实验一 进程控制

安全 1601 李婧祎 16281134

1. 实验目的：

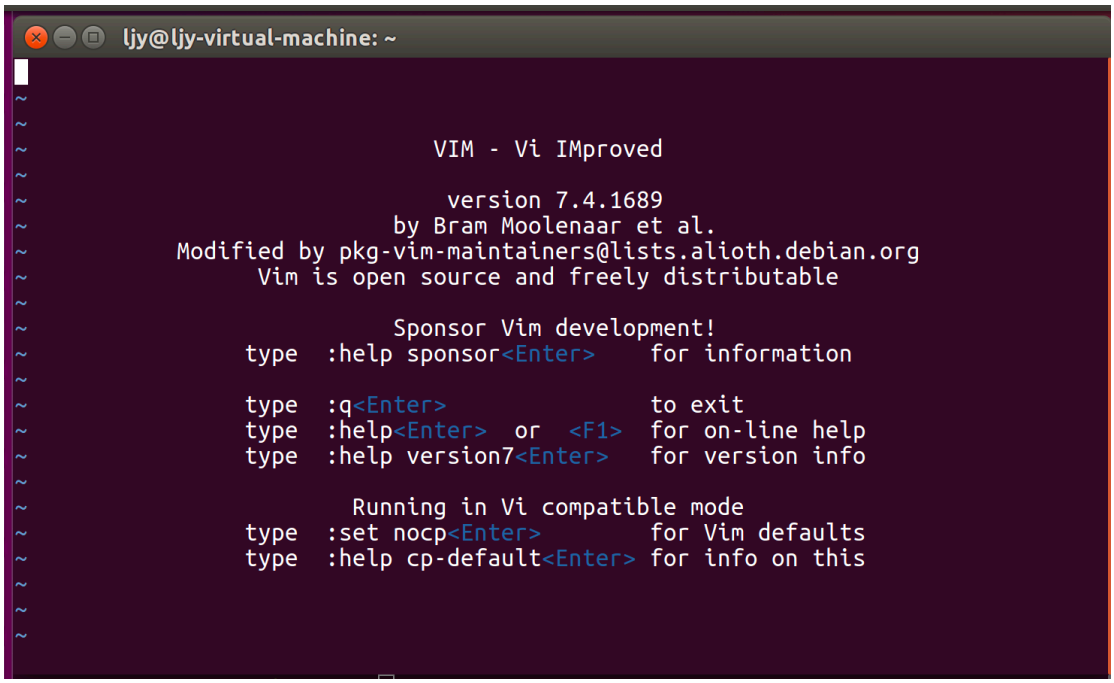
- 加深对进程概念的理解，明确进程和程序的区别。
- 掌握 Linux 系统中的进程创建，管理和删除等操作。
- 熟悉使用 Linux 下的命令和工具，如 man, find, grep, whereis, ps, pgrep, kill, ptree, top, vim, gcc，gdb，管道|等。

2. 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。

（1） 在终端输入 vi 打开 vi 进程



（2） 打开另一终端使用 ps 命令

使用 ps -A 命令查看进程 id，可得知 vi 命令的进程 id 为 3652

```
ljy@ljy-virtual-machine:~$ ps -A | grep vi
  1520 ?        00:00:00 VGAuthService
  3191 ?        00:00:00 hud-service
  3218 ?        00:00:00 dconf-service
  3652 pts/1    00:00:00 vi
```

（3） ps -p3652,查看 CPU 的运行时间

```
ljy@ljy-virtual-machine:~$ ps -p3652
   PID TTY          TIME CMD
  3652 pts/1    00:00:00 vi
```

（4） ps -lax | grep 3652 查看进程号为 3652 的进程的信息

uid ppid pri NI VSZ RSS STAT TTY TIME

```
ljy@ljy-virtual-machine:~$ ps -lax | grep 3652
0  1000   3652   3618  20   0  39112  2616 poll_s S+   pts/1      0:00 vi
0  1000   3809   3637  20   0  21316  1032 pipe_w S+   pts/11     0:00 grep --color=auto 3652
```

