

北京交通大学

《操作系统》实验报告

实验二：进程控制

学 号： 16281053

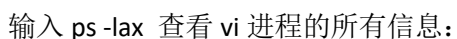
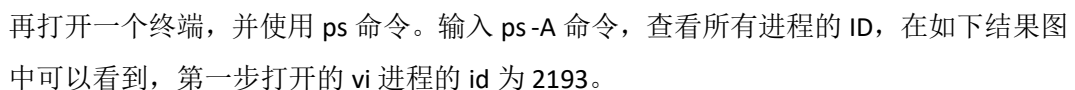
姓 名： 杨瑗彤

专 业： 计算机科学与技术

班 级： 计科 1601

提交日期： 2019 年 03 月 23 日

首先打开终端，开启一个新的 vi 进程，结果如下：



```
rdj@rdj-VirtualBox: ~  
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)  
0 1000 1946 1542 20 0 239236 1412 poll_s Sl+ tty2 0:00 /usr/lib/gn  
0 1000 1948 1542 20 0 892808 12088 poll_s Sl+ tty2 0:00 /usr/lib/ev  
0 1000 1953 1542 20 0 878192 46276 poll_s Sl+ tty2 0:01 nautilus-de  
0 1000 1975 1672 20 0 339468 2484 poll_s Sl ? 0:00 /usr/lib/gv  
0 1000 1981 1486 20 0 829836 0 poll_s Ssl ? 0:00 /usr/lib/ev  
0 1000 1989 1486 20 0 161396 2648 poll_s Sl ? 0:00 /usr/lib/dc  
0 1000 1995 1486 20 0 750196 0 poll_s Ssl ? 0:00 /usr/lib/ev  
0 1000 2053 1725 20 0 180424 952 poll_s Sl tty2 0:00 /usr/lib/ib  
0 1000 2122 1486 20 0 178180 36 poll_s Ssl ? 0:00 /usr/lib/gv  
0 1000 2140 1486 20 0 723680 17300 io_sch Dsl ? 0:00 /usr/lib/gn  
0 1000 2149 2140 20 0 21364 72 do_wai Ss pts/0 0:00 bash  
0 1000 2188 1725 20 0 262684 1176 poll_s Sl tty2 0:00 /usr/lib/ib  
0 1000 2193 2149 20 0 27080 4 poll_s S+ pts/0 0:00 vi  
0 1000 2209 2140 20 0 21324 2276 do_wai Ss pts/1 0:00 bash  
1 0 2280 2 0 -20 0 - S< ? 0:00 [loop13]  
0 1000 2321 1542 20 0 380804 22416 poll_s Sl+ tty2 0:00 update-noti  
0 1000 2323 1542 20 0 910160 114268 poll_s Sll+ tty2 0:03 /usr/bin/gn  
4 0 2360 1 20 0 628120 19284 - Ssl ? 0:01 /usr/lib/sn  
0 1000 2455 1542 20 0 684252 33204 poll_s Sl+ tty2 0:00 /usr/lib/de  
4 0 2466 1 20 0 434820 21296 - Ssl ? 0:00 /usr/lib/fw  
4 0 2495 1 20 0 266420 9292 - Ssl ? 0:00 /usr/lib/bo  
0 1000 2557 1 30 10 575832 143468 poll_s SNl+ tty2 0:03 /usr/bin/py  
0 1000 2578 2209 20 0 30124 1328 - R+ pts/1 0:00 ps -lax  
rdj@rdj-VirtualBox:~$
```

UID 是用户号, 为 2193;
PID 就是 vi 进程的进程号, 为 2193,
PPID 就是 vi 进程的父进程号,为 2149;
PRI 是内核调度优先级, 为 20;
NI 是进程优先级, 为缺省值 0;

输入 `ps -ef|grep vi` 命令, 搜索 vi 进程的父进程, 并逐次对父进程进行此操作, 直到寻找到进程号为 1 的 init 进程为止, 搜寻过程见下图。

