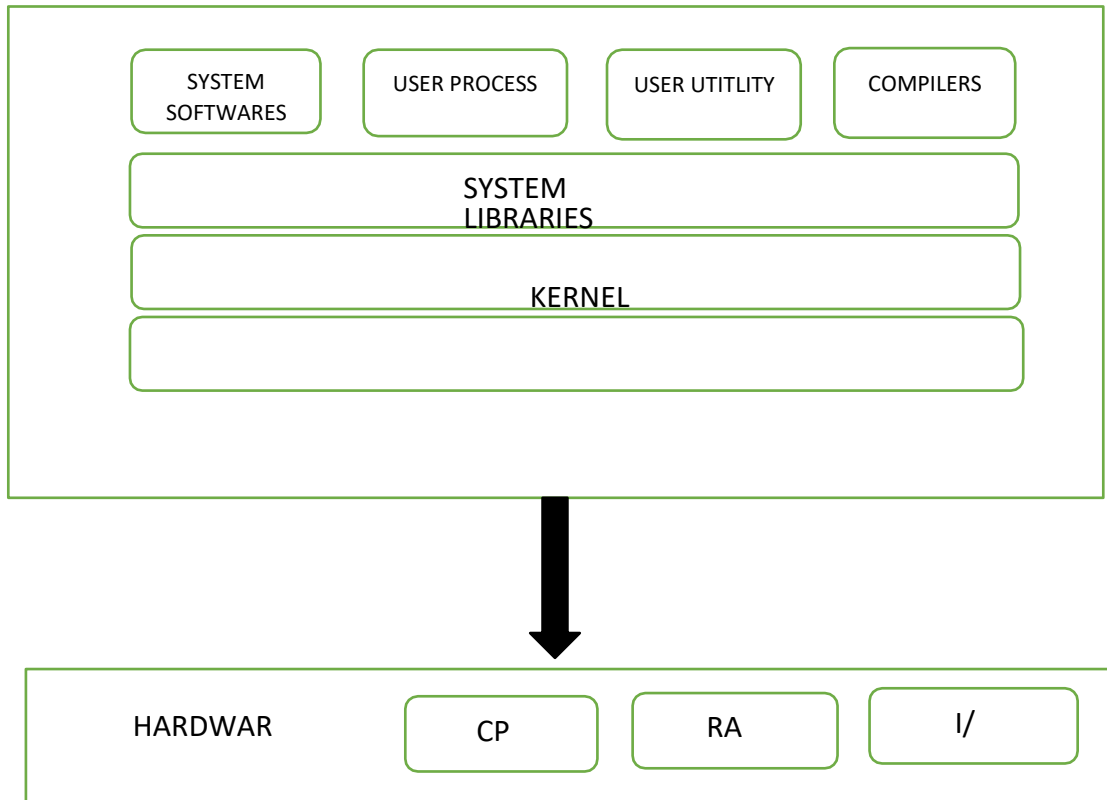


FIG 1.1 COMPONENTS OF LINUX SYSTEM

1.1.2 BASIC FEATURES:

- ❑ **Portable** – Portability means that software can function in the same way with various kinds of hardware. Linux kernel and application programmes can be installed in any various hardware platform.
- ❑ **Open Source** – Linux Source Code is an open source code freely available and a development project focused on a collective. Many teams work together to improve Linux operating system capabilities and are constantly changing.
- ❑ **Multi-User** – The Linux kernel is a multiuser system that means that several users can simultaneously access system resources such as memory/ram/application programmes.
- ❑ **Multiprogramming** – It is a multiprogramming system means multiple applications can run at same time.

- **Hierarchical File System** – It provides a standard file structure in which system files/ user files are arranged.
- **Shell** – Linux contains an interpreter programme for executing operating system commands. It can be used to do all kinds of activities, apps and so on.
- **Security** – Linux provides user security using safety features like password protection/ controlled access to specific files/ encryption of data.

1.1.3 LINUX DISTRIBUTIONS

Triple distributed families exist: Debian family systems like Ubuntu, SUSE family systems like openSUSE and Fedora family systems like CentOS

1.1.4 ARCHITECTURE OF LINUX KERNEL

It consists of three main components:-

- Call system interface – is the highest level and performs simple activities including reading and writing.
- The kernel code, which is common to the linux-supported processor architectures [architecture independent kernel code].
- Code depending on the architecture - it is called a Board support package (BSP).
BSP is a small bootloader programme which places drivers and operating systems.

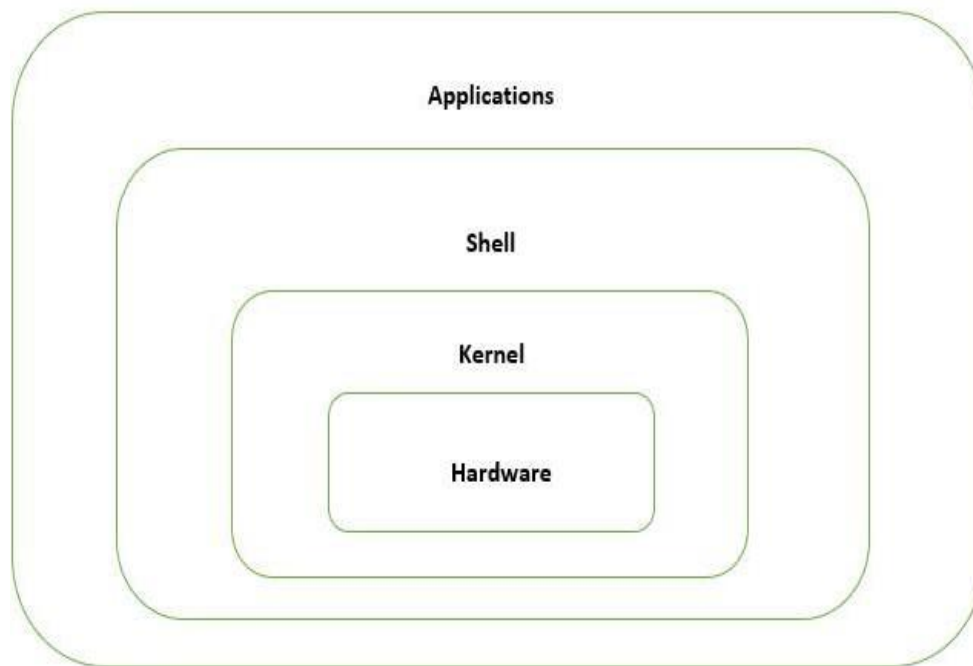


FIG 1.2 ARCHITECTURE OF LINUX KERNEL

User Space

These are the user applications that are currently running. The GNU C Library is still available (glibc). The system call interface connects to the kernel and provides a way for the user space application and the kernel to communicate.

Kernel Space

As a result, the Linux kernel is open, which can be divided into three stages. The computer call interface is a higher-level interface that employs simple reading and writing functions. At the system call interface below, the kernel can be represented more precisely in architecture independent kernel code. Both Linux-supported processor architectures use the same core code. The architecture-dependent code, commonly referred to as BSP, is located below the kernel code (Board Support Package). The BSP code serves as the processor and platform code for the basic architecture.

CORE SUBSYSTEMS OF THE LINUX KERNEL

☐ **Process scheduler**

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

☐ **Memory management unit**

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

☐ **Virtual file system**

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

☐ **Networking unit**

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

☐ **Inter process communication**

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

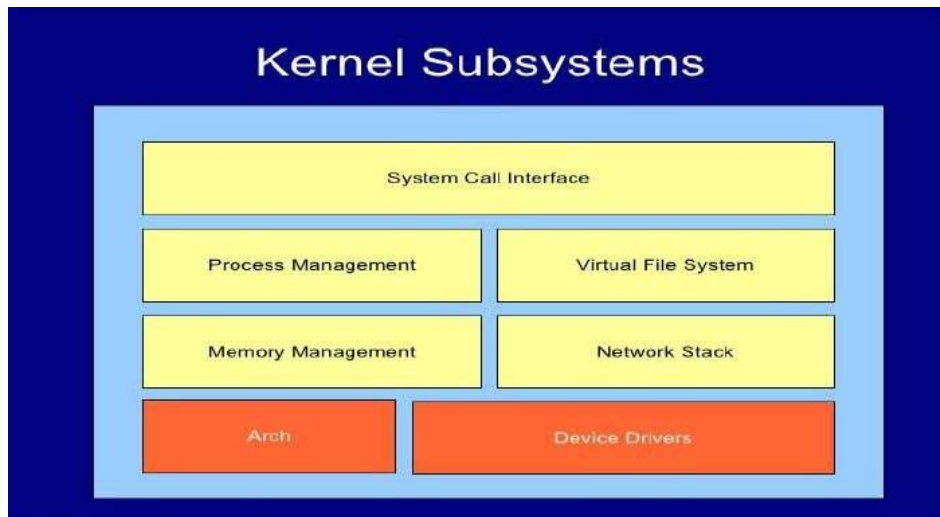


FIG 1.3 KERNEL SUBSYSTEMS

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.
- Kernel internal-

1.1.5 BOOTLOADER

Bootloader is a programme that loads the kernel with the optional kernel parameters and the initial initrd RAM drive. It normally includes several ways to boot the OS kernel and also includes debug and kernel environment commands. The heart of the linux operating system is kernel. It begins immediately after loading the init process such as the framework. RAM stores temporary storage space until a real root device can be installed for loading vital files into the memory.

The oldest BIOS bootloader computers are located in the Master Boot Record(MBR), with a 512 byte memory. New EUFI computers store them in an EFI system partition called Unified Extensible Firmware Interface (EFS).

After the active POST bootloader is loaded with BIOS or UEFI (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-testing operation that occurs when the computer is turned on. This is the first step in the Linux boot sequence. It's the system's booting hardware, and it's the same with all operating systems. When power is applied to the unit, the POST is activated, and the BIOS section is activated.

Many machines will not run if they crash, and the boot process will be terminated. POST guarantees that the hardware is operational. If POST is successful, it will interrupt the BIOS and locate the boot sectors on any attached bootable devices.

The boot loader sector is the first stage of the boot loader, and it searches for a valid boot record to load into RAM. This gives power to the code that was loaded from the boot area.

BOOTLOADER(GRUB 2)

GRUB 2 is the default boot loader for most modern Linux distributions. It allows the system to locate and save OS kernels. It is multiboot compliant, allowing you to boot a variety of Linux and other free operating systems.

It allows you to boot to a previous kernel if you are unable to boot an updated version or if an important software component is not compatible.

KERNEL INITIALISATION

To save space, both Linux kernels are compressed and self-extracted. The kernels are stored in /boot directories alongside the initial RAM disc picture and hard drive device maps.

When it's loaded, it needs to be removed from the compressed version before it can do anything useful.

The computer is then dismantled, loaded, and operated.

The boot process is now complete.

STARTUP PROCESS

Systemd is the mother of all process.



FIG 1.4 BOOTLOADER PROCESS

TYPES OF BOOTLOADER

GRUB (Grand Unified Bootloader)

It is the most common bootloader for Linux' operating system with most of its features. More potent and healthier, modern GRUB.

LIFO(Sound Loader):-

It was once common, but it gradually fell away because the multi-boot environment was not supported.

SYSLINUX

It is collection of several light weight boot loaded th 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 most basic part in Linux is the root file system. Anything necessary to support the complete Linux system is in a root file system. All apps, settings, computers, data and more are contained on the root file system. Linux framework can't run without the root file system.

All the files required for booting the Linux system must be included before mounting other file systems. It includes the Linux computer-style bootloader and executable kernel and configurations files.

