操作系统第二次实验：进程控制

16281035 陈琦 计科1601

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。

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);*

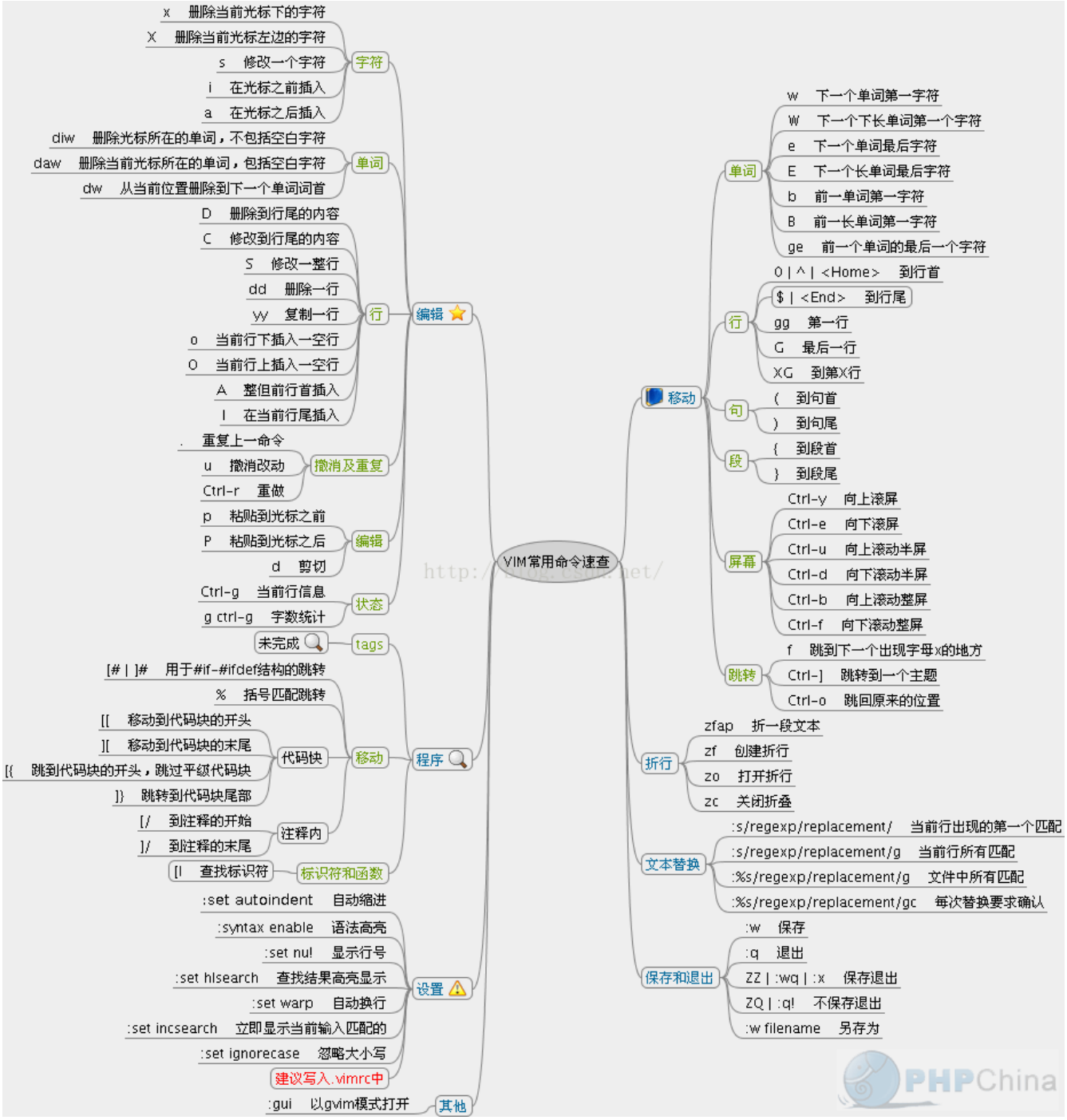
wait()会暂时停止目前进程的执行,直到有信号来到或子进程结束。

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



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。

1. 首先我先自己创建一个名为VI的vim文件，之后用vim打开

```
[xaviershank@xaviershank ~]$ touch VI
[xaviershank@xaviershank ~]$ vim VI
[xaviershank@xaviershank ~]$ ls
公共  模板  视频  图片  文档  下载  音乐  桌面  download  study  VI
```

