实验二 进程控制

1. 实验目的：

- 加深对进程概念的理解，明确进程和程序的区别。
- 掌握 Linux 系统中的进程创建，管理和删除等操作。
- 熟悉使用 Linux 下的命令和工具，如 man, find, grep, whereis, ps, pgrep, kill, ptree, top, vim, gcc，gdb，管道|等。

2. 基础知识：

- **进程的创建**

  Linux 中，载入内存并执行程序映像的操作与创建一个新进程的操作是分离的。将程序映像载入内存，并开始运行它，这个过程称为运行一个新的程序，相应的系统调用称为 exec 系统调用。而创建一个新的进程的系统调用是 fork 系统调用。

- **exec 系统调用**

  ```
  #include <unistd.h>
  int execl (const char *path, const char *arg,...);
  ```

  execl()将 path 所指路径的映像载入内存，arg 是它的第一个参数。参数可变长。参数列表必须以 NULL 结尾。

  通常 execl()不会返回。成功的调用会以跳到新的程序入口点作为结束。发生错误时，execl()返回-1，并设置 errno 值。

  例 编辑/home/kidd/hooks.txt：

  ```
  int ret;
  ret = execl ("/bin/vi", "vi","/home/kidd/hooks.txt", NULL);
  if (ret == -1)
     perror ("execl");
  ```

- **fork 系统调用**

  ```
  #include <sys/types.h>
  #include <unistd.h>
  pid_t fork (void);
  ```

成功调用 fork()会创建一个新的进程，它与调用 fork()的进程大致相同。发生错误时，fork()返回-1，并设置 errno 值。

例：

```
pid_t pid;
pid = fork ();
if (pid > 0)
   printf ("I am the parent of pid=%d!\n", pid);
else if (!pid)
   printf ("I am the baby!\n");
else if (pid == -1)
   perror ("fork");
```

- **终止进程**

exit()系统调用：

```
#include <stdlib.h>
void exit (int status);
```

- **进程挂起**

pause() 系统调用：

```
int pause( void );
```

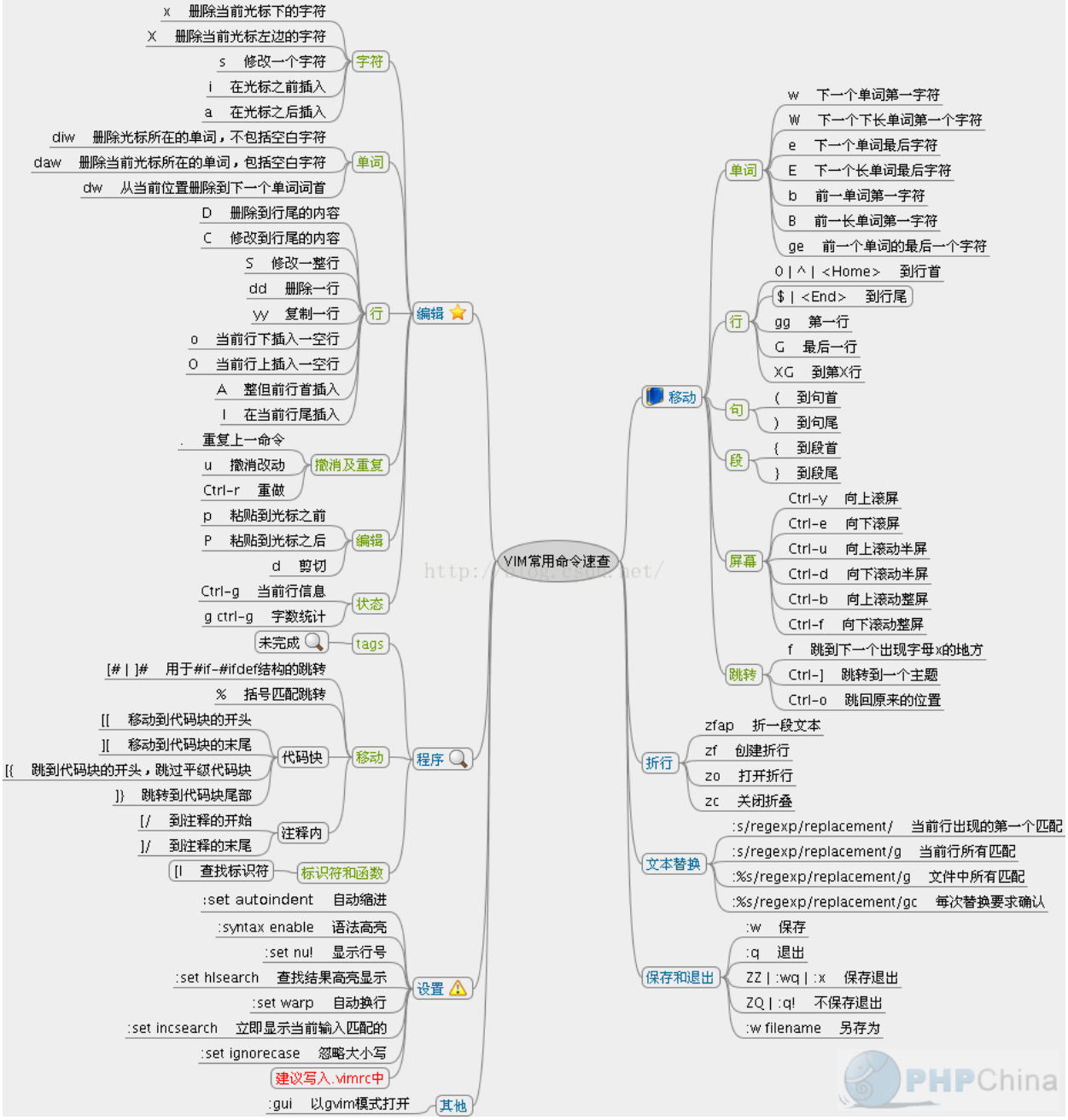
函数 pause 会把进程挂起，直到接收到信号。在信号接收后，进程会从 pause 函数中退出，继续运行。