UID 是用户号, PID 就是 vi 进程的进程号,PPID 是 vi 进程的父进程号,PRI 是内核调度优先级, NI 是进程优先级,VSZ 是总虚拟内存大小,RSS 是进程使用的总物理内存数, STAT 是进程状态,TTY 是终端的次要装置号码; TIME 为使用 cpu 的时间。

（5） 寻找父进程

```
0  1000   3652   3618  20   0  39112  2616 poll_s S+   pts/1      0:00 vi
```

```
ljy@ljy-virtual-machine:~$ ps -lax | grep 3618
0  1000   3618   3613  20   0  29592  4508 wait   Ss   pts/1      0:00 bash
```

```
ljy@ljy-virtual-machine:~$ ps -lax | grep 3613
0  1000   3613   2928  20   0 705844 33580 poll_s Sl   ?          0:00 /usr/lib/gnome-terminal/g
nome-terminal-server
```

```
ljy@ljy-virtual-machine:~$ ps -lax | grep 2928
4  1000   2928   2810  20   0  53724  2168 poll_s Ss   ?          0:00 /sbin/upstart --user
```

```
ljy@ljy-virtual-machine:~$ ps -lax | grep 2810
4     0   2810    870  20   0 230300  2128 -      Sl   ?          0:00 lightdm --session-child 1
1 16
```

```
ljy@ljy-virtual-machine:~$ ps -lax | grep 870
4     0    686      1  20   0 298704  1900 -      Ssl  ?          0:00 /usr/lib/accountsservice/
accounts-daemon
4     0    870      1  20   0 292196  2276 -      SLsl ?          0:00 /usr/sbin/lightdm
```

