Linux 内核漏洞利用教程（一）：环境配置

作者：[o0xmuhe](http://bobao.360.cn/member/contribute?uid=32403999)

[**【系列分享】Linux 内核漏洞利用教程（二）：两个Demo**](http://bobao.360.cn/learning/detail/3702.html)

**0x00: 前言**

一直想入门linux kernel exploit，但是网络上比较成熟的资料很少，只能找到一些slide和零碎的文档，对于入门选手来说真的很困难。还好在自己瞎摸索的过程中上了joker师傅的装甲车，师傅说：要有开源精神，要给大家学习的机会。

所以就有了这个系列的文章，第一篇记录是环境配置篇，包含了linux内核编译、添加系统调用并测试的过程。在这个过程中我还是遇到很多坑点的，踩了一段时间才把这些坑填好，成功搞定，希望我的经历能给大家一点帮助。

**0x01: 环境说明**

**ubuntu** 14.04 **x86**

**qemu**

使用的内核版本[2.6.32.1](https://www.kernel.org/pub/linux/kernel/v2.6/linux-2.6.32.1.tar.gz%20-O%20linux-2.6.32.1.tar.gz)

busybox版本[1.19.4](https://busybox.net/downloads/)

使用busybox是因为文件添加方便.

**0x02: 内核编译并测试**

**1. 下载内核源码**

$ wget https://www.kernel.org/pub/linux/kernel/v2.6/linux-2.6.32.1.tar.gz -O linux-2.6.32.1.tar.gz

$ tar -xjvf linux-2.6.32.1.tar.gz

**2. 编译过程**

首先要安装一些依赖库以及qemu。

**$** cd linux-2.6.32.1/

**$** sudo apt-get install libncurses5-dev

**$** sudo apt-get install qemu qemu-system

**$** make menuconfig

**$** make

**$** make all

**$** make modules

**3. 编译的时候遇到的问题以及解决方案**

**3.1 问题1**

问题

Can't **use** 'defined(@array)' (Maybe you should just omit the defined()?) **at** kernel/timeconst.pl line 373.

/home/muhe/linux\_kernel/linux-2.6.32.1/linux-2.6.32.1/kernel/Makefile:129: recipe **for** target 'kernel/timeconst.h' **failed**

make[1]: \*\*\* [kernel/timeconst.h] **Error** 255

Makefile:878: recipe **for** target 'kernel' **failed**

make: \*\*\* [kernel] **Error** 2

解决方案: 尝试[修改这个文件](http://www.playpenguin.net/an-zhuo-nei-he-bian-yi-cuo-wu-kerneltimeconsth-definedval.html)

    @val = @{$canned\_values{$hz}};

-   **if** (!**defined**(@val)) {

+   **if** (!@val) {

            @val = compute\_values($hz);

    }

    output($hz, @val);

--

**3.2 问题2**

问题描述

....

arch/x86/kernel/ptrace.c:1472:17: error: conflicting types **for** ‘syscall\_trace\_enter’

 asmregparm **long** **syscall\_trace\_enter**(**struct** pt\_regs \*regs)

                 ^

In file included **from** /home/muhe/linux\_kernel/linux-2.6.32.1/arch/x86/include/asm/vm86.h:130:0,

**from** /home/muhe/linux\_kernel/linux-2.6.32.1/arch/x86/include/asm/processor.h:10,

**from** /home/muhe/linux\_kernel/linux-2.6.32.1/arch/x86/include/asm/thread\_info.h:22,

**from** include/linux/thread\_info.h:56,

**from** include/linux/preempt.h:9,

**from** include/linux/spinlock.h:50,

**from** include/linux/seqlock.h:29,

**from** include/linux/time.h:8,

**from** include/linux/timex.h:56,

**from** include/linux/sched.h:56,

**from** arch/x86/kernel/ptrace.c:11:

/home/muhe/linux\_kernel/linux-2.6.32.1/arch/x86/include/asm/ptrace.h:145:13: note: previous declaration of ‘syscall\_trace\_enter’ was here

