



《操作系统》课第四次实验报告

学院:	软件学院
姓名:	郁万祥
学号:	2013852
邮箱:	yuwanxiang0114@163.com
时间:	2022.10.7

0. 开篇感言

经过了几节实验课的摧残，终于渐渐对 ubuntu 以及 linux 内核的本质熟络起来。1、对于 ubuntu 中的命令行操作其实有点类似于函数的调用，通过内置的一些库直接进行某些操作。2、linux 内核更像是用 C 语言编写的库，定义了一些可以控制、调配系统硬件设施的功能，并且提供了相应的接口，使得用户可以直接使用。理解了这些，回过头来看看 OS 的整体原理脉络，是要比刚开始清晰许多了的。

1. 实验题目

添加系统调用 Add a New System Call



2. 实验目标

添加系统调用并且使用 syscall 进行调用

3. 原理方法

一般情况下，用户进程是不能访问内核的。它既不能访问内核所在的内存空间，也不能调用内核中的函数。系统调用是一个例外。其原理是(1)进程先用适当的值填充寄存器，(2)然后调用一个特殊的指令，(3)这个指令会让用户程序跳转到一个事先定义好的内核中的一个位置。(4) 进程可以跳转到的固定的内核位置。这个过程检查系统调用号，这个号码告诉内核进程请求哪种服务。然后，它查看系统调用表(sys_call_table)找到所调用的内核函数入口地址。接着，就调用函数，等返回后，做一些系统检查，最后返回到进程。

4. 具体步骤

1、在 syscall.h 头文件当中添加 asmlinkage long sys_schello(void);

```
1271
1272
1273 /*
1274  * Not a real system call, but a placeholder for syscalls which are
1275  * not implemented -- see kernel/sys_ni.c
1276  */
1277 asmlinkage long sys_ni_syscall(void);
1278 asmlinkage long sys_schello(void);
1279 #endif /* CONFIG_ARCH_HAS_SYSCALL_WRAPPER */
1280
1281
1282 /*
1283  * Kernel code should not call syscalls (i.e., sys_xyzzyz())
1284  * directly.
1285  * Instead, use one of the functions which work equivalently, such
1286  * as
1287  * the ksys_xyzzyz() functions prototyped below.
1288  */
1289 ssize_t ksys_write(unsigned int fd, const char __user *buf, size_t
count);
1290 int ksys_fchown(unsigned int fd, uid_t user, gid_t group);
1291 ssize_t ksys_read(unsigned int fd, char __user *buf, size_t count);
1292 void ksys_sync(void);
1293 int ksys_unshare(unsigned long unshare_flags);
1294 int ksys_setsid(void);
1295 int ksys_sync_file_range(int fd, loff_t offset, loff_t nbytes,
unsigned int flags);
1296 ssize_t ksys_pread64(unsigned int fd, char __user *buf, size_t
count,
loff_t pos);
1297 ssize_t ksys_pwrite64(unsigned int fd, const char __user *buf,
```



2、在 kernal/sys.c 中添加自己的服务函数

```
SYSCALL_DEFINE0(schello)
{
    printk("Hello new system call schello!ywx2013852\n");
    return 0;
}
```

```
944 /* Inread id - the internal kernel pid */
945 SYSCALL_DEFINE0(gettid)
946 {
947     return task_pid_vnr(current);
948 }
949 SYSCALL_DEFINE0(schello)
950 {
951     printk("Hello new system call schello!ywx2013852\n");
952     return 0;
953 }
954 /*
955  * Accessing ->real_parent is not SMP-safe, it could
956  * change from under us. However, we can use a stale
957  * value of ->real_parent under rcu_read_lock(), see
958  * release_task()->call_rcu(delayed_put_task_struct).
959  */
960 SYSCALL_DEFINE0(getppid)
961 {
962     int pid;
963
964     rcu_read_lock();
965     pid = task_tgid_vnr(rcu_dereference(current->real_parent));
966     rcu_read_unlock();
967
968     return pid;
969 }
970
971 SYSCALL_DEFINE0(getuid)
972 {
973     /* Only we change this so SMP safe */
974     return from_kuid_munged(current_user_ns(), current_uid());
975 }
```

