实验报告

课程名称：操作系统

任课教师：何永忠

学生姓名：张云肖

学生学号：16281244

专业年级：安全 1601 班

学院名称：计算机学院

2018 年 3 月 16 日

实验名称:进程控制

班级：安全 1601　　姓名：张云肖　　学号：16281244

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 .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。

2 .1. 基础知识

2 .1.1.ps 命令

ps 命令是基本进程查看命令。使用该命令可以确定有哪些进程正在运行和运行的状态、进程是否结束，进程有没有僵尸、哪些进程占用了过多的资源等等。总之大部分信息都是可以通过执行该命令得到。需注意，ps 是显示瞬间进程的状态，并不是动态连续的。如果想对进程进行实时监控应该用 top 命令。

参数：

- -A：所有进程均显示出来，与-e 具有同样的效用；
- -a：显示现行终端机下的所有进程，包括其他用户的进程；
- -u：以用户为主的进程状态；
- x：通常与 a 这个参数一起使用可列出较完整信息。

输出格式规划：

- l：较长、较详细的讲 PID 的信息列出；
- j：工作的格式（jobs format）；
- -f：做出一个更为完整的输出。

各相关信息的意义：

- F：程序的旗标（flag）；
- S：程序的状态（state）；
- UID：执行者的身份；
- PID：进程的 ID 号；
- PPID：父进程的 ID 号；
- C：CPU 使用的资源百分比；
- PRI：进程的执行优先权，其值越小越早被执行；
- NI：这个经常的 nice 值，表示进程可被执行的优先级的修正数值；
- ADDR：内核函数，指出该程序在内存的哪个部分；
- SZ：使用掉的内存大小；
- WCHAN：目前这个程序是否正在运作当中，若为-表示正在运作；
- TTY：登录者的终端机位置；
- TIME：使用掉的 CPU 时间；

- CMD：所下达的指令名称；
- RSS：该进程占用的固定的内存量
- COMMAND：该程序的实际指令

2.1.2. pstree 命令

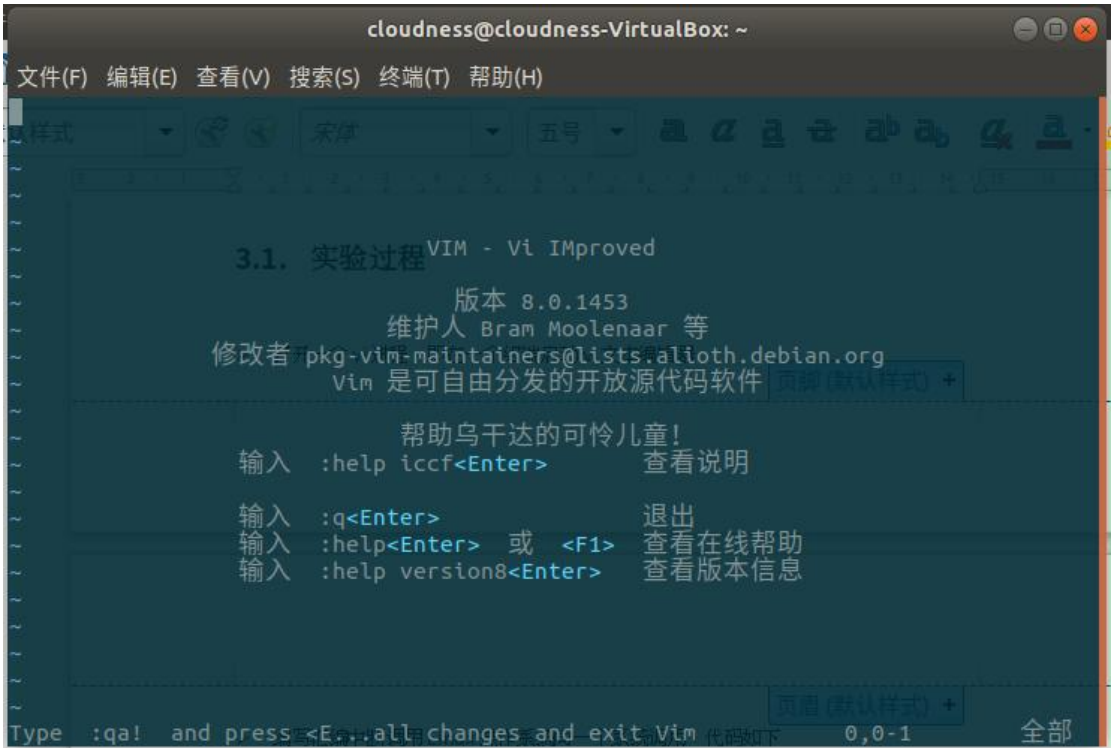
pstree 命令以树状图的方式展现进程之间的派生关系，显示效果比 p s 更加直观。

