

# 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。

要求：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？）。2、上机完成习题 1.13。3、阅读 pintos 操作系统源代码，画出系统调用实现的流程图。

1、

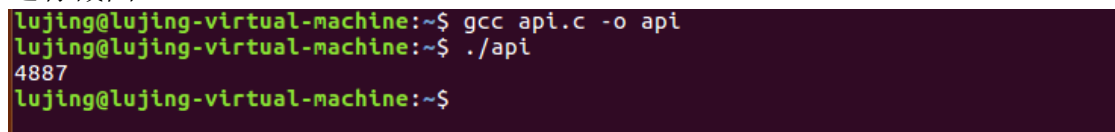
API 接口：

代码：

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

```
int main()
{
    pid_t pid;
    pid=getpid();
    printf("%d\n",pid);
    return 0;
}
```

运行截图：



```
lujing@lujing-virtual-machine:~$ gcc api.c -o api
lujing@lujing-virtual-machine:~$ ./api
4887
lujing@lujing-virtual-machine:~$
```

汇编中断调用：

代码：

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

```
int main()
{
    pid_t pid;
    asm volatile(
        "mov $0,%%ebx\n\t"
        "mov $0x14,%%eax\n\t"
        "int $0x80\n\t"
        "mov %%eax,%0\n\t"
        : "=m" (pid)
    );
    printf("%d\n",pid);
    return 0;
}
```

运行截图：

```
lujing@lujing-virtual-machine:~$ gcc ZD.c -o ZD
lujing@lujing-virtual-machine:~$ ./ZD
4879
lujing@lujing-virtual-machine:~$
```

分析:

getpid 的系统调用号是 20, linux 系统调用的中断向量号是 0x80;

遇到问题及解决方法:

getpid 的系统调用号是 20 是针对 32 位 linux 系统, 但是我是在 64 位系统中运行的, 运行结果正确。如果改成 64 位系统的系统调用号 39, 则嵌入汇编代码的调用输出结果错误。

2、

Linux 系统调用的 c 函数形式:

代码:

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

```
int main() {
printf("Hello World!\n");
return 0;
}
```

运行截图:

```
lujing@lujing-virtual-machine:~$ gcc hello.c -o hello
lujing@lujing-virtual-machine:~$ ./hello
Hello World!
lujing@lujing-virtual-machine:~$
```

Linux 系统调用的汇编代码:

代码:

```
.section .data
string:
    .ascii "hello world!\n"
.section .text
.globl _start
_start:
mov $4,%eax
mov $1,%ebx
mov $string,%ecx
mov $13,%edx
int $0x80
mov $1,%eax
mov $0,%ebx
int $0x80
```

运行截图:

```

lujing@lujing-virtual-machine:~$ as -o hello.o hello.s
lujing@lujing-virtual-machine:~$ ld -o hello hello.o
lujing@lujing-virtual-machine:~$ ./hello
hello world!
lujing@lujing-virtual-machine:~$ ./hello
hello world!
lujing@lujing-virtual-machine:~$ echo $?
0
lujing@lujing-virtual-machine:~$

```

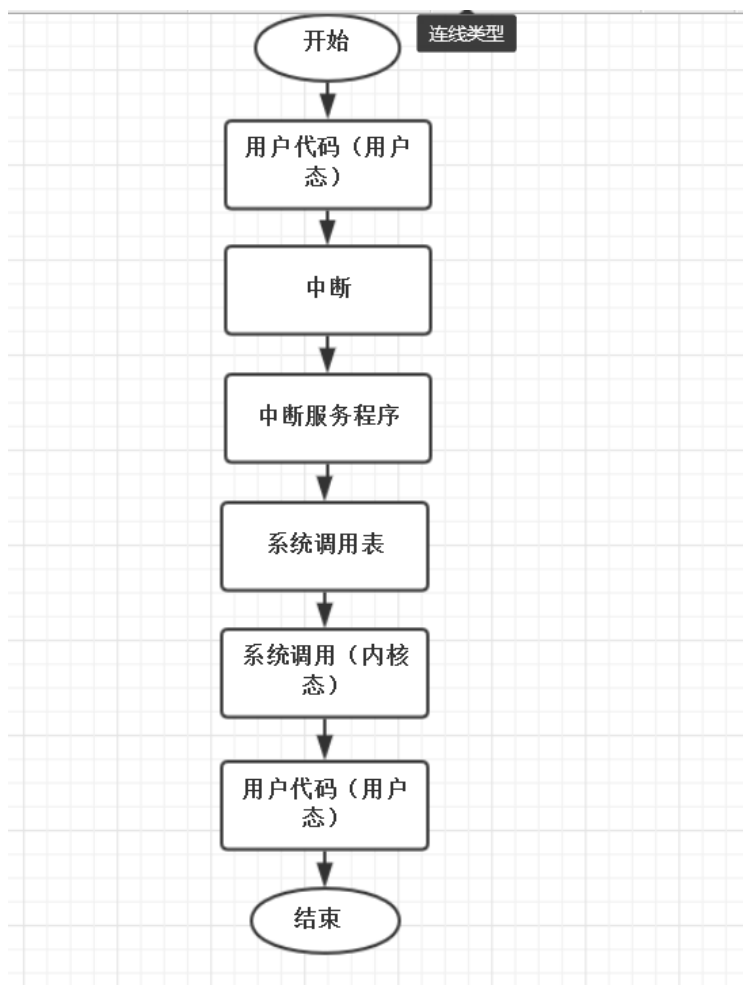
### 分析:

第一句的作用是数据段申明，第二句的作用是定义要输出的字符串，第三句的作用是代码段声明，第四句的作用是指定入口函数，第五句的作用是在屏幕上显示一个字符串，第六句的作用是系统调号 `sys_write`，第七句是参数一，作为文件描述符 `stdout`，第八句是参数二，表示要显示的字符串“hello world!” 第九句是参数三，表示字符串长度是 13，第十句是调用内核功能，最后三句的功能是退出程序，即第十一句是退出代码，第十二句是系统调用号 `sys_exit`，第十三句是调用内核功能。

### 遇到问题及解决方法:

其实一开始并不会写汇编代码，经过一些时间上网学习才明白了一些基本用法。`int` 调用属于系统调用，依赖于内核，是操作系统的一个入口点，`call` 属于函数调用，一个普通功能函数调用，属于过程调用，调用时开销比较小也没有堆栈交换。

3、



二、（并发实验）根据以下代码完成下面的实验。

要求：

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）

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。

1、

运行截图：

```
lujing@lujing-virtual-machine:~$ gcc -o cpu cpu.c -Wall
lujing@lujing-virtual-machine:~$ ./cpu
usage: cpu <string>
lujing@lujing-virtual-machine:~$
```

分析：

程序输出结果为 usage: cpu <string>, 此时程序运行的是 if 语句部分，因为 argc 表示输入的数据个数，此时输入个数不等于 2，即 argc 不等于 2，所以运行 if 语句部分，遇到 exit(1) 时结束程序。

```
if (argc != 2) {
    fprintf(stderr, "usage: cpu <string>\n");
    exit(1);
}
```

2、

运行截图：

```
lujing@lujing-virtual-machine:~$ gcc -o cpu cpu.c
lujing@lujing-virtual-machine:~$ ./cpu A & ./cpu B & ./cpu C & ./cpu D &
[1] 8016
[2] 8017
[3] 8018
[4] 8019
lujing@lujing-virtual-machine:~$ C
B
A
D

```

分析：

当输入 A & 时，输入个数等于 2，即 argc=2，故程序执行 while(1) 部分代码，此时输出 argv[1] 中所存的指 A、B、C、D 类似。cpu 运行过程中，4 个进程同时运行，但由于进程的异步性——推进相互独立、速度不可预知，故每次出现 A、B、C、D 的速度不可知，A、B、C、D 出现的顺序不可确定。

遇到问题及解决方法：

编译时 spin 不可用，解决办法是删除 spin，在其位置添加 for 循环的空循环，以保证循环间隔时间。

三、（内存分配实验）根据以下代码完成实验。

要求：

2、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)

2、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 & 1、

```
lujing@lujing-virtual-machine:~$ gcc -o mem mem.c -Wall
lujing@lujing-virtual-machine:~$ ./mem
(8285) address pointed to by p: 0x14f3010
(8285) p: 1
(8285) p: 2
(8285) p: 3
(8285) p: 4
(8285) p: 5
(8285) p: 6
(8285) p: 7
(8285) p: 8
(8285) p: 9
(8285) p: 10
(8285) p: 11
(8285) p: 12
(8285) p: 13
(8285) p: 14
(8285) p: 15
```

分析：

首先给指针 p 分配一块大小为 sizeof(int) 的空间，从该空间的首地址 0x14f3010 开始依次存入数据，由 \*p=0, \*p=\*p+1 可知，第一个数据为 1，其后每次的下一个数据是上一个数据加 1。进程号为 8285。

2、

```
lujing@lujing-virtual-machine: ~
lujing@lujing-virtual-machine:~$ ./mem & ./mem &
[1] 8338
[2] 8339
lujing@lujing-virtual-machine:~$ (8339) address pointed to by p: 0xc75010
(8338) address pointed to by p: 0x1173010
(8338) p: 1
(8339) p: 1
(8339) p: 2
(8338) p: 2
(8339) p: 3
(8338) p: 3
(8339) p: 4
(8338) p: 4
```

分析：

由上图可见，两个分别运行的程序分配的内存地址不相同，一个进程的内存地址是 0xc75010，另一个进程的内存地址为 0x1173010，是共享同一块物理内存区域，因为进程具有并发性——共存于内存、宏观同时运行。

四、（共享的问题）根据以下代码完成实验。

要求：

- 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）
- 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。）

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。

1、

```
lujing@lujing-virtual-machine:~$ ./thread 1000
Initial value : 0
Final value : 2000
lujing@lujing-virtual-machine:~$
```

分析：

当输入 1000 时，输出结果为 2000，初始数据为 0，由此可知，该程序运行的结果是将输入数据翻一倍，即输入数据乘以 2 等于输出数据。

2、

```
lujing@lujing-virtual-machine:~$ ./thread 100
Initial value : 0
Final value : 200
lujing@lujing-virtual-machine:~$ ./thread 1000
Initial value : 0
Final value : 2000
lujing@lujing-virtual-machine:~$ ./thread 10000
Initial value : 0
Final value : 20000
lujing@lujing-virtual-machine:~$ ./thread 100000
Initial value : 0
Final value : 200000
lujing@lujing-virtual-machine:~$ ./thread 600
Initial value : 0
Final value : 1200
lujing@lujing-virtual-machine:~$ ./thread 134
Initial value : 0
Final value : 268
lujing@lujing-virtual-machine:~$
```

分析：

执行结果为输出结果为输入结果的两倍，因为 p1, p2 每次运行都各执行依次，故总次数是输入的两倍。