# 实验一：操作系统初步

苏荣天
16281141
安全 1601

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？)。

```
搜索您的计算机
vinh@vinh-virtual-machine:~$ sudo apt-get install build-essential
[sudo] vinh 的密码：
正在读取软件包列表... 完成
正在分析软件包的依赖关系树
正在读取状态信息... 完成
build-essential 已经是最新版 (12.1ubuntu2)。
升级了 0 个软件包，新安装了 0 个软件包，要卸载 0 个软件包，有 295 个软件包未被升级。
vinh@vinh-virtual-machine:~$ sudo apt-get install gcc
正在读取软件包列表... 完成
正在分析软件包的依赖关系树
正在读取状态信息... 完成
gcc 已经是最新版 (4:5.3.1-1ubuntu1)。
升级了 0 个软件包，新安装了 0 个软件包，要卸载 0 个软件包，有 295 个软件包未被升级。
vinh@vinh-virtual-machine:~$ sudo apt-get install g++
正在读取软件包列表... 完成
正在分析软件包的依赖关系树
正在读取状态信息... 完成
g++ 已经是最新版 (4:5.3.1-1ubuntu1)。
升级了 0 个软件包，新安装了 0 个软件包，要卸载 0 个软件包，有 295 个软件包未被升级。
```

检查必要的软件包。

编译两段代码。

打开(O)

```c
#include<stdio.h>
#include<unistd.h>
int main()
{
pid_t pid;
pid=getpid();
printf("%d\n",pid);
return 0;
}
```

打开(O)

```
#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;
}
```

执行结果如下：

```
vinh@vinh-virtual-machine: ~
vinh@vinh-virtual-machine:~$ touch 1.c
vinh@vinh-virtual-machine:~$ gcc 1.c -o 1
vinh@vinh-virtual-machine:~$ ./1
2772
vinh@vinh-virtual-machine:~$ touch 11.c
vinh@vinh-virtual-machine:~$ gcc 11.c -o 11
vinh@vinh-virtual-machine:~$ ./11
2814
vinh@vinh-virtual-machine:~$
```

getpid 的系统调用号是 20，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>
int main()
{
printf("hello world\n");
return 0;
}
```

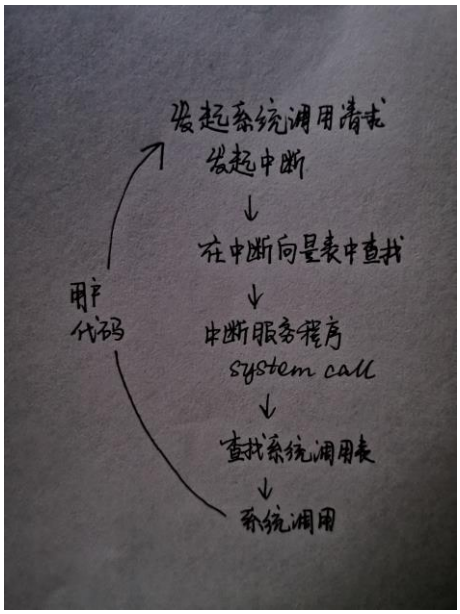
```
vinh@vinh-virtual-machine:~$ touch helloworld.c
vinh@vinh-virtual-machine:~$ gcc helloworld.c -o helloworld
vinh@vinh-virtual-machine:~$ ./helloworld
hello world
vinh@vinh-virtual-machine:~$
```

汇编：

```
        .section        .rodata
.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
.LFE2:
        .size   main, .-main
        .ident  "GCC: (Ubuntu 5.3.1-14ubuntu2) 5.3.1 20160413"
        .section        .note.GNU-stack,"",@progbits
```

```
vinh@vinh-virtual-machine:~$ ./helloworld
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。




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。

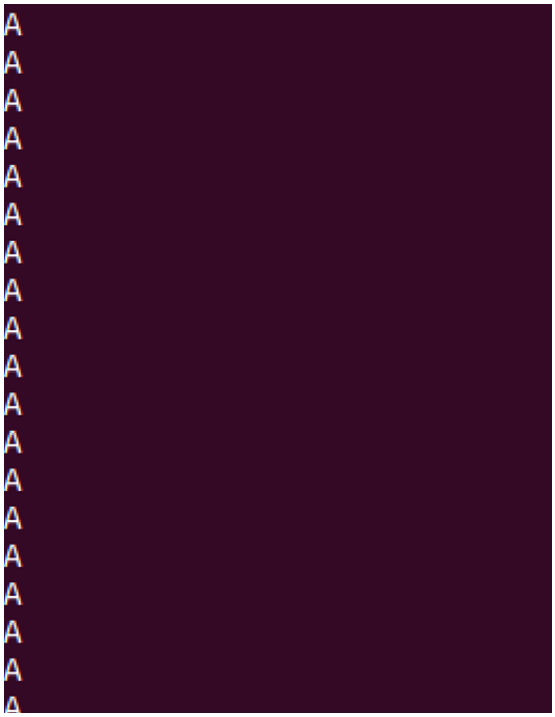
```
#include<stdio.h>
#include<stdlib.h>
#include<sys/time.h>
#include<assert.h>
#include"common.h"

int
main(int argc,char *argv[])
{
if(argc!=2){
fprintf(stderr,"usage:cpu<string>\n");
exit(1);
}
char *str=argv[1];
while(1){
spin(1);
printf("%s\n",str);
}
return 0;
```