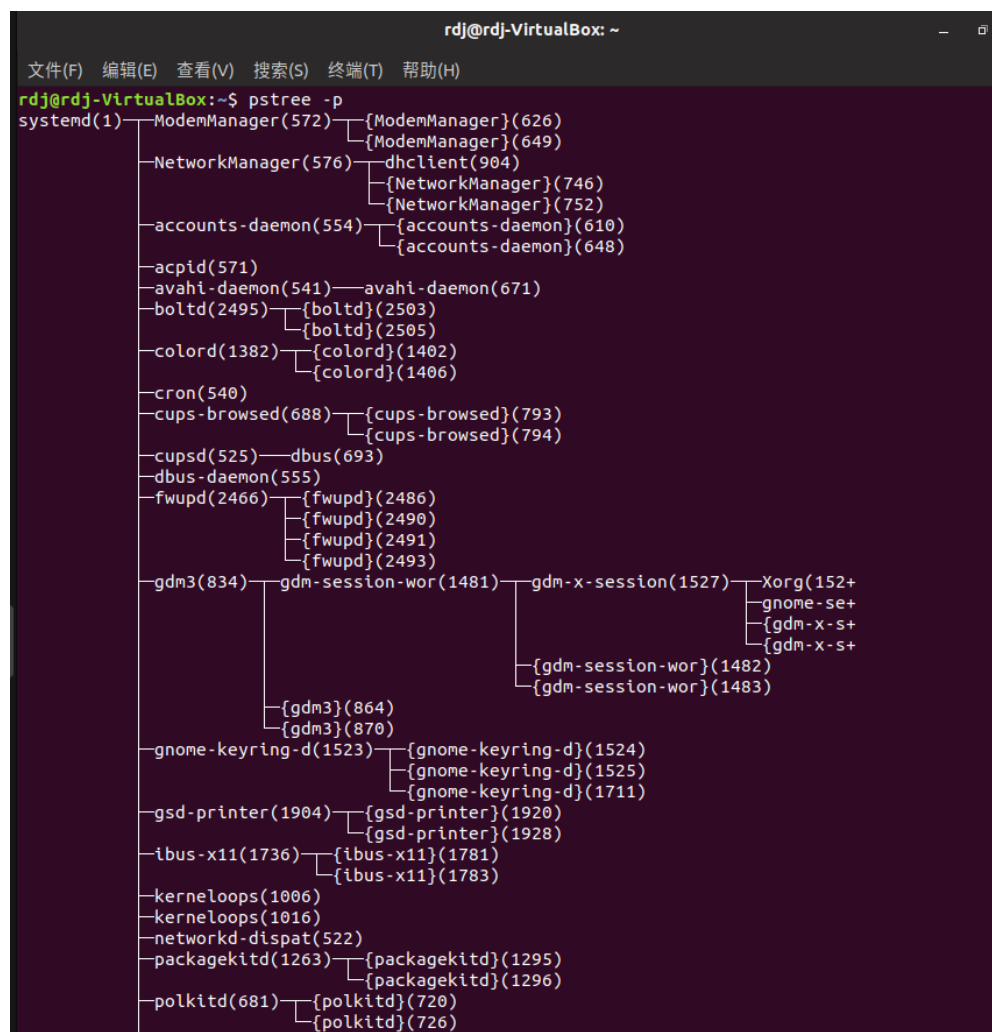
```
rdj@rdj-VirtualBox: ~  
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)  
4 0 2495 1 20 0 266420 9292 - Ssl ? 0:00 /usr/lib/bo  
0 1000 2557 1 30 10 575832 143468 poll_s SNl+ tty2 0:03 /usr/bin/py  
0 1000 2578 2209 20 0 30124 1328 - R+ pts/1 0:00 ps -lax  
rdj@rdj-VirtualBox:~$ grep  
用法: grep [选项]... PATTERN [FILE]...  
试用 'grep --help' 来获得更多的信息。  
rdj@rdj-VirtualBox:~$ grep PID vi.txt  
grep: vi.txt: 没有那个文件或目录  
rdj@rdj-VirtualBox:~$ grep PID vi.txt  
rdj@rdj-VirtualBox:~$ grep PID vi  
grep: vi: 没有那个文件或目录  
rdj@rdj-VirtualBox:~$ ps -ef|grep vi  
root      554      1  0 08:38 ?          00:00:00 /usr/lib/accountsservice/account  
s-daemon  
rdj      1826  1486  0 08:39 ?          00:00:00 /usr/lib/gnome-online-accounts/g  
oa-identity-service  
rdj      1989  1486  0 08:39 ?          00:00:00 /usr/lib/dconf/dconf-service  
rdj      2193  2149  0 08:40 pts/0    00:00:00 vi  
rdj      2323  1542  0 08:40 tty2    00:00:03 /usr/bin/gnome-software --appli  
cation-service  
rdj      2699  1486  0 08:49 ?          00:00:01 /usr/bin/nautilus --gapplication  
-service  
rdj      2887  2209  0 08:54 pts/1    00:00:00 grep --color=auto vi  
rdj@rdj-VirtualBox:~$
```

```
rdj@rdj-VirtualBox: ~
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
grep: vi: 没有那个文件或目录
rdj@rdj-VirtualBox:~$ ps -ef|grep vi
root      554      1    0 08:38 ?           00:00:00 /usr/lib/accountsservice/account
s-daemon
rdj       1826   1486    0 08:39 ?           00:00:00 /usr/lib/gnome-online-accounts/g
oa-identity-service
rdj       1989   1486    0 08:39 ?           00:00:00 /usr/lib/dconf/dconf-service
rdj       2193   2149    0 08:40 pts/0       00:00:00 vi
rdj       2323   1542    0 08:40 tty2      00:00:03 /usr/bin/gnome-software --gappli
cation-service
rdj       2699   1486    0 08:49 ?           00:00:01 /usr/bin/nautilus --gapplication
-service
rdj       2887   2209    0 08:54 pts/1       00:00:00 grep --color=auto vi
rdj@rdj-VirtualBox:~$ ps -ef|grep 2149
rdj       2149   2140    0 08:39 pts/0       00:00:00 bash
rdj       2193   2149    0 08:40 pts/0       00:00:00 vi
rdj       2895   2209    0 09:00 pts/1       00:00:00 grep --color=auto 2149
rdj@rdj-VirtualBox:~$ ps -ef|grep 2140
rdj       2140   1486    0 08:39 ?           00:00:02 /usr/lib/gnome-terminal/gnome-te
rminal-server
rdj       2149   2140    0 08:39 pts/0       00:00:00 bash
rdj       2209   2140    0 08:40 pts/1       00:00:00 bash
rdj       2923   2209    0 09:01 pts/1       00:00:00 grep --color=auto 2140
rdj@rdj-VirtualBox:~$ ps -ef|grep 1486
```

```
rdj@rdj-VirtualBox: ~
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
rdj       2699   1486    0 08:49 ?           00:00:01 /usr/bin/nautilus --gapplication
-service
rdj       2887   2209    0 08:54 pts/1       00:00:00 grep --color=auto vi
rdj@rdj-VirtualBox:~$ ps -ef|grep 2149
rdj       2149   2140    0 08:39 pts/0       00:00:00 bash
rdj       2193   2149    0 08:40 pts/0       00:00:00 vi
rdj       2895   2209    0 09:00 pts/1       00:00:00 grep --color=auto 2149
rdj@rdj-VirtualBox:~$ ps -ef|grep 2140
rdj       2140   1486    0 08:39 ?           00:00:02 /usr/lib/gnome-terminal/gnome-te
rminal-server
rdj       2149   2140    0 08:39 pts/0       00:00:00 bash
rdj       2209   2140    0 08:40 pts/1       00:00:00 bash
rdj       2923   2209    0 09:01 pts/1       00:00:00 grep --color=auto 2140
rdj@rdj-VirtualBox:~$ ps -ef|grep 1486
rdj       1486      1    0 08:39 ?           00:00:00 /lib/systemd/systemd --user
rdj       1497   1486    0 08:39 ?           00:00:00 (sd-pam)
rdj       1538   1486    0 08:39 ?           00:00:00 /usr/bin/dbus-daemon --session -
-address=systemd: --nofork --nopidfile
--systemd-activation --syslog-only
rdj       1672   1486    0 08:39 ?           00:00:00 /usr/lib/gvfs/gvfsd
rdj       1678   1486    0 08:39 ?           00:00:00 /usr/lib/gvfs/gvfsd-fuse /run/us
er/1000/gvfs -f -o big_writes
rdj       1688   1486    0 08:39 ?           00:00:00 /usr/lib/at-spi2-core/at-spi-bus
-launcher
rdj       1696   1486    0 08:39 ?           00:00:00 /usr/lib/at-spi2-core/at-spi2-re
```

