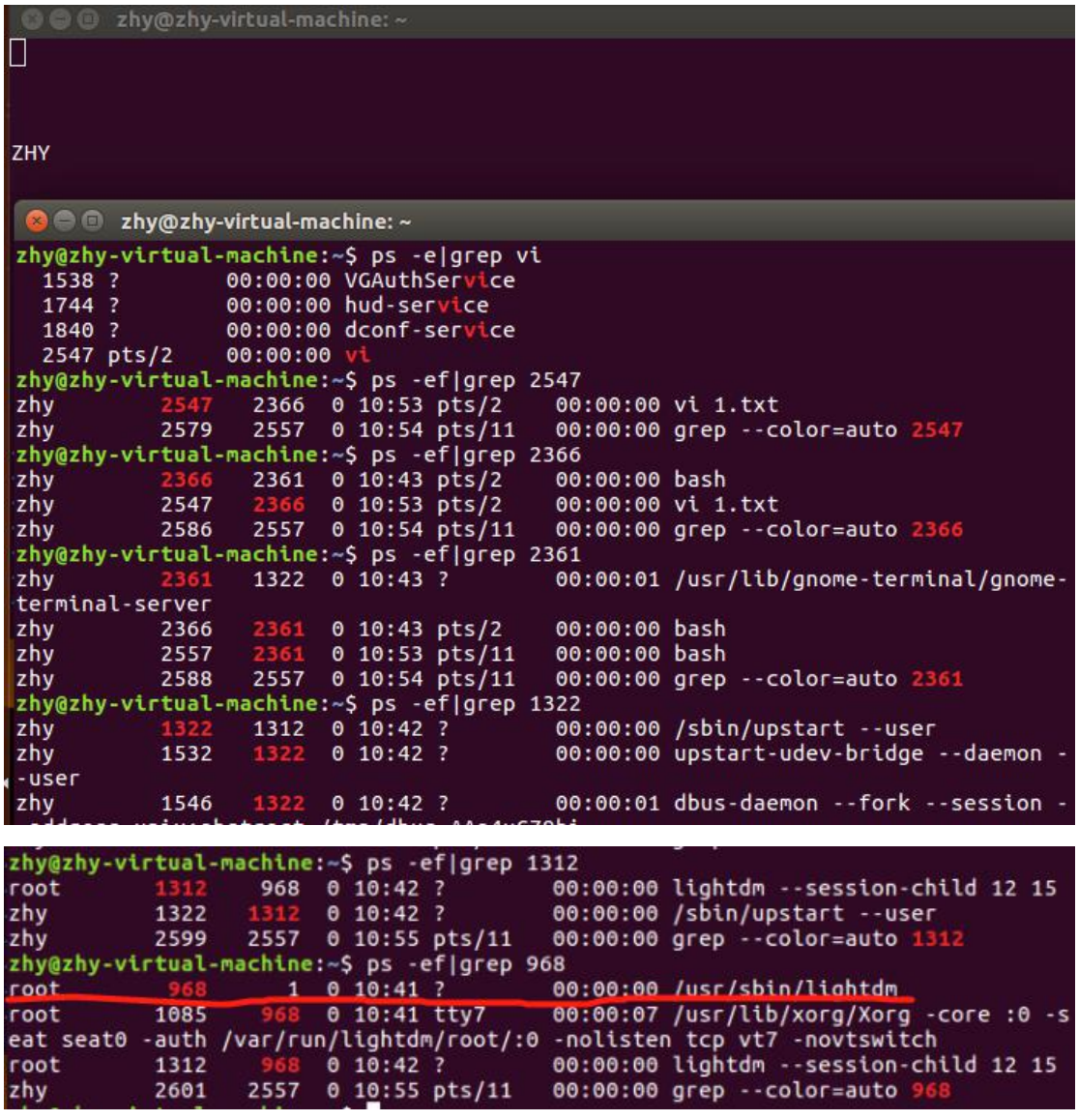
实验二 进程控制

安全 1601　　战泓夷　　16281022

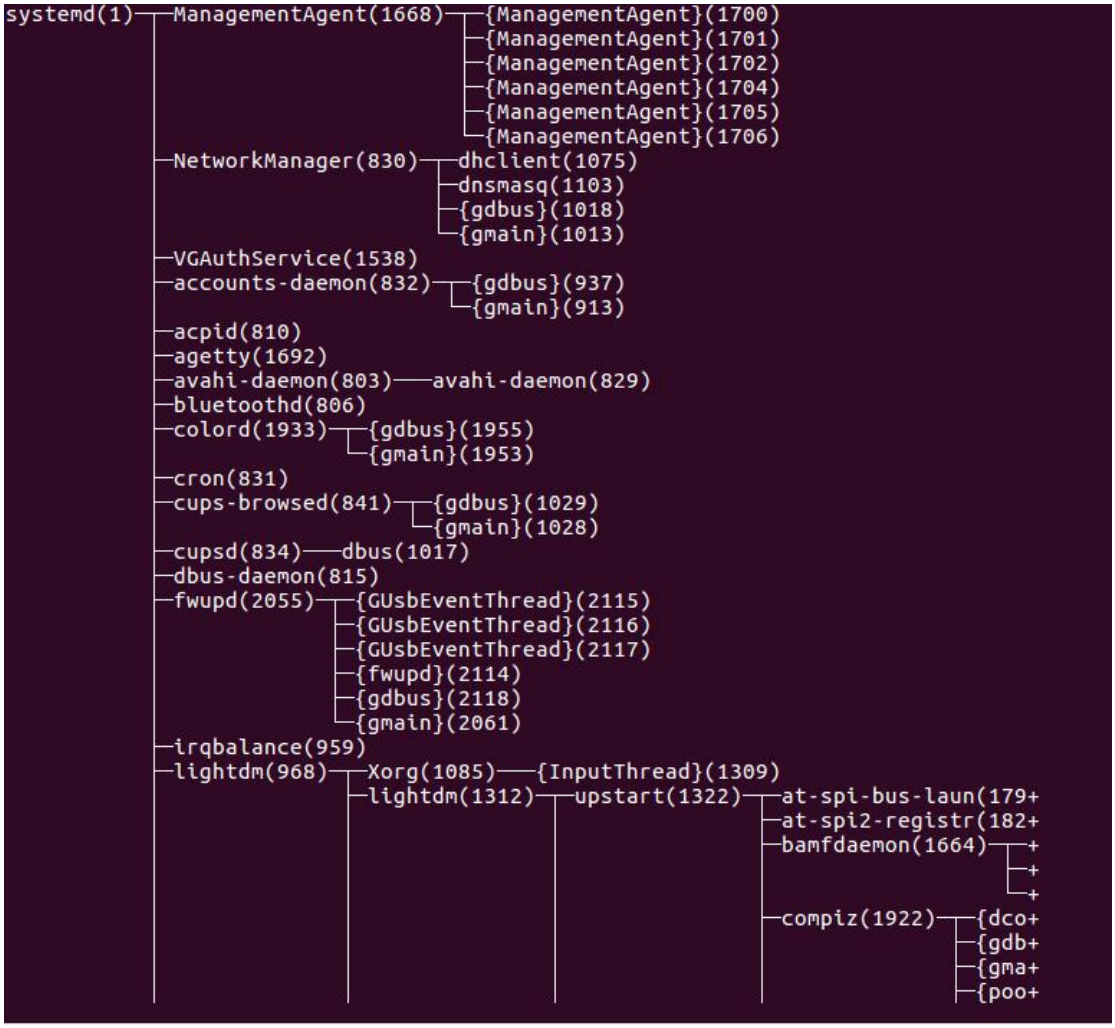
根据课堂所学内容和基础知识介绍，完成实验题目。

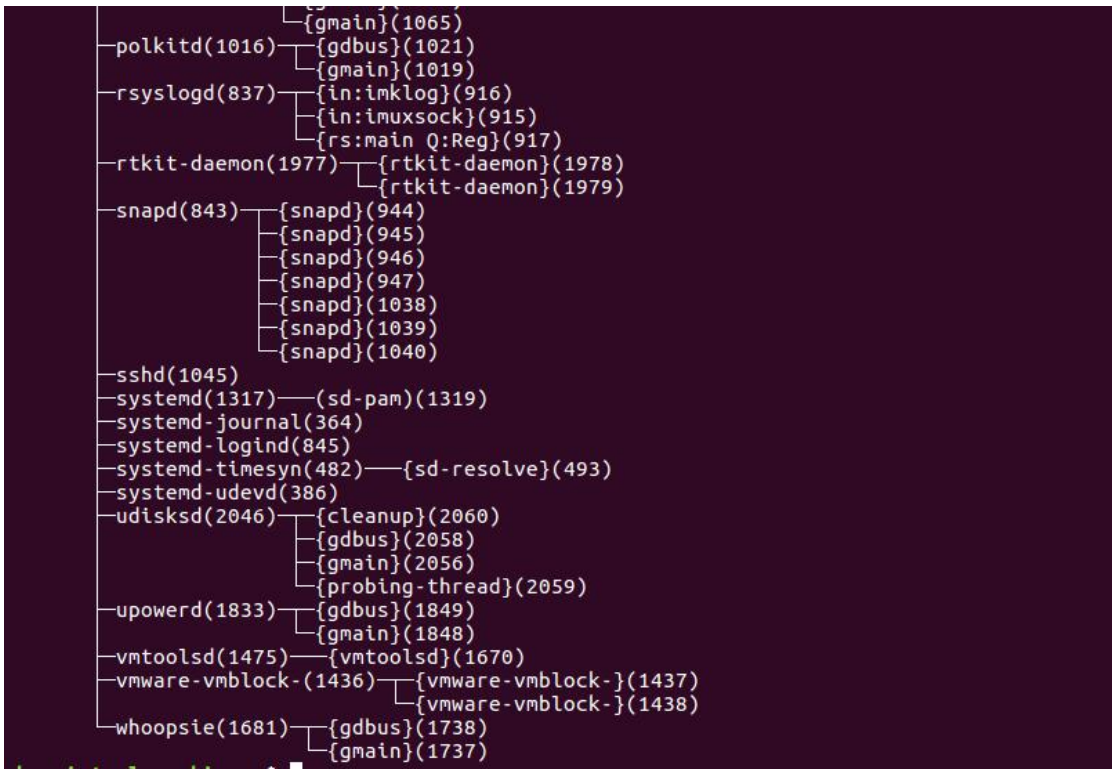
- **1、打开一个 vi 进程。通过 ps 命令以及选择合适的参数，只显示名字为 vi 的进程。寻找 vi 进程的父进程，直到 init 进程为止。记录过程中所有进程的 ID 和父进程 ID。将得到的进程树和由 pstree 命令的得到的进程树进行比较。**

首先使用 vi 打开编辑一个文件，输入内容后，在终端使用 ps 命令查找该进程，然后根据其父进程直至找到 init 进程。



在终端输入命令 pstree -p 1 查询进程树


systemd(1)