2. 打开界面如下：

我在其中启用了编辑命令，编辑了以下内容：

3. 保存

4. 查看vim进程号，5065

```
[xaviershank@xaviershank ~]$ ps -a
  PID TTY          TIME CMD
 5065 pts/0    00:00:00 vim
 5090 pts/1    00:00:00 ps
[xaviershank@xaviershank ~]$ ps -a -ocmd
CMD
vim VI
ps -a -ocmd
[xaviershank@xaviershank ~]$
```

5. 寻找vim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

```
F   UID   PID  PPID PRI  NI    VSZ   RSS WCHAN  STAT TTY        TIME COMMAND
xaviers+  4619   908  1 08:26 ?          00:00:16 /usr/lib/gnome-terminal-server
xaviers+  4626  4619  0 08:26 pts/0      00:00:00 bash
xaviers+  5065  4626  0 08:31 pts/0      00:00:00 vim VI
```

```
xaviers+   908     1  0 07:55 ?          00:00:00 /usr/lib/systemd/systemd --user
```

init进程的pid是1.

init进程是其他所有进程的父进程.

列出父进程号是1的进程，就是列出除了init外的所有进程.

所以寻找vim的父进程如下：

5065 -> 4626 -> 4619 -> 908 -> 1

6. 将得到的进程树和由pstree命令的得到的进程树进行比较。

```
        ├─evolution-sourc(1233)─┬─{evolution-sourc}(1235)
        │                       ├─{evolution-sourc}(1236)
        │                       └─{evolution-sourc}(1237)
        ├─gconfd-2(1254)
        ├─gnome-terminal-(4619)─┬─bash(4626)───vim(5065)
        │                       ├─bash(6884)───pstree(6911)
        │                       ├─{gnome-terminal-}(4620)
        │                       ├─{gnome-terminal-}(4621)
```

```
├─systemd(908)─┬─(sd-pam)(909)
│              ├─at-spi-bus-laun(945)─┬─dbus-daemon(951)
│              │                      └─{at-spi-bus-laun}(946)
```

```
[xaviershank@xaviershank ~]$ pstree -p
systemd(1)─┬─ModemManager(607)─┬─{ModemManager}(617)
           │                   └─{ModemManager}(622)
           ├─NetworkManager(606)─┬─{NetworkManager}(620)
           │                     └─{NetworkManager}(623)
           ├─accounts-daemon(636)─┬─{accounts-daemon}(637)
           │                      └─{accounts-daemon}(639)
           ├─avahi-daemon(602)───avahi-daemon(624)
           ├─bluetoothd(604)
```

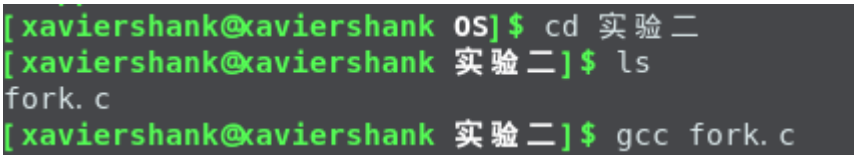
两个相比较，是一样的结果： 5065 -> 4626 -> 4619 -> 908 -> 1

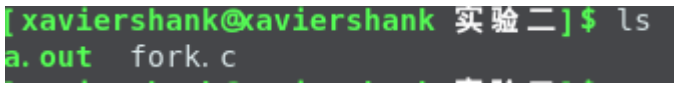
2、编写程序，首先使用fork系统调用，创建子进程。在父进程中继续执行空循环操作；在子进程中调用exec打开vi编辑器。然后在另外一个终端中，通过ps –Al命令、ps aux或者top等命令，查看vi进程及其父进程的运行状态，理解每个参数所表达的意义。选择合适的命令参数，对所有进程按照CPU占用率排序。

