北京交通大学

《操作系统》实验报告

学　号: **16281015**

姓　名: **王子谦**

专　业: **计算机科学与技术**

学　院: **计算机与信息技术学院**

提交日期: **2019 年 3 月 18 日**

《操作系统》实验二

实验题目：

一、进程控制一

实验要求

1、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。

实验思路：

当我们打开一个 vi 进行编写时，该过程就是一个进程。通过已知进程的关键字，利用 ps -e|grep 查找对应进程号，再通过 ps -ef|grep 进程号可以进行父进程的追溯。再通过 pstree 指令进行比较进程树是否一致。

实验结果展示：

使用 vi 编辑文件：

```
vanzque@van-ubuntu1:~$ clear
vanzque@van-ubuntu1:~$ vi start.txt

[1]+  已停止               vi start.txt
vanzque@van-ubuntu1:~$
```

以下是 ps -ef|grep 追溯结果

```
vanzque@van-ubuntu1: ~
vanzque@van-ubuntu1:~$ ps -e|grep vi
  1841 ?        00:00:00 VGAuthService
  2123 ?        00:00:02 hud-service
  2540 ?        00:00:00 dconf-service
  5889 pts/13   00:00:00 vi
vanzque@van-ubuntu1:~$ ps -ef|grep 5889
vanzque    5889   5838  0 17:15 pts/13   00:00:00 vi start.txt
vanzque    6019   5968  0 17:17 pts/1    00:00:00 grep --color=auto 5889
vanzque@van-ubuntu1:~$ ps -ef|grep 5838
vanzque    5838   5033  0 17:15 pts/13   00:00:00 bash
vanzque    5889   5838  0 17:15 pts/13   00:00:00 vi start.txt
vanzque    6021   5968  0 17:17 pts/1    00:00:00 grep --color=auto 5838
vanzque@van-ubuntu1:~$ ps -ef|grep 5033
vanzque    5033   1615  2 16:40 ?        00:00:51 gnome-terminal
vanzque    5076   5033  0 16:40 ?        00:00:00 gnome-pty-helper
vanzque    5129   5033  0 16:40 pts/16   00:00:00 bash
vanzque    5838   5033  0 17:15 pts/13   00:00:00 bash
vanzque    5968   5033  0 17:17 pts/1    00:00:00 bash
vanzque    6023   5968  0 17:18 pts/1    00:00:00 grep --color=auto 5033
```

```
vanzque@van-ubuntu1: ~
vanzque@van-ubuntu1:~$ ps -ef|grep 1615
vanzque    1615   1565  0 14:58 ?        00:00:00 init --user
vanzque    2057   1615  0 14:58 ?        00:00:02 dbus-daemon --fork --session -
-address=unix:abstract=/tmp/dbus-crnHYQ5i3m
vanzque    2068   1615  0 14:58 ?        00:00:00 ssh-agent -s
vanzque    2072   1615  0 14:58 ?        00:00:00 upstart-event-bridge
vanzque    2075   1615  0 14:58 ?        00:00:00 /usr/lib/x86_64-linux-gnu/hud/
window-stack-bridge
vanzque    2098   1615  0 14:58 ?        00:00:00 upstart-file-bridge --daemon -
-user
vanzque    2100   1615  0 14:58 ?        00:00:00 upstart-dbus-bridge --daemon -
-system --user --bus-name system
vanzque    2102   1615  0 14:58 ?        00:00:00 upstart-dbus-bridge --daemon -
-session --user --bus-name session
vanzque    2104   1615  0 14:58 ?        00:00:27 /usr/bin/ibus-daemon --daemoni
ze --xim
vanzque    2119   1615  0 14:58 ?        00:00:00 /usr/lib/unity-settings-daemon
/unity-settings-daemon
vanzque    2123   1615  0 14:58 ?        00:00:02 /usr/lib/x86_64-linux-gnu/hud/
hud-service
vanzque    2126   1615  0 14:58 ?        00:00:00 /usr/lib/at-spi2-core/at-spi-b
us-launcher --launch-immediately
vanzque    2127   1615  0 14:58 ?        00:00:00 gnome-session --session=ubuntu
vanzque    2133   1615  0 14:58 ?        00:00:35 /usr/lib/unity/unity-panel-ser
```