ManagementAgent(1668)
{ManagementAgent}(1700)
{ManagementAgent}(1701)
{ManagementAgent}(1702)
{ManagementAgent}(1704)
{ManagementAgent}(1705)
{ManagementAgent}(1706)
NetworkManager(830)
dhclient(1075)
dnsmasq(1103)
{gdbus}(1018)
{gmain}(1013)
VGAuthService(1538)
accounts-daemon(832)
{gdbus}(937)
{gmain}(913)
acpid(810)
agetty(1692)
avahi-daemon(803)
avahi-daemon(829)
bluetoothd(806)
colord(1933)
{gdbus}(1955)
{gmain}(1953)
cron(831)
cups-browsed(841)
{gdbus}(1029)
{gmain}(1028)
cupsd(834)
dbus(1017)
dbus-daemon(815)
fwupd(2055)
{GUsbEventThread}(2115)
{GUsbEventThread}(2116)
{GUsbEventThread}(2117)
{fwupd}(2114)
{gdbus}(2118)
{gmain}(2061)
irqbalance(959)
lightdm(968)
Xorg(1085)
{InputThread}(1309)
lightdm(1312)
upstart(1322)
at-spi-bus-laun(179+
at-spi2-registr(182+
bamfdaemon(1664)
compiz(1922)
{dco+
{gdb+
{gma+
{poo+