1. 程序代码 fork.c

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

int main(int argc, char *argv[]){
pid_t pid = fork();

//在子进程
if(pid == 0)
{
int num = execl("/usr/bin/vim","vim","NULL");
if(num == -1)
perror("execl");
}

else if(pid>0)
{
//在父进程
while(1){}
}

else
{
printf("fork调用失败");
}
return (0);
}
```

2. 运行结果：打开了vim





3. 查看vi进程及父进程的运行状态以及各个参数的意思：

```
[xaviershank@xaviershank ~]$ ps -al
F S   UID   PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
0 R  1000  2057  2029 99  80   0 -   541 -      pts/0    00:00:57 a.out
0 S  1000  2058  2057  0  80   0 -  5187 core_s pts/0    00:00:00 vim
0 R  1000  2321  2309  0  80   0 -  3947 -      pts/1    00:00:00 ps
[xaviershank@xaviershank ~]$ ps -aux
USER        PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root          1  0.2  0.1 191580 10296 ?        Ss   11:37   0:01 /sbin/init
root          2  0.0  0.0      0     0 ?        S    11:37   0:00 [kthreadd]
root          3  0.0  0.0      0     0 ?        I<   11:37   0:00 [rcu_gp]
root          4  0.0  0.0      0     0 ?        I<   11:37   0:00 [rcu_par_gp]
```

%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

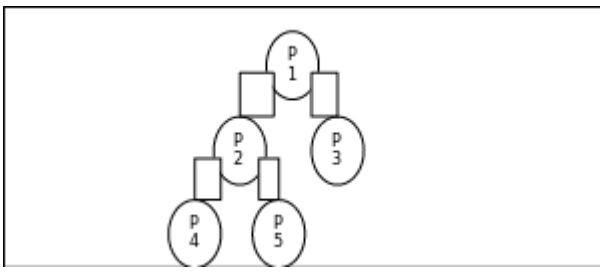
START （进程启动时间和日期）
TIME ；（进程使用的总cpu时间）
COMMAND （正在执行的命令行命令）
NI (nice)优先级
PRI 进程优先级编号
PPID 父进程的进程ID（parent process id）
SID 会话ID（session id）
WCHAN 进程正在睡眠的内核函数名称；该函数的名称是从/root/system.map文件中获得的。
FLAGS 与进程相关的数字标识

4. 选择合适的命令参数，对所有进程按照CPU占用率排序

```
[xaviershank@xaviershank ~]$ ps auxw --sort=%cpu
USER        PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root          2  0.0  0.0      0     0 ?        S    11:37   0:00 [kthreadd
root          3  0.0  0.0      0     0 ?        I<   11:37   0:00 [rcu_gp]
```

```
xaviers+  2801  1.7  1.4 630488 120364 ?       Sl   11:48   0:01 /opt/google/chrome/chrome --type=renderer --field-trial-handle=1677
xaviers+  1071  3.0  1.1 1062256 92668 ?       Sl   11:40   0:17 budgie-wm
xaviers+  2820  3.6  1.6 661744 130048 ?       Sl   11:48   0:03 /opt/google/chrome/chrome --type=renderer --field-trial-handle=1677
xaviers+  2329  4.0  2.1 973880 174320 ?       Sl   11:46   0:08 /opt/google/chrome/chrome
xaviers+  1410  4.2  3.5 5966896 287952 ?      Sl   11:41   0:21 /usr/lib/libreoffice/program/soffice.bin --writer file:///home/xavi
root       617  4.5  1.7 679232 139632 tty7    Ssl+ 11:38   0:31 /usr/lib/Xorg :0 -seat seat0 -auth /run/lightdm/root/:0 -nolisten t
xaviers+  2369  5.0  1.0 354752 85712 ?        Sl   11:46   0:09 /opt/google/chrome/chrome --type=gpu-process --field-trial-handle=1
xaviers+  2706 11.3  1.9 684560 159688 ?       Sl   11:46   0:19 /opt/google/chrome/chrome --type=renderer --field-trial-handle=1677
xaviers+  2057 99.7  0.0   2164   748 pts/0    R+   11:44   5:38 ./a.out
```

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. 首先对fork()函数调用进行简单的测试

4_1.cpp ● fork.c forktest.c ×

