操作系统课程设计报告

向Linux内核添加一个系统调用

1 项目简介

1.1 项目目的

- 学习Linux操作系统提供的系统调用接口。
- 学习用户程序如何通过接口与操作系统内核实现通信。

1.2 项目要求

在现有的系统中添加一个系统调用并通过用户程序调用。

2 项目实现

2.1 系统版本

- Ubuntu：16.04
- Linux内核：4.15.13

2.2 实现过程

2.2.1 下载内核源代码

git.kernel.org下载内核(linux-stable-4.15.13.tar.gz)，复制并解压到/usr/src，命令：

```
hg@ubuntu:/usr/src$ sudo tar -xzvf linux-stable-4.15.13.tar.gz
```

图 1: 下载并解压内核

2.2.2 向内核添加新的系统调用

1. **添加系统调用函数。**修改/kernel/sys.c，加入linkage.h头文件和系统调用函数。命令：

```
hg@ubuntu:/usr/src/linux-stable-4.15.13$ sudo vim kernel/sys.c
```

图 2: 使用vim编辑sys.c

sys.c修改部分：

```
hg@ubuntu: /usr/src/linux-stable-4.15.13
// SPDX-License-Identifier: GPL-2.0
/*
 *  linux/kernel/sys.c
 *
 *  Copyright (C) 1991, 1992  Linus Torvalds
 */

#include <linux/linkage.h>
#include <linux/export.h>
#include <linux/mm.h>
#include <linux/utsname.h>
#include <linux/mman.h>
#include <linux/reboot.h>
#include <linux/prctl.h>
#include <linux/highuid.h>
#include <linux/fs.h>
#include <linux/kmod.h>
#include <linux/perf_event.h>
#include <linux/resource.h>
#include <linux/kernel.h>
```

图 3: 添加linkage.h头文件

```
            __put_user(s.loads[2], &info->loads[2]) ||
            __put_user(s.totalram, &info->totalram) ||
            __put_user(s.freeram, &info->freeram) ||
            __put_user(s.sharedram, &info->sharedram) ||
            __put_user(s.bufferram, &info->bufferram) ||
            __put_user(s.totalswap, &info->totalswap) ||
            __put_user(s.freeswap, &info->freeswap) ||
            __put_user(s.procs, &info->procs) ||
            __put_user(s.totalhigh, &info->totalhigh) ||
            __put_user(s.freehigh, &info->freehigh) ||
            __put_user(s.mem_unit, &info->mem_unit))
                return -EFAULT;

        return 0;
}

asmlinkage int sys_helloworld(int s)
{
    printk(KERN_EMERG "hello world");

    return 1;
}
#endif /* CONFIG_COMPAT */
                                                              2571,
```

图 4: 添加系统调用实现

2. **声明系统调用函数**。在syscalls.h添加系统调用函数的声明：

```
hg@ubuntu:/usr/src/linux-stable-4.15.13$ sudo vim arch/x86/include/asm/syscalls.h
```

图 5: 使用vim编辑syscalls.h

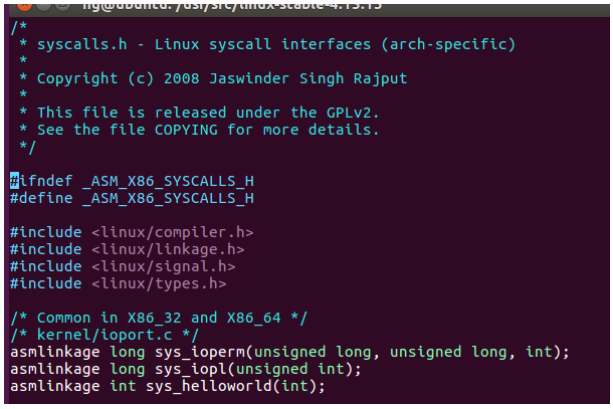

图 6: 添加系统调用函数声明

3. **添加系统调用号。** 在syscall_64.tbl中添加系统调用号

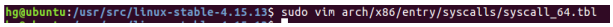

图 7: 使用vim编辑syscall_64.tbl

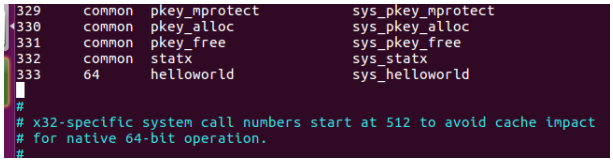

图 8: 添加系统调用号

2.2.3 编译安装内核

编译安装内核命令如下：

sudo make mrproper
sudo make clean
sudo make menuconfig
sudo make –j4
sudo make modules_install
sudo make install

