International Journal of Science and Research (IJSR)
ISSN (Online): 2319-7064
Index Copernicus Value (2015): 78.96 | Impact Factor (2015): 6.391

Survey Paper on Adding System Call in Linux Kernel 3.2+ & 3.16

Atiya Mumtaz

Computer Science and Engineering, Shivalik College of Engineering, Uttrakhand Technical University

Abstract: ***This paper explain how to add a system call in Linux Kernel 3.2 & 3.16 and covers prerequisites to add any system call in Linux Kernel, Paper also include Explanation of Process and commands required in kernel compilation, and briefly explain the process of system call addition, commands and packages used to do so.***

Keywords: Linux Kernel, System calls, Ubuntu 12.04, Ubuntu 14.04, System Call Table, Macro Definition, Prototype, Userspace Program, and Kernel Compilation

1. Introduction

System call addition process broadly involves adding a system call in system call table with its unique number, defining its structure (what system call will do, it can be any program) and its Macro definition, then compiling the edited kernel and rebooting system with new compiled kernel. There are simple steps which should be followed to add system call in Linux kernel but before that there are certain conditions or prerequisites which must be followed before starting with system call addition process, failure in any of the prerequisite step may create problem/error at the time of kernel compilation and hence addition of system call. This paper is divided into five sections, First section include perquisites to add system call, Second includes Steps to add system call in Linux kernel 3.2, third section provide with steps to add system call in Linux kernel 3.16, fourth explains steps to compile kernel and Userspace program to check added system call and fifth section explains briefly about system calls and process of system call execution.

2. Prerequisites

There are certain requirements which must be fulfilled before adding a system call in Linux kernel

a) **Get Kernel Source Code from kernel.org**
b) **Set root password:** After installation set root password by command: **sudo passwd root** and then execute **sudo passwd –u root** to unlock account.
c) **Access to root folder:** There are many ways to do so, one among them is to go to terminal (by alt+tab+T) and type the following **command: sudo chmod –R 777/root,** after running this command one might see some error but when one will go to root folder, can have access to that folder but even after one gain access to root folder one can't create, delete or make changes in any existing file in root, usr or src folder.(to get this liberty refer next point)
d) **Permission to alter files in root, usr or src folder:** There are various ways to get this but one may easily get this by typing **command: sudo nautilus,** in the terminal. After running this command, automatically a window will open and one will be able to do whatever one wants to do in root, src or usr folder.
e) **Installation of required packages:** go to terminal and type **command: sudo apt-get install gcc, sudo apt-get update, sudo apt-get install ncurses-devel** (if this doesn't work try using **command sudo apt-get install libncurses5-dev), sudo apt-get upgrade.**
f) **Extract your downloaded kernel source code in usr/src directory.**

3. Steps to add system call in Linux kernel 3.2

(Assuming that one has followed all the prerequisite steps)

1) **Add system call to system call table,** Open file **syscall_table_32.s**, path arch/x86/kernel.syscall_table_32.s, in the end of the file(EOF) append .long sys_hello
2) **Define Macro which are associated with system call,** Open file **unistd_64.h,** path arch/x86/include/asm/unistd_64.h, in EOF append new line #define __NR_hello 312 then in next line __SYSCALL(__NR_hello, sys_hello).
3) **Increment the value of Macro,** Open file **unistd_32.h,** path arch/x86/include/asm/unistd_32.h, Go to line #define NR_syscall 349 (you can search for this line by cntrl+F and by typing any word from the line) change this line to #define NR_syscall 350.
4) **Add prototype of the system call**, Open file syscalls.h, path include/linux/syscalls.h, in the EOF add line asmlinkage long sys_hello(void);.
5) **Create a directory** named hello in the root directory of the kernel i.e folder of the source co
6) de.
7) **In this hello directory create two files** named hello.c and Makefile.
8) **In hello.c write a program #include<linux/kernel.h> asmlinkage long sys_hello(void){printk("Hello world"\n);return 0;}**
9) **Open Makefile and write obj := hello.o**
10) **Now open Makefile in the root directory** and go to line _core-y += kernel/m/fs/ipc/security/crypto/block' (find it with the help of Cntrl+F) and change it to **_core-y += kernel/m/fs/ipc/security/crypto/block'/hello/'**
11) **System call is added, now follow kernel Compilation steps. (Mentioned in section 4).**