PURPOSE OF ROOT SYSTEM

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

- ❑ To boot the 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 any other partitions or filesystems.
- ❑ To enable recovery and 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, the necessary 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

Directory Description

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

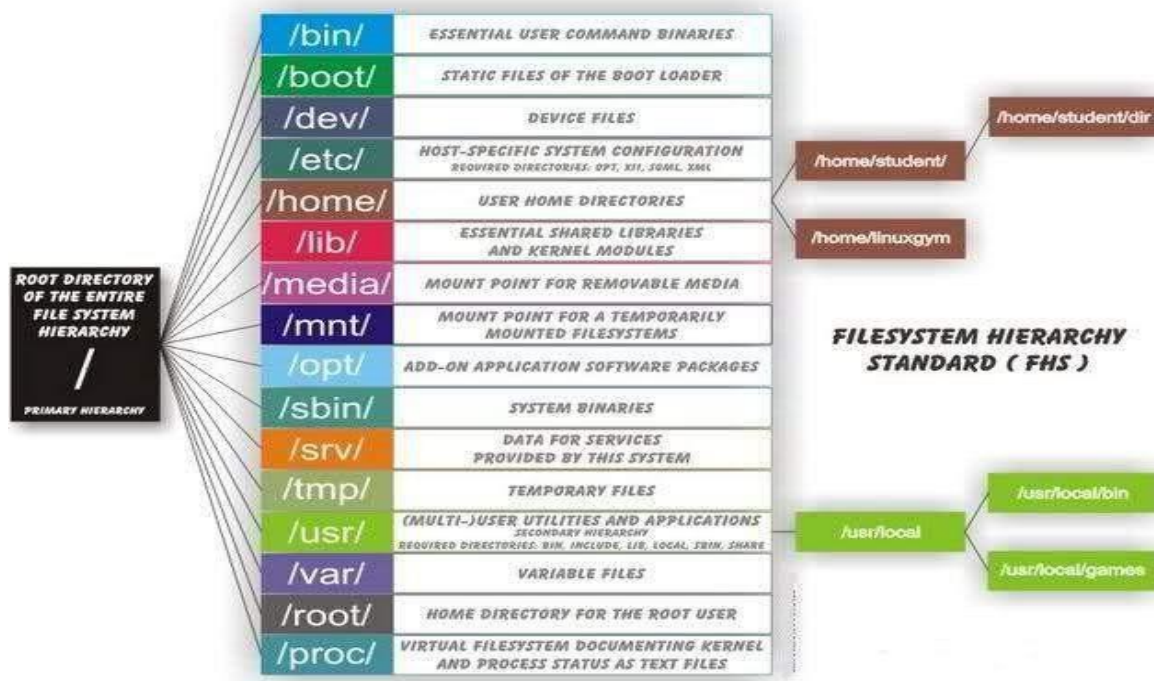


FIG 1.5 ROOTFILE SYSTEM

1.1.7 BUILD SYSTEM

Kernel Build System is the infrastructure for controlling image building and its modules (kbuild).

The four key components of the Linux kernel build system are:

- Symbols configuration: the compilation options for compiling code in source files on a conditional basis and for choosing which objects to include in an image or its modules of the kernel.
- Kconfig: it describe the form, definition and dependencies of any of the config symbols and attributes of them. The choice menu box (for example, make menu config) is created by this programme and read the menu items of those files.
- .cfg file: stores the selected value of each config symbol. We are able to edit it manually or use one of the several set-up goals, such as menuconfig and xconfig, to construct a tree-like menu and to update and create the .config.
- Makefiles: The Normal GNU Makefiles describe the connection between the source files and the commands that each make target, for example kernel images and modules, needs to generate.

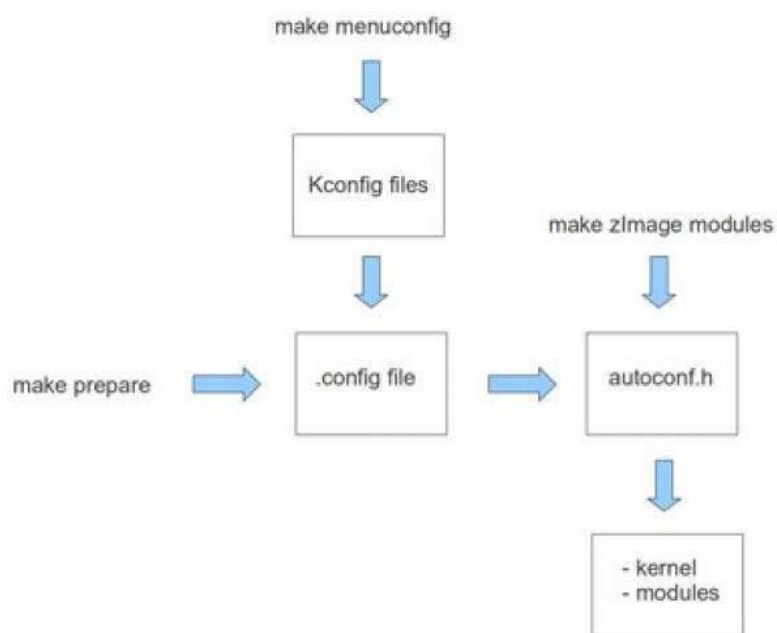


FIG 1.6 BUILD SYSTEM

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

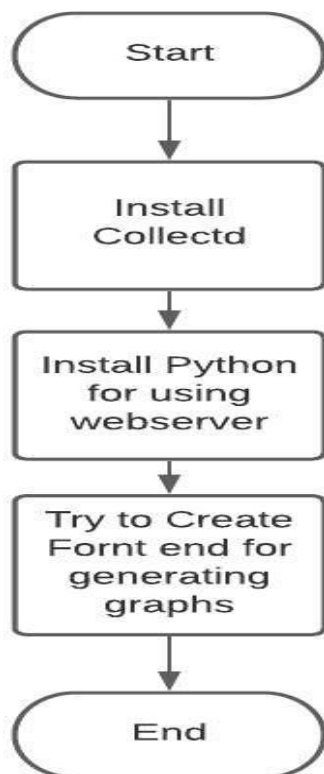
In this study, after completion of the starting phase, we proposed a method to analyse the performance of any hardware. This lets you know how hardware works. We used beaglebone black and raspberry pi for demonstration purposes. The analysis of output is reported by the collection and the results can be seen in a chart. The chart is shown with the RRD method.

1.4 SCOPE OF THE PROJECT

Developing our own Linux distro has several benefits, as only the necessary library files can be configured. The whole Linux distro is what we were looking for. We cannot get rid of redundant libraries that have more room and if we make some improvements, some mistakes might even occur. The need to build our own Linux Distro. By designing and analysing our own Linux distro

Hardware output helps us understand its behaviour. To facilitate this, we demonstrate with the aid of rrd tool the final result in a graph.

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

Shahid H. Bokhari (1995) Linux is an OS that is running on Intel-386/486/Pentium compatible IBM PC machines and processors equivalent. It was published at Helsinki University of Technology in Finland by Linus Torvalds, a graduate student. It does not contain proprietary code because it was written from scratch. It retains the copyright but allows the free release of the source code (see sidebar for "The Linux Software License"). In addition to Torvalds, the internet has been used to create an enthusiastic, worldwide volunteer community. Just a few years ago, Linux was being developed. The initial versions were unstable, and only diehard hackers were interesting. Linux has matured now, however, and can be used seriously. Many of the powerful compilers and tools created in the framework of its GNU (GNU is not Unix) project by the Free Software Foundation can run on Linux. For Linux in an Intel-386/486 version called XFree86, the de-facto standard of MIT's X Window System for windows with Uniz is also available. Therefore Linux is a full-fledged operating system which offers all the capabilities that Unix commercial systems generally offer. The C, C++, Pascal, Modula-2 and -3 compilers are excellent, Oberon, Smalltalk, Fortran. Many versions of Editor including Emacs and vi are available. The full support is the security of passwords, file encryption, multiple logins, virtual memory and multitasking. Linux networking has grown to support vast networks in their remote, remote and electronic mail connections. Linux was created with aversion from the Network File System (NFS). This allows file systems to be shared between several computers, which reduces use of hard disc and minimises overhead device administration. The Text processing framework TeX/LaTeX was ported to Linux with ghostview and xdvi previews and drawing packages (xfig and idraw). Both these features are free of charge and free of charge.

Ricardo P. Jasinski (2012), The use of an OS to facilitate software creation and liberate programmers from the use of low-level hardware and to facilitate programming of common tasks is advocated.

The high level of complexity of modern desktop computers needs an OS;

On the other hand, embedded systems are limited, typically cost-effective and power-resistant, architectures. Due to further requests from an OS, embedded developers are confronted with an important decision to decide whether or not to implement an OS. This article presents a case study that includes three separate scripts for developing a sample application (embedded weather station): using the μ C/Ost-II real-time OS without any OS, and using the general uClinux OS. An FPGA and SoPC provided a versatile hardware interface for all three configurations. By adopting an OS, development time was cut to 48%, while programme storage requirements were increased by at least 71%.

Zhongjiang Zhang (2015), Currently, because of its unique benefits such as open source and stability, Linux operating system has become an investigative subject in the field of embedded system. In this scenario, this article offers an idea for a transplantation of the navigation software from VxWorks to Linux. This paper therefore focuses on a study of the Linux and kernel start-up process based on the x86 platform, in order to achieve the goal described above. Then a full development environment and build an embedded Linux operating system will be resolved in order to satisfy the application requirements. In this article, it is recommended to back up and restore the embedded Linux using the Clonezilla tool.

Abhimanyu Sharma (2016) The Open Source Software (OSS) is an end user licence computer software with its source code. Agreement (EULA) in which the copyright holder grants every individual and for any reason the right to examine, modify and distributing the software. The public should build Open-Source software in cooperation. The most popular example of open-source development is open-source software. The open-source or collaborative competitive development from many different sources creates a wider range of design perspectives than any organisation is capable of long-term development and support. The Linux kernel is an operating system kernel used by the Unix-like Linux family. It is one of the leading examples of open source and free programming. Linux is the easiest

Conditions, a method of operations. It is the computer programme that allows applications and computer operators to have access to the computers they are looking for. The OS relays instructions to the computer, for example, from an application.

Hari Narayanan, (2017) Protection of the operating system has been a hot spot of research for decades. Several protection measures to protect the operating system have been implemented so far. We concentrate on securing Linux operating systems in this article. Though Linux is open-source and many people work with kernel patches for security troubles, many malware still exploit the current vulnerabilities. We aim to reduce damage by malware with our architecture, if we don't block it entirely. Our architecture is aimed at guaranteeing the least privilege principle. The least privileged principle ensures that a mechanism has sufficient rights to carry out its mission. This means that the system will do the least harm, even though the mechanism is compromised, since it runs in a sandbox. The biggest part of our framework runs at the consumer level to make the distribution portable. Our system uses a specially designed security ticket to give the user processes finely licenced which in conventional linux architecture is not currently possible.

The safety ticket is structured so that a child process can be inherited, shared and unforgeable. Secd, which authorises all requests and handles security tickets too, is the central module in the framework.

Marko Boras, Josip Balen, Kresimir Vdovjak, (2020), The consistency, reliability, and efficiency of the operating system is also a very important aspect in the context of software assessment[4].

However, since it requires managed environments and same conditions for all tested systems, it is a complex and time-consuming process, which remains a little challenging, in particular in the operating systems field. Several aspects of the performance of an operating system on personal computers were examined in our previous work. The full performance assessment of three versions of the Windows Betriebssystem: Windows XP, Windows [5], focused on CPU scheduling, drive management, graphic subsystem management, memory and Network Performance. It was done Vista and Windows 7. In two separate conditions measurements were also carried out. In the majority of tests performed on the low-end computer device Windows XP was higher than Windows Vista and Windows 7 for its efficiency measurement results. In addition, newer operating systems have shown increased performance in the fields of subsystems graphics and memory management on a high-end computer system. Furthermore, a model of performance

assessment has been developed and similar changes are to be made to Linux-based operating systems in this paper. In [6], the Linux Debian operating system evaluated three popular virtual private server hosting systems. In[7], the authors tested three versions of Windows, namely Windows XP, Windows Vista and Windows, in their networking performance. Networking efficiency research has continued and extended [8] in which the evaluation was carried out of three recent Windows systems, Windows 7, Windows 8.1 and Windows 10. Experimental findings indicate that Windows 10 has improved network management, delays have been decreased but use of CPUs has been increased.

I recognised the Linux operating system from the literature examined, which has been updated annually.

CHAPTER 3

METHODOLOGIES

3.1 GENERAL

We must first create our own customised distribution by carrying out a small study. These research should include the method to design the distribution we will follow, which will allow us to follow a certain pattern of growth, and all characteristics and requirements of the distribution should then be included in the research. By choosing this, we can incorporate it into the design process and eventually deliver our customised distribution that meets our requirements.

I have experienced two creation processes in my research:

1. SUV
2. Yocto

I notice that LFS has been launched early on so it can support the older versions as much as possible. I therefore chose yocto for my distribution growth. The Yocto Project is a project for the creation of Linux distributions by the Linux Foundation which is collaborative and open source. What is critical for yocto developers is that they update yocto frequently in the newer versions. The steps in your official web site are listed in detail. The next important step is the use of the collected daemon after each distribution is developed to check its performance.

3.1.1 LINUX FROM SCRATCH

Scratch Linux (LFS) is a process by which all its components can be manually constructed to instal a functioning Linux system. Of course, this is longer than installing a Linux distribution previously compiled. The benefits of this approach are a lightweight, scalable and stable framework and a better understanding of Linux-based

operating systems' internal operating systems, says Linux from Scratch. The Scratch Linux Project is like BitBake, which supports Linux cross-compiling for embedded ARM systems like the Raspberry Pi and BeagleBone.