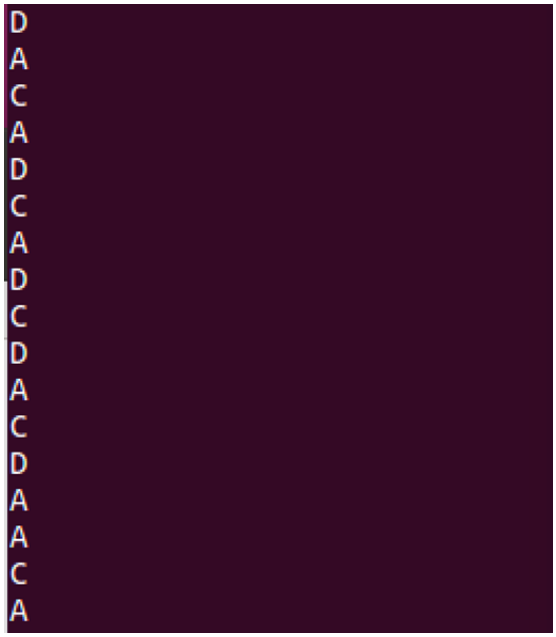
1.程序功能为每隔一秒输出一次参数

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>。

```
vinh@vinh-virtual-machine:~$ ./2
usage: cpu <string>
vinh@vinh-virtual-machine:~$
```

2. 对于 4 个完全相同的程序，CPU 的优先级是相同的，因此会随机地顺序执行。并发是指一个时间段中有几个程序都处于已启动运行到运行完毕之间，且这几个程序都是在同一个处理机上运行，但任一个时刻点上只有一个程序在处理机上运行。




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)

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.

```c
#include<unistd.h>
#include<stdio.h>
#include<stdlib.h>
#include"common.h"

int
main(int argc,char *argv[])
{
int*p=malloc(sizeof(int));//a1
assert(p!=NULL);
printf("(%d)address pointed to by p:%p\n",getpid(),p);//a2
*p=0;//a3
while(1){
spin(1);
*p=*p+1;
Printf("(%d)p:%d\n",getpid(),*p);//a4
}
return 0;
```

```
(4616) p: 1
(4617) p: 1
(4616) p: 2
(4617) p: 2
(4616) p: 3
(4617) p: 3
(4616) p: 4
(4617) p: 4
(4616) p: 5
(4617) p: 5
(4616) p: 6
(4617) p: 6
(4616) p: 7
(4617) p: 7
(4616) p: 8
(4617) p: 8
(4616) p: 9
(4617) p: 9
(4616) p: 10
(4617) p: 10
```

程序功能为分配内存，打印指针 p 指向的内存地址，以及每隔一秒将指针加一后打印指针的相对地址

2. 两次指针 p 指向的地址不相同，不是同一内存区域。

操作系统为不同的指针分配不同的地址，同时为了避免指针之间的冲突，所以不能把不同的指针指向同一内存区域。

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。

```
volatile int counter = 0;
int loops;

void *worker(void *arg) {
        int i;
        for (i = 0; i < loops; i++) {
                counter++;
        }
        return NULL;
}

int main(int argc, char *argv[])
{
        if (argc != 2) {
                fprintf(stderr, "usage: threads <value>\n");
                exit(1);
        }
        loops = atoi(argv[1]);
        pthread_t p1, p2;
        printf("Initial value : %d\n", counter);

        pthread_create(&p1, NULL, worker, NULL);
        pthread_create(&p2, NULL, worker, NULL);
        pthread_join(p1, NULL);
        pthread_join(p2, NULL);
        printf("Final value : %d\n", counter);
        return 0;
}
```

执行结果：

```
vinh@vinh-virtual-machine:~$ gcc -pthread 4.c -o thread
vinh@vinh-virtual-machine:~$ ./4 1000
bash: ./4: 没有那个文件或目录
vinh@vinh-virtual-machine:~$ ./thread 1000
Initial value : 0
Final value : 2000
vinh@vinh-virtual-machine:~$ ./thread 10000
Initial value : 0
Final value : 20000
vinh@vinh-virtual-machine:~$ ./thread 20000
Initial value : 0
Final value : 40000
vinh@vinh-virtual-machine:~$ ./thread 30000
Initial value : 0
Final value : 60000
vinh@vinh-virtual-machine:~$
```

可见终值与初值的差为输入参数的两倍。虽然开了两个线程，但是两个线程均对同一变量进行加一操作，参数即为循环的次数，所以为输入参数的两倍。

3. counter 和 loops 是各个线程共享的。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