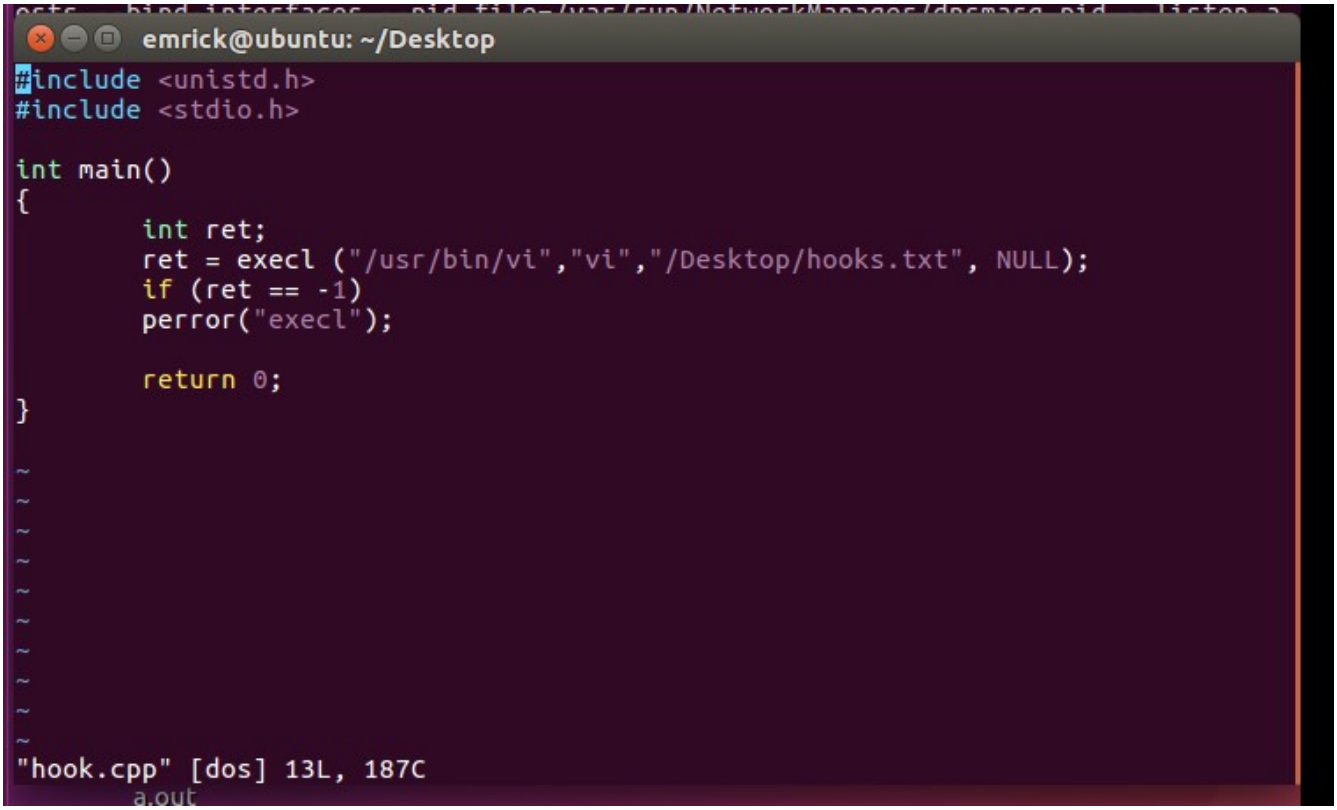
操作系统第二次实验报告

一、打开一个vi进程。通过ps命令以及选择合适的参数，只显示名字为vi的进程。寻找vi进程的父进程，直到init进程为止。记录过程中所有进程的ID和父进程ID。将得到的进程树和由pstree命令的得到的进程树进行比较。