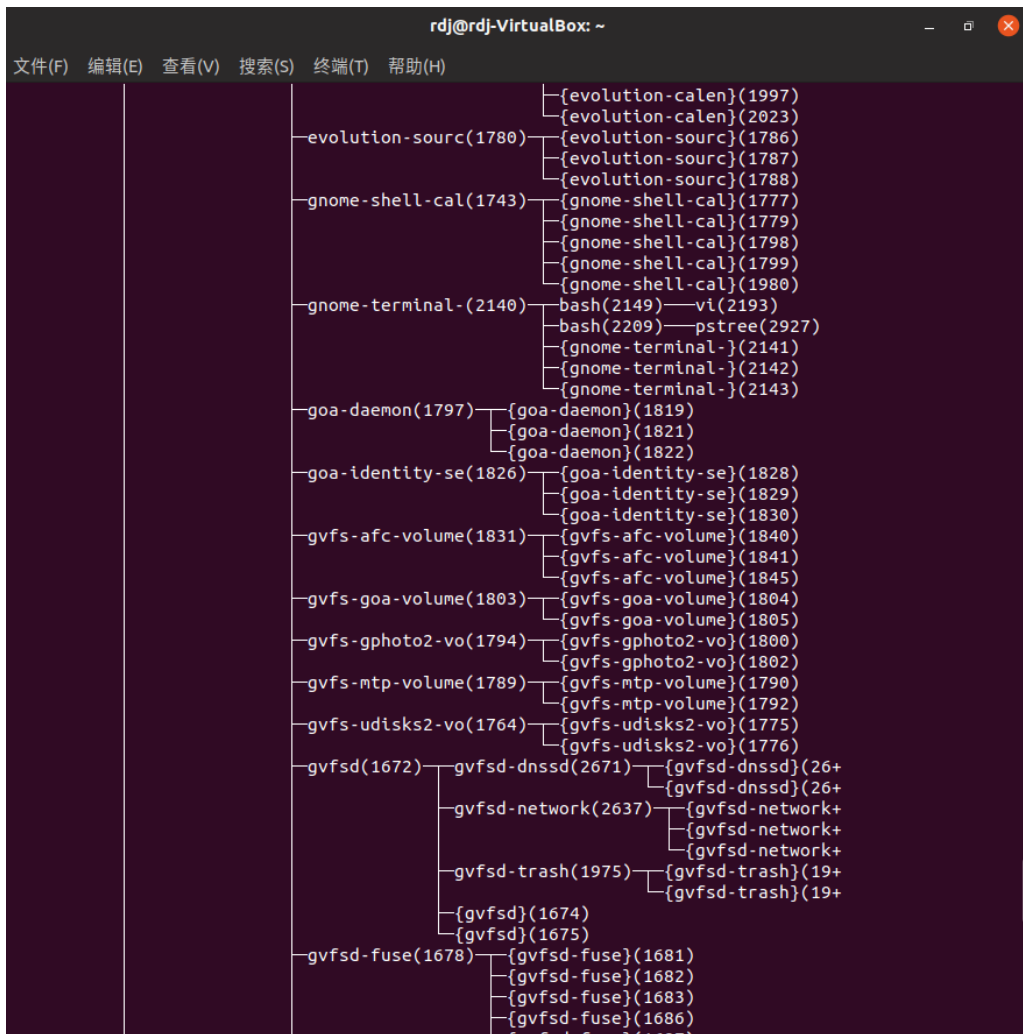
```
rdj@rdj-VirtualBox: ~
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
e-monitor
rdj       1797   1486    0 08:39 ?           00:00:00 /usr/lib/gnome-online-accounts/g
oa-daemon
rdj       1803   1486    0 08:39 ?           00:00:00 /usr/lib/gvfs/gvfs-goa-volume-mo
nitor
rdj       1826   1486    0 08:39 ?           00:00:00 /usr/lib/gnome-online-accounts/g
oa-identity-service
rdj       1831   1486    0 08:39 ?           00:00:00 /usr/lib/gvfs/gvfs-afc-volume-mo
nitor
rdj       1981   1486    0 08:39 ?           00:00:00 /usr/lib/evolution/evolution-cal
endar-factory
rdj       1989   1486    0 08:39 ?           00:00:00 /usr/lib/dconf/dconf-service
rdj       1995   1486    0 08:39 ?           00:00:00 /usr/lib/evolution/evolution-add
ressbook-factory
rdj       2122   1486    0 08:39 ?           00:00:00 /usr/lib/gvfs/gvfsd-metadata
rdj       2140   1486    0 08:39 ?           00:00:03 /usr/lib/gnome-terminal/gnome-te
rminal-server
rdj       2699   1486    0 08:49 ?           00:00:01 /usr/bin/nautilus --gapplication
-service
rdj       2742   1486    0 08:49 ?           00:00:00 /usr/bin/zeitgeist-daemon
rdj       2748   1486    0 08:49 ?           00:00:00 /usr/lib/zeitgeist/zeitgeist/zei
tgeist-fts
rdj       2925   2209    0 09:02 pts/1       00:00:00 grep --color=auto 1486
rdj@rdj-VirtualBox:~$
```

