

《操作系统原理》实验报告

姓名	黄浩岩	学号		专业班级		时间	2020.11.17
----	-----	----	--	------	--	----	------------

一、实验目的

- (1) 理解并应用操作系统生成的概念和过程；
- (2) 理解并应用操作系统操作界面，系统调用概念；

二、实验内容

- (1) 在 Ubuntu 下载剪和编译 Linux 内核，并启用新内核。（其他发行版本也可以）
- (2) 在 Ubuntu 下为 Linux 内核增加 3 个新的系统调用，并启用新的内核，并编写应用程序测试。（其他发行版本也可以）
- (3) 在 Windows 下，编写“算命大师.bat”批处理程序，输入出生年月日，输出属相和星座。
- (4) 在 Linux 下，编写“算命大师”脚本程序，输入出生年月日，输出属相和星座。

三、实验过程

1. 在 Ubuntu 下载剪、编译并启用 Linux 新内核

(1) 编译环境

操作系统：Ubuntu 18.04，内核：Linux 5.4.0-48-generic

```
hhy@hhy-virtual-machine:~$ lsb_release -a
No LSB modules are available.
Distributor ID: Ubuntu
Description:    Ubuntu 18.04.5 LTS
Release:        18.04
Codename:       bionic
hhy@hhy-virtual-machine:~$ uname -a
Linux hhy-virtual-machine 5.4.0-48-generic #52~18.04.1-Ubuntu SMP Thu Sep 10 12:50:22 UTC 2020 x86_64 x86_64 x86_64 GNU/Linux
hhy@hhy-virtual-machine:~$ uname -srn
Linux 5.4.0-48-generic x86_64
```

(2) 下载解压内核源码

① 在 Linux 官网 <https://www.kernel.org/> 下载内核 5.4.70 版本源码压缩包

② 解压源码并移至目录/usr/src/

```
$ tar xf linux-5.4.70.tar.xz
```

```
$ sudo mv linux-5.4.70 /usr/src/
```

(3) 安装编译依赖

```
$ sudo apt install make
```

```
$ sudo apt-get install -y build-essential
```

```
$ sudo apt install libncurses5-dev
```

```
$ sudo apt install flex
```

```
$ sudo apt install bison
```

```
$ sudo apt-get install libssl-dev
```

```
$ sudo apt-get install libelf-dev
```