选项：

- -a：显示每个程序的完整指令，包含路径，参数或是常驻服务的标示；
- -c：不使用精简标示法；
- -G：使用VT100终端机的列绘图字符；
- -h：列出树状图时，特别标明现在执行的程序；
- -H<程序识别码>：此参数的效果和指定"-h"参数类似，但特别标明指定的程序；
- -l：采用长列格式显示树状图；-n：用程序识别码排序。预设是以程序名称来排序；
- -p：显示程序识别码；
- -u：显示用户名称；
- -U：使用 UTF-8 列绘图字符；-V：显示版本信息。

2.2. 实验过程

➢ 打开一个 vi 进程，即在一个终端启动 vi 文本编辑器



- 打开另一个终端，输入 ps -f -C vi 命令查看名字为 vi 的进程详细信息

  发现：ｖｉ进程的执行者时：ｃｌｏｕｄｎｅｓｓ，进程号是３０９２，父进程号是２３５６，ＣＰＵ使用资源为０，进程启动时间为１８：１４，登入者终端机位置为ｐｔｓ/0，使用掉的ＣＰＵ时间为０，所下达的指令名称为ｖｉ

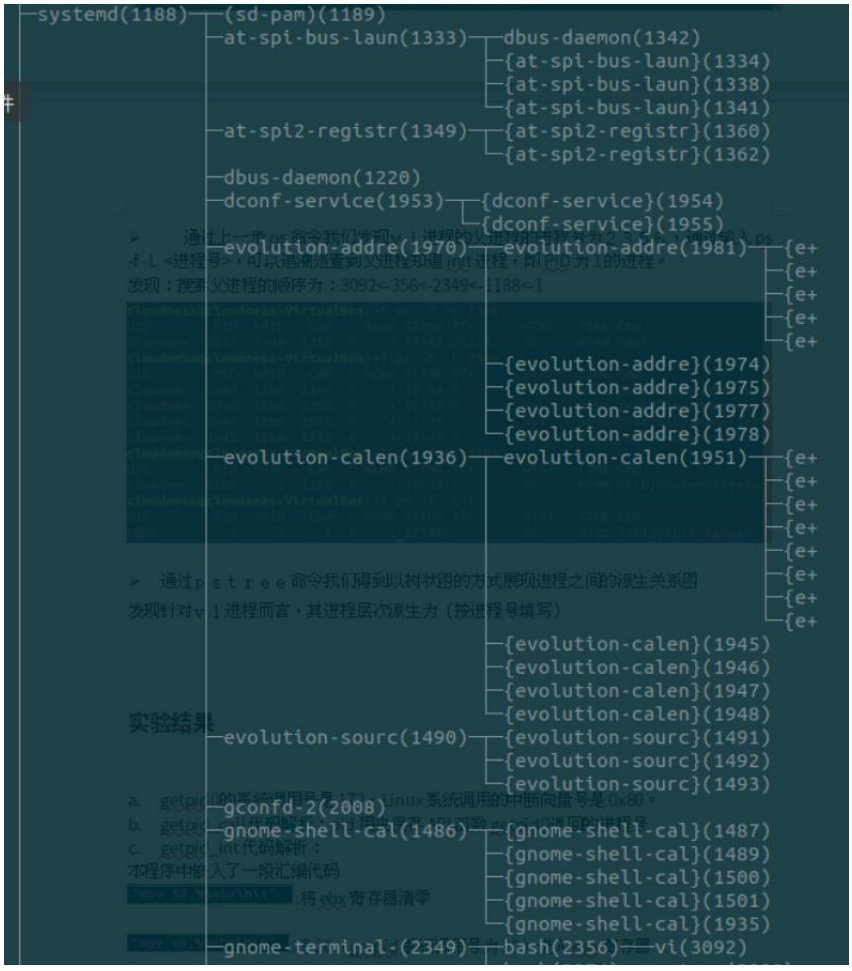
```
cloudness@cloudness-VirtualBox:~$ ps -f -C vi
UID        PID  PPID  C STIME TTY          TIME CMD
cloudne+  3092  2356  0 18:14 pts/0    00:00:00 vi
```

- 通过上一步 ps 命令我们发现ｖｉ进程的父进程的进程号为２３５６，通过输入 ps -f -L <进程号>，可以追溯巡查到父进程知道 init 进程，即 PID 为 1 的进程。

```
cloudness@cloudness-VirtualBox:~$ ps -f -L 2356
UID        PID  PPID   LWP  C NLWP STIME TTY      STAT   TIME CMD
cloudne+  2356  2349  2356  0    1 17:49 pts/0    Ss     0:00 bash
cloudness@cloudness-VirtualBox:~$ ps -f -L 2349
UID        PID  PPID   LWP  C NLWP STIME TTY      STAT   TIME CMD
cloudne+  2349  1188  2349  0    4 17:49 ?        Ssl    0:01 /usr/lib/gnome-ter
cloudne+  2349  1188  2350  0    4 17:49 ?        Ssl    0:00 /usr/lib/gnome-ter
cloudne+  2349  1188  2351  0    4 17:49 ?        Ssl    0:00 /usr/lib/gnome-ter
cloudne+  2349  1188  2352  0    4 17:49 ?        Ssl    0:00 /usr/lib/gnome-ter
cloudness@cloudness-VirtualBox:~$ ps -f -L 1188
UID        PID  PPID   LWP  C NLWP STIME TTY      STAT   TIME CMD
cloudne+  1188     1  1188  0    1 17:47 ?        Ss     0:00 /lib/systemd/syste
cloudness@cloudness-VirtualBox:~$ ps -f -L 1
UID        PID  PPID   LWP  C NLWP STIME TTY      STAT   TIME CMD
root         1     0     1  0    1 17:46 ?        Ss     0:01 /sbin/init splash
```

