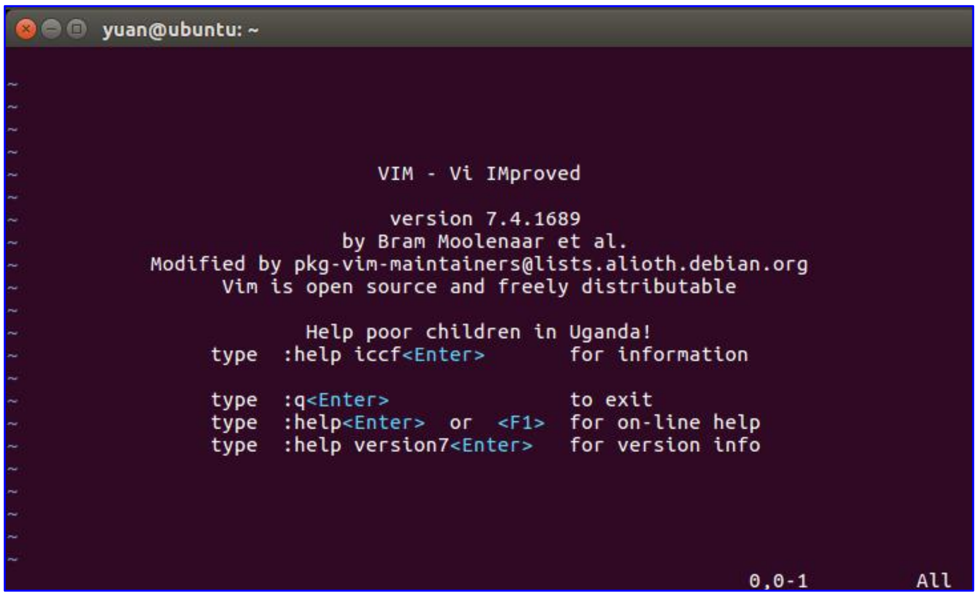
实验二：进程控制

袁少随　16281054　安全 1601

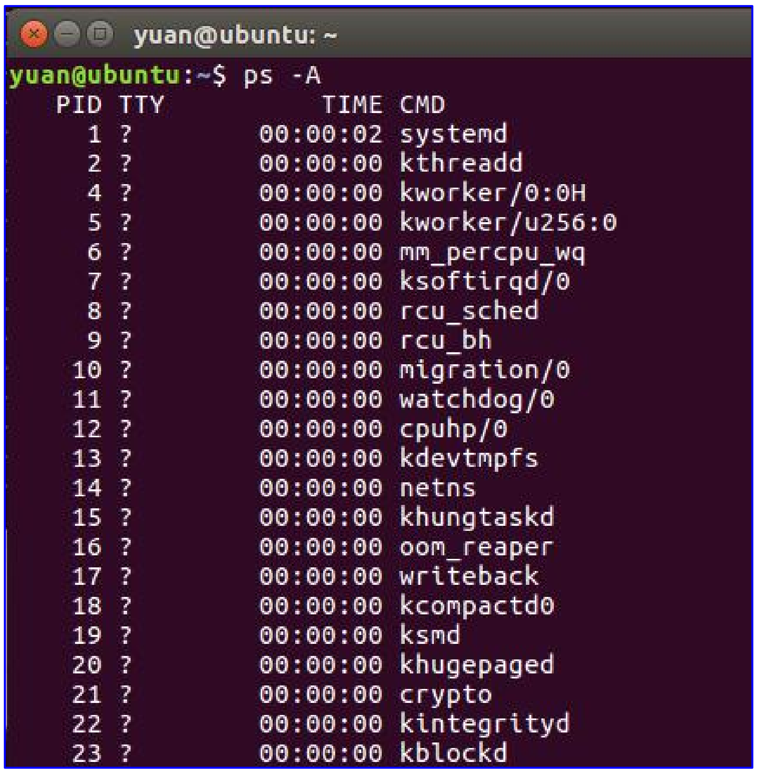
- Task 1:打开一个 vi 进程。通过 ps 命令以及选择合适的参数，只显示名字为 vi 的进程。寻找 vi 进程的父进程，直到 init 进程为止。记录过程中所有进程的 ID 和父进程 ID。将得到的进程树和由 pstree 命令的得到的进程树进行比较。

(1)打开一个终端，输入 vi,回车打开一个 vi 进程；

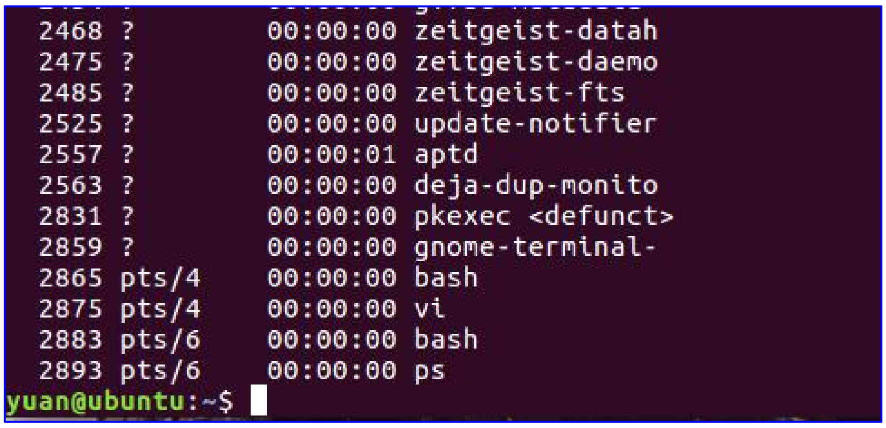


```
yuan@ubuntu: ~
~
~
~
~
~                          VIM - Vi IMproved
~
~                          version 7.4.1689
~                      by Bram Moolenaar et al.
~       Modified by pkg-vim-maintainers@lists.alioth.debian.org
~             Vim is open source and freely distributable
~
~                     Help poor children in Uganda!
~            type  :help iccf<Enter>       for information
~
~            type  :q<Enter>               to exit
~            type  :help<Enter>  or  <F1>  for on-line help
~            type  :help version7<Enter>   for version info
~
~
~
~
~
                                                    0,0-1         All
```

（2）打开另一个终端，通过 ps -A 显示所有进程；

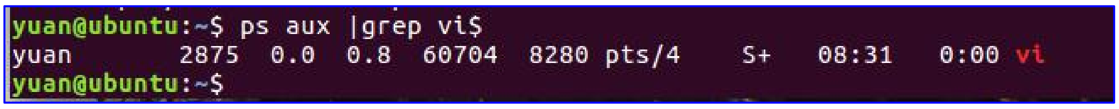


```
yuan@ubuntu: ~
yuan@ubuntu:~$ ps -A
   PID TTY          TIME CMD
     1 ?        00:00:02 systemd
     2 ?        00:00:00 kthreadd
     4 ?        00:00:00 kworker/0:0H
     5 ?        00:00:00 kworker/u256:0
     6 ?        00:00:00 mm_percpu_wq
     7 ?        00:00:00 ksoftirqd/0
     8 ?        00:00:00 rcu_sched
     9 ?        00:00:00 rcu_bh
    10 ?        00:00:00 migration/0
    11 ?        00:00:00 watchdog/0
    12 ?        00:00:00 cpuhp/0
    13 ?        00:00:00 kdevtmpfs
    14 ?        00:00:00 netns
    15 ?        00:00:00 khungtaskd
    16 ?        00:00:00 oom_reaper
    17 ?        00:00:00 writeback
    18 ?        00:00:00 kcompactd0
    19 ?        00:00:00 ksmd
    20 ?        00:00:00 khugepaged
    21 ?        00:00:00 crypto
    22 ?        00:00:00 kintegrityd
    23 ?        00:00:00 kblockd
```



```
 2468 ?        00:00:00 zeitgeist-datah
 2475 ?        00:00:00 zeitgeist-daemo
 2485 ?        00:00:00 zeitgeist-fts
 2525 ?        00:00:00 update-notifier
 2557 ?        00:00:01 aptd
 2563 ?        00:00:00 deja-dup-monito
 2831 ?        00:00:00 pkexec <defunct>
 2859 ?        00:00:00 gnome-terminal-
 2865 pts/4    00:00:00 bash
 2875 pts/4    00:00:00 vi
 2883 pts/6    00:00:00 bash
 2893 pts/6    00:00:00 ps
yuan@ubuntu:~$ 
```

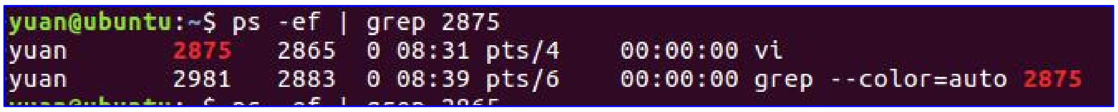
（3）通过 ps aux |grep vi 命令显示名字为 vi 的进程；



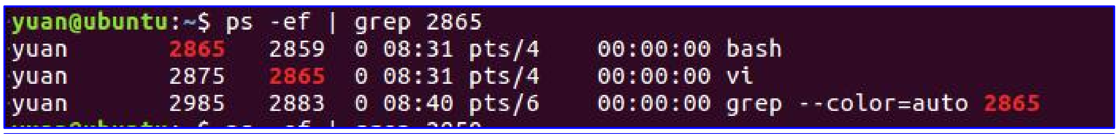
```
yuan@ubuntu:~$ ps aux |grep vi$
yuan        2875  0.0  0.8  60704  8280 pts/4    S+   08:31   0:00 vi
yuan@ubuntu:~$
```