(3) 编译安装内核

① 进入图形界面配置内核

```
$ make menuconfig
```

② 清除之前编译生成的文件

```
$ make clean
```

③ 编译新内核源码

```
$ sudo make -j8
```

④ 安装内核模块

```
$ sudo make modules_install -j8
```

⑤ 安装内核

```
$ sudo make install
```

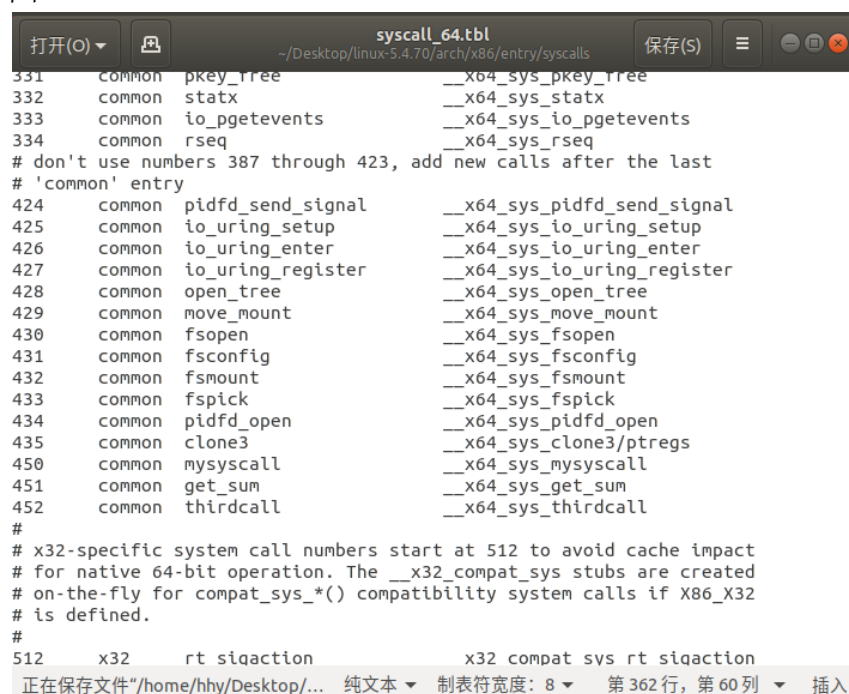
(4) 重启系统

```
$ reboot
```

2. 在 Ubuntu 添加新系统调用

(1) 增加系统调用号

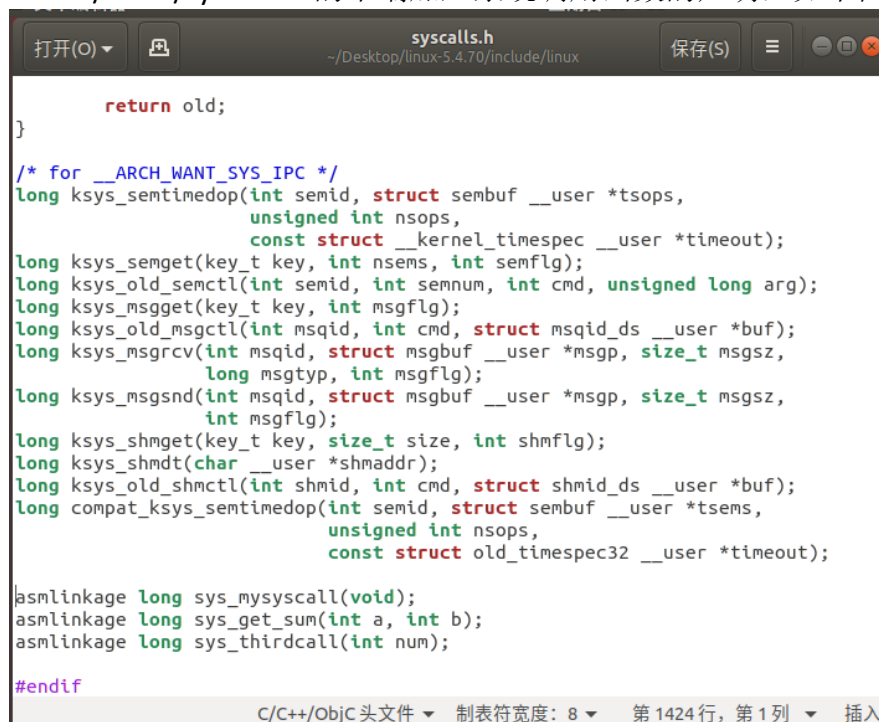
在 Linux 源码目录/arch/x86/entry/syscalls/syscall_64.tbl 中添加自己的系统调用函数和系统调用编号。此处添加了调用号为 450、451 和 452 的三个系统调用，如下图。



```
331 common pkey_free __x64_sys_pkey_free
332 common statx __x64_sys_statx
333 common io_pgetevents __x64_sys_io_pgetevents
334 common rseq __x64_sys_rseq
# don't use numbers 387 through 423, add new calls after the last
# 'common' entry
424 common pidfd_send_signal __x64_sys_pidfd_send_signal
425 common io_uring_setup __x64_sys_io_uring_setup
426 common io_uring_enter __x64_sys_io_uring_enter
427 common io_uring_register __x64_sys_io_uring_register
428 common open_tree __x64_sys_open_tree
429 common move_mount __x64_sys_move_mount
430 common fsopen __x64_sys_fsopen
431 common fsconfig __x64_sys_fsconfig
432 common fsmount __x64_sys_fsmount
433 common fspick __x64_sys_fspick
434 common pidfd_open __x64_sys_pidfd_open
435 common clone3 __x64_sys_clone3/ptregs
450 common mysyscall __x64_sys_mysyscall
451 common get_sum __x64_sys_get_sum
452 common thirdcall __x64_sys_thirdcall
#
# x32-specific system call numbers start at 512 to avoid cache impact
# for native 64-bit operation. The __x32_compat_sys stubs are created
# on-the-fly for compat_sys_*(()) compatibility system calls if X86_X32
# is defined.
#
512 x32 rt siaction x32 compat sys rt siaction
```

