

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 #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 }
```

答：

1、系统调用号和操作系统的位数和发行版本有关，其中 Ubuntu 64 位机器上 `getpid` 的系统调用号是 172。一般来说，Linux 内核中 `getpid` 的系统调用号是 39（64 位）或者 20（32 位），这里系统调用号不同的原因是 64 位系统对 32 位系统进行了兼容，其使用 `syscall` 指令来触发而不是 32 位系统时的开放 0x80 搭配系统调用号来实现功能。此程序中，linux 系统调

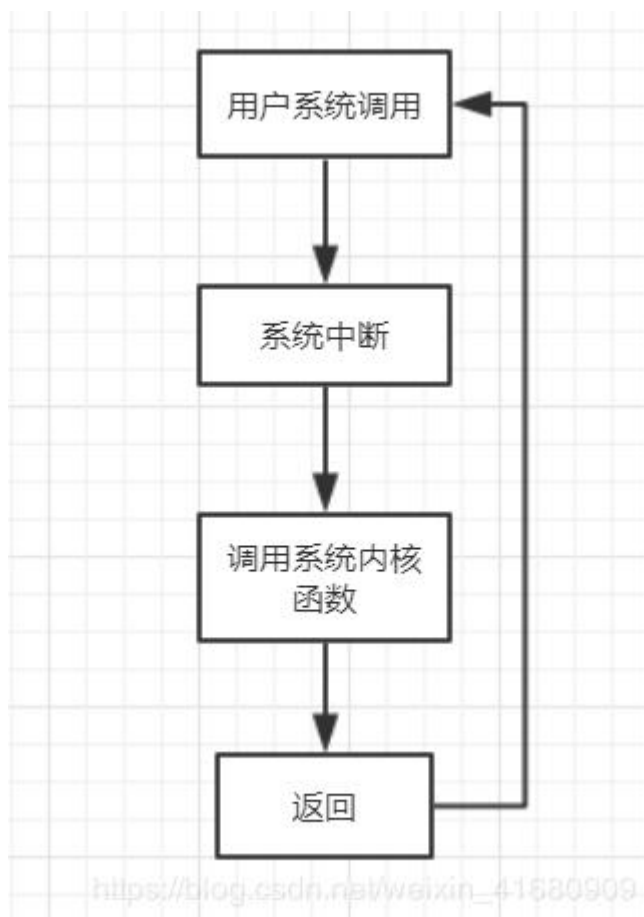
用的中断向量号是 0x80 ，系统调用号位 0x14 （十进制 20）。

2、

```
zhaoyu@zhaoyu-virtual-machine:~/osCode$ gcc -o hello hello.c -Wall
zhaoyu@zhaoyu-virtual-machine:~/osCode$ ./hello
Hello world!
zhaoyu@zhaoyu-virtual-machine:~/osCode$
```

编译运行，屏幕上打印出 Hello World!

3、阅读 pintos 操作系统源代码，系统调用实现的流程图如下。



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 <unistd.h>

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

```

1、答：程序功能是输出给定 argv[1]参数字母

2、答：

程序 cpu 运行 4 次，程序功能是输出给定 argv[1]参数字母，两次输出之间停顿 1s，并换行。输出不按照参数给定的顺序。这是由于程序并发执行失去了程序的封闭性和再现性，程序和机器执行程序的活动不再一一对应。

```

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

```

```

zhaoyu@zhaoyu-virtual-machine: ~/osCode
zhaoyu@zhaoyu-virtual-machine:~/osCode$ ./cpu A & ./cpu B & ./cpu C & ./cpu D
[1] 6636
[2] 6637
[3] 6638
[4] 6639
zhaoyu@zhaoyu-virtual-machine:~/osCode$ A
B
D
C
A
B
D
C
B
A
C
A
B
D