由上图可知 vi 进程的 PID 为 2875；

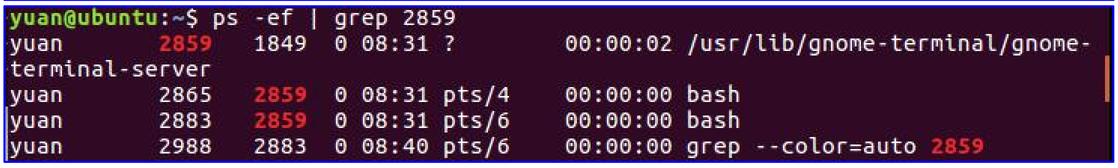
（4）通过命令 ps -ef | grep PID 寻找 vi 的父进程，直到 init 进程；



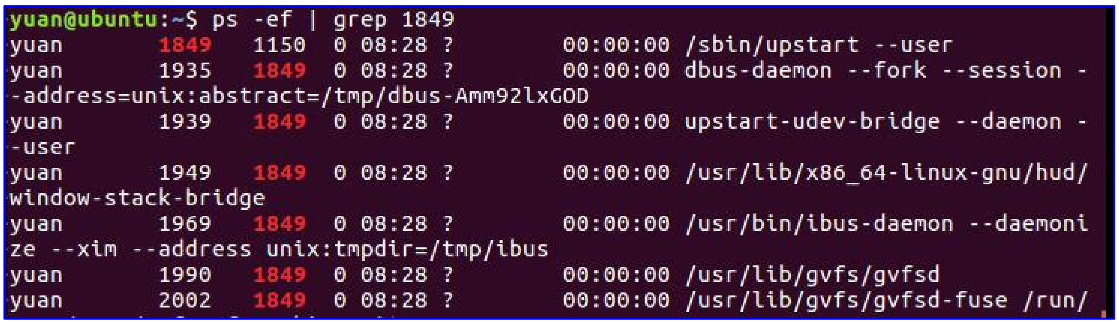
```
yuan@ubuntu:~$ ps -ef | grep 2875
yuan       2875   2865  0 08:31 pts/4    00:00:00 vi
yuan       2981   2883  0 08:39 pts/6    00:00:00 grep --color=auto 2875
```



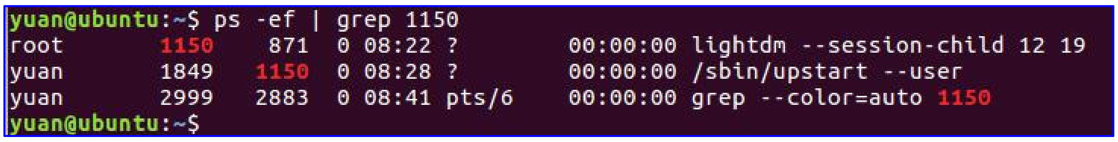
```
yuan@ubuntu:~$ ps -ef | grep 2865
yuan       2865   2859  0 08:31 pts/4    00:00:00 bash
yuan       2875   2865  0 08:31 pts/4    00:00:00 vi
yuan       2985   2883  0 08:40 pts/6    00:00:00 grep --color=auto 2865
```



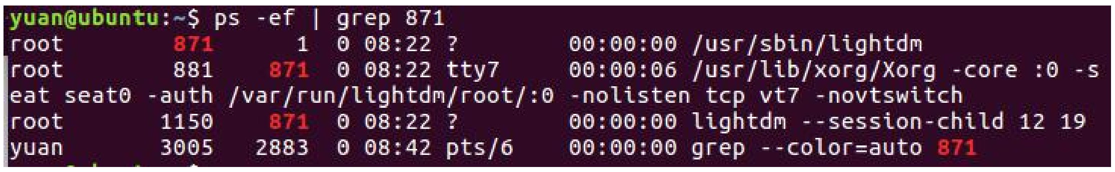
```
yuan@ubuntu:~$ ps -ef | grep 2859
yuan       2859   1849  0 08:31 ?        00:00:02 /usr/lib/gnome-terminal/gnome-
terminal-server
yuan       2865   2859  0 08:31 pts/4    00:00:00 bash
yuan       2883   2859  0 08:31 pts/6    00:00:00 bash
yuan       2988   2883  0 08:40 pts/6    00:00:00 grep --color=auto 2859
```



```
yuan@ubuntu:~$ ps -ef | grep 1849
yuan       1849   1150  0 08:28 ?        00:00:00 /sbin/upstart --user
yuan       1935   1849  0 08:28 ?        00:00:00 dbus-daemon --fork --session -
-address=unix:abstract=/tmp/dbus-Amm92lxGOD
yuan       1939   1849  0 08:28 ?        00:00:00 upstart-udev-bridge --daemon -
-user
yuan       1949   1849  0 08:28 ?        00:00:00 /usr/lib/x86_64-linux-gnu/hud/
window-stack-bridge
yuan       1969   1849  0 08:28 ?        00:00:00 /usr/bin/ibus-daemon --daemoni
ze --xim --address unix:tmpdir=/tmp/ibus
yuan       1990   1849  0 08:28 ?        00:00:00 /usr/lib/gvfs/gvfsd
yuan       2002   1849  0 08:28 ?        00:00:00 /usr/lib/gvfs/gvfsd-fuse /run/
```



```
yuan@ubuntu:~$ ps -ef | grep 1150
root       1150    871  0 08:22 ?        00:00:00 lightdm --session-child 12 19
yuan       1849   1150  0 08:28 ?        00:00:00 /sbin/upstart --user
yuan       2999   2883  0 08:41 pts/6    00:00:00 grep --color=auto 1150
yuan@ubuntu:~$
```



```
yuan@ubuntu:~$ ps -ef | grep 871
root        871      1  0 08:22 ?        00:00:00 /usr/sbin/lightdm
root        881    871  0 08:22 tty7     00:00:06 /usr/lib/xorg/Xorg -core :0 -s
eat seat0 -auth /var/run/lightdm/root/:0 -nolisten tcp vt7 -novtswitch
root       1150    871  0 08:22 ?        00:00:00 lightdm --session-child 12 19
yuan       3005   2883  0 08:42 pts/6    00:00:00 grep --color=auto 871
```

可知：

2875->2865->2859->1849->1150->871->1

ps -ef|grep 详解

ps 命令将某个进程显示出来;

grep 命令是查找;

中间的|是管道命令 是指 ps 命令与 grep 同时执行;

PS 是 LINUX 下最常用的也是非常强大的进程查看命令;

grep 命令是查找，是一种强大的文本搜索工具，它能使用正则表达式搜索文本，并把匹配的行打印出来。grep 全称是 Global Regular Expression Print，表示全局正则表达式版本，它的使用权限是所有用户。

以下这条命令是检查 java 进程是否存在：ps -ef |grep java

字段含义如下：

UID	PID	PPID	C	STIME	TTY	TIME	CMD
zzw	14124	13991	0	00:38	pts/0	00:00:00	grep --color=auto dae