打开进程：



```
emrick@ubuntu: ~/Desktop
emrick@ubuntu:~$ cd D
Desktop/   Documents/ Downloads/
emrick@ubuntu:~$ cd Desktop/
emrick@ubuntu:~/Desktop$ ls
1_1.cpp  1.3.c  4.1.c  ChangeSource.sh  hook.cpp   mem
1.2.cpp  3.1.c  a.out  cpu              hooks.txt  thread
emrick@ubuntu:~/Desktop$ vim hook
hook.cpp   hooks.txt
emrick@ubuntu:~/Desktop$ vim hook.cpp
emrick@ubuntu:~/Desktop$ gcc hook.cpp
emrick@ubuntu:~/Desktop$ ./a.out
```

查找进程并进行父进程的追踪：

```
emrick@ubuntu: ~
emrick@ubuntu:~$ ps -auxc | grep vi$
emrick     2995  0.0  0.4  55596  8508 pts/17   S+   19:25   0:00 vi
emrick@ubuntu:~$ ps -eo pid,ppid,user | grep -w ^2995
emrick@ubuntu:~$ ps -eo pid,ppid,user | grep -w 2995
  2995   2937 emrick
emrick@ubuntu:~$ ps -eo pid,ppid,user | grep -w 2937
  2937   2936 emrick
  2995   2937 emrick
emrick@ubuntu:~$ ps -eo pid,ppid,user | grep -w 2936
  2936   1920 emrick
  2937   2936 emrick
  2953   2936 emrick
emrick@ubuntu:~$ ps -eo pid,ppid,user,command | grep -w 2995
  2995   2937 emrick   vi /Desktop/hooks.txt
  3386   3293 emrick   grep --color=auto -w 2995
emrick@ubuntu:~$ ps -eo pid,ppid,user,command | grep -w 2937
  2937   2936 emrick   -bash
  2995   2937 emrick   vi /Desktop/hooks.txt
  3400   3293 emrick   grep --color=auto -w 2937
emrick@ubuntu:~$ ps -eo pid,ppid,user,command | grep -w 2936
  2936   1920 emrick   tmux
  2937   2936 emrick   -bash
  2953   2936 emrick   -bash
  3404   3293 emrick   grep --color=auto -w 2936
```

Terminal 19:37

emrick@ubuntu: ~

```
emrick@ubuntu:~$ ps -eo pid,ppid,user,command | grep -w 2995
  2995   2937 emrick   vi /Desktop/hooks.txt
  3386   3293 emrick   grep --color=auto -w 2995
emrick@ubuntu:~$ ps -eo pid,ppid,user,command | grep -w 2937
  2937   2936 emrick   -bash
  2995   2937 emrick   vi /Desktop/hooks.txt
  3400   3293 emrick   grep --color=auto -w 2937
emrick@ubuntu:~$ ps -eo pid,ppid,user,command | grep -w 2936
  2936   1920 emrick   tmux
  2937   2936 emrick   -bash
  2953   2936 emrick   -bash
  3404   3293 emrick   grep --color=auto -w 2936
emrick@ubuntu:~$ ps -eo pid,ppid,user,command | grep -w 1920
  1920   1194 emrick   /sbin/upstart --user
  1993   1920 emrick   upstart-udev-bridge --daemon --user
  2000   1920 emrick   dbus-daemon --fork --session --address=unix:abstract=/tmp
/dbus-dxer71XqyZ
  2012   1920 emrick   /usr/lib/x86_64-linux-gnu/hud/window-stack-bridge
  2039   1920 emrick   /usr/bin/ibus-daemon --daemonize --xim --address unix:tmp
dir=/tmp/ibus
  2046   1920 emrick   /usr/lib/x86_64-linux-gnu/bamf/bamfdaemon
  2053   1920 emrick   /usr/lib/gvfs/gvfsd
  2060   1920 emrick   /usr/lib/gvfs/gvfsd-fuse /run/user/1000/gvfs -f -o big_wr
ites
```