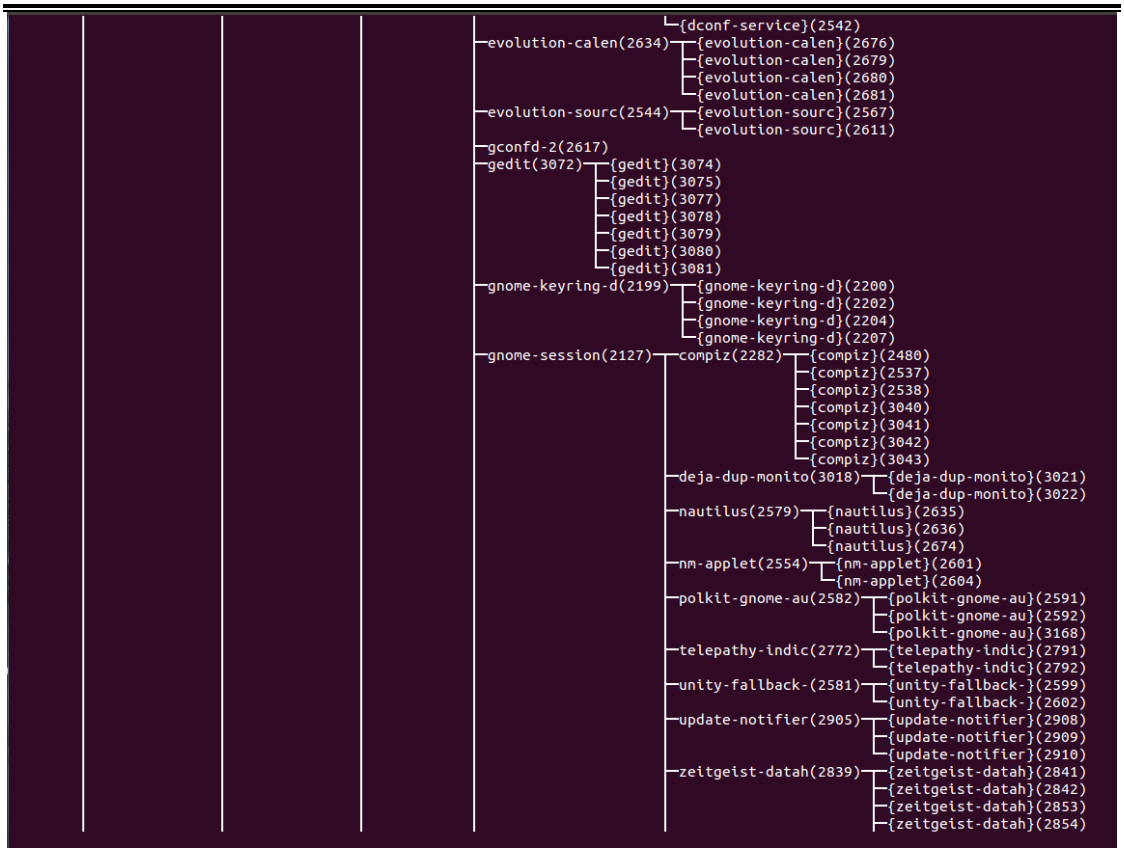
```
vanzque@van-ubuntu1:~$ ps -ef|grep 1565
root       1565   1348  0 14:58 ?        00:00:00 lightdm --session-child 12 15
vanzque    1615   1565  0 14:58 ?        00:00:00 init --user
vanzque    6032   5968  0 17:19 pts/1    00:00:00 grep --color=auto 1565
vanzque@van-ubuntu1:~$ ps -ef|grep 1348
root       1348      1  0 14:58 ?        00:00:00 lightdm
root       1475   1348  1 14:58 tty7     00:02:20 /usr/bin/X -core :0 -seat seat
0 -auth /var/run/lightdm/root/:0 -nolisten tcp vt7 -novtswitch
root       1565   1348  0 14:58 ?        00:00:00 lightdm --session-child 12 15
vanzque    6034   5968  0 17:20 pts/1    00:00:00 grep --color=auto 1348
vanzque@van-ubuntu1:~$
```

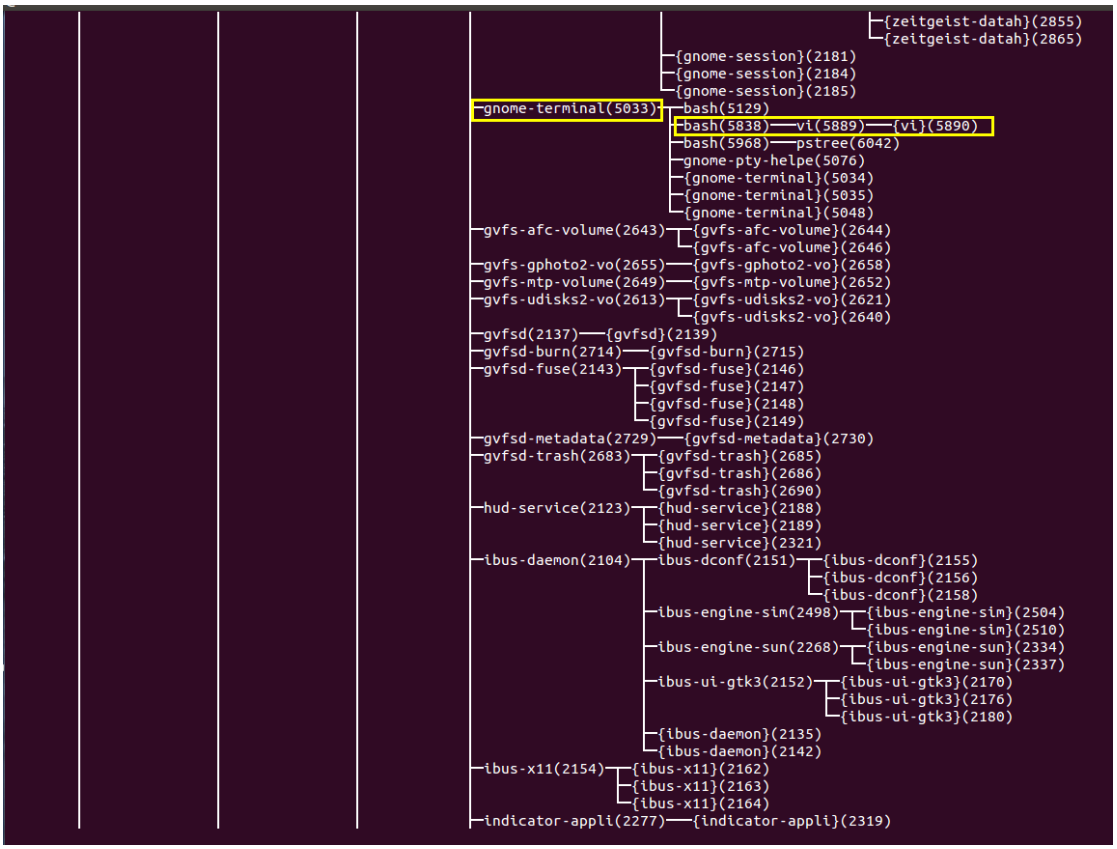
```
vanzque@van-ubuntu1:~$ ps -ef|grep 1
root           1       0  0 14:58 ?        00:00:01 /sbin/init
root           2       0  0 14:58 ?        00:00:00 [kthreadd]
root           3       2  0 14:58 ?        00:00:00 [ksoftirqd/0]
root           4       2  0 14:58 ?        00:00:00 [kworker/0:0]
root           5       2  0 14:58 ?        00:00:00 [kworker/0:0H]
root           7       2  0 14:58 ?        00:00:09 [rcu_sched]
root           8       2  0 14:58 ?        00:00:06 [rcuos/0]
root           9       2  0 14:58 ?        00:00:07 [rcuos/1]
root          10       2  0 14:58 ?        00:00:04 [rcuos/2]
root          11       2  0 14:58 ?        00:00:04 [rcuos/3]
root          12       2  0 14:58 ?        00:00:00 [rcuos/4]
root          13       2  0 14:58 ?        00:00:00 [rcuos/5]
root          14       2  0 14:58 ?        00:00:00 [rcuos/6]
root          15       2  0 14:58 ?        00:00:00 [rcuos/7]
root          16       2  0 14:58 ?        00:00:00 [rcuos/8]
root          17       2  0 14:58 ?        00:00:00 [rcuos/9]
root          18       2  0 14:58 ?        00:00:00 [rcuos/10]
root          19       2  0 14:58 ?        00:00:00 [rcuos/11]
root          20       2  0 14:58 ?        00:00:00 [rcuos/12]
root          21       2  0 14:58 ?        00:00:00 [rcuos/13]
root          22       2  0 14:58 ?        00:00:00 [rcuos/14]
root          23       2  0 14:58 ?        00:00:00 [rcuos/15]
```

