实验一 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 );
```

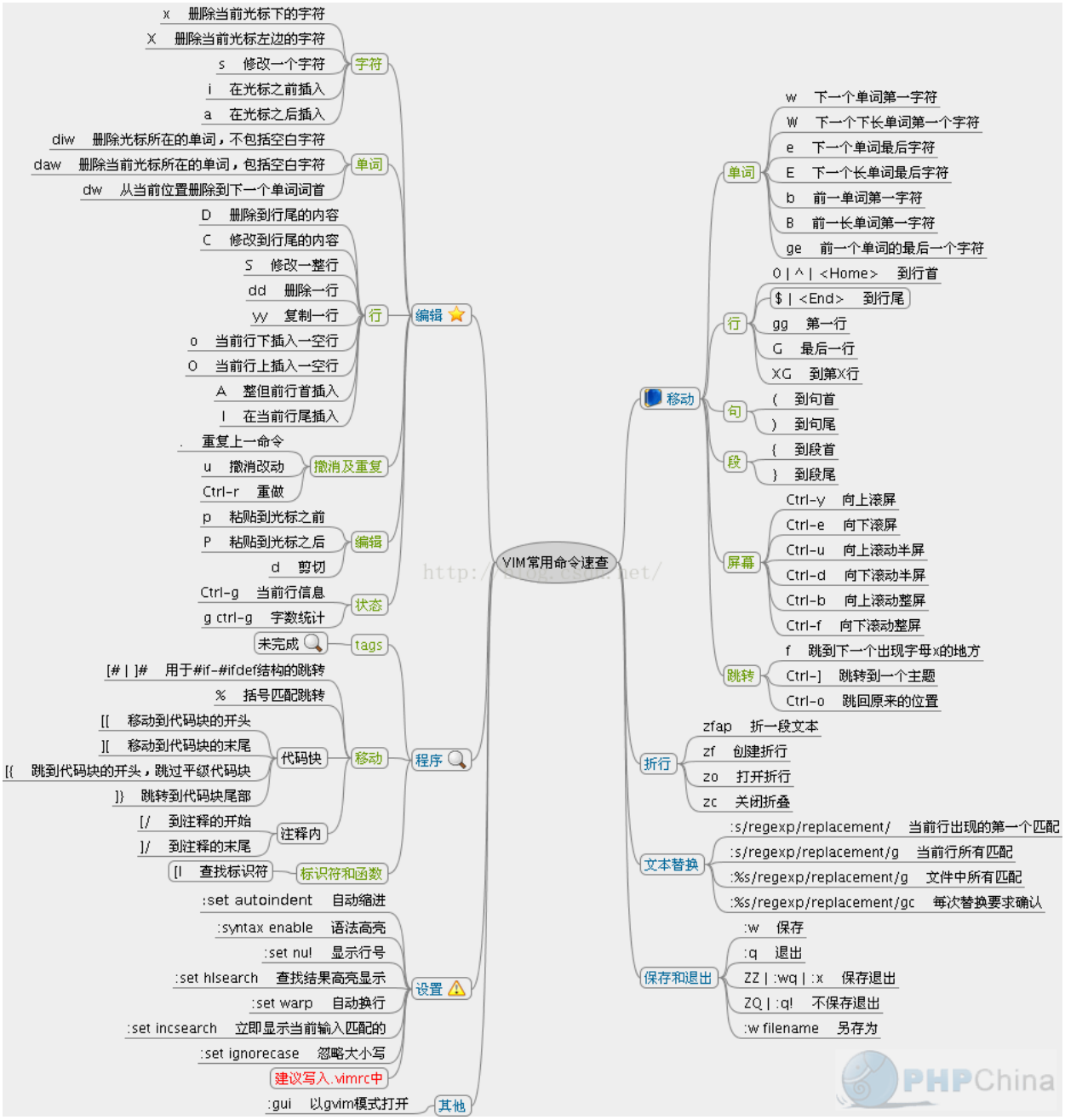
函数 pause 会把进程挂起，直到接收到信号。在信号接收后，进程会从 pause 函数中退出，继续运行。

- **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。

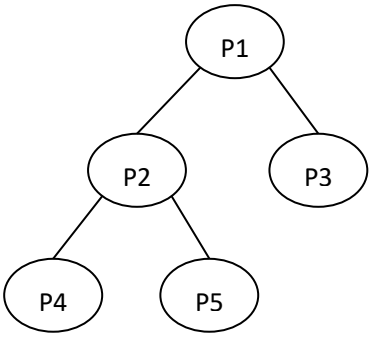
VIM 常用命令速查


VIM常用命令速查
编辑
字符
x 删除当前光标下的字符
X 删除当前光标左边的字符
s 修改一个字符
i 在光标之前插入
a 在光标之后插入
单词
diw 删除光标所在的单词，不包括空白字符
daw 删除当前光标所在的单词，包括空白字符
dw 从当前位置删除到下一个单词词首
行
D 删除到行尾的内容
C 修改到行尾的内容
S 修改一整行
dd 删除一行
yy 复制一行
o 当前行下插入一空行
O 当前行上插入一空行
A 整但前行首插入
I 在当前行尾插入
撤消及重复
. 重复上一命令
u 撤消改动
Ctrl-r 重做
编辑
p 粘贴到光标之前
P 粘贴到光标之后
d 剪切
状态
Ctrl-g 当前行信息
g ctrl-g 字数统计
程序
tags
未完成
移动
代码块
[# |]# 用于#if-#ifdef结构的跳转
% 括号匹配跳转
[[移动到代码块的开头
][移动到代码块的末尾
[{ 跳到代码块的开头，跳过平级代码块
]} 跳转到代码块尾部
注释内
[/ 到注释的开始
]/ 到注释的末尾
标识符和函数
[I 查找标识符
设置
:set autoindent 自动缩进
:syntax enable 语法高亮
:set nu! 显示行号
:set hlsearch 查找结果高亮显示
:set warp 自动换行
:set incsearch 立即显示当前输入匹配的
:set ignorecase 忽略大小写
建议写入.vimrc中
其他
:gui 以gvim模式打开
移动
单词
w 下一个单词第一字符
W 下一个下长单词第一个字符
e 下一个单词最后字符
E 下一个长单词最后字符
b 前一单词第一字符
B 前一长单词第一字符
ge 前一个单词的最后一个字符