过ｐｓｔｒｅｅ命令我们得到以树状图的方式展现进程之间的派生关系图

发现针对ｖｉ进程而言，其进程层次派生为（按进程号填写）:3092<-2356<-2349<-1188<-1。（vi<-bash<-gnomw-terminal<-systemd<-init）与ｐｓ命令下得到的派生树相同。



3 .Task 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。

3.1. 基础知识

3.1.1.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，这个过程被称为运行一个新的程序，相应的系统调用成为 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

3.1.2.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");
```

3.1.3.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”);
```

注意：fork 调用的奇妙之处在于，它仅仅被调用了一次，却能够返回两次，并可能有三种不同的返回值。

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函数返回新创建的子进程的进程ＩＤ，我们可以通过 fork 返回的值来判断当前是子进程还是父进程。

3.2. 实验过程

- 编写程序 task2.c，代码如下

程序解释：首先使用 fork 系统调用，创建子进程。fork 函数一共返回两次，一次为 0 ，一次为创建的子进程的进程号，当返回的 frok 大于 0 时，在父进程中继续执行空循环操作；当返回的 frok 等于 0 时，在子进程中调用 exec 打开 vi 编辑器。

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

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

int main(int argc, char *argv[]){
        pid_t fpid;
        fpid = fork();
        if(fpid>0){
                //printf("I am the parent of fpid=%d!\n",fpid);
                while(1){
                        sleep(1);
                }
        }
        else if(!fpid){
                //printf("I am the baby!\n");
                int ret;
                ret = execl("/usr/bin/vi","vi","test.txt",NULL);
                if(ret == -1)
                        perror("execl");
                }
        else if(fpid==-1)
                perror("fork");
        return 0;
}
```

- 编译源码为 task2，并执行文件发现成功通过 vi 编辑器打开 test.txt 文件

```
cloudness@cloudness-VirtualBox:~/桌面/操作系统/lab2$ sudo vim task2.c
cloudness@cloudness-VirtualBox:~/桌面/操作系统/lab2$ gcc task2.c -o task2
cloudness@cloudness-VirtualBox:~/桌面/操作系统/lab2$ ./task2
```

```
cloudness@cloudness-VirtualBox: ~/桌面/操作系统/lab2
文件(F)  编辑(E)  查看(V)  搜索(S)  终端(T)  帮助(H)
Test lalalalalalalala~~
~
```

- 在另一个终端通过 ps -Al 、ps-aux 获取所有当前进程的详细信息，得到了 vi 进程详细运行状态。

```
0 S  1000  6112  5826  0  80   0 -  1094 hrtime pts/0    00:00:00 task2
```

- 通过上述得到的ｖｉ进程的ｐｐｉｄ信号追溯逐步获取其父进程如下。

```
cloudness@cloudness-VirtualBox:~$ ps -f -L 6112
UID          PID  PPID   LWP  C NLWP STIME TTY      STAT   TIME CMD
cloudne+    6112  5826  6112  0    1 21:15 pts/0    S+     0:00 ./task2
cloudness@cloudness-VirtualBox:~$ ps -f -L 5826
UID          PID  PPID   LWP  C NLWP STIME TTY      STAT   TIME CMD
cloudne+    5826  2349  5826  0    1 20:57 pts/0    Ss     0:00 bash
cloudness@cloudness-VirtualBox:~$ ps -f -L 2349
UID          PID  PPID   LWP  C NLWP STIME TTY      STAT   TIME CMD
cloudne+    2349  1188  2349  0    4 17:49 ?        Ssl    0:29 /usr/lib/gnome-ter
cloudne+    2349  1188  2350  0    4 17:49 ?        Ssl    0:00 /usr/lib/gnome-ter
cloudne+    2349  1188  2351  0    4 17:49 ?        Ssl    0:02 /usr/lib/gnome-ter
cloudne+    2349  1188  2352  0    4 17:49 ?        Ssl    0:00 /usr/lib/gnome-ter
cloudness@cloudness-VirtualBox:~$ ps -f -L 1188
UID          PID  PPID   LWP  C NLWP STIME TTY      STAT   TIME CMD
cloudne+    1188     1  1188  0    1 17:47 ?        Ss     0:00 /lib/systemd/syste
cloudness@cloudness-VirtualBox:~$ ps -f -L 1
UID          PID  PPID   LWP  C NLWP STIME TTY      STAT   TIME CMD
root           1     0     1  0    1 17:46 ?        Ss     0:09 /sbin/init splash
```

- 重新调用 ps -Al 、ps-aux 命令获取，父进程详细信息

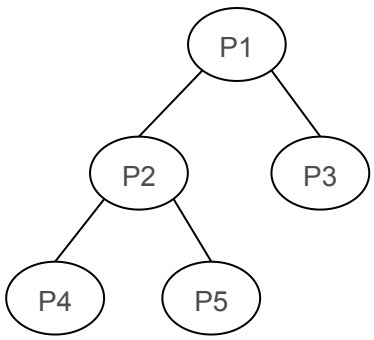
```
0 S  1000  5826  2349  0  80   0 -  6188 wait   pts/0    00:00:00 bash
0 S  1000  5826  2349  0  80   0 -  6188 wait   pts/0    00:00:00 bash
0 R  1000  2349  1188  0  80   0 - 192434 poll_s ?       00:00:27 gnome-terminal
4 S     0     1     0  0  80   0 - 56324 -      ?        00:00:08 systemd
```

使用 ps =aux --sort -pcpu | less 命令，对所有进程按照 cpu 占用率排序。