**extern** **long** **syscall\_trace\_enter**(**struct** pt\_regs \*);

             ^

arch/x86/kernel/ptrace.c:1517:17: error: conflicting types **for** ‘syscall\_trace\_leave’

 asmregparm **void** **syscall\_trace\_leave**(**struct** pt\_regs \*regs)

                 ^

In file included **from** /home/muhe/linux\_kernel/linux-2.6.32.1/arch/x86/include/asm/vm86.h:130:0,

**from** /home/muhe/linux\_kernel/linux-2.6.32.1/arch/x86/include/asm/processor.h:10,

**from** /home/muhe/linux\_kernel/linux-2.6.32.1/arch/x86/include/asm/thread\_info.h:22,

**from** include/linux/thread\_info.h:56,

**from** include/linux/preempt.h:9,

**from** include/linux/spinlock.h:50,

**from** include/linux/seqlock.h:29,

**from** include/linux/time.h:8,

**from** include/linux/timex.h:56,

**from** include/linux/sched.h:56,

**from** arch/x86/kernel/ptrace.c:11:

/home/muhe/linux\_kernel/linux-2.6.32.1/arch/x86/include/asm/ptrace.h:146:13: note: previous declaration of ‘syscall\_trace\_leave’ was here

**extern** **void** **syscall\_trace\_leave**(**struct** pt\_regs \*);

             ^

make[2]: \*\*\* [arch/x86/kernel/ptrace.o] 错误 1

make[1]: \*\*\* [arch/x86/kernel] 错误 2

make: \*\*\* [arch/x86] 错误 2

解决方案

patch patch -p1 < /tmp/1.patch

--- linux-2.6.32.59/arch/x86/include/**asm**/ptrace.h

+++ fix\_ptrace.o\_compile\_error/arch/x86/include/**asm**/ptrace.h

@@ -130,6 +130,7 @@

ifdef KERNEL

include

+**#include**

**struct** **cpuinfo\_x86**;

**struct** **task\_struct**;

@@ -142,8 +143,8 @@

**int** error\_code, **int** si\_code);

**void** **signal\_fault**(struct pt\_regs regs, **void** \_\_user frame, **char** \*where);

-**extern** **long** **syscall\_trace\_enter**(struct pt\_regs );

-**extern** **void** **syscall\_trace\_leave**(struct pt\_regs );

+**extern** asmregparm **long** **syscall\_trace\_enter**(struct pt\_regs );

+**extern** asmregparm **void** **syscall\_trace\_leave**(struct pt\_regs );

**static** **inline** **unsigned** **long** **regs\_return\_value**(struct pt\_regs \*regs)

{

**3.3 问题3**

问题描述

**gcc: error: elf\_i386: 没有那个文件或目录**

**gcc: error: unrecognized command line option ‘-m’**

解决方案

    arch/x86/vdso/Makefile

    VDSO\_LDFLAGS\_vdso.lds = -m elf\_x86\_64 -Wl,-soname=linux-vdso.so.1    -Wl,-z,max-page-size=4096 -Wl,-z,common-page-size=4096 把"-m elf\_x86\_64" 替换为 "-m64"

    VDSO\_LDFLAGS\_vdso32.lds = -m elf\_i386 -Wl,-soname=linux-gate.so.1中的 "-m elf\_i386" 替换为 "-m32"

**3.4 问题4**

问题描述

**drivers/net/igbvf/igbvf.h15: error: duplicate member ‘page’**

struct page page;

^

**make[3]: \*\* [drivers/net/igbvf/ethtool.o] 错误 1**

**make[2]: [drivers/net/igbvf] 错误 2**

**make[1]: [drivers/net] 错误 2**

**make: \* [drivers] 错误 2**

解决方案

*//修改名字重复*

**struct** {

**struct** **page** \*\_**page**;

                        u64 page\_dma;

**unsigned** **int** page\_offset;

                };

        };

