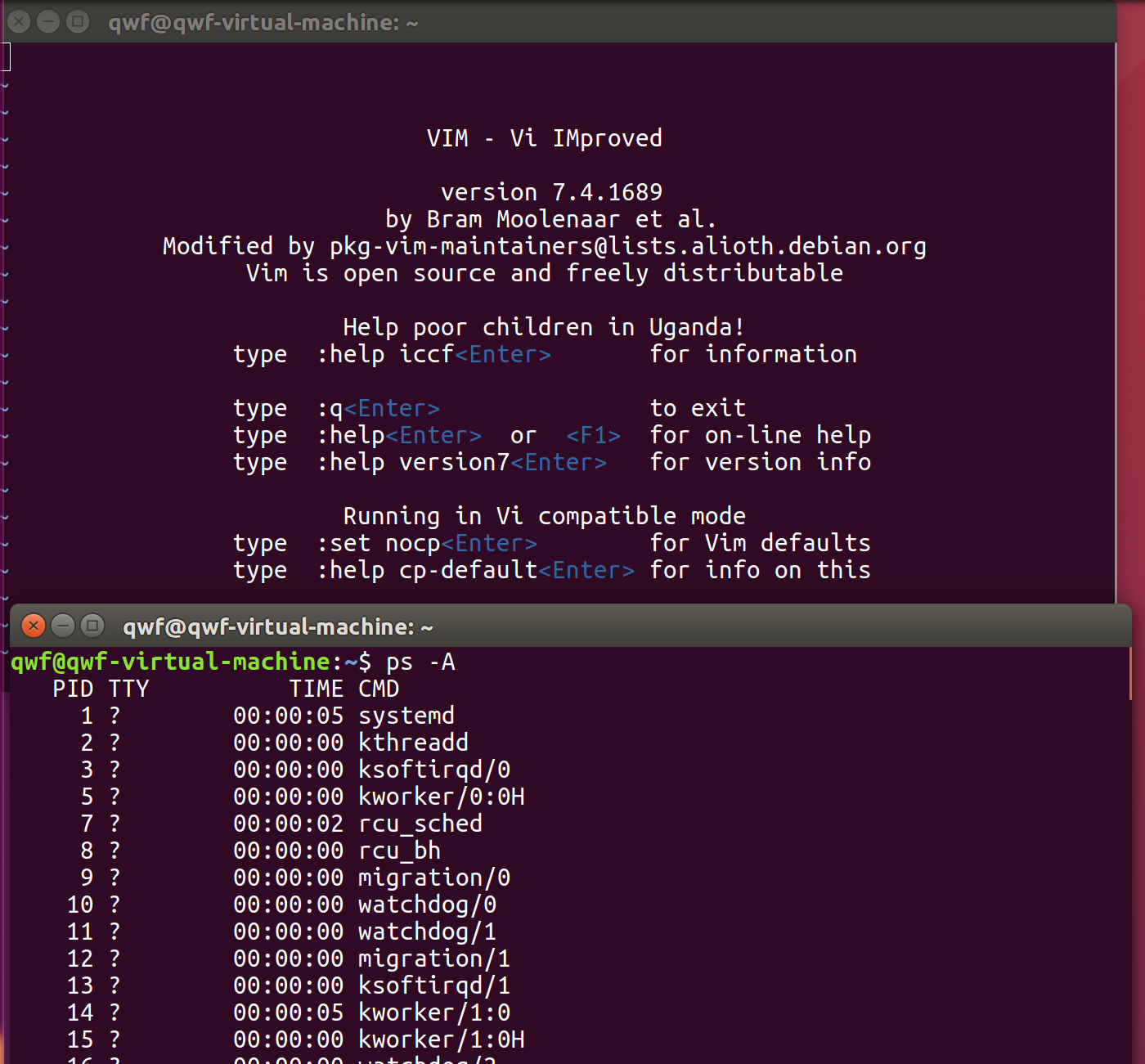
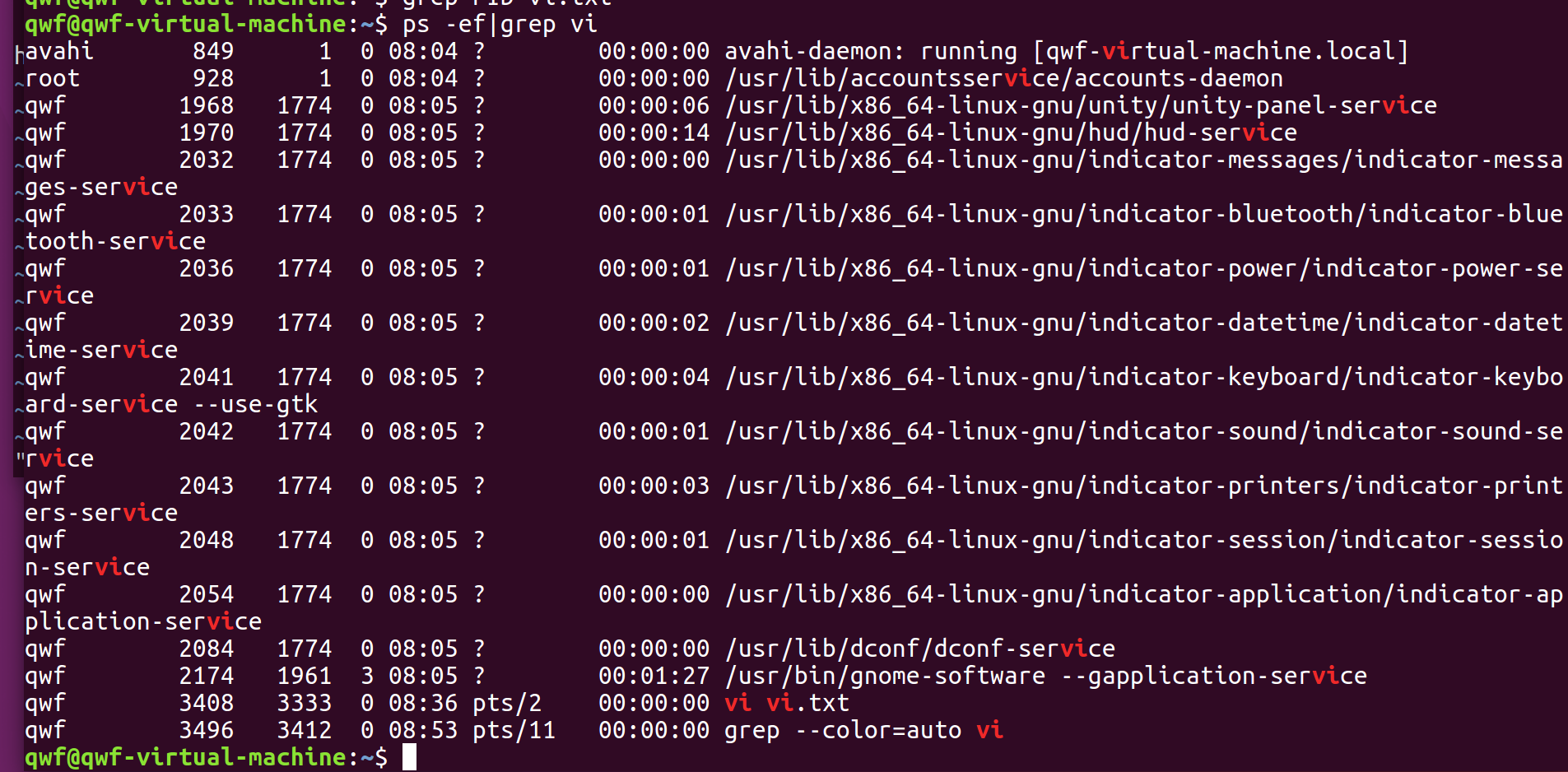
**实验二: 进程控制**