```
USER        PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
cloudne+   2132 25.9  7.3 1792680 149616 ?      Sl   17:47  57:18 /usr/lib/libreoffice/prog
hangYunxiao__lab2.doc --splash-pipe=5
cloudne+   1444 14.1 12.3 3652600 251444 tty2   Sl+  17:47  31:17 /usr/bin/gnome-shell
cloudne+   6209  5.2  7.7 1961536 158276 tty2   Sl+  21:16   0:36 /usr/lib/firefox/firefox
190215001155 -greomni /usr/lib/firefox/omni.ja -appomni /usr/lib/firefox/browser/omni.ja -
cloudne+   2537  3.6 11.5 2458152 236076 tty2   Sl+  17:53   7:56 /usr/lib/firefox/firefox
cloudne+   1208  2.9 12.6 2062804 257528 tty2   Dl+  17:47   6:32 /usr/lib/xorg/Xorg vt2 -d
root         40  0.6  0.0       0     0 ?        S    17:46   1:27 [kswapd0]
cloudne+   6163  0.6  8.8 3286128 180852 ?      Sl   21:16   0:04 sogou-qimpanel
cloudne+   6344  0.5  2.5 700192 51788 ?        Sl   21:24   0:01 eog /home/cloudness/?片/2
cloudne+   5929  0.4  1.9 1378836 39852 ?       Sl   21:04   0:07 /usr/bin/nautilus --gappl
cloudne+   2349  0.2  1.1 769736 22648 ?        Ssl  17:49   0:32 /usr/lib/gnome-terminal/g
cloudne+   5095  0.2  2.6 1708540 54780 tty2    Sl+  19:52   0:12 /usr/lib/firefox/firefox
90215001155 -greomni /usr/lib/firefox/omni.ja -appomni /usr/lib/firefox/browser/omni.ja -a
cloudne+   1302  0.1  0.0 120412     0 ?        Sl   17:47   0:17 /usr/bin/VBoxClient --dra
cloudne+   2771  0.1  1.5 1590808 32448 tty2    Sl+  17:53   0:20 /usr/lib/firefox/firefox
0215001155 -greomni /usr/lib/firefox/omni.ja -appomni /usr/lib/firefox/browser/omni.ja -ap
root          1  0.0  0.1 225296  2144 ?        Ss   17:46   0:09 /sbin/init splash
root          2  0.0  0.0      0     0 ?        S    17:46   0:00 [kthreadd]
root          4  0.0  0.0      0     0 ?        I<   17:46   0:00 [kworker/0:0H]
```

4 .Task 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 .1 .实验过程

- 编写程序 task3.c，代码如下所示

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

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

int main(int argc, char *argv[]){
        int i=0;
        pid_t ppid = getppid();
        printf("i son/pa ppid pid fpid\n");
        //ppid指当前父进程的pid
        //pid指当前进程的pid
        //fpid指fork返回的当前进程的值
        for (i=0;i<2;i++){
                pid_t fpid = fork();
                if(fpid>0){
                        printf("%d parent %4d %4d %4d\n",i,getppid(),getpid(),fpid);
                        if(i==1 && ppid!=getppid()){
                                fpid =fork();
                                if(fpid>0)
                                        printf("%d parent %4d %4d %4d\n",i,getppid(),getpid(),fpid);
                                else if(!fpid){
                                        //printf("%d son %4d %4d %4d\n",i,getppid(),getpid(),fpid);
                                }
                                else if(fpid==-1)
                                        printf("fork");
                        }
                }
                 else if(!fpid)
                        //printf("%d son %4d %4d %4d\n",i,getppid(),getpid(),fpid);
                 ;
                else if(fpid==-1)
                        perror("fork");
        }
        return 0;
}
```