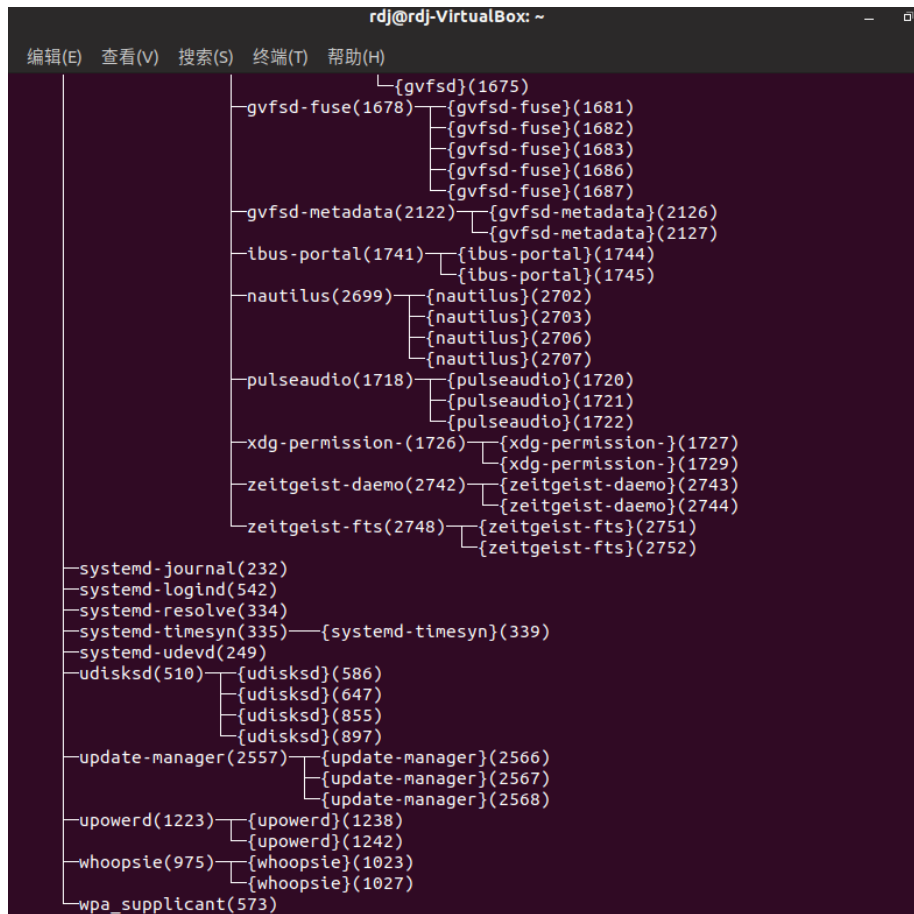
输入 `ps tree -p` 命令，查看进程树如下图：



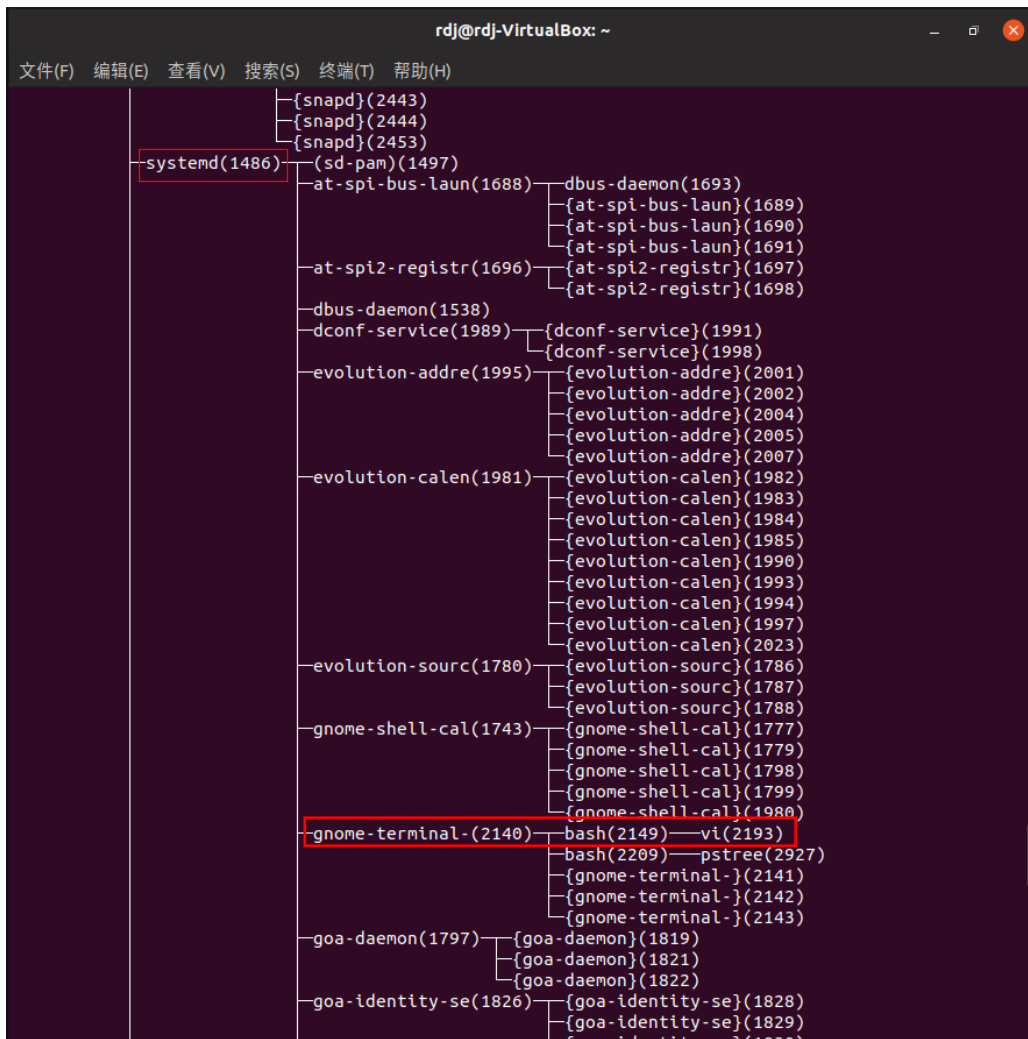
```
rdj@rdj-VirtualBox: ~
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)

--networkd-dispat(522)
--packagekitd(1263)
|   |--{packagekitd}(1295)
|   |--{packagekitd}(1296)
--polkitd(681)
|   |--{polkitd}(720)
|   |--{polkitd}(726)
--rsyslogd(523)
|   |--{rsyslogd}(677)
|   |--{rsyslogd}(678)
|   |--{rsyslogd}(679)
--rtkit-daemon(1155)
|   |--{rtkit-daemon}(1166)
|   |--{rtkit-daemon}(1167)
--snapd(2360)
|   |--{snapd}(2383)
|   |--{snapd}(2384)
|   |--{snapd}(2385)
|   |--{snapd}(2398)
|   |--{snapd}(2399)
|   |--{snapd}(2443)
|   |--{snapd}(2444)
|   |--{snapd}(2453)
--systemd(1486)
|   |--(sd-pam)(1497)
|   |--at-spi-bus-laun(1688)
|   |   |--dbus-daemon(1693)
|   |   |--{at-spi-bus-laun}(1689)
|   |   |--{at-spi-bus-laun}(1690)
|   |   |--{at-spi-bus-laun}(1691)
|   |--at-spi2-registr(1696)
|   |   |--{at-spi2-registr}(1697)
|   |   |--{at-spi2-registr}(1698)
|   |--dbus-daemon(1538)
|   |--dconf-service(1989)
|   |   |--{dconf-service}(1991)
|   |   |--{dconf-service}(1998)
|   |--evolution-addre(1995)
|   |   |--{evolution-addre}(2001)
|   |   |--{evolution-addre}(2002)
|   |   |--{evolution-addre}(2004)
|   |   |--{evolution-addre}(2005)
|   |   |--{evolution-addre}(2007)
|   |--evolution-calen(1981)
|   |   |--{evolution-calen}(1982)
|   |   |--{evolution-calen}(1983)
|   |   |--{evolution-calen}(1984)
|   |   |--{evolution-calen}(1985)
|   |   |--{evolution-calen}(1990)
|   |   |--{evolution-calen}(1993)
|   |   |--{evolution-calen}(1994)
|   |   |--{evolution-calen}(1997)
|   |   |--{evolution-calen}(2023)
|   |--evolution-sourc(1780)
|   |   |--{evolution-sourc}(1786)
|   |   |--{evolution-sourc}(1787)
|   |   |--{evolution-sourc}(1788)
```





在此进程树中，可以看到 vi 进程的各级父进程，与之前逐步查看父进程结果相同：



ps 命令向上寻找父进程，发现父子进程之间的关系很像一个链表，寻找父进程的过程就是不断寻找其前驱的过程。pstree 命令获得的进程树则更为详细地表现了进程之间的关系，从根部 init 出发，生长出其他进程。从根到叶子，与上述 ps 命令寻找过程是对应的。

Linux 系统中通过 ps、pstree 等命令查看系统进程状态。ps 命令提供了一个正在运行的进程的列表。pstree 命令可以以树状结构来显示所有的进程信息。一些命令参数如下：

- 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(和其它的命令合用，如：`ps -Ha`) (utility)