**struct** **page** \***page**;

**0x03:增加syscall**

增加syscall的方式和之前[文章](http://o0xmuhe.me/2017/02/08/Adding-your-own-syscall-in-linux-kernel/)写的差不多，只是这次内核版本更低，所以更简单一点。我这里添加了两个系统调用进去。

**1. 在syscall table中添加信息**

文件 arch/x86/kernel/syscall\_table\_32.S中添加自己的调用

    .**long** sys\_muhe\_test

    .**long** sys\_hello

**2. 定义syscall的宏**

文件arch/x86/include/asm/unistd\_32.h中添加

**#define \_\_NR\_hello 337**

**#define \_\_NR\_muhe\_test    338**

**#ifdef \_\_KERNEL\_\_**

**#define NR\_syscalls 339**

要注意NR\_syscalls要修改成现有的调用数目，比如原来有0~336一共337个调用，现在增加了两个，那就改成339。

**3. 添加函数定义**

文件include/linux/syscalls.h

asmlinkage **long** **sys\_muhe\_test**(**int** arg0);

asmlinkage **long** **sys\_hello**(**void**);

**4. 编写syscall代码**

新建目录放自定义syscall的代码

**# muhe @ ubuntu in ~/linux\_kernel/linux-2.6.32.1/linux-2.6.32.1/muhe\_test [2:43:06]**

$ cat muhe\_test.c

**#include <linux/kernel.h>**

asmlinkage **long** **sys\_muhe\_test**(**int** arg0){

    printk("I am syscall");

    printk("syscall arg %d",arg0);

**return** ((**long**)arg0);

}

asmlinkage **long** **sys\_hello**(**void**){

    printk("hello my kernel worldn");

**return** 0;

}

**# muhe @ ubuntu in ~/linux\_kernel/linux-2.6.32.1/linux-2.6.32.1/muhe\_test [2:43:12]**

$ cat Makefile

obj-y := muhe\_test.o

**5. 修改Makefile**

*# muhe @ ubuntu in ~/linux\_kernel/linux-2.6.32.1/linux-2.6.32.1 [2:44:59]*

$ cat Makefile| grep muhe

core-y        += kernel/ mm/ fs/ ipc/ security/ crypto/ block/ muhe\_test/

**6. 编译**

make -j2

我虚拟机分配了两个核，所以使用-j2 这样能稍微快一点。

## 0x04: busybox编译配置

**1. 编译步骤**

**$** make menuconfig

**$** make

**$** make install

**2. 遇到的问题**

**2.1 问题一以及解决方案**

错误

    loginutils/passwd.c:188:12: error: ‘RLIMIT\_FSIZE’ undeclared (first use **in** **this** **function**)