➢ 编译源码为 task 3 ，并执行文件，结果如下所示

```
cloudness@cloudness-VirtualBox:~/桌面/操作系统/lab2$ vim task3.c
cloudness@cloudness-VirtualBox:~/桌面/操作系统/lab2$ gcc task3.c -o task3
cloudness@cloudness-VirtualBox:~/桌面/操作系统/lab2$ ./task3
i son/pa ppid pid fpid
0 parent 2219 3106 3107
1 parent 2219 3106 3108
1 parent 1173 3107 3109
1 parent 1173 3107 3110
```

❖ 程序解释

第一步：在父进程中，指令执行到 for 循环中，i=0，接着执行 fork，fork 执行完后，系统出现两个进程，分别是 p3106 和 p3107。可以看到父进程是 p2219，子进程 p3107 的父进程正好是 p3106。我们使用一个链表来表示这个关系：p2219->p3106->p3107。

第一次 fork 之后,p3106（父进程）的变量 i=0,fpid=3107,p3107（子进程）的变量 i=0,fpid=0

第二步：假设父进程先执行，当进入下一个循环时,i=1,接着执行 fork,系统新增一个进程 p3108,对于此时的父进程，p2219->p3106（当前进程）->p3108（父进程）

对于子进程 p3107 ,执行完第一次循环后 ,i=1 ,接着执行 fork,系统中又新增一个进程 p3109，对于此进程，p1173->p3107->p3109。从输出可以看出来 p3107 是原来 p3106 的子进程，

现在变成了 p3109 的父进程。

第四步：当父进程执行时，判断 i ==1 并且当前进程的父进程不是 p2219,即当前进程为 p2017 时,执行 fork，系统中又新增一个进程 p3110，对于此进程，p1173->p3107->p3110。

注意：关于最后两个输出的 p3107 的父进程变为 p1173 主要在于在执行完第二虚幻后，main 函数退出，进程死亡，所以 p1173 失去父进程，被重置为 p1173。

❖ 结果分析

调用程序得到进程树，符合题干要求，p1 为 3106,p2 为 3107,p3 为 3108,p4 为 3109,p5 为 3110。

5 .Task 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。

5.1. 基础知识

5.1.1. 终止进程

exit()系统调用：

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

5.1.2. kill 命令

命令格式：kill[参数][进程号]

命令功能：发送指定的信号到相应进程。不指定型号将发送 SIGTERM（15）终止指定进程。如果任无法终止该程序可用“-KILL” 参数，其发送的信号为 SIGKILL(9) ，将强制结束进程，使用 ps 命令或者 jobs 命令可以查看进程号。root 用户将影响用户的进程，非 root 用户只能影响自己的进程。

命令参数：-l 信号，若果不加信号的编号参数，则使用“-l”参数会列出全部的信号名称；-a 当处理当前进程时，不限制命令名和进程号的对应关系；-p 指定 kill 命令只打印相关进程的进程号，而不发送任何信号；-s 指定发送信号；-u 指定用户

常用信号：HUP 1 终端断线；INT 2 中断（同 Ctrl + C）；QUIT 3 退出（同 Ctrl + \）；TERM 15 终止；KILL 9 强制终止；CONT 18 继续（与 STOP 相反， fg/bg 命令）；STOP 19 暂停（同 Ctrl + Z）

5.2. 实验过程

- 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，编写程序为 task4.c，代码如下所示

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

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

int main(int argc, char *argv[]){
        int i=0;
        pid_t ppid = getppid();
        printf("i son/pa ppid pid fpid\n");
        //ppid指当前父进程的pid
        //pid指当前进程的pid
        //fpid指fork返回的当前进程的值
        for (i=0;i<2;i++){
                pid_t fpid = fork();
                if(fpid>0){
                        printf("%d parent %4d %4d %4d\n",i,getppid(),getpid(),fpid);
                        if(i==1 && ppid!=getppid()){
                                fpid =fork();
                                if(fpid>0)
                                        printf("%d parent %4d %4d %4d\n",i,getppid(),getpid(),fpid);
                                else if(!fpid){
                                        //printf("%d son %4d %4d %4d\n",i,getppid(),getpid(),fpid);
                                }
                                else if(fpid==-1)
                                         printf("fork");
                        }
                }
                 else if(!fpid){
                        //printf("%d son %4d %4d %4d\n",i,getppid(),getpid(),fpid);
                 }
                else if(fpid==-1)
                        perror("fork");
        }
        while(1){
                sleep(1);
                printf("mypid is %4d myppid is %4d\n",getpid(),getppid());
        }
        return 0;
}
```

➢ 编并运行上述代码

发现：成功循环输出自己的ＩＤ和父进程ＩＤ

进程	Ｐ１	Ｐ２	Ｐ３	Ｐ４	Ｐ５
进程ＩＤ	５９３３	５９３４	５９３５	５９３６	５９３７