行
0 | ^ | <Home> 到行首
$ | <End> 到行尾
gg 第一行
G 最后一行
XG 到第X行
句
(到句首
) 到句尾
段
{ 到段首
} 到段尾
屏幕
Ctrl-y 向上滚屏
Ctrl-e 向下滚屏
Ctrl-u 向上滚动半屏
Ctrl-d 向下滚动半屏
Ctrl-b 向上滚动整屏
Ctrl-f 向下滚动整屏
跳转
f 跳到下一个出现字母x的地方
Ctrl-] 跳转到一个主题
Ctrl-o 跳回原来的位置
折行
zfap 折一段文本
zf 创建折行
zo 打开折行
zc 关闭折叠
文本替换
:s/regexp/replacement/ 当前行出现的第一个匹配
:s/regexp/replacement/g 当前行所有匹配
:%s/regexp/replacement/g 文件中所有匹配
:%s/regexp/replacement/gc 每次替换要求确认
保存和退出
:w 保存
:q 退出
ZZ | :wq | :x 保存退出
ZQ | :q! 不保存退出
:w filename 另存为
PHPChina

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。

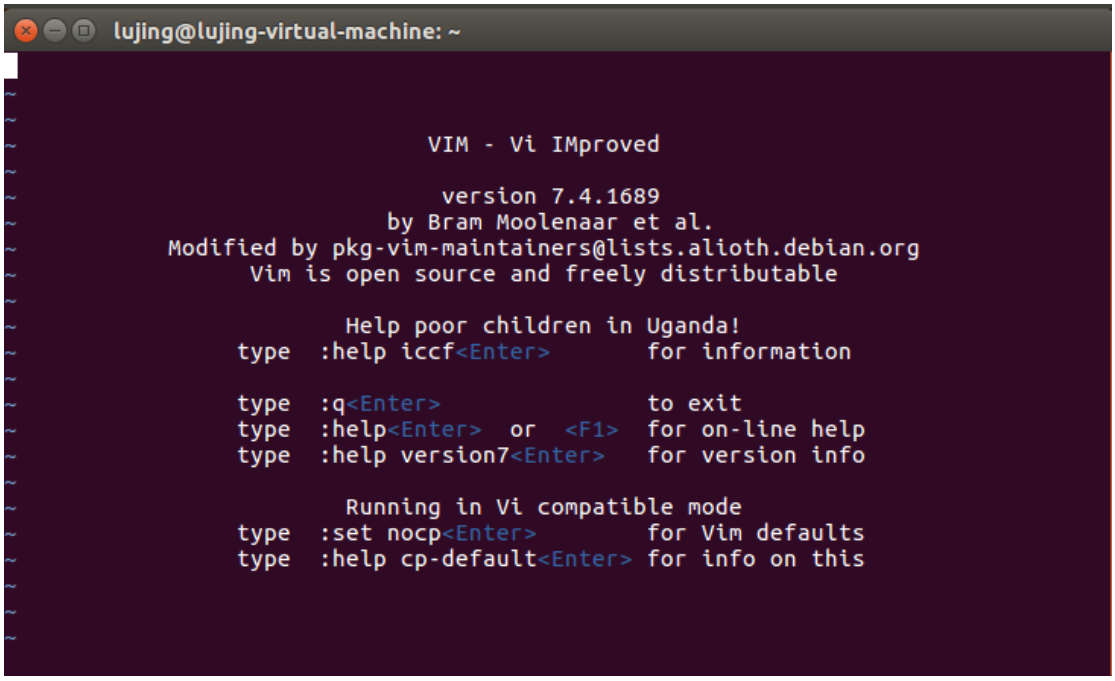


- 4、修改上述进程树中的进程，使得所有进程都循环输出自己的 ID 和父进程的 ID。然后终止 p2 进程(分别采用 kill -9 、自己正常退出 exit()、段错误退出)，观察 p1、p3、p4、p5 进程的运行状态和其他相关参数有何改变。

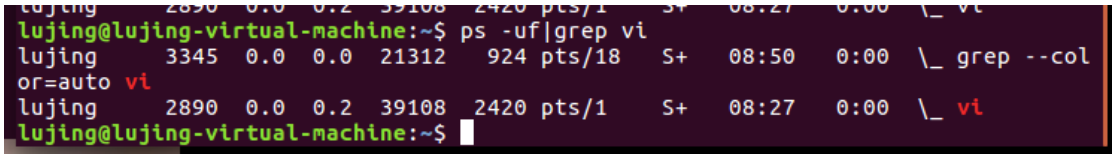
4. 实验过程及结果：

1、

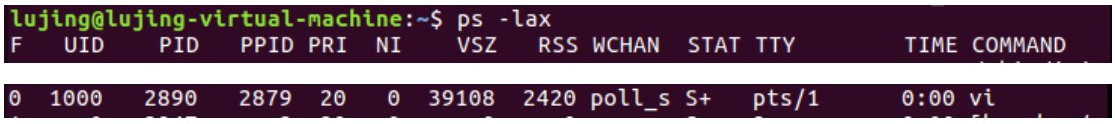
（1）打开一个终端，输入 vi，开启一个新的进程



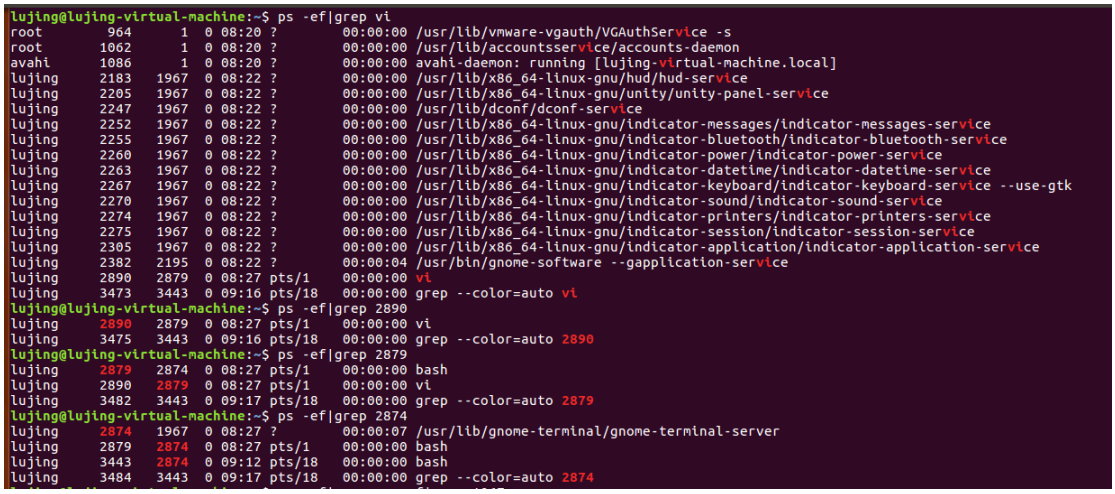
（2）打开另一个终端，输入 ps -uf|grep vi 命令，可见进程 vi 的 id 为 2890



（3）输入 ps -lax 查看父进程 id 为 2879，父进程即 PPID 的值



（4）输入 ps -ef|grep 进程号，按此格式一直往前查询，知道查询到父进程为 1