**安全1601 16281221 邓子轩**

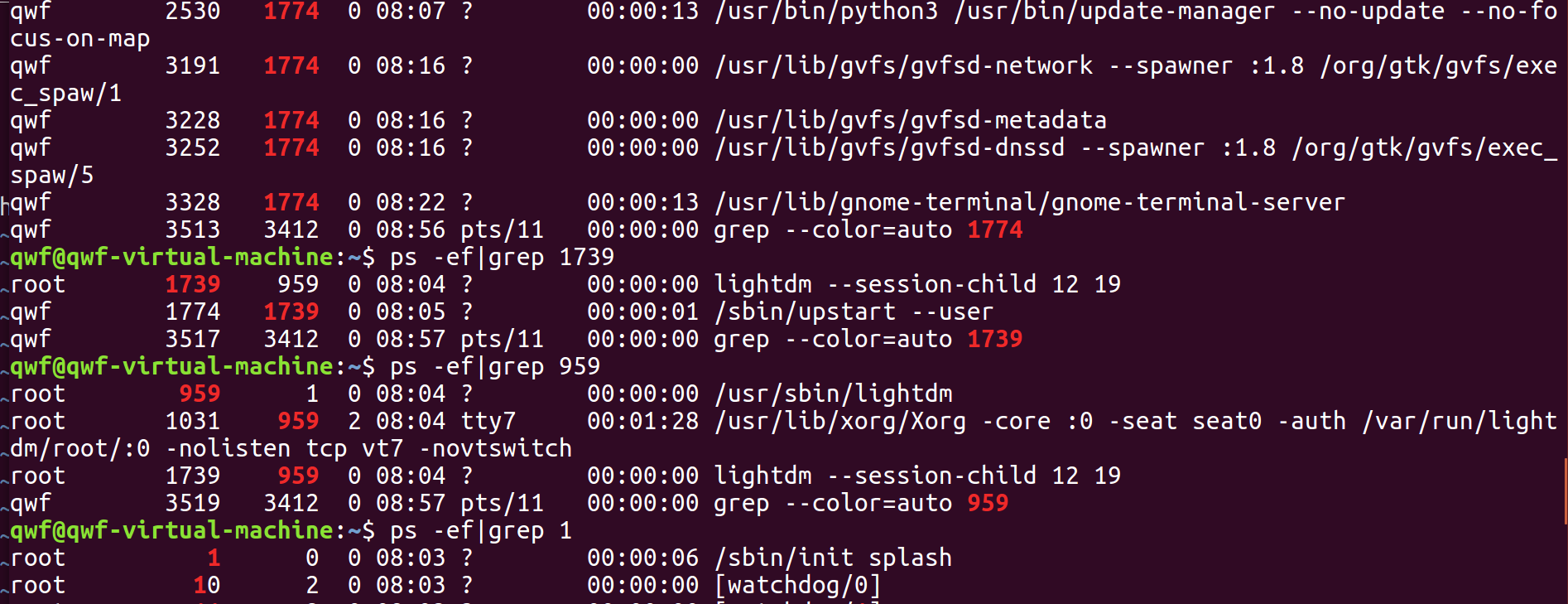
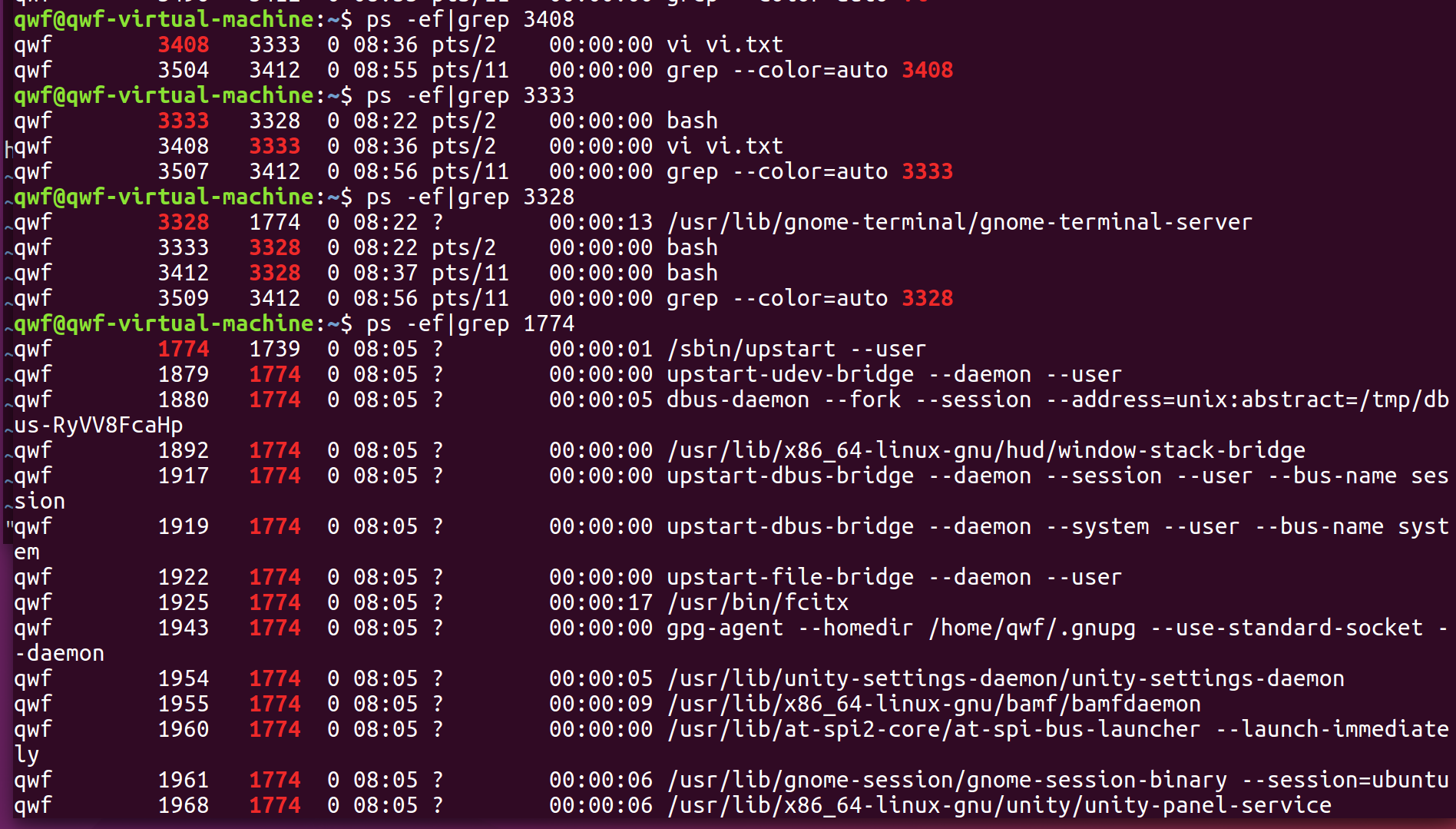
**1、打开一个vi进程。通过ps命令以及选择合适的参数，只显示名字为vi的进程。寻找vi进程的父进程，直到init进程为止。记录过程中所有进程的ID和父进程ID。将得到的进程树和由pstree命令的得到的进程树进行比较。**