- **wait(等待子进程中断或结束)**

```
#include<sys/types.h>
#include<sys/wait.h>
pid_t wait (int * status);
```

wait()会暂时停止目前进程的执行,直到有信号来到或子进程结束。
如果在调用 wait()时子进程已经结束,则 wait()会立即返回子进程结束状态值。
子进程的结束状态值会由参数 status 返回,而子进程的进程识别码也会一起返回。
如果不在意结束状态值,则参数 status 可以设成 NULL。

VIM 常用命令速查



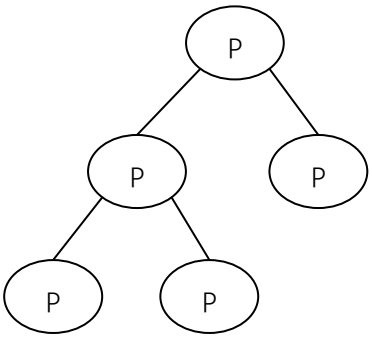
3. 实验题目：

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。
- 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。

- 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。



- 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。

4. 实验过程：

1、 在终端输入 vi 打开 vi 进程。在另一个终端输入 ps -A，查看所有进程的 ID。

```
 2746 ?        00:00:00 gconfd-2
 2759 ?        00:00:00 evolution-calen
 2764 ?        00:00:01 sogou-qimpanel
 2785 ?        00:00:00 evolution-calen
 2788 ?        00:00:00 evolution-addre
 2808 ?        00:00:00 gvfsd-trash
 2859 ?        00:00:00 evolution-addre
 2870 ?        00:00:00 sh
 2893 ?        00:00:00 zeitgeist-daemo
 2908 ?        00:00:00 zeitgeist-fts
 2909 ?        00:00:00 zeitgeist-datah
 2931 ?        00:00:00 gvfsd-metadata
 3000 ?        00:00:00 update-notifier
 3018 ?        00:00:00 gnome-terminal-
 3054 pts/4    00:00:00 bash
 3498 pts/4    00:00:00 vi
 3505 ?        00:00:01 aptd
 3527 ?        00:00:00 deja-dup-monito
 3826 ?        00:00:00 sh
 3827 ?        00:00:00 run-parts
 3972 ?        00:00:00 logrotate
 3977 ?        00:00:00 logrotate
 4015 pts/11   00:00:00 bash
 4041 pts/11   00:00:00 ps
```