```
lujing@lujing-virtual-machine:~$ ps -ef|grep 1967
lujing     1967   1461  0 08:21 ?        00:00:00 /sbin/upstart --user
lujing     2040   1967  0 08:22 ?        00:00:00 upstart-udev-bridge --daemon --user
lujing     2054   1967  0 08:22 ?        00:00:01 dbus-daemon --fork --session --address=unix:abstract=/tmp/dbus-TdT65wJMiI
lujing     2066   1967  0 08:22 ?        00:00:00 /usr/lib/x86_64-linux-gnu/hud/window-stack-bridge
lujing     2104   1967  0 08:22 ?        00:00:00 /usr/lib/x86_64-linux-gnu/bamf/bamfdaemon
lujing     2109   1967  0 08:22 ?        00:00:00 upstart-dbus-bridge --daemon --session --user --bus-name session
lujing     2111   1967  0 08:22 ?        00:00:00 upstart-dbus-bridge --daemon --system --user --bus-name system
lujing     2113   1967  0 08:22 ?        00:00:00 upstart-file-bridge --daemon --user
lujing     2120   1967  0 08:22 ?        00:00:00 /usr/lib/gvfs/gvfsd
lujing     2124   1967  0 08:22 ?        00:00:01 /usr/bin/fcitx
lujing     2129   1967  0 08:22 ?        00:00:00 /usr/lib/gvfs/gvfsd-fuse /run/user/1000/gvfs -f -o big_writes
lujing     2139   1967  0 08:22 ?        00:00:00 /usr/lib/at-spi2-core/at-spi-bus-launcher
lujing     2142   1967  0 08:22 ?        00:00:00 /usr/bin/dbus-daemon --fork --print-pid 5 --print-address 7 --config-file /usr/share/fcitx/dbus/daemon.conf
lujing     2149   1967  0 08:22 ?        00:00:00 /usr/bin/fcitx-dbus-watcher unix:abstract=/tmp/dbus-xqyEAKT3zO,guid=3f0d4f83e15b5b5cdd6fdf8f5c899ea8 2142
lujing     2158   1967  0 08:22 ?        00:00:00 /usr/lib/at-spi2-core/at-spi2-registryd --use-gnome-session
lujing     2162   1967  0 08:22 ?        00:00:00 /usr/lib/x86_64-linux-gnu/notify-osd
lujing     2173   1967  0 08:22 ?        00:00:00 gpg-agent --homedir /home/lujing/.gnupg --use-standard-socket --daemon
lujing     2183   1967  0 08:22 ?        00:00:00 /usr/lib/x86_64-linux-gnu/hud/hud-service
lujing     2185   1967  0 08:22 ?        00:00:00 /usr/lib/unity-settings-daemon/unity-settings-daemon
lujing     2195   1967  0 08:22 ?        00:00:00 /usr/lib/gnome-session/gnome-session-binary --session=ubuntu
lujing     2205   1967  0 08:22 ?        00:00:00 /usr/lib/x86_64-linux-gnu/unity/unity-panel-service
lujing     2247   1967  0 08:22 ?        00:00:00 /usr/lib/dconf/dconf-service
lujing     2252   1967  0 08:22 ?        00:00:00 /usr/lib/x86_64-linux-gnu/indicator-messages/indicator-messages-service
lujing     2255   1967  0 08:22 ?        00:00:00 /usr/lib/x86_64-linux-gnu/indicator-bluetooth/indicator-bluetooth-service
lujing     2260   1967  0 08:22 ?        00:00:00 /usr/lib/x86_64-linux-gnu/indicator-power/indicator-power-service
lujing     2263   1967  0 08:22 ?        00:00:00 /usr/lib/x86_64-linux-gnu/indicator-datetime/indicator-datetime-service
lujing     2267   1967  0 08:22 ?        00:00:00 /usr/lib/x86_64-linux-gnu/indicator-keyboard/indicator-keyboard-service --use-gtk
lujing     2270   1967  0 08:22 ?        00:00:00 /usr/lib/x86_64-linux-gnu/indicator-sound/indicator-sound-service
lujing     2274   1967  0 08:22 ?        00:00:00 /usr/lib/x86_64-linux-gnu/indicator-printers/indicator-printers-service
lujing     2275   1967  0 08:22 ?        00:00:00 /usr/lib/x86_64-linux-gnu/indicator-session/indicator-session-service
lujing     2303   1967  0 08:22 ?        00:00:00 /usr/lib/evolution/evolution-source-registry
lujing     2304   1967  0 08:22 ?        00:00:13 compiz
lujing     2305   1967  0 08:22 ?        00:00:00 /usr/lib/x86_64-linux-gnu/indicator-application/indicator-application-service
lujing     2312   1967  0 08:22 ?        00:00:00 /usr/bin/pulseaudio --start --log-target=syslog
lujing     2346   1967  0 08:22 ?        00:00:00 /usr/lib/evolution/evolution-calendar-factory
lujing     2386   1967  0 08:22 ?        00:00:04 /usr/lib/vmware-tools/sbin64/vmtoolsd -n vmusr --blockFd 3
lujing     2392   1967  0 08:22 ?        00:00:00 /usr/lib/gvfs/gvfs-udisks2-volume-monitor
lujing     2424   1967  0 08:22 ?        00:00:00 /usr/lib/gvfs/gvfs-afc-volume-monitor
lujing     2433   1967  0 08:22 ?        00:00:00 /usr/lib/gvfs/gvfs-mtp-volume-monitor
lujing     2440   1967  0 08:22 ?        00:00:00 /usr/lib/gvfs/gvfs-goa-volume-monitor
lujing     2446   1967  0 08:22 ?        00:00:00 /usr/lib/gvfs/gvfs-gphoto2-volume-monitor
lujing     2460   1967  0 08:22 ?        00:00:00 /usr/lib/evolution/evolution-addressbook-factory
lujing     2485   1967  0 08:22 ?        00:00:01 fcitx-qimpanel
lujing     2522   1967  0 08:22 ?        00:00:00 /usr/lib/x86_64-linux-gnu/gconf/gconfd-2
lujing     2544   1967  0 08:22 ?        00:00:00 /usr/lib/gvfs/gvfsd-trash --spawner :1.5 /org/gtk/gvfs/exec_spaw/0
lujing     2571   1967  0 08:22 ?        00:00:00 /bin/sh -c /usr/lib/x86_64-linux-gnu/zeitgeist/zeitgeist-maybe-vacuum; /usr/bin/zeitgeist-daemon
lujing     2582   1967  0 08:22 ?        00:00:00 /usr/lib/x86_64-linux-gnu/zeitgeist-fts
```