3652->3618->3613->2928->2810->870->1

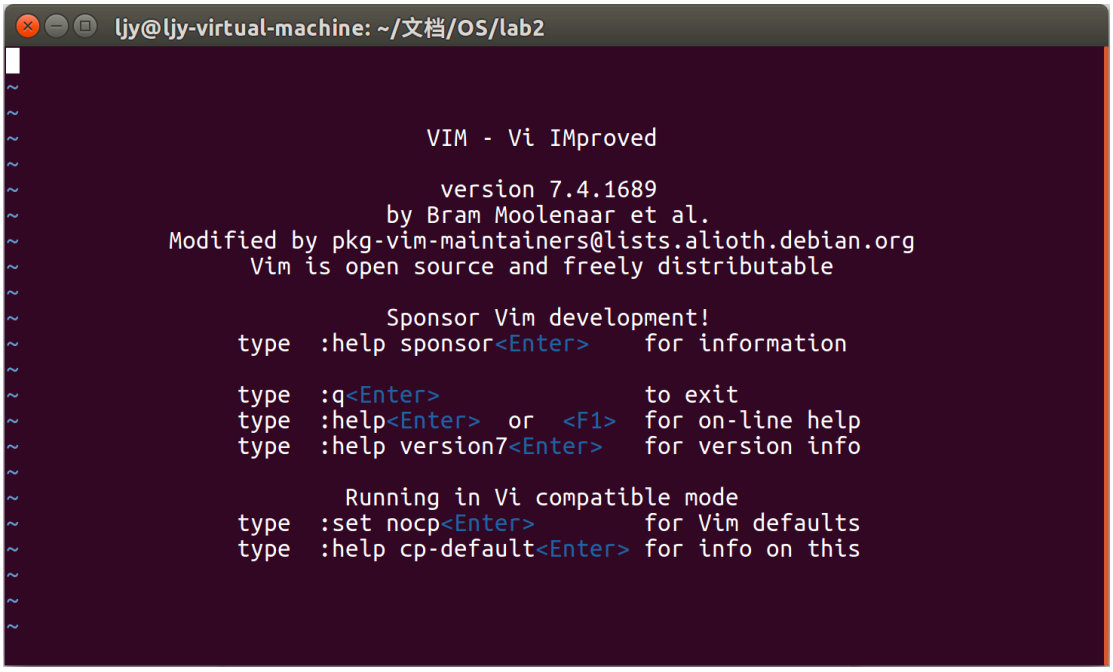
（6） 使用 pstree -p

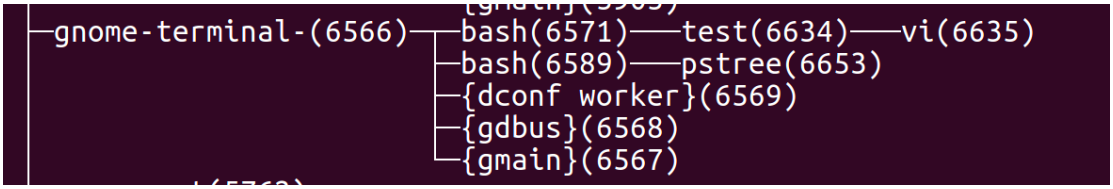
```
├─gnome-terminal-(3613)─┬─bash(3618)───vi(3652)
│                       ├─bash(3637)───pstree(3884)
│                       ├─{dconf worker}(3616)
│                       ├─{gdbus}(3615)
│                       └─{gmain}(3614)
```

- 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。

```
#include <unistd.h>
#include <stdio.h>
 int main (){
     pid_t fpid;
     int count=0;
     fpid=fork();
     if (fpid < 0)
         printf("error in fork!");
     else if (fpid == 0) {
         execl("/usr/bin/vi","vi",NULL);
     }
     else { for(;;){}
     }
     return 0;
 }
```

