# 实验二 进程控制

李许增16281042

操作系统arch Linux

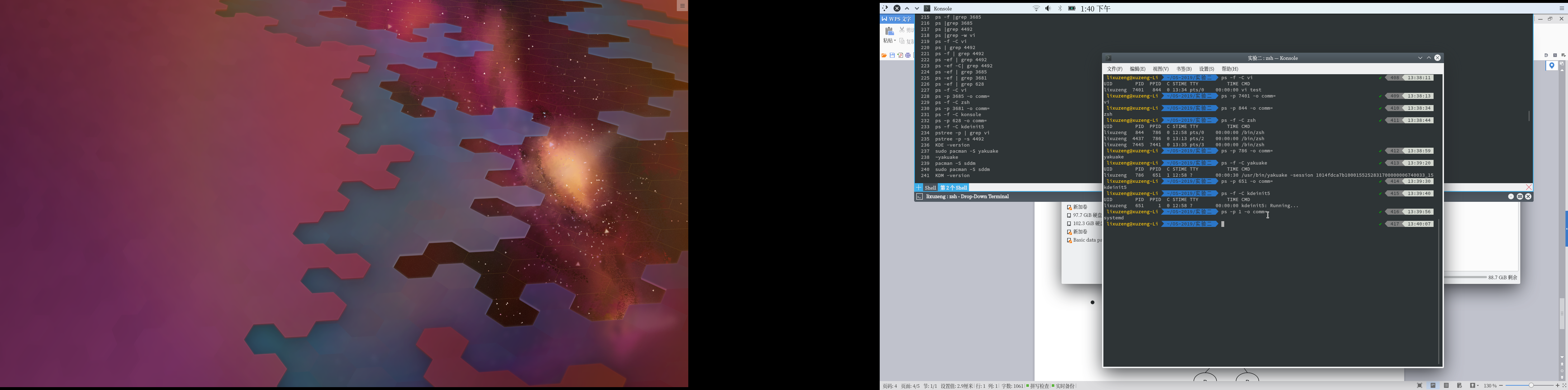
**1、打开一个vi进程。通过ps命令以及选择合适的参数，只显示名字为vi的进程。寻找vi进程的父进程，直到init进程为止。记录过程中所有进程的ID和父进程ID。将得到的进程树和由pstree命令的得到的进程树进行比较。**

单步ps命令选择得到的进程树和用pstree命令得到的进程树相同。

ps -f -C name 命令为筛选指定名称进程的信息；

ps -p 进程号 -o comm= 为查找pid对应进程的名称；

ps -p -s 进程号 命令为查找指定进程号的进程树。



截图_2019-03-23_13-44-40

**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。**

程序代码:

#include<unistd.h>

#include<stdio.h>

int main()

{

pid\_t fork\_pid;//fork\_pid为fork函数返回值

fork\_pid = fork();

if(fork\_pid < 0)//当返回值为负值时表示出现错误

{

printf("fork error");

}

else if(fork\_pid == 0)//当返回值为0时表示创建的子进程在运行

{

printf("我是子进程，我的pid为：%d\n",getpid());

if((execlp("vi","vi","/home/os-2019/test",NULL))<0) //注意execlp参数使用

printf("execlp error\n");

}

else//当返回值为其他正值时表示父进程在运行

{

printf("我是父进程，我的pid为：%d\n",getpid());

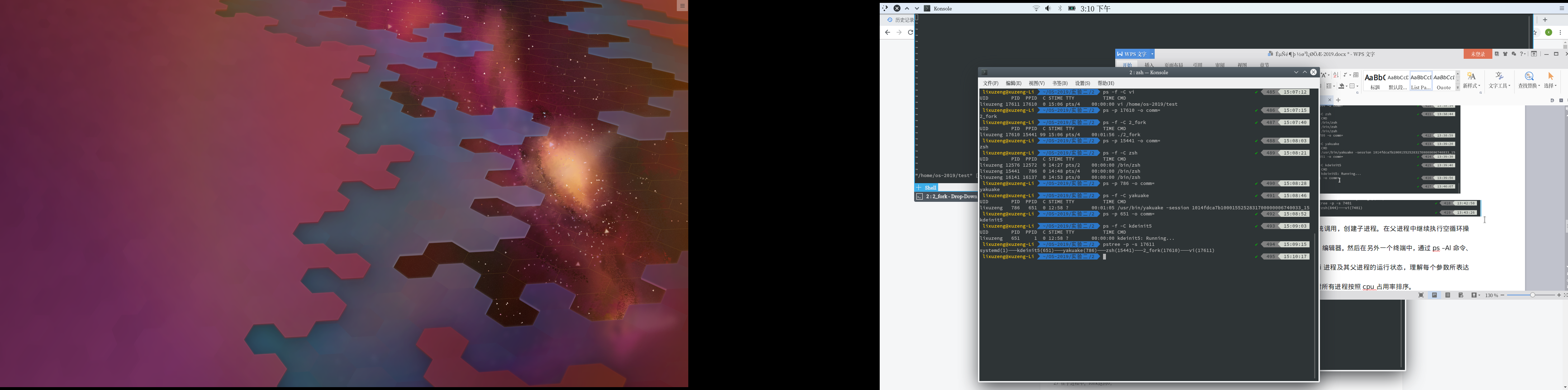
while(1){};

