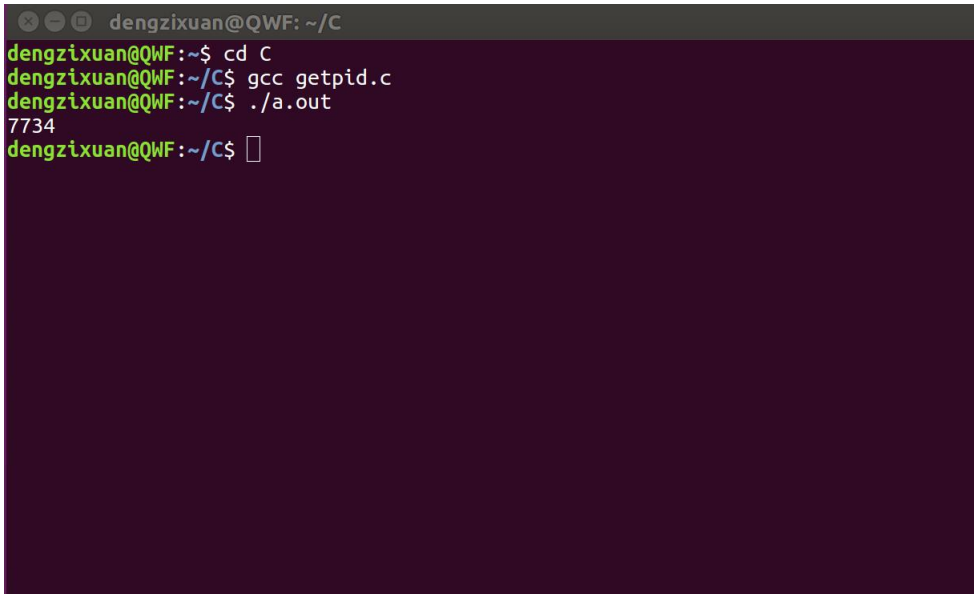
# 实验一：操作系统初步

安全 1601 16281221 邓子轩

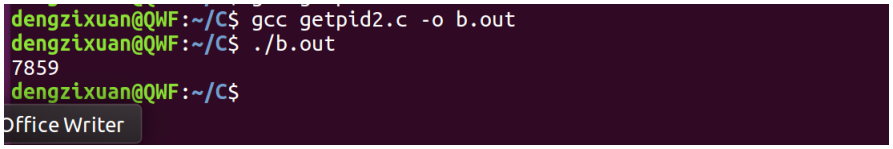
## 一、 系统调用实验

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？)。

直接调用：




汇编终端调用：




getpid 的系统调用号是 7734，系统调用的中断向量号是 7859。

此处两个文件的代码为实验指导书里代码，由于在之后使用虚拟机做实验的时候

导致系统崩了，重装了虚拟机，文件丢失，故此处没有代码截图。

2、上机完成习题 1.13。

直接调用：

```
dengzixuan@QWF:~/C$ gcc 1.13.c -o 13.out
dengzixuan@QWF:~/C$ ./13.out
Hello World!dengzixuan@QWF:~/C$
```

汇编中断调用：

```
#include<stdio.h>

int main()
{
char*msg = "Hello World";
int len = 11;
int result = 0;

asm volatile
(
"movl %2,%%edx;\n\r"
"movl %1,%%ecx;\n\r"
"movl $1,%%ebx;\n\r"
"movl $4,%%eax;\n\r"
"int $0x80"
:"=m"(result)
:"m"(msg),"r"(len)
:"%eax");
return 0;
}
```

```
dengzixuan@QWF:~/C$ gcc 13.2.c -o 13.2.out
dengzixuan@QWF:~/C$ ./13.2.out
Hello Worlddengzixuan@QWF:~/C$
```

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

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）

```
dengzixuan@QWF:~/C$ gcc -o cpu cpu.c
cpu.c: In function ‘main’:
cpu.c:15:2: warning: implicit declaration of function ‘sleep’ [-Wimplicit-functi
on-declaration]
  sleep(1);
  ^
dengzixuan@QWF:~/C$ ./cpu.out
usage: cpu <string>
dengzixuan@QWF:~/C$ ./cpu a
a
a
a
a
a
a
```

程序的功能是不停输出输入的参数。

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。

```
qwf@qwf-virtual-machine:~$ ./cpu A& ./cpu B& ./cpu C& ./cpu D&
[1] 13514
[2] 13515
[3] 13516
[4] 13517
qwf@qwf-virtual-machine:~$ A
B
C
D
B
A
C
D
A
C
D
B
```

运行结果顺序与程序执行顺序不是对应的，并且每轮运行结果的顺序会有改变。并发环境下，由于程序的封闭性被打破，程序与计算不再一一对应，一个程序副本可以有多个计算。

三、　内存分配实验

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)

```
qwf@qwf-virtual-machine: ~
qwf@qwf-virtual-machine:~$ gcc -o mem mem.c -Wall
qwf@qwf-virtual-machine:~$ ./mem
(14512) address pointed to by p: 0x2436010
(14512) p: 1
(14512) p: 2
(14512) p: 3
^C
qwf@qwf-virtual-machine:~$
```

首先，通过 malloc()函数申请了部分内存。然后打印出了内存地址，再将数字 0 赋值给最新申请的内存地址中。最后，每延迟一秒，打印该程序的进程号，该进程号是唯一的，以及循环递增存储在该地址上的值。

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 &

```
qwf@qwf-virtual-machine:~$ ./mem& ./mem&
[1] 15029
[2] 15030
qwf@qwf-virtual-machine:~$ (15029) address pointed to by p: 0xf37010
(15030) address pointed to by p: 0x1337010
(15029) p: 1
(15030) p: 1
(15030) p: 2
(15029) p: 2
(15030) p: 3
(15029) p: 3
^C
qwf@qwf-virtual-machine:~$ (15030) p: 4
(15029) p: 4
(15030) p: 5
(15029) p: 5
(15030) p: 6
```

每个进程都是从相同的地址开始分配内存，独立地去更新该地址的数值的。

操作系统中真正发生的事情是虚拟内存。每个进程都访问它们自己的私有的虚拟地址空间，操作系统将这些虚拟地址空间以某种方式映射到机器的物理内存。一个运行的程序引用的内存不会影响其他进程的地址空间；就正在运行的程序而言，它独自占有所有的物理内存。然而，事实却是物理内存是一个由操作系统管理的共享资源。

四、　共享的问题

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）

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

此程序输出两个 worker 函数循环中共享的计数器的增长次数。

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。）

```
qwf@qwf-virtual-machine:~$ ./thread 10000
Initial value : 0
Final value : 20000
qwf@qwf-virtual-machine:~$ ./thread 100000
Initial value : 0
Final value : 200000
qwf@qwf-virtual-machine:~$ ./thread 1000000
Initial value : 0
Final value : 2000000
qwf@qwf-virtual-machine:~$ ./thread 10000000
Initial value : 0
Final value : 11477167
qwf@qwf-virtual-machine:~$ ./thread 10000000
Initial value : 0
Final value : 13093116
qwf@qwf-virtual-machine:~$
```

不断增大输入的值，直到 10000000 时，输出的结果是 11477167，再次输入得到的结果是 13093116，可看出，当数字足够大时，多次得到的结果是不固定且不正确的。

这是因为，上面程序共享的计数器递增主要分为三条指令，一个是从内存加载计数器的值到寄存器，一个时增加，一个将它存入内存。因为这三条指令不是原子执行的，所以会出现问题。

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。

全局变量是各个线程共享的，线程并发执行时访问共享变量会导致原子性问题，可见性问题，有序性问题等。