Ex-2 KERNEL CONFIGURATION, COMPILATION AND INSTALLATION

Date: 25.08.20

Aim:

To study and implement the kernel configuration, compilation and installation.

Description:

The Linux Kernel :

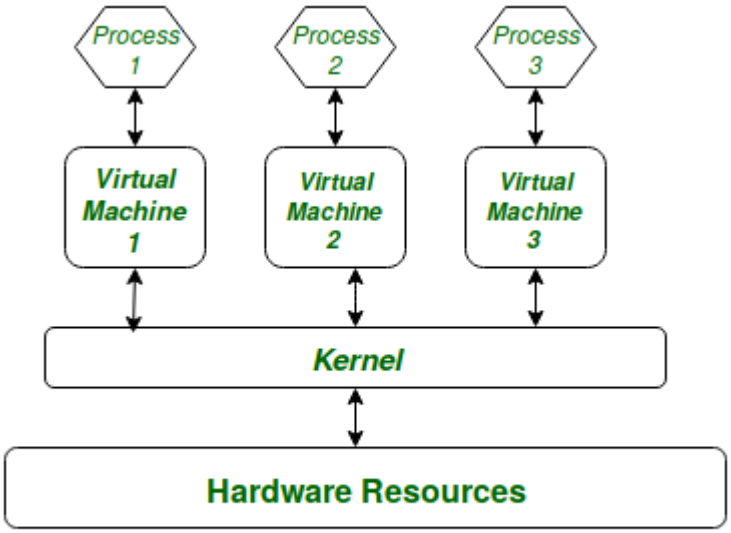
The main purpose of a computer is to run a predefined sequence of instructions, known as a program. A program under execution is often referred to as a process. Now, most special purpose computers are meant to run a single process, but in a sophisticated system such a general purpose computer, are intended to run many processes simultaneously. Any kind of process requires hardware resources such are Memory, Processor time, Storage space, etc.

In a General Purpose Computer running many processes simultaneously, we need a middle layer to manage the distribution of the hardware resources of the computer efficiently and fairly among all the various processes running on the computer. This middle layer is referred to as the kernel. Basically the kernel virtualizes the common hardware resources of the computer to provide each process with its own virtual resources. This makes the process seem as it is the sole process running on the machine. The kernel is also responsible for preventing and mitigating conflicts between different processes.

The Core Subsystems of the Linux Kernel are as follows:

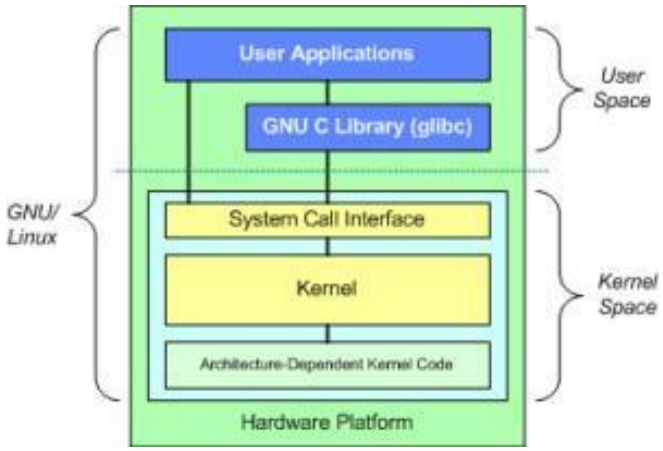
1. The Process Scheduler
2. The Memory Management Unit (MMU)
3. The Virtual File System (VFS)
4. The Networking Unit
5. Inter-Process Communication Unit

This schematically represented below:



Architecture of system kernel :

We can think of Linux Kernel architecture to be divided into two levels – User Space and Kernel Space.



At the top is the user space. Below the user space is the kernel space. Here, the Linux kernel exists.

User Space:

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

Kernel Space:

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

Sl. No.	**Command Name**	**Meaning**	**Description**
1.	**rpm -qa kernel-devel**	It displays the version of the kernel.	Kernel-devel - This package provides kernel headers and makes files sufficient to build modules against the kernel package.
2.	**uname –r**	uname displays the information about the system.	The command ‘uname‘ displays the information about the system. option : **-a** It prints all the system information in the following order: Kernel name, network node hostname, kernel release date, kernel version, machine hardware name, hardware platform, operating system **-s** It prints the kernel name.