REQUIREMENTS AND PROCEDURE

A clean partition and a working Linux framework must be built with a compiler and some important software libraries. You may also use a Live CD to set up an LFS system rather than instal an existing system installed for Linux. Initially, a compilation must be made of the LFS compilation tools, such as GCC, glibc, binutils and other necessary tools. The root directory must then be changed to the toolchain partition by chroot to initiate the construction of the final structure. Glibc is a first package to build in gcc; after that the connector of the toolchain must be changed so that all other packages which make up the completed framework that can be linked against the newly developed glibc are linked to it. The bash hacking function is switched off and the bin directory of the temporary toolchain is moved to the end of PATH during the chroot process. The newly compiled programmes that are first developed in PATH and the new framework are therefore based on their own new components.

STANDARD BUILT UNIT(SBU)

The standard build unit represents the time taken to create the first package in LFS on the provided machine during the initial set-up of the device. This was driven by the length of time it took to create an LFS system and by the desire of many users to know how long a source tarball would take in advance.

The first package developed by the user is GNU binutils from Linux From the Scratch version 6.3. Users are invited to test construction processes using shell constructs when constructing the Linux kernel using LFS and dub this time the standard system build device.

The GNU C library (rated at 4.1 SBUs) and GNU Compiler Collection have so many packages which are constructed during compilation that they take a lot longer than binutils (rated at 8.4 SBUs). This unit should be understood as an approximation; different factors affect the actual time needed to construct a package.

KERNEL/DISTRO VERSION

In this project, we use Fedora Core 33. We can use a 5.x kernel for any distribution but be mindful that this may lead to discrepancies in the build process. Some distributions may use older and incompatible releases to bundle utilities than is necessary in this course.

CLEAN INSTALL

We must clean the build system installation. Any virtualization environment can be finished, as long as we have access to the console at different stages. Real hardware is also acceptable if we can readily find such tools.

DEVELOPMENT TOOLS

GCC, binutils, and other packages from the package group "development tools" must be installed. This is something we can choose during installation or install with the group install option for yum or dnf. The kit's names can differ depending on the delivery, but in general, we will need to install groups of "development instruments" and "C development tools."

3.1.2 YOCTO

Via the Free Embedded Build system, the Yocto Project provides an open source development environment for a variety of platforms, including x86-64, MIPS, PowerPC, and x86 architectures. To develop, build, debug, simulate, and test Yocto Project components, the entire software stack with Linux, the X Windows System, GTK+ framework systems, and Qt frameworks can be used.

- Yocto includes a recent Linux kernel and an embedded environment collection of system commands and libraries.

It also allows components such as X11, GTK+, Qt, Clutter and SDL (including) to create a rich user experience on display-hardware devices. These components do not need to be assembled if you do not have a monitor or choose to use alternative UI framing.

- Creates an open embedded core that's oriented and secure, compliant with the OpenEmbedded project.
- It supports completely a wide variety of EMUlator hardware and system emulation (QEMU).
- It offers a layer structure to conveniently enlarge, customise and maintain the framework.

We can make images for many types of devices with the Yocto Project. With Yocto Project, we can boot and emulate using QEMU for the development of reference images. The standard machines aim QEMU complete device emulation of x86, ARM, MIPS and PowerPC versions for 32-bit and 64-bit. Over and above emulation, we can extend support for almost every platform Linux can run and a toolchain can aim through the layer mechanism.

PREREQUISITES FOR THE DEVELOPMENT MACHINE:

- At least 4 - 6 GB RAM.
- Recent Ubuntu OS (20.04 LTS in this case).
- There is a need of atleast 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.

3.1.3 BUILDING IMAGES

- Create and run a qemu86 reference image in the QEMU emulator.
- Adjust configurations easily so that you can generate a second picture for MinnowBoard MAX-compatible boards.
- ***Be Sure our Build Host is Set Up:*** The steps to build an image is depend on our build host being properly set up.

- ***Check Out our Branch:*** Be sure you are in the Source Directory, for example poky, which check out the branch associated with the latest Yocto Project Release:

```
$ cd ~/poky
```

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

Git's checkout command will check 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 our own branch based on the released branch ensures we are using the latest files for that release.

Initialize the Build Environment: The oe-init-build-env environment setting script will be executed to identify your build host with the OpenEmbedded environment.

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

This, among other things, creates the Build Directory, which is created and placed in the source directory in this case. After the script runs, our working directory is now set to the Build Directory. When the building is finished, the Build Directory contains all of the files created during the build.

Examine our Local Configuration File: A local configuration file called loc.conf will be available in the conf subdirectory of the project directory when you set up a build setting. You should look at this file before you use BitBake when beginning the building process and ensure that your general settings are as follows:

You should add a declaration to your project's configuration file, which in this case is poky/build/conf/local.conf, to help save disk space during builds. After the recipe has been created, this will delete the directory that was used to build it.

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

- The target build machine is Qemux86, which generates a picture that can be used on an Intel® 32 bit architecture using the QEMU emulator. This default is easily modified by the MACHINE variable to quickly construct an image for another machine in this example.

- The package manager used to create a picture is another factor before we design. The RPM package manager is chosen for the default local.conf file. The PACKAGE CLASSES variable allows us to manage this configuration.

Start the Build: 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, we can start QEMU and run the image:

```
$ runqemu qemux86
```

Exit QEMU: This is 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 periodically collects system and application performance metrics and provides mechanisms for the saving of values in various ways, such as RRD files. Collects and preserves information from or across the network, collects metrics from different sources, such as operating systems, software, logfiles and external devices. These statistics can be used for the surveillance of systems, the detection of performance bottlenecks, and prediction of future loading of the system (i.e. capacity planning).

3.2.1 LIMITATIONS

You do not make graphics but RRD files. Cannot generate the graphs from these files. But there's a tiny sample script in the contribution.

The monitoring function has been added in version 4.3 but is so far limited to simple threshold controls.

3.3 RRD TOOL

RRDtool is the industry standard for open-source, high-performance time series data logging and graphics systems. In shell, perl, python, ruby, lua and tcl applications RRDtool is easy to integrate into.

It is very simple to collect data on all kinds of items, from our office's temperature to the amount of octets passed through our router's FDDI interface. However, the effective and systematic storage of this data is not so straightforward. This is where RRDtool is useful. It allows us to log and review the information you collect from all types (DS). This tool's data analysis is focused on the capability of generating graphical representations of the data values obtained over a specified period of time.

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

3.4 WORKING OF RRD TOOL

DATA ACQUISITION

It is convenient to have the data accessible at constant time intervals when tracking the status of a device. Sadly, at the exact moment you like, we can not always be able to retrieve data. RRDtool allows us, therefore, to refresh the log file whenever you want. The value of the data source (DS) at the latest official time-slot (interval) shall be automatically interpolated and the interpolated value shall be added to the log. The original value we have given is also stored and considered when the next log entry is interpellated.

CONSOLIDATION

The log data at a time interval of 1 minute, but the creation of the information over the last year may also be of interest to you. We can do that only by storing the data for the entire year in 1 minute intervals. It would take a lot of discretion to interpret the data when you tried to build an entire year-round graph. It would also take a lot of time. When establishing a Round Robin Database (RRD), the interval of consolidation is defined and the average, minimum, maximum and last consolidation function (CF) is used to establish the consolidated values. There are various configurations within one RRD that we can describe. Both of these are kept on flying as new information is loaded into the RRD.

ROUND ROBIN ARCHIVES

In the Round Robin Archives, consolidation value is stored (RRA). It is a very effective way, when using a known consistently large amount of storage space, to store data for a while.

RRDtool will reserve space for 1'000 data and a headerspace to store 1,000 values in the 5-minute interval. It stores a pointer in the header to say which slots were last written to in the storage field. This limits the history to the last 1'000 values automatically. We can set a different RRA for a storage of about 750 data values at 2 hour intervals since we can identify multiple RRAs in one single RRD.

The use of RRAs ensures that the RRD is not expanded over time and that old data is discarded automatically. With the consolidation function, data can be retained very long, and the data resolution along the time axis can be progressively reduced.

Through the use of various CF functions, we can store precisely the sort of information we really need.

UNKNOWN DATA

At a constant interval, the RRD stores data. Often, when a value needs to be written to the RRD, no new data is available. By saving an uncertain value in a data base, we can manage these circumstances with RRD tool. Both functions of the tool endorse the unknown value. Data consolidation

In collection, a new unified value is ready to be entered into the Round Robin archive the unknown values are taken into account (RRA). A validity check is carried out to verify that the unknown data point percentage is greater than a configurable number. Otherwise, the RRA is given an undefined value.

GRAPHING

The RRD tool helps us to produce reports on the basis of data stored in one or more RRDs in numerical and graphical form. The graphic function can be completely configured. The graph can be described freely in the size, colour, and contents.

To function with RRDtool graphs, data should be collected using one or more statements on data classification. The collection of data from two or more databases is not limited to one single database, it is completely lawful.

Then the data from the Round Robin Archives are consolidated to provide an exact pixel-by-pixel data point. If we do not take care, the RRD tool can, if possible, extend the range slightly.

Data is often not exactly in the format that we want to view. For example, we can collect bytes per second, but want bits per second displayed. This is the intent of the data calculation command. After the data is consolidated, a copy is made and a powerful RPN command set is used for modification of this copy.

We will graph it when we have the data fetched and processed. This finishes the series of the rrdtool graph.

CHAPTER 4

RESULT

The kernel image should be built in order to set up our own linux distro for measuring the performance analysis. For building image we can do it in any kind of distributions, such as ubuntu, fedora or openSUSE.

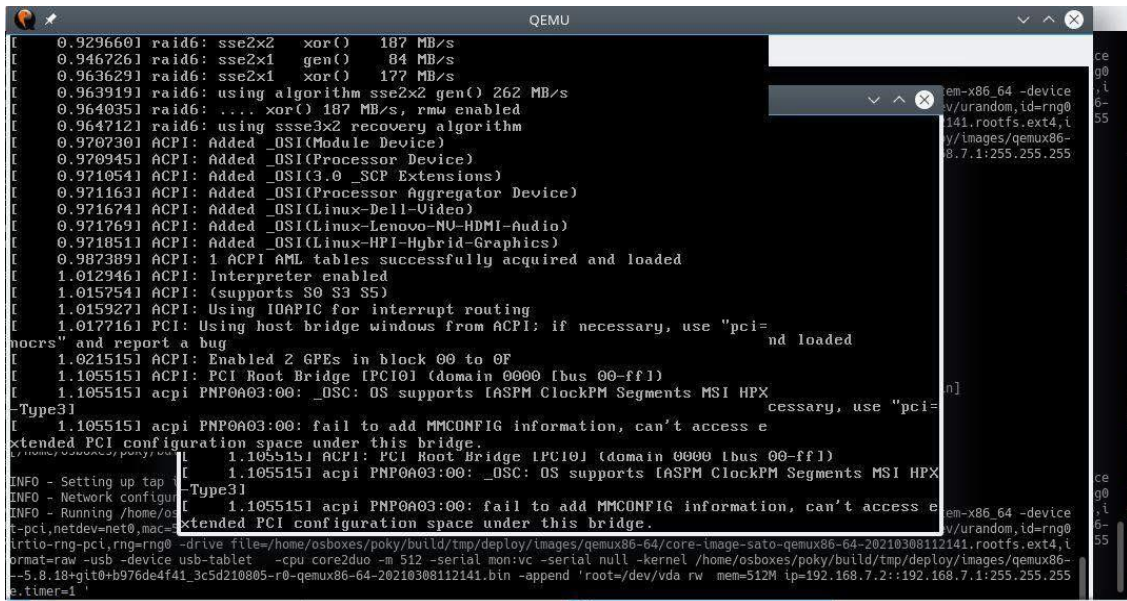
The medium in which we built can also defer such as LFS or Yocto.

LFS supports the older version and showing error with the most updated version.

Even yocto supports till the next lesser version to the updated one.

Thus I set up the yocto in openSUSE 15.1, this version is the next lesser one to the updated one. Below image is the built image in openSUSE.