}

return 0;

}

首先利用pstree命令查看vi进程以及父进程：



执行ps aux命令查看当前系统中运行程序情况:

制表项内容说明：

USER: 进程拥有者

PID: 进程pid

%CPU: 占用的 CPU 使用率

%MEM: 占用的内存使用率

VSZ: 占用的虚拟内存大小

RSS: 占用的内存大小

TTY: 终端的次要装置号码 (minor device number of tty)

STAT: 该进程的状态，linux的进程有以下状态：

D 不可中断 Uninterruptible（usually IO）

R 正在运行，或在队列中的进程

S 处于休眠状态

T 停止或被追踪

Z 僵尸进程

W 进入内存交换（从内核2.6开始无效）

X 死掉的进程

< 高优先级

n 低优先级

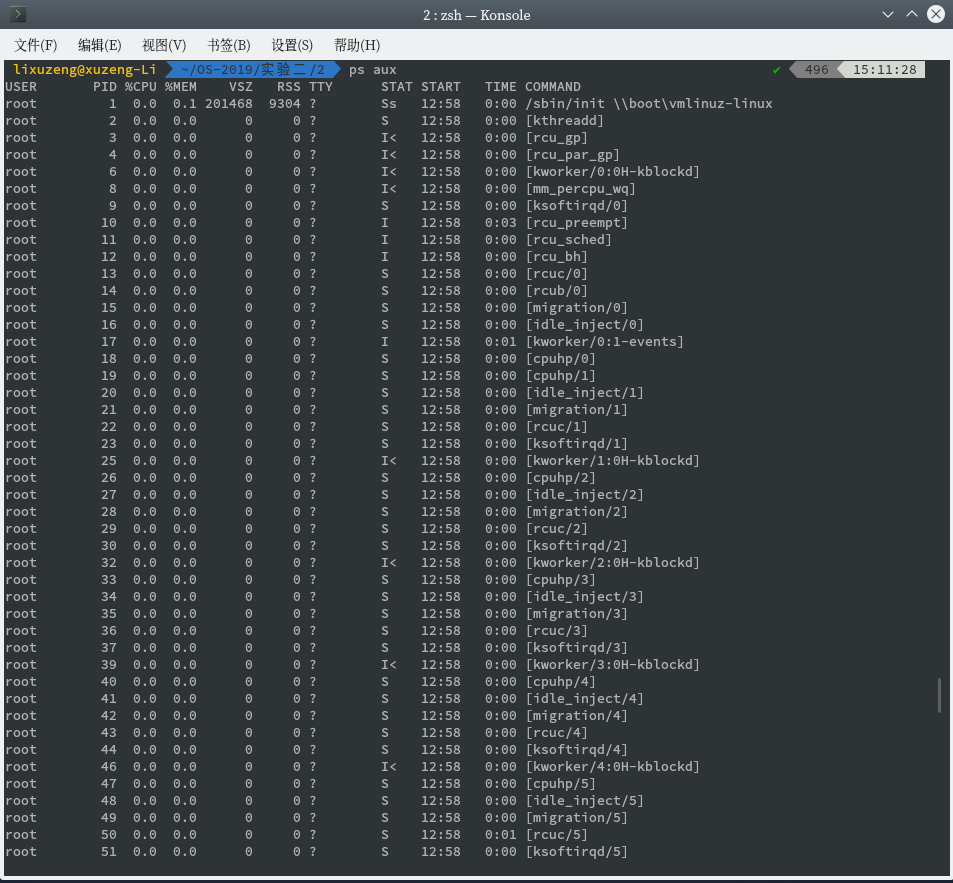
s 包含子进程

+ 位于后台的进程组

START: 进程开始时间

TIME: 执行的时间

COMMAND:对应执行的指令



对应和vi进程相关的进程信息：

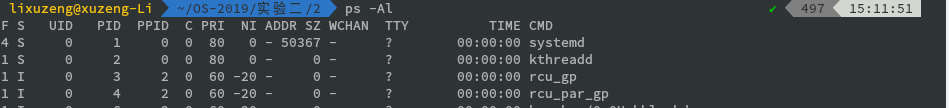








ps –Al命令查看进程信息：



制表项内容说明：

UID：用户ID（User ID）

PPID：父进程pid

S：进程或内核线程的状态：

O：不存在

A：活动

W：已交换

I：空闲（等待启动）

Z：已取消

T：已停止

S：正在休眠

R：正在运行

C：（-f、l 和 -l 标志）每次系统时钟周期和发现线程或进程需要运行时增加进程或线程的 CPU 利用率。调度程序通过每秒将该值除以2一次来使其衰减。对于sched\_other策略，CPU 利用率用于确定进程调度优先级。如果值较大，那么表示一个将耗用大量 CPU 资源的进程，该进程的优先级将更低；如果值较小，那么表示一个要执行大量 I/O 操作的进程，其优先级将更高。

PRI：（-l 和 l 标志）进程或内核线程的优先级；数字越大优先级越低。

WCHAN：（-l 标志）进程或内核线程为之等待或休眠的事件。对于内核线程，如果内核线程正在运行，该字段为空。 对于进程，如果只有一个内核线程正在休眠，等待通道定义为该休眠内核线程的等待通道；否则显示一个星号。

NI：（-l 和 l 标志）为 sched other 策略计算优先级中使用的细调值。

ADDR：通常情况下，（-l 和 l 标志）包含 进程栈的段号；如果为内核进程，那么为预处理数据区的地址。

SZ：（-l 和 l 标志）进程的核心映像大小（以 1 KB 为单位）。

TIME：（所有标志）进程的运行时间总和。如果运行时间达到 100 分钟，以 mm:ss 或 mmmm:ss 格式显示时间，这与使用 -o time 标志时的显示格式不同。