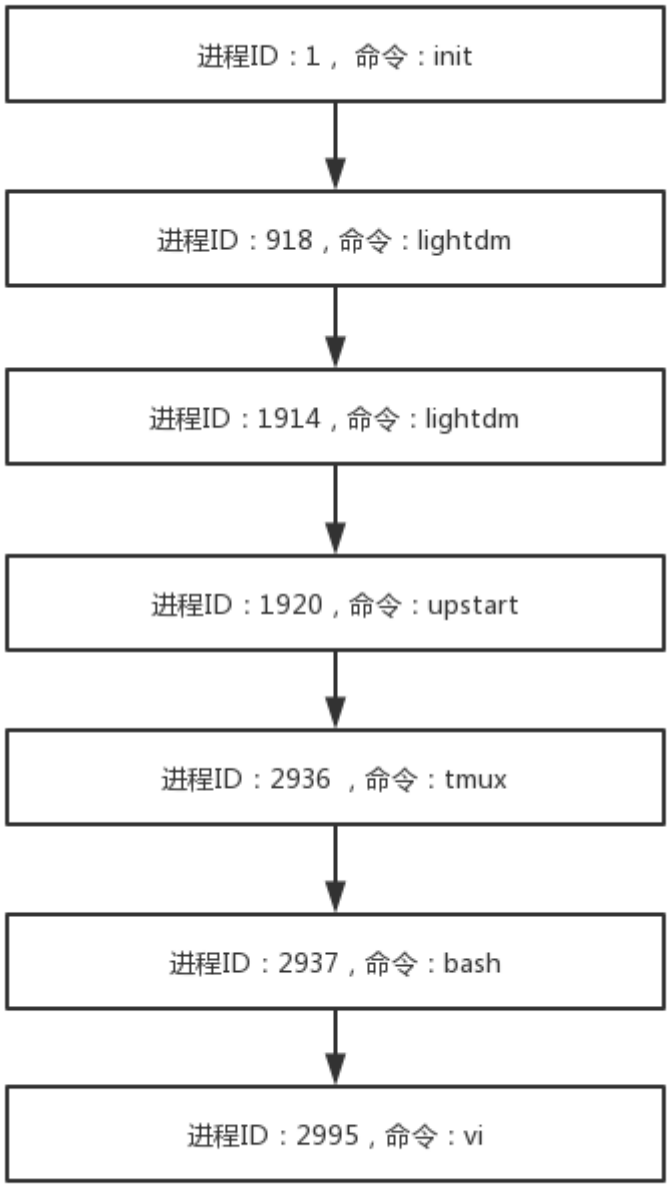
Terminal 19:38

emrick@ubuntu: ~

```
  2566   1920 emrick   zeitgeist-datahub
  2875   1920 emrick   /usr/lib/x86_64-linux-gnu/unity-scope-home/unity-scope-ho
me
  2887   1920 emrick   /usr/bin/unity-scope-loader applications/applications.sco
pe applications/scopes.scope commands.scope
  2888   1920 emrick   /usr/lib/x86_64-linux-gnu/unity-lens-files/unity-files-da
emon
  2914   1920 emrick   /usr/lib/gnome-terminal/gnome-terminal-server
  2936   1920 emrick   tmux
  3408   3293 emrick   grep --color=auto -w 1920
emrick@ubuntu:~$ ps -eo pid,ppid,user,command | grep -w 1194
  1194    918 root     lightdm --session-child 12 19
  1920   1194 emrick   /sbin/upstart --user
  3423   3293 emrick   grep --color=auto -w 1194
emrick@ubuntu:~$ ps -eo pid,ppid,user,command | grep -w 918
   918      1 root     /usr/sbin/lightdm
   972    918 root     /usr/lib/xorg/Xorg -core :0 -seat seat0 -auth /var/run/li
ghtdm/root/:0 -nolisten tcp vt7 -novtswitch
  1194    918 root     lightdm --session-child 12 19
  3435   3293 emrick   grep --color=auto -w 918
emrick@ubuntu:~$ ps -eo pid,ppid,user,command | grep -w 1
     1      0 root     /sbin/init auto noprompt
   357      1 root     /lib/systemd/systemd-journald
   386      1 root     /lib/systemd/systemd-udevd
```