```
cloudness@cloudness-VirtualBox:~/桌面/操作系统/lab2$ vim task4.c
cloudness@cloudness-VirtualBox:~/桌面/操作系统/lab2$ gcc task4.c -o task4
cloudness@cloudness-VirtualBox:~/桌面/操作系统/lab2$ ./task4
i son/pa ppid pid fpid
0 parent 3977 5933 5934
1 parent 3977 5933 5935
1 parent 5933 5934 5936
1 parent 5933 5934 5937
mypid is 5933 myppid is 3977
mypid is 5934 myppid is 5933
mypid is 5936 myppid is 5934
mypid is 5935 myppid is 5933
mypid is 5937 myppid is 5934
mypid is 5933 myppid is 3977
mypid is 5936 myppid is 5934
mypid is 5937 myppid is 5934
mypid is 5934 myppid is 5933
mypid is 5935 myppid is 5933
mypid is 5934 myppid is 5933
mypid is 5935 myppid is 5933
mypid is 5933 myppid is 3977
mypid is 5936 myppid is 5934
mypid is 5937 myppid is 5934
mypid is 5933 myppid is 3977
mypid is 5934 myppid is 5933
mypid is 5936 myppid is 5934
```

➢ 调用 kill -9 5934 命令，终止ｐ 2 进程，p1,p3,p4,p5 进程运行如下所示

```
mypid is 5936 myppid is 1193
mypid is 5933 myppid is 3977
mypid is 5937 myppid is 1193
mypid is 5935 myppid is 5933
mypid is 5933 myppid is 3977
mypid is 5937 myppid is 1193
mypid is 5936 myppid is 1193
mypid is 5935 myppid is 5933
mypid is 5937 myppid is 1193
mypid is 5933 myppid is 3977
mypid is 5936 myppid is 1193
mypid is 5935 myppid is 5933
mypid is 5933 myppid is 3977
mypid is 5937 myppid is 1193
```

- 输入 ps -Al、ps -aux 观察相关进程参数变化如下所示。

```
F S   UID   PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
0 S  1000  5933  3977  0  80   0 -  1127 hrtime pts/0    00:00:00 task4
1 Z  1000  5934  5933  0  80   0 -     0 -      pts/0    00:00:00 task <defunct>
1 S  1000  5935  5933  0  80   0 -  1127 hrtime pts/0    00:00:00 task4
1 S  1000  5936  1193  0  80   0 -  1127 hrtime pts/0    00:00:00 task4
1 S  1000  5937  1193  0  80   0 -  1127 hrtime pts/0    00:00:00 task4
```

```
USER        PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
cloudne+   5933  0.0  0.0   4508   796 pts/0    S+   11:42   0:00 ./task4
cloudne+   5934  0.0  0.0      0     0 pts/0    Z+   11:42   0:00 [task4] <defunc
cloudne+   5935  0.0  0.0   4508    80 pts/0    S+   11:42   0:00 ./task4
cloudne+   5936  0.0  0.0   4508    80 pts/0    S+   11:42   0:00 ./task4
cloudne+   5937  0.0  0.0   4508    80 pts/0    S+   11:42   0:00 ./task4
```

- 修改 task4.c 程序为 task_exit.c，使用 exit()退出 p2 进程，代码如下所示。

程序解释：在上一个生成进程树，并使得所有进程都循环输出自己的ＩＤ和父进程的ＩＤ的基础上，我们增加了一个判断，当 i>5&&p1==(getpid()-1)时，即要求循环输出五个进程至少五次，并要求当前进程为 p2 时，调用 exit(0)退出进程。

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

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

int main(int argc, char *argv[]){
        int i=0;
        pid_t ppid = getppid();
        pid_t p1=getpid();
        printf("i son/pa ppid pid fpid\n");
        //ppid指当前父进程的pid
        //pid指当前进程的pid
        //fpid指fork返回的当前进程的值
        for (i=0;i<2;i++){
                pid_t fpid = fork();
                if(fpid>0){
                        printf("%d parent %4d %4d %4d\n",i,getppid(),getpid(),fpid);
                        if(i==1 && ppid!=getppid()){
                                fpid =fork();
                                if(fpid>0)
                                        printf("%d parent %4d %4d %4d\n",i,getppid(),getpid(),fpid);
                                else if(!fpid){
                                        //printf("%d son %4d %4d %4d\n",i,getppid(),getpid(),fpid);
                                }
                                else if(fpid==-1)
                                         printf("fork");
                        }
                }
                 else if(!fpid){
                        //printf("%d son %4d %4d %4d\n",i,getppid(),getpid(),fpid);
                 }
                else if(fpid==-1)
                        perror("fork");
        }
        i=0;
        while(1){
                sleep(1);
                printf("mypid is %4d myppid is %4d\n",getpid(),getppid());
                i++;
                if(i>5 && (p1 == (getpid()-1)))
                        exit(0);
        }
        return 0;
"task4_exit.c" [只读] 47L, 1582C
```