以下是 pstree 指令追溯结果：

```
vanzque@van-ubuntu1: ~/os
vanzque@van-ubuntu1:~$ cd os
vanzque@van-ubuntu1:~/os$ touch helloworld.c
vanzque@van-ubuntu1:~/os$ gedit helloworld.c
vanzque@van-ubuntu1:~/os$ gcc -o hellow helloworld.c
vanzque@van-ubuntu1:~/os$ ./hellow
Hello, world!
vanzque@van-ubuntu1:~/os$
```

```
vanzque@van-ubuntu1:~$ pstree -p 1
init(1)─┬─ManagementAgent(1924)─┬─{ManagementAgent}(1934)
        │                       ├─{ManagementAgent}(1935)
        │                       ├─{ManagementAgent}(1936)
        │                       ├─{ManagementAgent}(1937)
        │                       ├─{ManagementAgent}(1938)
        │                       └─{ManagementAgent}(1939)
        ├─ModemManager(975)─┬─{ModemManager}(1089)
        │                   └─{ModemManager}(1095)
        ├─NetworkManager(1096)─┬─{NetworkManager}(1098)
        │                      ├─{NetworkManager}(1099)
        │                      └─{NetworkManager}(1100)
        ├─VGAuthService(1841)
        ├─accounts-daemon(1387)─┬─{accounts-daemon}(1462)
        │                       └─{accounts-daemon}(1463)
        ├─acpid(1364)
        ├─apache2(1543)─┬─apache2(3454)
        │               ├─apache2(3455)
        │               ├─apache2(3456)
        │               ├─apache2(3457)
        │               └─apache2(3458)
        ├─avahi-daemon(1024)───avahi-daemon(1029)
        ├─bluetoothd(989)
        ├─colord(2547)─┬─{colord}(2562)
        │              └─{colord}(2564)
        ├─cron(1335)
        ├─cups-browsed(1371)
        ├─cupsd(2769)
        ├─dbus-daemon(914)
        ├─dbus-daemon(2050)
        ├─dbus-launch(2026)
        ├─getty(1215)
        ├─getty(1220)
        ├─getty(1227)
        ├─getty(1228)
        ├─getty(1232)
        ├─getty(1799)
        ├─irqbalance(1342)
        ├─kerneloops(1327)
        ├─lightdm(1348)─┬─Xorg(1475)
        │               └─lightdm(1565)─┬─init(1615)─┬─at-spi-bus-laun(2126)─┬─dbus-daemon(2132)
        │                                │            │                       ├─{at-spi-bus-laun}(2129)
        │                                │            │                       ├─{at-spi-bus-laun}(2131)
        │                                │            │                       └─{at-spi-bus-laun}(2134)
        │                                │            ├─at-spi2-registr(2166)───{at-spi2-registr}(2179)
        │                                │            └─bamfdaemon(2183)─┬─{bamfdaemon}(2210)
        │                                │                               └─{bamfdaemon}(2211)
```





可以看出来二者追溯得到的进程树一致。

实验题目：

二、进程控制二

实验要求

2、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。

实验过程与思路：

1. 该实验关键在于对 fork()函数的理解，fork（）函数通过系统调用创建一个与原来进程几乎完全相同的进程，也就是两个进程可以做完全相同的事。返回值有两个，返回值为 0 时，表示为子进程。返回值>0 时，为父进程，且返回值为子进程 Id.该实验对 fork 返回值进行条件分支比较。