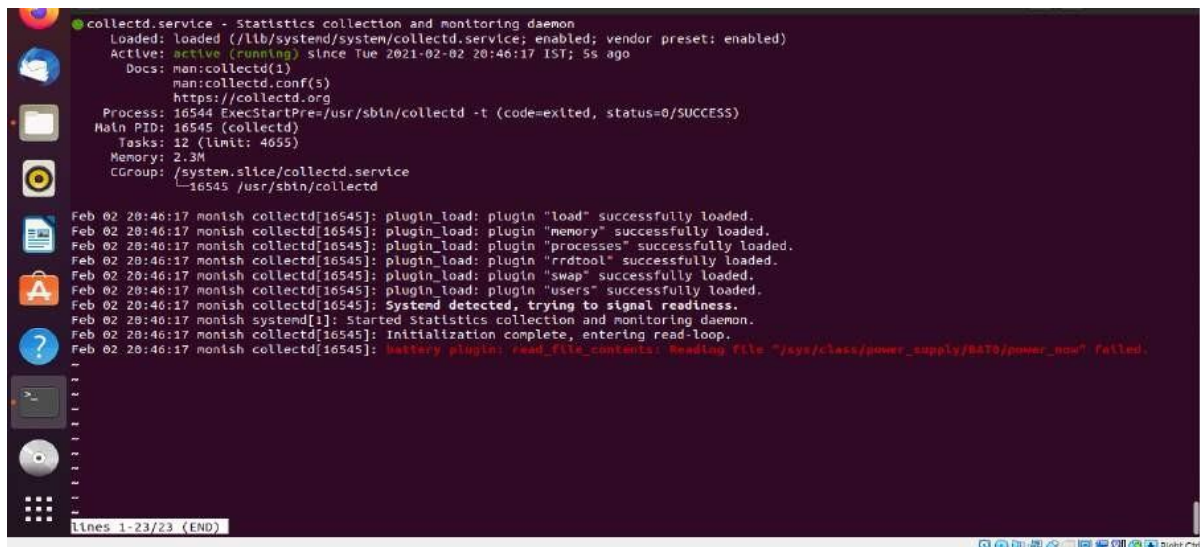
```
osboxes@linux-qtek:~/poky/build/tmp/deploy/images/qemux86-64> ls
bzImage
bzImage--5.8.18+git0+b976de4f41_3c5d210805-r0-qemux86-64-20210308112141.bin
bzImage-qemux86-64.bin
core-image-sato-qemux86-64-20210308112141.qemuboot.conf
core-image-sato-qemux86-64-20210308112141.rootfs.ext4
core-image-sato-qemux86-64-20210308112141.rootfs.manifest
core-image-sato-qemux86-64-20210308112141.rootfs.tar.bz2
core-image-sato-qemux86-64-20210308112141.testdata.json
core-image-sato-qemux86-64.ext4
core-image-sato-qemux86-64.manifest
core-image-sato-qemux86-64.qemuboot.conf
core-image-sato-qemux86-64.tar.bz2
core-image-sato-qemux86-64.testdata.json
modules--5.8.18+git0+b976de4f41_3c5d210805-r0-qemux86-64-20210308112141.tgz
modules-qemux86-64.tgz
osboxes@linux-qtek:~/poky/build/tmp/deploy/images/qemux86-64> █
```



```
0.9296601 raid6: sse2x2 xor() 187 MB/s
0.9467261 raid6: sse2x1 gen() 84 MB/s
0.9636291 raid6: sse2x1 xor() 177 MB/s
0.9639191 raid6: using algorithm sse2x2 gen() 262 MB/s
0.9640351 raid6: .... xor() 187 MB/s, rmw enabled
0.9647121 raid6: using sse3x2 recovery algorithm
0.9707301 ACPI: Added _OSI(Module Device)
0.9709451 ACPI: Added _OSI(Processor Device)
0.9710541 ACPI: Added _OSI(3.0 _SCP Extensions)
0.9711631 ACPI: Added _OSI(Processor Aggregator Device)
0.9716741 ACPI: Added _OSI(Linux-Dell-Video)
0.9717691 ACPI: Added _OSI(Linux-Lenovo-NV-HDMI-Audio)
0.9718511 ACPI: Added _OSI(Linux-HPI-Hybrid-Graphics)
0.9873891 ACPI: 1 ACPI AML tables successfully acquired and loaded
1.0129461 ACPI: Interpreter enabled
1.0157541 ACPI: (supports S0 S3 S5)
1.0159271 ACPI: Using IOAPIC for interrupt routing
1.0177161 PCI: Using host bridge windows from ACPI: if necessary, use "pci=
nocrs" and report a bug
1.0215151 ACPI: Enabled 2 GPEs in block 00 to 0F
1.1055151 ACPI: PCI Root Bridge [PCI0] (domain 0000 [bus 00-ff])
1.1055151 acpi PNP0A03:00: _OSC: OS supports [ASPM ClockPM Segments MSI HPX
-Type3]
1.1055151 acpi PNP0A03:00: fail to add MMCONFIG information, can't access e
extended PCI configuration space under this bridge.
1.1055151 ACPI: PCI Root Bridge [PCI0] (domain 0000 [bus 00-ff])
1.1055151 acpi PNP0A03:00: _OSC: OS supports [ASPM ClockPM Segments MSI HPX
-Type3]
1.1055151 acpi PNP0A03:00: fail to add MMCONFIG information, can't access e
extended PCI configuration space under this bridge.
INFO - Setting up tap
INFO - Network configu
INFO - Running /home/os
t-pci,netdev=net0,mac=
rtio-rng-pci,rng=rng0 -drive file=/home/osboxes/poky/build/tmp/deploy/images/qemux86-64/core-image-sato-qemux86-64-20210308112141.rootfs.ext4,i
format=raw -usb -device usb-tablet -cpu core2duo -m 512 -serial mon:vc -serial null -kernel /home/osboxes/poky/build/tmp/deploy/images/qemux86-
-5.8.18+git0+b97de4f41_3c5d210805-r0-qemux86-64-20210308112141.bin -append 'root=/dev/vda rw mem=512M ip=192.168.7.2::192.168.7.1:255.255.255
etimer=1'
```

This image will be booted through QEMU for further process.

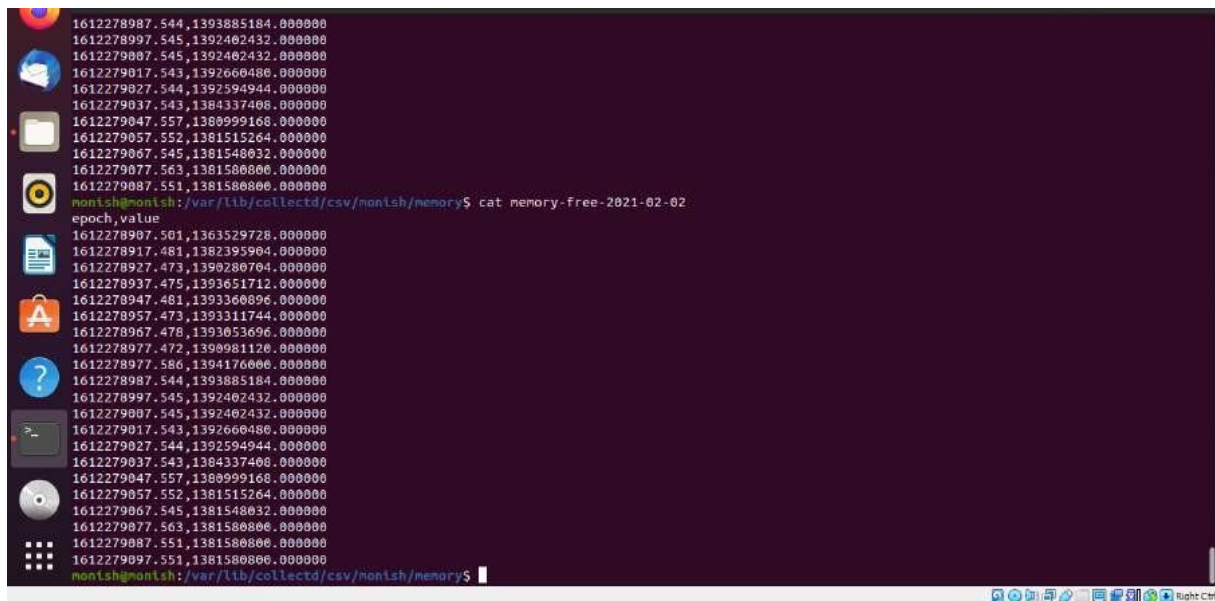
Then the measured values will be collected by collectd. This collectd is checked in ubuntu. The below image will depict the running status of collectd.



```
collectd.service - Statistics collection and monitoring daemon
Loaded: loaded (/lib/systemd/system/collectd.service; enabled; vendor preset: enabled)
Active: active (running) since Tue 2021-02-02 20:46:17 IST; 5s ago
Docs: man:collectd.conf(5)
      https://collectd.org
Process: 16544 ExecStartPre=/usr/sbin/collectd -t (code=exited, status=0/SUCCESS)
Main PID: 16545 (collectd)
Tasks: 12 (limit: 4055)
Memory: 2.3M
CGroup: /system.slice/collectd.service
        └─16545 /usr/sbin/collectd

Feb 02 20:46:17 monish collectd[16545]: plugin_load: plugin "load" successfully loaded.
Feb 02 20:46:17 monish collectd[16545]: plugin_load: plugin "memory" successfully loaded.
Feb 02 20:46:17 monish collectd[16545]: plugin_load: plugin "processes" successfully loaded.
Feb 02 20:46:17 monish collectd[16545]: plugin_load: plugin "rrdtool" successfully loaded.
Feb 02 20:46:17 monish collectd[16545]: plugin_load: plugin "swap" successfully loaded.
Feb 02 20:46:17 monish collectd[16545]: plugin_load: plugin "users" successfully loaded.
Feb 02 20:46:17 monish collectd[16545]: Systemd detected, trying to signal readiness.
Feb 02 20:46:17 monish system[1]: Started Statistics collection and monitoring daemon.
Feb 02 20:46:17 monish collectd[16545]: Initialization complete, entering read-loop.
Feb 02 20:46:17 monish collectd[16545]: battery plugin: read_file_contents: Reading file "/sys/class/power_supply/BAT0/power_now" failed.
```

The below image shows how the collectd collects data of different parameters. The parameters can be cpu, memory, load etc. These parameters are then represented by graph with the help of rrd tool.



```

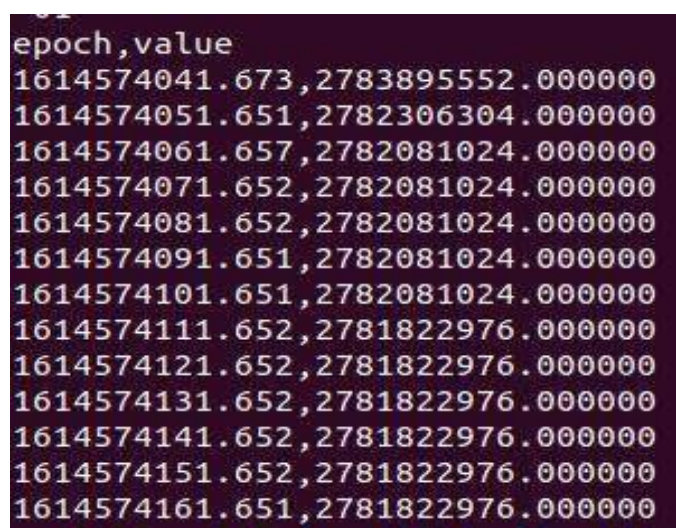
1612278987.544,1393885184.000000
1612278997.545,1392462432.000000
1612279007.545,1392462432.000000
1612279017.543,1392660480.000000
1612279027.544,1392594944.000000
1612279037.543,1384337408.000000
1612279047.557,1380999168.000000
1612279057.552,1381515264.000000
1612279067.545,1381548032.000000
1612279077.563,1381580800.000000
1612279087.551,1381580800.000000
nonish@nonish: /var/lib/collectd/csv/nonish/memory$ cat memory-free-2021-02-02
epoch,value
1612278907.501,1363529728.000000
1612278917.481,1382395904.000000
1612278927.473,1390280704.000000
1612278937.475,1393651712.000000
1612278947.481,1393360896.000000
1612278957.473,1393311744.000000
1612278967.478,1393053696.000000
1612278977.472,1390981120.000000
1612278977.586,1394176000.000000
1612278987.544,1393885184.000000
1612278997.545,1392462432.000000
1612279007.545,1392462432.000000
1612279017.543,1392660480.000000
1612279027.544,1392594944.000000
1612279037.543,1384337408.000000
1612279047.557,1380999168.000000
1612279057.552,1381515264.000000
1612279067.545,1381548032.000000
1612279077.563,1381580800.000000
1612279087.551,1381580800.000000
nonish@nonish: /var/lib/collectd/csv/nonish/memory$

```

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

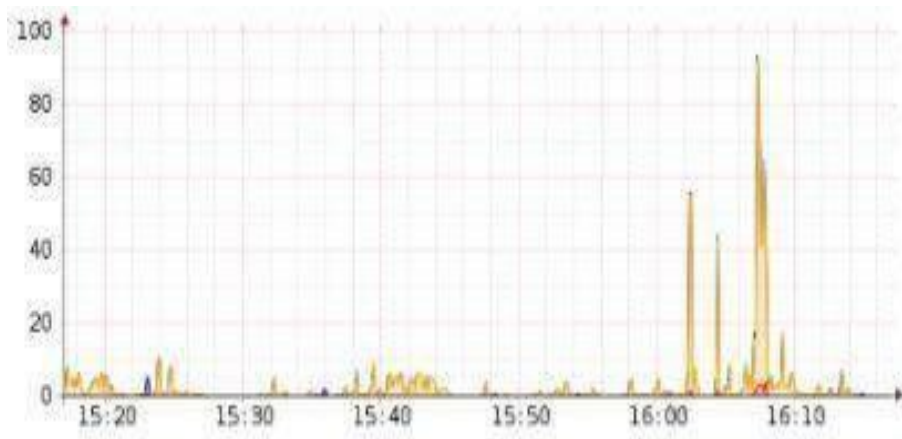
Epoch means the no. of seconds since the distribution is developed. The value refers to the value of the respected plugin that we enabled.



