**“Adding A System Call To The Linux Kernel”**

Prabhakar Srivastava

Department of Computer Science and Engineering

Amity University Jharkhand

Jharkhand 834001, India

Email: [prabhakarsinha12345.kay@gmail.com](mailto:prabhakarsinha12345.kay@gmail.com)

ABSTRACT

*The goal of this project is to download the kernel source using the git version control software and build a new branch based on specific version of Linux and then adding a new system call in it.*

*The Linux kernel services are made available to the application layer through so-called system calls. All system calls collectively make up the kernel's application programming interface (API).*

*System calls therefore provide insight into the development process and outline decisions made to the Linux kernel.*

*Following modifications done on this project:*

1. *Examining and compiling Linux source code. Installation of modified kernel.*
2. *Running the modified kernel.*
3. *User/kernel space communication: system calls*
4. INTRODUCTION

In computing, a programmatic way for a computer program to request a service from the kernel of the running operating system is known as System calls. System calls are the means by which programs interconnect with the operating system. Computer programs make system calls when they make requests to the operating system kernel. System calls provide services to user programs from operating system using an application program interface (API). Provides an alliance between the os and the processes, permitting user-level processes to request os services. The only possible way of entering into the kernel system is possible using the System calls.

All programs that require resources ought to use system calls.

System calls provides following services:

* administration and process making
* managing memory
* file accessing, list and management of system
* handling the device i.e., (I/O)
* prevention
* networking and many more

System Calls types: 5 different types of system calls are as follows–

* Process authorization/control: end, abort, create, terminate, allocate and free memory.
* File monitoring: create, open, close, delete, read file etc.
* Device monitoring
* Information care
* Communications

**Keywords:** Linux, Kernel, git version, System calls

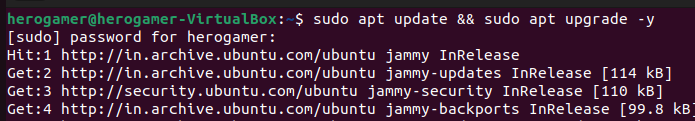
1. PREQUISITES

There are some perquisites required for adding the system call:

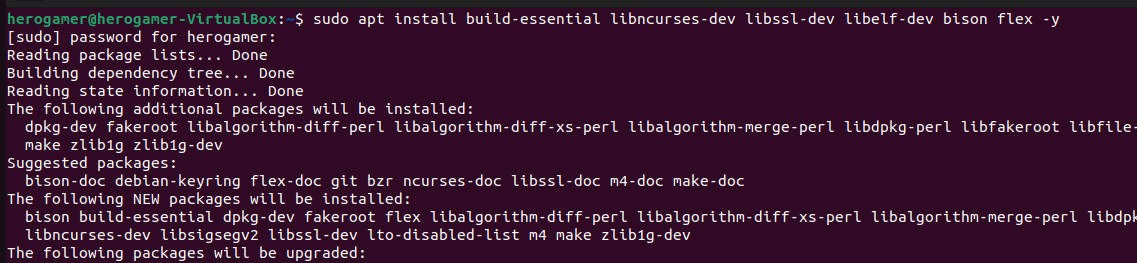
1. Kernel Source code is required from kernel.org.
2. Accessing the root directory using the sudo –s command.
3. Installation of required packages.
4. Permission of altering files in root directory.
5. Extraction of downloaded kernel in the usr/src folder.
6. ABOUT ROOT DIRECTORY AND THE PATH

The Linux OS has a root directory just like other OS. To get access in the root directory sudo –scommand is used. After reaching the root directory, the update and installation process of the kernel can be done. So in this paper, the latest Linux kernel is extracted in the usr/src path. After extracting it. The Linux folder will be made. Inside that folder again a folder is made for the Makefile and the C file of the system call. The Linux folder itself have a Makefile file, so it is also need to be updated. After doing this, there is a folder named #arch/x86/entry/syscalls/syscall\_64.tbl in the Linux folder. This also needs to be modified. #include/linux/syscalls.hthis also needs to be modified for the appropriate function. In this paper, I have included all the modification and creation one by one of the root directory and path. So, the command in which **‘#’** is before the command that means those commands are executed in the root directory and those command in which **‘$’** is before the command that means those command aren’t executed in the root directory, they don’t need root access.