使用ps –ef|gref vi来查看带vi的进程

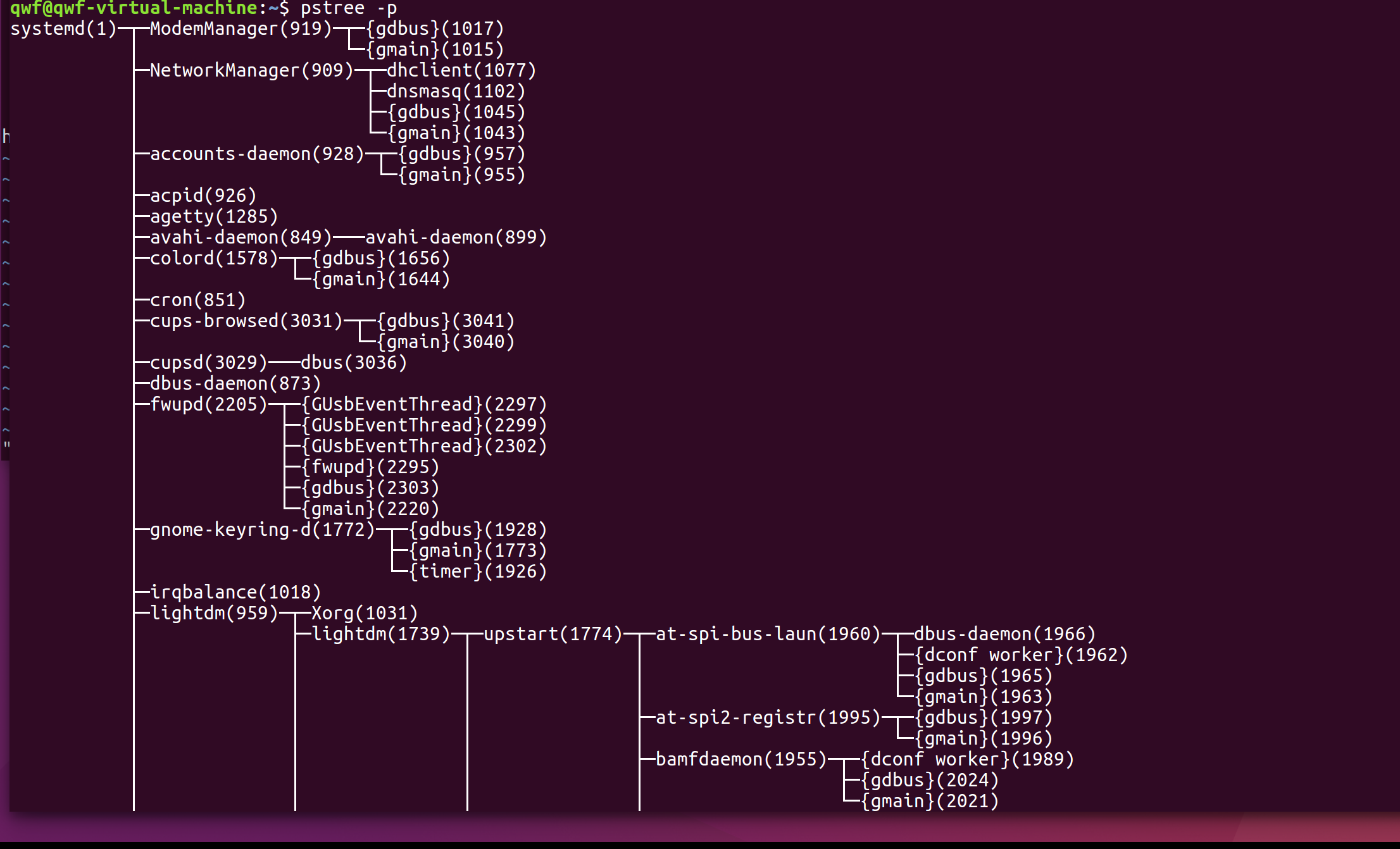


然后继续使用ps –ef|grep PID来寻找vi的父进程



可以看到搜索父进程顺序为：3408——>3333——>3328——>1774——>1739——>959——>1

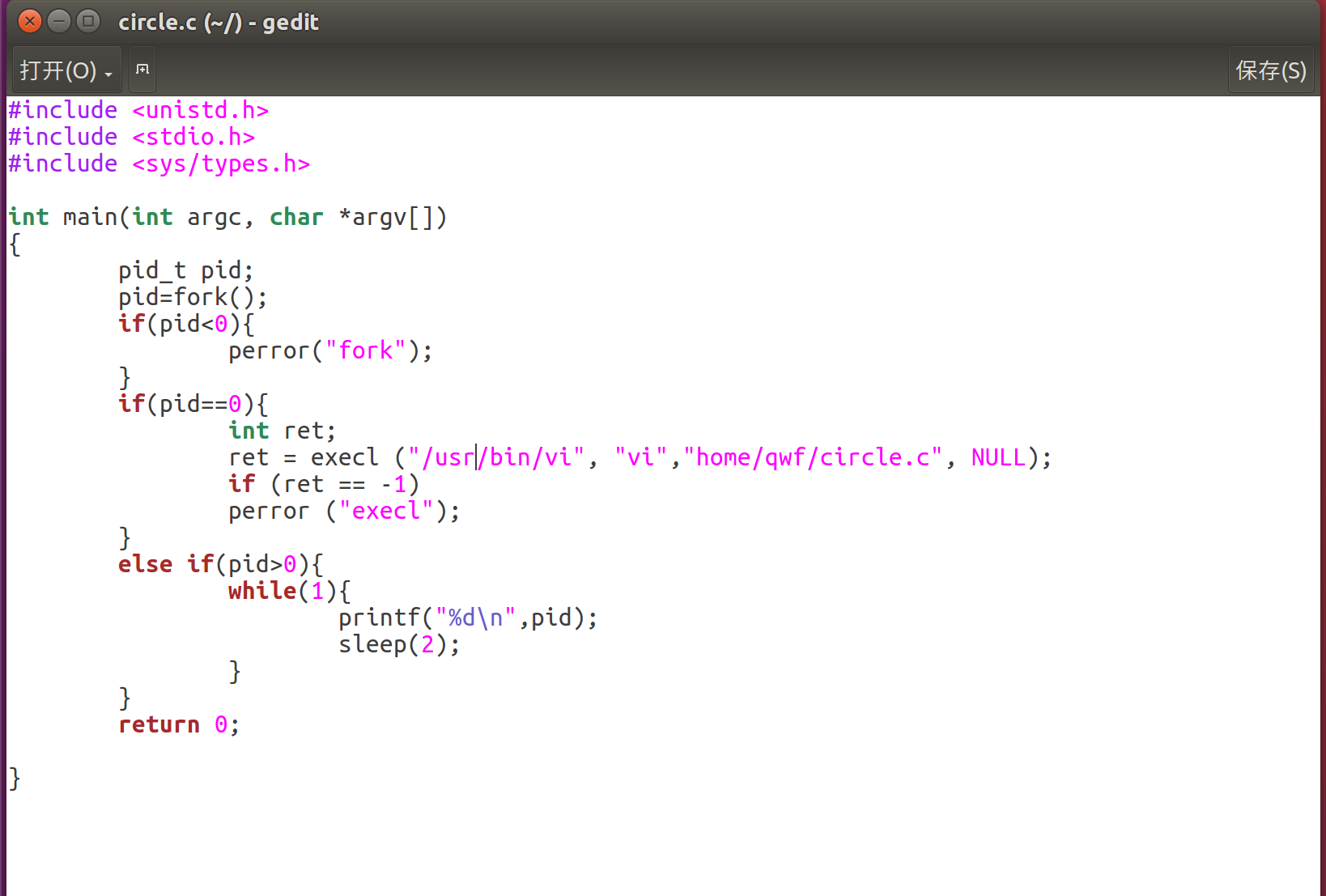
输入命令 pstree -p 查看进程树：



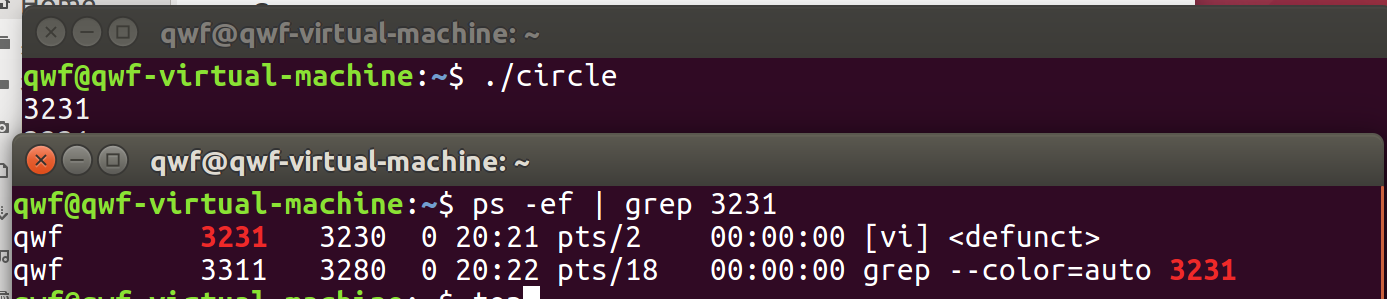
可以看到，从进程树中可以得到相同的结果。