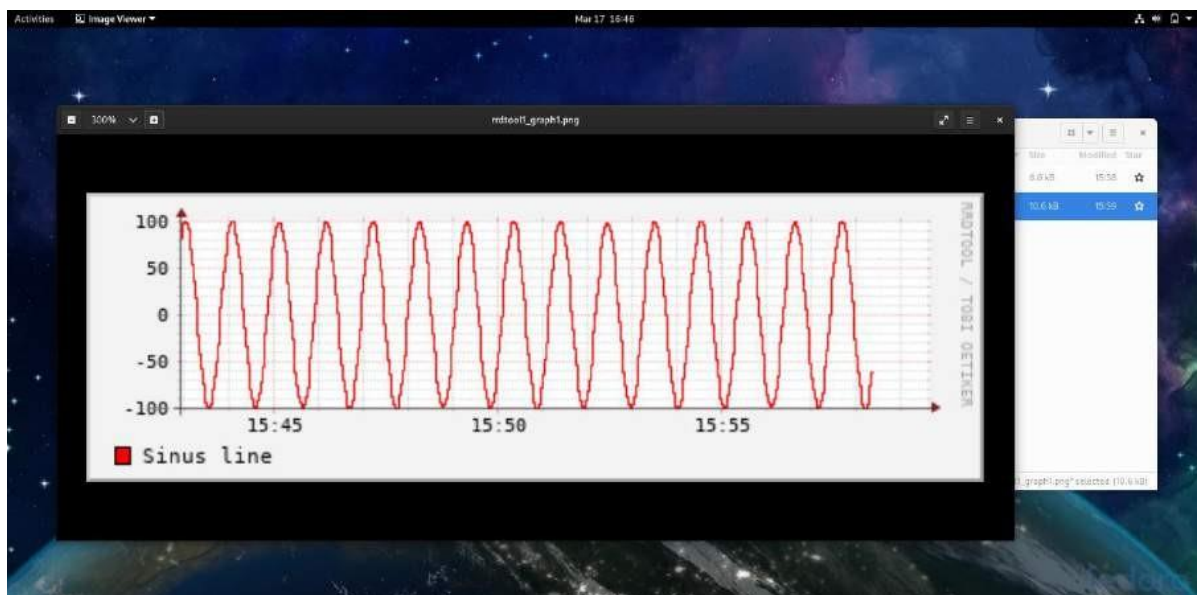
```

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

```



Below is the graph which I got it for the data base file. In this, the x axis denotes the time and the y axis denotes the average. This graph is represented as sine wave.



CHAPTER 5

CONCLUSION

From this project it can be concluded that any Linux distribution can be used, it doesn't concern in checking the performance of the hardware. For understanding the performance analysis in detail, a tool is used which is called as collectd. Collectd is used to collect all the data and can be customized according to our need. The data can be cpu-info, memory, load, etc. These data can be represented in a graph. A tool named rrd tool is used for representing the data in graph. Finally the data will be represented in graph and the final output is in the form of sine wave. Sine wave will be quiet easy to understand the performance analysis similar to the one that can be observed in task manager which is in our own laptops.

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/>

15. <https://docs.yoctoproject.org/ref-manual/system-requirements.html#opensuse-packages>
16. <https://docs.yoctoproject.org/brief-yoctoprojectqs/index.html>
17. <https://docs.yoctoproject.org/dev-manual/start.html#setting-up-to-use-windows-subsystem-for-linux-wslv2>
18. <https://blog.tinned-software.net/understand-the-basics-of-rrdtool-to-create-a-simple-graph/>

CUSTOM LINUX DISTRO FOR PERFORMANCE ANALYSIS USING LINUX FROM SCRATCH (LFS)

by S. Vishnu Preetha

Submission date: 17-Apr-2021 05:10AM (UTC-0700)

Submission ID: 1561769403

File name: 37130441_Vishnu_Preetha._1704.docx (808.33K)

Word count: 7108

Character count: 37643

CUSTOM LINUX DISTRO FOR PERFORMANCE ANALYSIS USING LINUX FROM SCRATCH (LFS)

Submitted in partial fulfilment of the requirements for the award of Bachelor of
Engineering Degree in Electronics and Communication Engineering

by

S. VISHNU PREETHA (37130441)



**DEPARTMENT OF ELECTRONICS AND COMMUNICATION
ENGINEERING SCHOOL OF ELECTRICAL AND
ELECTRONICS ENGINEERING**

**SATHYABAMA INSTITUTE OF SCIENCE AND
TECHNOLOGY (DEEMED TO BE UNIVERSITY)**

Accredited with Grade "A" by NAAC
JEPPIAAR NAGAR, RAJIV GANDHI SALAI, CHENNAI - 600 119

APRIL 2021



¹³
SATHYABAMA
INSTITUTE OF SCIENCE AND
TECHNOLOGY
(DEEMED TO BE UNIVERSITY)



Accredited with Grade “A” by NAAC

JEPPIAAR NAGAR, RAJIV GANDHI SALAI, CHENNAI - 600 119

www.sathyabama.ac.in

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

BONAFIDE CERTIFICATE

This is to certify that this Project Report is the bonafide work of **S.VISHNU PREETHA (37130441)** who carried out the project entitled “**CUSTOM LINUX DISTRO FOR PERFORMANCE ANALYSIS USING LINUX FROM SCRATCH (LFS)**” under our supervision from September 2020 to April 2021.

Internal Guide

Dr. P.CHITRA., Ph.D.

Department of ECE

Head of the Department

Dr. T. RAVI, M.E., Ph.D.

Submitted for Viva voce examination held on _____

Internal Examiner

External Examiner

DECLARATION

I, **S. VISHNU PREETHA(37130441)** hereby declare that the Project Report entitled **“CUSTOM LINUX DISTRO FOR PERFORMANCE ANALYSIS USING LINUX FROM SCRATCH (LFS)”** done by us under the guidance of **Dr.P.CHITRA., Ph.D,** is submitted in partial fulfilment of the requirements for the award of Bachelor of Engineering degree in Electronics and Communication Engineering.

DATE:

PLACE:

SIGNATURE OF THE CANDIDATE

ACKNOWLEDGEMENT

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

I convey my thanks to **Dr. N. M. NANDHITHA, M.E., Ph.D. Dean, School of Electrical and Electronics Engineering** and **Dr. T. RAVI, M.E., Ph.D. Head of the Department, Department of Electronics and Communication Engineering** for providing me with 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 **Dr.P.CHITRA., Ph.D, Department of Electronics and Communication Engineering** for her valuable guidance, suggestions and constant encouragement paved way for the successful completion of the project work.

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

I express my gratitude to my parents for their constant encouragement and support for the completion of the project.

ABSTRACT

In the early 1990s, Linus Torvalds developed Linux as a UNIX-like open-source framework. Many Linux discs, including Fedora, Red Hat, Ubuntu, Centos and others, are available. A Linux-like distribution but without any of the Linux utilities. This type of delivery is usually used to decrease load and improve output only by providing the necessary utilities. Other distributions are well-known for their specifications such as Fedora, Red Hat, Ubuntu, and others. Why not distribute ourselves using the requested packages only if you can build distributions of your own to fulfil your privileges? This can be done by many Linux development distributions, including Yocto, LFS and others. The main thing is to find out what to do first. In my project, I use Yocto to build the very least bundle of my own personalised distro that meets my requirements. The goal is not just to create our own results, it is also to analyse how it functions and how its CPU, memory and other resources are used. I will use the Collected Daemon for performance analysis. A decision has been reached. Daemon is a programme that collects data and analyses our distro efficiency in a context. The collected data is then used to measure the consistency of any distribution.

TABLE OF CONTENTS

TITLE	PAGE NO.
ABSTRACT	v
LIST OF FIGURES	VII I

CHAPTER NO.	TITLE	PAGE NO.
1.	INTRODUCTION	9
1.1	GENERAL	9
1.1.1	COMPONENTS OF LINUX SYSTEM	9
1.1.2	BASIC FEATURES	10
1.1.3	LINUX DISTRIBUTIONS	10
1.1.4	ARCHITECTURE OF LINUX KERNEL	11
1.1.5	BOOTLOADER	14
1.1.6	ROOTFILE SYSTEM	16
1.1.7	BUILD SYSTEM	18
1.2	OBJECTIVE	19
1.2.1	REQUIREMENTS	20
1.2.2	SOFTWARE AND HARDWARE CONFIGURATION	20
1.3	PROPOSED METHOD	20
1.4	SCOPE OF THE PROJECT	20
1.5	ADVANTAGES OF PROPOSED SYSTEM	21
2.	LITERATURE SURVEY	22
3.	METHODOLOGIES	26
3.1	GENERAL	26
3.1.1	LINUX FROM SCRATCH	26
3.1.2	YOCTO	28
3.1.3	BUILDING IMAGES	29
3.2	COLLECTD	31
3.2.1	LIMITATIONS	

3.3	RRD TOOL	32
3.3.1	WORKING OF RRDTOOL	32
4.	RESULT	35
5.	CONCLUSION	39
6.	REFERENCES	40

LIST OF FIGURES

FIGURE NO.	TITLE	PAGE NO.
1.	COMPONENTS OF LINUX SYSTEM	10
1	ARCHITECTURE OF LINUX KERNEL	12
1.	KERNEL SUBSYSTEMS	13
2	BOOTLOADER PROCESS ROOTFILE	16
1.	SYSTEM	18
3	BUILD SYSTEM	19
1.	BLOCK DIAGRAM	21
4		
1.		
5		
1.		
6		
1.		
7		

CHAPTER 1

INTRODUCTION

1.1 GENERAL

Linux is an open source operating system (OS). An operating system is the software that manages a system's hardware and resources like CPU, memory, and storage. The OS lies 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** – The main component of Linux is the kernel. It is in charge of all major operating system operations. It consists of a number of modules and interfaces with the base hardware directly. It provides the necessary abstraction to hide device and application programmes with low-level hardware data.
- **System Library** – There are special features or programmes that control kernel features with applications or machine utilities. These system libraries implement much of the operating system functionality and do not include code access rights to the kernel module.
- **System Utility** – These programs are responsible to do specialized individual level tasks.

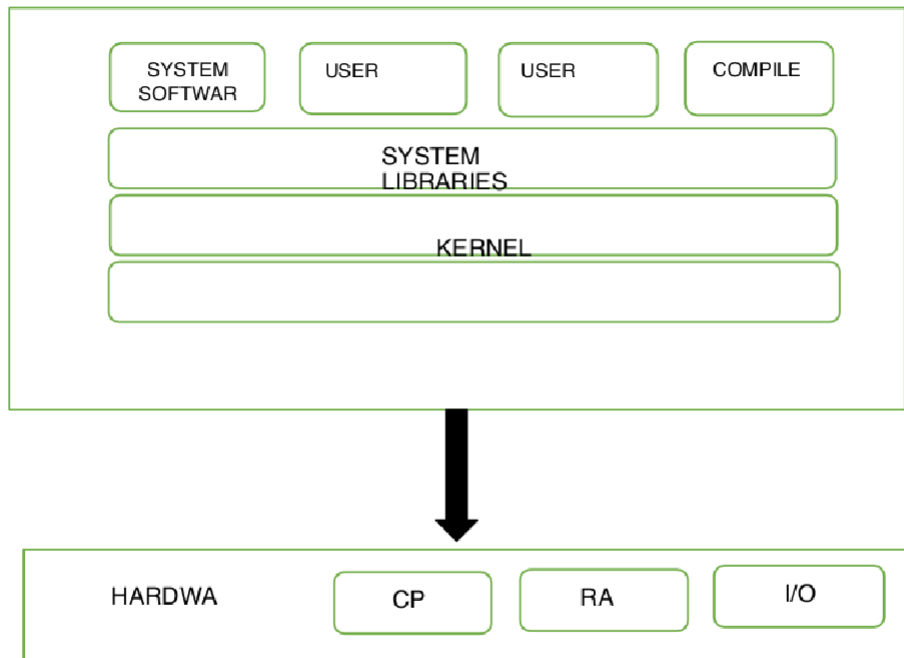


FIG 1.1 COMPONENTS OF LINUX SYSTEM

1.1.2 BASIC FEATURES:

- **Portable** – Portability means that software can function in the same way with various kinds of hardware. Linux kernel and application programmes can be installed in any various hardware platform.
- **Open Source** – Linux Source Code is an open source code freely available and a development project focused on a collective. Many teams work together to improve Linux operating system capabilities and are constantly changing.
- **Multi-User** – The Linux kernel is a multiuser system that means that several users can simultaneously access system resources such as memory/ram/application programmes.
- **Multiprogramming** – It is a multiprogramming system means multiple applications can run at same time.

