

实验报告

课程名称： 操作系统

任课教师： 何永忠

学生姓名： 战泓夷

学生学号： 16281022

专业班级： 安全 1601

学院名称： 计算机与信息技术学院

2019 年 3 月 10 日

实验一：操作系统初步

一、（系统调用实验）了解系统调用不同的封装形式。

1. 参考下列网址中的程序。阅读分别运行用 API 接口函数 `getpid()` 直接调用和汇编中断调用两种方式调用 Linux 操作系统的同一个系统调用 `getpid` 的程序(请问 `getpid` 的系统调用号是多少？linux 系统调用的中断向量号是多少？)。

Getpid 获取进程标志号

```
zhy@zhy-virtual-machine:~$ gcc -o getpid getpid.c
zhy@zhy-virtual-machine:~$ ls
examples.desktop  getpid.c  模板  图片  下载  桌面
getpid           公共的  视频  文档  音乐
zhy@zhy-virtual-machine:~$ ./getpid
3573
zhy@zhy-virtual-machine:~$ touch getpid1.c
zhy@zhy-virtual-machine:~$ getdit getpid1.c
未找到 'getdit' 命令，您要输入的是否是：
命令 'gedit' 来自于包 'gedit' (main)
getdit: 未找到命令
zhy@zhy-virtual-machine:~$ gedit getpid.c
zhy@zhy-virtual-machine:~$ gedit getpid1.c
zhy@zhy-virtual-machine:~$ gcc -o getpid1 getpid1.c
zhy@zhy-virtual-machine:~$ ./getpid1
4184
```

答：getpid 的系统调用号是 20

linux 系统调用的中断向量号是 0x80

2. 上机完成习题 1.13

（1）用 c 语言

程序代码：

```
#include <stdio.h>
int main()
{
    printf("hello world\n");
    return 0;
}
```

结果：

```
zhy@zhy-virtual-machine:~$ gedit helloworld1.c
zhy@zhy-virtual-machine:~$ gcc -o helloworld1 helloworld1.c
zhy@zhy-virtual-machine:~$ ./helloworld1
hello world
```

（2）用汇编语言

程序代码：

```
.section .data                # 数据段声明
    msg: .string "Hello, world!\\n" # 要输出的字符串
    len= . - msg             # 字符串长度
.section .text                # 代码段声明
.global _start               # 指定入口函数

_start:                      # 在屏幕上显示一个字符串
    movl $len, %edx # 参数三：字符串长度
    movl $msg, %ecx # 参数二：要显示的字符串
    movl $1, %ebx  # 参数一：文件描述符(stdout)
    movl $4, %eax  # 系统调用号(sys_write)
    int $0x80      # 调用内核功能

# 退出程序
    movl $0, %ebx  # 参数一：退出代码
    movl $1, %eax  # 系统调用号(sys_exit)
    int $0x80      # 调用内核功能
```

结果：

```
zhy@zhy-virtual-machine:~$ gedit hello.asm
zhy@zhy-virtual-machine:~$ as hello.asm -o hello.o
zhy@zhy-virtual-machine:~$ ld hello.o -o hello
zhy@zhy-virtual-machine:~$ ./hello
Hello, world!\\nzhy@zhy-virtual-machine:~$
```

3. 阅读 pintos 操作系统源代码，画出系统调用实现的流程图。



二、（并发实验）根据以下代码完成下面的实验。

1、编译运行该程序（cpu.c），观察输出结果，说明程序功能。

（编译命令： `gcc -o cpu cpu.c -Wall`）（执行命令： `./cpu`）

结果：

```
zhy@zhy-virtual-machine:~$ ls
a.out  cpu.c  getpid  getpid1.c  公共的  视频  文档  音乐
cpu    examples.desktop  getpid1  getpid.c  模板  图片  下载  桌面
zhy@zhy-virtual-machine:~$ ./cpu
usage:cpu <string>
```

程序功能：

`argc` 为参数个数， `*argv[]` 为参数，当 `argc` 等于 1 或者大于 2 时，输出 `usage: cpu <string>`，将第二个参数赋值给字符串 `str`（这里的字符相当于进程），用 `while` 循环输出字符（`sleep(1)`；就是休眠 1 秒）

2、再次按下面的运行并观察结果：执行命令： `./cpu A & ; ./cpu B & ; ./cpu C & ; ./cpu D &` 程序 `cpu` 运行了几次？他们运行的顺序有何特点和规律？请结合操作系统的特征进行解释。

结果：

```
zhy@zhy-virtual-machine:~$ ./cpu A & ./cpu B & ./cpu C & ./cpu D &
[1] 6633
[2] 6634
[3] 6635
[4] 6636
zhy@zhy-virtual-machine:~$ B
C
D
A
B
C
D
A
B
C
D
A
B
```