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

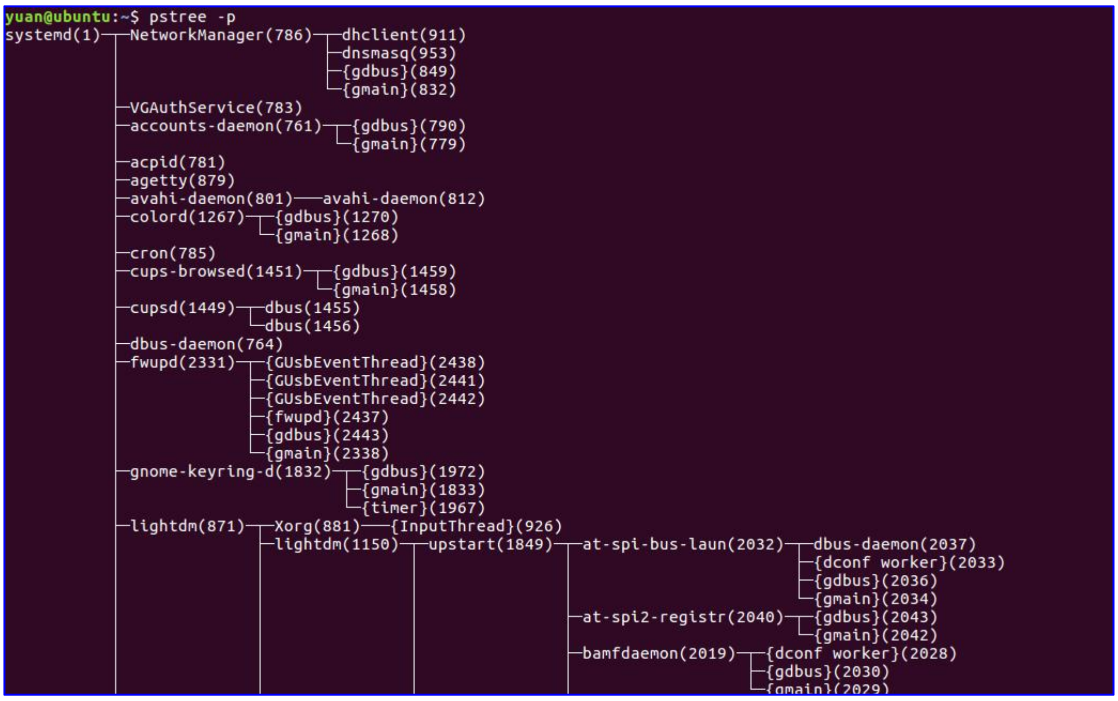
STIME：系统启动时间

TTY：登入者的终端机位置

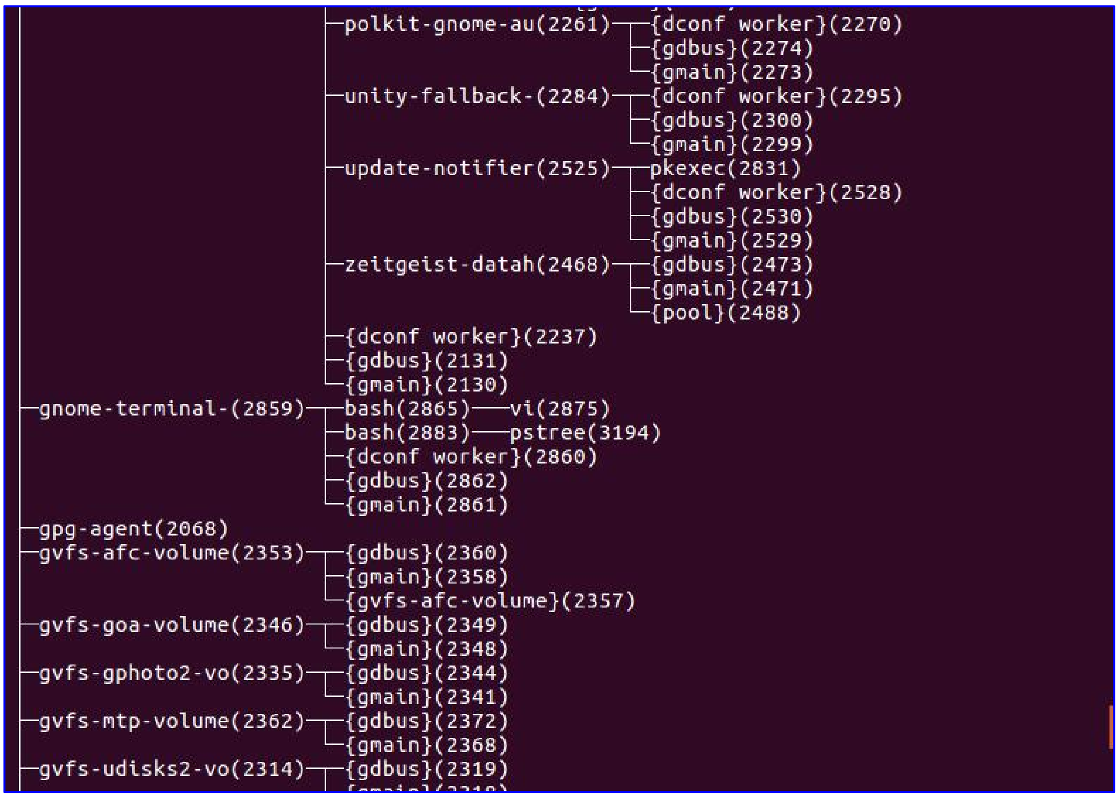
TIME：使用掉的 CPU 时间。

CMD：所下达的是什么指令

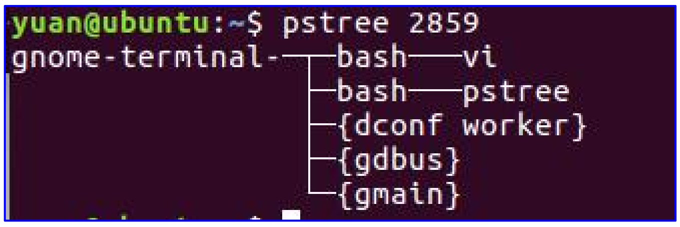
（5）通过命令 pstree PID 得到进程树；(pstree -p 为查询整个进程树）



```
yuan@ubuntu:~$ pstree -p
systemd(1)─┬─NetworkManager(786)─┬─dhclient(911)
           │                     ├─dnsmasq(953)
           │                     ├─{gdbus}(849)
           │                     └─{gmain}(832)
           ├─VGAuthService(783)
           ├─accounts-daemon(761)─┬─{gdbus}(790)
           │                      └─{gmain}(779)
           ├─acpid(781)
           ├─agetty(879)
           ├─avahi-daemon(801)───avahi-daemon(812)
           ├─colord(1267)─┬─{gdbus}(1270)
           │              └─{gmain}(1268)
           ├─cron(785)
           ├─cups-browsed(1451)─┬─{gdbus}(1459)
           │                    └─{gmain}(1458)
           ├─cupsd(1449)─┬─dbus(1455)
           │             └─dbus(1456)
           ├─dbus-daemon(764)
           ├─fwupd(2331)─┬─{GUsbEventThread}(2438)
           │             ├─{GUsbEventThread}(2441)
           │             ├─{GUsbEventThread}(2442)
           │             ├─{fwupd}(2437)
           │             ├─{gdbus}(2443)
           │             └─{gmain}(2338)
           ├─gnome-keyring-d(1832)─┬─{gdbus}(1972)
           │                       ├─{gmain}(1833)
           │                       └─{timer}(1967)
           ├─lightdm(871)─┬─Xorg(881)───{InputThread}(926)
           │              ├─lightdm(1150)─┬─upstart(1849)─┬─at-spi-bus-laun(2032)─┬─dbus-daemon(2037)
           │              │               │               │                       ├─{dconf worker}(2033)
           │              │               │               │                       ├─{gdbus}(2036)
           │              │               │               │                       └─{gmain}(2034)
           │              │               │               ├─at-spi2-registr(2040)─┬─{gdbus}(2043)
           │              │               │               │                       └─{gmain}(2042)
           │              │               │               ├─bamfdaemon(2019)─┬─{dconf worker}(2028)
           │              │               │               │                  ├─{gdbus}(2030)
           │              │               │               │                  └─{gmain}(2029)
```



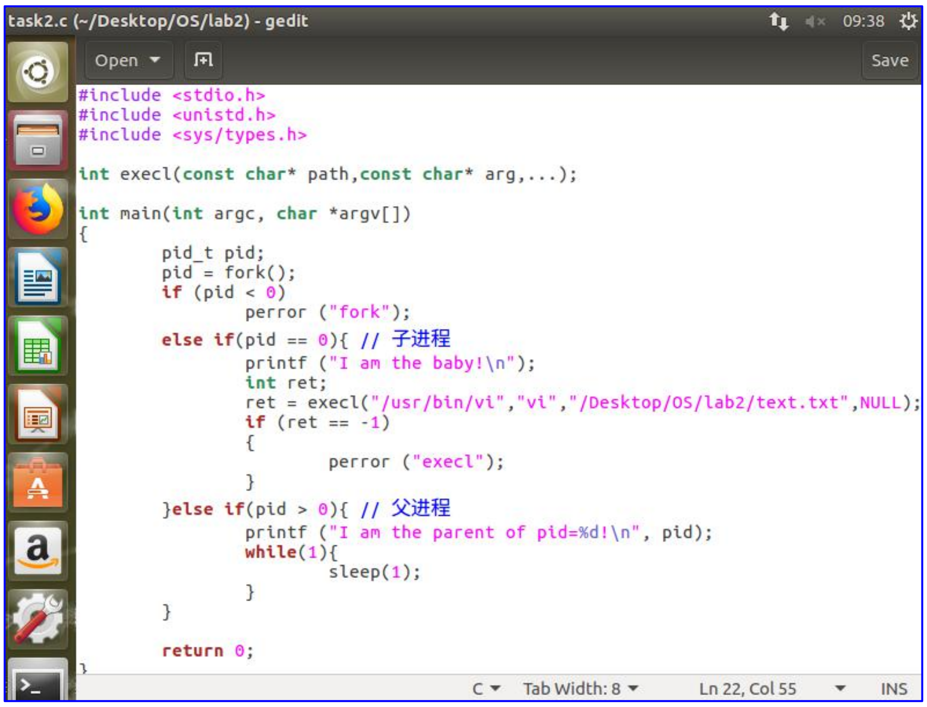
```
 │                                    ├─polkit-gnome-au(2261)─┬─{dconf worker}(2270)
 │                                    │                        ├─{gdbus}(2274)
 │                                    │                        └─{gmain}(2273)
 │                                    ├─unity-fallback-(2284)─┬─{dconf worker}(2295)
 │                                    │                        ├─{gdbus}(2300)
 │                                    │                        └─{gmain}(2299)
 │                                    ├─update-notifier(2525)─┬─pkexec(2831)
 │                                    │                        ├─{dconf worker}(2528)
 │                                    │                        ├─{gdbus}(2530)
 │                                    │                        └─{gmain}(2529)
 │                                    ├─zeitgeist-datah(2468)─┬─{gdbus}(2473)
 │                                    │                        ├─{gmain}(2471)
 │                                    │                        └─{pool}(2488)
 │                                    ├─{dconf worker}(2237)
 │                                    ├─{gdbus}(2131)
 │                                    └─{gmain}(2130)
 ├─gnome-terminal-(2859)─┬─bash(2865)───vi(2875)
 │                        ├─bash(2883)───pstree(3194)
 │                        ├─{dconf worker}(2860)
 │                        ├─{gdbus}(2862)
 │                        └─{gmain}(2861)
 ├─gpg-agent(2068)
 ├─gvfs-afc-volume(2353)─┬─{gdbus}(2360)
 │                        ├─{gmain}(2358)
 │                        └─{gvfs-afc-volume}(2357)
 ├─gvfs-goa-volume(2346)─┬─{gdbus}(2349)
 │                        └─{gmain}(2348)
 ├─gvfs-gphoto2-vo(2335)─┬─{gdbus}(2344)
 │                        └─{gmain}(2341)
 ├─gvfs-mtp-volume(2362)─┬─{gdbus}(2372)
 │                        └─{gmain}(2368)
 ├─gvfs-udisks2-vo(2314)─┬─{gdbus}(2319)
```



```
yuan@ubuntu:~$ pstree 2859
gnome-terminal-─┬─bash───vi
                ├─bash───pstree
                ├─{dconf worker}
                ├─{gdbus}
                └─{gmain}
```