			-n It prints the hostname of the network node **-r** It prints the kernel release date **-v** It prints the version of the current kernel
3.	**tar**	tar' stands for tape archive, is used to create Archive and extract the Archive files	tar command in Linux is one of the important commands which provides archiving functionality in Linux. We can use Linux tar command to create compressed or uncompressed Archive files and also maintain and modify them. **Options:** **-c :** Creates Archive **-x :** Extract the archive **-f :** creates archive with given filename **-t :** displays or lists files in archive file **-u :** archives and adds to an existing archive file **-v :** Displays Verbose Information **-A :** Concatenates the archive files **-z :** zip, tells tar command that create tar file using gzip **-j :** filter archive tar file using tbzip **-W :** Verify a archive file **-r :** update or add file or directory in already existed .tar file
4.	**ln**	A symbolic link, also known as a symlink or soft link, is a special type of file that points to another file or directory.	There are two types of links in Linux/UNIX systems: 1. Hard links 2. Soft links
5.	**make**	utility for building and maintaining groups of programs.	The purpose of the make utility is to determine automatically which pieces of a large program need to be recompiled, and issue the commands to recompile them. you can use make with any programming language whose compiler can be run with a shell command. In

			fact, make is not limited to programs. You can use it to describe any task where some files must be updated automatically from others whenever the others change. Options : **-b,-m** prints online help and exitThese options are ignored for compatibility with other versions of make. **-B** Unconditionally make all targets **-d** Print debugging information in addition to normal processing **-e** Give variables taken from the environment precedence over variables from makefiles **-k** Continue as much as possible after an error

Exercise

Configure and compilation the kernel

Step 1 :

Download the latest kernel source from www.kernel.org or from a repository.

Step 2 :

Check the current kernel version and name of the kernel.

step 3 :

Move the module from downloads to /usr/src and uzip the file.

step 4 :

Make a systemlink to the existing kernel and clean the existing kernel.

step 5 :

Building kernel and its modules.

step 6 :

Check the current kernel version and name of the kernel.

Output :

Kernel Version

```
kali@kali:~$ cat /proc/version
Linux version 5.7.0-kali1-amd64 (devel@kali.org) (gcc version 9.3.0 (Debian 9.3.0-14), GNU
 ld (GNU Binutils for Debian) 2.34) #1 SMP Debian 5.7.6-1kali2 (2020-07-01)
kali@kali:~$ uname -r
5.7.0-kali1-amd64
```

Kernel Name

```
kali@kali:~$ cat /proc/version
Linux version 5.7.0-kali1-amd64 (devel@kali.org) (gcc version 9.3.0 (Debian 9.3.0-14), GNU
 ld (GNU Binutils for Debian) 2.34) #1 SMP Debian 5.7.6-1kali2 (2020-07-01)
kali@kali:~$ uname -r
5.7.0-kali1-amd64
```

Moving the kernel to /usr/src

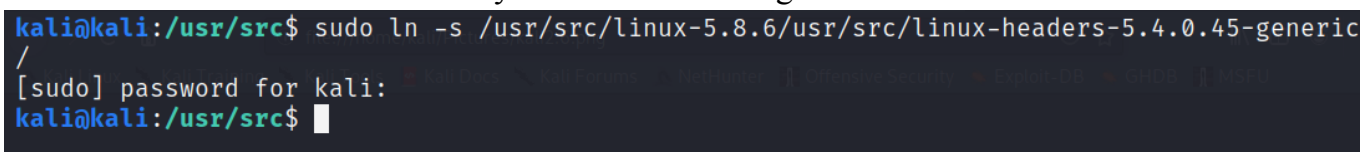
```
kali@kali:~$ cd Downloads
kali@kali:~/Downloads$ mv linux-5.8.6 /usr/src
```

```
kali@kali:~$ cd /usr/src
kali@kali:/usr/src$ ls
linux        linux-headers-5.7.0-kali1-amd64   linux-kbuild-5.7
linux-5.8.6  linux-headers-5.7.0-kali1-common  virtualbox-guest-6.1.12
```

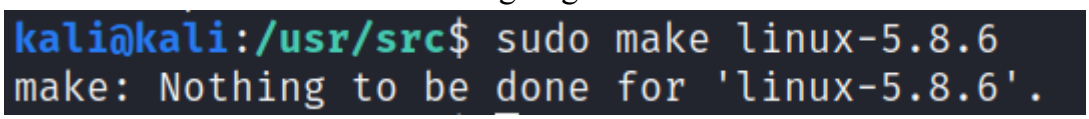
Cleaning the kernel using make

```
kali@kali:/usr/src$ sudo make clean
[sudo] password for kali:
```

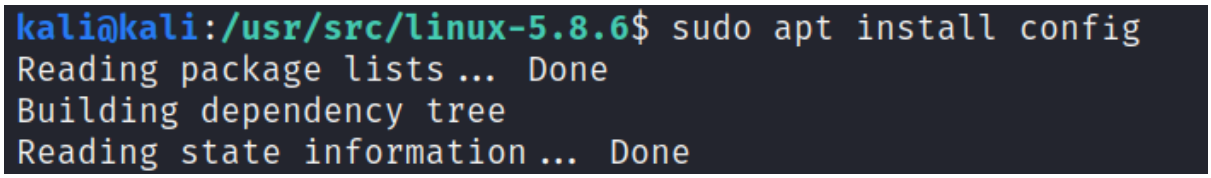
System link to existing kernel



Making target files



Installing Kernel config



Results:

The study and implement the kernel configuration, compilation and installation is studied and executed.

Video : [URK17CS027 EXP-2 LINK]