**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。**

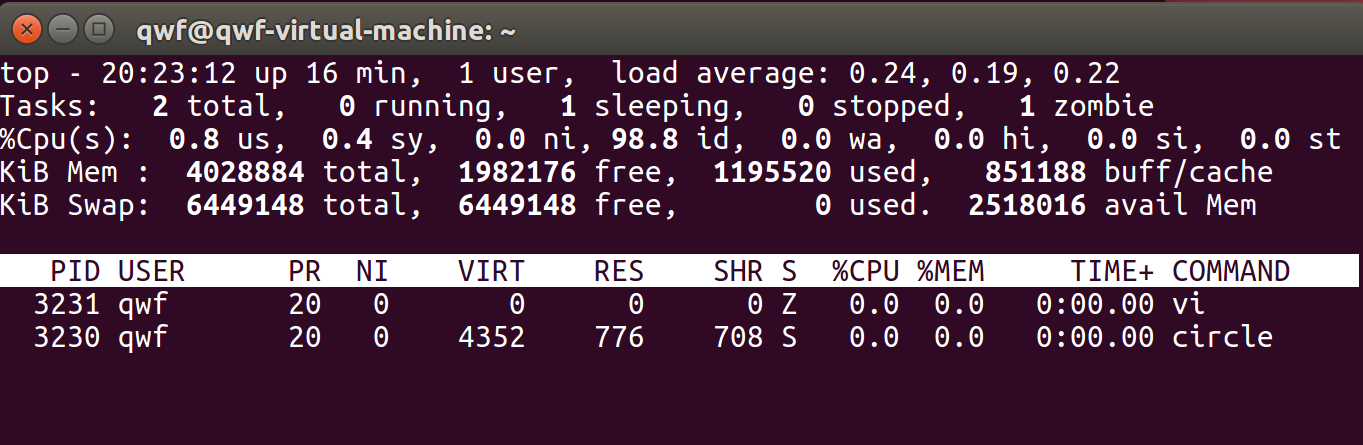
依照题意编写程序：



然后使用gcc编译并运行，并使用ps命令查询父进程的运行状况



之后使用top命令查看两个进程的运行状况：



图中的参数含义如下：

PID: 进程描述符

USER： 进程的拥有者

PRI：进程的优先级

NI： nice level

SIZE： 进程拥有的内存（包括code segment + data segment + stack segment）

RSS: 物理内存使用

VIRT（virtul memory usage）: 进程需要的虚拟内存大小