(2) 声明系统调用函数

在/include/Linux/syscalls.h 的末端加入系统调用函数的声明，如下图。



```
return old;
}

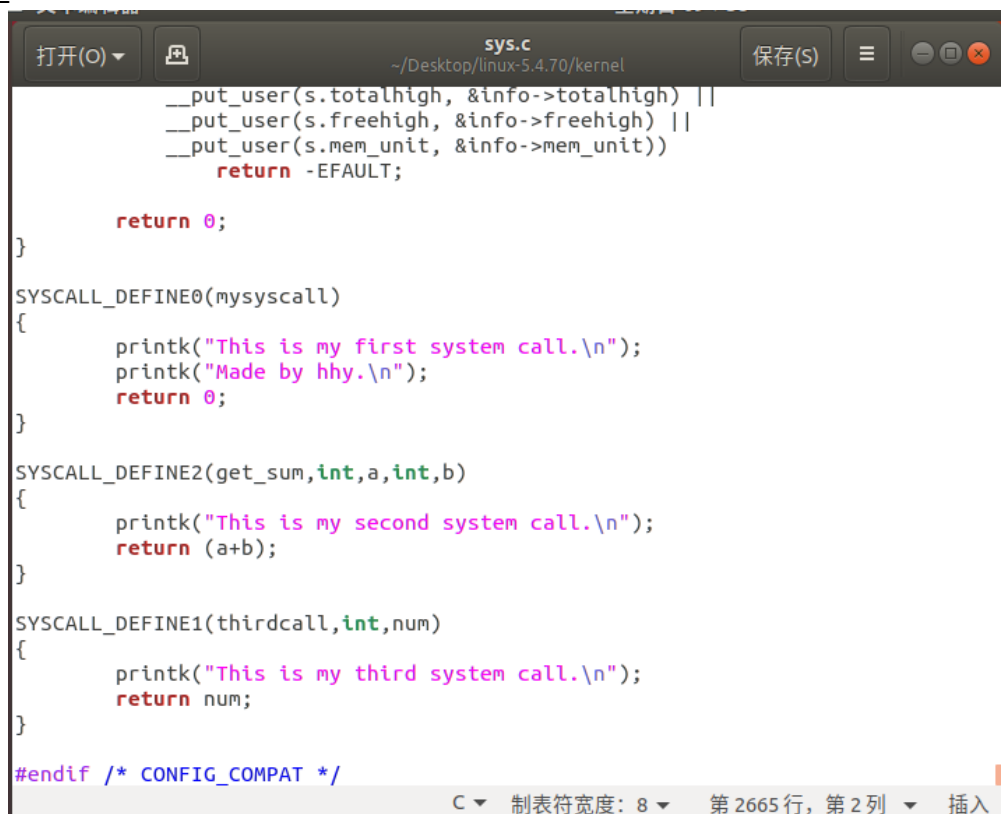
/* for __ARCH_WANT_SYS_IPC */
long ksys_semtimeop(int semid, struct sembuf __user *tsops,
    unsigned int nsops,
    const struct __kernel_timespec __user *timeout);
long ksys_semget(key_t key, int nsems, int semflg);
long ksys_old_semctl(int semid, int semnum, int cmd, unsigned long arg);
long ksys_msgget(key_t key, int msgflg);
long ksys_old_msgctl(int msqid, int cmd, struct msqid_ds __user *buf);
long ksys_msgrcv(int msqid, struct msgbuf __user *msgp, size_t msgsz,
    long msgtyp, int msgflg);
long ksys_msgsnd(int msqid, struct msgbuf __user *msgp, size_t msgsz,
    int msgflg);
long ksys_shmget(key_t key, size_t size, int shmflg);
long ksys_shmdt(char __user *shmaddr);
long ksys_old_shmctl(int shmid, int cmd, struct shmid_ds __user *buf);
long compat_ksys_semtimeop(int semid, struct sembuf __user *tsems,
    unsigned int nsops,
    const struct old_timespec32 __user *timeout);

asmlinkage long sys_mysyscall(void);
asmlinkage long sys_get_sum(int a, int b);
asmlinkage long sys_thirdcall(int num);

#endif
```