```
1  #include <stdio.h>
2  #include <unistd.h>
3
4  int main()
5  {
6      fork();
7      printf("hello");
8      return 0;
9  }
```

运行结果

```
文件(F)  编辑(E)  查看(V)  搜索(S)  终端(T)  帮助(H)
[xaviershank@xaviershank ~]$ code
[xaviershank@xaviershank ~]$ ls
公共  模板  视频  图片  文档  下载  音乐  桌面  download  study  VI
[xaviershank@xaviershank ~]$ cd study
[xaviershank@xaviershank study]$ ls
OS  Software
[xaviershank@xaviershank study]$ cd OS
[xaviershank@xaviershank OS]$ ls
实验报告  实验二  实验目录  实验一
[xaviershank@xaviershank OS]$ cd 实验二
[xaviershank@xaviershank 实验二]$ ls
a.out  fork.c  forktest.c
[xaviershank@xaviershank 实验二]$ gcc forktest.c
[xaviershank@xaviershank 实验二]$ ls
a.out  fork.c  forktest.c
[xaviershank@xaviershank 实验二]$ ./a.out
hellohello[xaviershank@xaviershank 实验二]$
```

输出两个hello

所以：函数通过系统调用创建一个与原来进程几乎完全相同的进程，这个新产生的进程称为子进程。一个进程调用fork（）函数后，系统先给新的进程分配资源，例如存储数据和代码的空间。然后把原来的进程的所有值都复制到新的新进程中，只有少数值与原来的进程的值不同。相当于克隆了一个自己。需要注意的一点：就是调用fork函数之后，一定是两个进程同时执行的代码段是fork函数之后的代码，而之前的代码以及由父进程执行完毕。

fork()返回值意义如下：

=0：在子进程中

0：在父进程中

<0：创建失败 转自：

作者：Sunrise永不言弃

来源：CSDN

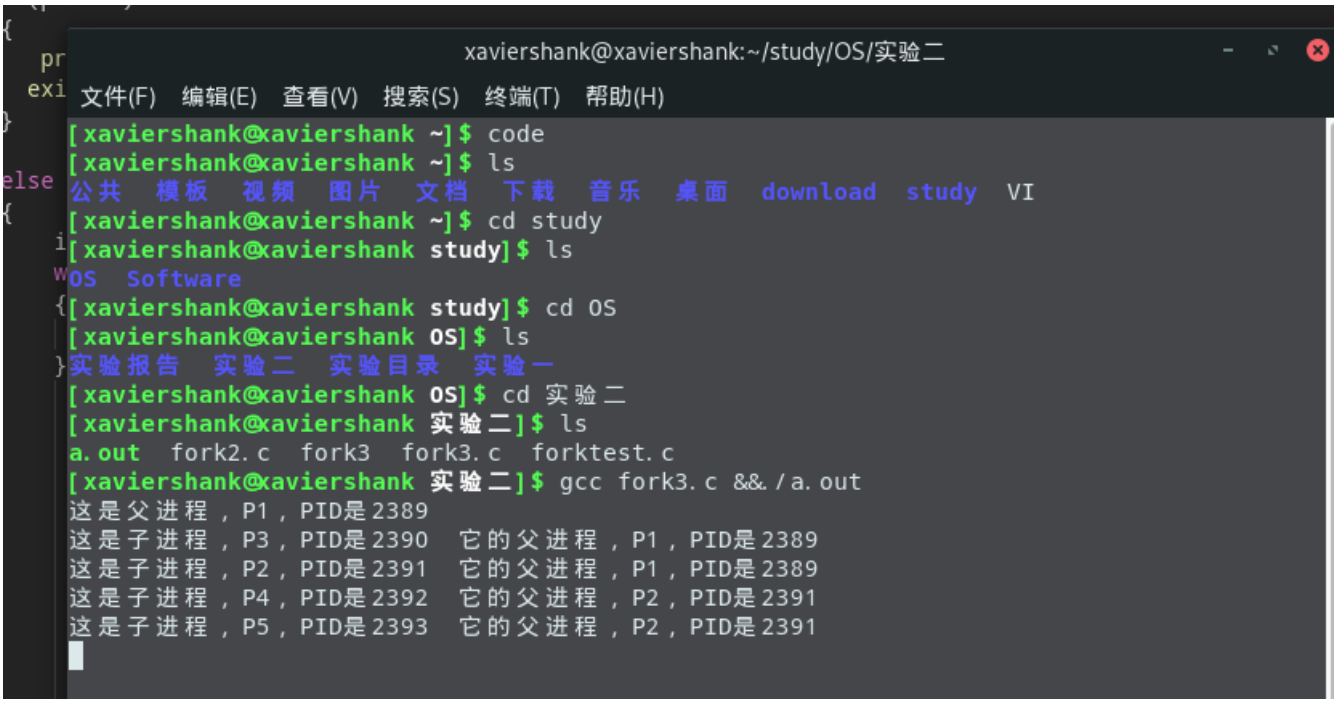
原文：https://blog.csdn.net/qq_38898129/article/details/80827280

2. 源代码 fork3.c