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

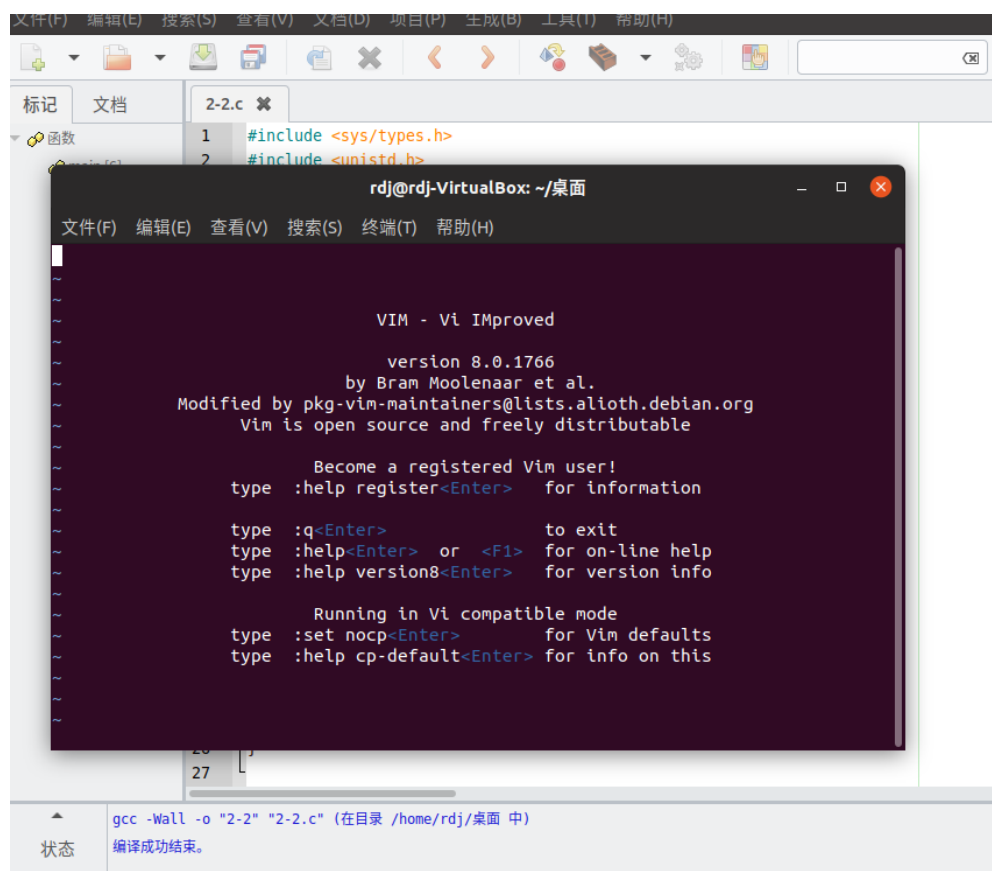
h 不显示第一行

- 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。

使用 `fork` 系统调用，创建子进程，在子进程中使用 `execl()` 函数打开 `vi` 编辑器，然后在父进程中执行一个空循环。代码见下图：

```
6] 2-2.c
1  #include <sys/types.h>
2  #include <unistd.h>
3  #include <stdio.h>
4  #include <stdlib.h>
5
6  int main()
7  {
8      //create process
9      pid_t pid;
10     pid=fork();
11     if (pid < 0)
12     {
13         perror("fork");
14     }
15     if (pid == 0)
16     {
17         //son process
18         pid = execl ("/usr/bin/vi", "vi", NULL);
19     }
20     else if(pid >0)
21     {
22         //father process
23         while(1){};
24     }
25     return 0;
26 }
27
```

打开终端运行此程序，结果见下图



可见子程序已经打开了 vi 编辑器，与预期结果相同。再打开另一终端，输入 ps-al 命令，

查看所有进程，以及此程序的子进程、父进程、vi 的关系。

```
rdj@rdj-VirtualBox: ~/桌面
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
rdj@rdj-VirtualBox:~/桌面$ ps -al
F S      UID      PID     PPID    C  PRI   NI  ADDR  SZ  WCHAN    TTY          TIME CMD
4 S      1000     1341     1339    1   80    0  - 115248 ep_pol  tty2         00:00:40 Xorg
0 S      1000     1354     1339    0   80    0  - 117239 poll_s  tty2         00:00:00 gnome-session-
0 S      1000     1523     1354    6   80    0  - 692565 poll_s  tty2         00:03:57 gnome-shell
0 S      1000     1560     1523    0   80    0  - 100807 poll_s  tty2         00:00:03 ibus-daemon
0 S      1000     1564     1560    0   80    0  - 63561  poll_s  tty2         00:00:00 ibus-dconf
0 S      1000     1565     1560    0   80    0  - 80481  poll_s  tty2         00:00:02 ibus-extension
0 S      1000     1569      1   0   80    0  - 61275  poll_s  tty2         00:00:00 ibus-x11
0 S      1000     1664     1354    0   80    0  - 102076 poll_s  tty2         00:00:00 gsd-power
0 S      1000     1665     1354    0   80    0  - 70040  poll_s  tty2         00:00:00 gsd-print-noti
0 S      1000     1667     1354    0   80    0  - 99262  poll_s  tty2         00:00:00 gsd-rfkill
0 S      1000     1668     1354    0   80    0  - 62361  poll_s  tty2         00:00:00 gsd-screensave
0 S      1000     1670     1354    0   80    0  - 104201 poll_s  tty2         00:00:00 gsd-sharing
0 S      1000     1675     1354    0   80    0  - 84962  poll_s  tty2         00:00:00 gsd-smartcard
0 S      1000     1677     1354    0   80    0  - 72121  poll_s  tty2         00:00:00 gsd-sound
0 S      1000     1681     1354    0   80    0  - 98489  poll_s  tty2         00:00:00 gsd-xsettings
0 S      1000     1683     1354    0   80    0  - 80860  poll_s  tty2         00:00:00 gsd-wacom
0 S      1000     1689     1354    0   80    0  - 63528  poll_s  tty2         00:00:00 gsd-a11y-setti
0 S      1000     1691     1354    0   80    0  - 61174  poll_s  tty2         00:00:00 gsd-clipboard
0 S      1000     1693     1354    0   80    0  - 90657  poll_s  tty2         00:00:00 gsd-datetime
0 S      1000     1697     1354    0   80    0  - 140419 poll_s  tty2         00:00:00 gsd-color
0 S      1000     1698     1354    0   80    0  - 98744  poll_s  tty2         00:00:00 gsd-keyboard
0 S      1000     1699     1354    0   80    0  - 82527  poll_s  tty2         00:00:00 gsd-housekeepi
0 S      1000     1701     1354    0   80    0  - 63529  poll_s  tty2         00:00:00 gsd-mouse
0 S      1000     1705     1354    0   80    0  - 170534 poll_s  tty2         00:00:00 gsd-media-keys
0 S      1000     1726      1   0   80    0  - 100450 poll_s  tty2         00:00:00 gsd-printer
0 S      1000     1788     1354    0   80    0  - 59809  poll_s  tty2         00:00:00 gsd-disk-utili
0 S      1000     1789     1354    0   80    0  - 186326 poll_s  tty2         00:00:00 evolution-alar
0 S      1000     1793     1354    0   80    0  - 226428 poll_s  tty2         00:00:02 nautilus-deskt
0 S      1000     1943     1560    0   80    0  - 45106  poll_s  tty2         00:00:00 ibus-engine-si
0 S      1000     2136     1560    0   80    0  - 65671  poll_s  tty2         00:00:00 ibus-engine-li
0 S      1000     2245     1354    0   80    0  - 95205  poll_s  tty2         00:00:01 update-notifie
0 S      1000     2248     1354    0   80    0  - 226464 poll_s  tty2         00:00:06 gnome-software
0 S      1000     2281     1523    1   80    0  - 212249 poll_s  tty2         00:01:14 geany
0 S      1000     2493     1354    0   80    0  - 171063 poll_s  tty2         00:00:00 deja-dup-monit
0 R      1000     3268     3260   83   80    0  - 579 -    pts/1         00:00:24 2-2
0 S      1000     3269     3268    0   80    0  - 6770  poll_s  pts/1         00:00:00 vi
0 R      1000     3281     3273    0   80    0  - 7531 -    pts/2         00:00:00 ps
rdj@rdj-VirtualBox:~/桌面$
```