{gmain}(1065)
polkitd(1016)
{gdbus}(1021)
{gmain}(1019)
rsyslogd(837)
{in:imklog}(916)
{in:imuxsock}(915)
{rs:main Q:Reg}(917)
rtkit-daemon(1977)
{rtkit-daemon}(1978)
{rtkit-daemon}(1979)
snapd(843)
{snapd}(944)
{snapd}(945)
{snapd}(946)
{snapd}(947)
{snapd}(1038)
{snapd}(1039)
{snapd}(1040)
sshd(1045)
systemd(1317)
(sd-pam)(1319)
systemd-journal(364)
systemd-logind(845)
systemd-timesyn(482)
{sd-resolve}(493)
systemd-udevd(386)
udisksd(2046)
{cleanup}(2060)
{gdbus}(2058)
{gmain}(2056)
{probing-thread}(2059)
upowerd(1833)
{gdbus}(1849)
{gmain}(1848)
vmtoolsd(1475)
{vmtoolsd}(1670)
vmware-vmblock-(1436)
{vmware-vmblock-}(1437)
{vmware-vmblock-}(1438)
whoopsie(1681)
{gdbus}(1738)
{gmain}(1737)

二者比较：

ps 命令是从当前进程一步步寻找父进程，查找的信息更丰富、全面，但是操作繁琐，不是很简便；而 pstree 是从初始进程开始，一次给出全部进程，结构清晰，但所包含的内容比较少。

- **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。**

按要求编写程序：

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

int main()
{
    pid_t pid;
    pid = fork ();
    if ( pid == 0 )        // 子进程
    {
        int ret;
        ret = execl ("/usr/bin/vi", "vi","/home/zhy/2.c", NULL);
        if (ret == -1)
           perror ("execl");
        printf ("I am the baby!\n");
    }
    else if( pid > 0)
    {
        printf ("I am the parent of pid=%d!\n", pid);
        while(1){
        }
    }
    return 0;
}
```

编译程序，打开另一终端

```
zhy@zhy-virtual-machine:~$ ps -ef|grep 4533
zhy         4533    4532  0 17:08 pts/1    00:00:00 vi /home/zhy/2.c
zhy         4576    4545  0 17:09 pts/11   00:00:00 grep --color=auto 4533
```

ps-l：将目前属于您自己这次登入的 PID 与相关信息列示出来

ps-A：显示所有进程信息

```
zhy@zhy-virtual-machine:~$ ps -Al
F S   UID     PID    PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
4 S     0       1       0  0  80   0 - 46344 -      ?        00:00:03 systemd
1 S     0       2       0  0  80   0 -     0 -      ?        00:00:00 kthreadd
1 S     0       4       2  0  60 -20 -     0 -      ?        00:00:00 kworker/0:0H
1 S     0       6       2  0  60 -20 -     0 -      ?        00:00:00 mm_percpu_wq
1 S     0       7       2  0  80   0 -     0 -      ?        00:00:00 ksoftirqd/0
1 S     0       8       2  0  80   0 -     0 -      ?        00:00:00 rcu_sched
1 S     0       9       2  0  80   0 -     0 -      ?        00:00:00 rcu_bh
1 S     0      10       2  0 -40   - -     0 -      ?        00:00:00 migration/0
5 S     0      11       2  0 -40   - -     0 -      ?        00:00:00 watchdog/0
1 S     0      12       2  0  80   0 -     0 -      ?        00:00:00 cpuhp/0
1 S     0      13       2  0  80   0 -     0 -      ?        00:00:00 cpuhp/1
```

```
0 S  1000    4533    4532  0  80   0 -  9818 poll_s pts/1    00:00:00 vi
```

F 代表这个程序的旗标 (flag)， 4 代表使用者为 super user

S 代表这个程序的状态 (STAT)

UID 程序被该 UID 所拥有

PID 就是这个程序的 ID

PPID 则是其上级父程序的 ID

C CPU 使用的资源百分比

PRI 这个是 Priority (优先执行序) 的缩写

NI 这个是 Nice 值

ADDR 这个是 kernel function，指出该程序在内存的那个部分。如果是个 running 的程序，一般就是 "-"

SZ 使用掉的内存大小

WCHAN 目前这个程序是否正在运作当中，若为 - 表示正在运作

TTY 登入者的终端机位置

TIME 使用掉的 CPU 时间。

CMD 所下达的指令为何。

ps aux： 列出目前所有的正在内存当中的程序

```
zhy@zhy-virtual-machine:~$ ps aux
USER         PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root           1  0.0  0.2 185376  6020 ?        Ss   16:16   0:03 /sbin/init spl
root           2  0.0  0.0      0     0 ?        S    16:16   0:00 [kthreadd]
root           4  0.0  0.0      0     0 ?        S<   16:16   0:00 [kworker/0:0H]
root           6  0.0  0.0      0     0 ?        S<   16:16   0:00 [mm_percpu_wq]
root           7  0.0  0.0      0     0 ?        S    16:16   0:00 [ksoftirqd/0]
root           8  0.0  0.0      0     0 ?        S    16:16   0:01 [rcu_sched]
```