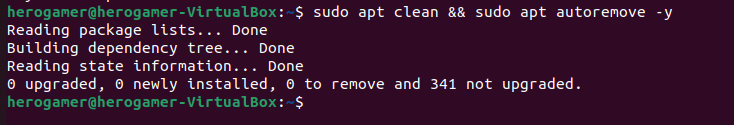
1. CREATION
2. Fully updating the Operating System - $sudo apt update && sudo apt upgrade



1. Installation of package **-** $sudo apt install build-essential libncurses-dev libssl-dev libelf-dev bison flex



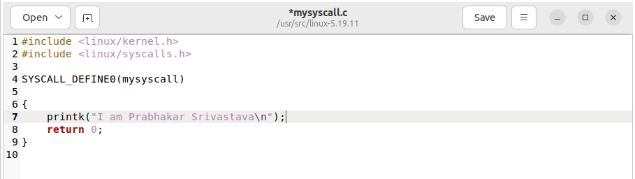
1. Cleaning the installed package **-**$sudo apt clean && sudo apt autoremove –y



1. Creation of home directory **-** #mkdir mysyscall



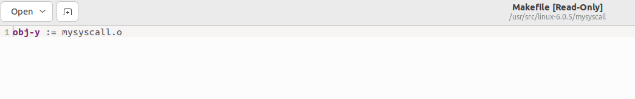
1. Creation of system call using C file in the home directory **-** #gedit mysyscall.c



Save and exit.

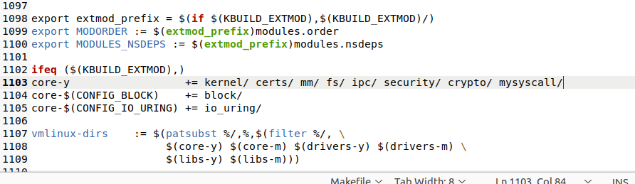
1. Creation of Makefile in the home directory **-** #cd mysyscall

**#**gedit Makefile



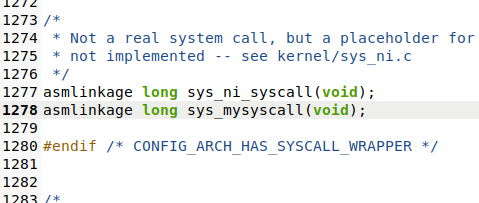
Save and exit.

1. Updation of Linux Makefile **-** #gedit Makefile



Save and exit.

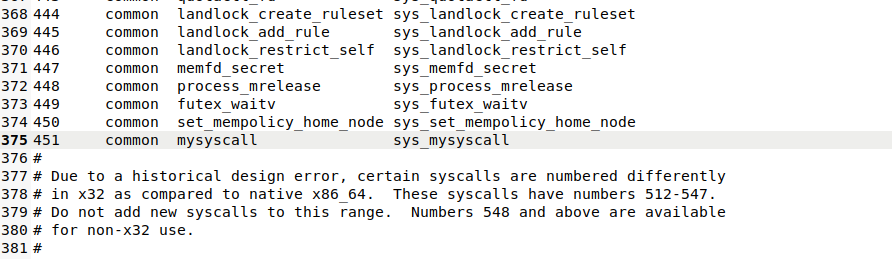
1. Adding appropriate function prototype for the created system call **- #**gedit syscalls.h