4. Steps to Add System Call in Linux Kernel 3.16

(Assuming that one has followed all the prerequisite steps)

1) Create a directory **named hello** within the directory linux-3.16, path usr/src/linux-3.16
2) Create two files within hello directory, **hello.c and Makefile**
3) In hello.c write a program

```
#include<linux/kernel.h>
asmlinkage long sys_hello(void)
{
printk(―hello\n”);
return 0;
}
```

4) In **Makefile** within hello directory write obj-y := hello.o
5) Edit **Makefile in the kernel's root directory**, goto line ‗core-y += kernel/mm/fs//ipc/security/crypto/block‘ and change it to

```
core-y += kernel/mm/fs//ipc/security/crypto/block/hello/
```

6) Add new system call into system call table, open file syscall_32.tbl (if your system is a 64 bit alter the syscall_64/tbl file) path arch/x86/syscalls/syscalls_64.tbl, add following line in the end of the file(EOF) 354 i386 hello sys_hello
7) Add new system call (sys_hello()) in the system call header file, Open file syscalls.h, path include/linux/syscalls.h and add following in the EOF asmlinkage long sys_hello(void)
8) System call is added, Now follow Kernel compilation steps. (Mentioned in next section)

5. Steps to Compile Kernel

1) Go to terminal(Cntrl+Alt+T) and go to directory where kernel source code is i.e type command: cd /usr/src/linux-3.2 or linux-3.16 according to kernel source directory name, to get path automatically open kernel source code directory then press Cntrl+L and copy highlighted path.
2) Type command sudo make menuconfig (pop up will come navigate to file system check whether ext4 is selected or not, if not select it and exit).
3) Type command sudo make oldconfig
4) Type command sudo make, this may take a while like 30-40 mins to compile kernel.
5) Type command sudo make modules_install install
6) Reboot system with compiled kernel.

5.1 Userspace Program

After rebooting system make a Userspace program which can test that system call is successfully added or not. Whenever we make library functions or system calls we should always check the return status of the call in order to determine its successful execution.
Userspace.c

```
#include<stdio.h>
#include<sys/syscall.h>
#include<linux/kernel.h>
#include<unistd.h>
int main()
{
long int ana =syscall(312);// kernel 3.2.82 64 bit – 312, 32 bit- 349, kernel 3.16- 354 (according to system call number in the system call table)
printf(―system call sys_hello returned %d”, ana);
return 0;
}
```

1) Compile and run userspace program
2) On successful compilation and execution it will print ―system call sys_hello returned 0”
3) To check message of the kernel
4) Type command dmesg and on its execution in the end hello world will be displayed**.**

6. System Call Addition Process

This section briefly explains what are a system calls, how it works and what are the various function running when any system calls are executed.

6.1 System Call

System call is a controlled way of entering into kernel, when process requests that kernel should perform some process on its behalf. System call API (Application Programming Interface) make this service available to program, these service include, New Process Creation, Performing Input/ Output, Creating a pipe for Inter Process Communication. When System call is going to be executed process‘s state change from user mode to kernel mode, to make CPU access protected kernel memory. System call has fixed set and is uniquely identified by a unique number (which is not normally visible to program, which identify system call by its name).

Process of system call execution

1) Application Program invokes a system call by wrapper function.
2) Wrapper function makes assurance that trap handling routine get system call arguments.
3) Generally arguments are passed to wrapper function using stack, but kernel looks for specific registers for these arguments, hence wrapper function also copies these arguments to specific registers.
4) Kernel need to identify each system call to know which system call is invoked and so each system call has its unique number. The wrapper functions copies system call number into CPU‘s specific registers
5) Then Trap Instruction (int 0x80) is executed by wrapper function, which causes the process to switch to kernel mode from user mode.
6) Then code pointed by location 0x80 is executed.
7) Kernel Invokes a system call routine which is located in assembler file as a response to trap location 0x80.
8) Register‘s values are saved onto stack by handler and some validation are done, like verifying system call number etc.
9) Handler uses system call table to invoke appropriate system call service routine and also validates the arguments if there are any. A system call table is a map

of system call number as a key and appropriate system call value.

