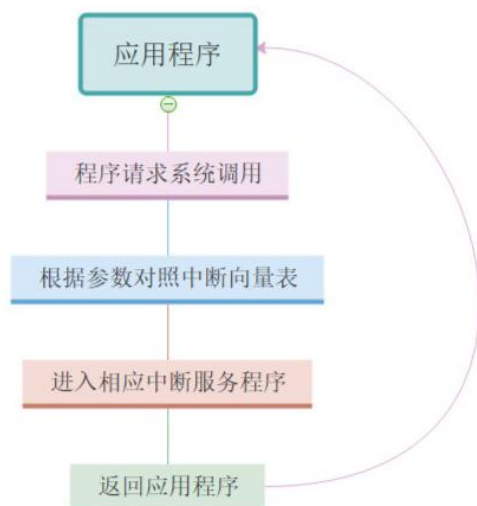


一、（系统调用实验）了解系统调用不同的封装形式。

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. `getpid` 的系统调用号是 14H、中断向量号是 80H。

2.



3.

实验截图：

```
emrick@ubuntu: ~/Desktop
* vim-athena
* vim-athena-py2
* vim-gnome-py2
* vim-gtk
* vim-gtk-py2
* vim-gtk3
* vim-gtk3-py2
* vim-nox
* vim-nox-py2
Try: sudo apt install <selected package>
emrick@ubuntu:~/Desktop$ gcc 1_2.cpp
gcc: error: 1_2.cpp: No such file or directory
gcc: fatal error: no input files
compilation terminated.
emrick@ubuntu:~/Desktop$ gcc 1.2.cpp
/tmp/ccUis0sz.o: In function 'main':
1.2.cpp:(.text+0x1d): undefined reference to 'ox14'
collect2: error: ld returned 1 exit status
emrick@ubuntu:~/Desktop$ gcc 1.2.cpp
emrick@ubuntu:~/Desktop$ ls
1_1.cpp 1.2.cpp a.out
emrick@ubuntu:~/Desktop$ ./a.out
3738
emrick@ubuntu:~/Desktop$
```

The screenshot shows a terminal window with the following commands and output:

- `emrick@ubuntu:~/Desktop$ gcc 1_2.cpp` results in an error: `gcc: error: 1_2.cpp: No such file or directory`.
- `emrick@ubuntu:~/Desktop$ gcc 1.2.cpp` results in a linker error: `collect2: error: ld returned 1 exit status`.
- `emrick@ubuntu:~/Desktop$ gcc 1.2.cpp` successfully compiles the program.
- `emrick@ubuntu:~/Desktop$ ls` shows the files: `1_1.cpp 1.2.cpp a.out`.
- `emrick@ubuntu:~/Desktop$./a.out` outputs the value `3738`.

```
emrick@ubuntu: ~/Desktop
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

emrick@ubuntu:~$ ls
Desktop  Downloads  Music  Public  Videos
Documents  examples.desktop  Pictures  Templates
emrick@ubuntu:~$ cd Desktop
emrick@ubuntu:~/Desktop$ ls
1_1.cpp
emrick@ubuntu:~/Desktop$ gcc 1_1.cpp
emrick@ubuntu:~/Desktop$ ls
1_1.cpp  a.out
emrick@ubuntu:~/Desktop$ ./a.out
3446
emrick@ubuntu:~/Desktop$ ls
1_1.cpp  1.2.cpp  a.out
emrick@ubuntu:~/Desktop$ gcc 1.2.cpp
1.2.cpp: In function 'int main()':
1.2.cpp:16:2: error: expected ';' before 'printf'
printf("%d\n",pid );
^
emrick@ubuntu:~/Desktop$ vim 1_2.cpp
The program 'vim' can be found in the following packages:
* vim
```