输入 ps -e|grep vi 显示出名字含有 vi 的进程的 ID。

```
li@ubuntu:~$ ps -e|grep vi
  1395 ?        00:00:00 VGAuthService
  2434 ?        00:00:00 hud-service
  2506 ?        00:00:00 dconf-service
  3498 pts/4    00:00:00 vi
```

如图所示，打开 vi 编辑器的 ID 是 3498。

输入 ps -ef|grep vi 第二列为进程的 ID，第三列为父进程的 PID：

```
li          3498    3054  0 17:41 pts/4    00:00:00 vi
```

```
li@ubuntu:~$ ps -ef|grep 3054
li          3054    3018  0 17:40 pts/4    00:00:00 bash
```

```
li@ubuntu:~$ ps -ef|grep 3018
li          3018    2205  0 17:40 ?        00:00:04 /usr/lib/gnome-terminal/gnome-
terminal-server
```

```
li@ubuntu:~$ ps -ef|grep 2205
li          2205    1588  0 17:39 ?        00:00:00 /sbin/upstart --user
```

```
li@ubuntu:~$ ps -ef|grep 1588
root        1588    1006  0 17:37 ?        00:00:00 lightdm --session-child 12 19
```

```
li@ubuntu:~$ ps -ef|grep 1006
root        1006       1  0 17:37 ?        00:00:00 /usr/sbin/lightdm
```

依次查询父进程，直到进程号为 1。可得父进程的顺序为：

3498->3054->3018->2205->1588->1006->1

终端输入 pstree -p 查看进程树。

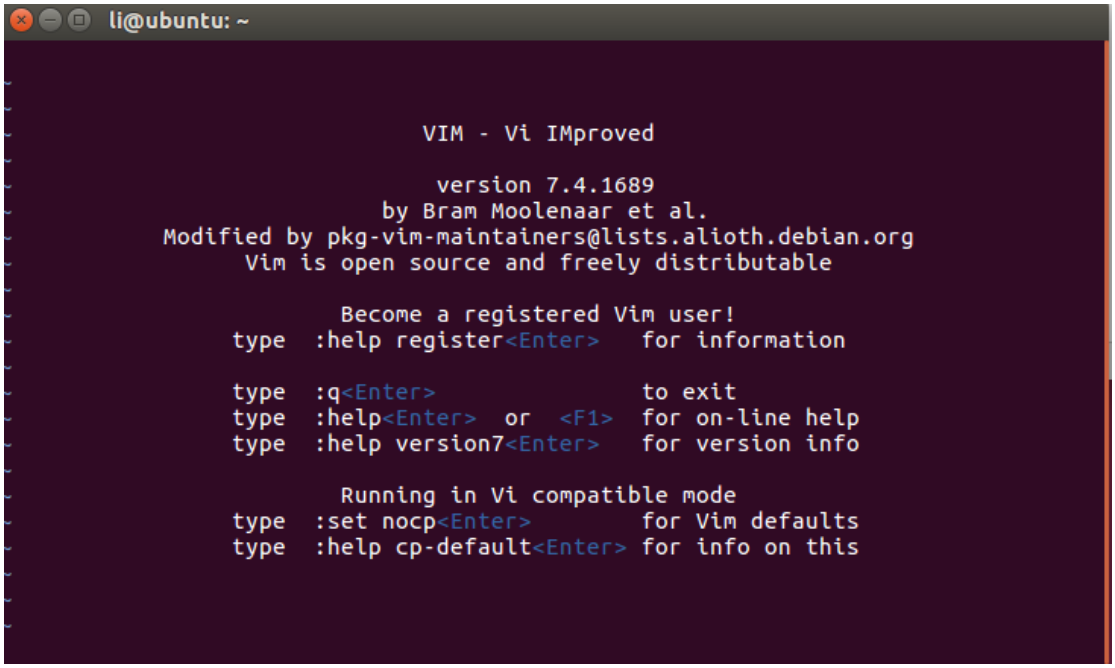
```
li@ubuntu:~$ pstree -p
systemd(1)─┬─ManagementAgent(1447)─┬─{ManagementAgent}(1456)
           │                       ├─{ManagementAgent}(1457)
           │                       ├─{ManagementAgent}(1458)
           │                       ├─{ManagementAgent}(1459)
           │                       ├─{ManagementAgent}(1460)
           │                       └─{ManagementAgent}(1461)
           ├─NetworkManager(836)─┬─dhclient(1056)
           │                     ├─dnsmasq(1078)
           │                     ├─{gdbus}(921)
           │                     └─{gmain}(919)
           ├─VGAuthService(1395)
           ├─accounts-daemon(802)─┬─{gdbus}(916)
           │                      └─{gmain}(914)
           ├─acpid(812)
           ├─agetty(897)
           ├─avahi-daemon(835)───avahi-daemon(918)
           ├─bluetoothd(805)
           ├─colord(1794)─┬─{gdbus}(1855)
           │              └─{gmain}(1850)
           ├─cron(821)
           ├─cups-browsed(4044)─┬─{gdbus}(4047)
           │                    └─{gmain}(4046)
           ├─cupsd(4043)───dbus(4045)
```