a.out

通过追踪父进程，我们可以得到进程树：



和pstree显示的进程树进行比较：

```
│                    ├─indicator-sessi─┬─{dconf worker}
│                    │                 ├─{gdbus}
│                    │                 └─{gmain}
│                    ├─indicator-sound─┬─{dconf worker}
│                    │                 ├─{gdbus}
│                    │                 └─{gmain}
│                    ├─pulseaudio─┬─{alsa-sink-ES137}
│                    │            └─{alsa-source-ES1}
│                    ├─sh───zeitgeist-daemo─┬─{gdbus}
│                    │                      └─{gmain}
│                    ├─tmux─┬─bash───vi
│                    │      └─bash
│                    ├─unity-files-dae─┬─{dconf worker}
│                    │                 ├─{gdbus}
│                    │                 ├─{gmain}
│                    │                 └─{pool}
│                    ├─unity-panel-ser─┬─{dconf worker}
│                    │                 ├─{gdbus}
│                    │                 └─{gmain}
│                    ├─unity-scope-hom─┬─{dconf worker}
│                    │                 ├─{gdbus}
│                    │                 └─{gmain}
│                    ├─unity-scope-loa─┬─{dconf worker}
```

我们发现路径是相同的。

二、编写程序，首先使用fork系统调用，创建子进程。在父进程中继续执行空循环操作；在子进程中调用exec打开vi编辑器。然后在另外一个终端中，通过ps –Al命令、ps aux或者top等命令，查看vi进程及其父进程的运行状态，理解每个参数所表达的意义。选择合适的命令参数，对所有进程按照cpu占用率排序。

代码：

```
emrick@ubuntu: ~/Desktop
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>

int main(int argc, char *argv[])
{
        pid_t pid;
        pid = fork();

        if( pid < 0 ){  // 没有创建成功
                perror("fork");
        }

        if(0 == pid){ // 子进程
                int ret;
                ret = execl ("/usr/bin/vi","vi","/Desktop/hooks.txt", NULL);
                if (ret == -1)
                perror("execl");
        }else if(pid > 0){ // 父进程
                while(1){
                        printf("%d",pid);
                        sleep(2);
                }
        }

        return 0;
}
```

查看进程:

```
For more details see ps(1).
emrick@ubuntu:~$ ps -al
F S   UID    PID   PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
0 S  1000   3646   3323  0  80   0 -  5644 poll_s pts/4    00:00:00 tmux
0 S  1000   5461   3649  0  80   0 -  1088 hrtime pts/18   00:00:00 a.out
0 S  1000   5462   5461  0  80   0 - 13917 poll_s pts/18   00:00:00 vi
0 R  1000   5485   3353  0  80   0 -  7664 -      pts/17   00:00:00 ps
emrick@ubuntu:~$ ps -Al
```

各个参数含义:

```
emrick@ubuntu: ~/Desktop
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>

int main(int argc, char *argv[])
{
        pid_t pid;
        pid = fork();

        if( pid < 0 ){  // 没有创建成功
                perror("fork");
        }

        if(0 == pid){ // 子进程
                int ret;
                ret = execl ("/usr/bin/vi","vi","/Desktop/hooks.txt", NULL);
                if (ret == -1)
                perror("execl");
        }else if(pid > 0){ // 父进程
                while(1){
                        printf("%d",pid);
                        sleep(2);
                }
        }

        return 0;
}
```