2.2.4 测试系统调用

重启后选择进入linux-stable-4.15.13。

1. **编写测试程序。**编写程序调用新的系统调用：

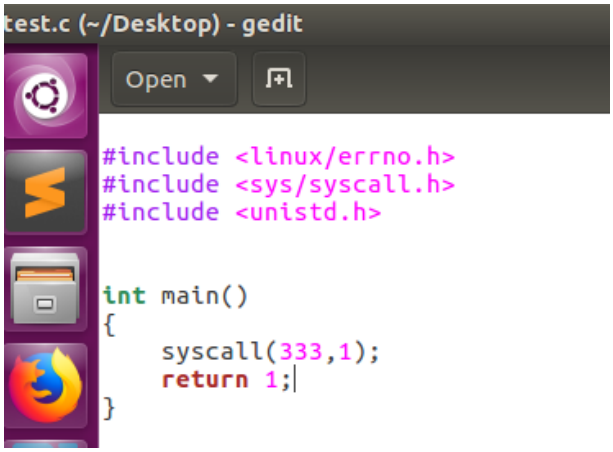

图 9: 测试程序

2. **运行**。运行测试程序并通过dmesg查看内核是否发出hello world消息:

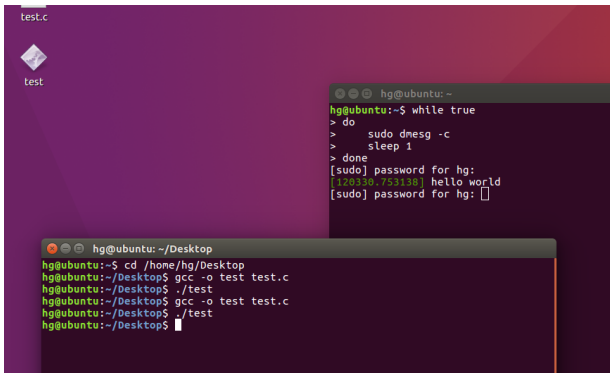

图 10: 运行测试程序并查看内核消息

2.3 项目中出现的问题及解决方案

1. 安装包缺失

编译时，终端报错如图11所示。

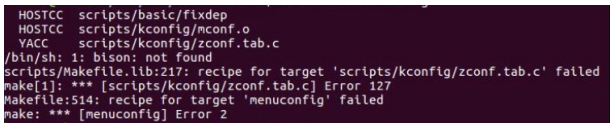

图 11: 编译测试程序时报错

安装缺失的安装包bison即可。命令：

sudo apt install bison

对于其他安装包缺失的报错可类似解决。

2. 硬盘空间不足。

内核编译过程中，硬盘空间耗尽，如图：

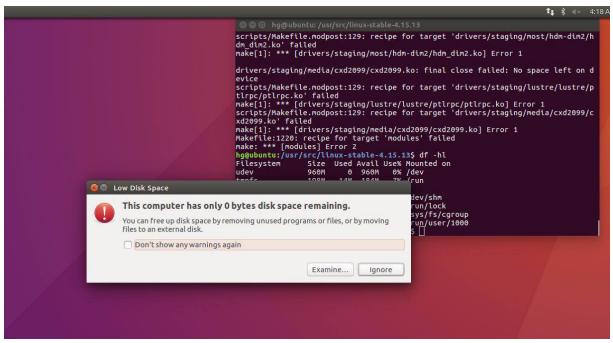

图 12: 硬盘空间不足

解决方案为，在vmware增大硬盘空间分配之后，使用Gparted工具修改分区。

3 总结及反思

通过这次项目实现我进一步熟悉了ubuntu系统，常用终端命令和vim编辑操作，通过实践理解了系统调用是如何具体实现的。