10) Service routine performs various actions like transferring data between user memory and kernel memory or modifying values at address specified in arguments after proper validations from service routines.
11) After this service routine returns status of execution to the system call routine.
12) Returning process to user mode handler retuned to wrapper function.
13) If system call‘s return value indicated error then wrapper function sets _errno‘ on global variables and then returns to called process providing integer value indicating execution state.

7. Conclusion

This paper can help anyone to add system call in Linux kernel and to understand what system call is and how it is executed.

References

[1] Linux Open Projects http://nevonprojects.com/year-projects-for-computer-engineering/
[2] ts honor thesis project http://ts.data61.csiro.au/students/theses.pml#ug-GH
[3] 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.
[4] 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.
[5] Stolcke, A. and Omohundro, S. 1993. Hidden Markov Model Induction by Bayesian Model Merging. Advances in Neural Information Processing Systems.
[6] FREE SOFTWARE FOUNDATION. cvs - Concurrent Versions System. http://www.nongnu.org/cvs/.
[7] McConnell, S (1999). Open-Source Methodology: Ready for Prime Time ? In IEEE Software, 16 (4), 6-11.
[8] linux Vulnerbility trends https://www.cvedetails.com/product/47/Linux-Linux-Kernel.html? vendor_id=33
[9] arstechnica http://arstechnica.com/security/2016/09/linux-kernel-security-needs-fixing/
[10] sL4 Projects https://wiki.sel4.systems/Getting%20started
[11] L4 kernel https://en.wikipedia.org/wiki/L4_microkernel_family
[12] linux uses http://www.comparebusinessproducts.com/fyi/50-places-linux-running-you- might-not-expect
[13] Greg heley explaining FreeBSD https://www.freebsd.org/doc/en_US.ISO8859-1/articles/explaining- bsd/article.html
[14] Comparison of operating systems https://en.wikipedia.org/wiki/Comparison_of_operating_systems
[15] Mount DM (2004). Bioinformatics: Sequence and Genome Analysis (2 Ed.). Cold Spring Harbor, NY: Cold Spring Harbor Laboratory Press. ISBN 0- 87969-712-1. OCLC 55106399 .
[16] System call addition Mount DM (2004). Bioinformatics: Sequence and Genome Analysis (2 Ed.). Cold Spring Harbor, NY: Cold Spring Harbor Laboratory Press. ISBN 0-87969-712-1. OCLC 55106399 .
[17] 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.
[18] 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.
[19] 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.
[20] S. Basu and P. Uppuluri. Proxi-Annotated Control Flow Graphs: Deterministic Context-Sensitive Monitoring for Intrusion Detection, pages 353–362. Springer, 2004.
[21] M. K. Aguilera, M. Lillibridge, and X. Li. Transaction rate limiters for peer-to- peer systems. IEEE International Conference on Peer-to-Peer Computing, 0:3– 11, 2008
[22] D. Anderson, T. Frivold, and A. Valdes. Next-generation intrusion detection expert system (nides): A summary. Technical Report SRI-CSL-95-07,
[23] Computer Science Laboratory, SRI International, May 1995.
[24] S. Chen, J. Xu, and E. C. Sezer. Non-control-data attacks are realistic threats. In 14th Annual Usenix Security Symposium, Aug 2005
[25] 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.
[26] H. Xu, W. Du, and S. J. Chapin. Context sensitive anomaly monitoring of process control flow to detect mimicry attacks and impossible paths. In Proceedings of the International Symposium on Recent Advances in Intrusion Detection (RAID, pages 21–38. Springer, 2004.
[27] 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.