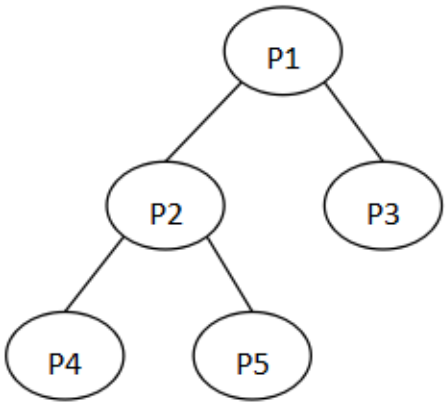
CPU占用率排序：都占用0%

打开火狐浏览器：使用 ps -Al 查看：

```
1 I     0   5827      2  0  80   0 -     0 -      ?        00:00:00 kworker/u256:2
1 I     0   5916      2  0  80   0 -     0 -      ?        00:00:00 kworker/u256:1
0 S  1000   5998   2300  0  80   0 - 36016 futex_ ?        00:00:00 oosplash
0 S  1000   6017   5998  2  80   0 - 292171 poll_s ?       00:00:02 soffice.bin
4 S  1000   6061   2300 40  80   0 - 447836 poll_s ?       00:00:04 firefox
4 S  1000   6133   6061 16  80   0 - 394855 poll_s ?       00:00:01 Web Content
4 S  1000   6202   6061 14  80   0 - 385991 poll_s ?       00:00:00 WebExtensions
4 S  1000   6248   6061  8  80   0 - 376262 poll_s ?       00:00:00 Web Content
4 R  1000   6272   3353  0  80   0 -  7664 -      pts/17   00:00:00 ps
```

可以看到不是0%的只有浏览器和web服务的进程，其中浏览器最多占40%

三、使用fork系统调用，创建如下进程树，并使每个进程输出自己的ID和父进程的ID。观察进程的执行顺序和运行状态的变化。



代码如下：

emrick@ubuntu: ~/Desktop

```
#include<stdio.h>
#include<stdlib.h>
#include <sys/types.h>

int main(int argc, char const *argv[])
{
        int p,p11,p12,p111,p112;
        while((p=fork())==-1);
        if (!p)
        {
                printf("PID：%d,PPID:%d\n",getpid(),getppid());
                while((p11=fork())==-1);
                if (!p11)
                {
                        printf("PID：%d,PPID:%d\n",getpid(),getppid());
                        while((p111=fork())==-1);
                        if (!p111)
                        {
                                printf("PID：%d,PPID:%d\n",getpid(),getppid());
                                exit(0);
                        }
                        else
                                wait(0);
                        while((p112=fork())==-1);
                        if (!p112)
                        {
                                printf("PID：%d,PPID:%d\n",getpid(),getppid());
                                exit(0);
                        }
                        else
                                wait(0);
                        exit(0);
                }
                else
                        wait(0);
                while((p12=fork())==-1);
                if (!p12)
                {
                        printf("PID：%d,PPID:%d\n",getpid(),getppid());
                        exit(0);
                }
                else
                        wait(0);
                exit(0);
        }
        return 0;
}
~
```

运行结果：

```
        wait(0);
        ^
emrick@ubuntu:~/Desktop$ ./a.out
emrick@ubuntu:~/Desktop$ PID：6956,PPID:2300
PID：6957,PPID:6956
PID：6958,PPID:6957
PID：6959,PPID:6957
PID：6960,PPID:6956
```

可以看到输出的顺序为P1 ->P2->P4->P5->P3(看起来像前序遍历)