```
zhy         4533  0.0  0.1  39272  3864 pts/1    S+   17:08   0:00 vi /home/zhy/2
```

USER 该 process 属于那个使用者账号的

PID 该 process 的号码

%CPU 该 process 使用掉的 CPU 资源百分比

%MEM 该 process 所占用的物理内存百分比

VSZ 该 process 使用掉的虚拟内存量 (Kbytes)

RSS 该 process 占用的固定的内存量 (Kbytes)

TTY 该 process 是在那个终端机上面运作， tty1-tty6 是本机上面的登入者程序，若为 pts/0 等等的，则表示为由网络连接进主机的程序。

STAT 该程序目前的状态

R 该程序目前正在运作，或者是可被运作

S 该程序目前正在睡眠当中 (可说是 idle 状态)，但可被某些讯号 (signal) 唤醒。

T 该程序目前正在侦测或者是停止了

Z 该程序应该已经终止，但是其父程序却无法正常的终止他，造成 zombie (疆尸) 程序的状态

START 该 process 被触发启动的时间

TIME 该 process 实际使用 CPU 运作的时间

COMMAND 该程序的实际指令

```
top - 17:14:53 up 58 min,  1 user,  load average: 1.00, 0.81, 0.50
Tasks: 249 total,   2 running, 246 sleeping,   0 stopped,   1 zombie
%Cpu(s): 25.2 us,  0.3 sy,  0.0 ni, 74.4 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem :  2018084 total,   272244 free,   762816 used,   983024 buff/cache
KiB Swap:  2094076 total,  2094076 free,        0 used.  1025900 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
 4532 zhy       20   0    4352    648    584 R 100.0  0.0   6:22.43 2
 1086 root      20   0  360384  59540  29804 S   1.3  3.0   0:22.35 Xorg
 4447 zhy       20   0  619204  44132  34416 S   1.3  2.2   0:01.68 gnome-ter+
 1984 zhy       20   0 1377500 121036  81316 S   0.3  6.0   0:37.46 compiz
    1 root      20   0  185376   6020   4028 S   0.0  0.3   0:03.39 systemd
    2 root      20   0       0      0      0 S   0.0  0.0   0:00.03 kthreadd
    4 root       0 -20       0      0      0 S   0.0  0.0   0:00.00 kworker/0+
    6 root       0 -20       0      0      0 S   0.0  0.0   0:00.00 mm_percpu+
    7 root      20   0       0      0      0 S   0.0  0.0   0:00.08 ksoftirqd+
    8 root      20   0       0      0      0 S   0.0  0.0   0:01.02 rcu_sched
```

PID 进程 id

USER 进程所有者的用户名

PR 优先级

VIRT 进程使用的虚拟内存总量，单位 kb。VIRT=SWAP+RES

RES 进程使用的、未被换出的物理内存大小，单位 kb。RES=CODE+DATA

SHR　共享内存大小，单位 kb

S　进程状态(D=不可中断的睡眠状态,R=运行,S=睡眠,T=跟踪/停止,Z=僵尸进程)

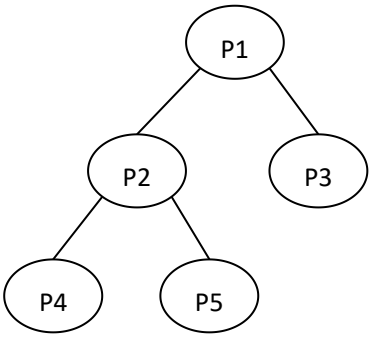
%CPU　上次更新到现在的 CPU 时间占用百分比

%MEM　进程使用的物理内存百分比

TIME+　进程使用的 CPU 时间总计，单位 1/100 秒

COMMAND 命令名/命令行

- 3、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。



（1）代码截图：

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

int main(){
    pid_t pid;
    pid = fork();
    if (pid == 0){
        pid = fork();
        if(pid > 0)
            fork();
        printf ("pid = %d ppid = %d!\n",getpid(),getppid());
        sleep(1);
    }
    else if (pid > 0){
        fork();
        printf ("pid = %d ppid = %d!\n",getpid(),getppid());
        sleep(1);
    }
    else if (pid == -1)
        perror ("fork");
    return 0;
}
```