源代码：

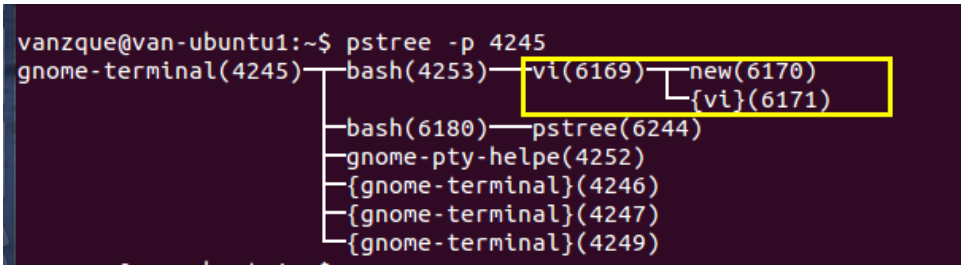
```
#include <unistd.h>
#include <stdio.h>

int main()
{
   int count=0;
   pid_t fpid;
   fpid = fork();
   if(fpid==0){
      int ret;
      ret = execl ("/usr/bin/vi",
"vi","/home/vanzque/new.txt", NULL);
      if (ret == -1)
         printf("GG");
```

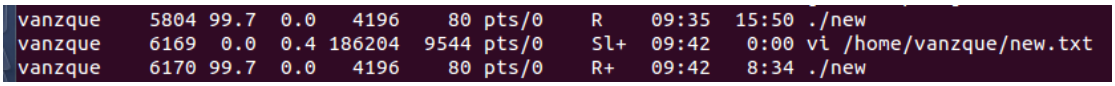
```
        }
        else if(fpid>0)
            {
                while(1)
                {count++;count--;}
            }
    return 0;
}
```

实验结果展示：

通过 pstree 查看其进程树：

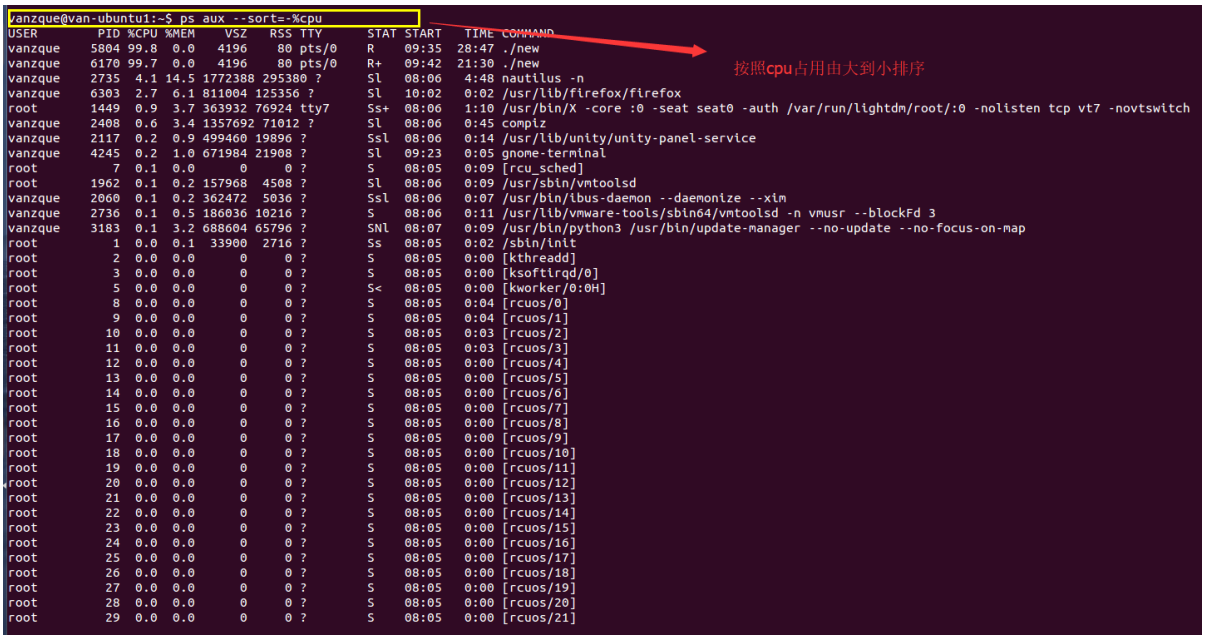


通过 ps aux 指令查看进程状态信息，在下图中，由于之前操作另一个进程未关闭，其状态为 R+，表示在后台正在运行，另一个 R 表示正在运行。



```
vanzque     5804 99.7  0.0   4196     80 pts/0    R    09:35  15:50 ./new
vanzque     6169  0.0  0.4 186204   9544 pts/0    Sl+  09:42   0:00 vi /home/vanzque/new.txt
vanzque     6170 99.7  0.0   4196     80 pts/0    R+   09:42   8:34 ./new
```