RES(resident memory usage)： 常驻内存

SHARE: 和其他进程共享的物理内存空间

STAT：进程的状态，有 S=sleeping，R=running，T=stopped or traced，D=interruptible sleep（不可中断的睡眠状态），Z=zombie。

%CPU： CPU使用率

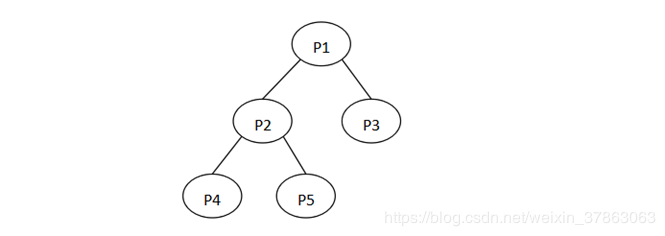
%MEM： 物理内存的使用

TIME： 进程占用的总共cpu时间

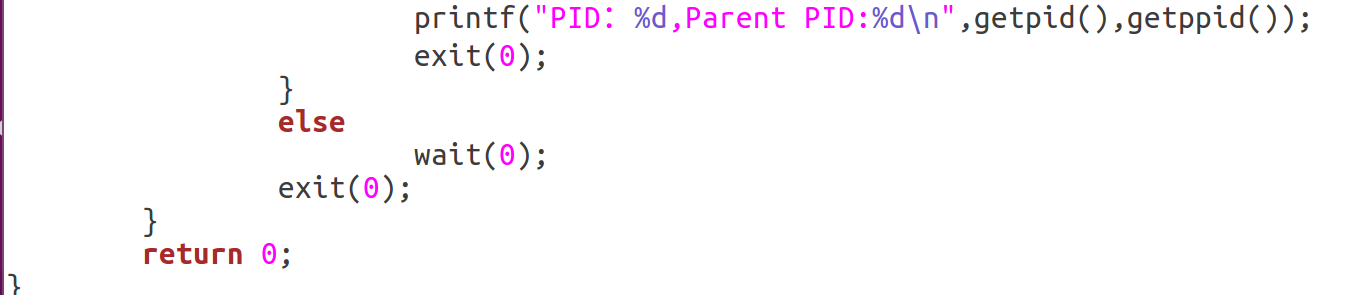
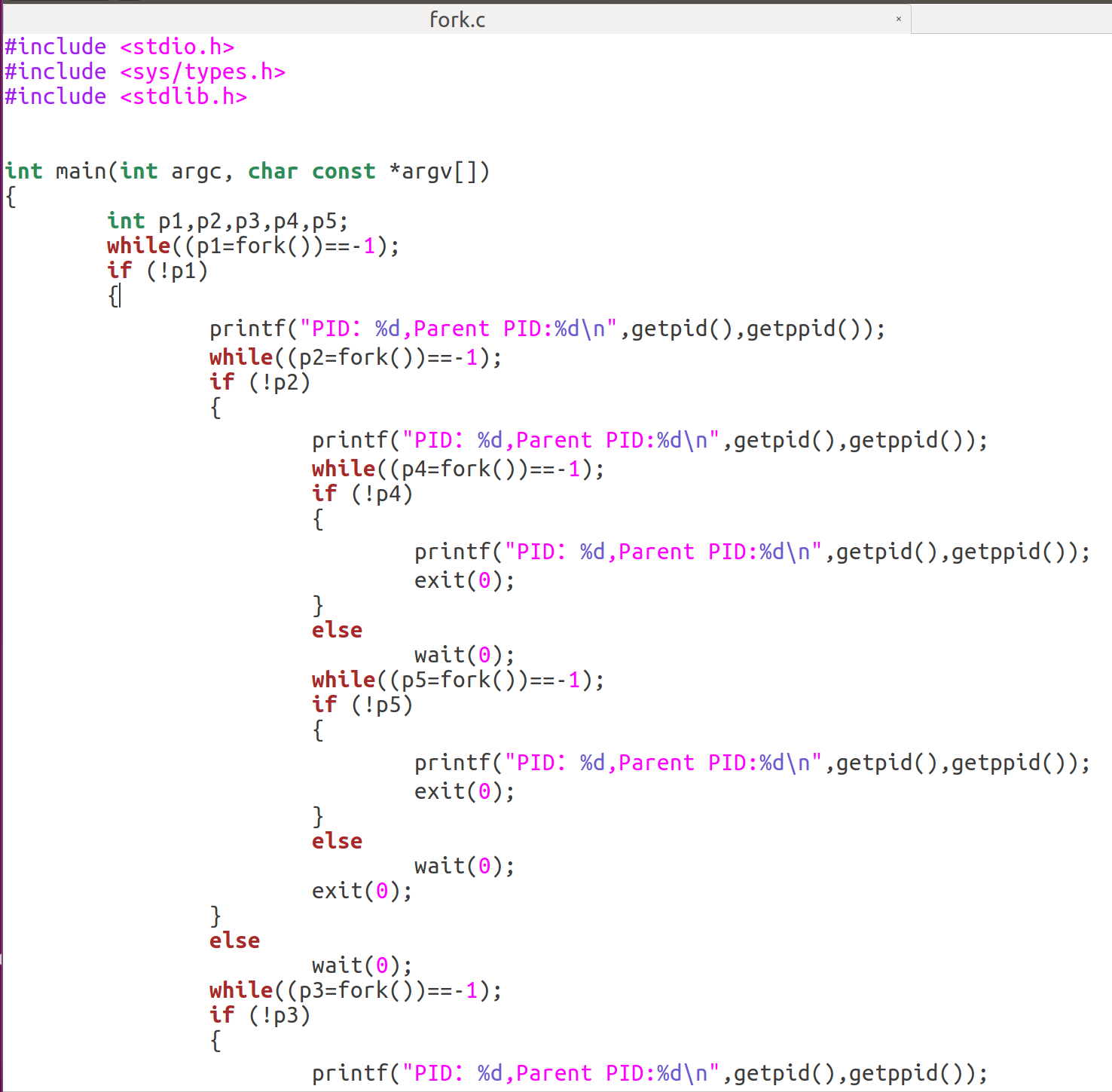
COMMAND：进程的命令