- **Hierarchical File System** – It provides a standard file structure in which system files/ user files are arranged.
- **Shell** – Linux contains an interpreter programme for executing operating system commands. It can be used to do all kinds of activities, apps and so on.
- **Security** – Linux provides user security using safety features like password protection/ controlled access to specific files/ encryption of data.

1.1.3 LINUX DISTRIBUTIONS

Triple distributed families exist: Debian family systems like Ubuntu, SUSE family systems like openSUSE and Fedora family systems like CentOS

1.1.4 ARCHITECTURE OF LINUX KERNEL

It consists of three main components:-

- Call system interface – is the highest level and performs simple activities including reading and writing.
- The kernel code, which is common to the linux-supported processor architectures [architecture independent kernel code].
- Code depending on the architecture - it is called a Board support package (BSP). BSP is a small bootloader programme which places drivers and operating systems.

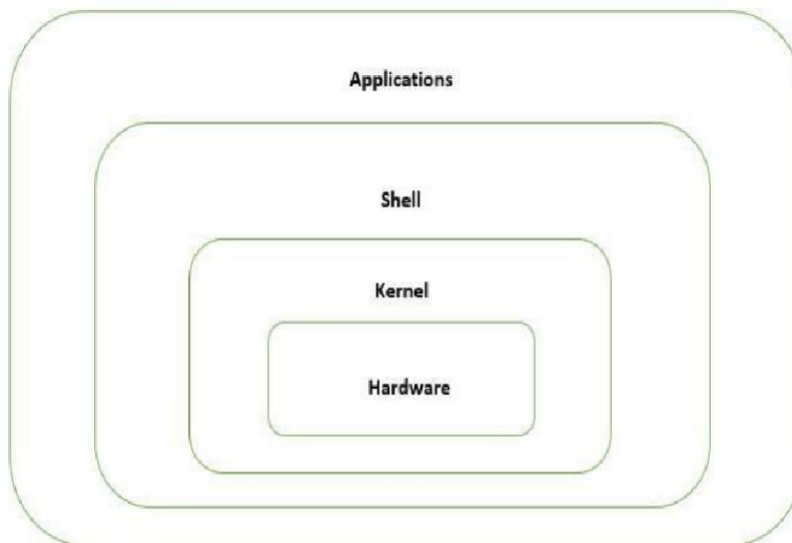


FIG 1.2 ARCHITECTURE OF LINUX KERNEL

User Space

These are the user applications that are currently running. The GNU C Library is still available (glibc). The system call interface connects to the kernel and provides a way for the user space application and the kernel to communicate.

Kernel Space

As a result, the Linux kernel is open, which can be divided into three stages. The computer call interface is a higher-level interface that employs simple reading and writing functions. At the system call interface below, the kernel can be represented more precisely in architecture independent kernel code. Both Linux-supported processor architectures use the same core code. The architecture-dependent code, commonly referred to as BSP, is located below the kernel code (Board Support Package). The BSP code serves as the processor and platform code for the basic architecture.

CORE SUBSYSTEMS OF THE LINUX KERNEL

- **Process scheduler**

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

- **Memory management unit**

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

- **Virtual file system**

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

- **Networking unit**

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

- **Inter process communication**

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

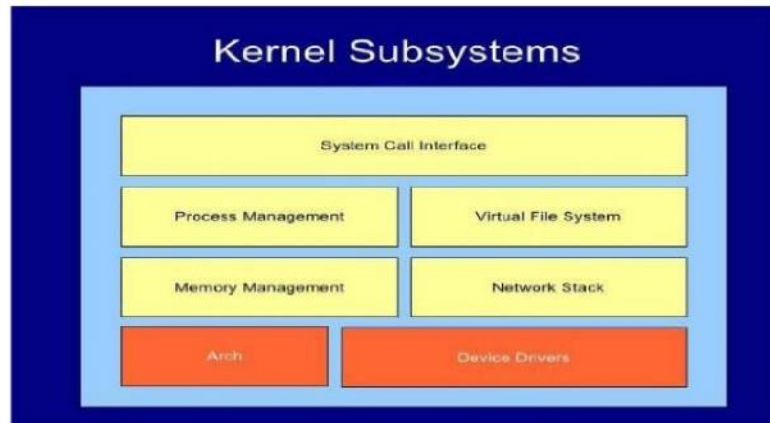


FIG 1.3 KERNEL SUBSYSTEMS -

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.
- Kernel internal-

1.1.5 BOOTLOADER

Bootloader is a programme that loads the kernel with the optional kernel parameters and the initial initrd RAM drive. It normally includes several ways to boot the OS kernel and also includes debug and kernel environment commands. The heart of the linux operating system is kernel. It begins immediately after loading the init process such as the framework. RAM stores temporary storage space until a real root device can be installed for loading vital files into the memory.

The oldest BIOS bootloader computers are located in the Master Boot Record(MBR), with a 512 byte memory. New EFI computers store them in an EFI system partition called Unified Extensible Firmware Interface (EFI).

After the active POST bootloader is loaded with BIOS or UEFI (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-testing operation that occurs when the computer is turned on. This is the first step in the Linux boot sequence. It's the system's booting hardware, and it's the same with all operating systems. When power is applied to the unit, the POST is activated, and the BIOS section is activated.

Many machines will not run if they crash, and the boot process will be terminated. POST guarantees that the hardware is operational. If POST is successful, it will interrupt the BIOS and locate the boot sectors on any attached bootable devices.

The boot loader sector is the first stage of the boot loader, and it searches for a valid boot record to load into RAM. This gives power to the code that was loaded from the boot area.

BOOTLOADER(GRUB 2)

GRUB 2 is the default boot loader for most modern Linux distributions. It allows the system to locate and save OS kernels. It is multiboot compliant, allowing you to boot a variety of Linux and other free operating systems.

It allows you to boot to a previous kernel if you are unable to boot an updated version or if an important software component is not compatible.

KERNEL INITIALISATION

To save space, both Linux kernels are compressed and self-extracted. The kernels are stored in /boot directories alongside the initial RAM disc picture and hard drive device maps.

When it's loaded, it needs to be removed from the compressed version before it can do anything useful.

The computer is then dismantled, loaded, and operated.

The boot process is now complete.

STARTUP PROCESS

Systemd is the mother of all process.



FIG 1.4 BOOTLOADER PROCESS

TYPES OF BOOTLOADER

GRUB (Grand Unified Bootloader)

It is the most common bootloader for Linux' operating system with most of its features. More potent and healthier, modern GRUB.

LIFO(Sound Loader):-

It was once common, but it gradually fell away because the multi-boot environment was not supported.

SYSLINUX

It is collection of several light weight boot loaded th 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 most basic part in Linux is the root file system. Anything necessary to support the complete Linux system is in a root file system. All apps, settings, computers, data and more are contained on the root file system. Linux framework can't run without the root file system.

All the files required for booting the Linux system must be included before mounting other file systems. It includes the Linux computer-style bootloader and executable kernel and configurations files.

PURPOSE OF ROOT SYSTEM

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

- To boot the 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 any other partitions or filesystems.
- To enable recovery and 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, the necessary 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

Directory Description

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

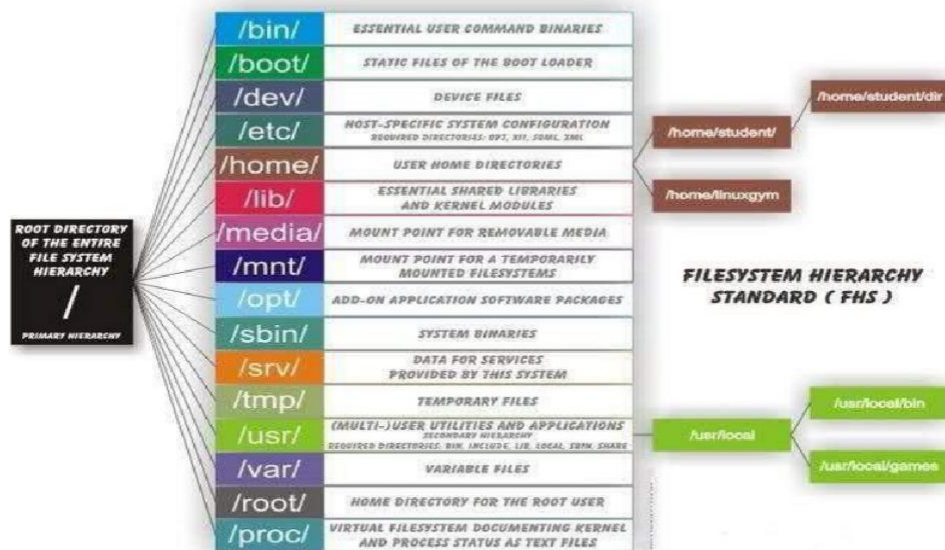


FIG 1.5 ROOTFILE SYSTEM

1.1.7 BUILD SYSTEM

Kernel Build System is the infrastructure for controlling image building and its modules (kbuild).

The four key components of the Linux kernel build system are:

- Symbols configuration: the compilation options for compiling code in source files on a conditional basis and for choosing which objects to include in an image or its modules of the kernel.
- Kconfig: it describes the form, definition and dependencies of any of the config symbols and attributes of them. The choice menu box (for example, make menu config) is created by this programme and reads the menu items of those files.
- .cfg file: stores the selected value of each config symbol. We are able to edit it

manually or use one of the several set-up goals, such as menuconfig and xconfig, to construct a tree-like menu and to update and create the .config.

- Makefiles: The Normal GNU Makefiles describe the connection between the source files and the commands that each make target, for example kernel images and modules, needs to generate.

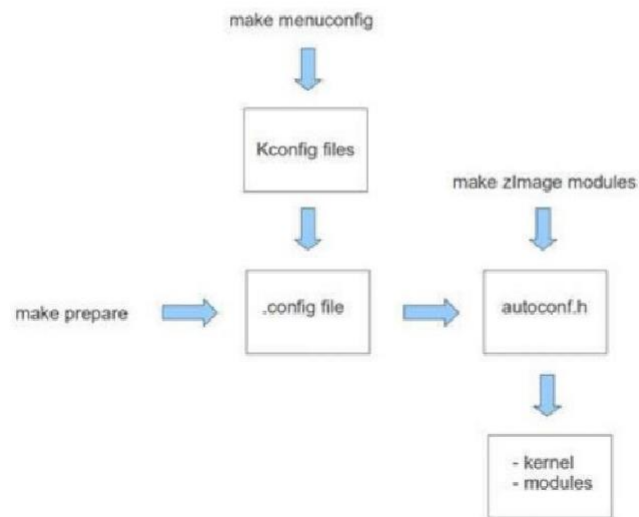


FIG 1.6 BUILD SYSTEM

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 study, after completion of the starting phase, we proposed a method to analyse the performance of any hardware. This lets you know how hardware works. We used beaglebone black and raspberry pi for demonstration purposes. The analysis of output is reported by the collection and the results can be seen in a chart. The chart is shown with the RRD method.

1.4 SCOPE OF THE PROJECT

Developing our own Linux distro has several benefits, as only the necessary library files can be configured. The whole Linux distro is what we were looking for. We cannot get rid of redundant libraries that have more room and if we make some improvements, some mistakes might even occur. The need to build our own Linux Distro. By designing and analysing our own Linux distro

Hardware output helps us understand its behaviour. To facilitate this, we demonstrate with the aid of rrd tool the final result in a graph.