对比发现两种方式所得结果一样；

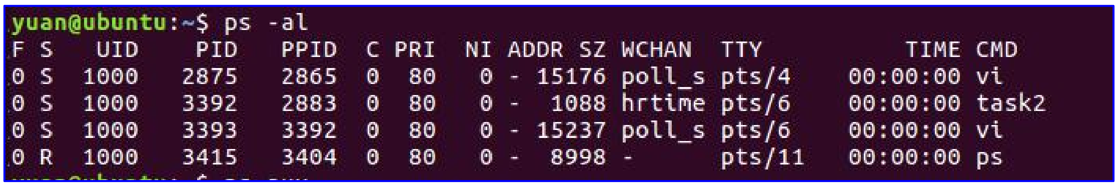
- 2、编写程序，首先使用 fork 系统调用，创建子进程。在父进程中继续执行空循环操作；在子进程中调用 exec 打开 vi 编辑器。然后在另外一个终端中，通过 ps －Al 命令、ps aux 或者 top 等命令，查看 vi 进程及其父进程的运行状态，理解每个参数所表达的意义。选择合适的命令参数，对所有进程按照 cpu 占用率排序。



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

int execl(const char* path,const char* arg,...);

int main(int argc, char *argv[])
{
        pid_t pid;
        pid = fork();
        if (pid < 0)
                perror ("fork");
        else if(pid == 0){ // 子进程
                printf ("I am the baby!\n");
                int ret;
                ret = execl("/usr/bin/vi","vi","/Desktop/OS/lab2/text.txt",NULL);
                if (ret == -1)
                {
                        perror ("execl");
                }
        }else if(pid > 0){ // 父进程
                printf ("I am the parent of pid=%d!\n", pid);
                while(1){
                        sleep(1);
                }
        }

        return 0;
}
```

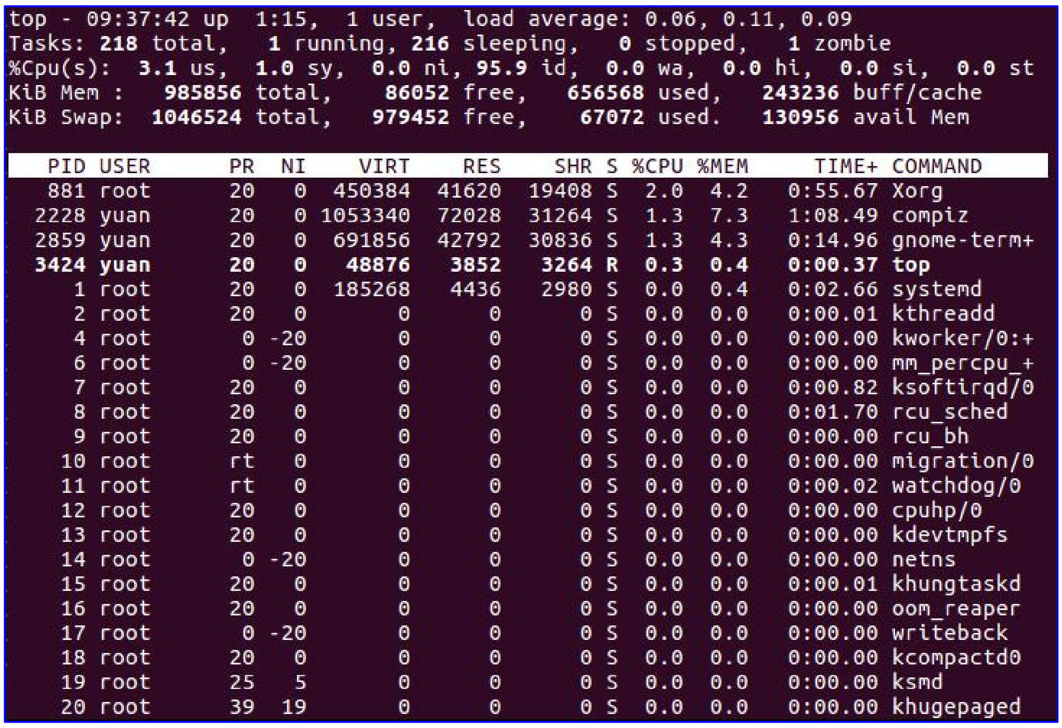
程序代码截图

ps －Al 命令查看 vi 进程及其父进程的运行状态：



```
yuan@ubuntu:~$ ps -al
F S   UID   PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
0 S  1000  2875  2865  0  80   0 - 15176 poll_s pts/4    00:00:00 vi
0 S  1000  3392  2883  0  80   0 -  1088 hrtime pts/6    00:00:00 task2
0 S  1000  3393  3392  0  80   0 - 15237 poll_s pts/6    00:00:00 vi
0 R  1000  3415  3404  0  80   0 -  8998 -      pts/11   00:00:00 ps
```

所有进程按照 cpu 占用率排序：



```
top - 09:37:42 up  1:15,  1 user,  load average: 0.06, 0.11, 0.09
Tasks: 218 total,   1 running, 216 sleeping,   0 stopped,   1 zombie
%Cpu(s):  3.1 us,  1.0 sy,  0.0 ni, 95.9 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem :   985856 total,    86052 free,   656568 used,   243236 buff/cache
KiB Swap:  1046524 total,   979452 free,    67072 used.   130956 avail Mem