3、在 arch/x86/entry/syscalls/syscall_64.tbl 中添加系统调用号，添加系统调用号和系统调用函数的对应关系。



《操作系统》课程实验报告

```
10月7日 14:32
syscalls_64.tbl
~/linux-5.19.10/arch/x86/entry/syscalls
保存(S)

366 442 common mount_setattr sys_mount_setattr
367 443 common quotactl_fd sys_quotactl_fd
368 444 common landlock_create_ruleset sys_landlock_create_ruleset
369 445 common landlock_add_rule sys_landlock_add_rule
370 446 common landlock_restrict_self sys_landlock_restrict_self
371 447 common memfd_secret sys_memfd_secret
372 448 common process_mrelease sys_process_mrelease
373 449 common futex_waitv sys_futex_waitv
374 450 common set_mempolicy_home_node sys_set_mempolicy_home_node
375 460 common schello sys_schello
376 #
377 # Due to a historical design error, certain syscalls are numbered
    differently
378 # in x32 as compared to native x86_64. These syscalls have numbers
    512-547.
379 # Do not add new syscalls to this range. Numbers 548 and above are
    available
380 # for non-x32 use.
381 #
382 512 x32 rt_sigaction compat_sys_rt_sigaction
383 513 x32 rt_sigreturn compat_sys_x32_rt_sigreturn
384 514 x32 ioctl compat_sys_ioctl
385 515 x32 readv sys_readv
386 516 x32 writev sys_writev
387 517 x32 recvfrom compat_sys_recvfrom
388 518 x32 sendmsg compat_sys_sendmsg
389 519 x32 recvmsg compat_sys_recvmsg
390 520 x32 execve compat_sys_execve
391 521 x32 ptrace compat_sys_ptrace
392 522 x32 rt_sigpending compat_sys_rt_sigpending

纯文本 制表符宽度: 8 第 375 行, 第 1 列 插入
```

4、重新编译内核

```
ywx2013852@ywx2013852-virtual-machine: /usr/src/linux
ywx2013852@ywx2013852-virtual-machine: /usr/src/linux$ make -j5
SYSHDR arch/x86/include/generated/uapi/asm/unistd_32.h
SYSHDR arch/x86/include/generated/uapi/asm/unistd_64.h
SYSHDR arch/x86/include/generated/uapi/asm/unistd_x32.h
HOSTCC scripts/basic/fixdep
SYSTBL arch/x86/include/generated/asm/syscalls_32.h
SYSHDR arch/x86/include/generated/asm/unistd_32_ia32.h
SYSHDR arch/x86/include/generated/asm/unistd_64_x32.h
SYSTBL arch/x86/include/generated/asm/syscalls_64.h
HYPERCALLS arch/x86/include/generated/asm/xen-hypercalls.h
DESCEND objtool
HOSTCC arch/x86/tools/relocs_64.o
HOSTCC arch/x86/tools/relocs_32.o
HOSTCC arch/x86/tools/relocs_common.o
HOSTCC /home/ywx2013852/linux-5.19.10/tools/objtool/fixdep.o
HOSTCC scripts/genksyms/genksyms.o
HOSTLD /home/ywx2013852/linux-5.19.10/tools/objtool/fixdep-in.o
YACC scripts/genksyms/parse.tab.[ch]
LINK /home/ywx2013852/linux-5.19.10/tools/objtool/fixdep
LEX scripts/genksyms/lex.lex.c
CC /home/ywx2013852/linux-5.19.10/tools/objtool/exec_cmd.o
HOSTCC scripts/genksyms/parse.tab.o
CC /home/ywx2013852/linux-5.19.10/tools/objtool/arch/x86/special.
```