再通过 ps aux –sort=-%cpu 按照 cpu 占用由大到小排序。如下图所示。



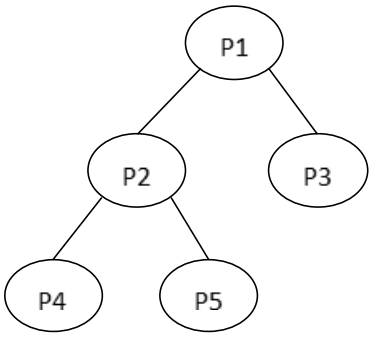
```
vanzque@van-ubuntu1:~$ ps aux --sort=-%cpu
USER        PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
vanzque    5804 99.8  0.0   4196    80 pts/0    R    09:35  28:47 ./new
vanzque    6170 99.7  0.0   4196    80 pts/0    R+   09:42  21:30 ./new
vanzque    2735  4.1 14.5 1772388 295380 ?      Sl   08:06   4:48 nautilus -n
vanzque    6303  2.7  6.1 811004 125356 ?       Sl   10:02   0:02 /usr/lib/firefox/firefox
root       1449  0.9  3.7 363932 76924 tty7     Ss+  08:06   1:10 /usr/bin/X -core :0 -seat seat0 -auth /var/run/lightdm/root/:0 -nolisten tcp vt7 -novtswitch
vanzque    2408  0.6  3.4 1357692 71012 ?       Sl   08:06   0:45 compiz
vanzque    2117  0.2  0.9 499460 19896 ?        Ssl  08:06   0:14 /usr/lib/unity/unity-panel-service
vanzque    4245  0.2  1.0 671984 21908 ?        Sl   09:23   0:05 gnome-terminal
root          7  0.1  0.0      0     0 ?        S    08:05   0:09 [rcu_sched]
root       1962  0.1  0.2 157968  4508 ?        Sl   08:06   0:09 /usr/sbin/vmtoolsd
vanzque    2060  0.1  0.2 362472  5036 ?        Ssl  08:06   0:07 /usr/bin/ibus-daemon --daemonize --xim
vanzque    2736  0.1  0.5 186036 10216 ?        S    08:06   0:11 /usr/lib/vmware-tools/sbin64/vmtoolsd -n vmusr --blockFd 3
vanzque    3183  0.1  3.2 688604 65796 ?        SNl  08:07   0:09 /usr/bin/python3 /usr/bin/update-manager --no-update --no-focus-on-map
root          1  0.0  0.1  33900  2716 ?        Ss   08:05   0:02 /sbin/init
root          2  0.0  0.0      0     0 ?        S    08:05   0:00 [kthreadd]
root          3  0.0  0.0      0     0 ?        S    08:05   0:00 [ksoftirqd/0]
root          5  0.0  0.0      0     0 ?        S<   08:05   0:00 [kworker/0:0H]
root          8  0.0  0.0      0     0 ?        S    08:05   0:04 [rcuos/0]
root          9  0.0  0.0      0     0 ?        S    08:05   0:04 [rcuos/1]
root         10  0.0  0.0      0     0 ?        S    08:05   0:03 [rcuos/2]
root         11  0.0  0.0      0     0 ?        S    08:05   0:03 [rcuos/3]
root         12  0.0  0.0      0     0 ?        S    08:05   0:00 [rcuos/4]
root         13  0.0  0.0      0     0 ?        S    08:05   0:00 [rcuos/5]
root         14  0.0  0.0      0     0 ?        S    08:05   0:00 [rcuos/6]
root         15  0.0  0.0      0     0 ?        S    08:05   0:00 [rcuos/7]
root         16  0.0  0.0      0     0 ?        S    08:05   0:00 [rcuos/8]
root         17  0.0  0.0      0     0 ?        S    08:05   0:00 [rcuos/9]
root         18  0.0  0.0      0     0 ?        S    08:05   0:00 [rcuos/10]
root         19  0.0  0.0      0     0 ?        S    08:05   0:00 [rcuos/11]
root         20  0.0  0.0      0     0 ?        S    08:05   0:00 [rcuos/12]
root         21  0.0  0.0      0     0 ?        S    08:05   0:00 [rcuos/13]
root         22  0.0  0.0      0     0 ?        S    08:05   0:00 [rcuos/14]
root         23  0.0  0.0      0     0 ?        S    08:05   0:00 [rcuos/15]
root         24  0.0  0.0      0     0 ?        S    08:05   0:00 [rcuos/16]
root         25  0.0  0.0      0     0 ?        S    08:05   0:00 [rcuos/17]
root         26  0.0  0.0      0     0 ?        S    08:05   0:00 [rcuos/18]
root         27  0.0  0.0      0     0 ?        S    08:05   0:00 [rcuos/19]
root         28  0.0  0.0      0     0 ?        S    08:05   0:00 [rcuos/20]
root         29  0.0  0.0      0     0 ?        S    08:05   0:00 [rcuos/21]
```