   PID USER      PR  NI    VIRT    RES    SHR S %CPU %MEM     TIME+ COMMAND
   881 root      20   0  450384  41620  19408 S  2.0  4.2   0:55.67 Xorg
  2228 yuan      20   0 1053340  72028  31264 S  1.3  7.3   1:08.49 compiz
  2859 yuan      20   0  691856  42792  30836 S  1.3  4.3   0:14.96 gnome-term+
  3424 yuan      20   0   48876   3852   3264 R  0.3  0.4   0:00.37 top
     1 root      20   0  185268   4436   2980 S  0.0  0.4   0:02.66 systemd
     2 root      20   0       0      0      0 S  0.0  0.0   0:00.01 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   0:00.82 ksoftirqd/0
     8 root      20   0       0      0      0 S  0.0  0.0   0:01.70 rcu_sched
     9 root      20   0       0      0      0 S  0.0  0.0   0:00.00 rcu_bh
    10 root      rt   0       0      0      0 S  0.0  0.0   0:00.00 migration/0
    11 root      rt   0       0      0      0 S  0.0  0.0   0:00.02 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 kdevtmpfs
    14 root       0 -20       0      0      0 S  0.0  0.0   0:00.00 netns
    15 root      20   0       0      0      0 S  0.0  0.0   0:00.01 khungtaskd
    16 root      20   0       0      0      0 S  0.0  0.0   0:00.00 oom_reaper
    17 root       0 -20       0      0      0 S  0.0  0.0   0:00.00 writeback
    18 root      20   0       0      0      0 S  0.0  0.0   0:00.00 kcompactd0
    19 root      25   5       0      0      0 S  0.0  0.0   0:00.00 ksmd
    20 root      39  19       0      0      0 S  0.0  0.0   0:00.00 khugepaged
```

参数命令解析：

PID 进程 id

PPID 父进程 id

RUSER Real user name

UID 进程所有者的用户 id

USER 进程所有者的用户名

GROUP 进程所有者的组名

TTY 启动进程的终端名。不是从终端启动的进程则显示为 ?

PR 优先级

NI nice 值。负值表示高优先级，正值表示低优先级

P 最后使用的 CPU，仅在多 CPU 环境下有意义

%CPU 上次更新到现在的 CPU 时间占用百分比

TIME 进程使用的 CPU 时间总计，单位秒

TIME+ 进程使用的 CPU 时间总计，单位 1/100 秒

%MEM 进程使用的物理内存百分比

VIRT 进程使用的虚拟内存总量，单位 kb。VIRT=SWAP+RES

SWAP 进程使用的虚拟内存中，被换出的大小，单位 kb。

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

CODE 可执行代码占用的物理内存大小，单位 kb

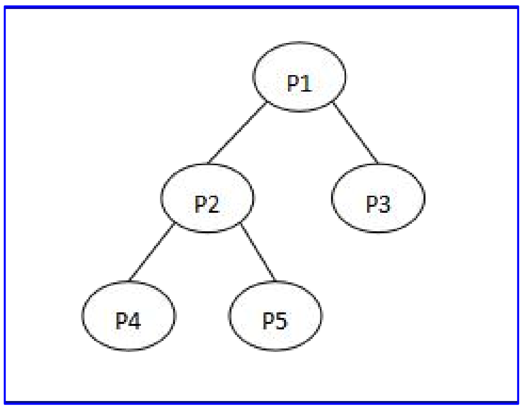
DATA 可执行代码以外的部分(数据段+栈)占用的物理内存大小，单位 kb

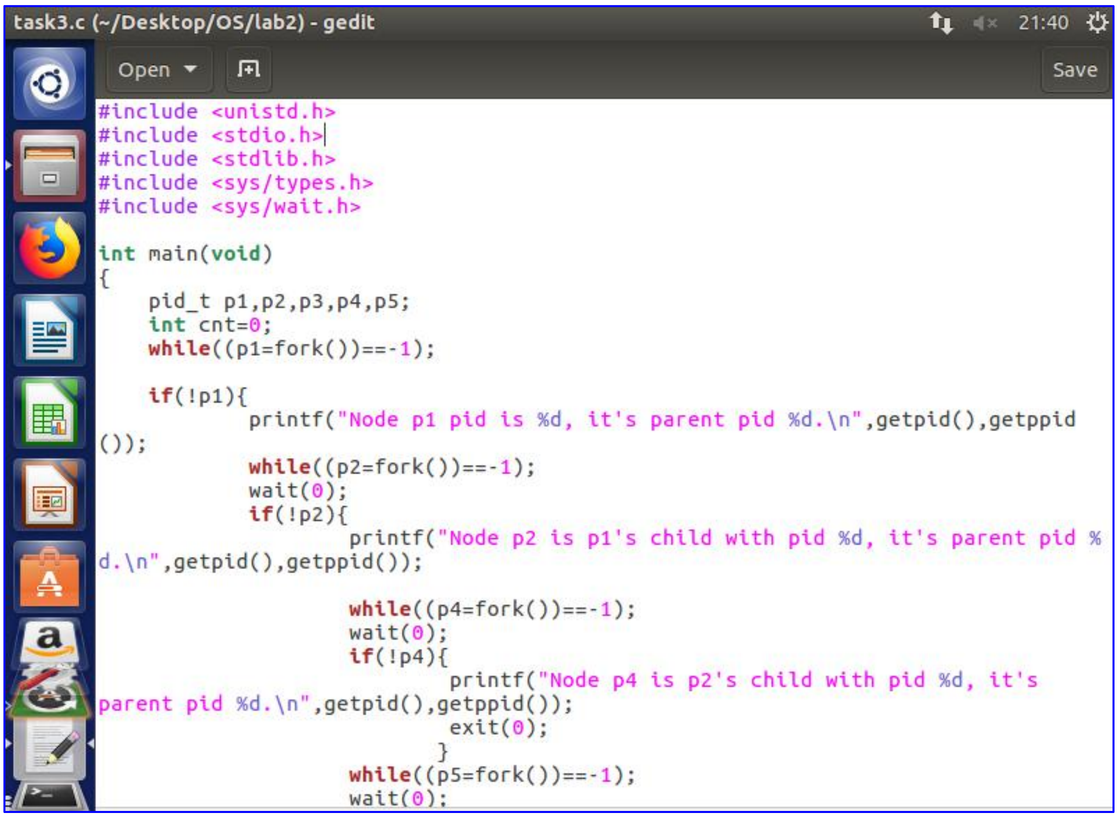
SHR 共享内存大小，单位 kb

nFLT 页面错误次数

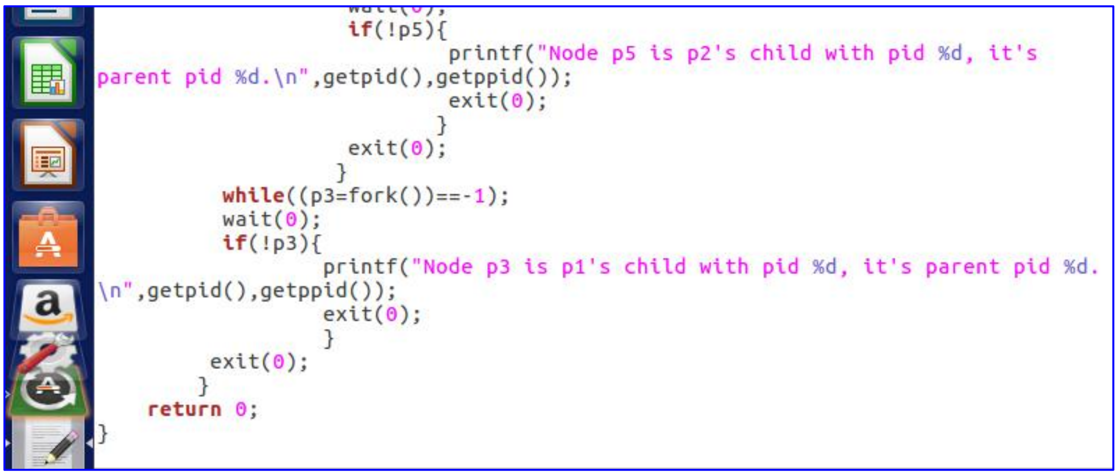
nDRT 最后一次写入到现在，被修改过的页面数。

- **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。**



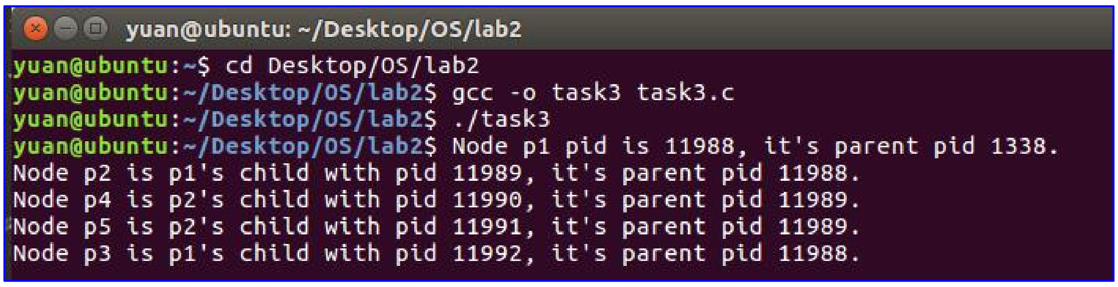


```c
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/wait.h>

int main(void)
{
    pid_t p1,p2,p3,p4,p5;
    int cnt=0;
    while((p1=fork())==-1);

    if(!p1){
            printf("Node p1 pid is %d, it's parent pid %d.\n",getpid(),getppid
());
            while((p2=fork())==-1);
            wait(0);
            if(!p2){
                    printf("Node p2 is p1's child with pid %d, it's parent pid %
d.\n",getpid(),getppid());

                    while((p4=fork())==-1);
                    wait(0);
                    if(!p4){
                            printf("Node p4 is p2's child with pid %d, it's
parent pid %d.\n",getpid(),getppid());
                            exit(0);
                    }
                    while((p5=fork())==-1);
                    wait(0);
```



```c
                    if(!p5){
                            printf("Node p5 is p2's child with pid %d, it's
parent pid %d.\n",getpid(),getppid());
                            exit(0);
                    }
                    exit(0);
            }
          while((p3=fork())==-1);
          wait(0);
          if(!p3){
                  printf("Node p3 is p1's child with pid %d, it's parent pid %d.
\n",getpid(),getppid());
                  exit(0);
          }
         exit(0);
        }
    return 0;
}
```

程序代码截图

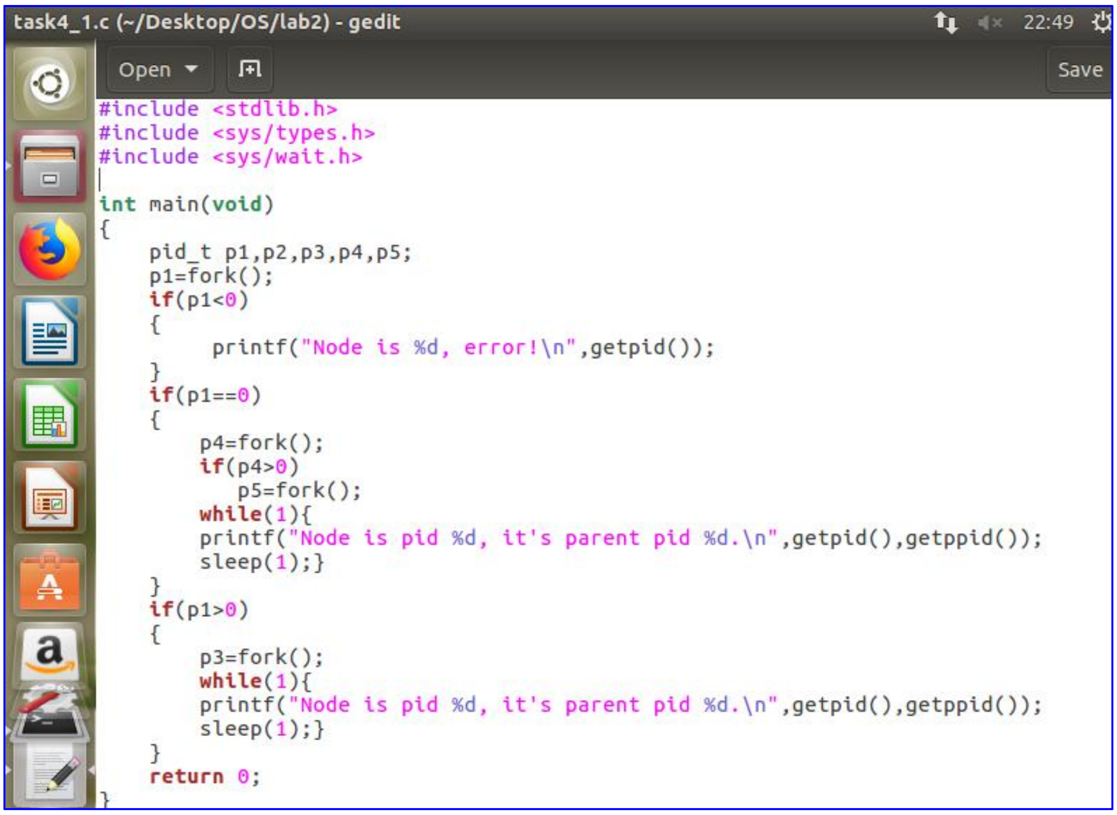