（2）编译结果：

```
zhy@zhy-virtual-machine:~$ gcc -o 3 3.c -Wall
zhy@zhy-virtual-machine:~$ ./3
pid = 5255 ppid = 5125!
pid = 5257 ppid = 5255!
pid = 5256 ppid = 5255!
pid = 5258 ppid = 5256!
pid = 5259 ppid = 5256!
```

（3）结果分析：

P1：5255

P2：5256

P3：5257

P4：5258

P5：5259

- **4、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。**

（1）使用 kill -9

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

int main(){
    pid_t pid;
    pid = fork();
    if (pid == 0){
        pid = fork();
        if(pid > 0)
            fork();
        while(1){
            printf ("pid = %d ppid = %d!\n",getpid(),getppid());
            sleep(1);
        }
    }
    else if (pid > 0){
        fork();
        while(1){
            printf ("pid = %d ppid = %d!\n",getpid(),getppid());
            sleep(1);
        }
    }
    else if (pid == -1)
        perror ("fork");
    return 0;
}
```

结果如下：

```
zhy@zhy-virtual-machine:~$ ./4
pid = 5959 ppid = 5914!
pid = 5962 ppid = 5959!
pid = 5960 ppid = 5959!
pid = 5961 ppid = 5960!
pid = 5963 ppid = 5960!
pid = 5959 ppid = 5914!
pid = 5963 ppid = 5960!
pid = 5962 ppid = 5959!
pid = 5960 ppid = 5959!
pid = 5961 ppid = 5960!
pid = 5959 ppid = 5914!
pid = 5963 ppid = 1427!
pid = 5962 ppid = 5959!
pid = 5961 ppid = 1427!
```

终止 p2 进程

```
zhy@zhy-virtual-machine:~$ kill -9 5960
zhy@zhy-virtual-machine:~$
```

（2）使用 exit（1）

```c
int main(){
        pid_t pid;
        pid = fork();
        if (pid > 0){
        fork();
        while(1){
            printf ("%d I am the child of pid=%d!\n",getpid(),getppid());
            sleep(10);
        }

        }
        else if (pid == 0){
        pid = fork();
        if(pid > 0){
            pid = fork();
            printf ("%d I am the child of pid=%d!\n",getpid(),getppid());
            sleep(1);
            if(pid > 0)
                exit(0);
        }
        while(1){
            printf ("%d I am the child of pid=%d!\n",getpid(),getppid());
            sleep(1);
        }
        }
        else if (pid == -1)
        perror ("fork");
        return 0;
}
```

结果如下：

```
zhy@zhy-virtual-machine:~$ ./4-2
6029 I am the child of pid=6016!
6031 I am the child of pid=6029!
6032 I am the child of pid=6030!
6030 I am the child of pid=6029!
6033 I am the child of pid=6030!
6029 I am the child of pid=6016!
6032 I am the child of pid=6030!
6031 I am the child of pid=6029!
6033 I am the child of pid=6030!
```

（3）段错误退出

```
    pid_t pid;
    pid = fork();
    if (pid > 0){
        fork();
        while(1){
            printf ("%d I am the child of pid=%d!\n",getpid(),getppid());
            sleep(1);
        }
    }
    else if (pid == 0){
        pid = fork();
        if(pid > 0){
            pid = fork();
            printf ("%d I am the child of pid=%d!\n",getpid(),getppid());
            sleep(1);
            if (pid > 0){
                int* p;
                *p = 1;
            }
        }
        while(1){
            printf ("%d I am the child of pid=%d!\n",getpid(),getppid());
            sleep(10);
        }
    }
    else if (pid == -1)
        perror ("fork");
    return 0;
}
```

结果如下：

```
zhy@zhy-virtual-machine:~$ ./4-3
6128 I am the child of pid=6016!
6130 I am the child of pid=6128!
6129 I am the child of pid=6128!
6131 I am the child of pid=6129!
6132 I am the child of pid=6129!
6130 I am the child of pid=6128!
6128 I am the child of pid=6016!
6132 I am the child of pid=6129!
6130 I am the child of pid=6128!
6128 I am the child of pid=6016!
6130 I am the child of pid=6128!
6128 I am the child of pid=6016!
```