针对Linux x86_64内核，如何自己写系统调用

写一个helloworld系统调用

写Linux系统调用是Linux编程中经常的需求，本文将详细讲述这个过程，需要说明的是：**本文针对Linux Ubuntu 64位操作系统，32位不支持**

1. 依赖安装

注意：如果下面所示的依赖不能用apt-get的方式安装，请自行使用源码安装，当然可能你的系统中已经完成了这些依赖的安装。

```
sudo apt-get update  #更新
sudo apt-get upgrade
sudo -s
apt-get install gcc
apt-get install python-pip python-dev libffi-dev libssl-dev libxml2-dev libxslt1-dev libjpeg8-dev zlib1g-dev
apt-get install libncursesw5-dev
```

2. 下载一个内核版本

在https://www.kernel.org/ 下载一个需要的Linux内核版本（究竟选用那个Linux系统版本，取决于你自己的需求），并解压该文件。

3. 写helloworld系统调用

```
cd ***.x   #进入到Linux内核解压后的根目录下
mkdir helloworld
cd helloworld
vim helloworld.c
```

在helloworld.c中写入下面的内容：

```
#include <linux/kernel.h>
asmlinkage long sys_helloworld(void){
    printk("Hello World~\n");
    return 0;
}
```

接着执行

```
vim Makefile  #和helloworld.c在同一个目录下
```

在Makefile文件中写入下面的内容：

```
obj-y  := helloworld.o
```

接着回到Linux内核源码的根目录中，修改Linux内核自带的Makefile文件，在其中写入下面内容：

```
core-y += kernel/.../ helloworld/   #只有helloworld是需要我们添加的，其它的在Linux本身的Makefile文件中就全部都有
```

接着执行

```
cd include/linux
vim syscalls.h
```

在打开的文件的最后面写入下面内容

```
asmlinkage long sys_helloworld(void);
```

接着回到Linux内核根目录，并执行下面指令

```
cd arch/x86/entry/syscalls
vim syscall_64.tbl
```

在打开的文件中写入下面内容：

```
333  64 helloworld  sys_helloworld   #333 is index of our system call
```

回到Linux内核的根目录

```
make menuconfig   #保存即可
make oldconfig
make -j 4  #编译内核
make modules_install install #安装新的内核
reboot   #重启计算机
uname -a   #查看是否进入到新内核中
```

4. 写C语言程序查看成功插入helloworld系统调用模块

```
//将该文件命名为test.c，并写入下面的代码，测试helloworld系统调用是否能用
#include <stdio.h>
#include <linux/kernel.h>
#include <sys/syscall.h>
#include <unistd.h>

int main(){
    long int s = syscall(333);   //333 is index of helloworld system call
    printf("System call : sys_helloworld : return %1d\n" , s);
```

```
    return 0;
}
```

写完test.c之后编译并运行，查看结果，如果return后面输出的值为0，说明上述系统调用完全正确。

```
gcc test.c
./a.out  // 输出return 0 ，值为0说明所有的系统调用都是成功的
dmesg   #查看kernel日志，最后一行看到存在helloworld，说明成功写出helloworld系统调用
```

如何判断是否真正以模块化方式插入系统调用

用于测试，我们写出下面的测试函数，

```
//文件命名为test.c
#include <stdio.h>
#include <linux/kernel.h>
#include <sys/syscall.h>
#include <unistd.h>

long helloworld(){   //在上文已经插入的helloworld系统调用模块，该系统调用的作用是申请一块连续的地址空间，并返回其首地址，malloc本身并不是系统调用
  int ret;
  __asm__("movl $333,%eax"); //将helloworld系统调用对应的系统调用号推到寄存器中
  //__asm__("movl $参数值1,%ebx");
  //__asm__("movl $参数值2,%ecx");
  //__asm__("movl $参数值,%edx");
  __asm__("int $0x80");    //陷入到系统调用中，完成用户态到内核态的切换，Linux选择128号（0x80）作为陷入指令入口
  __asm__("movl %eax, -4(%ebp)");
  return ret;
}

int main (){
  //long int s = syscall(333);   //如果采用这条命令，则不需要使用上面的helloworld封装例程，直接使用syscall(系统调用号，参数值1，参数值2...)
```

```
  long int s = helloworld();   //调用上述封装例程，在这个封装例程中，能够写入指定个数的参数，一般是6个参数
  printf("return %1d\n",s);
  return 0;
}
```

上述程序的大致流程为：应用程序调用helloworld，接着调用封装例程（也就是写着int $0x80的函数），通过陷入指令进入到内核中，或者直接使用syscall()函数，直接进入到系统调用。

接着对上述代码执行下面的命令：

```
sudo -s    #进入到root权限下执行下面的命令
dmesg   #查看kernel日志，能够看到执行的系统调用
gcc test.c -o test
./test
dmesg    #在kernel日志中应该能够看到有helloworld字样输出，但是没有helloworld1字样输出，system(333)的返回值为0，由这三个输出能够得出：按照上述过程，我们成功在内核中插入了helloworld系统调用模块
```

参考资料

1. 相对下面几个参考资料，该资料实现系统调用的方式比较正确 http://www.franksthinktank.com/howto/addsyscall/
2. https://linux.cn/article-9628-1.html
3. http://www.cnblogs.com/hazir/p/three_methods_of_syscall.html
4. http://mooc.study.163.com/course/1000072000#/info
5. https://www.linux.it/~rubini/docs/ksys/
6. https://www.tldp.org/LDP/lkmpg/2.6/html/lkmpg.html
7. https://www.slideshare.net/mehershree/how-to-add-system-call-in-ubuntu-os
8. http://gityuan.com/2016/05/21/syscall/
9. https://blog.csdn.net/sdulibh/article/details/51889279/
10. 正确的资料 https://blog.csdn.net/sinat_28750977/article/details/50837996
11. 较为正确的资料 https://blog.csdn.net/JackLiu16/article/details/79477967 这个资料告诉我 想要写更加复杂的系统调用逻辑，必须要学会Linux系统编程

12. malloc系统调用实现

    https://blog.csdn.net/zhangyang249/article/details/78582809