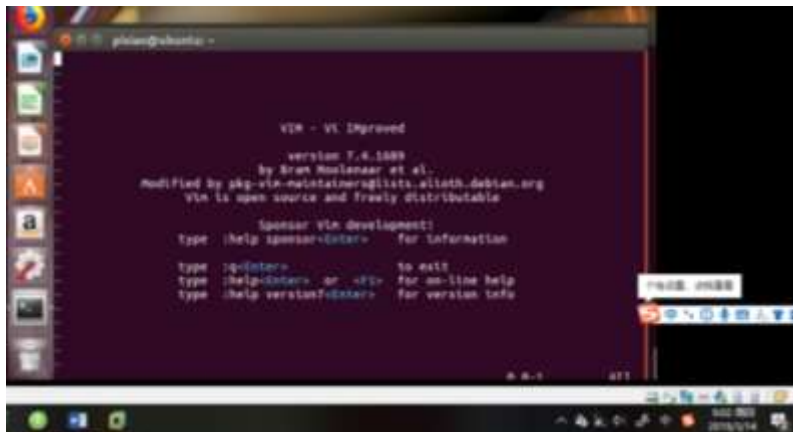


## 实验二 进程控制

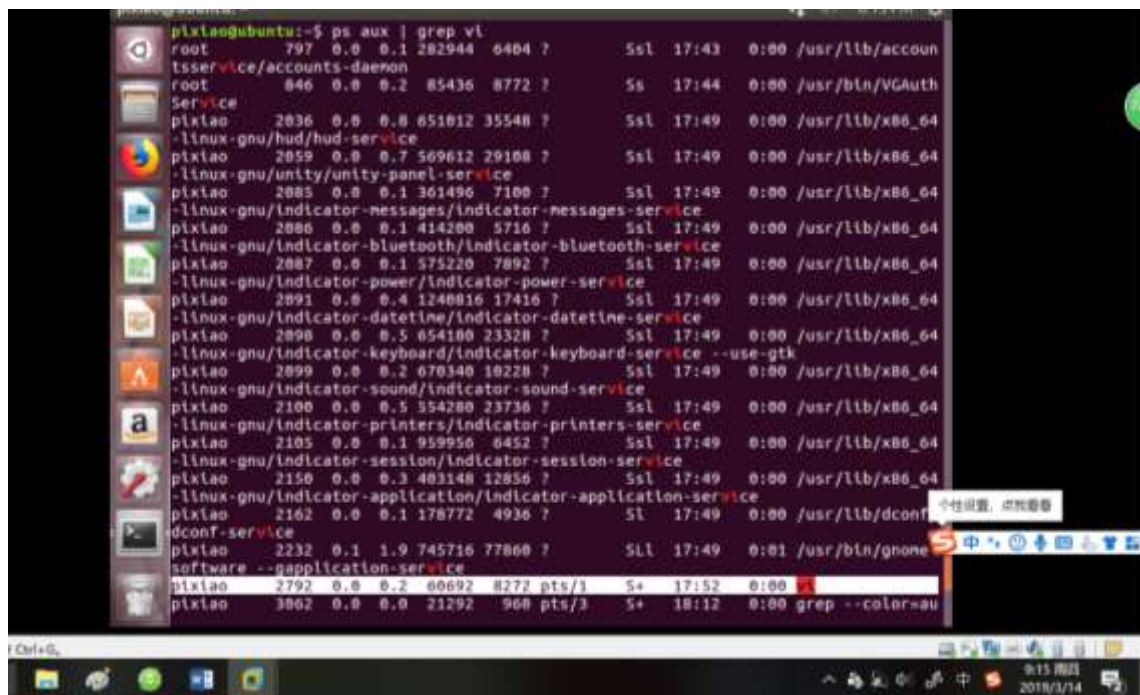
15282028 赵旭宏 安全 1601

### 实验题目及过程：

- 1. 打开一个 vi 进程。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。