```

三、（内存分配实验）根据以下代码完成实验。

要求：

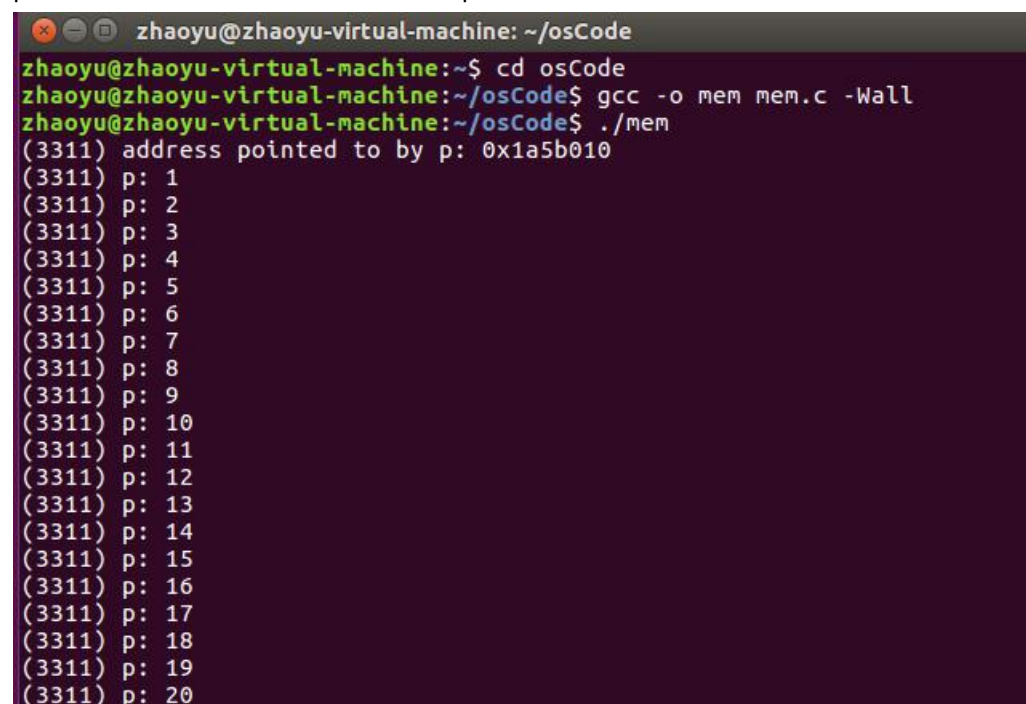
2、阅读并编译运行该程序(mem.c)，观察输出结果，说明程序功能。(命令： gcc -o mem mem.c -Wall)

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 &

```
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <assert.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) {
        sleep(1);
        *p = *p + 1;
        printf("(%d) p: %d\n", getpid(), *p); // a4
    }
    return 0;
}
```

答：

1、该程序功能是创建一个进程，令指针 p 指向该内存空间，并输出该内存的地址，然后令 p 的值累加，依次输出进程识别码和 p 的当前值。运行结果如下图：



```
zhaoyu@zhaoyu-virtual-machine: ~/osCode
zhaoyu@zhaoyu-virtual-machine:~$ cd osCode
zhaoyu@zhaoyu-virtual-machine:~/osCode$ gcc -o mem mem.c -Wall
zhaoyu@zhaoyu-virtual-machine:~/osCode$ ./mem
(3311) address pointed to by p: 0x1a5b010
(3311) p: 1
(3311) p: 2
(3311) p: 3
(3311) p: 4
(3311) p: 5
(3311) p: 6
(3311) p: 7
(3311) p: 8
(3311) p: 9
(3311) p: 10
(3311) p: 11
(3311) p: 12
(3311) p: 13
(3311) p: 14
(3311) p: 15
(3311) p: 16
(3311) p: 17
(3311) p: 18
(3311) p: 19
(3311) p: 20
```

2、

执行命令，发现为两个进程分别分配了两个物理地址，分别为 0x11cb01，0x112f010。两个分别运行的程序分配的内存地址不相同，不共享同一块物理内存区域。运行结果如下图：

```
zhaoyu@zhaoyu-virtual-machine: ~/osCode
zhaoyu@zhaoyu-virtual-machine:~/osCode$ ./mem & ./mem &
[1] 3325
[2] 3326
zhaoyu@zhaoyu-virtual-machine:~/osCode$ (3325) address pointed to by p: 0x11cb010
(3326) address pointed to by p: 0x112f010
(3326) p: 1
(3325) p: 1
(3326) p: 2
(3325) p: 2
(3326) p: 3
(3325) p: 3
(3326) p: 4
(3325) p: 4
(3326) p: 5
(3325) p: 5
(3326) p: 6
(3325) p: 6
(3326) p: 7
(3325) p: 7
(3326) p: 8
(3325) p: 8
(3326) p: 9
(3325) p: 9
```

当关闭了 ASLR 地址空间随机化后再次运行程序，会发现两个分别运行的程序分配的内存地址不相同，但是共享同一块物理内存区域，均为 0x602010，运行结果如下：

```
root@zhaoyu-virtual-machine:~/osCode# gcc -o mem mem.c -Wall
root@zhaoyu-virtual-machine:~/osCode# ./mem & ./mem &
[1] 2576
[2] 2577
root@zhaoyu-virtual-machine:~/osCode# (2576) address pointed to by p: 0x602010
(2577) address pointed to by p: 0x602010
(2576) p: 1
(2577) p: 1
(2577) p: 2
(2576) p: 2
(2576) p: 3
(2577) p: 3
(2577) p: 4
(2576) p: 4
(2576) p: 5
```

https://blog.csdn.net/weixin_4168

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。

```
getpid_C.c

#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

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;
}
```

答：

1、程序的功能是创建两个线程，每个线程内都会将 counter 累加相应的次数，最终输出 counter 的值。

2、执行结果，Initial value 为 0，当输入参数较小时，Final value 为输入参数的两倍。当输入参数较大时，Final value 小于输入参数的两倍。

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

分析：Loop 和 counter 是两个线程共享的。不同线程对同一全局变量操作时，可能会导致错误的结果，是由于在执行一个线程的某一个功能时可能会跳跃到另一个线程，发生读脏数据的情况。此题、一个线程在进行累加操作时，另一个线程读入了旧的 counter 的值，从而使得累加不够输入参数的两倍。