****

Go to the end and write the following code just above **#**endif

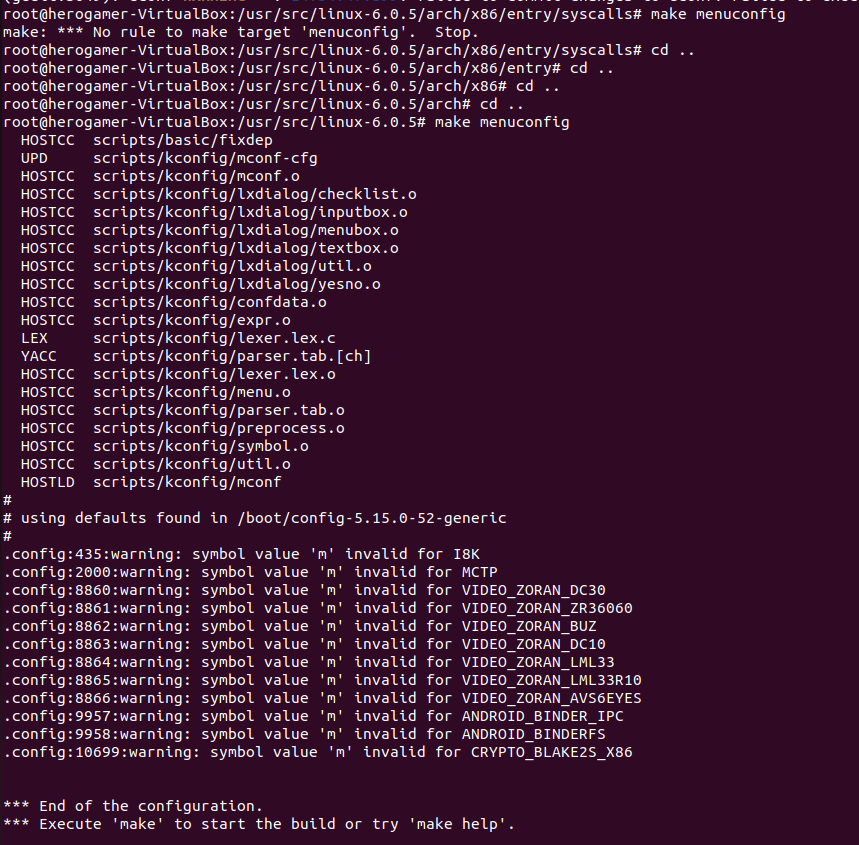
Save and exit.

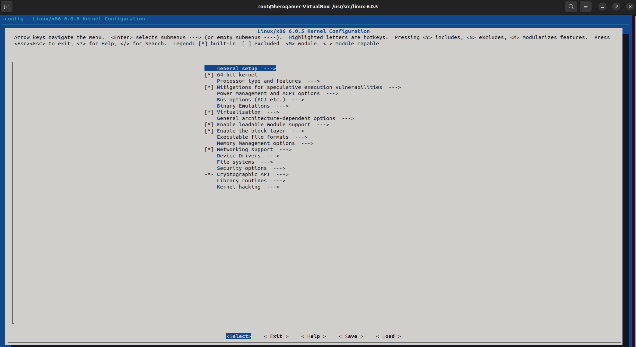
1. Updation of the syscalls\_64.tbl **-** #gedit syscalls\_64.tbl

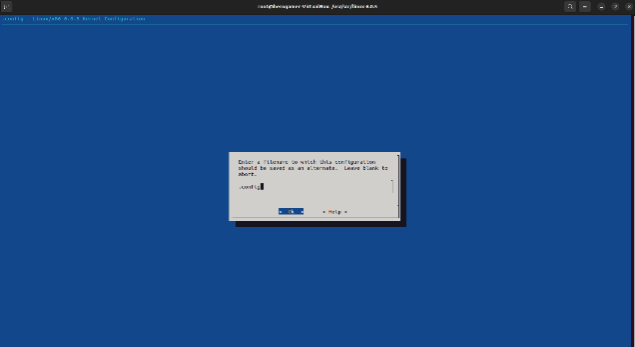
****

Save and exit.