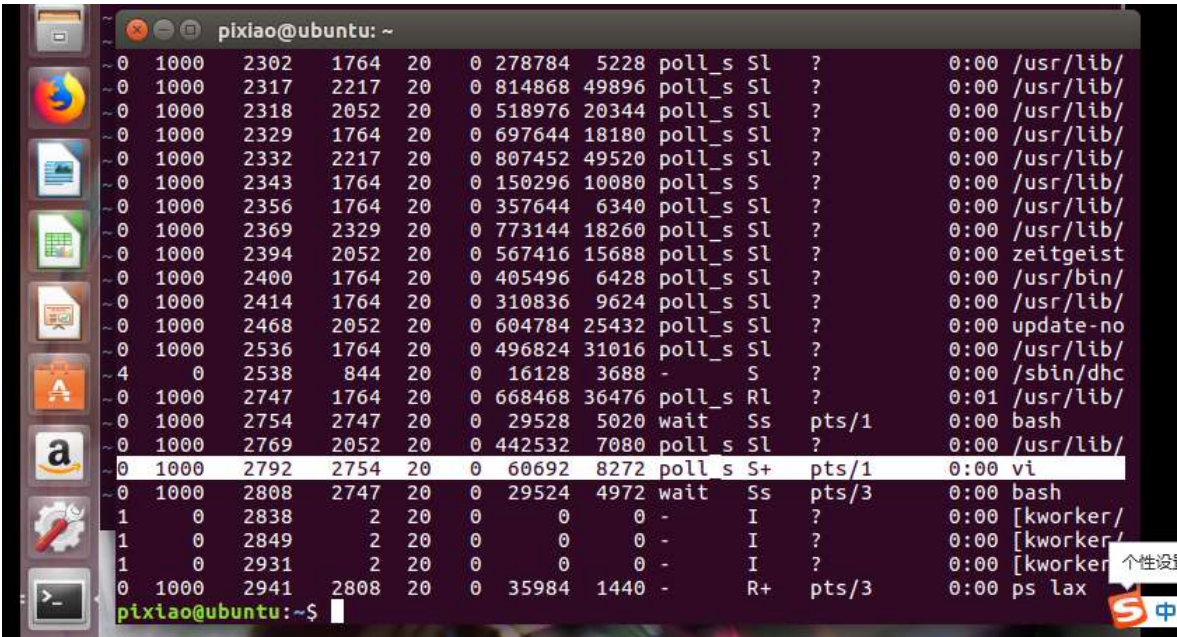


通过 ps 命令以及选择合适的参数，只显示名字为 vi 的进程：

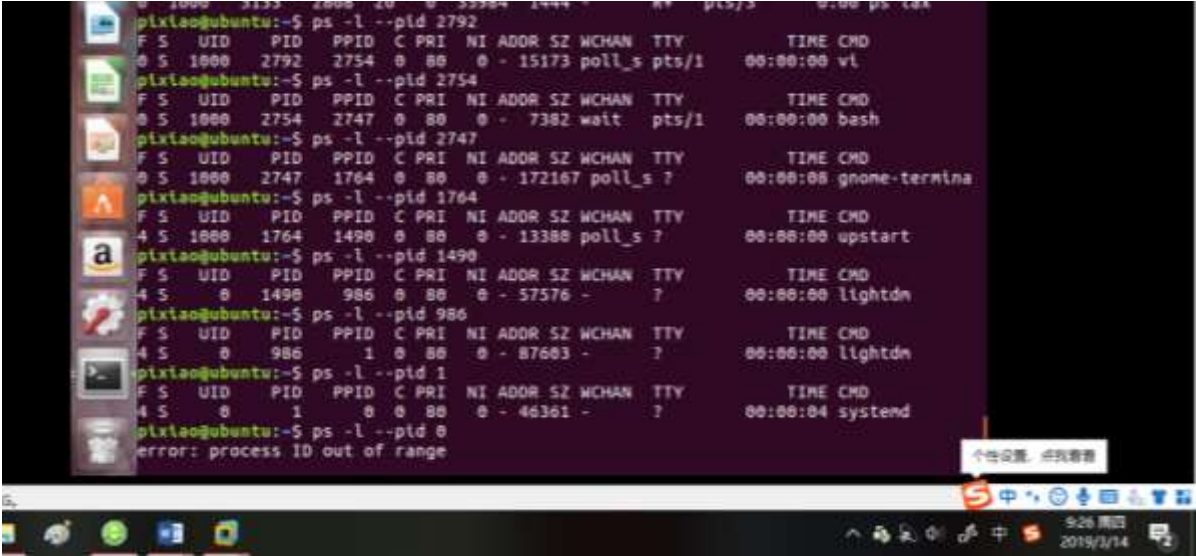


输入 ps -A 命令，查看所有进程的 ID，如下图可以看到，第一步打开的 vi 编辑器的 id

是 2792:



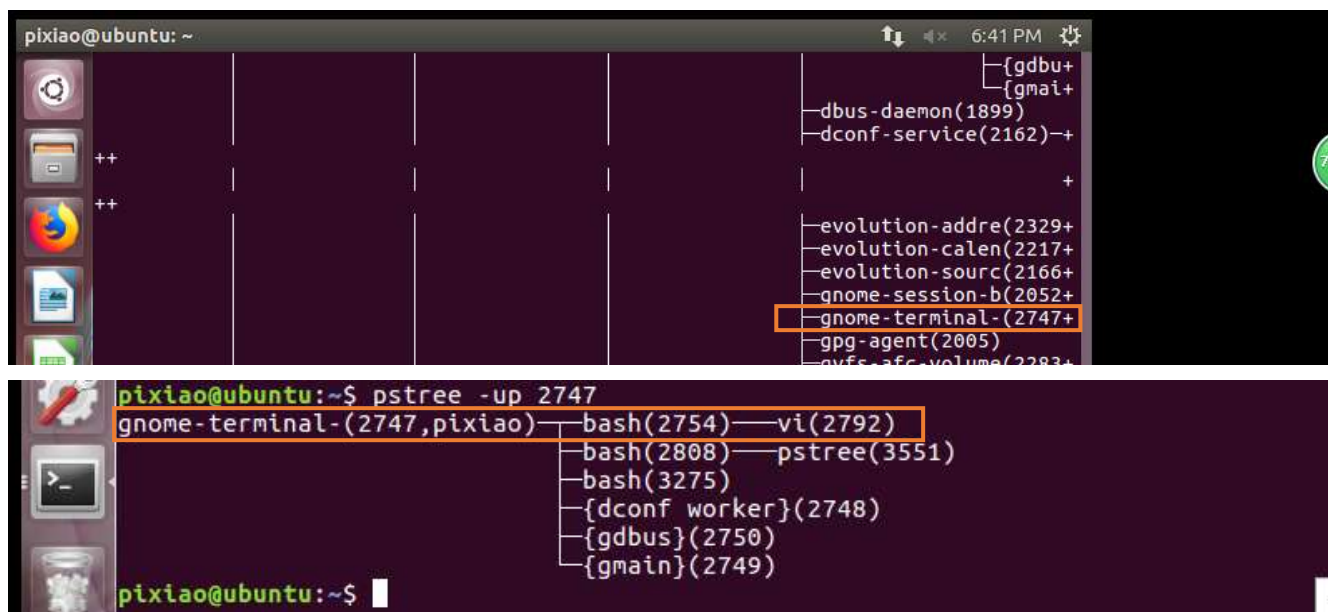
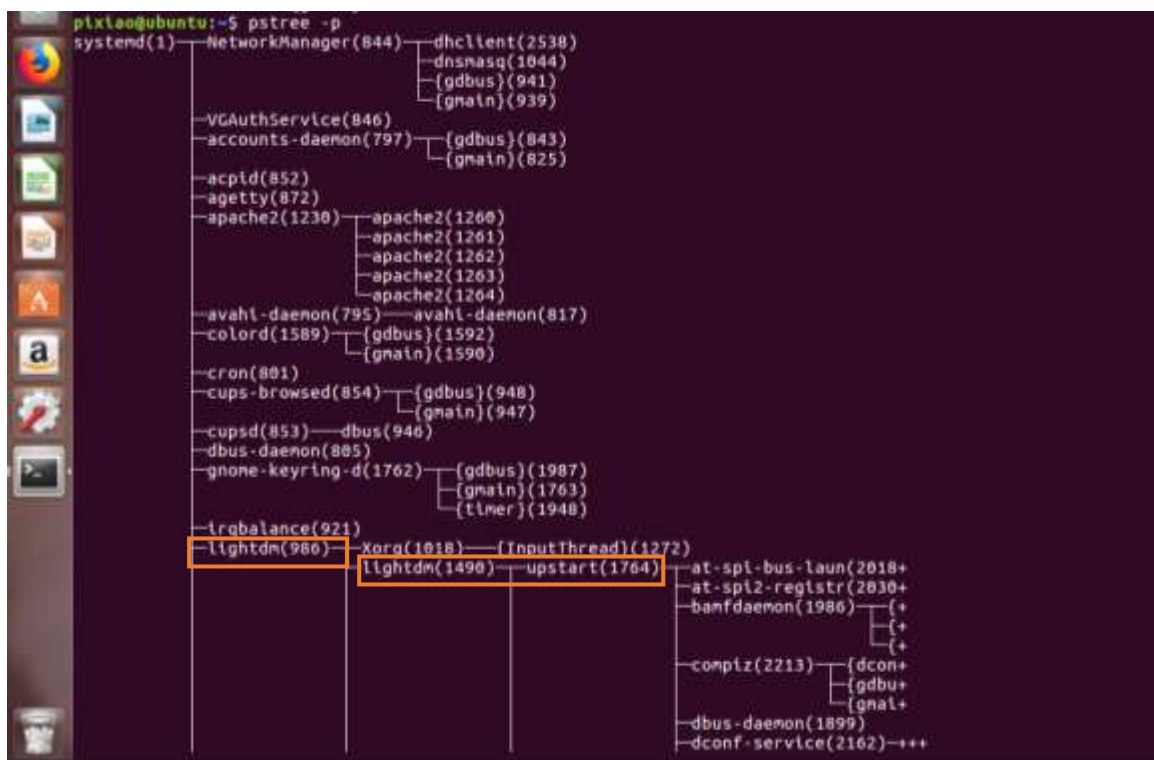
寻找 vi 进程的父进程，直到 init 进程为止：