(3) 实现系统调用函数

在/kernel/sys.c 的末尾加入实现系统调用函数 sys_mycall、sys_get_sum 和 sys_thirdcall，如下图。



```
__put_user(s.totalhigh, &info->totalhigh) ||
__put_user(s.freehigh, &info->freehigh) ||
__put_user(s.mem_unit, &info->mem_unit))
    return -EFAULT;

    return 0;
}

SYSCALL_DEFINE0(mysyscall)
{
    printk("This is my first system call.\n");
    printk("Made by hhy.\n");
    return 0;
}

SYSCALL_DEFINE2(get_sum, int, a, int, b)
{
    printk("This is my second system call.\n");
    return (a+b);
}

SYSCALL_DEFINE1(thirdcall, int, num)
{
    printk("This is my third system call.\n");
    return num;
}

#endif /* CONFIG_COMPAT */
```

(4) 编译安装内核

对添加系统调用后的内核进行编译安装。代码如下：

```
$ make menuconfig
```

```
$ make clean
```

```
$ sudo make -j8
```

```
$ sudo make modules_install -j8
```

```
$ sudo make install
```

```
$ reboot
```

(5) 测试系统调用

① 创建一个带有系统的 C 文件 test.c 进行测试，代码如下：

```
#include <stdio.h>
```

```
#include <unistd.h>

int main(){

    int a=2,b=3,c=10;

    syscall(450);    //第 1 个系统调用

    printf("%d\n",syscall(451,a,b));    //第 2 个系统调用

    printf("%d\n",syscall(452,c));    //第 3 个系统调用

    return 0;

}
```

② 编译并执行该测试代码

```
$ gcc -o test test.c
```

```
$ ./test
```

③ 查看内核缓冲区信息 \$ sudo dmesg

3. Windows 下“算命大师”批处理程序

代码如下：

```
@echo off

echo=

echo  ++++++++

echo  +      算命大师      +

echo  +                      by:hhy +

echo  ++++++++

:start

echo=
```

::输入生日

set /p birthday=请输入出生年月日(例如 20010521),按 q 退出:

:: /p 用于接收输入

REM echo %birthday%

:: %%用于引用变量

if /i %birthday%==q exit

:: /i 表示不区分大小写

::计算年月日

set /a year=%birthday%/10000

:: set /a 表示计算表达式

set /a month=%birthday%/100%%100

set /a day=(%birthday%)%%100

REM echo %year%, %month%, %day%

::判断输入合法性

if %year% equ 0 goto error

if %month% equ 0 goto error

if %day% equ 0 goto error

if %month% gtr 12 goto error

::判断闰年

set leap=0

set /a tmp1=(%year%)%%400

set /a tmp2=(%year%)%%100

set /a tmp3=(%year%)%%4

if %tmp1% equ 0 (

set leap=1

) else if %tmp2% neq 0 if %tmp3% equ 0 (

set leap=1

)