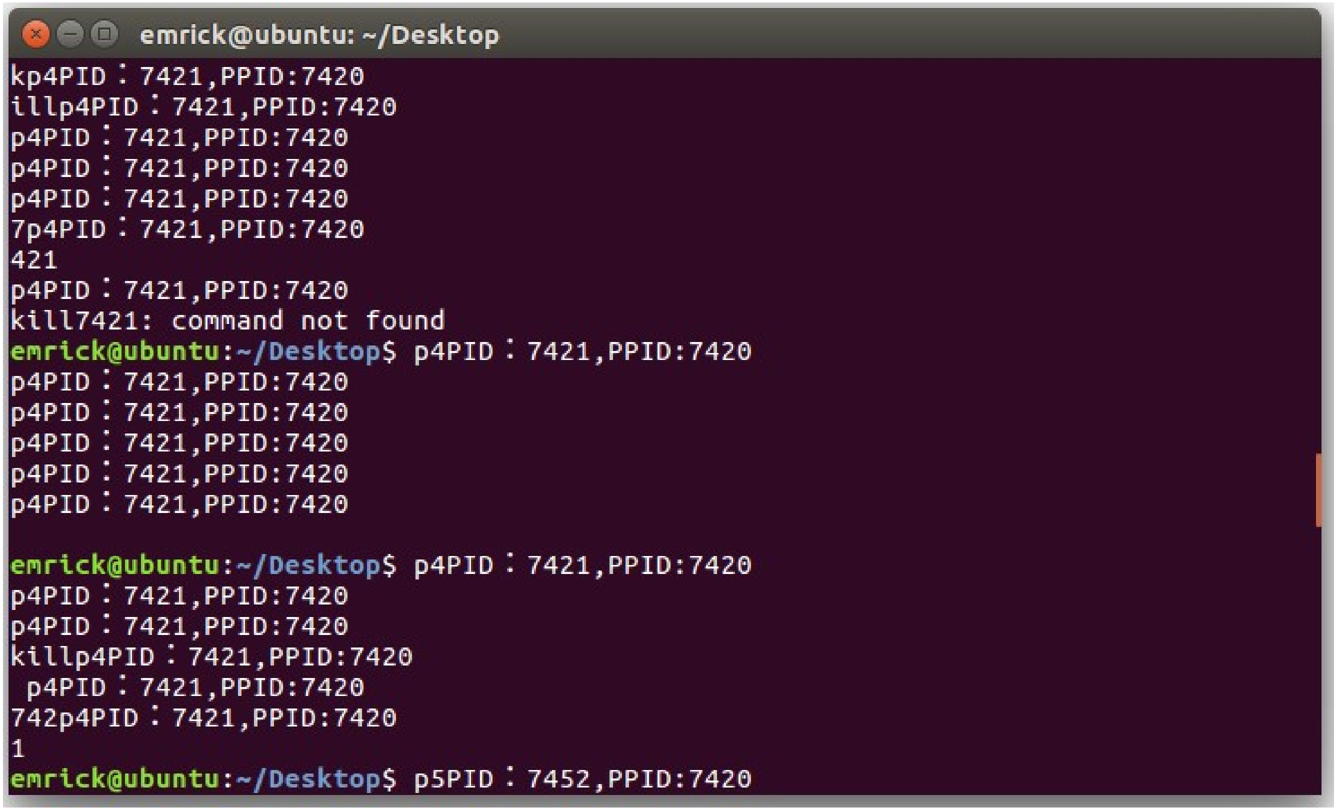
四、修改上述进程树中的进程，使得所有进程都循环输出自己的ID和父进程的ID。然后终止p2进程(分别采用kill -9 、自己正常退出exit()、段错误退出)，观察p1、p3、p4、p5进程的运行状态和其他相关参数有何改变。

代码：

```
@ubuntu: ~/Desktop
#include<stdio.h>
#include<stdlib.h>
#include <sys/types.h>

int main(int argc, char const *argv[])
{
        int p,p11,p12,p111,p112;
        while((p=fork())==-1);
        if (!p)
        {

                while((p11=fork())==-1);
                if (!p11)
                {

                        while((p111=fork())==-1);
                        if (!p111)
                        {
                                while(1){printf("p4PID：%d,PPID:%d\n",getpid(),getppid());sleep(1);}
                                exit(0);
                        }

                        while((p112=fork())==-1);
                        if (!p112)
                        {
                                while(1){printf("p5PID：%d,PPID:%d\n",getpid(),getppid());sleep(1);}
                                exit(0);
                        }

                        for(int i=0;i<5;i++){printf("P2ID：%d,PPID:%d\n",getpid(),getppid());sleep(1);}
                        exit(0);
                }

                while((p12=fork())==-1);
                if (!p12)
                {
                        while(1){printf("p3PID：%d,PPID:%d\n",getpid(),getppid());sleep(1);}
                        exit(0);
                }

                while(1){printf("p1PID：%d,PPID:%d\n",getpid(),getppid());sleep(1);}
        }
        return 0;
}
~
~
~
~
~
```