实验题目：

三、进程控制三

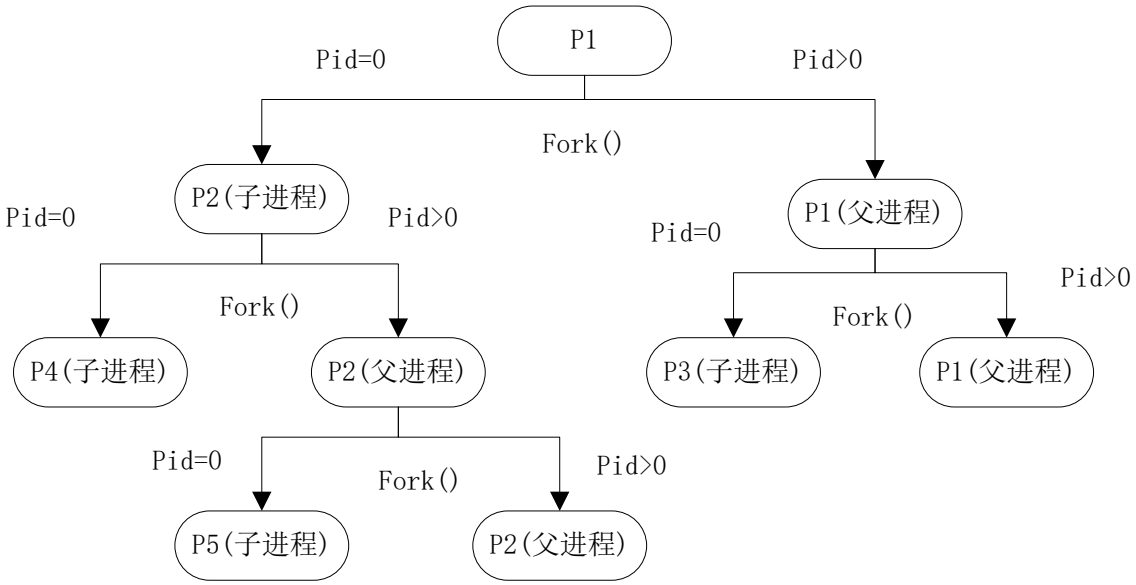
实验要求

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。



实验过程与思路：

个人思路流程图如下：



由以上流程图可设计条件语句进行实现：

有问题的源代码：

```
#include <unistd.h>
#include <stdio.h>
#include <sys/types.h>
#include <stdlib.h>
/*
疑惑。。。如果不加 sleep(1),所有 else 的父进程都是 1615。
*/
int main()
{
printf("当前进程(p1)ID 为%d\n",getpid());
pid_t fpid2;
//sleep(1);
fpid2 = fork();
if(fpid2 == 0){
 printf("当前进程(p2)ID 为%d,父进程 ID
为%d\n",getpid(),getppid());
 pid_t fpid4,fpid5;
 sleep(1);
 fpid4 = fork();
 if(fpid4 == 0)
 {
    printf("当前进程(p4)ID 为%d,父进程 ID
为%d\n",getpid(),getppid());
 }
 else
 {
 sleep(1);
 fpid5 = fork();
 if(fpid5 == 0)
 {
    printf("当前进程(p5)ID 为%d,父进程 ID
为%d\n",getpid(),getppid());
```

```
    }
    }
    }
   else{
   sleep(1);
   pid_t fpid3 = fork();
   if(fpid3 == 0){
    printf("当前进程(p3)ID 为%d,父进程 ID
为%d\n",getpid(),getppid());

   }
   }

   return 0;
   }
```

没问题的源代码：

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


int main()
{
printf("当前进程(p1)ID 为%d\n",getpid());
pid_t fpid2;

fpid2 = fork();
if(fpid2 == 0){
```

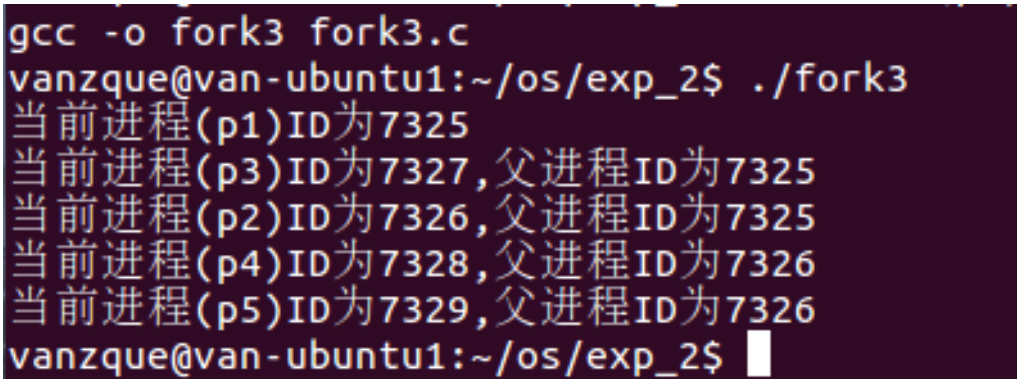
```
   printf(" 当 前 进 程 (p2)ID 为 %d, 父 进 程 ID
为%d\n",getpid(),getppid());
   pid_t fpid4,fpid5;
   fpid4 = fork();
   if(fpid4 == 0)
   {
      printf(" 当 前 进 程 (p4)ID 为 %d, 父 进 程 ID
为%d\n",getpid(),getppid());
   }
   else
   {
   fpid5 = fork();
   if(fpid5 == 0)
   {
      printf(" 当 前 进 程 (p5)ID 为 %d, 父 进 程 ID
为%d\n",getpid(),getppid());
   }
   }
   }
else{

pid_t fpid3 = fork();
if(fpid3 > 0){
   printf("当前进程(p3)ID 为%d,父进程 ID 为%d\n",fpid3,getpid());

}
}

return 0;
}
```

实验结果展示：



实验题目：

四、进程操作四

实验要求

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。

实验过程与思路：

源代码：

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


int main()
{
printf("当前进程(p1)ID 为%d\n",getpid());
pid_t fpid2;
```

```
   fpid2 = fork();
   if(fpid2 == 0){

   printf("当前进程(p2)ID 为%d,父进程 ID
为%d\n",getpid(),getppid());
   pid_t fpid4,fpid5;
   fpid4 = fork();
   if(fpid4 == 0)
   {
      printf("当前进程(p4)ID 为%d,父进程 ID
为%d\n",getpid(),getppid());
   }
   else
   {
   fpid5 = fork();
   if(fpid5 == 0)
   {
      printf("当前进程(p5)ID 为%d,父进程 ID
为%d\n",getpid(),getppid());
   }
   }
   }
   else{

   pid_t fpid3 = fork();
   if(fpid3 > 0){
   printf("当前进程(p3)ID 为%d,父进程 ID 为%d\n",fpid3,getpid());

   }
   }
   while(1)
      {
```

```
            sleep(3);
            printf("当前进程 ID 为%d,父进程 ID
为%d\n",getpid(),getppid());
        }
    return 0;
    }
```