```
yuan@ubuntu:~$ cd Desktop/OS/lab2
yuan@ubuntu:~/Desktop/OS/lab2$ gcc -o task3 task3.c
yuan@ubuntu:~/Desktop/OS/lab2$ ./task3
yuan@ubuntu:~/Desktop/OS/lab2$ Node p1 pid is 11988, it's parent pid 1338.
Node p2 is p1's child with pid 11989, it's parent pid 11988.
Node p4 is p2's child with pid 11990, it's parent pid 11989.
Node p5 is p2's child with pid 11991, it's parent pid 11989.
Node p3 is p1's child with pid 11992, it's parent pid 11988.
```

程序输出结果

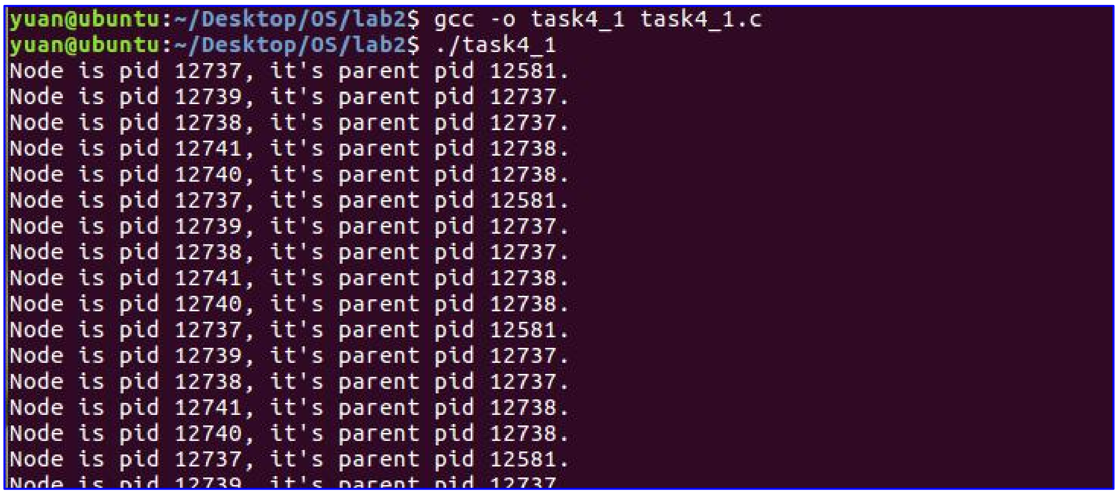
执行顺序为：P1->P2->P4->P5->P3

- 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。



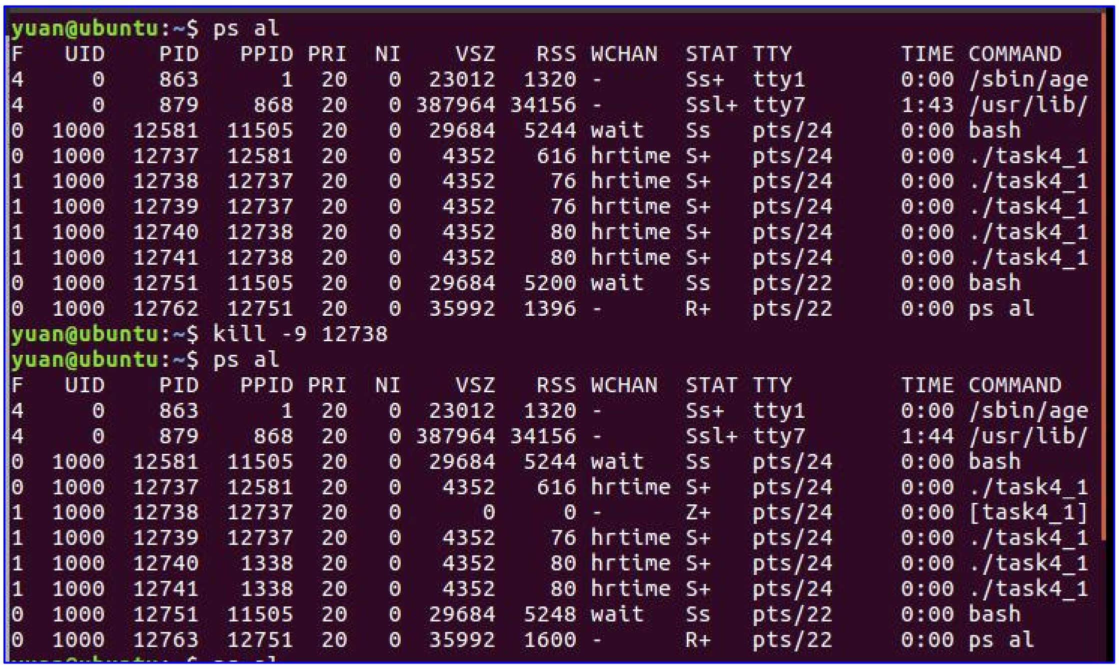
```c
#include <stdlib.h>
#include <sys/types.h>
#include <sys/wait.h>

int main(void)
{
    pid_t p1,p2,p3,p4,p5;
    p1=fork();
    if(p1<0)
    {
         printf("Node is %d, error!\n",getpid());
    }
    if(p1==0)
    {
        p4=fork();
        if(p4>0)
           p5=fork();
        while(1){
        printf("Node is pid %d, it's parent pid %d.\n",getpid(),getppid());
        sleep(1);}
    }
    if(p1>0)
    {
        p3=fork();
        while(1){
        printf("Node is pid %d, it's parent pid %d.\n",getpid(),getppid());
        sleep(1);}
    }
    return 0;
}
```

程序代码截图