并且可以看出，父进程和子进程的cpu占用率都是0。

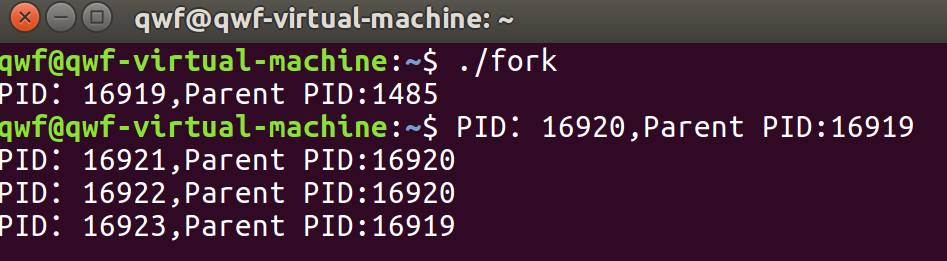
**3、使用fork系统调用，创建如下进程树，并使每个进程输出自己的ID和父进程的ID。观察进程的执行顺序和运行状态的变化。**



代码如下：



运行结果如图



从图中可以看出：

p1的进程号为16919；

p2的进程号为16920，其父进程为16919，即p1；

p3的进程号为16923，其父进程为16919，即p1；

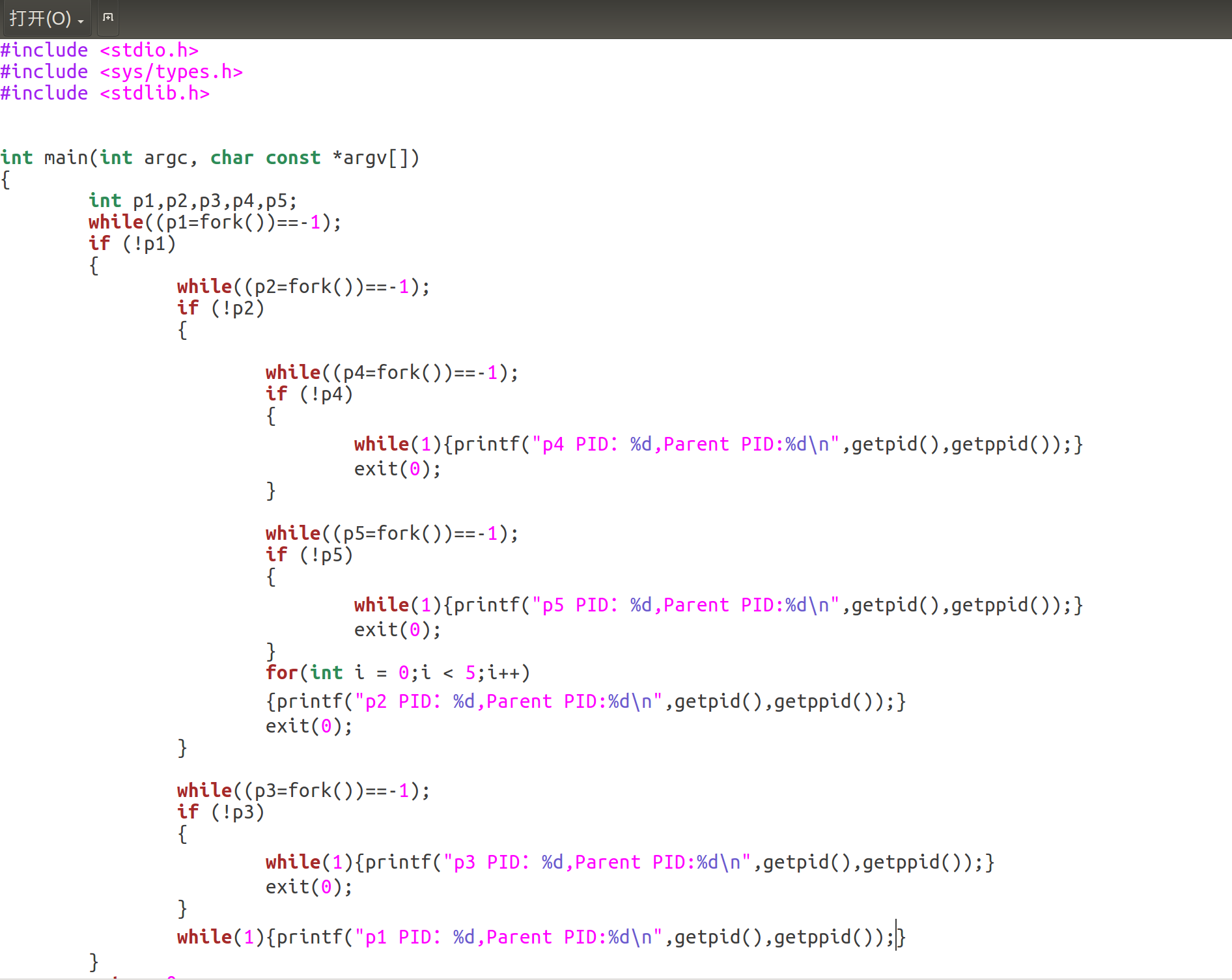