```
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/wait.h>
#include <time.h>
#include <stdarg.h>
#include<sys/types.h> //这个头文件不能少，否则pid_t没有定义

int main(int argc, char *argv[])
{
pid_t pidA,pidB,pidC,pidD;
printf("这是父进程，P1，PID是%d\n",getpid());

pidA = fork(); //创建新进程

if(pidA<0)
{
printf("新进程创建失败\n");
return 0;
}

else if(pidA == 0)
{
printf("这是子进程，P3，PID是%d\n",getpid());
return 0;
}

else
{
pidB = fork();
if(pidB<0)
{
printf("新进程创建失败\n");
return 0;
}

else if(pidB == 0)
```

```
{
printf("我是子进程，P2，PID是%d\n",getpid());

pidC = fork();
if(pidC == 0)
{
printf("这是子进程，P4，PID是%d\n",getpid());
return 0;//防止产生孙进程
}

pidD = fork();
if(pidD == 0)
{
printf("这是子进程，P5，PID是%d\n",getpid());
return 0;//防止产生孙进程
}
}
}
return 0;
}
```

3. 运行结果



4. 代码改进：我发现代码如上述，不论是父进程还是子进程都仍旧在执行，不会停止。

   所以我在原来的基础上加入了如下的代码：

```
int *status;
            waitpid(pidC,status, 0);
            waitpid(pidD,status, 0);
            return 0;
        }
    }
    int *status;
    waitpid(pidB,status, 0);
    waitpid(pidA,status, 0);
    return 0;
```

得到的运行结果自动停止

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. kill -9

```
viershank@xaviershank ~]$ ps -al
  UID   PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
 1000  5226  5078  0  80   0 -   574 -      pts/1    00:00:00 a.out
 1000  5227  5226 33  80   0 -   574 -      pts/1    00:00:01 a.out
 1000  5228  5226 33  80   0 -   574 -      pts/1    00:00:01 a.out
 1000  5229  5228 33  80   0 -   574 -      pts/1    00:00:01 a.out
 1000  5230  5228 33  80   0 -   574 -      pts/1    00:00:01 a.out
 1000  5231  5161  0  80   0 -  3947 -      pts/0    00:00:00 ps
viershank@xaviershank ~]$ kill -9 5228
viershank@xaviershank ~]$ ps -al
  UID   PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
 1000  5226  5078  0  80   0 -   574 -      pts/1    00:00:00 a.out
 1000  5227  5226 32  80   0 -   574 -      pts/1    00:00:08 a.out
 1000  5229   895 32  80   0 -   574 -      pts/1    00:00:08 a.out
 1000  5230   895 32  80   0 -   574 -      pts/1    00:00:08 a.out
 1000  5233  5161  0  80   0 -  3947 -      pts/0    00:00:00 ps
viershank@xaviershank ~]$
```

在杀掉进程P2，进程号为5228之后，P4和P5的PPID不再是5228，而变成了895.

我在进程树中查询，发现systemd(895), 也就是相当于最原始的一个根进程。那么我就有很多的问题：

（1）为什么这些失去父进程的“孤儿”子进程的PPID不是systemd(1)?

（2）systemd(1)和systemd(895)之间的关联是什么？

2. 自己正常退出

   源代码