记录过程中所有进程的 ID 和父进程 ID：

子进程 ID	父进程 ID
2792 (vi)	2754
2754	2747
2747	1764
1764	1490
1490	986
986	1

由 pstree 命令的得到的进程树如下：



在 Linux 下，每个进程有唯一的 PID 标识进程。PID 是一个从 1 到 32768 的正整数，其中 1 一般是特殊进程 init，其它进程从 2 开始依次编号。当用完 32768 后，从 2 重新开始。进程在 Linux 中呈树状结构，init 为根节点。将得到的进程树和由 pstree 命令的得到的进程树进行比较，发现是一致的。



- 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。

```

pixiao@ubuntu: ~/lab2
pixiao@ubuntu:~$ cd lab2
pixiao@ubuntu:~/lab2$ vim fork.c
pixiao@ubuntu:~/lab2$ gcc fork.c -o fork
pixiao@ubuntu:~/lab2$ ./fork
子进程的pid为4313
父进程的pid为4312

```

在另外一个终端中，通过 ps aux 等命令，查看 vi 进程及其父进程的运行状态。

```

pixiao@ubuntu:~/lab2$ cd ~
pixiao@ubuntu:~$ ps -aux |grep -w vi
pixiao  4200  0.0  0.0      0   0 ?        Z    05:48   0:00 [vi] <defunct>
pixiao  4206  0.0  0.0      0   0 ?        Z    05:49   0:00 [vi] <defunct>
pixiao  4209  0.0  0.0      0   0 ?        Z    05:49   0:00 [vi] <defunct>
pixiao  4313  0.0  0.0      0   0 pts/3    Z+   05:51   0:00 [vi] <defunct>
pixiao  10484 0.0  0.0  21292 1092 pts/19   S+   05:57   0:00 grep --color=auto -w vi

```

```

pixiao@ubuntu:~$ ps -ef |grep -w vi
pixiao  4200  4199  0 05:48 ?        00:00:00 [vi] <defunct>
pixiao  4206  4204  0 05:49 ?        00:00:00 [vi] <defunct>
pixiao  4209  4208  0 05:49 ?        00:00:00 [vi] <defunct>
pixiao  4313  4312  0 05:51 pts/3    00:00:00 [vi] <defunct>
pixiao  10518 5576  0 05:57 pts/19   00:00:00 grep --color=auto -w vi

```

## 了解 linux 下的 ps 命令及参数含义:

%CPU 进程的 cpu 占用率

%MEM 进程的内存占用率

VSZ 进程所使用的虚存的大小

RSS 进程使用的驻留集大小或者是实际内存的大小

TTY 与进程关联的终端 (tty)

STAT 检查的状态: 进程状态使用字符表示的, 如 R (running 正在运行或准备运行)、S (sleeping 睡眠)、I (idle 空闲)、Z (僵死)、D (不可中断的睡眠, 通常是 I/O)、P (等待交换页)、W (换出, 表示当前页面不在内存)、N (低优先级任务) T(terminate 终止)、W has no resident pages

START (进程启动时间和日期)

TIME ; (进程使用的总 cpu 时间)

COMMAND (正在执行的命令行命令)

NI (nice)优先级

PRI 进程优先级编号

PPID 父进程的进程 ID (parent process id)

SID 会话 ID (session id)

WCHAN 进程正在睡眠的内核函数名称; 该函数的名称是从 /root/system.map 文件中获得的。

FLAGS 与进程相关的数字标识

常用参数

-A 显示所有进程 (等价于-e) (utility)

-a 显示一个终端的所有进程, 除了会话引线

-N 忽略选择。

-d 显示所有进程, 但省略所有的会话引线(utility)

-x 显示没有控制终端的进程, 同时显示各个命令的具体路径。dx 不可合用。(utility)

-p pid 进程使用 cpu 的时间

-u uid or username 选择有效的用户 id 或者是用户名

-g gid or groupname 显示组的所有进程。

U username 显示该用户下的所有进程, 且显示各个命令的详细路径。如: ps U zhang;(utility)

-f 全部列出, 通常和其他选项联用。如: ps -fa or ps -fx and so on.