p4的进程号为16921，其父进程为16920，即p2；

p5的进程号为16922，其父进程为16920，即p2；

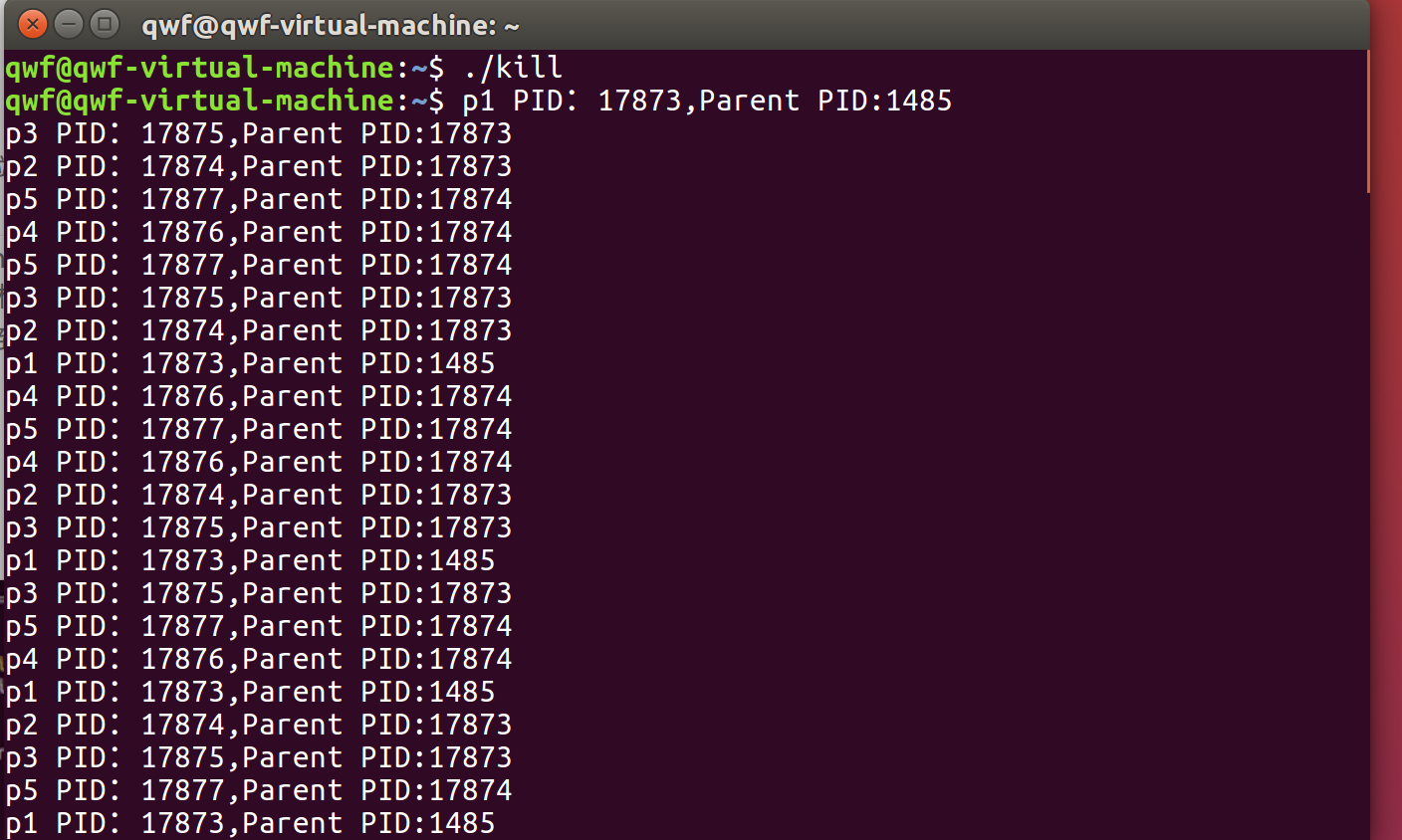
可以看出输出的顺序为：p1->p2->p4->p5->p3

**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。**

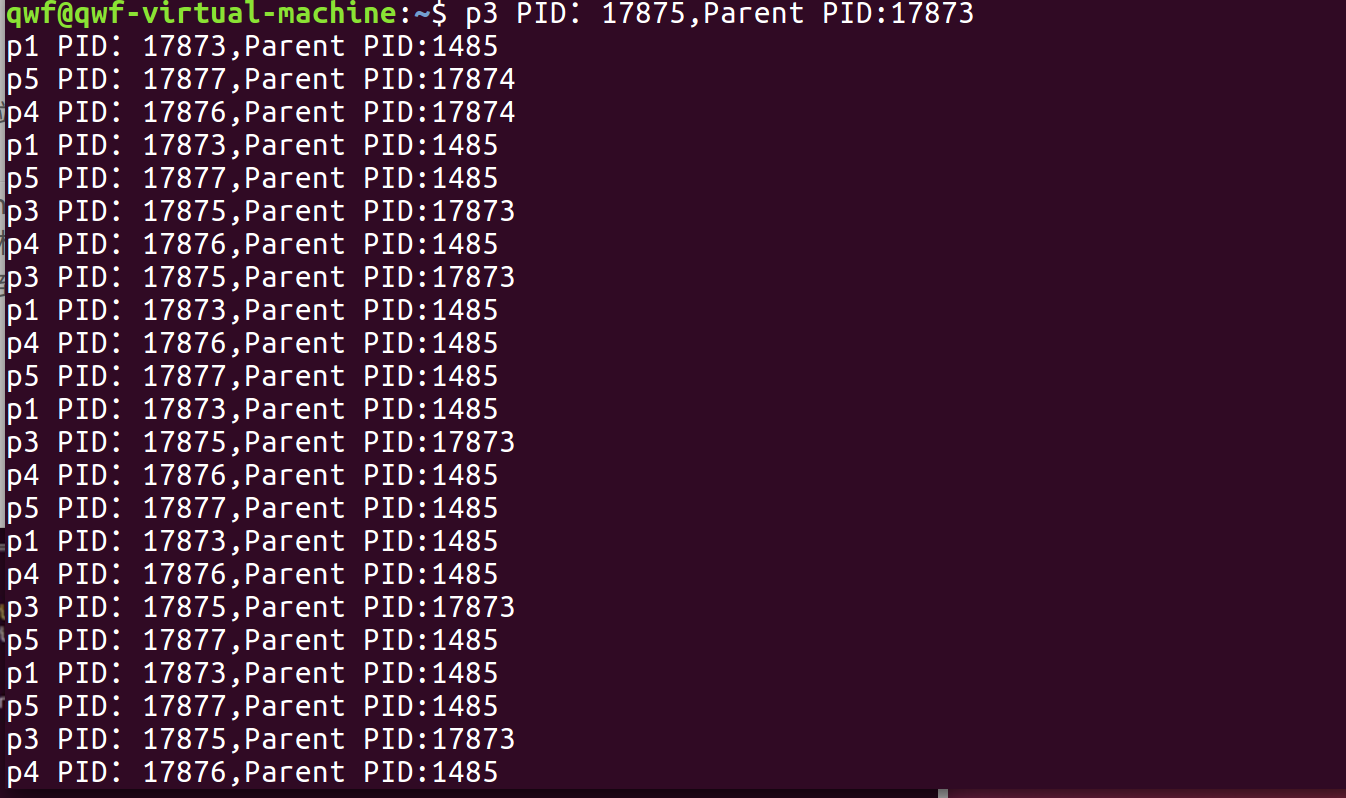
**代码如下**



运行结果如下：

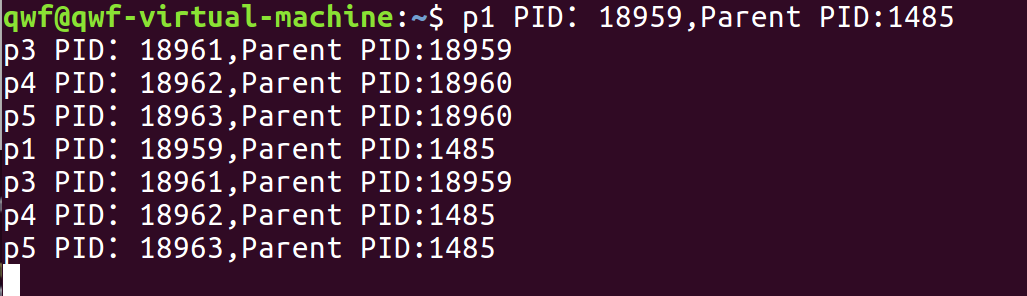