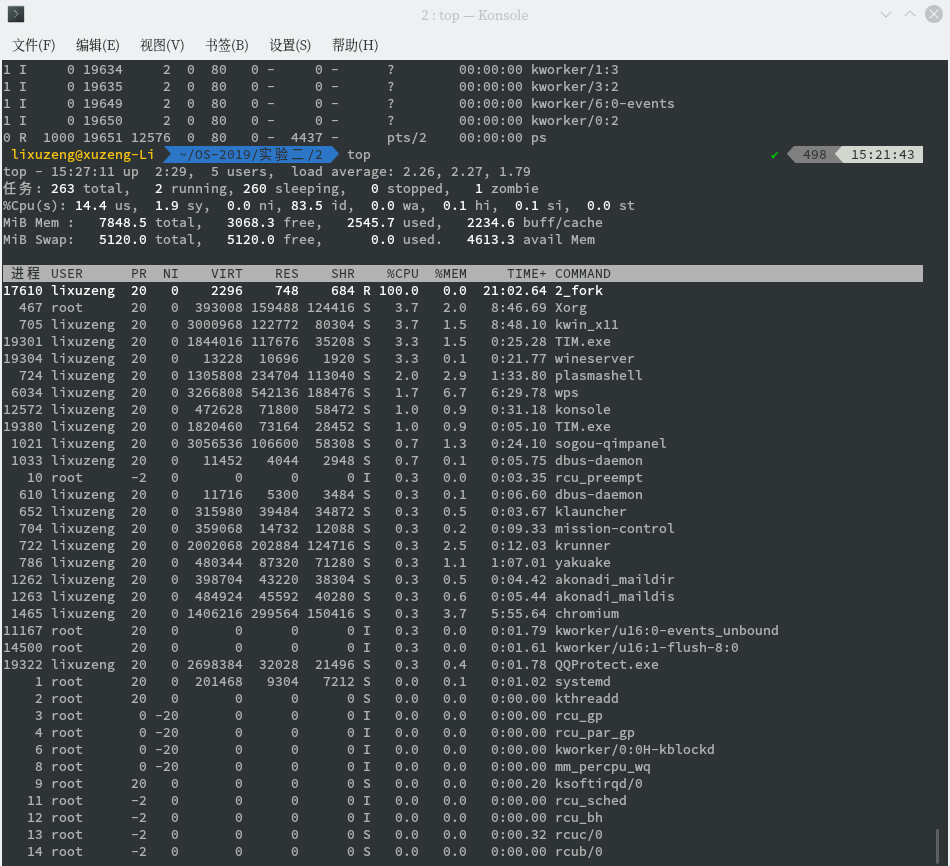
对应和vi进程相关的进程信息：



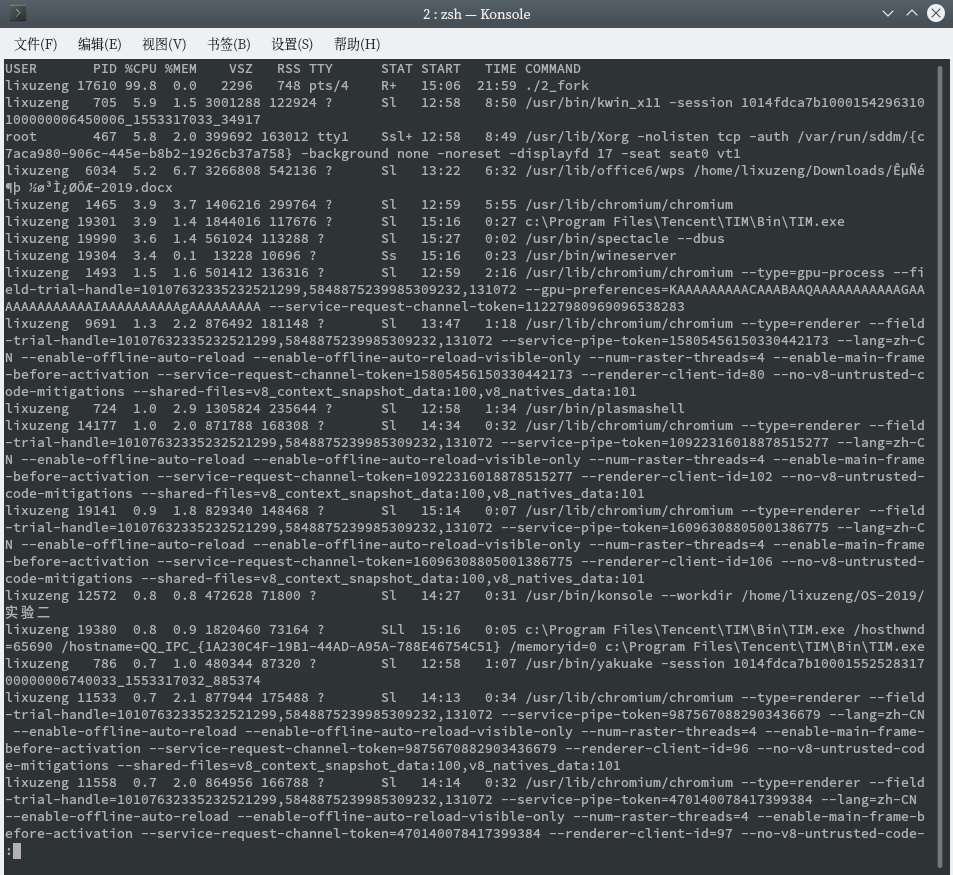




top命令实时显示系统中各个进程的资源占用状况：

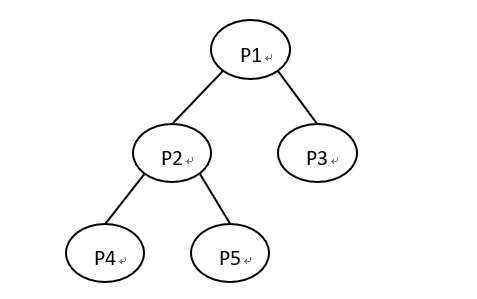


利用“ps -aux –sort -pcpu | less”命令根据cpu占用率进行降序排序：



从得到的结果可以看出父进程在进行不断循环cpu占用率高达99.8%。

**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。**



程序代码：

#include<unistd.h>

#include<stdio.h>

int main()