-l 长格式 (有 F,wchan,C 等字段)

-j 作业格式

-o 用户自定义格式。

v 以虚拟存储器格式显示

s 以信号格式显示

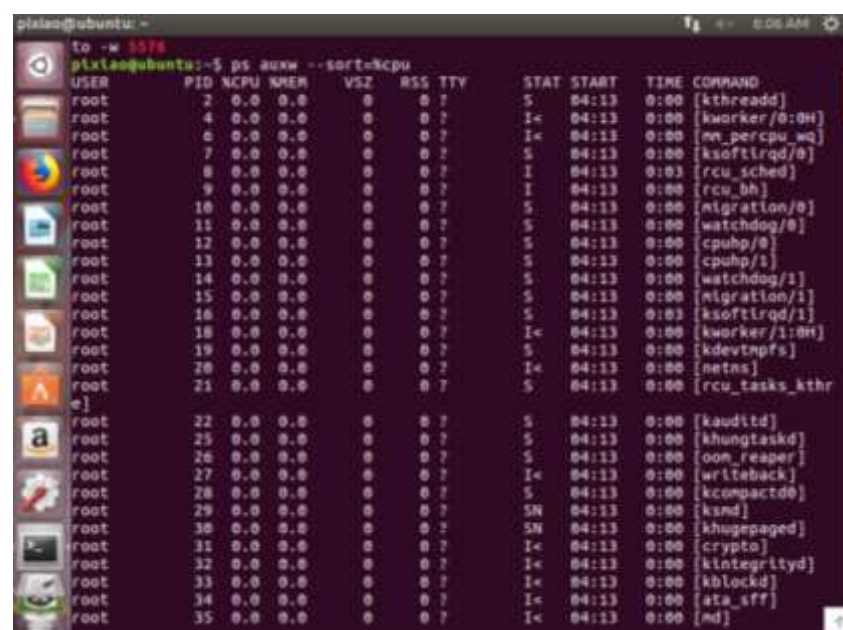
-m 显示所有的线程

-H 显示进程的层次(和别的命令合用, 如: ps -Ha) (utility)

e 命令之后显示环境 (如: ps -d e; ps -a e) (utility)

h 不显示第一行

使用 ps -aux 的命令对所有进程占用 cpu 率进行排序:



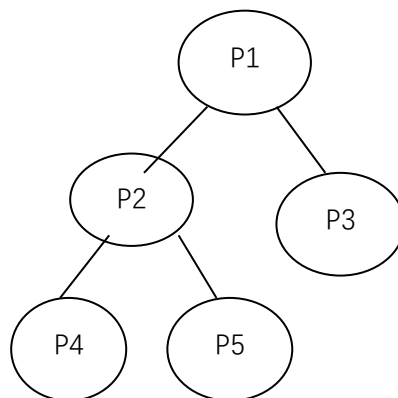
```
to -w 55Fs
pi@ubuntu:~$ ps aux --sort=pcpu
USER      PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         2  0.0  0.0      0     0 ?        S    04:13   0:00 [kthreadd]
root         4  0.0  0.0      0     0 ?        I<   04:13   0:00 [kworker/0:0H]
root         6  0.0  0.0      0     0 ?        I<   04:13   0:00 [nv_percpu_wq]
root         7  0.0  0.0      0     0 ?        S    04:13   0:00 [ksoftirqd/0]
root         8  0.0  0.0      0     0 ?        I    04:13   0:00 [rcu_sched]
root         9  0.0  0.0      0     0 ?        I    04:13   0:00 [rcu_bh]
root        10  0.0  0.0      0     0 ?        S    04:13   0:00 [migration/0]
root        11  0.0  0.0      0     0 ?        S    04:13   0:00 [watchdog/0]
root        12  0.0  0.0      0     0 ?        S    04:13   0:00 [cpuhp/0]
root        13  0.0  0.0      0     0 ?        S    04:13   0:00 [cpuhp/1]
root        14  0.0  0.0      0     0 ?        S    04:13   0:00 [watchdog/1]
root        15  0.0  0.0      0     0 ?        S    04:13   0:00 [migration/1]
root        16  0.0  0.0      0     0 ?        S    04:13   0:00 [ksoftirqd/1]
root        18  0.0  0.0      0     0 ?        I<   04:13   0:00 [kworker/1:0H]
root        19  0.0  0.0      0     0 ?        S    04:13   0:00 [kdevtmpfs]
root        20  0.0  0.0      0     0 ?        I<   04:13   0:00 [netns]
root        21  0.0  0.0      0     0 ?        S    04:13   0:00 [rcu_tasks_kthr]
root        22  0.0  0.0      0     0 ?        S    04:13   0:00 [kauditd]
root        25  0.0  0.0      0     0 ?        S    04:13   0:00 [khungtaskd]
root        26  0.0  0.0      0     0 ?        S    04:13   0:00 [oom_reaper]
root        27  0.0  0.0      0     0 ?        I<   04:13   0:00 [writeback]
root        28  0.0  0.0      0     0 ?        S    04:13   0:00 [kcompactd0]
root        29  0.0  0.0      0     0 ?        SN   04:13   0:00 [ksmd]
root        30  0.0  0.0      0     0 ?        SN   04:13   0:00 [khugepaged]
root        31  0.0  0.0      0     0 ?        I<   04:13   0:00 [crypto]
root        32  0.0  0.0      0     0 ?        I<   04:13   0:00 [kintegrityd]
root        33  0.0  0.0      0     0 ?        I<   04:13   0:00 [kblockd]
root        34  0.0  0.0      0     0 ?        I<   04:13   0:00 [ata_sff]
root        35  0.0  0.0      0     0 ?        I<   04:13   0:00 [nd]
```