```
lujing     2584   1967  0 08:22 ?        00:00:00 zeitgeist-datahub
lujing     2874   1967  0 08:27 ?        00:00:09 /usr/lib/gnome-terminal/gnome-terminal-server
lujing     3252   1967  0 08:35 ?        00:00:12 /usr/bin/python3 /usr/bin/update-manager --no-update --no-focus-on-map
lujing     3493   3443  0 09:19 pts/18   00:00:00 grep --color=auto 1967
lujing@lujing-virtual-machine:~$ ps -ef|grep 1461
root       1461   1181  0 08:20 ?        00:00:00 lightdm --session-child 12 19
lujing     1967   1461  0 08:21 ?        00:00:00 /sbin/upstart --user
lujing     3517   3443  0 09:24 pts/18   00:00:00 grep --color=auto 1461
lujing@lujing-virtual-machine:~$ ps -ef|grep 1181
root       1181      1  0 08:20 ?        00:00:00 /usr/sbin/lightdm
root       1200   1181  0 08:20 tty7     00:00:19 /usr/lib/xorg/Xorg -core :0 -seat seat0 -auth /var/run/lightdm/root/:0 -nolisten tcp vt7 -novtswitch
root       1461   1181  0 08:20 ?        00:00:00 lightdm --session-child 12 19
lujing     3519   3443  0 09:24 pts/18   00:00:00 grep --color=auto 1181
lujing@lujing-virtual-machine:~$
```

由上图可得进程号顺序为：2890—>2879—>2874—>1967—>1461—>1181—>1

(5)输入 pstree -p 命令可得进程树如下：