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

Shahid H. Bokhari (1995) Linux is an OS that is running on Intel-386/486/Pentium compatible IBM PC machines and processors equivalent. It was published at Helsinki University of Technology in Finland by Linus Torvalds, a graduate student. It does not contain proprietary code because it was written from scratch. It retains the copyright but allows the free release of the source code (see sidebar for "The Linux Software License"). In addition to Torvalds, the internet has been used to create an enthusiastic, worldwide volunteer community. Just a few years ago, Linux was being developed. The initial versions were unstable, and only diehard hackers were interesting. Linux has matured now, however, and can be used seriously. Many of the powerful compilers and tools created in the framework of its GNU (GNU is not Unix) project by the Free Software Foundation can run on Linux. For Linux in an Intel-386/486 version called XFree86, the de-facto standard of MIT's X Window System for windows with Uniz is also available. Therefore Linux is a full-fledged operating system which offers all the capabilities that Unix commercial systems generally offer. The C, C++, Pascal, Modula-2 and -3 compilers are excellent, Oberon, Smalltalk, Fortran. Many versions of Editor including Emacs and vi are available. The full support is the security of passwords, file encryption, multiple logins, virtual memory and multitasking. Linux networking has grown to support vast networks in their remote, remote and electronic mail connections. Linux was created with aversion from the Network File System (NFS). This allows file systems to be shared between several computers, which reduces use of hard disc and minimises overhead device administration. The Text processing framework TeX/LaTeX was ported to Linux with ghostview and xdvi previews and drawing packages (xfig and idraw). Both these features are free of charge and free of charge.

Ricardo P. Jasinski (2012), The use of an OS to facilitate software creation and liberate programmers from the use of low-level hardware and to facilitate programming of common tasks is advocated.

The high level of complexity of modern desktop computers needs an OS;

On the other hand, embedded systems are limited, typically cost-effective and power-resistant, architectures. Due to further requests from an OS, embedded developers are confronted with an important decision to decide whether or not to implement an OS. This article presents a case study that includes three separate scripts for developing a sample application (embedded weather station): using the μ C/Ost-II real-time OS without any OS, and using the general uClinux OS. An FPGA and SoPC provided a versatile hardware interface for all three configurations. By adopting an OS, development time was cut to 48%, while programme storage requirements were increased by at least 71%.

Zhongjiang Zhang (2015), Currently, because of its unique benefits such as open source and stability, Linux operating system has become an investigative subject in the field of embedded system. In this scenario, this article offers an idea for a transplantation of the navigation software from VxWorks to Linux. This paper therefore focuses on a study of the Linux and kernel start-up process based on the x86 platform, in order to achieve the goal described above. Then a full development environment and build an embedded Linux operating system will be resolved in order to satisfy the application requirements. In this article, it is recommended to back up and restore the embedded Linux using the Clonezilla tool.

Abhimanyu Sharma (2016) The Open Source Software (OSS) is an end user licence computer software with its source code. Agreement (EULA) in which the copyright holder grants every individual and for any reason the right to examine, modify and distributing the software. The public should build Open-Source software in cooperation. The most popular example of open-source development is open-source software. The open-source or collaborative competitive development from many different sources creates a wider range of design perspectives than any organisation is capable of long-term development and support. The Linux kernel is an operating system kernel used by the Unix-like Linux family. It is one of the leading examples of open source and free programming. Linux is the easiest Conditions, a method of operations. It is the computer programme that allows applications and computer operators to have access to the computers they are looking for. The OS relays instructions to the computer, for example, from an

application.

Hari Narayanan, (2017) Protection of the operating system has been a hot spot of research for decades. Several protection measures to protect the operating system have been implemented so far. We concentrate on securing Linux operating systems in this article. Though Linux is open-source and many people work with kernel patches for security troubles, many malware still exploit the current vulnerabilities. We aim to reduce damage by malware with our architecture, if we don't block it entirely. Our architecture is aimed at guaranteeing the least privilege principle. The least privileged principle ensures that a mechanism has sufficient rights to carry out its mission. This means that the system will do the least harm, even though the mechanism is compromised, since it runs in a sandbox. The biggest part of our framework runs at the consumer level to make the distribution portable. Our system uses a specially designed security ticket to give the user processes finely licenced which in conventional linux architecture is not currently possible.

The safety ticket is structured so that a child process can be inherited, shared and unforgeable. Secd, which authorises all requests and handles security tickets too, is the central module in the framework.

Marko Boras, Josip Balen, Kresimir Vdovjak, (2020), The consistency, reliability, and efficiency of the operating system is also a very important aspect in the context of software assessment[4].

However, since it requires managed environments and same conditions for all tested systems, it is a complex and time-consuming process, which remains a little challenging, in particular in the operating systems field. Several aspects of the performance of an operating system on personal computers were examined in our previous work. The full performance assessment of three versions of the Windows Betriebssystem: Windows XP, Windows [5], focused on CPU scheduling, drive management, graphic subsystem management, memory and Network Performance. It was done Vista and Windows 7. In two separate conditions measurements were also carried out. In the majority of tests performed on the low-end computer device Windows XP was higher than Windows Vista and Windows 7 for its efficiency measurement results. In addition, newer operating systems have shown increased performance in the fields of subsystems graphics and memory

management on a high-end computer system. Furthermore, a model of performance assessment has been developed and similar changes are to be made to Linux-based operating systems in this paper. In [6], the Linux Debian operating system evaluated three popular virtual private server hosting systems. In[7], the authors tested three versions of Windows, namely Windows XP, Windows Vista and Windows, in their networking performance. Networking efficiency research has continued and extended [8] in which the evaluation was carried out of three recent Windows systems, Windows 7, Windows 8.1 and Windows 10. Experimental findings indicate that Windows 10 has improved network management, delays have been decreased but use of CPUs has been increased.

I recognised the Linux operating system from the literature examined, which has been updated annually.

CHAPTER 3

METHODOLOGIES

3.1 GENERAL

We must first create our own customised distribution by carrying out a small study. These research should include the method to design the distribution we will follow, which will allow us to follow a certain pattern of growth, and all characteristics and requirements of the distribution should then be included in the research. By choosing this, we can incorporate it into the design process and eventually deliver our customised distribution that meets our requirements.

I have experienced two creation processes in my research:

1. SP V 2. Yocto

I notice that LFS has been launched early on so it can support the older versions as much as possible. I therefore chose yocto for my distribution growth. The Yocto Project is a project for the creation of Linux distributions by the Linux Foundation which is collaborative and open source. What is critical for yocto developers is that they update yocto frequently in the newer versions. The steps in your official web site are listed in detail. The next important step is the use of the collected daemon after each distribution is developed to check its performance.

3.1.1 LINUX FROM SCRATCH

Scratch Linux (LFS) is a process by which all its components can be manually constructed to install a functioning Linux system. Of course, this is longer than installing a Linux distribution previously compiled. The benefits of this approach are a lightweight, scalable and stable framework and a better understanding of Linux-based operating systems' internal operating systems, says Linux from Scratch. The Scratch Linux Project is like BitBake, which supports Linux cross-compiling for embedded ARM systems like the Raspberry Pi and BeagleBone.

REQUIREMENTS AND PROCEDURE

A clean partition and a working Linux framework must be built with a compiler and some important software libraries. You may also use a Live CD to set up an LFS system rather than install an existing system installed for Linux. Initially, a compilation must be made of the LFS compilation tools, such as GCC, glibc, binutils and other necessary tools. The `/usr` directory must then be changed to the toolchain partition by chroot to initiate the construction of the final structure. Glibc is a first package to build in gcc; after that the connector of the toolchain must be changed so that all other packages which make up the completed framework that can be linked against the newly developed glibc are linked to it. The bash hacking function is switched off and the `bin` directory of the temporary toolchain is moved to the end of `PATH` during the chroot process. The newly compiled programmes that are first developed in `PATH` and the new framework are therefore based on their own new components.

STANDARD BUILT UNIT(SBU)

The standard build unit represents the time taken to create the first package in US on the provided machine during the initial set-up or the device. This was driven by the length of time it took to create an US system and by the desire of many users to know 'how' long a source tarball would take in advance.

The first package developed by the user is GNU binutils from Linux from the Scratch session 6.3. Users are invited to test construction processes using shell constructs when constructing the Linux kernel using US and dub this time the standard system build device.

The GNU C library (rated at 4.1 SH As) and GNU Compiler Collection have so many packages which are constructed during compilation that they take a lot longer than binutils (rated at 8.4 SH Us). This unit should be understood as an approximation: different factors affect the actual time needed to construct a package.

KERNEL DISTORTION

In this project, we use Fedora Core 33. We can use a 5.x kernel for any distribution but be mindful that this may lead to discrepancies in the build process. Some distributions may use older and incompatible releases to bundle utilities than is necessary in this course.

CLEAN INSTALL

We must clean the build system installation. Any virtualization environment can be finished, as long as we have access to the console at different stages. Real hardware is also acceptable if we can readily find such tools.

DEVELOPMENT TOOLS

GCC, binutils, and other packages from the package group "development tools" must be installed. This is something we can choose during installation or install with the group install option for yum or dnf. The kit's names can differ depending on the delivery, but in general, we will need to install groups of "development instruments" and "C development tools."

3.1.2 YOCTO

Via the Free Embedded Build system, the Yocto Project provides an open source development environment for a variety of platforms, including x86-64, MIPS, PowerPC, and x86 architectures. To develop, build, debug, simulate, and test Yocto Project components, the entire software stack with Linux, the X Windows System, GTK+ framework systems, and Qt frameworks can be used.

- Yocto includes a recent Linux kernel and an embedded environment collection of system commands and libraries.

It also allows components such as X11, GTK+, Qt, Clutter and SDL (including) to create a rich user experience on display-hardware devices. These components do not need to be assembled if you do not have a monitor or choose to use alternative UI framing.

- Creates an open embedded core that's oriented and secure, compliant with the OpenEmbedded project.

- It supports completely a wide variety of EMUlator hardware and system emulation (OEMU).
- It offers a layer structure to conveniently enlarge, customise and maintain the framework.

We can make images for many types of devices with the Yocto Project. With Yocto Project, we can boot and emulate using OEMU for the development of reference images. The standard machines aim OEMU complete device emulation of x86, ARM, MIPS and PowerPC versions for 32-bit and 64-bit. Over and above emulation, we can extend support for almost every platform Linux can run and a toolchain can aim through the layer mechanism.

RREREQUISITES FOR THE DEVELORUENT MACHINE:

- At least 4 - 6 GB RAM.
- Recent Ubuntu OS (20.04 LTS in this case).
- There is a need of atleast 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.

3.1.3 BUILDING IMAGES

- •Create and run a qemu86 reference image in the OEMU emulator.
- Adjust configurations easily so that you can generate a second picture for MinnowBoard MAX-compatible boards.
- *Be Sure our Build Host is Set Up:* The steps to build an image is depend on our build host being properly set up.
- *Check Out our Branch:* Be sure you are in the Source Directory, fro example poky, which check out the branch associated with the latest Yocto Project Release:

```
$ cd /poky
```

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

Git's checkout command will check 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 our own branch based on the released branch ensures we are using the latest files for that release.

Initialize the Build Environment: The `oe-init-build-env` environment setting script will be executed to identify your build host with the OpenEmbedded environment.

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

This, among other things, creates the Build Directory, which is created and placed in the source directory in this case. After the script runs, our working directory is now set to the Build Directory. When the building is finished, the Build Directory contains all of the files created during the build.

Examine our Local Configuration File: A local configuration file called `loc.conf` will be available in the `conf` subdirectory of the project directory when you set up a build setting. You should look at this file before you use BitBake when beginning the building process and ensure that your general settings are as follows:

You should add a declaration to your project's configuration file, which in this case is `poky/build/conf/local.conf`, to help save disk space during builds. After the recipe has been created, this will delete the directory that was used to build it.

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

- The target build machine is Qemux86, which generates a picture that can be used on an Intel® 32 bit architecture using the QEMU emulator. This default is easily modified by the `MACHINE` variable to quickly construct an image for another machine in this example.
- The package manager used to create a picture is another factor before we design. The RPM package manager is chosen for the default `local.conf` file. The `PACKAGE_CLASSES` variable allows us to manage this configuration.

Start the Build: 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, we can start QEMU and run the image:

```
$ runqemu qemux86
```

Exit QEMU: This is 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 periodically collects system and application performance metrics and provides mechanisms for the saving of values in various ways, such as RRD files. Collects and preserves information from or across the network, collects metrics from different sources, such as operating systems, software, logfiles and external devices. These statistics can be used for the surveillance of systems, the detection of performance bottlenecks, and prediction of future loading of the system (i.e. capacity planning).