由进程 ID 可知两个进程树一样。

2、代码如下：（见 start.c）

```
#include <stdio.h>
#include <unistd.h>
int main(){
pid_t p;
p = fork();
if(p > 0){
int res = execl("/etc/alternatives/vi","vi",NULL);
if(res == -1)
        perror("execl");
}
else if(p == 0)
  while(1){}
return 0;
}
```

打开 vi 如图所示：



输入 ps -A 命令，得到 vi 的进程 ID 为 3880。

```
 3865 pts/4    00:00:00 bash
 3880 pts/4    00:00:00 vi
 3881 pts/4    00:30:57 start
```

输入 ps -ef|grep 3880 查看父进程的进程 ID。

```
li@ubuntu:~$ ps -ef|grep 3880
li          3880    3865  0 02:21 pts/4    00:00:00 vi
li          3881    3880 98 02:21 pts/4    00:34:47 ./start
```

输入 ps -lax 查看进程参数:

```
li@ubuntu:~$ ps -lax
F   UID    PID   PPID PRI  NI    VSZ   RSS WCHAN  STAT TTY        TIME COMMAND
0  1000   3880   3865  20   0  39108  3552 poll_s S+   pts/4      0:00 vi
1  1000   3881   3880  20   0   4220    76 -      R+   pts/4     33:14 ./start
```

可知 vi 进程状态在后台进程组中处于休眠状态，父进程为后台进程组中处于运行状态。

其中 PID 为 vi 进程的进程号。

PPID 为 vi 进程的父进程号。

PRI 是内核调度的优先级。

NI 是进程优先级。

VSZ 是总虚拟内存大小，以 byte 计。

RSS 是进程使用的总物理内存数，以 Kbytes 计。

STAT 为进程状态。

【D:不可终端；R：正在运行或在队列中的进程；S:处于休眠状态；T：停止或被追踪；Z：僵尸进程；W：进入内存交换；X：死掉的进程；<:高优先级；N：低优先级；L：有些页被缩进内存；s:包含子进程；+：位于后台的进程组；|：多线程】

TTY 为终端的次要装置码。

TIME 为使用 CPU 的时间。

输入 top，打开大写键盘，敲入 P。可得到按 CPU 占用率排序的所有进程:

```
li@ubuntu:~$ top

top - 03:19:56 up  2:04,  1 user,  load average: 1.42, 1.11, 1.03
Tasks: 224 total,   2 running, 222 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
KiB Mem :   985856 total,   108040 free,   542464 used,   335352 buff/cache
KiB Swap:  1046524 total,   785916 free,   260608 used.   227508 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S %CPU %MEM     TIME+ COMMAND
 3881 li        20   0    4220     76      0 R 98.7  0.0  57:19.31 start
 1027 root      20   0  433468  32636  15524 S  0.7  3.3   0:37.28 Xorg
 2253 li        20   0 1223920  40188  24920 S  0.3  4.1   0:53.39 compiz
 3860 li        20   0  596968  34012  25712 S  0.3  3.4   0:01.85 gnome-term+
    1 root      20   0  119940   4108   2588 S  0.0  0.4   0:02.70 systemd
    2 root      20   0       0      0      0 S  0.0  0.0   0:00.01 kthreadd
```

Problems:

(1)找不到 vi 的路径，用往常的 execl("/user/bin/vi","vi",NULL);无法打开 vi
在命令行中输入 find -name vi 查找 vi 的路径，

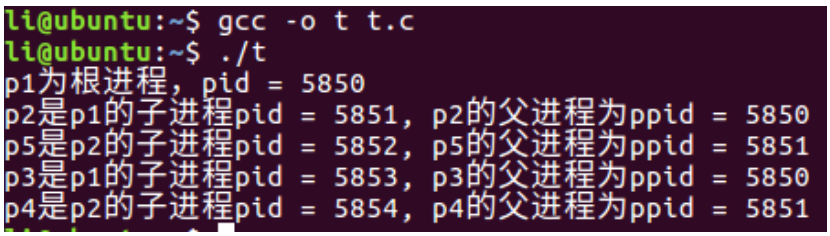
```
find: ‘./etc/polkit-1/localauthority’: Permission denied
./etc/alternatives/vi
find: ‘./root’: Permission denied
```

3、代码如下：(见 t.c)

```
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
int main(){
   pid_t p1,p2,p3,p4;
   p1 = fork();
   if(p1 > 0){
      printf("p1 为根进程， pid = %d\n", getpid());
      sleep(1);
      while((p3=fork())==-1);
      if(p3>0)sleep(1);
      else if(p3 == 0){
      printf("p3 是 p1 的子进程 pid = %d, p3 的父进程为 ppid = %d\n", getpid(), getppid());
      }
   }else if(p1 == 0){
      while((p2=fork())==-1);
      if(p2>0){
         printf("p2 是 p1 的子进程 pid = %d, p2 的父进程为 ppid = %d\n", getpid(), getppid());
         sleep(1);
         while((p4=fork())==-1);
         if(p4>0)sleep(1);
         else if(p4 == 0){
         printf("p4 是 p2 的子进程 pid = %d, p4 的父进程为 ppid = %d\n", getpid(), getppid());
         }
```

```
    }else if(p2 == 0){
    printf("p5 是 p2 的子进程 pid = %d, p5 的父进程为 ppid = %d\n", getpid(), getppid());
    }
  }
  return 0;
}
```

运行结果如下所示：



可得 p1 进程 ID 为 5850。

p2 进程 ID 为 5851，父进程为 5850，即 p1。p3 进程 ID 为 5853，父进程为 5850，即 p1。

p4 进程 ID 为 5854，父进程为 5851，即 p2。p5 进程 ID 为 5852，父进程为 5851，即 p2。

4、循环输出进程代码如下：（见 fff.c）

```
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
int main(){
  pid_t p1,p2,p3,p4;
  p1 = fork();
  if(p1 > 0){
    sleep(1);
    while((p3=fork())==-1);
    if(p3>0)sleep(1);
    else if(p3 == 0){
    while(1){printf("p3 是 p1 的子进程 pid = %d, p3 的父进程为 ppid = %d\n", getpid(), getppid());sleep(1);}
    }
    while(1){printf("p1 为根进程， pid = %d\n", getpid());sleep(1);}
  }else if(p1 == 0){
    while((p2=fork())==-1);
    if(p2>0){
      sleep(1);
      while((p4=fork())==-1);
      if(p4>0)sleep(1);
      else if(p4 == 0){
      while(1){printf("p4 是 p2 的子进程 pid = %d, p4 的父进程为 ppid = %d\n", getpid(), getppid());sleep(1);}
      }
      while(1){printf("p2 是 p1 的子进程 pid = %d, p2 的父进程为 ppid = %d\n", getpid(), getppid());sleep(1);}
```

```
    }else if(p2 == 0){
    while(1){printf("p5 是 p2 的子进程 pid = %d, p5 的父进程为 ppid = %d\n", getpid(), getppid());sleep(1);}
    }
  }
  return 0;
}
```

```
li@ubuntu:~$ gcc -o fff fff.c
li@ubuntu:~$ ./fff
p5是p2的子进程pid = 6569, p5的父进程为ppid = 6568
p5是p2的子进程pid = 6569, p5的父进程为ppid = 6568
p3是p1的子进程pid = 6570, p3的父进程为ppid = 6567
p4是p2的子进程pid = 6571, p4的父进程为ppid = 6568
p1为根进程, pid = 6567
p2是p1的子进程pid = 6568, p2的父进程为ppid = 6567
p5是p2的子进程pid = 6569, p5的父进程为ppid = 6568
p4是p2的子进程pid = 6571, p4的父进程为ppid = 6568
p3是p1的子进程pid = 6570, p3的父进程为ppid = 6567
```