**setrlimit**(RLIMIT\_FSIZE, &rlimit\_fsize);

解决

$  vim include/libbb.h

$  add a line **#include <sys/resource.h>**

**#include <sys/mman.h>**

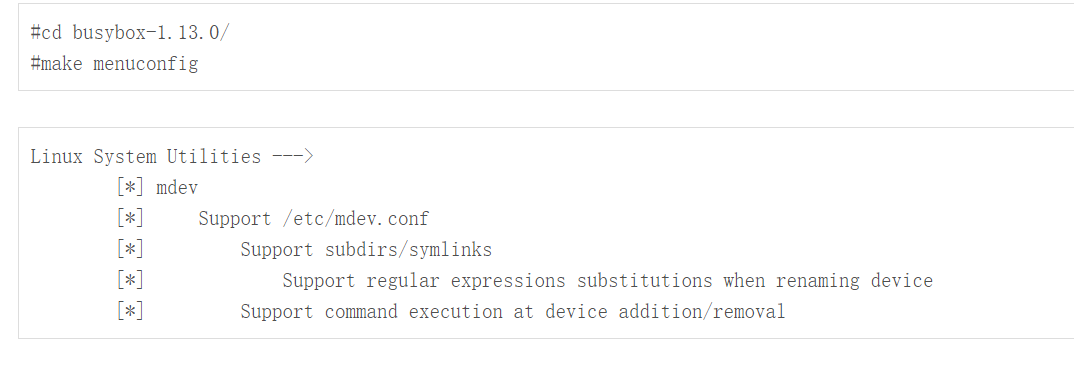
**#include <sys/resource.h>**

**#include <sys/socket.h>**

**2.2 问题二以及解决方案**

错误

linux/ext2\_fs.h: 没有那个文件或目录



解决

    Linux System Utilities --->

        [] mkfs\_ext2

        [] mkfs\_vfat

**3. 编译完成之后如下配置**

**1. 方案1 有效果，方案2 不明朗**

$ cd \_install

$ mkdir -pv {bin,sbin,etc,proc,sys,usr/{bin,sbin}}

$ cat init

**#!/bin/sh**

echo "INIT SCRIPT"

mount -t proc none /proc

mount -t sysfs none /sys

mount -t debugfs none /sys/kernel/debug

mkdir /tmp

mount -t tmpfs none /tmp

mdev -s *# We need this to find /dev/sda later*

echo -e "nBoot took $(cut -d' ' -f1 /proc/uptime) secondsn"

exec /bin/sh

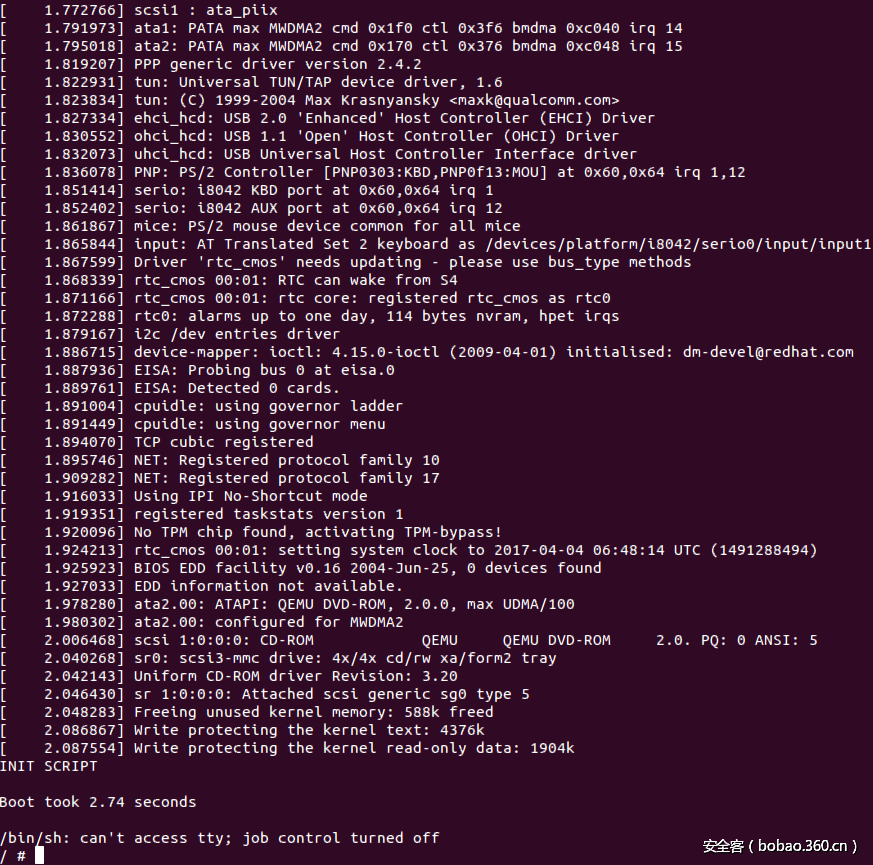
$ chmod +x init

$ find . -print0

    | cpio --null -ov --format=newc

    | gzip -9 > /tmp/initramfs-busybox-x86.cpio.gz

$ qemu-system-i386 -kernel arch/i386/boot/bzImage -initrd /tmp/initramfs-busybox-x86.cpio.gz