::判断日期合法性

if %month% equ 1 goto checkDay31

if %month% equ 3 goto checkDay31

if %month% equ 5 goto checkDay31

if %month% equ 7 goto checkDay31

if %month% equ 8 goto checkDay31

if %month% equ 10 goto checkDay31

if %month% equ 12 goto checkDay31

if %month% equ 4 goto checkDay30

if %month% equ 6 goto checkDay30

if %month% equ 9 goto checkDay30

```
if %month% equ 11 goto checkDay30
```

```
if %month% equ 2 (
```

```
    if %leap% equ 0 (
```

```
        if %day% gtr 28 goto error
```

```
    ) else if %day% gtr 29 goto error
```

```
)
```

```
:next
```

```
::计算生肖
```

```
set /a animal=(%year%)%%12
```

```
if %animal% equ 0 (
```

```
    echo 生肖: 猴
```

```
) else if %animal% equ 1 (
```

```
    echo 生肖: 鸡
```

```
) else if %animal% equ 2 (
```

```
    echo 生肖: 狗
```

```
) else if %animal% equ 3 (
```

```
    echo 生肖: 猪
```

```
) else if %animal% equ 4 (
```

```
    echo 生肖: 鼠
```

```
) else if %animal% equ 5 (
```

```
    echo 生肖: 牛
```



```
) else if %animal% equ 6 (
```

```
    echo 生肖: 虎
```

```
) else if %animal% equ 7 (
```

```
    echo 生肖: 兔
```

```
) else if %animal% equ 8 (
```

```
    echo 生肖: 龙
```

```
) else if %animal% equ 9 (
```

```
    echo 生肖: 蛇
```

```
) else if %animal% equ 10 (
```

```
    echo 生肖: 马
```

```
) else if %animal% equ 11 (
```

```
    echo 生肖: 羊
```

```
)
```

```
::计算星座
```

```
set /a star=(%birthday%)%10000
```

```
REM echo %star%
```

```
if %star% leq 119 (
```

```
    echo 星座: 摩羯座
```

```
) else if %star% leq 218 (
```

```
    echo 星座: 水瓶座
```

```
) else if %star% leq 320 (
```

```
    echo 星座: 双鱼座
```

```
) else if %star% leq 419 (  
    echo 星座: 白羊座  
) else if %star% leq 520 (  
    echo 星座: 金牛座  
) else if %star% leq 621 (  
    echo 星座: 双子座  
) else if %star% leq 722 (  
    echo 星座: 巨蟹座  
) else if %star% leq 822 (  
    echo 星座: 狮子座  
) else if %star% leq 922 (  
    echo 星座: 处女座  
) else if %star% leq 1023 (  
    echo 星座: 天秤座  
) else if %star% leq 1122 (  
    echo 星座: 天蝎座  
) else if %star% leq 1221 (  
    echo 星座: 射手座  
) else (  
    echo 星座: 摩羯座  
)  
  
goto start
```

::日期合法性检查

:checkDay31

if %day% gtr 31 goto error

goto next

:checkDay30

if %day% gtr 30 goto error

goto next

::错误处理

:error

echo 输入有误,请重新输入

goto start

4. Linux 下“算命大师”脚本程序

代码如下：

错误输出函数

error()

{

 echo "输入有误,请重新输入"

}

函数定义需放在使用前

```
echo ""
```

```
echo "      ++++++"
```

```
echo "      +      算命大师      +"
```

```
echo "      +              by:hhy +"
```

```
echo "      ++++++"
```

```
# 生肖数组
```

```
animals=("猴" "鸡" "狗" "猪" "鼠" "牛" "虎" "兔" "龙" "蛇" "马" "羊")
```

```
# 空格为分隔符
```

```
# 主函数
```

```
main()
```

```
{
```

```
    echo ""
```

```
    # 输入生日
```

```
    read -p "请输入出生年月日(例如 20010521),按 q 退出:" birthday
```

```
    # 按 q 退出
```

```
    if (($birthday == "q"));then
```

```
        exit 0
```

```
    fi
```

```
    year=${birthday/10000} # 注意[后不能有空格
```

```
    month=${birthday/100%100}
```

```

day=${birthday%100]

#echo $year,$month,$day

# 判断输入合法性

if ((year == 0) || ((month == 0) || ((day == 0) || ((month > 12));then

    error

    return

fi

# 判断闰年

leap=0

tmp1=${year%400]

tmp2=${year%100]

tmp3=${year%4]

if ((tmp1==0) || ((tmp2!=0) && ((tmp3==0)) ;then

    leap=1

fi

# 判断日期合法性

flag=1

for i in {1,3,5,7,8,10,12}; do

    if ((month==i));then

        if ((day>31));then

            error

            return