3.2.1 LIMITATIONS

You do not make graphics but RRD files. Cannot generate the graphs from these files. But there's a tiny sample script in the contribution. The monitoring function has been added in version 4.3 but is so far limited to simple threshold controls.

3.3 RRD TOOL

RRDtool is the industry standard for open-source, high-performance time series data logging and graphics systems. In shell, perl, python, ruby, lua and tcl applications RRDtool is easy to integrate into.

It is very simple to collect data on all kinds of items, from our office's temperature to the amount of octets passed through our router's FDDI interface. However, the effective and systematic storage of this data is not so straightforward. This is where RRDtool is useful. It allows us to log and review the information you collect from all types (DS). This tool's data analysis is focused on the capability of generating graphical representations of the data values obtained over a specified period of time.

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

3.3.1 WORKING OF RRD TOOL

DATA ACQUISITION

It is convenient to have the data accessible at constant time intervals when tracking the status of a device. Sadly, at the exact moment you like, we can not always be able to retrieve data. RRDtool allows us, therefore, to refresh the log file whenever you want. The value of the data source (DS) at the latest official time-slot (interval) shall be automatically interpolated and the interpolated value shall be added to the log. The original value we have given is also stored and considered when the next log entry is interpellated.

CONSOLIDATION

The log data at a time interval of 1 minute, but the creation of the information over the last year may also be of interest to you. We can do that only by storing the data for

the entire year in 1 minute intervals. It would take a lot of discretion to interpret the 24

data when you tried to build an entire year-round graph. It would also take a lot of time. When establishing a Round Robin Database (RRD), the interval of consolidation is defined and the average, minimum, maximum and last consolidation function (CF) is used to establish the consolidated values. There are various configurations within one RRD that we can describe. Both of these are kept on flying as new information is loaded into the RRD.

ROUND ROBIN ARCHIVES

In the Round Robin Archives, consolidation value is stored (RRA). It is a very effective way, when using a known consistently large amount of storage space, to store data for a while.

RRDtool will reserve space for 1'000 data and a headerspace to store 1,000 values in the 5-minute interval. It stores a pointer in the header to say which slots were last written to in the storage field. This limits the history to the last 1'000 values automatically. We can set a different RRA for a storage of about 750 data values at 2 hour intervals since we can identify multiple RRAs in one single RRD.

The use of RRAs ensures that the RRD is not expanded over time and that old data is discarded automatically. With the consolidation function, data can be retained very long, and the data resolution along the time axis can be progressively reduced.

Through the use of various CF functions, we can store precisely the sort of information we really need.

At a constant interval, the RRD stores data. Often, when a value needs to be written to the RRD, no new data is available. By saving an uncertain value in a data base, we can manage these circumstances with RRD tool. Both functions of the tool endorse the unknown value. Data consolidation

In collection, a new unified value is ready to be entered into the Round Robin archive the unknown values are taken into account (RRA). A validity check is carried out to verify that the unknown data point percentage is greater than a configurable number. Otherwise, the RRA is given an undefined value.

GRAPHING

The RRD tool helps us to produce reports on the basis of data stored in one or more RRDs in numerical and graphical form. The graphic function can be completely configured. The graph can be described freely in the size, colour, and contents.

To function with RRDtool graphs, data should be collected using one or more statements on data classification. The collection of data from two or more databases is not limited to one single database, it is completely lawful.

Then the data from the Round Robin Archives are consolidated to provide an exact pixel-by-pixel data point. If we do not take care, the RRD tool can, if possible, extend the range slightly.

Data is often not exactly in the format that we want to view. For example, we can collect bytes per second, but want bits per second displayed. This is the intent of the data calculation command. After the data is consolidated, a copy is made and a powerful RPN command set is used for modification of this copy.

We will graph it when we have the data fetched and processed. This finishes the series of the rrdtool graph.

z6

CHAPTER 4

RESULT

The kernel image should be built in order to set up our own linux distro for measuring the performance analysis. For building image we can do it in any kind of distributions, such as ubuntu, fedora or openSUSE.

The medium in which we built can also defer such as LFS or Yocto.

LFS supports the older version and showing error with the most updated version.

Even yocto supports till the next lesser version to the updated one.

Thus I set up the yocto in openSUSE 15.1, this version is the next lesser one to the updated one. Below image is the built image in openSUSE.



```
osboxes@linux-qtck:~/poky/build/tmp/deploy/images/qemux86-64> ls
bzImage
bzImage--5.8.18+git0+b976de4f41_3c5d210805-r0-qemux86-64-20210308112141.bin
bzImage-qemux86-64.bin
core-image-sato-qemux86-64-20210308112141.qemuboot.conf
core-image-sato-qemux86-64-20210308112141.rootfs.ext4
core-image-sato-qemux86-64-20210308112141.rootfs.manifest
core-image-sato-qemux86-64-20210308112141.rootfs.tar.bz2
core-image-sato-qemux86-64-20210308112141.testdata.json
core-image-sato-qemux86-64.ext4
core-image-sato-qemux86-64.manifest
core-image-sato-qemux86-64.qemuboot.conf
core-image-sato-qemux86-64.tar.bz2
core-image-sato-qemux86-64.testdata.json
modules--5.8.18+git0+b976de4f41_3c5d210805-r0-qemux86-64-20210308112141.tgz
modules-qemux86-64.tgz
osboxes@linux-qtck:~/poky/build/tmp/deploy/images/qemux86-64> █
```


The below image shows how the collectd collects data of different parameters. The parameters can be cpu, memory, load etc. These parameters are then represented by graph with the help of rrd tool

```

1612278987.544,1353885184.000000
1612278997.545,1392462432.000000
1612279007.545,1392462432.000000
1612279017.543,1392600480.000000
1612279027.544,1392594944.000000
1612279037.543,1384337408.000000
1612279047.557,1388999168.000000
1612279057.552,1381515264.000000
1612279067.545,1381548032.000000
1612279077.563,1381580800.000000
1612279087.551,1381580800.000000
nanish@nanish: /var/lib/collectd/csv/nanish/memory$ cat memory-free-2021-02-02
epoch,value
1612278987.544,1363529728.000000
1612278997.545,1382395904.000000
1612279007.543,1390200704.000000
1612279017.543,1393651712.000000
1612279027.543,139360096.000000
1612279037.543,1393311744.000000
1612279047.543,1393853696.000000
1612279057.543,1390981120.000000
1612279067.543,1364176000.000000
1612279077.544,1363885184.000000
1612279087.545,1392462432.000000
1612279097.545,1392462432.000000
1612279107.543,1392600480.000000
1612279117.544,1392594944.000000
1612279127.543,1384337408.000000
1612279137.557,1388999168.000000
1612279147.552,1381515264.000000
1612279157.545,1381548032.000000
1612279167.563,1381580800.000000
1612279177.551,1381580800.000000
1612279187.551,1381580800.000000
nanish@nanish: /var/lib/collectd/csv/nanish/memory$

```

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

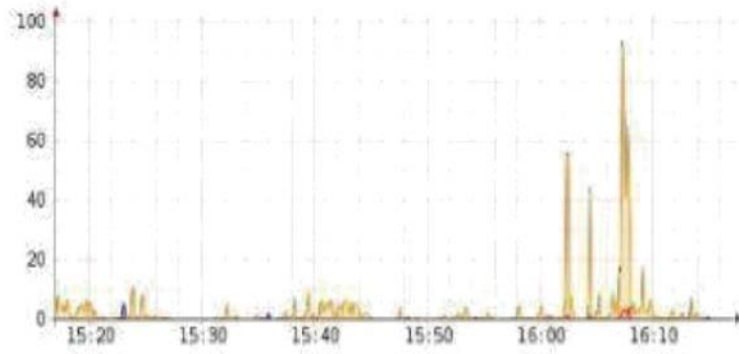
1. Epoch
2. Value

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

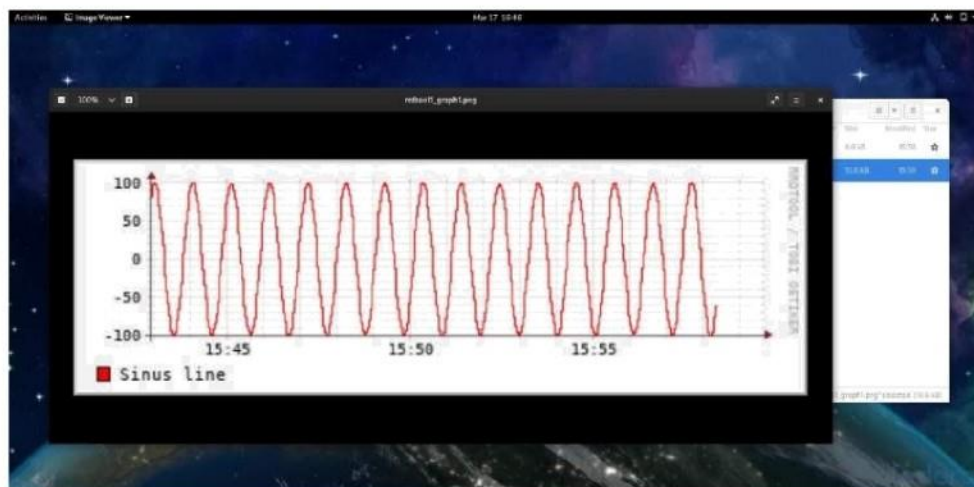
```

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

```



Below is the graph which I got it for the database file. In this, the x axis denotes the time and the y axis denotes the average. This graph is represented as sine wave.



CHAPTER 5

CONCLUSION

From this project it can be concluded that any Linux distribution can be used, it doesn't concern in checking the performance of the hardware. For understanding the performance analysis in detail, a tool is used which is called as collectd. Collectd is used to collect all the data and can be customized according to our need. The data can be cpu-info, memory, load, etc. These data can be represented in a graph. A tool named rrd tool is used for representing the data in graph. Finally the data will be represented in graph and the final output is in the form of sine wave. Sine wave will be quiet easy to understand the performance analysis similar to the one that can be observed in task manager which is in our own laptops.

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(P DF).
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/>

15. <https://docs.yoctoproject.org/ref-manual/system-requirements.html#opensuse-packages>
16. <https://docs.yoctoproject.org/brief-yoctoprojectqs/index.html>
17. <https://docs.yoctoproject.org/dev-manual/start.html#setting-up-to-use-windows-subsystem-for-linux-wsl2>
18. <https://blog.tinned-software.net/understand-the-basics-of-rrdtool-to-create-a-simple-graph/>

CUSTOM LINUX DISTRO FOR PERFORMANCE ANALYSIS USING LINUX FROM SCRATCH (LFS)

ORIGINALITY REPORT

9 %
SIMILARITY INDEX

6 %
INTERNET SOURCES

2 %
PUBLICATIONS

6 %
STUDENT PAPERS

PRIMARY SOURCES

1	unh.edu Internet Source	< 1 %
2	Submitted to Karunya University Student Paper	< 1 %
3	Submitted to SIM Global Education Student Paper	< 1 %
4	www.ijser.org Internet Source	< 1 %
5	Submitted to Amity University Student Paper	< 1 %
6	Gene Sally. "Pro Linux Embedded Systems", Springer Science and Business Media LLC, 2010 Publication	< 1 %
7	www.eukhost.com Internet Source	< 1 %
8	Submitted to Mzumbe University Student Paper	< 1 %

9	www.tandfonline.com Internet Source	<1%
10	Submitted to Universiti Putra Malaysia Student Paper	<1%
11	Submitted to Segi University College Student Paper	<1%
12	Submitted to City and Islington College, London Student Paper	<1%
13	www.intechopen.com Internet Source	<1%
14	Submitted to KIIT University Student Paper	<1%
15	Submitted to University of Hertfordshire Student Paper	<1%
16	www.guru99.com Internet Source	<1%
17	Submitted to Study Group Australia Student Paper	<1%
18	Submitted to Sunway Education Group Student Paper	<1%
19	www.ipcsit.com Internet Source	<1%
20	www.docstoc.com	

<1%

21

Christopher Negus. "Linux® Bible", Wiley, 2015

Publication

<1%

22

Submitted to Shinas College of Technology

Student Paper

<1%

23

Submitted to Sim University

Student Paper

<1%

24

Doug Abbott. "Integrated build environments", Elsevier BV, 2018

Publication

<1%

25

linuxhint.com

Internet Source

<1%

26

Doug Abbott. "Introducing linux", Elsevier BV, 2018

Publication

<1%

27

Submitted to National College of Ireland

Student Paper

<1%

28

Submitted to Bannari Amman Institute of Technology

Student Paper

<1%

29

Submitted to Nizwa College of Technology

Student Paper

<1%

Exclude quotes On

Exclude bibliography On

Exclude matches

< 14 words