使用kill-9命令Kill掉p2后会变成如下结果：

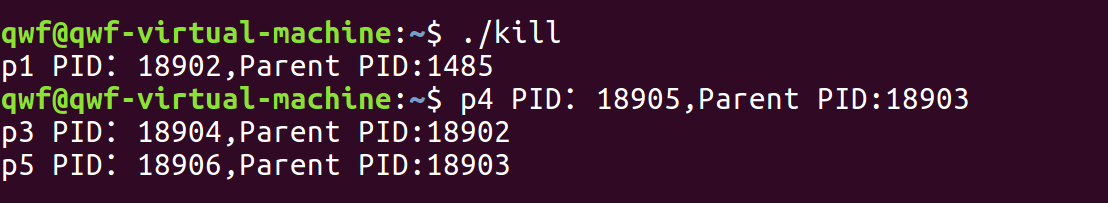
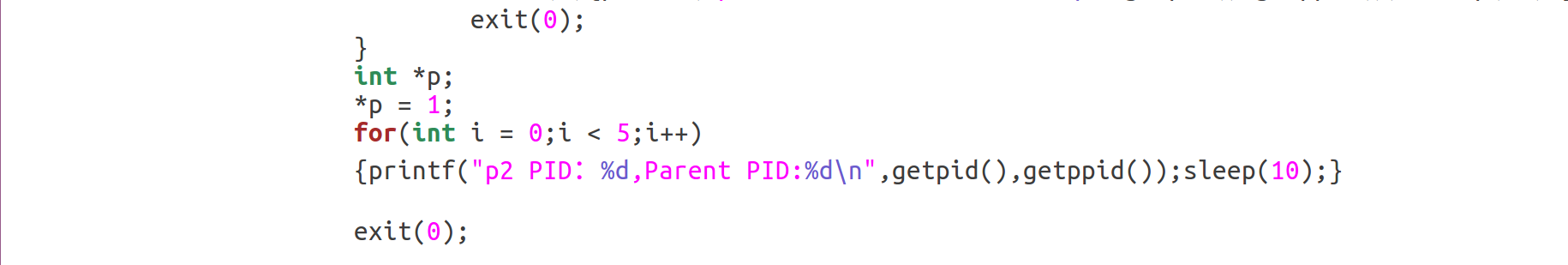


可以看出，失去了父进程p2的p4和p5的父进程变成了1485也就是p1的父进程。

使用exit退出会得到如下结果：



若使用段错误的方法，则在程序中插入一个野指针，再运行会变成如下结果：



结果和上面的两种情况相同，p2的两个子进程直接被重新分配了父进程PID，他们的父进程变成了p1的父进程。