（1）输入 kill -9 6568，关闭 ID 为 6568 的进程，即 p2。可看到 p4 和 p5 的父进程成为 p1 的父进程。

```
p4是p2的子进程pid = 6571, p4的父进程为ppid = 1719
p1为根进程, pid = 6567
p5是p2的子进程pid = 6569, p5的父进程为ppid = 1719
p4是p2的子进程pid = 6571, p4的父进程为ppid = 1719
p3是p1的子进程pid = 6570, p3的父进程为ppid = 6567
p1为根进程, pid = 6567
p4是p2的子进程pid = 6571, p4的父进程为ppid = 1719
li@ubuntu: ~
li@ubuntu:~$ kill -9 6568
li@ubuntu:~$ ps -lax
F   UID     PID    PPID PRI  NI    VSZ   RSS WCHAN  STAT TTY        TIME COMMAND
```

输入 ps -lax 查看各进程的运行状态。

```
0  1000    6567    6551  20   0   4352   652 hrtime S+   pts/20     0:00 ./fff
1  1000    6568    6567  20   0      0     0 -      Z+   pts/20     0:00 [fff] <de
1  1000    6569    1719  20   0   4352    76 hrtime S+   pts/20     0:00 ./fff
1  1000    6570    6567  20   0   4352    72 hrtime S+   pts/20     0:00 ./fff
1  1000    6571    1719  20   0   4352    76 hrtime S+   pts/20     0:00 ./fff
```

VSZ 为总虚拟内存大小，p2 进程虚拟内存变为 0。P2 状态为退出状态，进程称为僵尸进程。p1，p3，p4，p5 进程为可中断的睡眠状态。

（2）exit(0)退出

（代码见 t2.c)

在 p2 进程中插入 exit(0)语句，关闭进程。可看到 p4 和 p5 的父进程为 p1 的父进程。

```
p1为根进程, pid = 4395
p5是p2的子进程pid = 4397, p5的父进程为ppid = 1874
p4是p2的子进程pid = 4399, p4的父进程为ppid = 1874
p3是p1的子进程pid = 4398, p3的父进程为ppid = 4395
```

输入 ps -lax 查看各进程的运行状态：

```
0  1000   4395   4360  20   0   4352   664 hrtime S+   pts/4      0:00 ./t2
1  1000   4396   4395  20   0      0     0 -      Z+   pts/4      0:00 [t2] <def
1  1000   4397   1874  20   0   4352    76 hrtime S+   pts/4      0:00 ./t2
1  1000   4398   4395  20   0   4352    72 hrtime S+   pts/4      0:00 ./t2
1  1000   4399   1874  20   0   4352    76 hrtime S+   pts/4      0:00 ./t2
```

结果与 kill-9 得到的结果相同。

（3）段错误退出

（代码见 y.c）

在 p2 进程中插入语句: int *ptr = NULL;*ptr = 0;关闭进程。可看到 p4 和 p5 的父进程为 p1 的父进程。

```
p1为根进程, pid = 4505
p5是p2的子进程pid = 4507, p5的父进程为ppid = 4506
p4是p2的子进程pid = 4509, p4的父进程为ppid = 4506
p3是p1的子进程pid = 4508, p3的父进程为ppid = 4505
```

输入 ps -lax 查看各进程的运行状态，结果与（1）（2）相同。

```
0  1000   4505   4449  20   0   4352   724 hrtime S+   pts/4      0:00 ./y
1  1000   4506   4505  20   0      0     0 -      Z+   pts/4      0:00 [y] <defu
1  1000   4507   1874  20   0   4352    80 hrtime S+   pts/4      0:00 ./y
1  1000   4508   4505  20   0   4352    76 hrtime S+   pts/4      0:00 ./y
1  1000   4509   1874  20   0   4352    80 hrtime S+   pts/4      0:00 ./y
```