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、程序功能：循环输出传入参数到屏幕，如果无参数则输出 usage:cpu<string>。
- 2、运行结果：

```
emrick@ubuntu: ~/Desktop
emrick@ubuntu:~$ ls
Desktop  Downloads  Music  Public  Videos
Documents  examples.desktop  Pictures  Templates
emrick@ubuntu:~$ cd Desktop/
emrick@ubuntu:~/Desktop$ ls
1_1.cpp  1.2.cpp  1.3.c  a.out  cpu
emrick@ubuntu:~/Desktop$ ./cpu A & ./cpu B & ./cpu C & ./cpu D &
[1] 4715
[2] 4716
[3] 4717
[4] 4718
emrick@ubuntu:~/Desktop$ C
D
B
A
D
C
B
A
C
D
B
A
D
```

只要不手动停止，CPU 就一直运行，运行的特点：可以看到程序运行的次序是不确定的，原因可能是 Ubuntu 系统是支持并行的，因此我们的四个程序在 CPU 内同时运行，但速度有快有慢，因此才会有这样的输出结果。

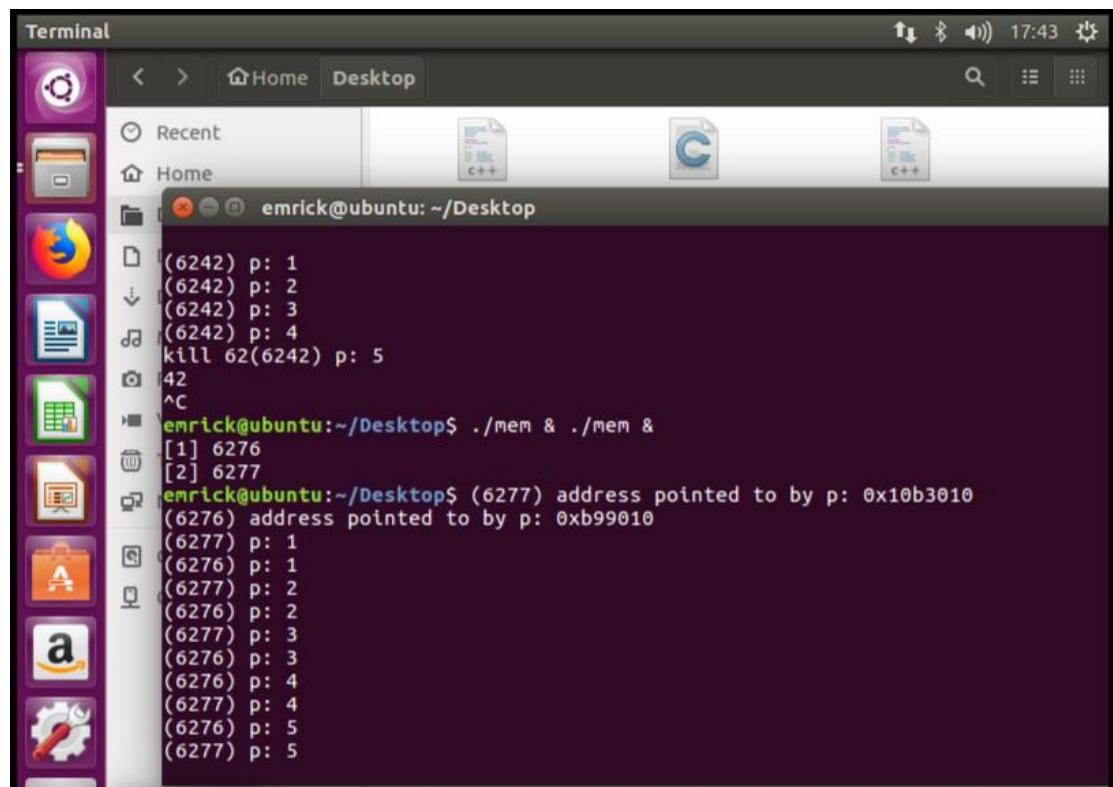
三、（内存分配实验）根据以下代码完成实验。

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、程序的功能是新建进程并打印进程识别码：

输出结果：



```
emrick@ubuntu: ~/Desktop
(6242) p: 1
(6242) p: 2
(6242) p: 3
(6242) p: 4
kill 62(6242) p: 5
42
^C
emrick@ubuntu:~/Desktop$ ./mem & ./mem &
[1] 6276
[2] 6277
emrick@ubuntu:~/Desktop$ (6277) address pointed to by p: 0x10b3010
(6276) address pointed to by p: 0xb99010
(6277) p: 1
(6276) p: 1
(6277) p: 2
(6276) p: 2
(6277) p: 3
(6276) p: 3
(6277) p: 4
(6276) p: 4
(6277) p: 5
(6276) p: 5
(6277) p: 5
```