```
lujing@lujing-virtual-machine:~$ pstree -p
systemd(1)─┬─ManagementAgent(1054)─┬─{CThreadUtils::s}(1067)
           │                       └─{CThreadUtils::s}(1078)
```

```
           ├─lightdm(1181)─┬─Xorg(1200)─┬─{InputThread}(1247)
           │               │            └─{InputThread}(1248)
           │               └─lightdm(1461)─┬─upstart(1967)─┬─at-spi-bus-laun(2139)─┬─dbus-daemon(2155)
           │                               │               │                       └─{dconf worker}(2152)
```

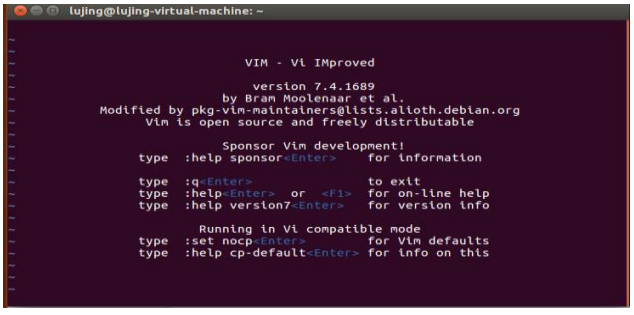
```
           │                               │               │                       ├─gnome-terminal-(2874)─┬─bash(2879)─┬─vi(2890)
           │                               │               │                       │                       │            └─{dconf worker}(2877)
           │                               │               │                       │                       ├─bash(3443)─┬─pstree(3535)
           │                               │               │                       │                       │            └─{dconf worker}(2877)
           │                               │               │                       │                       ├─{gdbus}(2876)
           │                               │               │                       │                       └─{gmain}(2875)
```

由上图可得进程号顺序为：2890—>2879—>2874—>1967—>1461—>1181—>1

所以由实验可知两种方法所得结果一致

2、

（1） 编写代码在子进程中调用 exec 打开编辑器运行如下图所示



（2）输入 ps-Al 命令

```
lujing@lujing-virtual-machine:~$ ps -Al
F S   UID   PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
0 S  1000  3880  3875  0  80   0 -  7386 wait   pts/1    00:00:00 bash
0 S  1000  3893  3880  0  80   0 -  9777 poll_s pts/1    00:00:00 vi
```

F:进程的标志

S:进程的状态

UID:用户号，1000

PID:vi 进程号，3893

PPID:vi 进程的父进程号，3880

C:cpu 使用资源的百分比

PRI:内核调度优先级，80

NI:进程优先级，为缺省值 0

ADDR:核心功能，指出该进程在内存的那一部分，如果是运行的进程，一般都是”-“

SZ:用掉的内存大小

WCHAN:当前进程是否正在运行，若为“-”则为正在运行

TTY:登录者终端位置

TIME:使用 cpu 的时间

CMD：进程名

输入 ps aux 命令