执行 top 命令，查看 cpu 占用率：

```
pixiao@ubuntu:~$ top
top - 06:08:57 up 1:55, 1 user, load average: 3.12, 3.32, 2.63
Tasks: 255 total, 4 running, 181 sleeping, 0 stopped, 4 zombie
%Cpu(s): 100.0 us, 0.0 sy, 0.0 ni, 0.0 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
KiB Mem : 4015664 total, 799852 free, 841676 used, 2374136 buff/cache
KiB Swap: 4191228 total, 4191228 free, 0 used, 2804628 avail Mem

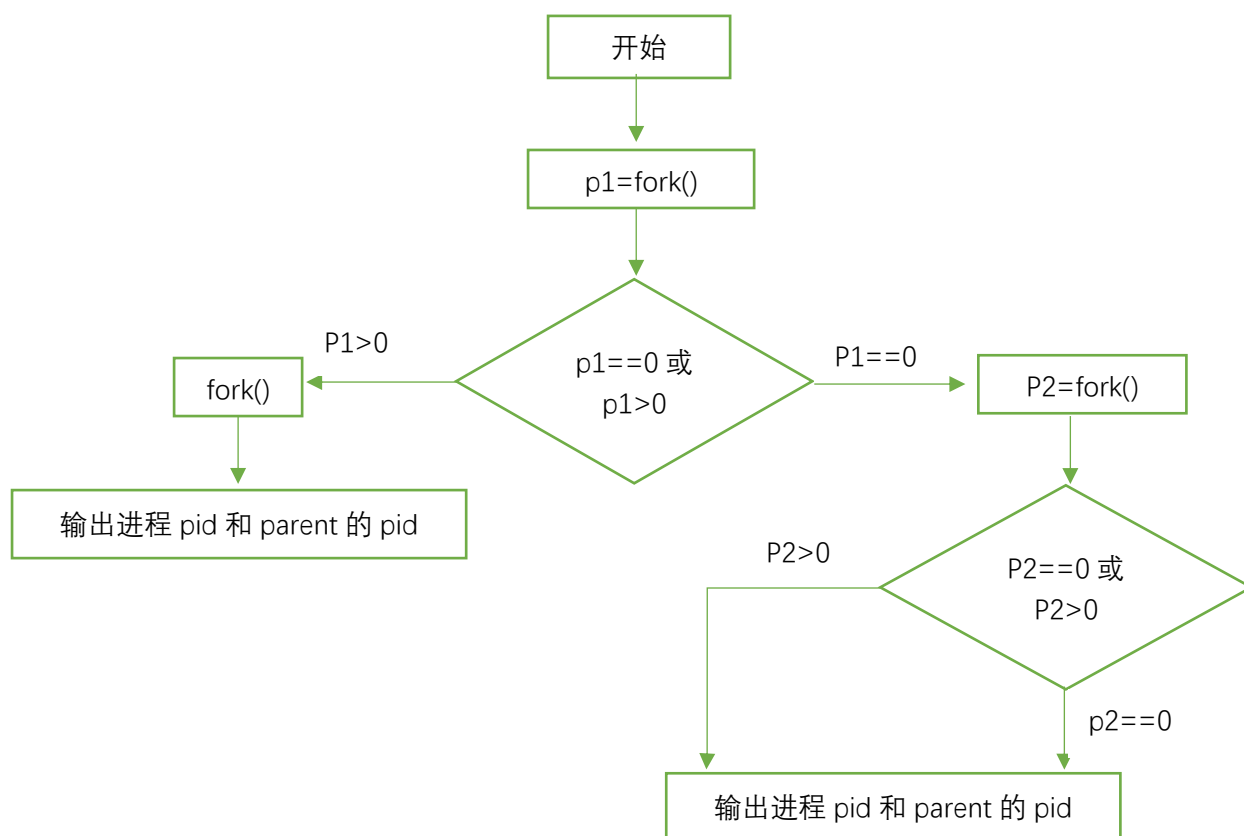
  PID USER      PR  NI  VIRT  RES  SHR  S  %CPU  %MEM    TIME+  COMMAND
 4208 pixiao    20   0   4220   636   568  R   72.0   0.0   11:57.78 a.out
 4204 pixiao    20   0   4220   776   708  R   65.0   0.0   12:01.02 a.out
 4199 pixiao    20   0   4220   676   608  R   62.7   0.0   12:12.52 a.out
1080 mysql      20   0 1173600 135436 16332  S    0.3   3.4   0:23.66 mysqld
2430 pixiao    20   0 541956 39104 29748  S    0.3   1.0   0:17.59 vntoolsd
2915 pixiao    20   0 689660 45676 34708  S    0.3   1.1   0:07.99 gnome-tern+
   1 root       20   0   119900 5960 3944  S    0.0   0.1   0:13.31 systemd
   2 root       20   0      0      0      0  S    0.0   0.0   0:00.13 kthreadd
   4 root       0 -20      0      0      0  I    0.0   0.0   0:00.00 kworker/0:0
   6 root       0 -20      0      0      0  I    0.0   0.0   0:00.00 mm_percpu_+
   7 root       20   0      0      0      0  S    0.0   0.0   0:00.48 ksoftirqd/0
   8 root       20   0      0      0      0  I    0.0   0.0   0:03.21 rcu_sched
   9 root       20   0      0      0      0  I    0.0   0.0   0:00.00 rcu_bh
  10 root       rt    0      0      0      0  S    0.0   0.0   0:00.05 migration/0
  11 root       rt    0      0      0      0  S    0.0   0.0   0:00.07 watchdog/0
  12 root       20   0      0      0      0  S    0.0   0.0   0:00.00 cpuhp/0
  13 root       20   0      0      0      0  S    0.0   0.0   0:00.00 cpuhp/1
  14 root       rt    0      0      0      0  S    0.0   0.0   0:00.02 watchdog/1
  15 root       rt    0      0      0      0  S    0.0   0.0   0:00.03 migration/1
  16 root       20   0      0      0      0  S    0.0   0.0   0:03.36 ksoftirqd/1
  18 root       0 -20      0      0      0  I    0.0   0.0   0:00.00 kworker/1:0
  19 root       20   0      0      0      0  S    0.0   0.0   0:00.02 kdevtmpfs
```

- 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。



```
pixiao@ubuntu:~$ ./3.out
parent with pid 11602,parent pid11601.
child2 with pid 11603,parent pid11602.
child4 with pid 11604,parent pid11603.
child5 with pid 11605,parent pid11603.
child3 with pid 11606,parent pid11602.
pixiao@ubuntu:~$
```

流程图：



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,观察发现,终止 p2 进程后, p4, p5 进程 fork 失败。

```
pixiao@ubuntu: ~/lab2
#include <stdio.h>
#include <stdlib.h>
int main(){
    int p1,p2,p3,p4,p5;
    while ((p1=fork())!=-1);
    if(!p1){
        printf("parent with pid %d,parent pid%d.\n",getpid(),getppid());
        while((p2=fork())!=-1);
        if(!p2){
            printf("child2 with pid %d,parent pid%d.\n",getpid(),getppid());
            exit(0);
        }
        while((p4=fork())!=-1);
    }
}
```

```
pixiao@ubuntu:~/lab2$ ./fork2
parent with pid 15812,parent pid15811.
child2 with pid 15813,parent pid15812.
child3 with pid 15814,parent pid15812.
pixiao@ubuntu:~/lab2$
```