```

```
        fi

        flag=0

        break

    fi

done

if ((flag));then

    if ((month!=2));then

        if((day>30)); then

            error

            return

        fi

        elif((leap==0));then

            if((day>28));then

                error

                return

            fi

            elif ((day>29));then

                error

                return

            fi

        fi

    fi

# 输出生肖
```

```
animal=${year%12}

echo "生肖: ${animals[animal]}"

# 输出星座

star=${birthday%10000}

if ((star<=119));then

    echo "星座: 摩羯座"

elif ((star<=218));then

    echo "星座: 水瓶座"

elif ((star<=320));then

    echo "星座: 双鱼座"

elif ((star<=419));then

    echo "星座: 白羊座"

elif ((star<=520));then

    echo "星座: 金牛座"

elif ((star<=621));then

    echo "星座: 双子座"

elif ((star<=722));then

    echo "星座: 巨蟹座"

elif ((star<=822));then

    echo "星座: 狮子座"

elif ((star<=922));then

    echo "星座: 处女座"
```

```
elif ((star<=1023));then

    echo "星座: 天秤座"

elif ((star<=1122));then

    echo "星座: 天蝎座"

elif ((star<=1221));then

    echo "星座: 射手座"

else

    echo "星座: 摩羯座"

fi

}
```

```
while true
```

```
do
```

```
    main
```

```
done
```

四、实验结果

1. 在 Ubuntu 下载、编译并启用 Linux 新内核




```

hhy@hhy-virtual-machine:~$ ./fortuneMaster.sh
+++++
+      算命大师      +
+             by:hhy +
+++++
请输入出生年月日(例如20010521),按q退出:20020515
生肖: 马
星座: 金牛座
请输入出生年月日(例如20010521),按q退出:19990110
生肖: 兔
星座: 摩羯座
请输入出生年月日(例如20010521),按q退出:20010229
输入有误,请重新输入
请输入出生年月日(例如20010521),按q退出:20000336
输入有误,请重新输入
请输入出生年月日(例如20010521),按q退出:20050607
生肖: 鸡
星座: 双子座
请输入出生年月日(例如20010521),按q退出:q
hhy@hhy-virtual-machine:~$

```

五、体会

1. 学会了如何编译安装 Linux 内核
2. 学会了如何为 Linux 内核添加系统调用并进行调用。

(1) 其中，对于较新版本的 Linux 内核，添加系统调用的函数源码时需要使用宏定义“SYSCALL_DEFINE(x, mycall, ...)”定义（x 表示系统调用的参数个数，...为传递的参数类型和参数名），而不是“asm linkage long sys_mycall(...)”定义，否则编译时会产生错误。

(2) 对于系统调用中“printk”的输出并不是直接显示在控制台中，而是需要通过命令“sudo dmesg”进行查看。

3. 学习并使用了 Windows 批处理程序和 Linux shell 脚本程序。

简单了解到两种脚本程序的异同：比如两种脚本程序都类似的形式，包括使用“echo”进行输出，都有 if-else 的流程控制等；但两种基本程序在许多语法规则上也有不同，比如 bat 脚本使用 set 指令设置变量，%%进行变量引用，不支持数组类型，if 条件中也不能使用符合语句等，而 shell 脚本变量可直接新建变量，\$进行变量引用，同时支持数组类型，if 条件中可以使用与或等构成的符号逻辑表达式，支持函数调用等。总体上个人感觉 shell 脚本要比 bat 脚本语

法规则更全面，使用起来也更加方便，更接近现在常用的一些编程语言。