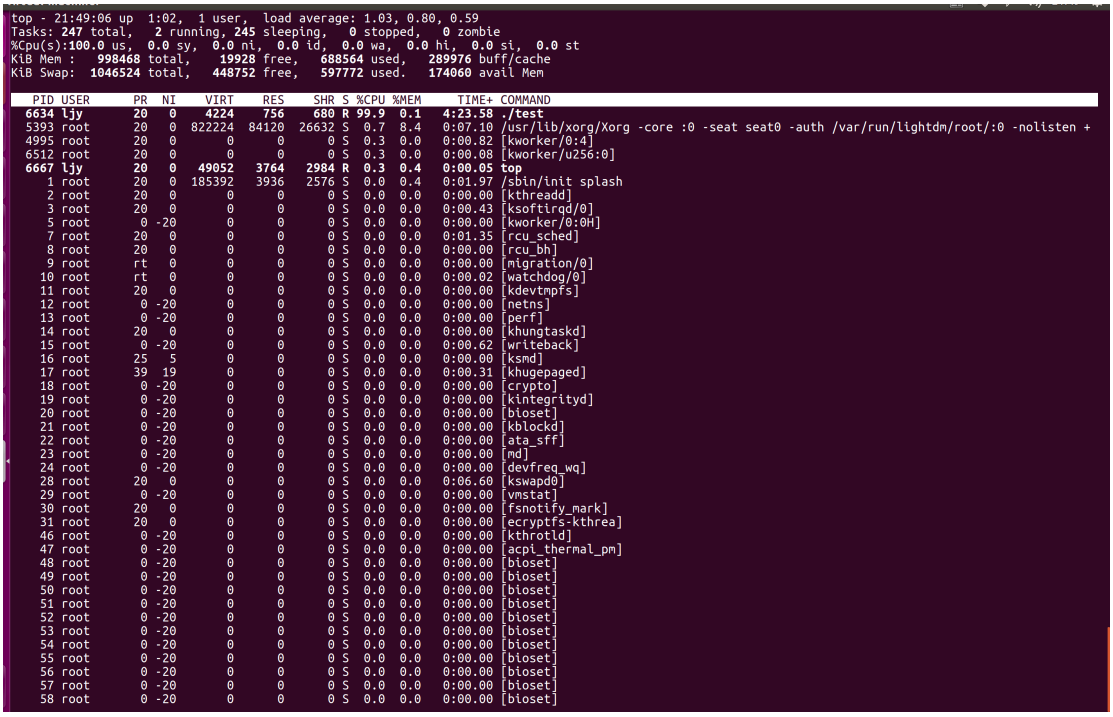
运行以上程序：





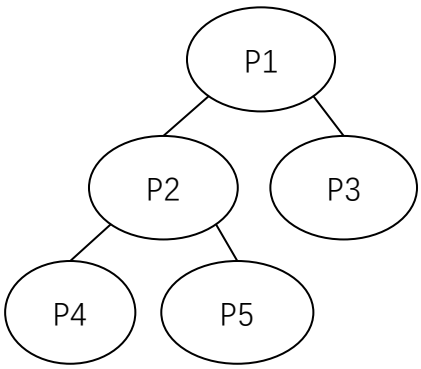
可以看到第一行 vi 是 1 的子进程

使用 top;c 按 CPU 使用排序:



- 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。



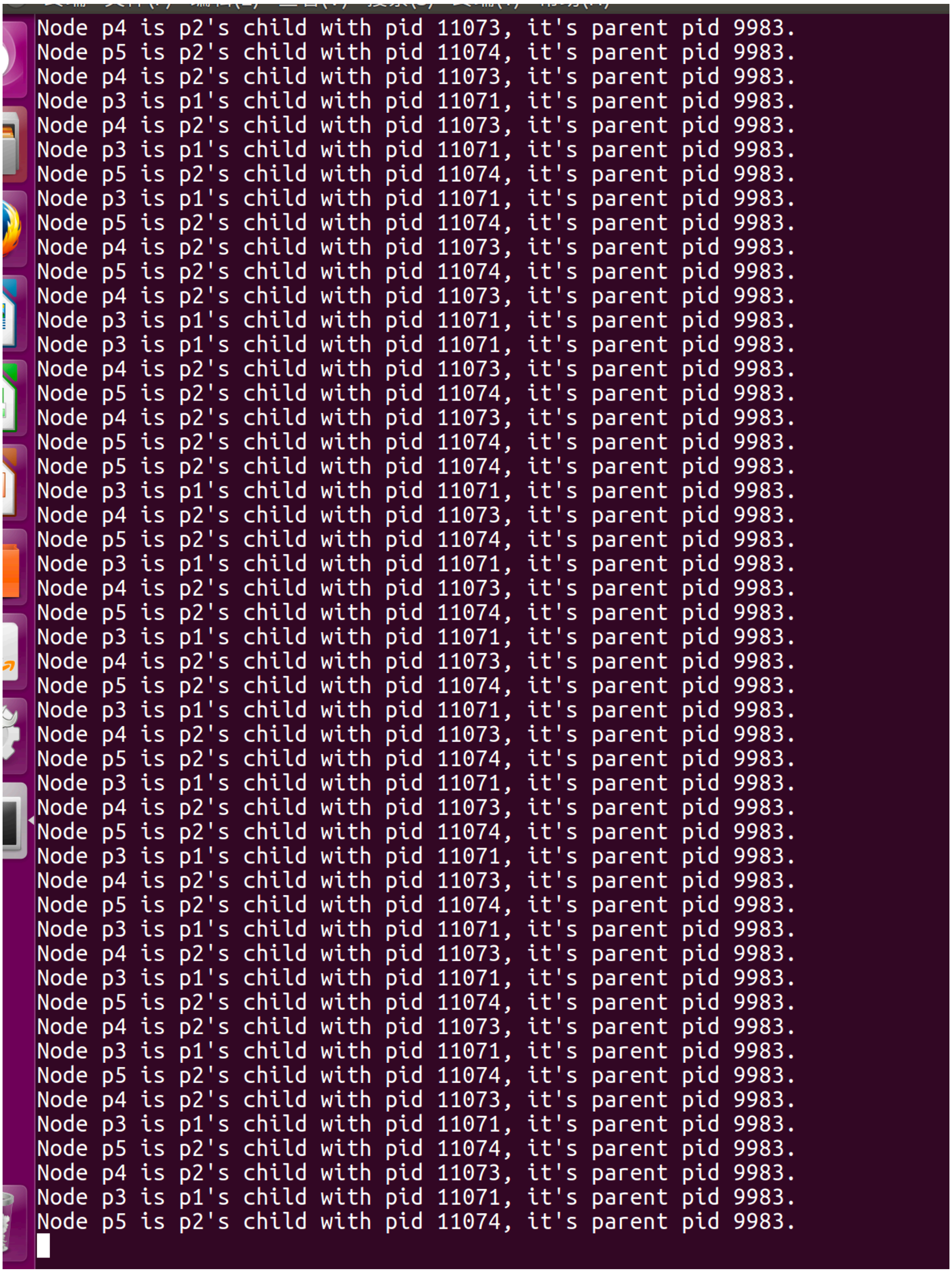

```
Node p5 is p2's child with pid 10816, it's parent pid 10814.
Node p3 is p1's child with pid 10813, it's parent pid 9983.
Node p4 is p2's child with pid 10815, it's parent pid 10814.
Node p5 is p2's child with pid 10816, it's parent pid 10814.
Node p3 is p1's child with pid 10813, it's parent pid 9983.
Node p2 is p1's child with pid 10814, it's parent pid 9983.
Node p4 is p2's child with pid 10815, it's parent pid 10814.
Node p2 is p1's child with pid 10814, it's parent pid 9983.
Node p3 is p1's child with pid 10813, it's parent pid 9983.
Node p5 is p2's child with pid 10816, it's parent pid 10814.
Node p4 is p2's child with pid 10815, it's parent pid 10814.
Node p3 is p1's child with pid 10813, it's parent pid 9983.
Node p5 is p2's child with pid 10816, it's parent pid 10814.
Node p2 is p1's child with pid 10814, it's parent pid 9983.
Node p4 is p2's child with pid 10815, it's parent pid 10814.
Node p5 is p2's child with pid 10816, it's parent pid 10814.
Node p2 is p1's child with pid 10814, it's parent pid 9983.
Node p3 is p1's child with pid 10813, it's parent pid 9983.
Node p4 is p2's child with pid 10815, it's parent pid 10814.
Node p2 is p1's child with pid 10814, it's parent pid 9983.
Node p3 is p1's child with pid 10813, it's parent pid 9983.
Node p5 is p2's child with pid 10816, it's parent pid 10814.
Node p4 is p2's child with pid 10815, it's parent pid 10814.
```

```
├─test2
├─test2───2*[test2]
```