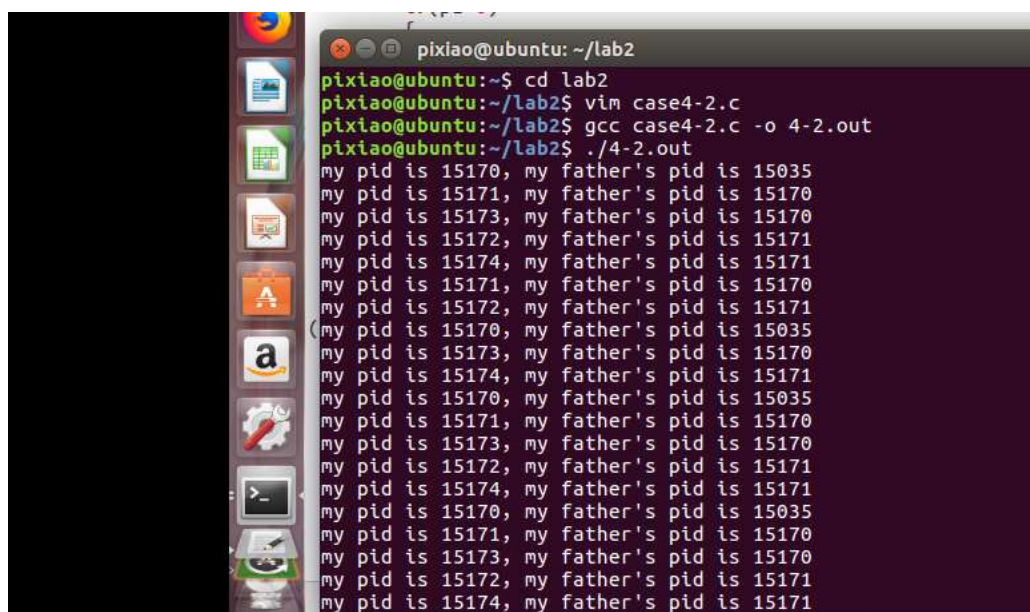


采用 kill -9:



```
case4-1.c:29:3: warning: useless type name in empty declaration
pid_t;
^
pixiao@ubuntu:~/lab2$ ./4-1.out
my pid is 15207, my father's pid is 15190
my pid is 15208, my father's pid is 15207
my pid is 15211, my father's pid is 15208
my pid is 15209, my father's pid is 15207
my pid is 15210, my father's pid is 15208
my pid is 15207, my father's pid is 15190
my pid is 15208, my father's pid is 15207
my pid is 15211, my father's pid is 15208
my pid is 15209, my father's pid is 15207
my pid is 15210, my father's pid is 15208
my pid is 15208, my father's pid is 15207
my pid is 15207, my father's pid is 15190
my pid is 15211, my father's pid is 15208
my pid is 15209, my father's pid is 15207
```

自己正常退出 exit(): p2 在执行十次以后输出就结束。



```
pixiao@ubuntu:~/lab2
pixiao@ubuntu:~$ cd lab2
pixiao@ubuntu:~/lab2$ vim case4-2.c
pixiao@ubuntu:~/lab2$ gcc case4-2.c -o 4-2.out
pixiao@ubuntu:~/lab2$ ./4-2.out
my pid is 15170, my father's pid is 15035
my pid is 15171, my father's pid is 15170
my pid is 15173, my father's pid is 15170
my pid is 15172, my father's pid is 15171
my pid is 15174, my father's pid is 15171
my pid is 15171, my father's pid is 15170
my pid is 15172, my father's pid is 15171
my pid is 15170, my father's pid is 15035
my pid is 15173, my father's pid is 15170
my pid is 15174, my father's pid is 15171
my pid is 15170, my father's pid is 15035
my pid is 15171, my father's pid is 15170
my pid is 15173, my father's pid is 15170
my pid is 15172, my father's pid is 15171
my pid is 15174, my father's pid is 15171
my pid is 15170, my father's pid is 15035
my pid is 15171, my father's pid is 15170
my pid is 15173, my father's pid is 15170
my pid is 15172, my father's pid is 15171
my pid is 15174, my father's pid is 15171
```

段错误退出: 在进程 p2 的进程段里面定义一个野指针, 野指针没有初始化会产生段错误导致进程退出。后面 15433 就消失了。