```
lujing@lujing-virtual-machine:~$ ps aux
USER         PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
lujing      3880  0.0  0.4  29544  4752 pts/1    Ss   10:09   0:00 bash
lujing      3893  0.0  0.3  39108  3676 pts/1    S+   10:09   0:00 vi
```

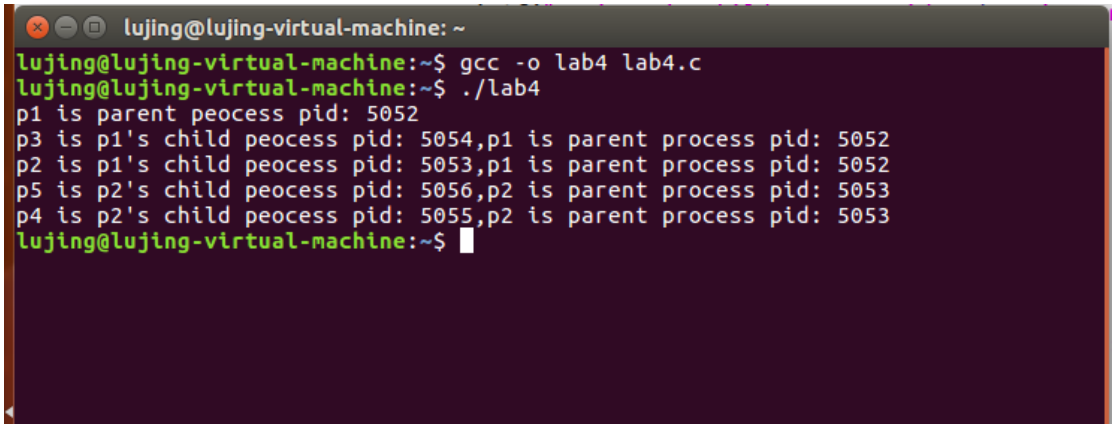
输入 top 命令，对 cpu 占用率排序

```
lujing@lujing-virtual-machine:~$ top

top - 10:25:59 up  2:05,  1 user,  load average: 0.05, 0.04, 0.06
Tasks: 219 total,   1 running, 218 sleeping,   0 stopped,   0 zombie
%Cpu(s):  6.9 us,  2.4 sy,  0.0 ni, 90.6 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem :   985856 total,    73588 free,   656412 used,   255856 buff/cache
KiB Swap:  1046524 total,   800764 free,   245760 used.   132284 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S %CPU %MEM     TIME+ COMMAND
 1200 root      20   0  454536  45856  16992 S  5.3  4.7   0:47.58 Xorg
 3875 lujing    20   0  620296  43428  32884 S  2.7  4.4   0:04.73 gnome-terminal-
 2304 lujing    20   0 1285064  98132  38492 S  2.3 10.0   0:50.12 compiz
 3963 lujing    20   0   48884   3668   2972 R  0.7  0.4   0:00.33 top
 2158 lujing    20   0  206964   2652   2364 S  0.3  0.3   0:01.19 at-spi2-registr
 2485 lujing    20   0  814092  17112   6996 S  0.3  1.7   0:02.72 fcitx-qimpanel
 2661 lujing    20   0  553560  14812   9496 S  0.3  1.5   0:00.88 update-notifier
 3947 root      20   0       0      0      0 S  0.3  0.0   0:00.45 kworker/0:0
    1 root      20   0  185292   4136   2540 S  0.0  0.4   0:03.36 systemd
top - 10:28:18 up  2:08,  1 user,  load average: 0.00, 0.02, 0.05
Tasks: 219 total,   1 running, 218 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.3 us,  0.7 sy,  0.0 ni, 99.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem :   985856 total,    73176 free,   656732 used,   255948 buff/cache
KiB Swap:  1046524 total,   800764 free,   245760 used.   131916 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S %CPU %MEM     TIME+ COMMAND
 2386 lujing    20   0  238676  21176  14488 S  0.3  2.1   0:12.12 vmtoolsd
 3875 lujing    20   0  620424  43644  32884 S  0.3  4.4   0:05.91 gnome-terminal-
 3947 root      20   0       0      0      0 S  0.3  0.0   0:00.52 kworker/0:0
 3963 lujing    20   0   48884   3668   2972 R  0.3  0.4   0:00.66 top
    1 root      20   0  185292   4136   2540 S  0.0  0.4   0:03.36 systemd
    2 root      20   0       0      0      0 S  0.0  0.0   0:00.01 kthreadd
    4 root       0 -20       0      0      0 S  0.0  0.0   0:00.00 kworker/0:0H
    6 root       0 -20       0      0      0 S  0.0  0.0   0:00.00 mm_percpu_wq
    7 root      20   0       0      0      0 S  0.0  0.0   0:00.64 ksoftirqd/0
    8 root      20   0       0      0      0 S  0.0  0.0   0:01.96 rcu_sched
    9 root      20   0       0      0      0 S  0.0  0.0   0:00.00 rcu_bh
   10 root      rt   0       0      0      0 S  0.0  0.0   0:00.00 migration/0
   11 root      rt   0       0      0      0 S  0.0  0.0   0:00.03 watchdog/0
```

代码：

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

int main(){
    int res;
    res=execl("/usr/bin/vi","vi",NULL);
    pid_t pid;
    pid=fork();
    if(pid<0){
    perror("fork");
    }
    if(0==pid){
        while(1){
        printf("child process");
        if(res==-1){
            perror("execl");
        }
        sleep(1);
        }
    }
    else if (pid>0){
        while(1){
        printf("parent process");
        sleep(1);
        }
    }
    return 0;
}
```