程序名为 test2，可以看到进程树呈上图相同结构。

- 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。

修改程序，使 p2 exit(0):



```
Node p4 is p2's child with pid 11073, it's parent pid 9983.
Node p5 is p2's child with pid 11074, it's parent pid 9983.
Node p4 is p2's child with pid 11073, it's parent pid 9983.
Node p3 is p1's child with pid 11071, it's parent pid 9983.
Node p4 is p2's child with pid 11073, it's parent pid 9983.
Node p3 is p1's child with pid 11071, it's parent pid 9983.
Node p5 is p2's child with pid 11074, it's parent pid 9983.
Node p3 is p1's child with pid 11071, it's parent pid 9983.
Node p5 is p2's child with pid 11074, it's parent pid 9983.
Node p4 is p2's child with pid 11073, it's parent pid 9983.
Node p5 is p2's child with pid 11074, it's parent pid 9983.
Node p4 is p2's child with pid 11073, it's parent pid 9983.
Node p3 is p1's child with pid 11071, it's parent pid 9983.
Node p3 is p1's child with pid 11071, it's parent pid 9983.
Node p4 is p2's child with pid 11073, it's parent pid 9983.
Node p5 is p2's child with pid 11074, it's parent pid 9983.
Node p4 is p2's child with pid 11073, it's parent pid 9983.
Node p5 is p2's child with pid 11074, it's parent pid 9983.
Node p5 is p2's child with pid 11074, it's parent pid 9983.
Node p3 is p1's child with pid 11071, it's parent pid 9983.
Node p4 is p2's child with pid 11073, it's parent pid 9983.
Node p5 is p2's child with pid 11074, it's parent pid 9983.
Node p3 is p1's child with pid 11071, it's parent pid 9983.
Node p4 is p2's child with pid 11073, it's parent pid 9983.
Node p5 is p2's child with pid 11074, it's parent pid 9983.
Node p3 is p1's child with pid 11071, it's parent pid 9983.
Node p4 is p2's child with pid 11073, it's parent pid 9983.
Node p5 is p2's child with pid 11074, it's parent pid 9983.
Node p3 is p1's child with pid 11071, it's parent pid 9983.
Node p4 is p2's child with pid 11073, it's parent pid 9983.
Node p5 is p2's child with pid 11074, it's parent pid 9983.
Node p3 is p1's child with pid 11071, it's parent pid 9983.
Node p4 is p2's child with pid 11073, it's parent pid 9983.
Node p5 is p2's child with pid 11074, it's parent pid 9983.
Node p3 is p1's child with pid 11071, it's parent pid 9983.
Node p4 is p2's child with pid 11073, it's parent pid 9983.
Node p5 is p2's child with pid 11074, it's parent pid 9983.
Node p3 is p1's child with pid 11071, it's parent pid 9983.
Node p4 is p2's child with pid 11073, it's parent pid 9983.
Node p3 is p1's child with pid 11071, it's parent pid 9983.
Node p5 is p2's child with pid 11074, it's parent pid 9983.
Node p4 is p2's child with pid 11073, it's parent pid 9983.
Node p3 is p1's child with pid 11071, it's parent pid 9983.
Node p5 is p2's child with pid 11074, it's parent pid 9983.
Node p4 is p2's child with pid 11073, it's parent pid 9983.
Node p3 is p1's child with pid 11071, it's parent pid 9983.
Node p5 is p2's child with pid 11074, it's parent pid 9983.
Node p4 is p2's child with pid 11073, it's parent pid 9983.
Node p3 is p1's child with pid 11071, it's parent pid 9983.
Node p5 is p2's child with pid 11074, it's parent pid 9983.
```

Kill -9 p2:

子进 p4,p5 继续运行，父进程变为 p1，p3 与其无关，继续运行。

```
Node p4 is p2's child with pid 11073, it's parent pid 9983.
Node p3 is p1's child with pid 11071, it's parent pid 9983.
Node p4 is p2's child with pid 11073, it's parent pid 9983.
Node p4 is p2's child with pid 11073, it's parent pid 9983.
Node p3 is p1's child with pid 11071, it's parent pid 9983.
Node p3 is p1's child with pid 11071, it's parent pid 9983.
Node p4 is p2's child with pid 11073, it's parent pid 9983.
Node p3 is p1's child with pid 11071, it's parent pid 9983.
Node p4 is p2's child with pid 11073, it's parent pid 9983.
Node p3 is p1's child with pid 11071, it's parent pid 9983.
Node p4 is p2's child with pid 11073, it's parent pid 9983.
Node p4 is p2's child with pid 11073, it's parent pid 9983.
Node p3 is p1's child with pid 11071, it's parent pid 9983.
Node p3 is p1's child with pid 11071, it's parent pid 9983.
Node p4 is p2's child with pid 11073, it's parent pid 9983.
Node p3 is p1's child with pid 11071, it's parent pid 9983.
Node p4 is p2's child with pid 11073, it's parent pid 9983.
Node p3 is p1's child with pid 11071, it's parent pid 9983.
Node p4 is p2's child with pid 11073, it's parent pid 9983.
Node p4 is p2's child with pid 11073, it's parent pid 9983.
Node p3 is p1's child with pid 11071, it's parent pid 9983.
Node p4 is p2's child with pid 11073, it's parent pid 9983.
Node p3 is p1's child with pid 11071, it's parent pid 9983.
```