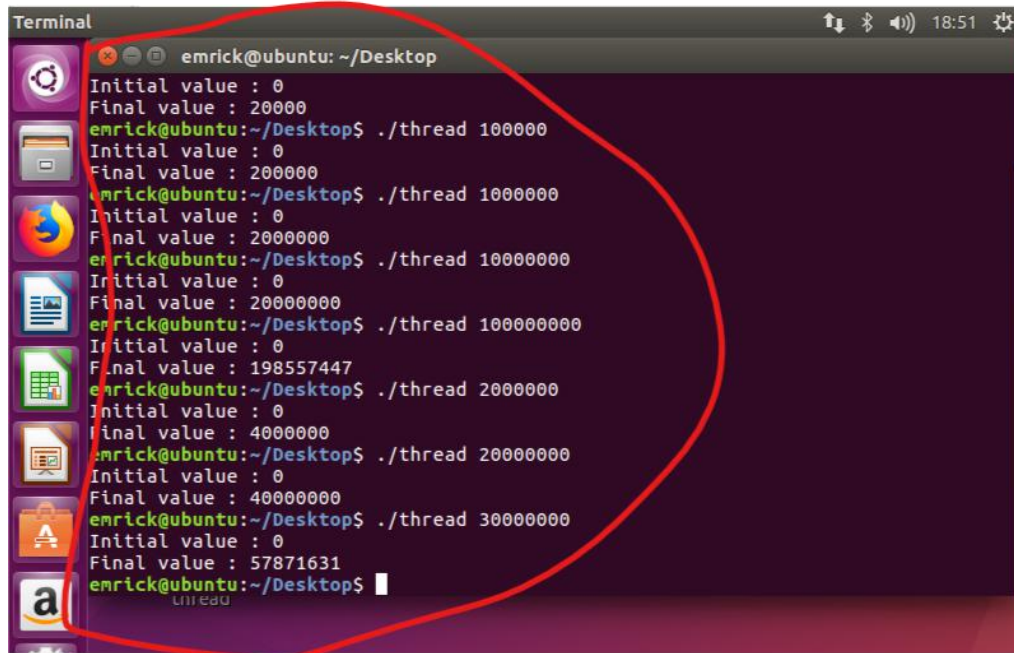
2、内存地址不相同，不共享同一物理内存区域，因为在同一物理内存区的两个进程识别码不可能相同，他们可能是运行在不同内核里面的程序。

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。）

- 1、程序作用：创建两个线程对 **counter** 进行累加操作，并输出初始值和累加后的值：
运行截图：



```
Terminal
emrick@ubuntu: ~/Desktop
Initial value : 0
Final value : 20000
emrick@ubuntu:~/Desktop$ ./thread 100000
Initial value : 0
Final value : 200000
emrick@ubuntu:~/Desktop$ ./thread 1000000
Initial value : 0
Final value : 2000000
emrick@ubuntu:~/Desktop$ ./thread 10000000
Initial value : 0
Final value : 20000000
emrick@ubuntu:~/Desktop$ ./thread 100000000
Initial value : 0
Final value : 198557447
emrick@ubuntu:~/Desktop$ ./thread 2000000
Initial value : 0
Final value : 4000000
emrick@ubuntu:~/Desktop$ ./thread 20000000
Initial value : 0
Final value : 40000000
emrick@ubuntu:~/Desktop$ ./thread 30000000
Initial value : 0
Final value : 57871631
emrick@ubuntu:~/Desktop$
```

- 2、可以看到当参数大到一定程度时，输出的值就不再是传进参数的 2 倍了，可能是因为 CPU 处理的过程时间太长，两个线程的运行发生了时间上的重合，其中一个线程在进行累加操作时，另一个线程读入了过时的共享变量 **counter** 的数据，造成了累加值不足的情况。