

# 实验一：操作系统初步

班级：安全 1601

姓名：李婧祎

学号：16281134

(注意：本次所有实验都在 Linux 中完成)

## 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？)。

中断向量号 80H，系统调用号 14H。

- 2、上机完成习题 1.13。

C:

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

```
#include<stdlib.h>
```

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

```
Hello worldljj@ljj-virtual-machine:~/文档/os/lab1$ gcc -o 1 1.c  
ljj@ljj-virtual-machine:~/文档/os/lab1$ ./1  
Hello world
```

汇编:

.LC0:

```
.string "Hello world"  
.text  
.globl main  
.type main, @function
```

main:

.LFB2:

```
.cfi_startproc  
pushq %rbp  
.cfi_def_cfa_offset 16  
.cfi_offset 6, -16  
movq %rsp, %rbp  
.cfi_def_cfa_register 6
```

```

movl    $.LC0, %edi
call    puts
movl    $0, %eax
popq    %rbp
.cfi_def_cfa 7, 8
ret
.cfi_endproc

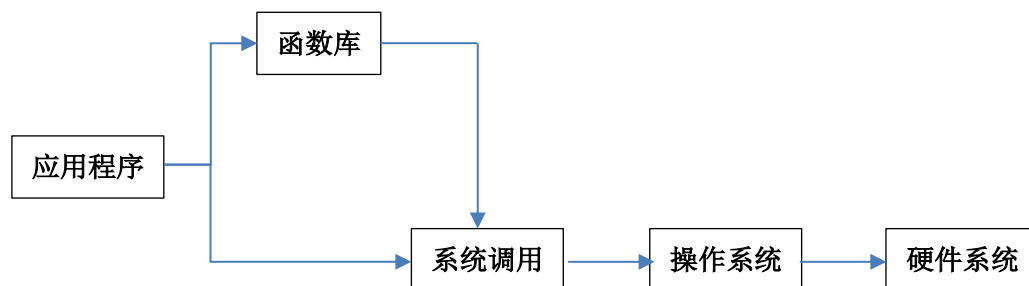
```

```

ljj@ljj-virtual-machine:~/文档/os/lab1$ gcc -o 1 1.s
ljj@ljj-virtual-machine:~/文档/os/lab1$ ./1
Hello world

```

3、阅读 pintos 操作系统源代码，画出系统调用实现的流程图。



```

1 #include <stdio.h>
2 #include <unistd.h>
3
4 int main()
5 {
6     pid_t pid;
7
8     pid = getpid();
9     printf("%d\n", pid);
10
11     return 0;
12 }

```

```

1 #include <stdio.h>
2 #include <unistd.h>
3
4 int main()
5 {
6     pid_t pid;
7
8     asm volatile(
9         "mov $0,%%ebx\n\t"
10        "mov $0x14,%%eax\n\t"
11        "int $0x80\n\t"
12        "mov %%eax,%0\n\t"
13        : "=m" (pid)
14        );
15
16     printf("%d\n", pid);
17     return 0;
18 }