{

pid\_t fpid = fork();

if(fpid == 0)

{

printf("我是子进程 P3 ，我的pid为：%d,我的ppid为：%d\n",getpid(),getppid());

}

else

{

printf("我是父进程 P1 ，我的pid为：%d,我的ppid为：%d\n",getpid(),getppid());

fpid = fork();

if(fpid == 0)

{

printf("我是子进程 P2 ，我的pid为：%d,我的ppid为：%d\n",getpid(),getppid());

fpid = fork();

if(fpid == 0)

{

printf("我是孙子进程 P4， 我的pid为：%d,我的ppid为：%d\n",getpid(),getppid());

}

else

{

fpid = fork();

if(fpid == 0)

{

printf("我是孙子进程 P5 ,我的pid为：%d,ppid为：%d\n",getpid(),getppid());

}

else

{

}

}

}

else

{

}

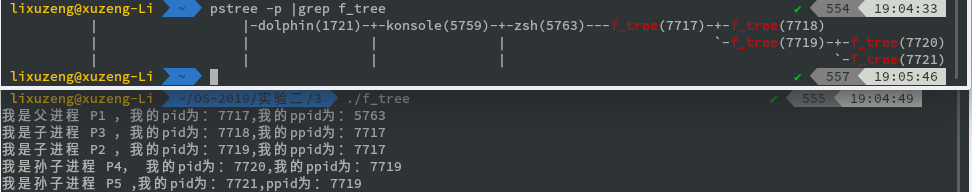
}

sleep(20);

return 0;

}

运行结果：



一般来说，在 fork() 之后是父进程先执行还是子进程先执行是不确定的。这取决于内核所使用的调度算法。（然而实验多次并为出现次序改变）

**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。**

程序代码：

#include<unistd.h>

#include<stdlib.h>

#include<stdio.h>

int main()

{

pid\_t fpid = fork();

if(fpid == 0)

{

while(1){

printf("我是子进程 P3 ，我的pid为：%d,我的ppid为：%d\n",getpid(),getppid());

sleep(2);

}

}

else

{

fpid = fork();

if(fpid == 0)

{

fpid = fork();

if(fpid == 0)

{

while(1){

printf("我是孙子进程 P4， 我的pid为：%d,我的ppid为：%d\n",getpid(),getppid());

sleep(2);

}

}

else

{

fpid = fork();

if(fpid == 0)

{

while(1){

printf("我是孙子进程 P5 ,我的pid为：%d,ppid为：%d\n",getpid(),getppid());

sleep(2);

}

}

else

{

}

}

while(1){

printf("我是子进程 P2 ，我的pid为：%d,我的ppid为：%d\n",getpid(),getppid());

sleep(2);

int \*ptr = (int \*)0;

\*ptr = 100;

}

}

else

{

}

while(1){

printf("我是父进程 P1 ，我的pid为：%d,我的ppid为：%d\n",getpid(),getppid());

sleep(2);

}

}

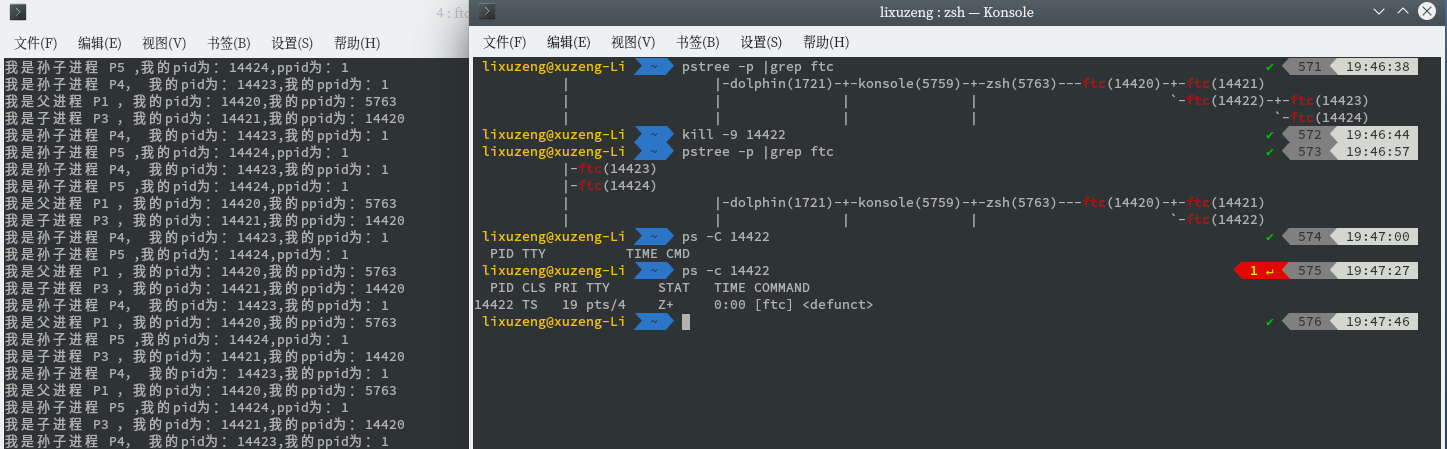
sleep(20);