```
#include <unistd.h>
```

```c
#include <stdio.h>
#include <stdlib.h>
#include <sys/wait.h>
#include <time.h>
#include <stdarg.h>
#include<sys/types.h> //这个头文件不能少，否则pid_t没有定义

int main(int argc, char *argv[])
{
pid_t pidA,pidB,pidC,pidD;
int i;
printf("这是父进程，P1，PID是%d\n",getpid());

pidA = fork(); //创建新进程

if(pidA<0)
{
printf("新进程创建失败\n");
exit(0);
}

else if(pidA == 0)
{
i=100;
while(i-->0)
{
printf("这是子进程，P3，PID是%d 它的父进程，P1，PID是%d\n",getpid(),getppid());
}
return 0;
}

else
{
pidB = fork();
if(pidB<0)
{
printf("新进程创建失败\n");
return 0;
}

else if(pidB == 0)
{
i=100;
pidC = fork();
if(pidC == 0)
{
i=100;
while(i-->0){
printf("这是子进程，P4，PID是%d 它的父进程，P2，PID是%d\n",getpid(),getppid());}
return 0;//防止产生孙进程
}

pidD = fork();
```

```
if(pidD == 0)
{
i=100;
while(i-->0){
printf("这是子进程，P5，PID是%d 它的父进程，P2，PID是%d\n",getpid(),getppid());
}
return 0;//防止产生孙进程
}
i=100;
while(i-->0)
{
printf("这是子进程，P2，PID是%d 它的父进程，P1，PID是%d\n",getpid(),getppid());
if(i==50) {
    exit(0);
}
}

int *status;
waitpid(pidC,status, 0);
waitpid(pidD,status, 0);
return 0;
}
}
int *status;
waitpid(pidB,status, 0);
waitpid(pidA,status, 0);
return 0;