[](https://p4.ssl.qhimg.com/t018e0f1a330b995e1f.png)

ubzhangji@ubuntu:~/qemu\_study/busybox-1.19.4/\_install$ find . -print0 | cpio --null -ov --format=newc | gzip -9 > /tmp/initramfs-busybox-x86.cpio.gz

ubzhangji@ubuntu:~/qemu\_study/linux-2.6.32.1$ qemu-system-x86\_64 -kernel arch/x86/boot/bzImage -initrd /tmp/initramfs-busybox-x86.cpio.gz

ubzhangji@ubuntu:~/qemu\_study/linux-2.6.32.1$ qemu-system-x86\_64 -kernel arch/x86\_64/boot/bzImage -initrd /tmp/initramfs-busybox-x86.cpio.gz

**2. 方案2**

后面为了方便，使用了另一种方式：

目录结构和之前差不多，添加inittab文件：

$ cat etc/inittab

::sysinit:/etc/init.d/rcS

::askfirst:/bin/ash

::ctrlaltdel:/sbin/reboot

::shutdown:/sbin/swapoff -a

::shutdown:/bin/umount -a -r

::restart:/sbin/init

添加rcS文件

$ cat etc/init.d/rcS

**#!/bin/sh**

**#!/bin/sh**

mount -t proc none /proc

mount -t sys none /sys

/bin/mount -n -t sysfs none /sys

/bin/mount -t ramfs none /dev

/sbin/mdev -

$ chmod +x ./etc/init.d/rcS

配置下dev目录

mkdir dev

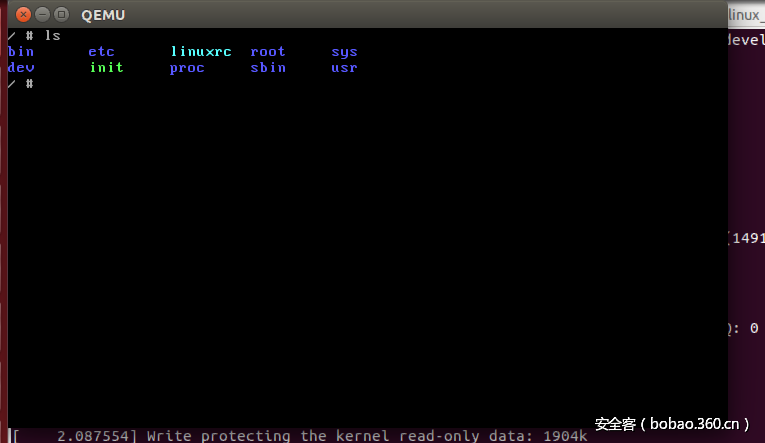
sudo mknod dev/ttyAMA0 c 204 64

sudo mknod dev/null c 1 3

sudo mknod dev/console c 5 1

**$** find . | cpio -o --format=newc > ../rootfs.img

**$** qemu-system-i386 -kernel arch/i386/boot/bzImage -initrd ../busybox-1.19.4/rootfs.img -append "root=/dev/ram rdinit=/sbin/init"

[](https://p1.ssl.qhimg.com/t013d5950b18882bfa5.png)

**0x05: 测试系统调用**

**#** muhe @ ubuntu **in** ~/linux\_kernel/linux-2.6.32.1/linux-2.6.32.1 [2:45:04]

**$** cd muhe\_test\_syscall\_lib

**#** muhe @ ubuntu **in** ~/linux\_kernel/linux-2.6.32.1/linux-2.6.32.1/muhe\_test\_syscall\_lib [2:51:48]

**$** cat muhe\_test\_syscall\_lib.c

**#**include <stdio.h>

**#**include <linux/unistd.h>

**#**include <sys/syscall.h>

int main(int argc,char \*\*argv)

{

        printf("n Diving to kernel levelnn");

        syscall(337,1337);

        return 0;