➢ 编译程序为 task4_exit，并运行，结果如下。

```
cloudness@cloudness-VirtualBox:~/桌面/操作系统/lab2$ vim task4_exit.c
cloudness@cloudness-VirtualBox:~/桌面/操作系统/lab2$ gcc task4_exit.c -o task4_exit
cloudness@cloudness-VirtualBox:~/桌面/操作系统/lab2$ ./task4_exit
i son/pa ppid pid fpid
0 parent 2152 11956 11957
1 parent 2152 11956 11958
1 parent 11956 11957 11959
1 parent 11956 11957 11960
mypid is 11956 myppid is 2152
mypid is 11957 myppid is 11956
mypid is 11959 myppid is 11957
mypid is 11960 myppid is 11957
mypid is 11958 myppid is 11956
```

```
mypid is 11959 myppid is 1178
mypid is 11958 myppid is 11956
mypid is 11960 myppid is 1178
mypid is 11956 myppid is 2152
mypid is 11959 myppid is 1178
mypid is 11958 myppid is 11956
mypid is 11960 myppid is 1178
mypid is 11956 myppid is 2152
```

➢ 输入 ps -Al、ps -aux 观察相关进程参数变化如下所示

```
F S   UID   PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
```

```
0 S  1000 11956  2152  0  80   0 -  1127 hrtime pts/0    00:00:00 task4_exit
1 Z  1000 11957 11956  0  80   0 -     0 -      pts/0    00:00:00 task4_exit <defunct>
1 S  1000 11958 11956  0  80   0 -  1127 hrtime pts/0    00:00:00 task4_exit
1 S  1000 11959  1178  0  80   0 -  1127 hrtime pts/0    00:00:00 task4_exit
1 S  1000 11960  1178  0  80   0 -  1127 hrtime pts/0    00:00:00 task4_exit
```

```
USER        PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
cloudne+ 11956  0.0  0.0   4508   716 pts/0    S+   14:37   0:00 ./task4_exit
cloudne+ 11957  0.0  0.0      0     0 pts/0    Z+   14:37   0:00 [task4_exit] <defunct>
cloudne+ 11958  0.0  0.0   4508    80 pts/0    S+   14:37   0:00 ./task4_exit
cloudne+ 11959  0.0  0.0   4508    80 pts/0    S+   14:37   0:00 ./task4_exit
cloudne+ 11960  0.0  0.0   4508    80 pts/0    S+   14:37   0:00 ./task4_exit
```

➢ 修改 task4.c 程序为 task_Sefault.c，使用段错误退出 p2 进程，代码如下所示。

程序解释：在上一个生成进程树，并使得所有进程都循环输出自己的ＩＤ和父进程的ＩＤ的基础上，我们增加了一个判断，当 i>5&&p1==(getpid()-1)时，即要求循环输出五个进程至少五次，并要求当前进程为 p2 时，创建一个段错误，此处，我创建了一个一个指针，尝试操作访问地址为 0 的内存区域，而这个内存区域通常时不可访问的禁区，所以出现了段错误。

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

int main(int argc, char *argv[])
{
        int i=0;
        pid_t ppid = getppid();
        pid_t p1=getpid();
        printf("i son/pa ppid pid fpid\n");
        //ppid指当前父进程的pid
        //pid指当前进程的pid
        //fpid指fork返回的当前进程的值
        for (i=0;i<2;i++){
                pid_t fpid = fork();
                if(fpid>0){
                        printf("%d parent %4d %4d %4d\n",i,getppid(),getpid(),fpid);
                        if(i==1 && ppid!=getppid()){
                                fpid =fork();
                                if(fpid>0)
                                        printf("%d parent %4d %4d %4d\n",i,getppid(),getpid(),fpid);
                                else if(!fpid){
                                        //printf("%d son %4d %4d %4d\n",i,getppid(),getpid(),fpid);
                                }
                                else if(fpid==-1)
                                        printf("fork");
                        }
                }
                 else if(!fpid){
                        //printf("%d son %4d %4d %4d\n",i,getppid(),getpid(),fpid);
                 }
                else if(fpid==-1)
                        perror("fork");
        }
        i=0;
        while(1){
                sleep(1);
                printf("mypid is %4d myppid is %4d\n",getpid(),getppid());
                i++;
                if(i>5 && (p1 == (getpid()-1))){
                //制造一个段错误,访问地址为 0 的内存区域
                unsigned char* pstr = 0x00;
                *pstr = 0x00;
                }
        }
        return 0;
}
```

➢ 编译程序为 task4_SeFault，并运行，结果如下。