1. IMPLEMENTATION
2. Configuration of the kernel **-** #make menuconfig

****

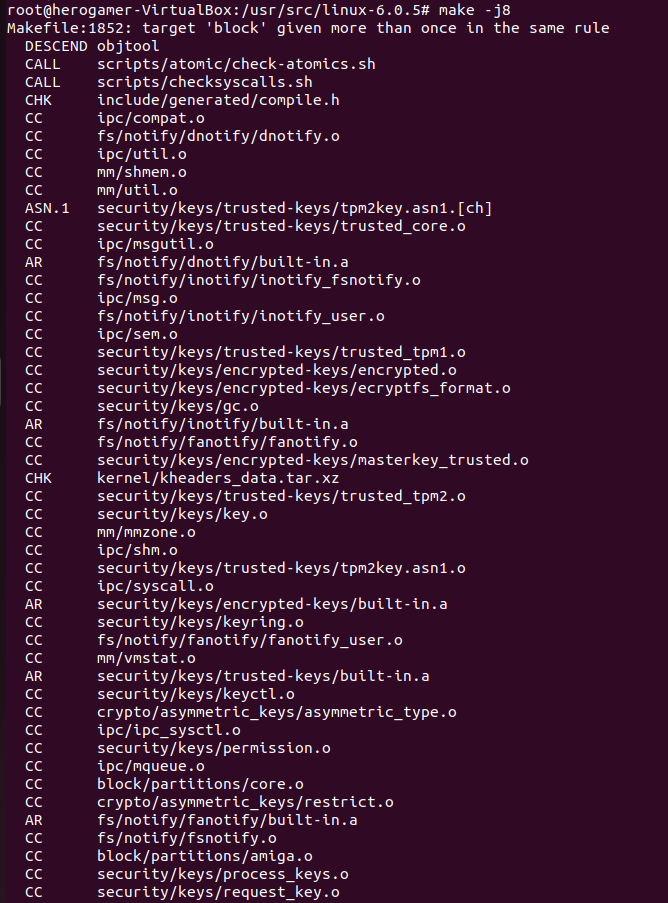




Use the **Tab** key to toggle between options. Do not make any changes to keep the default settings.

1. Compilation of the kernel’s source code **-** #make –j8

First check your linux core using the command nproc**.** Then according to that the core will change i.e, I used –j8 as per your nproc result that number will change.



You may get these kind of errors while compiling the linux kernel:

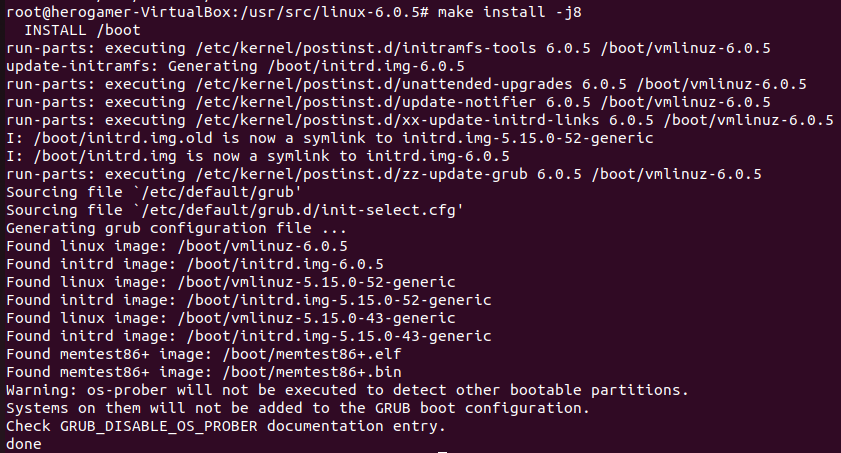
* #make[1]: \*\*\* No rule to make target 'debian/canonical-certs.pem', needed by 'certs/x509\_certificate\_list'. Stop.
* #make: \*\*\* [Makefile:1809: certs] Error 2

To solve this use these commands in the root:

* #scripts/config --disable SYSTEM\_TRUSTED\_KEYS
* #scripts/config --set-str SYSTEM\_TRUSTED\_KEYS ""
* #scripts/config --disable SYSTEM\_REVOCATION\_KEYS

After solving the error, again compile the kernel using makecommand.

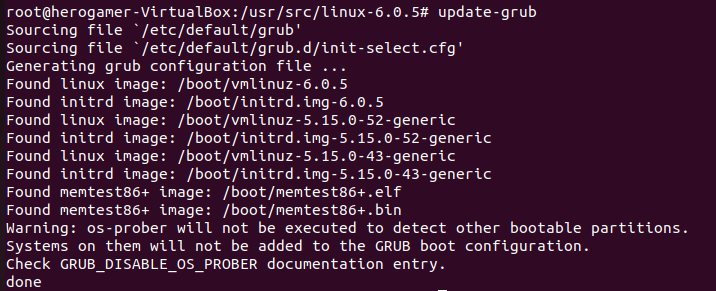
1. After compiling the kernel successfully install the kernel **-** #make install –j8



1. Preparation of the installer of the kernel **-** #sudo make modules\_install install.

This command executed for more than an hour. So, I was unable to take the screenshot.