}

**#** muhe @ ubuntu **in** ~/linux\_kernel/linux-2.6.32.1/linux-2.6.32.1/muhe\_test\_syscall\_lib [2:51:51]

**$** gcc muhe\_test\_syscall\_lib.c -o muhe -static

一定要静态链接，因为你进busybox链接库那些是没有的。

**#** muhe @ ubuntu **in** ~/linux\_kernel/linux-2.6.32.1/linux-2.6.32.1/muhe\_test\_syscall\_lib [2:52:20]

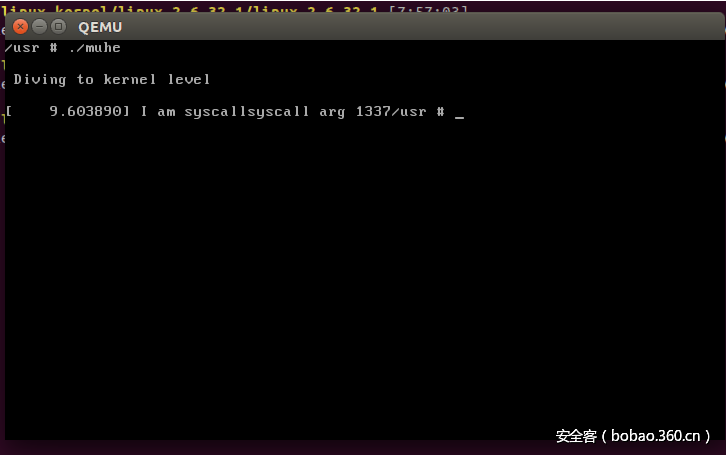
**$** cp muhe\_test\_syscall\_lib/muhe ../busybox-1.19.4/\_install/usr/muhe

这里要注意，每次拷贝新文件到busybox的文件系统中去，都要执行find . | cpio -o –format=newc > ../rootfs.img去生成新的rootfs。

然后qemu起系统

**#** muhe @ ubuntu **in** ~/linux\_kernel/linux-2.6.32.1/linux-2.6.32.1 [2:53:33]

**$**  qemu-system-i386 -kernel arch/i386/boot/bzImage -initrd ../busybox-1.19.4/rootfs.img -append "root=/dev/ram rdinit=/sbin/init"

[](https://p4.ssl.qhimg.com/t01b9a4ea1b41531779.png)

**0x06:引用与参考**

[adding-hello-world-system-call-to-linux](https://arvindsraj.wordpress.com/2012/10/05/adding-hello-world-system-call-to-linux/)

[Adding a new system call to the Linux kernel](http://www.cs.rochester.edu/~sandhya/csc256/)

[Adding a system call in X86 QEMU Environment](http://linuxseekernel.blogspot.ie/2014/07/adding-system-call-in-x86-qemu.html)

[Create a simple file system](http://linuxseekernel.blogspot.com/2014/06/create-simple-file-system.html)

[Setup for linux kernel dev using qemu](https://beyermatthias.de/blog/2016/11/01/setup-for-linux-kernel-dev-using-qemu/)

[root-file-system-for-embedded-system](http://blog.3mdeb.com/2013/06/07/root-file-system-for-embedded-system/)

**Adding hello world system call to Linux**

October 5, 2012（针对本机添加系统调用）

I just finished doing this and so thought I’d write about it in my blog since it took me a while to get this done after many an experimentation I suppose :P. So here goes!

I did this on an Ubuntu 12.04.1 LTS OS running on an 64bit machine but this post should mostly work on any Linux OS(with a few differences here and there).

**Step 1: Get the source**

The first step is to download the source code of the Linux kernel. I used the one available from the repositories but feel free to get the sources from [**kernel.org**](http://kernel.org/).

|  |  |
| --- | --- |
| 1 | apt-get source linux-image-$(uname -r) |

This would download all the archives and unpack it into a directory linux-3.2.0.