```
cloudness@cloudness-VirtualBox:~/桌面/操作系统/lab2$ vim task4_SeFaults.c
cloudness@cloudness-VirtualBox:~/桌面/操作系统/lab2$ gcc task4_SeFaults.c -o task4_SeFaults
cloudness@cloudness-VirtualBox:~/桌面/操作系统/lab2$ ./task4_SeFaults
i son/pa ppid pid fpid
0 parent 2152 12884 12885
1 parent 2152 12884 12886
1 parent 12884 12885 12887
1 parent 12884 12885 12888
mypid is 12884 myppid is 2152
mypid is 12887 myppid is 12885
mypid is 12888 myppid is 12885
mypid is 12885 myppid is 12884
mypid is 12886 myppid is 12884
```

```
mypid is 12888 myppid is 1178
mypid is 12886 myppid is 12884
mypid is 12884 myppid is 2152
mypid is 12887 myppid is 1178
mypid is 12888 myppid is 1178
mypid is 12886 myppid is 12884
mypid is 12884 myppid is 2152
mypid is 12887 myppid is 1178
```

➢ 输入 ps -Al、ps -aux 观察相关进程参数变化如下所示

```
F S   UID   PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
0 S  1000 12884  2152  0  80   0 -  1127 hrtime pts/0    00:00:00 task4_SeFaults
1 Z  1000 12885 12884  0  80   0 -     0 -      pts/0    00:00:00 task4_SeFaults <defunct>
1 S  1000 12886 12884  0  80   0 -  1127 hrtime pts/0    00:00:00 task4_SeFaults
1 S  1000 12887  1178  0  80   0 -  1127 hrtime pts/0    00:00:00 task4_SeFaults
1 S  1000 12888  1178  0  80   0 -  1127 hrtime pts/0    00:00:00 task4_SeFaults
```

```
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
cloudne+ 12884  0.0  0.0   4508   732 pts/0    S+   14:52   0:00 ./task4_SeFaults
cloudne+ 12885  0.0  0.0      0     0 pts/0    Z+   14:52   0:00 [task4_SeFaults] <defunct>
cloudne+ 12886  0.0  0.0   4508    80 pts/0    S+   14:52   0:00 ./task4_SeFaults
cloudne+ 12887  0.0  0.0   4508    80 pts/0    S+   14:52   0:00 ./task4_SeFaults
cloudne+ 12888  0.0  0.0   4508    80 pts/0    S+   14:52   0:00 ./task4_SeFaults
```

5.3. 实验结论

通过实验我们发现，分别采用 kill -9 、自己正常退出 exit()、段错误退出均可实现 p2 进程退出的功能，且 p2 进程退出后，发现 p1.p2 的循环运行状态正常没有变化，p4，p5 的父进程由于 p2 进程退出重新挂载到了父进程 systemd 上。

另外发现每一次重新启动计算机后，父进程 systemd 的进程号都不同，这是要因为进程号只是一个标示，每次重新计算机后，都相当于一次重新启动 systemd 进程，进程号重新分配。

关于父进程关闭后，子进程不是直接挂载到 init 上的情况，是因为与虚拟机的版本相关，本机是配有 systemd 的虚拟机。而又由于 Systemd 是 Linux 系统工具，用来启动守护进程，已成为大多数发行版的标准配置。使用了 Systemd，就不需要再用 init 了。Systemd 取代了 initd，成为系统的第一个进程（PID 等于 1），其他进程都是它的子进程。

同时，我们还发现 p2 结束后，p2 的状态除却 SZ,WCHAN,VSZ,RSS 变为 0（即占用内存为 0 ，运行状态停止）之外，S（STATE）变为 Z ，TASK_DEAD - EXIT_ZOMBIE，表示退出状态，进程成为僵尸进程。

进程在退出的过程中，处于 TASK_DEAD 状态。在这个退出过程中，进程占有的所有资源将被回收，除了 task_struct 结构（以及少数资源）以外。于是进程就只剩下 task_struct 这

么个空壳，故称为僵尸。之所以保留 task_struct，是因为 task_struct 里面保存了进程的退出码、以及一些统计信息。而其父进程很可能会关心这些信息。比如在 shell 中，$?变量就保存了最后一个退出的前台进程的退出码，而这个退出码往往被作为 if 语句的判断条件。

当然，内核也可以将这些信息保存在别的地方，而将 task_struct 结构释放掉，以节省一些空间。但是使用 task_struct 结构更为方便，因为在内核中已经建立了从 pid 到 task_struct 查找关系，还有进程间的父子关系。释放掉 task_struct，则需要建立一些新的数据结构，以便让父进程找到它的子进程的退出信息。

父进程可以通过 wait 系列的系统调用（如 wait4、waitid）来等待某个或某些子进程的退出，并获取它的退出信息。然后 wait 系列的系统调用会顺便将子进程的尸体（task_struct）也释放掉。

子进程在退出的过程中，内核会给其父进程发送一个信号，通知父进程来“收尸”。这个信号默认是 SIGCHLD，但是在通过 clone 系统调用创建子进程时，可以设置这个信号。