3、



由上图可知：

p1：5052

p2：5053

p3：5054

p4：5055

p5：5056

进程执行依据执行前后顺序依次占用进程 id 号。

代码：

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

int main(){
    int p1,p2,p3,p4,p5;
    while((p1=fork())==-1);
    if(p1==0){
        printf("p2 is p1's child peocess pid: %d,p1 is parent process pid: %d\n",getpid(),getppid());
        while((p2=fork())==-1);
        if(p2==0){//4
            printf("p4 is p2's child peocess pid: %d,p2 is parent process pid: %d\n",getpid(),getppid());
            waitpid(p2,NULL,0);
        }
        else{
```

```
                while((p5=fork())==-1);
                if(p5>0)sleep(1);
            else if(p5==0){
                    printf("p5 is p2's child peocess pid: %d,p2 is parent process pid: %d\n",getpid(),getppid());
                }
                waitpid(p5,NULL,0);
            }
        waitpid(p1,NULL,0);
        exit(0);
        }
        else{
            printf("p1 is parent peocess pid: %d\n",getpid());
            while((p3=fork())==-1);
            if(p3>0)sleep(1);
            else if(p3==0){
              printf("p3 is p1's child peocess pid: %d,p1 is parent process pid: %d\n",getpid(),getppid());
           }
              waitpid(p3,NULL,0);
              exit(0);
        }

}
```

4、

正常循环输出：

```
p1 is parent peocess pid: 3167
p3 is p1's child peocess pid: 3169,p1 is parent process pid: 3167
p5 is p2's child peocess pid: 3171,p2 is parent process pid: 3168
p2 is p1's child peocess pid: 3168,p1 is parent process pid: 3167
p3 is p1's child peocess pid: 3169,p1 is parent process pid: 3167
p4 is p2's child peocess pid: 3170,p2 is parent process pid: 3168
p1 is parent peocess pid: 3167
p5 is p2's child peocess pid: 3171,p2 is parent process pid: 3168
p2 is p1's child peocess pid: 3168,p1 is parent process pid: 3167
p5 is p2's child peocess pid: 3171,p2 is parent process pid: 3168
p4 is p2's child peocess pid: 3170,p2 is parent process pid: 3168
p3 is p1's child peocess pid: 3169,p1 is parent process pid: 3167
p2 is p1's child peocess pid: 3168,p1 is parent process pid: 3167
p2 is p1's child peocess pid: 3168,p1 is parent process pid: 3167
```

Kill -9 终止进程 p2：

```
lujing@lujing-virtual-machine:~$ kill -9 3168
lujing@lujing-virtual-machine:~$
lujing@lujing-virtual-machine:~$ ps -ef|grep lab4
lujing      3115   2191  1 14:24 ?        00:00:04 gedit /home/lujing/lab4.c
lujing      3167   3146  0 14:25 pts/0    00:00:00 ./lab4
lujing      3168   3167  0 14:25 pts/0    00:00:00 [lab4] <defunct>
lujing      3169   3167  0 14:25 pts/0    00:00:00 ./lab4
lujing      3170   2191  0 14:25 pts/0    00:00:00 ./lab4
lujing      3171   2191  0 14:25 pts/0    00:00:00 ./lab4
lujing      3186   3146  0 14:31 pts/0    00:00:00 grep --color=auto lab4
lujing@lujing-virtual-machine:~$ ps -Al|egrep -e lab4
0 T  1000   3167   3146  0  80   0 -  1088 signal pts/0    00:00:00 lab4
1 Z  1000   3168   3167  0  80   0 -     0 -      pts/0    00:00:00 lab4 <defunc
t>
1 T  1000   3169   3167  0  80   0 -  1088 signal pts/0    00:00:00 lab4
1 T  1000   3170   2191  0  80   0 -  1088 signal pts/0    00:00:00 lab4
1 T  1000   3171   2191  0  80   0 -  1088 signal pts/0    00:00:00 lab4
lujing@lujing-virtual-machine:~$
```

exit 终止 p2 循环输出：