答：这个程序体现了 `cpu` 的并发过程，当程序有 `ABCD` 四个进程同时进行，程序 `cpu` 运行了 4 次，他们的特点和规律：宏观上多个程序在同时进行，但是在微观上这些程序只能是分时的交替进行。体现了并发性的多道特点，成批性是它的特点，内存中有若干个程序在不同道作业，用户不能干预，同时多道批处理系统中的 I/O 设备具有异步性，更无法判断它开始结束的规律性。

三、（内存分配实验）根据以下代码完成实验。

1. 阅读并编译运行该程序(mem.c)，观察输出结果，说明程序功能。(命令： gcc -o mem mem.c -Wall)

结果：

```
zhy@zhy-virtual-machine:~$ gedit mem.c
zhy@zhy-virtual-machine:~$ gcc -o mem mem.c -Wall
zhy@zhy-virtual-machine:~$ ./mem
(3654)address pointed to by p:0x25b8010
(3654) p:1
(3654) p:2
(3654) p:3
(3654) p:4
(3654) p:5
(3654) p:6
(3654) p:7
```

程序功能：

首先给 p 分配空间，使用断言，当 p 不为空，则输出 p 的地址和进程号，给 p 赋值为 0，用 while 循环加 1，并输出当前 p 的值和进程号。

3. 再次按下面的命令运行并观察结果。两个分别运行的程序分配的内存地址是否相同？是否共享同一块物理内存区域？为什么？命令： ./mem & ./mem &

结果：

```
zhy@zhy-virtual-machine:~$ ./mem & ./mem &
[1] 3694
[2] 3695
zhy@zhy-virtual-machine:~$ (3695)address pointed to by p:0x20e7010
(3694)address pointed to by p:0x1ea1010
(3694) p:1
(3695) p:1
(3694) p:2
(3695) p:2
(3694) p:3
(3695) p:3
(3694) p:4
(3695) p:4
(3694) p:5
(3695) p:5
(3695) p:6
(3694) p:6
(3694) p:7
(3695) p:7
(3695) p:8
(3694) p:8
(3694) p:9
(3695) p:9
(3694) p:10
```

答：如图可以看出，两个分别运行的程序分配的内存地址不相同，由于内存地址

不同，我认为它们不共享同一块物理内存。从进程的不同性上面来说，有可能对应相同的虚拟地址，但是真实的物理地址不会相同，这种虚拟内存也不保证不同进程的相互隔离，错误程序不会干扰别的正确的进程。

四、（共享的问题）根据以下代码完成实验。

1、阅读并编译运行该程序，观察输出结果，说明程序功能。（编译命令：`gcc -o thread thread.c -Wall -pthread`）（执行命令 1：`./thread 1000`）

答：`worker` 是一个计数功能的函数，`argc` 为参数个数，`*argv[]` 为参数，当 `argc` 等于 1 或者大于 2 时，输出 `usage: cpu <string>`，把命令行参数中文件名后的第二个字符串转化为整数，赋值给 `loops`，创建两个线程，观察操作系统中，两个线程同时运行对共享变量的影响。

2、尝试其他输入参数并执行，并总结执行结果的有何规律？你能尝试解释它吗？（例如执行命令 2：`./thread 100000`）（或者其他参数。）

答：由图我们可以看出，共享变量的存在，使得每个线程都可以使用它，因此执行的结果是参数的二倍（因为是两个线程）。

3、提示：哪些变量是各个线程共享的，线程并发执行时访问共享变量会不会导致意想不到的问题。

答：可能会导致，由于每个 `cpu` 线程都有一定的时间限制，当达到这个限制时，超过的次数就不会计入，因此导致可能输出的 `final value` 比实际只要小。

当输入参数比较小，一个 `CPU` 的核心足够处理，就是单核 `CPU` 运行多线程，由于每个核心有内存锁机制，所以计算结果没有问题。但是当输入参数比较大时，就会无法满足，进而导致出现读取脏数据的情况。

结果：

```
zhy@zhy-virtual-machine:~$ gedit thread.c
zhy@zhy-virtual-machine:~$ gcc -o thread thread.c -Wall -pthread
zhy@zhy-virtual-machine:~$ ./thread 1000
Initial value : 0
Final value : 2000
zhy@zhy-virtual-machine:~$ ./thread 100000
Initial value : 0
Final value : 200000
zhy@zhy-virtual-machine:~$
```

五、实验心得

通过本次实验，我了解了操作系统给应用成提供服务的方式-系统调用，通过 c 语言和汇编编程，理解了系统调用与普通函数调用的不同之处，同时通过执行不同的程序，更直观的体会到在操作系统的学习中并发、内存分配、共享的特点，对以后更深入地学习打下坚实的基础。