```

## 二、（并发实验）根据以下代码完成下面的实验。

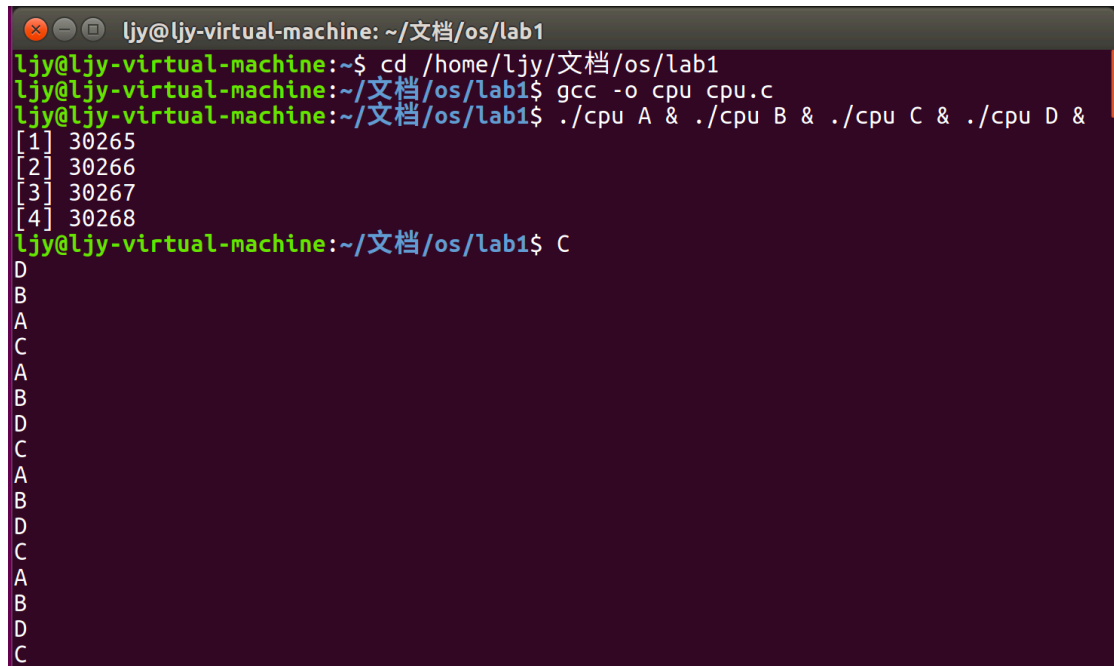
要求：

- 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`）

程序功能：每隔 1s 输出传入参数，若传入参数正确，则输出；若传入参数不正确，则输出错误提示 `usage: cpu <string>`。

- 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。



```
ljj@ljj-virtual-machine: ~/文档/os/lab1
ljj@ljj-virtual-machine:~$ cd /home/ljj/文档/os/lab1
ljj@ljj-virtual-machine:~/文档/os/lab1$ gcc -o cpu cpu.c
ljj@ljj-virtual-machine:~/文档/os/lab1$ ./cpu A & ./cpu B & ./cpu C & ./cpu D &
[1] 30265
[2] 30266
[3] 30267
[4] 30268
ljj@ljj-virtual-machine:~/文档/os/lab1$ C
D
B
A
C
A
B
D
C
A
B
D
C
A
B
D
C
```

解释：程序 `cpu` 运行了 4 次。由以上运行结果可知，程序的运行顺序没有规律：字符的输出顺序并非命令的执行顺序，每一组的输出顺序也不相同。操作系统对 4 个进程进行了线程调度，进程的运行都是并发实现的；在调度过程中等待时长不同，有细微差别，导致了每一组顺序不同。

## 三、（内存分配实验）根据以下代码完成实验。

要求：

- 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`）

```

ljj@ljj-virtual-machine:~/文档/os/lab1$ gcc -o mem mem.c
ljj@ljj-virtual-machine:~/文档/os/lab1$ ./mem
(30964) address pointed to by p: 0x118d010
(30964) p: 1
(30964) p: 2
(30964) p: 3
(30964) p: 4
(30964) p: 5
(30964) p: 6
(30964) p: 7
(30964) p: 8
(30964) p: 9
(30964) p: 10
(30964) p: 11
(30964) p: 12
^C
ljj@ljj-virtual-machine:~/文档/os/lab1$ █

```

程序功能：输出程序 pid 和申请的内存空间地址，并对空间内的数据进行自增输出。

- 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 &

```

ljj@ljj-virtual-machine:~/文档/os/lab1$ ./mem & ./mem &
[1] 30998
[2] 30999
ljj@ljj-virtual-machine:~/文档/os/lab1$ (30999) address pointed to by p: 0x793010
(30998) address pointed to by p: 0x1646010
(30999) p: 1
(30998) p: 1
(30999) p: 2
(30998) p: 2
(30999) p: 3
(30998) p: 3
(30999) p: 4
(30998) p: 4
(30999) p: 5
(30998) p: 5
(30999) p: 6
(30998) p: 6
(30999) p: 7
(30998) p: 7
(30999) p: 8

```

解释：分配的内存地址不相同，但申请的内存地址理论上可以一样：内存地址相对于每个进程，并不是真实的物理地址，所以可以一致。

两程序并不共享同一块内存区域，用户态下只能访问自己内存空间里的地址。

## 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 -pthread -Wall -pthread）（执行命令 1：./thread 1000）

```

ljj@ljj-virtual-machine:~$ cd /home/ljj/文档/os/lab1
ljj@ljj-virtual-machine:~/文档/os/lab1$ gcc -o thread thread.c -pthread
ljj@ljj-virtual-machine:~/文档/os/lab1$ ./thread 1000
Initial value: 0
Final value: 2000

```

程序功能：对于输入的参数 n，程序创建了两个线程，对同一地址内的数进行自增，输

出初始数值和两次自增后的结果。

- 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。）

```
ljl@ljl-virtual-machine:~$ cd /home/ljl/文档/os/lab1
ljl@ljl-virtual-machine:~/文档/os/lab1$ gcc -o thread thread.c -pthread
ljl@ljl-virtual-machine:~/文档/os/lab1$ ./thread 1000
Initial value: 0
Final value: 2000
ljl@ljl-virtual-machine:~/文档/os/lab1$ gcc -o thread thread.c -pthread
ljl@ljl-virtual-machine:~/文档/os/lab1$ ./thread 10000
Initial value: 0
Final value: 20000
ljl@ljl-virtual-machine:~/文档/os/lab1$ ./thread 10000000
Initial value: 0
Final value: 20000000
ljl@ljl-virtual-machine:~/文档/os/lab1$ ./thread 30000000
Initial value: 0
Final value: 60000000
ljl@ljl-virtual-machine:~/文档/os/lab1$ ./thread 30000000
Initial value: 0
Final value: 60000000
ljl@ljl-virtual-machine:~/文档/os/lab1$ ./thread 300000000
Initial value: 0
Final value: 598548942
ljl@ljl-virtual-machine:~/文档/os/lab1$ ./thread 300000000000
Initial value: 0
Final value: 0
ljl@ljl-virtual-machine:~/文档/os/lab1$ ./thread 893649
Initial value: 0
Final value: 1787298
ljl@ljl-virtual-machine:~/文档/os/lab1$ ./thread 80000004
Initial value: 0
Final value: 158076189
```

规律：final value 在输入参数的 1~2 倍之间。

解释：两个线程同时访问可能会导致数据的丢失，导致两次自增表现为一次自增。这种情况小概率发生，所以输入参数  $n$  较小时，没有这种现象发生。（输入参数  $n$  类型为 `int`，故只有 32 位）

- 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。

`counter` 和 `loops`。两个线程同时访问可能会导致数据的丢失，导致两次自增表现为一次自增。