**Step 2: Add system call to system call table**

Open the file arch/x86/kernel/syscall\_table\_32.S and add the following line.

|  |  |
| --- | --- |
| 1 | .long sys\_hello |

**Step 3: Define macros associated with system call**

Open the file arch/x86/include/asm/unistd\_32.h. You will notice that a macro is defined for each system call. At the end of the huge macro definition, add a definition for our new system call. I added the following line:

|  |  |
| --- | --- |
| 1 | #define \_\_NR\_hello 349 |

and accordingly incremented the value of the macro NR\_SYSCALLS:

|  |  |
| --- | --- |
| 1 | #define NR\_syscalls 350 |

Also, add the macro definition to the file arch/x86/include/asm/unistd\_64.h

|  |  |
| --- | --- |
| 1  2 | #define \_\_NR\_hello 312 // is right   \_\_SYSCALL(\_\_NR\_hello, sys\_hello) |

Now to the file include/linux/syscalls.h, add the prototype of the system call.

|  |  |
| --- | --- |
| 1 | asmlinkage long sys\_hello(void); |

Now, in the root directory of the kernel sources, create a directory named hello and in it, a file hello.c with the following content:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | #include <linux/kernel.h>    asmlinkage long sys\_hello(void)  {      printk("Hello world\n");      return 0;  } |

printk is similar to printf function of C but writes to the kernel log instead of the screen. asmlinkage is a key word used to indicate that all parameters of the function(here none of course  ) would be available on the stack.

After creating the function definition, create a file named Makefile within the hello directory and the following content to the file:

|  |  |
| --- | --- |
| 1 | obj-y := hello.o |

This is to ensure that our hello.c file is compiled and included in the kernel.

Now, to the Makefile in the root directory of the kernel sources, edit the following line:

|  |  |
| --- | --- |
| 1 | core-y          += kernel/ mm/ fs/ ipc/ security/ crypto/ block/ |

to:

|  |  |
| --- | --- |
| 1 | core-y          += kernel/ mm/ fs/ ipc/ security/ crypto/ block/ hello/ |

This is to tell the compiler that the source files of our new system call are in present in the hello directory.

That’s it-you have added your own system call! Now all you need to compile the kernel. [1] is a good place to get information on how to compile the kernel. You needn’t do all the steps since you have the sources-read it carefully :)!

After you compile and reboot into the kernel you just compiled, try running the following program.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | #include <stdio.h>  #include <linux/kernel.h>  #include <sys/syscall.h>  #include <unistd.h>    #define \_\_NR\_hello 312 //349 if you are running a 32bit kernel and following my tutorial    long hello\_syscall(void)  {      return syscall(\_\_NR\_hello);  }    int main(int argc, char \*argv[])  {      long int a = hello\_syscall();      printf("System call returned %ld\n", a);      return 0;  } |

The output of the program would be:

|  |  |
| --- | --- |
| 1 | System call returned 0 |

The printk’s output get written to the kernel log. To view it, run the command

|  |  |
| --- | --- |
| 1 | dmesg |

and sure enough you will see the Hello World on the very last line.

Congratulations you have just added a system call to Linux!

**Helpful resources**

[1] [**http://www.howopensource.com/2011/08/how-to-compile-and-install-linux-kernel-3-0-in-ubuntu-11-04-10-10-and-10-04/**](http://www.howopensource.com/2011/08/how-to-compile-and-install-linux-kernel-3-0-in-ubuntu-11-04-10-10-and-10-04/)

[2] [**http://bluegrit.cs.umbc.edu/~lsebald1/cmsc421/new-syscall.php**](http://bluegrit.cs.umbc.edu/~lsebald1/cmsc421/new-syscall.php)

[3] [**https://wiki.ubuntu.com/Kernel/BuildYourOwnKernel**](https://wiki.ubuntu.com/Kernel/BuildYourOwnKernel) (easier method)

[4] [**https://help.ubuntu.com/community/Kernel/Compile**](https://help.ubuntu.com/community/Kernel/Compile)