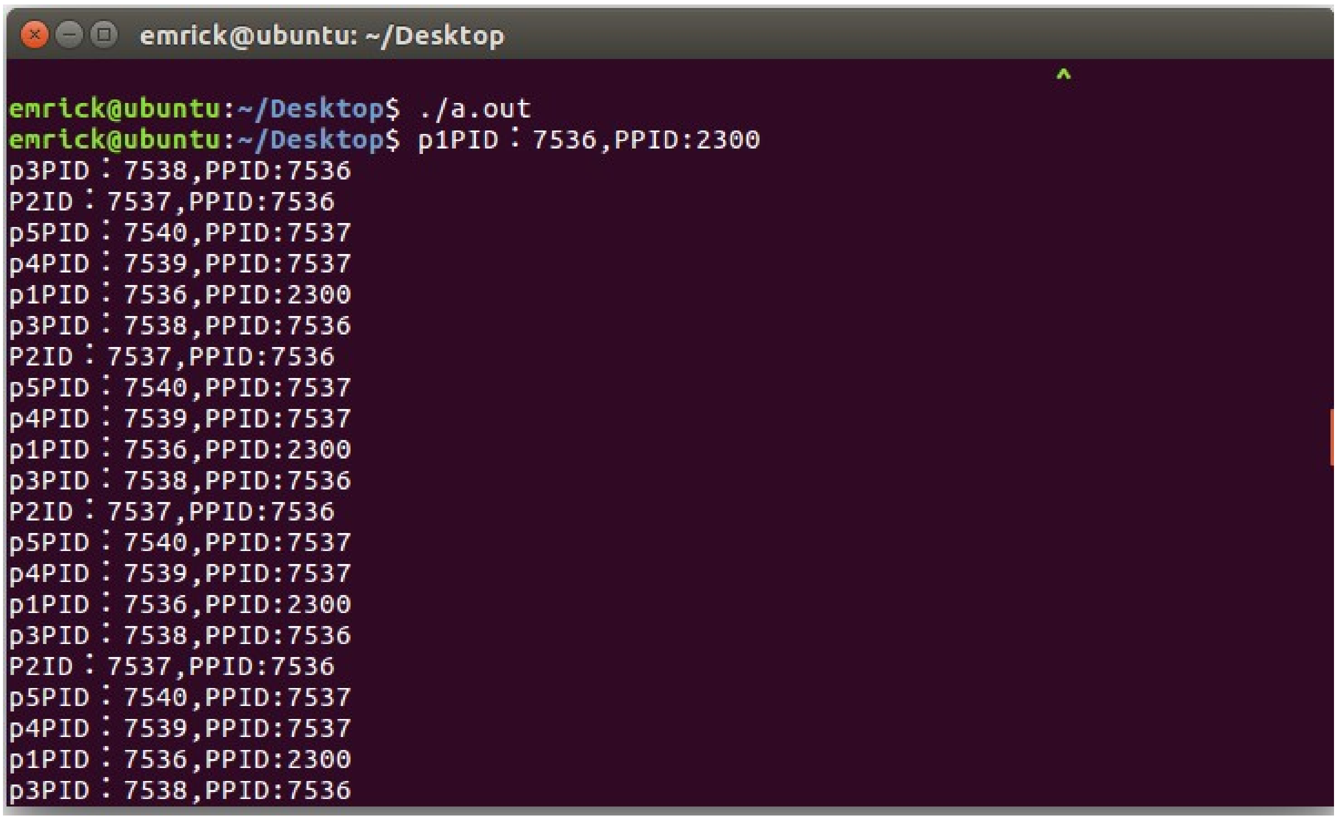
开始时我在每个进程前加上了wait（0）；

导致子进程不结束父进程就会一直处于等待状态，只有kill掉当前子进程才会执行父进程，有如下结果：



```
kp4PID：7421,PPID:7420
illp4PID：7421,PPID:7420
p4PID：7421,PPID:7420
p4PID：7421,PPID:7420
p4PID：7421,PPID:7420
7p4PID：7421,PPID:7420
421
p4PID：7421,PPID:7420
kill7421: command not found
emrick@ubuntu:~/Desktop$ p4PID：7421,PPID:7420
p4PID：7421,PPID:7420
p4PID：7421,PPID:7420
p4PID：7421,PPID:7420
p4PID：7421,PPID:7420
p4PID：7421,PPID:7420

emrick@ubuntu:~/Desktop$ p4PID：7421,PPID:7420
p4PID：7421,PPID:7420
p4PID：7421,PPID:7420
killp4PID：7421,PPID:7420
 p4PID：7421,PPID:7420
742p4PID：7421,PPID:7420
1
emrick@ubuntu:~/Desktop$ p5PID：7452,PPID:7420
```

去掉wait之后就会正常的进行输出：



```
                                                                 ^
emrick@ubuntu:~/Desktop$ ./a.out
emrick@ubuntu:~/Desktop$ p1PID：7536,PPID:2300
p3PID：7538,PPID:7536
P2ID：7537,PPID:7536
p5PID：7540,PPID:7537
p4PID：7539,PPID:7537
p1PID：7536,PPID:2300
p3PID：7538,PPID:7536
P2ID：7537,PPID:7536
p5PID：7540,PPID:7537
p4PID：7539,PPID:7537
p1PID：7536,PPID:2300
p3PID：7538,PPID:7536
P2ID：7537,PPID:7536
p5PID：7540,PPID:7537
p4PID：7539,PPID:7537
p1PID：7536,PPID:2300
p3PID：7538,PPID:7536
P2ID：7537,PPID:7536
p5PID：7540,PPID:7537
p4PID：7539,PPID:7537
p1PID：7536,PPID:2300