}
```

运行结果：

```
这是子进程，P2，PID是6879  它的父进程，P1，PID是6877
这是子进程，P4，PID是6880  它的父进程，P2，PID是6879
这是子进程，P5，PID是6881  它的父进程，P2，PID是6879
这是子进程，P3，PID是6878  它的父进程，P1，PID是6877
这是子进程，P4，PID是6880  它的父进程，P2，PID是6879
这是子进程，P5，PID是6881  它的父进程，P2，PID是6879
这是子进程，P3，PID是6878  它的父进程，P1，PID是6877
这是子进程，P4，PID是6880  它的父进程，P2，PID是6879
这是子进程，P5，PID是6881  它的父进程，P2，PID是6879
这是子进程，P3，PID是6878  它的父进程，P1，PID是6877
这是子进程，P4，PID是6880  它的父进程，P2，PID是6879
这是子进程，P5，PID是6881  它的父进程，P2，PID是6879
这是子进程，P3，PID是6878  它的父进程，P1，PID是6877
这是子进程，P4，PID是6880  它的父进程，P2，PID是6879
这是子进程，P5，PID是6881  它的父进程，P2，PID是6879
这是子进程，P3，PID是6878  它的父进程，P1，PID是6877
这是子进程，P4，PID是6880  它的父进程，P2，PID是6879
这是子进程，P5，PID是6881  它的父进程，P2，PID是6879
这是子进程，P3，PID是6878  它的父进程，P1，PID是6877
这是子进程，P4，PID是6880  它的父进程，P2，PID是6879
这是子进程，P3，PID是6878  它的父进程，P1，PID是6877
这是子进程，P5，PID是6881  它的父进程，P2，PID是6879
这是子进程，P4，PID是6880  它的父进程，P2，PID是6879
这是子进程，P3，PID是6878  它的父进程，P1，PID是6877
这是子进程，P5，PID是6881  它的父进程，P2，PID是895
这是子进程，P4，PID是6880  它的父进程，P2，PID是895
这是子进程，P3，PID是6878  它的父进程，P1，PID是6877
这是子进程，P5，PID是6881  它的父进程，P2，PID是895
这是子进程，P4，PID是6880  它的父进程，P2，PID是895
这是子进程，P3，PID是6878  它的父进程，P1，PID是6877
这是子进程，P5，PID是6881  它的父进程，P2，PID是895
这是子进程，P4，PID是6880  它的父进程，P2，PID是895
```

和第1问结果一样，只不过它运行过快，未自动结束前的部分没来得及截图。

3. 段错误退出

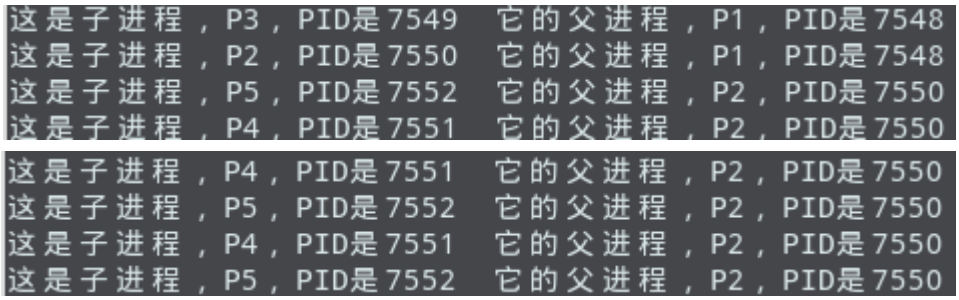
源代码

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/wait.h>
#include <time.h>
#include <stdarg.h>
#include<sys/types.h> //这个头文件不能少，否则pid_t没有定义

int main(int argc, char *argv[])
{
pid_t pidA,pidB,pidC,pidD;
int i;
```

```
printf("这是父进程，P1，PID是%d\n",getpid());

pidA = fork(); //创建新进程

if(pidA<0)
{
printf("新进程创建失败\n");
exit(0);
}

else if(pidA == 0)
{
i=100;
while(i-->0)
{
printf("这是子进程，P3，PID是%d 它的父进程，P1，PID是%d\n",getpid(),getppid());
}
return 0;
}

else
{
pidB = fork();
if(pidB<0)
{
printf("新进程创建失败\n");
return 0;
}

else if(pidB == 0)
{
i=100;
pidC = fork();
if(pidC == 0)
{
i=100;
while(i-->0){
printf("这是子进程，P4，PID是%d 它的父进程，P2，PID是%d\n",getpid(),getppid());}
return 0;//防止产生孙进程
}

pidD = fork();
if(pidD == 0)
{
i=100;
while(i-->0){
printf("这是子进程，P5，PID是%d 它的父进程，P2，PID是%d\n",getpid(),getppid());
}
return 0;//防止产生孙进程
}
i=100;
while(i-->0)
{
```

```
printf("这是子进程，P2，PID是%d 它的父进程，P1，PID是%d\n",getpid(),getppid());
if(i==50) {
int *ptr = NULL;
*ptr = 0;
}
}

int *status;
waitpid(pidC,status, 0);
waitpid(pidD,status, 0);
return 0;
}
}
int *status;
waitpid(pidB,status, 0);
waitpid(pidA,status, 0);
return 0;

}
```

运行结果：



段错误退出从以上结果来看不会影响P2？它仍旧是P4和P5的父进程。