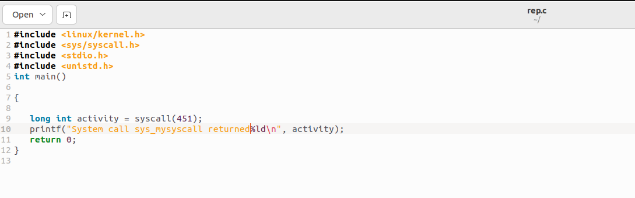
1. Updating the bootloader of the OS of the new kernel **-** #sudo update-grub



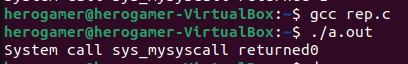
1. RESULT
2. Creation of a C file in desktop to generate a report of success, compiling it and generating the output **-** $gedit rep.c

$gcc rep.c

$./a.out

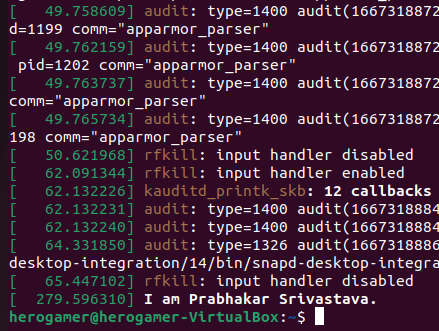


syscall(451), 451 is the updated line number of the syscalls\_64.tbl table. So, the line number will be according to your updated line in the table. Save and exit.



The output returns 0, this means that the addition of the system call is successful. If the output returns -1, this indicates that the system call isn’t implemented.

1. Finally the addition of system call **- #**dmesg



So this added system call can be used to make a custom device driver for our own devices by changing the **mysyscall.c** file in the home directory.

1. CONCLUSION

This paper is about adding a system call to the Linux Kernel. So anyone who is willing to add their own system call in the Linux Kernel. This paper will be useful for them. They will also know about system calls and and how to execute them.

1. REFERENCES

[1] D. Endler. Intrusion detection: applying machine learning to solaris audit data. In In Proc. of the IEEE Annual Computer Security Applications Conference, pages 268–279. Society Press, 1998.

[2] S. Chen, J. Xu, and E. C. Sezer. Non-control-data attacks are realistic threats. In 14th Annual Usenix Security Symposium, Aug 2005

[3] FREE SOFTWARE FOUNDATION. cvs - Concurrent Versions System. http://www.nongnu.org/cvs/.

[4] linux uses http://www.comparebusinessproducts.com/fyi/50- places-linux-running-you- might-not-expect

[5] H. Feng, O. Kolesnikov, P. Fogla, W. Lee, and W. Gong. Anomaly detection using call stack information. In Proceedings of the 2003 IEEE Symposium on Security and Privacy, May 2003.

[6] Wagner, D. and Soto, P. 2002. Mimicry Attacks on Host-Based Intrusion Detection Systems. In Proceedings of the 9 th ACM Conference on Computer and Communications Security. Washington DC, USA, 255–264.

[7] arstechnica http://arstechnica.com/security/2016/09/linux-kernel-security-needs-fixing/

[8] McConnell, S (1999). Open-Source Methodology: Ready for Prime Time ? In IEEE Software, 16 (4), 6-11.

[9] Greg heley explaining FreeBSD https://www.freebsd.org/doc/en\_US.ISO8859- 1/articles/explaining- bsd/article.html

[10] E. H. Spafford. Computer viruses—a form of artificial life? In C. G. Langton, C. Taylor, J. D. Farmer, and S. Rasmussen, editors, Artificial Life II, pages 727–745. Addison-Wesley, Redwood City, CA, 1992.

[11] S. Basu and P. Uppuluri. Proxi-Annotated Control Flow Graphs: Deterministic Context-Sensitive Monitoring for Intrusion Detection, pages 353–362. Springer, 2004.

[12] D. Gao, M. Reiter, and D. Song. Gray-box extraction of execution graphs for anomaly detection. In Proceedings of the 11th ACM Conference on Computer and Communications Security, pages 318–329, October 2004.

[13] Linux Open Projects http://nevonprojects.com/year-projects-for-computer-engineering/

[14] sL4 Projects https://wiki.sel4.systems/Getting%20started

[15] ts honor thesis project http://ts.data61.csiro.au/students/theses.pml#ug-GH

[16] Comparison of operating systems https://en.wikipedia.org/wiki/Comparison\_of\_operating \_systems

[17] L. Zhen, S. M. Bridges, and R. B. Vaughn. Combining static analysis and dynamic learning to build accurate intrusion detection models. In Proceedingsof the 3rd IEEE International Workshop on Information Assurance, March 2005

[18] Tan, K. and Maxion, R. 2002. ‖Why 6?‖ Defining the Operational Limits of Stide, an AnomalyBased Intrusion Detector. In Proceedings of the IEEE Symposium on Security and Privacy. Oakland, CA, 188–202.

[18] L4 kernel https://en.wikipedia.org/wiki/L4\_microkernel\_family