5、重新启动后，检查内核版本，编写 demo 进行系统调用，进行测试：

内核版本检查：



```
ywx2013852@ywx2013852-virtual-machine: ~  
ywx2013852@ywx2013852-virtual-machine:~$ uname -a  
Linux ywx2013852-virtual-machine 5.19.10 #2 SMP PREEMPT_DYNAMIC Fri Oct  
7 14:47:29 CST 2022 x86_64 x86_64 x86_64 GNU/Linux  
ywx2013852@ywx2013852-virtual-machine:~$
```

系统调用 c 文件编译：

```
ywx2013852@ywx2013852-virtual-machine: ~/Documents  
ywx2013852@ywx2013852-virtual-machine:~/Documents$ gcc -o testsc testsc  
ello.c  
ywx2013852@ywx2013852-virtual-machine:~/Documents$ sudo dmesg -c  
[sudo] ywx2013852 的密码:  
[ 0.000000] Linux version 5.19.10 (ywx2013852@ywx2013852-virtual-mach  
ine) (gcc (Ubuntu 11.2.0-19ubuntu1) 11.2.0, GNU ld (GNU Binutils for Ubu  
ntu) 2.38) #2 SMP PREEMPT_DYNAMIC Fri Oct 7 14:47:29 CST 2022  
[ 0.000000] Command line: BOOT_IMAGE=/boot/vmlinuz-5.19.10 root=UUID=  
ada09b56-eb7f-4dd2-bb25-5eff5969f6be ro find_preseed=/preseed.cfg auto n  
oprompt priority=critical locale=en_US quiet splash  
[ 0.000000] KERNEL supported cpus:  
[ 0.000000] Intel GenuineIntel  
[ 0.000000] AMD AuthenticAMD  
[ 0.000000] Hygon HygonGenuine  
[ 0.000000] Centaur CentaurHauls  
[ 0.000000] zhaoxin Shanghai  
[ 0.000000] x86/fpu: Supporting XSAVE feature 0x001: 'x87 floating po  
int registers'  
[ 0.000000] x86/fpu: Supporting XSAVE feature 0x002: 'SSE registers'  
[ 0.000000] x86/fpu: Supporting XSAVE feature 0x004: 'AVX registers'  
[ 0.000000] x86/fpu: xstate_offset[2]: 576, xstate_sizes[2]: 256  
[ 0.000000] x86/fpu: Enabled xstate features 0x7, context size is 832  
bytes, using 'compacted' format.  
[ 0.000000] signal: max sigframe size: 1776
```

调用结果：



```
yxw2013852@yxw2013852-virtual-machine: ~/Documents
yxw2013852@yxw2013852-virtual-machine:~/Documents$ gcc -o testsc testschello.c
yxw2013852@yxw2013852-virtual-machine:~/Documents$ sudo dmesg -C
[sudo] yxw2013852 的密码:
yxw2013852@yxw2013852-virtual-machine:~/Documents$ ./testsc
ok! run dmesg | grep hello in terminal!
yxw2013852@yxw2013852-virtual-machine:~/Documents$ sudo dmesg | grep schello
[ 157.875813] Hello new system call schello!yxw2013852
yxw2013852@yxw2013852-virtual-machine:~/Documents$
```

5. 总结心得

其实操作系统说到底还是系统软件程序，所谓 linux 内核，就是一些用 C 语言编写的能够调用内存等硬件设备的一些程序，所以，我们通过修改内核中的用 C 语言编写的程序，就可以增加或者删除（最好不要）一些我们希望能够达到直接访问内核的操作，其实这就是所谓的系统调用，那么系统调用有什么样的好处呢？系统调用大的好处：提供了统一的接口，比如读取数据，API 就不必理会数据存储的物理介质。保护了 OS 的稳定，因为系统调用、异常和中断是外界进入内核的仅有通道，这就保证了不同进程对内核空间的操作是可知并可控的，这为 OS 多任务调度和虚拟内存实现提供了基础。



6. 参考资料

源代码

testschello.c:

```
#include<unistd.h>

#include<sys/syscall.h>

#include<sys/types.h>

#include<stdio.h>

#define _NR_schello 460

int main(int argc,char *argv[])
{
    syscall(_NR_schello);

    printf("ok! run dmesg | grep hello in terminal!\n");

    return 0;
}
```