```
yuan@ubuntu:~/Desktop/OS/lab2$ gcc -o task4_1 task4_1.c
yuan@ubuntu:~/Desktop/OS/lab2$ ./task4_1
Node is pid 12737, it's parent pid 12581.
Node is pid 12739, it's parent pid 12737.
Node is pid 12738, it's parent pid 12737.
Node is pid 12741, it's parent pid 12738.
Node is pid 12740, it's parent pid 12738.
Node is pid 12737, it's parent pid 12581.
Node is pid 12739, it's parent pid 12737.
Node is pid 12738, it's parent pid 12737.
Node is pid 12741, it's parent pid 12738.
Node is pid 12740, it's parent pid 12738.
Node is pid 12737, it's parent pid 12581.
Node is pid 12739, it's parent pid 12737.
Node is pid 12738, it's parent pid 12737.
Node is pid 12741, it's parent pid 12738.
Node is pid 12740, it's parent pid 12738.
Node is pid 12737, it's parent pid 12581.
Node is pid 12739, it's parent pid 12737
```

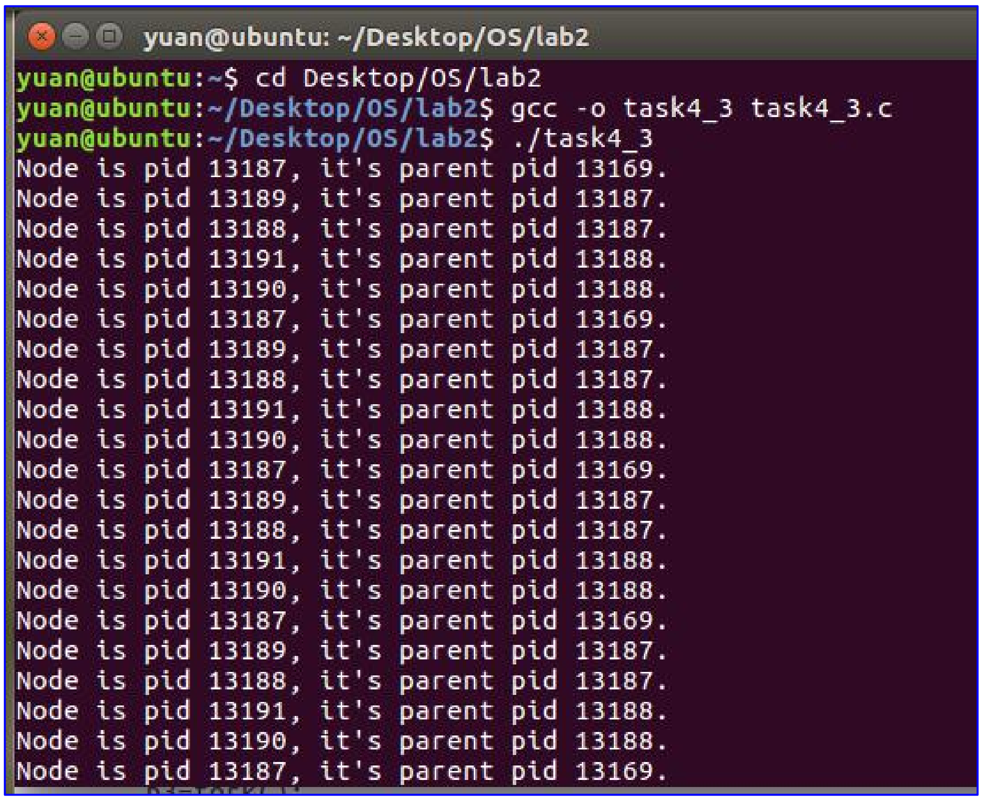
程序输出结果



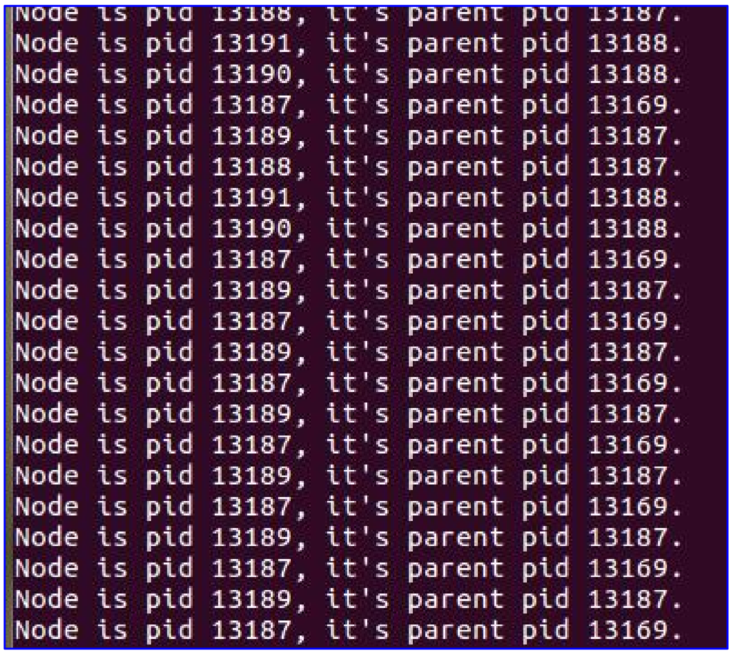
```
yuan@ubuntu:~$ ps al
F   UID     PID    PPID PRI  NI    VSZ   RSS WCHAN  STAT TTY        TIME COMMAND
4     0     863       1  20   0  23012  1320 -      Ss+  tty1       0:00 /sbin/age
4     0     879     868  20   0 387964 34156 -      Ssl+ tty7       1:43 /usr/lib/
0  1000   12581   11505  20   0  29684  5244 wait   Ss   pts/24     0:00 bash
0  1000   12737   12581  20   0   4352   616 hrtime S+   pts/24     0:00 ./task4_1
1  1000   12738   12737  20   0   4352    76 hrtime S+   pts/24     0:00 ./task4_1
1  1000   12739   12737  20   0   4352    76 hrtime S+   pts/24     0:00 ./task4_1
1  1000   12740   12738  20   0   4352    80 hrtime S+   pts/24     0:00 ./task4_1
1  1000   12741   12738  20   0   4352    80 hrtime S+   pts/24     0:00 ./task4_1
0  1000   12751   11505  20   0  29684  5200 wait   Ss   pts/22     0:00 bash
0  1000   12762   12751  20   0  35992  1396 -      R+   pts/22     0:00 ps al
yuan@ubuntu:~$ kill -9 12738
yuan@ubuntu:~$ ps al
F   UID     PID    PPID PRI  NI    VSZ   RSS WCHAN  STAT TTY        TIME COMMAND
4     0     863       1  20   0  23012  1320 -      Ss+  tty1       0:00 /sbin/age
4     0     879     868  20   0 387964 34156 -      Ssl+ tty7       1:44 /usr/lib/
0  1000   12581   11505  20   0  29684  5244 wait   Ss   pts/24     0:00 bash
0  1000   12737   12581  20   0   4352   616 hrtime S+   pts/24     0:00 ./task4_1
1  1000   12738   12737  20   0      0     0 -      Z+   pts/24     0:00 [task4_1]
1  1000   12739   12737  20   0   4352    76 hrtime S+   pts/24     0:00 ./task4_1
1  1000   12740    1338  20   0   4352    80 hrtime S+   pts/24     0:00 ./task4_1
1  1000   12741    1338  20   0   4352    80 hrtime S+   pts/24     0:00 ./task4_1
0  1000   12751   11505  20   0  29684  5248 wait   Ss   pts/22     0:00 bash
0  1000   12763   12751  20   0  35992  1600 -      R+   pts/22     0:00 ps al
```

kill -9 终止 P2 进程结果

P2 进程被杀死后，P4、P5 进程还在，不过父进程改变了。P2 进程的 STAT 状态由 S+变为 Z+;

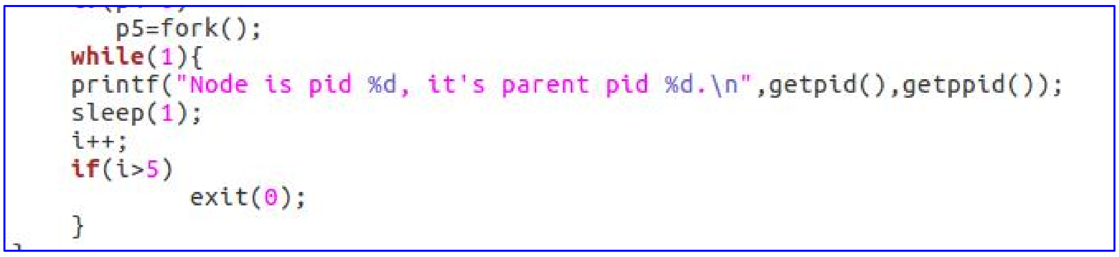


```
yuan@ubuntu: ~/Desktop/OS/lab2
yuan@ubuntu:~$ cd Desktop/OS/lab2
yuan@ubuntu:~/Desktop/OS/lab2$ gcc -o task4_3 task4_3.c
yuan@ubuntu:~/Desktop/OS/lab2$ ./task4_3
Node is pid 13187, it's parent pid 13169.
Node is pid 13189, it's parent pid 13187.
Node is pid 13188, it's parent pid 13187.
Node is pid 13191, it's parent pid 13188.
Node is pid 13190, it's parent pid 13188.
Node is pid 13187, it's parent pid 13169.
Node is pid 13189, it's parent pid 13187.
Node is pid 13188, it's parent pid 13187.
Node is pid 13191, it's parent pid 13188.
Node is pid 13190, it's parent pid 13188.
Node is pid 13187, it's parent pid 13169.
Node is pid 13189, it's parent pid 13187.
Node is pid 13188, it's parent pid 13187.
Node is pid 13191, it's parent pid 13188.
Node is pid 13190, it's parent pid 13188.
Node is pid 13187, it's parent pid 13169.
Node is pid 13189, it's parent pid 13187.
Node is pid 13188, it's parent pid 13187.
Node is pid 13191, it's parent pid 13188.
Node is pid 13190, it's parent pid 13188.
Node is pid 13187, it's parent pid 13169.
```



```
Node is pid 13188, it's parent pid 13187.
Node is pid 13191, it's parent pid 13188.
Node is pid 13190, it's parent pid 13188.
Node is pid 13187, it's parent pid 13169.
Node is pid 13189, it's parent pid 13187.
Node is pid 13188, it's parent pid 13187.
Node is pid 13191, it's parent pid 13188.
Node is pid 13190, it's parent pid 13188.
Node is pid 13187, it's parent pid 13169.
Node is pid 13189, it's parent pid 13187.
Node is pid 13187, it's parent pid 13169.
Node is pid 13189, it's parent pid 13187.
Node is pid 13187, it's parent pid 13169.
Node is pid 13189, it's parent pid 13187.
Node is pid 13187, it's parent pid 13169.
Node is pid 13189, it's parent pid 13187.
Node is pid 13187, it's parent pid 13169.
Node is pid 13189, it's parent pid 13187.
Node is pid 13187, it's parent pid 13169.
Node is pid 13189, it's parent pid 13187.
Node is pid 13187, it's parent pid 13169.
```

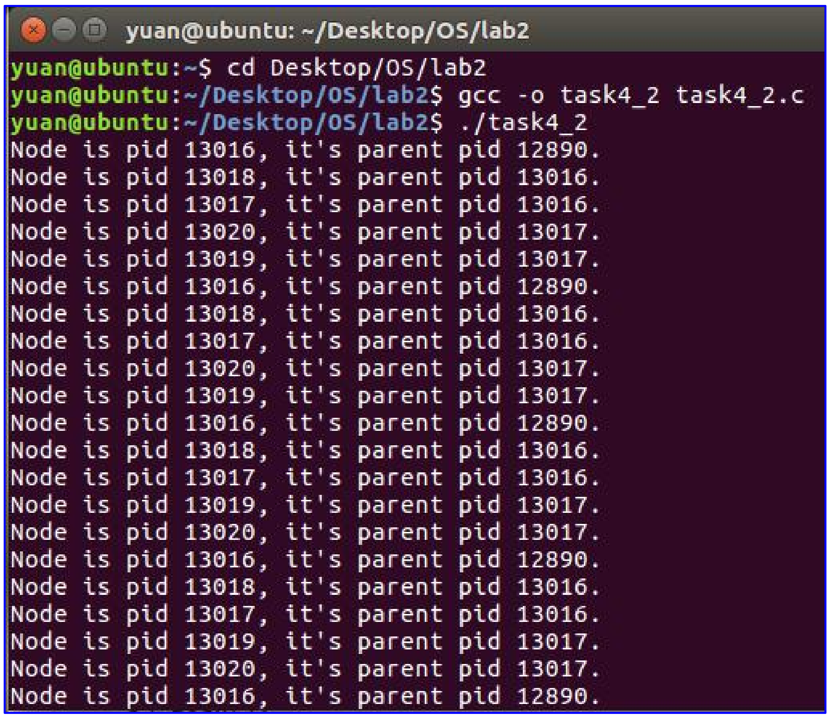
自己正常退出 exit()程序结果



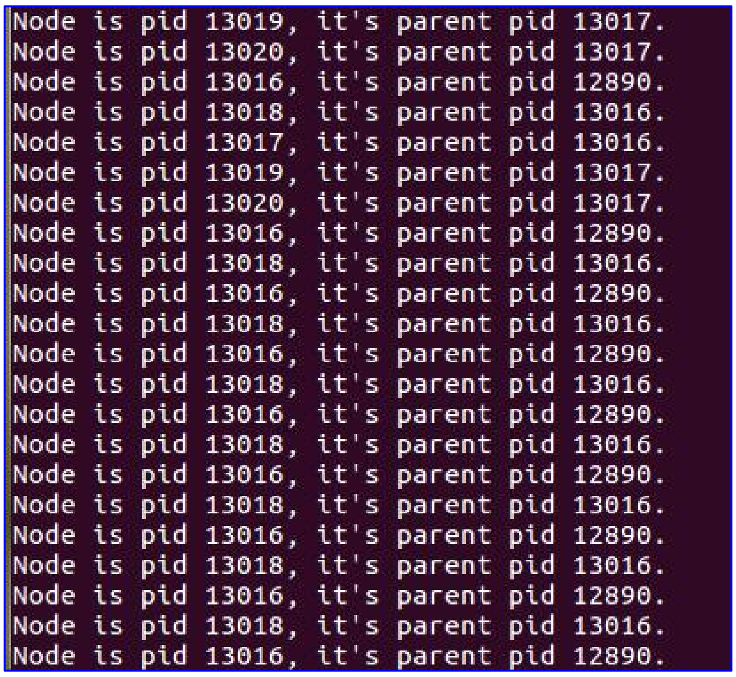
```
      p5=fork();
    while(1){
    printf("Node is pid %d, it's parent pid %d.\n",getpid(),getppid());
    sleep(1);
    i++;
    if(i>5)
            exit(0);
    }
```

自己正常退出 exit()部分代码更改

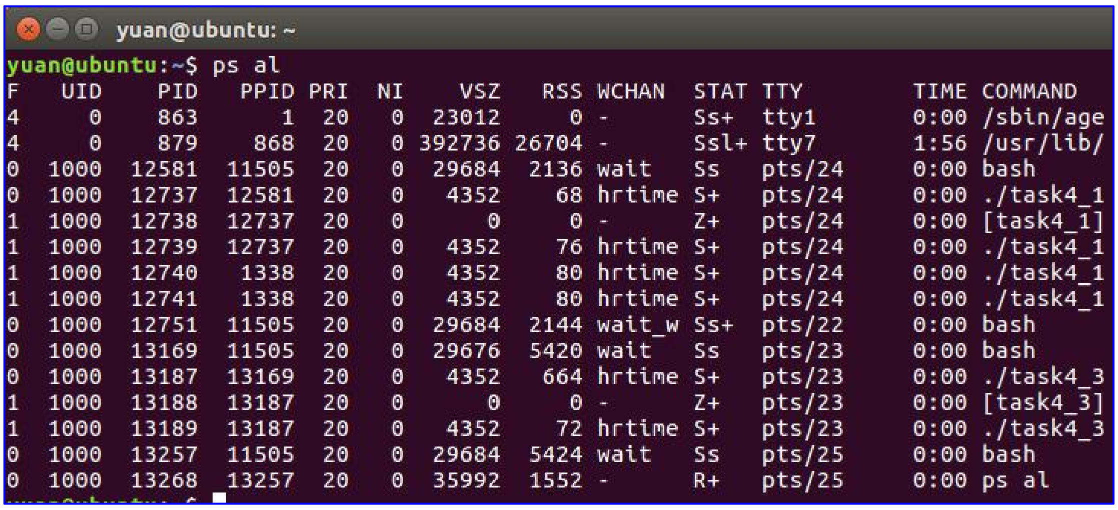
自己正常退出 exit():利用 i 变量设置循环 5 次后执行 exit(0),退出。