根据 2-2 程序和 vi 的 pid 可以看出，我编写的 2-2 程序是 vi 编辑器的父进程，与题目相符。各个参数的含义为：

- UID 是用户号,;
- PID 是进程号;
- PPID 是进程的父进程号;
- PRI 是内核调度优先级;
- NI 是进程优先级，为缺省值 0;
- TTY 是终端的次要装置号码;
- TIME 为使用 cpu 的时间;

输入 top 命令，查看 CPU 占用率，结果如下图：

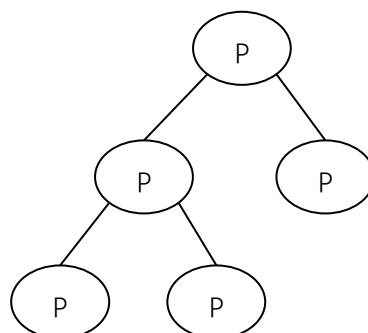
```

文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
top - 16:58:01 up 2:18, 1 user, load average: 1.16, 0.80, 0.39
任务: 181 total, 3 running, 178 sleeping, 0 stopped, 0 zombie
%Cpu(s): 99.3 us, 0.7 sy, 0.0 ni, 0.0 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
MiB Mem : 985.3 total, 71.1 free, 663.8 used, 250.3 buff/cache
MiB Swap: 472.5 total, 277.2 free, 195.2 used. 62.1 avail Mem

进程 USER      PR    NI   VIRT   RES   SHR  %CPU  %MEM   TIME+ COMMAND
3171 rdj        20     0   2316    676   612 R   96.4   0.1   3:41.17 2-2
1649 rdj        20     0 2838192 265408 62444 S    1.7  26.3   2:59.01 gnome-shell
3158 rdj        20     0 715816 39676 25720 S    0.7   3.9   0:02.01 gnome-terminal-
1466 rdj        20     0 502444 187668 129560 S    0.3  18.6   0:34.36 Xorg
3222 rdj        20     0 34808 3612 2924 R    0.3   0.4   0:00.05 top
1 root        20     0 196868 3196 1776 R    0.0   0.3   0:03.12 systemd
2 root        20     0      0      0      0 S    0.0   0.0   0:00.00 kthreadd
3 root         0 -20      0      0      0 I    0.0   0.0   0:00.00 rcu_gp
4 root         0 -20      0      0      0 I    0.0   0.0   0:00.00 rcu_par_gp
6 root         0 -20      0      0      0 I    0.0   0.0   0:00.00 kworker/0:0H-kblockd
8 root         0 -20      0      0      0 I    0.0   0.0   0:00.00 mm_percpu_wq
9 root        20     0      0      0      0 S    0.0   0.0   0:00.33 ksoftirqd/0
10 root       20     0      0      0      0 I    0.0   0.0   0:01.32 rcu_sched
11 root       20     0      0      0      0 I    0.0   0.0   0:00.00 rcu_bh
12 root       rt     0      0      0      0 S    0.0   0.0   0:00.00 migration/0
13 root       rt     0      0      0      0 S    0.0   0.0   0:00.04 watchdog/0
14 root       20     0      0      0      0 S    0.0   0.0   0:00.00 cpuhp/0
15 root       20     0      0      0      0 S    0.0   0.0   0:00.00 kdevtmpfs
16 root         0 -20      0      0      0 I    0.0   0.0   0:00.00 netns
17 root       20     0      0      0      0 S    0.0   0.0   0:00.00 rcu_tasks_kthre
18 root       20     0      0      0      0 S    0.0   0.0   0:00.00 kauditd
19 root       20     0      0      0      0 S    0.0   0.0   0:00.00 khungtaskd
20 root       20     0      0      0      0 S    0.0   0.0   0:00.00 oom_reaper
21 root         0 -20      0      0      0 I    0.0   0.0   0:00.00 writeback
22 root       20     0      0      0      0 S    0.0   0.0   0:00.00 kcompactd0
23 root       25     5      0      0      0 S    0.0   0.0   0:00.00 ksm
24 root       39    19      0      0      0 S    0.0   0.0   0:00.00 khugepaged
25 root         0 -20      0      0      0 I    0.0   0.0   0:00.00 crypto
26 root         0 -20      0      0      0 I    0.0   0.0   0:00.00 kintegrityd
27 root         0 -20      0      0      0 I    0.0   0.0   0:00.00 kblockd
28 root         0 -20      0      0      0 I    0.0   0.0   0:00.00 ata_sff
29 root         0 -20      0      0      0 I    0.0   0.0   0:00.00 md
30 root         0 -20      0      0      0 I    0.0   0.0   0:00.00 edac-poller
31 root         0 -20      0      0      0 I    0.0   0.0   0:00.00 devfreq_wq
32 root       rt     0      0      0      0 S    0.0   0.0   0:00.00 watchdogd
36 root       20     0      0      0      0 S    0.0   0.0   0:03.07 kswapd0
37 root         0 -20      0      0      0 I    0.0   0.0   0:00.00 kworker/u3:0
38 root       20     0      0      0      0 S    0.0   0.0   0:00.00 ecryptfs-kthrea
83 root         0 -20      0      0      0 I    0.0   0.0   0:00.00 kthrotld

```

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。



从上到下、从左至右将 5 个进程依次命名为 p1、p2、p3、p4、p5。

编写代码如下图：

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <unistd.h>
4 #include <sys/types.h>
5 #include <wait.h>
6 int main(void) {
7     int p2,p3,p4,p5;
8     while ((p2=fork())!=-1); //create p2
9     if (!p2) {
10         printf("I am p2,with ID[%d],    my father is p1 with ID[%d].\n",getpid(),getppid());
11
12         while ((p4=fork())!=-1); //create p4
13         if (!p4) {
14             printf("I am p4,with ID[%d],    my father is p2 with ID[%d].\n",getpid(),getppid());
15             exit(0);
16         }
17         else wait(0); //p2 gets waiting
18
19         while ((p5=fork())!=-1); //create p5
20         if (!p5) {
21             printf("I am p5,with ID[%d],    my father is p2 with ID[%d].\n",getpid(),getppid());
22             exit(0);
23         }
24         else wait(0); //p2 gets waiting
25         exit(0);
26     }
27 }