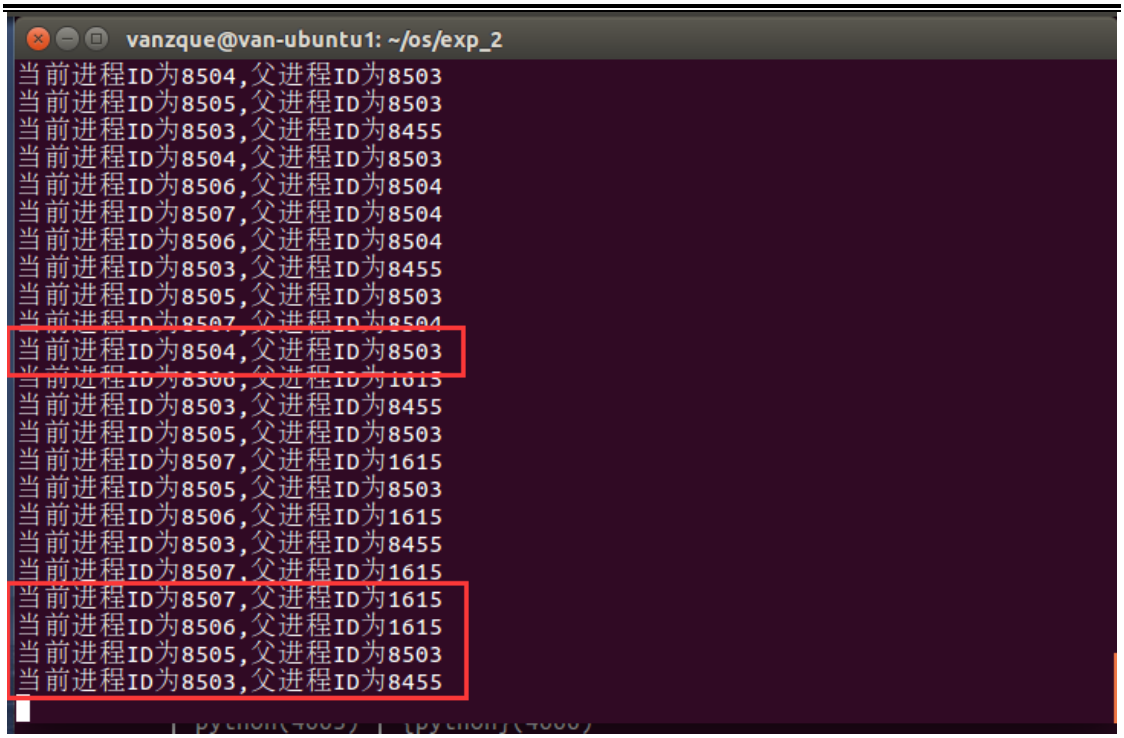
实验结果展示：

输出结果如下：

```
vanzque@van-ubuntu1:~/os/exp_2$ ./fork3
当前进程(p1)ID为8503
当前进程(p3)ID为8505,父进程ID为8503
当前进程(p2)ID为8504,父进程ID为8503
当前进程(p4)ID为8506,父进程ID为8504
当前进程(p5)ID为8507,父进程ID为8504
当前进程ID为8504,父进程ID为8503
当前进程ID为8503,父进程ID为8455
当前进程ID为8505,父进程ID为8503
当前进程ID为8506,父进程ID为8504
当前进程ID为8507,父进程ID为8504
```

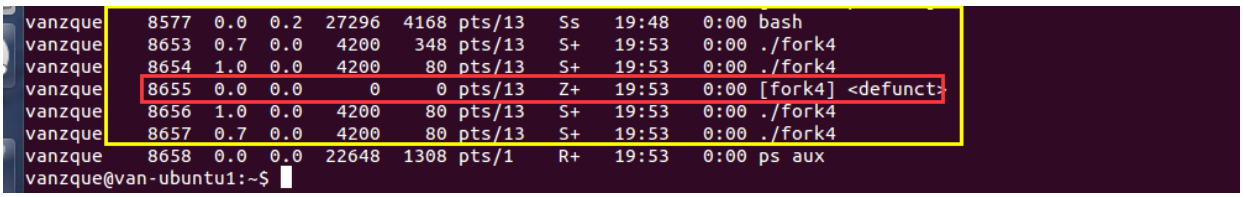
进程树如下：

使用 kill -9 后：p2 结束，并且 P4,P5 的父进程变成了 1615。（此处代码循环结构进行了修改）



各个进程状态如下：

P2 进程变成了 Z+表示该进程处于"Zombie"状态



使用 exit(0):

代码需修改处：

```
#include <stdlib.h>


int main()
{
printf("当前进程(p1)ID为%d\n",getpid());
pid_t fpid2;

fpid2 = fork();
if(fpid2 == 0){

        printf("当前进程(p2)ID为%d,父进程ID为%d\n",getpid(),getppid());
        pid_t fpid4,fpid5;
        fpid4 = fork();
        if(fpid4 == 0)
        {       while(1){
                printf("当前进程(p4)ID为%d,父进程ID为%d\n",getpid(),getppid());}
        }
        else
        {
        fpid5 = fork();
        if(fpid5 == 0)
        {       while(1){
                printf("当前进程(p5)ID为%d,父进程ID为%d\n",getpid(),getppid());}
        }
        }
        while(1){
        printf("当前进程(p2)ID为%d,父进程ID为%d\n",getpid(),getppid());}
        exit(0);
        }
else{

pid_t fpid3 = fork();
if(fpid3 > 0){
        while(1){
        printf("当前进程(p3)ID为%d,父进程ID为%d\n",fpid3,getpid());}

}
}

return 0;
}
```

exit()设置

Z+表示该进程处于"Zombie"状态

```
vanzque    8089  0.0  0.2  27308  4184 pts/16   Ss   19:17   0:00 bash
vanzque    8327  0.0  0.0      0     0 pts/16   Z+   19:21   0:00 [fork3] <defunct>
vanzque    8328  0.0  0.0      0     0 pts/16   Z+   19:21   0:00 [fork3] <defunct>
vanzque    8336  0.0  0.2  27288  4148 pts/1    Ss   19:21   0:00 bash
vanzque    8386  0.0  0.0  23172  1860 pts/1    R+   19:23   0:00 ps aux --sort=-%cpu
```