p3PID：7538,PPID:7536
```

kill掉P2后：

```
emrick@ubuntu: ~/Desktop
p5PID：7540,PPID:7537
p4PID：7539,PPID:7537
P2ID：7537,PPID:7536
p3PID：7538,PPID:7536
37p5PID：7540,PPID:7537
p1PID：7536,PPID:2300
p4PID：7539,PPID:7537
P2ID：7537,PPID:7536
p3PID：7538,PPID:7536

emrick@ubuntu:~/Desktop$ p5PID：7540,PPID:2300
p1PID：7536,PPID:2300
p4PID：7539,PPID:2300
p3PID：7538,PPID:7536
p5PID：7540,PPID:2300
p1PID：7536,PPID:2300
p4PID：7539,PPID:2300
p3PID：7538,PPID:7536
p1PID：7536,PPID:2300
p5PID：7540,PPID:2300
p4PID：7539,PPID:2300
p3PID：7538,PPID:7536
p5PID：7540,PPID:2300
p1PID：7536,PPID:2300
```

可以看到kill掉p2之后p2的两个子进程父进程就发生了改变，直接变成了2300（P1的父进程）这是因为在计算机中不允许进程没有父进程（root除外）。

如果是正常的exit运行如下：

```
emrick@ubuntu: ~/Desktop
P2ID：7762,PPID:7761
p5PID：7765,PPID:7762
p4PID：7764,PPID:7762
p1PID：7761,PPID:2300
p3PID：7763,PPID:7761
P2ID：7762,PPID:7761
p5PID：7765,PPID:7762
p4PID：7764,PPID:7762
p1PID：7761,PPID:2300
p3PID：7763,PPID:7761
P2ID：7762,PPID:7761
p5PID：7765,PPID:7762
p4PID：7764,PPID:7762
p1PID：7761,PPID:2300
p3PID：7763,PPID:7761
p5PID：7765,PPID:2300
p4PID：7764,PPID:2300
p1PID：7761,PPID:2300
p3PID：7763,PPID:7761
p5PID：7765,PPID:2300
p4PID：7764,PPID:2300
p1PID：7761,PPID:2300
p3PID：7763,PPID:7761
p5PID：7765,PPID:2300
```

可以看到在五次循环后p2正常exit，这时和上面的kill情况时相同的，P4P5直接变为了2300；