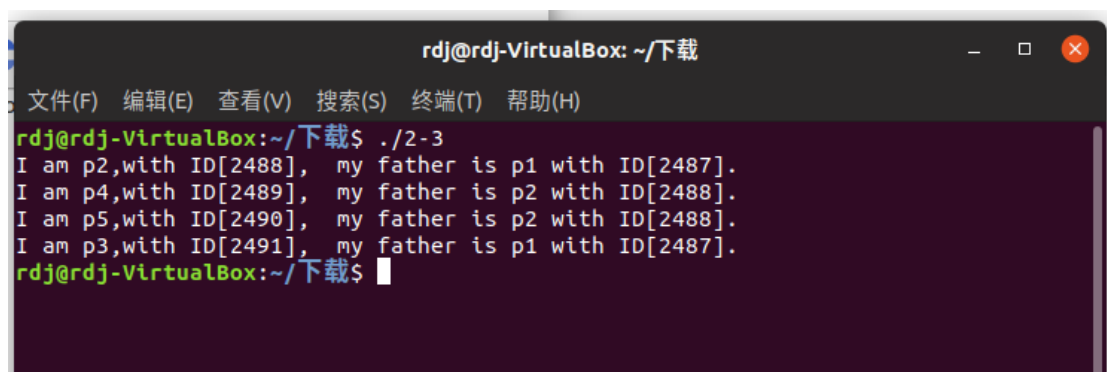
else wait(0); //p1 gets waiting

while ((p3=fork())!=-1); //create p3
if(!p3){
    printf("I am p3,with ID[%d],    my father is p1 with ID[%d].\n",getpid(),getppid());
    exit(0);
}
else wait(0); //p1 gets waiting

return 0;
}

```

编译并运行，运行结果如下图：



```

rdj@rdj-VirtualBox: ~/下载
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
rdj@rdj-VirtualBox:~/下载$ ./2-3
I am p2,with ID[2488],    my father is p1 with ID[2487].
I am p4,with ID[2489],    my father is p2 with ID[2488].
I am p5,with ID[2490],    my father is p2 with ID[2488].
I am p3,with ID[2491],    my father is p1 with ID[2487].
rdj@rdj-VirtualBox:~/下载$

```

根据各进程的父进程号可见，我创建的进程树符合题目要求。

fork 系统调用的作用是创建一个新的进程。fork 调用的一个特点就是它仅仅被调用一次，却能够返回两次，它可能有三种不同的返回值：

- 1) 在父进程中，fork 返回新创建子进程的进程 ID；
- 2) 在子进程中，fork 返回 0；
- 3) 如果出现错误，fork 返回一个负值；

fork()不会复制代码，父进程和子进程会共享代码。在 fork 函数执行完毕后，如果创建新进程成功，则出现两个进程，一个是子进程，一个是父进程。在子进程中，fork 函数返回 0，在父进程中，fork 返回新创建子进程的进程 ID。我们可以通过 fork 返回的值来判断当前进程是子进程还是父进程。

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。

根据题目要求，编写代码如下图：

```
2  #include <unistd.h>
3  #include <stdio.h>
4  #include <stdlib.h>
5  int main(void) {
6      int p2,p3,p4,p5;
7      while ((p2=fork())!=-1); //p1 create p2
8      if (!p2) {
9
10         while ((p4=fork())!=-1); //p2 create p4
11         if (!p4) {
12             while(1) {
13                 printf("I am p4,with ID[%d],    my father's ID[%d].\n",getpid(),getppid());
14                 sleep(1);
15             }
16         }
17
18         while ((p5=fork())!=-1); //p2 create p5
19         if (!p5) {
20             while(1) {
21                 printf("I am p5,with ID[%d],    my father's ID[%d].\n",getpid(),getppid());
22                 sleep(1);
23             }
24         }
25     }
26
27     for(int i = 0;i<5;i++){
```

```

        for(int i = 0;i<5;i++){
            printf("I am p2,with ID[%d],    my father's ID[%d].\n",getpid(),getppid());
            sleep(1);
        }
        exit(0);
    }
}

while ((p3=fork())!=-1); //p1 create p3
if(!p3){
    while(1) {
        printf("I am p3,with ID[%d],    my father's ID[%d].\n",getpid(),getppid());
        sleep(1);
    }
}

while(1) {
    printf("I am p1,with ID[%d],    my father's ID[%d].\n",getpid(),getppid());
    sleep(1);
}