（实验疑惑）此处发现 P4,P5 的父节点并没有改变，说明 P2 并没有正常结束。但是其状态已经是 zombie 了。

纠正：

exit(0)应加在 while 循环中。以下截图 P1 也 zombie 是由于没有写循环了。

```c
#include <stdlib.h>


int main()
{
printf("当前进程(p1)ID为%d\n",getpid());
pid_t fpid2;

fpid2 = fork();
if(fpid2 == 0){

        printf("当前进程(p2)ID为%d,父进程ID为%d\n",getpid(),getppid());
        pid_t fpid4,fpid5;
        fpid4 = fork();
        if(fpid4 == 0)
        {       while(1){
                printf("当前进程(p4)ID为%d,父进程ID为%d\n",getpid(),getppid());}
        }
        else
        {
        fpid5 = fork();
        if(fpid5 == 0)
        {       while(1){
                printf("当前进程(p5)ID为%d,父进程ID为%d\n",getpid(),getppid());}
        }
        }

        while(1){
        printf("当前进程(p2)ID为%d,父进程ID为%d\n",getpid(),getppid());
        exit(0);}

        }
else{

pid_t fpid3 = fork();
if(fpid3 > 0){
        while(1){
        printf("当前进程(p3)ID为%d,父进程ID为%d\n",fpid3,getpid());}

}
}

return 0;
```

```
当前进程(p3)ID为9012,父进程ID为9010
当前进程(p4)ID为9013,父进程ID为1615
当前进程(p5)ID为9014,父进程ID为1615
当前进程(p3)ID为9012,父进程ID为9010
当前进程(p4)ID为9013,父进程ID为1615
当前进程(p3)ID为9012,父进程ID为9010
当前进程(p5)ID为9014,父进程ID为1615
当前进程(p3)ID为9012,父进程ID为9010
当前进程(p4)ID为9013,父进程ID为1615
当前进程(p5)ID为9014,父进程ID为1615
当前进程(p3)ID为9012,父进程ID为9010
当前进程(p4)ID为9013,父进程ID为1615
当前进程(p3)ID为9012,父进程ID为9010
当前进程(p5)ID为9014,父进程ID为1615
当前进程(p3)ID为9012,父进程ID为9010
当前进程(p4)ID为9013,父进程ID为1615
当前进程(p5)ID为9014,父进程ID为1615
当前进程(p3)ID为9012,父进程ID为9010
当前进程(p4)ID为9013,父进程ID为1615
当前进程(p5)ID为9014,父进程ID为1615
当前进程(p3)ID为9012,父进程ID为9010
当前进程(p4)ID为9013,父进程ID为1615
当前进程(p3)ID为9012,父进程ID为9010
^Cyanzque@yan-ubuntu1:~/os/exp_2$
```

段错误退出：

此处对指针进行错误操作。函数取自百科段错误例子

try.c × fork3.c × fork4.c ×

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

dummy_function (void)
{
unsigned char *ptr = 0x00;
*ptr = 0x00;
}

int main()
{

pid_t fpid2;

fpid2 = fork();
if(fpid2 == 0){

        printf("当前进程(p2)ID为%d,父进程ID为%d\n",getpid(),getppid());
        pid_t fpid4,fpid5;
        fpid4 = fork();
        if(fpid4 == 0)
        {       while(1){
                printf("当前进程(p4)ID为%d,父进程ID为%d\n",getpid(),getppid());}
        }
        else
        {
        fpid5 = fork();
        if(fpid5 == 0)
        {       while(1){
                printf("当前进程(p5)ID为%d,父进程ID为%d\n",getpid(),getppid());}
        }
        }

        while(1){
        printf("当前进程(p2)ID为%d,父进程ID为%d\n",getpid(),getppid());
        /
        dummy_function ();
        }

        }
else{
```

```
vanzque    8935  0.0  0.2  27296  4172 pts/13   Ss   20:03   0:00 bash
root       8998  0.0  0.0      0     0 ?        S    20:03   0:00 [kworker/u256:0]
root       9052  0.1  0.0      0     0 ?        S    20:11   0:00 [kworker/u256:2]
vanzque    9112  0.8  0.0   4200   352 pts/13   S+   20:15   0:00 ./try
vanzque    9113  0.8  0.0      0     0 pts/13   Z+   20:15   0:00 [try] <defunct>
vanzque    9114  0.8  0.0   4200    80 pts/13   S+   20:15   0:00 ./try
vanzque    9115  0.6  0.0   4200    84 pts/13   S+   20:15   0:00 ./try
vanzque    9116  0.6  0.0   4200    84 pts/13   S+   20:15   0:00 ./try
vanzque    9118  0.0  0.0  22648  1304 pts/1    R+   20:16   0:00 ps aux
```

P4,P5 父进程号变成 1615

实验总结和心得

通过此次实验，对于如何通过 fork()创建线程有了了解，学习了父进程与子进程的创建关系与顺序。对于整体进程控制的使用操作有了更进一步的学习。