```
p1 is parent peocess pid: 3265
p5 is p2's child peocess pid: 3269,p2 is parent process pid: 2191
p3 is p1's child peocess pid: 3267,p1 is parent process pid: 3265
p4 is p2's child peocess pid: 3268,p2 is parent process pid: 2191
p1 is parent peocess pid: 3265
p1 is parent peocess pid: 3265
p5 is p2's child peocess pid: 3269,p2 is parent process pid: 2191
p3 is p1's child peocess pid: 3267,p1 is parent process pid: 3265
p4 is p2's child peocess pid: 3268,p2 is parent process pid: 2191
p1 is parent peocess pid: 3265
p3 is p1's child peocess pid: 3267,p1 is parent process pid: 3265
p4 is p2's child peocess pid: 3268,p2 is parent process pid: 2191
p5 is p2's child peocess pid: 3269,p2 is parent process pid: 2191
p3 is p1's child peocess pid: 3267,p1 is parent process pid: 3265
p4 is p2's child peocess pid: 3268,p2 is parent process pid: 2191
```

exit 终止 p2：

```
lujing@lujing-virtual-machine:~$ ps -Al|egrep -e lab4
0 T  1000   3265   3244  0  80   0 -  1088 signal pts/0    00:00:00 lab4
1 Z  1000   3266   3265  0  80   0 -     0 -      pts/0    00:00:00 lab4 <defunc
t>
1 T  1000   3267   3265  0  80   0 -  1088 signal pts/0    00:00:00 lab4
1 T  1000   3268   2191  0  80   0 -  1088 signal pts/0    00:00:00 lab4
1 T  1000   3269   2191  0  80   0 -  1088 signal pts/0    00:00:00 lab4
lujing@lujing-virtual-machine:~$
```

段错误退出：循环段错误退出未做到，下图为运行一遍段错误退出截图

```
lujing@lujing-virtual-machine: ~
lujing@lujing-virtual-machine:~$ ./lab3
p2 is p1's child peocess pid: 4230,p1 is parent process pid: 4229
p4 is p2's child peocess pid: 4231,p2 is parent process pid: 4230
p5 is p2's child peocess pid: 4232,p2 is parent process pid: 4230
p3 is p1's child peocess pid: 4233,p1 is parent process pid: 4229
p1 is parent peocess pid: 4229
lab3: cxa_atexit.c:100: __new_exitfn: Assertion `l != NULL' failed.
已放弃 (核心已转储)
lujing@lujing-virtual-machine:~$
```

p1、p3、p4、p5 进程的运行状态和其他相关参数在终止进程 p2 后没有任何改变，p2 被终止后用掉内存大小为 0，即 p2 被终止后不运行也不占用内存

遇到的问题及解决办法：

Q:在第三题调用函数 fork()创建进程树时，用 getppid()函数取父进程的 id 号时，总是错误的，用 getpid()函数取得当前进程号时却是对的，进程树的所有父进程 id 号都为 1880，而不是正确的父进程。

A: 这个问题是孤儿进程收养问题，即在执行过程中，先打印出父进程的进程信息，然后再打印出子进程的进程信息，那么就相当于父进程完成之后，被杀死了，然后再执行的是子进程的信息，而此时子进程就成了孤儿进程，它被 upstart 这个进程收养了，此时调用 getppid()返回的就是 upstart 的 pid。（如果返回 1，那么对应的就是 init 的 pid）故需加入 sleep(1)降低父进程的速度，即可解决此问题。

补充：

1.fork 之后，父子进程交替运行

2.如果父进程死亡，子进程还活着，叫孤儿进程。（孤儿进程托管给 1 号进程，1 号进程也叫孤儿院）

3.如果父进程活着，子进程死亡，子进程死亡就会成为僵尸进程，僵尸进程会占用少量系统图资源

Q:实现段错误退出

A: 段错误是指访问的内存超出了系统给这个程序所设定的内存空间，例如访问了不存在的内存地址、访问了系统保护的内存地址、访问了只读的内存地址等等情况。在实现进程树中，vfork 也是用来创建进程子进程的，但是 vfork 创建的子进程和父进程共享一块地址空间，vfork 保证子进程先运行，在它调用 exec 或 exit 之后父进程才可能被调度运行, 所以可以用 vfork 来生产进程树，且在 p2 处不写语句 exit，从而实现 p2 段错误退出。

补充：

1.父进程阻塞，直到子程序运行完毕

2.就算写时，也不拷贝

3.必须使用 exit 或者 exec 才不会出现段错误

4.每个系统对 vfork 的实现或多或少都有问题，所以尽量不要使用