return 0;
}

```

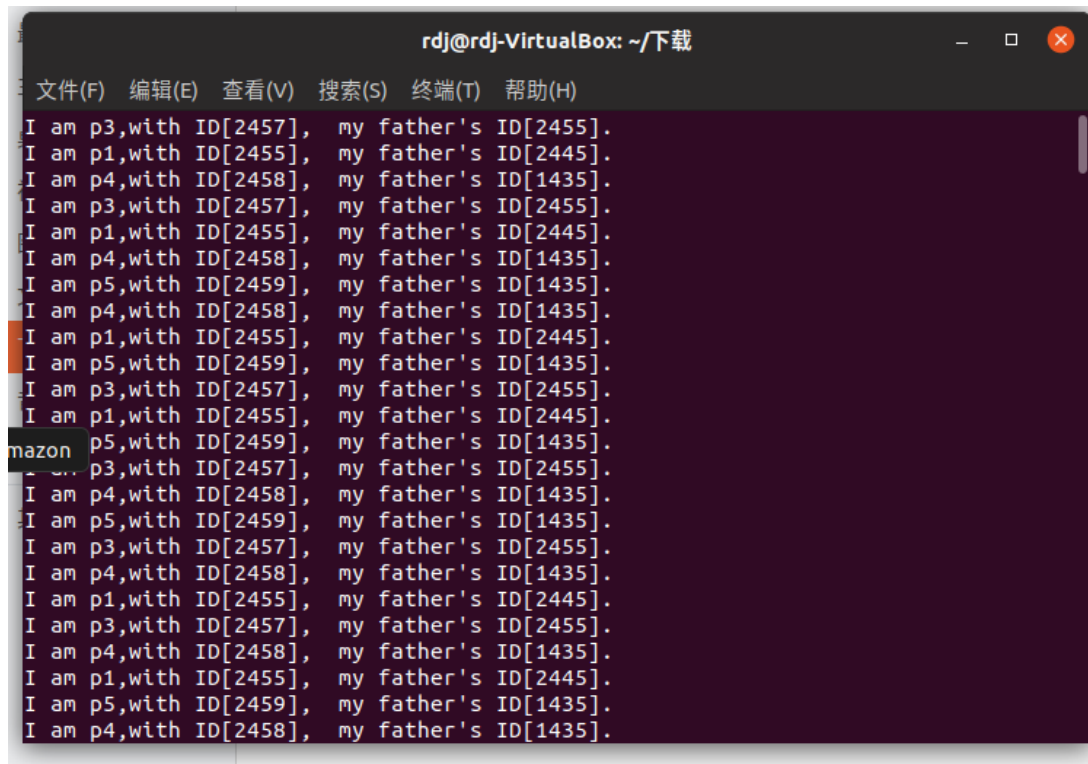
直接运行结果如下图：

```

rdj@rdj-VirtualBox: ~/下载
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
rdj@rdj-VirtualBox:~/下载$ ./2-4
I am p1,with ID[2455],    my father's ID[2445].
I am p3,with ID[2457],    my father's ID[2455].
I am p2,with ID[2456],    my father's ID[2455].
I am p5,with ID[2459],    my father's ID[2456].
I am p4,with ID[2458],    my father's ID[2456].
I am p1,with ID[2455],    my father's ID[2445].
I am p3,with ID[2457],    my father's ID[2455].
I am p2,with ID[2456],    my father's ID[2455].
I am p5,with ID[2459],    my father's ID[2456].
I am p4,with ID[2458],    my father's ID[2456].
I am p1,with ID[2455],    my father's ID[2445].
I am p3,with ID[2457],    my father's ID[2455].
I am p2,with ID[2456],    my father's ID[2455].
I am p5,with ID[2459],    my father's ID[2456].
I am p4,with ID[2458],    my father's ID[2456].
I am p1,with ID[2455],    my father's ID[2445].
I am p3,with ID[2457],    my father's ID[2455].
I am p2,with ID[2456],    my father's ID[2455].
I am p5,with ID[2459],    my father's ID[2456].
I am p4,with ID[2458],    my father's ID[2456].
I am p1,with ID[2455],    my father's ID[2445].
I am p3,with ID[2457],    my father's ID[2455].
I am p2,with ID[2456],    my father's ID[2455].

```

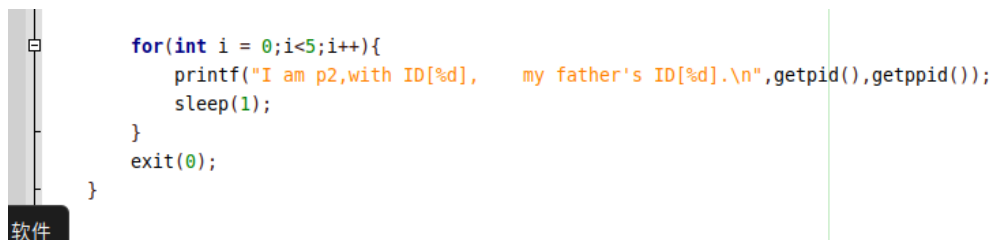
第一种方法：根据运行结果，p2 进程的进程号为 2456。打开一个新的终端，输入命令 kill -9 2456，可见运行结果变为如下图：



```
rdj@rdj-VirtualBox: ~/下载
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
I am p3,with ID[2457], my father's ID[2455].
I am p1,with ID[2455], my father's ID[2445].
I am p4,with ID[2458], my father's ID[1435].
I am p3,with ID[2457], my father's ID[2455].
I am p1,with ID[2455], my father's ID[2445].
I am p4,with ID[2458], my father's ID[1435].
I am p5,with ID[2459], my father's ID[1435].
I am p4,with ID[2458], my father's ID[1435].
I am p1,with ID[2455], my father's ID[2445].
I am p5,with ID[2459], my father's ID[1435].
I am p3,with ID[2457], my father's ID[2455].
I am p1,with ID[2455], my father's ID[2445].
I am p5,with ID[2459], my father's ID[1435].
I am p3,with ID[2457], my father's ID[2455].
I am p4,with ID[2458], my father's ID[1435].
I am p5,with ID[2459], my father's ID[1435].
I am p3,with ID[2457], my father's ID[2455].
I am p4,with ID[2458], my father's ID[1435].
I am p1,with ID[2455], my father's ID[2445].
I am p3,with ID[2457], my father's ID[2455].
I am p4,with ID[2458], my father's ID[1435].
I am p1,with ID[2455], my father's ID[2445].
I am p5,with ID[2459], my father's ID[1435].
I am p4,with ID[2458], my father's ID[1435].
```

可见 p2 进程已被“杀死”。

第二种方法：修改代码，将 p2 的循环次数改为 5 次，循环 5 次后则不再输出 p2.修改部分见下图：



```
for(int i = 0; i < 5; i++){
    printf("I am p2,with ID[%d], my father's ID[%d].\\n",getpid(),getppid());
    sleep(1);
}
exit(0);
```

前五次循环为：

```
#include <unistd.h>

rdj@rdj-VirtualBox: ~/下载
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
rdj@rdj-VirtualBox:~/下载$ ./2-4
I am p1,with ID[2934], my father's ID[2926].
I am p3,with ID[2936], my father's ID[2934].
I am p2,with ID[2935], my father's ID[2934].
I am p5,with ID[2938], my father's ID[2935].
I am p4,with ID[2937], my father's ID[2935].
I am p1,with ID[2934], my father's ID[2926].
I am p3,with ID[2936], my father's ID[2934].
I am p2,with ID[2935], my father's ID[2934].
I am p5,with ID[2938], my father's ID[2935].
I am p4,with ID[2937], my father's ID[2935].
I am p1,with ID[2934], my father's ID[2926].
I am p3,with ID[2936], my father's ID[2934].
I am p2,with ID[2935], my father's ID[2934].
I am p5,with ID[2938], my father's ID[2935].
I am p4,with ID[2937], my father's ID[2935].
I am p1,with ID[2934], my father's ID[2926].
I am p3,with ID[2936], my father's ID[2934].
I am p2,with ID[2935], my father's ID[2934].
```

5 次过后，p2 进程自动 exit 退出，退出后循环结果如下图：

```
#include <unistd.h>

rdj@rdj-VirtualBox: ~/下载
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
I am p1,with ID[2934], my father's ID[2926].
I am p3,with ID[2936], my father's ID[2934].
I am p5,with ID[2938], my father's ID[1435].
I am p4,with ID[2937], my father's ID[1435].
I am p1,with ID[2934], my father's ID[2926].
I am p3,with ID[2936], my father's ID[2934].
I am p5,with ID[2938], my father's ID[1435].
I am p4,with ID[2937], my father's ID[1435].
I am p1,with ID[2934], my father's ID[2926].
I am p3,with ID[2936], my father's ID[2934].
I am p5,with ID[2938], my father's ID[1435].
I am p4,with ID[2937], my father's ID[1435].
I am p1,with ID[2934], my father's ID[2926].
I am p3,with ID[2936], my father's ID[2934].
I am p5,with ID[2938], my father's ID[1435].
I am p4,with ID[2937], my father's ID[1435].
I am p1,with ID[2934], my father's ID[2926].
I am p3,with ID[2936], my father's ID[2934].
I am p5,with ID[2938], my father's ID[1435].
I am p4,with ID[2937], my father's ID[1435].
```

第三种方法：制造段错误，修改代码见如下截图：