```
yuan@ubuntu: ~/Desktop/OS/lab2
yuan@ubuntu:~$ cd Desktop/OS/lab2
yuan@ubuntu:~/Desktop/OS/lab2$ gcc -o task4_2 task4_2.c
yuan@ubuntu:~/Desktop/OS/lab2$ ./task4_2
Node is pid 13016, it's parent pid 12890.
Node is pid 13018, it's parent pid 13016.
Node is pid 13017, it's parent pid 13016.
Node is pid 13020, it's parent pid 13017.
Node is pid 13019, it's parent pid 13017.
Node is pid 13016, it's parent pid 12890.
Node is pid 13018, it's parent pid 13016.
Node is pid 13017, it's parent pid 13016.
Node is pid 13020, it's parent pid 13017.
Node is pid 13019, it's parent pid 13017.
Node is pid 13016, it's parent pid 12890.
Node is pid 13018, it's parent pid 13016.
Node is pid 13017, it's parent pid 13016.
Node is pid 13019, it's parent pid 13017.
Node is pid 13020, it's parent pid 13017.
Node is pid 13016, it's parent pid 12890.
Node is pid 13018, it's parent pid 13016.
Node is pid 13017, it's parent pid 13016.
Node is pid 13019, it's parent pid 13017.
Node is pid 13020, it's parent pid 13017.
Node is pid 13016, it's parent pid 12890.
```

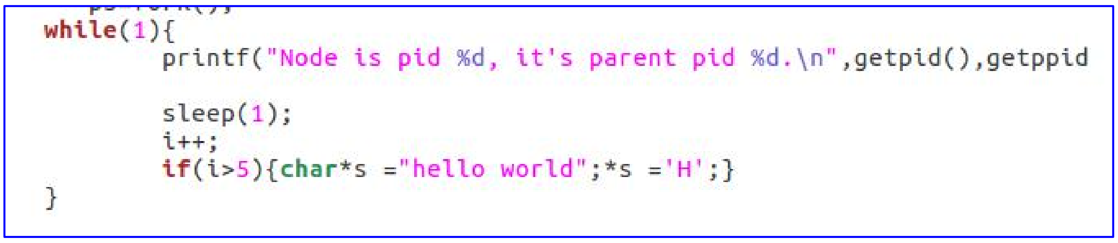


```
Node is pid 13019, it's parent pid 13017.
Node is pid 13020, it's parent pid 13017.
Node is pid 13016, it's parent pid 12890.
Node is pid 13018, it's parent pid 13016.
Node is pid 13017, it's parent pid 13016.
Node is pid 13019, it's parent pid 13017.
Node is pid 13020, it's parent pid 13017.
Node is pid 13016, it's parent pid 12890.
Node is pid 13018, it's parent pid 13016.
Node is pid 13016, it's parent pid 12890.
Node is pid 13018, it's parent pid 13016.
Node is pid 13016, it's parent pid 12890.
Node is pid 13018, it's parent pid 13016.
Node is pid 13016, it's parent pid 12890.
Node is pid 13018, it's parent pid 13016.
Node is pid 13016, it's parent pid 12890.
Node is pid 13018, it's parent pid 13016.
Node is pid 13016, it's parent pid 12890.
Node is pid 13018, it's parent pid 13016.
Node is pid 13016, it's parent pid 12890.
Node is pid 13018, it's parent pid 13016.
Node is pid 13016, it's parent pid 12890.
```



```
yuan@ubuntu: ~
yuan@ubuntu:~$ ps al
F   UID   PID  PPID PRI  NI    VSZ   RSS WCHAN  STAT TTY        TIME COMMAND
4     0   863     1  20   0  23012     0 -      Ss+  tty1       0:00 /sbin/age
4     0   879   868  20   0 392736 26704 -      Ssl+ tty7       1:56 /usr/lib/
0  1000 12581 11505  20   0  29684  2136 wait   Ss   pts/24     0:00 bash
0  1000 12737 12581  20   0   4352    68 hrtime S+   pts/24     0:00 ./task4_1
1  1000 12738 12737  20   0      0     0 -      Z+   pts/24     0:00 [task4_1]
1  1000 12739 12737  20   0   4352    76 hrtime S+   pts/24     0:00 ./task4_1
1  1000 12740  1338  20   0   4352    80 hrtime S+   pts/24     0:00 ./task4_1
1  1000 12741  1338  20   0   4352    80 hrtime S+   pts/24     0:00 ./task4_1
0  1000 12751 11505  20   0  29684  2144 wait_w Ss+  pts/22     0:00 bash
0  1000 13169 11505  20   0  29676  5420 wait   Ss   pts/23     0:00 bash
0  1000 13187 13169  20   0   4352   664 hrtime S+   pts/23     0:00 ./task4_3
1  1000 13188 13187  20   0      0     0 -      Z+   pts/23     0:00 [task4_3]
1  1000 13189 13187  20   0   4352    72 hrtime S+   pts/23     0:00 ./task4_3
0  1000 13257 11505  20   0  29684  5424 wait   Ss   pts/25     0:00 bash
0  1000 13268 13257  20   0  35992  1552 -      R+   pts/25     0:00 ps al
```

段错误退出程序结果



```
    while(1){
            printf("Node is pid %d, it's parent pid %d.\n",getpid(),getppid

            sleep(1);
            i++;
            if(i>5){char*s ="hello world";*s ='H';}
    }
```

段错误退出部分代码更改

段错误注解：

段错误就是指访问的内存超出了系统所给这个程序的内存空间，通常这个值是由 gd tr 来保存的，他是一个 48 位的寄存器，其中的 32 位是保存由它指向的 gdt 表，后 13 位保存相应于 gdt 的下标，最后 3 位包括了程序是否在内存中以及程序的在 cpu 中的运行级别，指向 的 gdt 是由以 64 位为一个单位的表，在这张表中就保存着程序运行的代码段以及数据段的起 始地址以及与此相应的段限和页面交换还有程序运行级别还有内存粒度等等的信息。

常用段错误：

1，
```
int main(void){
char*s ="hello world";
*s ='H';
}
```
被装载时，系统把“hello world” 连同其它的字符串和 const 型数据放入到内存的只读区。执行时，一个变量 s 被设为指向该字符串的位置，当再试图向该位置写时，就会产生段错误。

2，
```
int*ptr = NULL;
*ptr =1;
```
因为该代码只创建了一个空指针，并没有指向一个具体空间，当赋值时，产生段错误。

3，
```
int main(void){
main();
return0;
}
```
无限递归，这会导致栈溢出，也会产生段错误。