return 0;

}

运行结果：

1>使用kill -9 进程号命令将P2进程终止：



P1正常运行，从ps命令可以看出P2进程变为僵尸进程（stat状态为z，zombie），其子进程P4,P5仍在运行，但是变为pid为1的子进程，相对于P1的进程树不显示P4,P5。

2>使用exit()使进程P2正常退出：

修改代码：

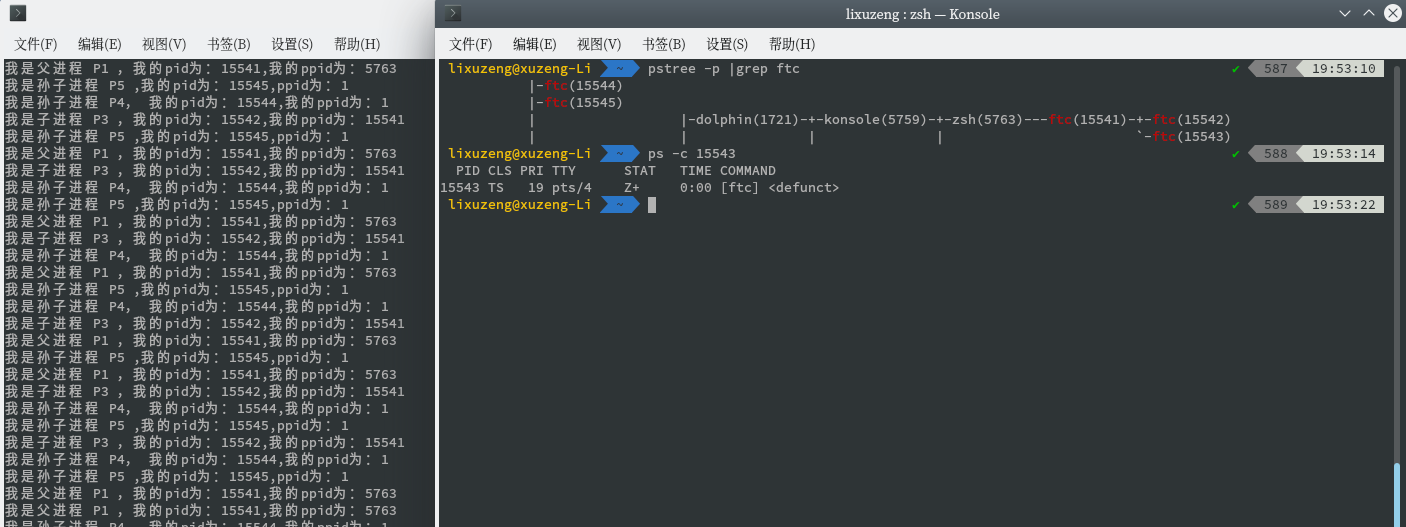
while(1){

printf("我是子进程 P2 ，我的pid为：%d,我的ppid为：%d\n",getpid(),getppid());

sleep(2);

exit(1)

}



P1正常运行，同使用kill命令杀死进程相同P2进程变为僵尸进程（stat状态为z，zombie），其子进程P4,P5仍在运行，但是变为pid为1的子进程，相对于P1的进程树不显示P4,P5。

3>采用段错误方式退出

段错误产生原因：

1、访问不存在的内存地址；

2、访问系统保护的内存地址

3、访问只读的内存地址

4、栈溢出

这里我采用访问系统保护的地址使段错误产生：

while(1){

printf("我是子进程 P2 ，我的pid为：%d,我的ppid为：%d\n",getpid(),getppid());

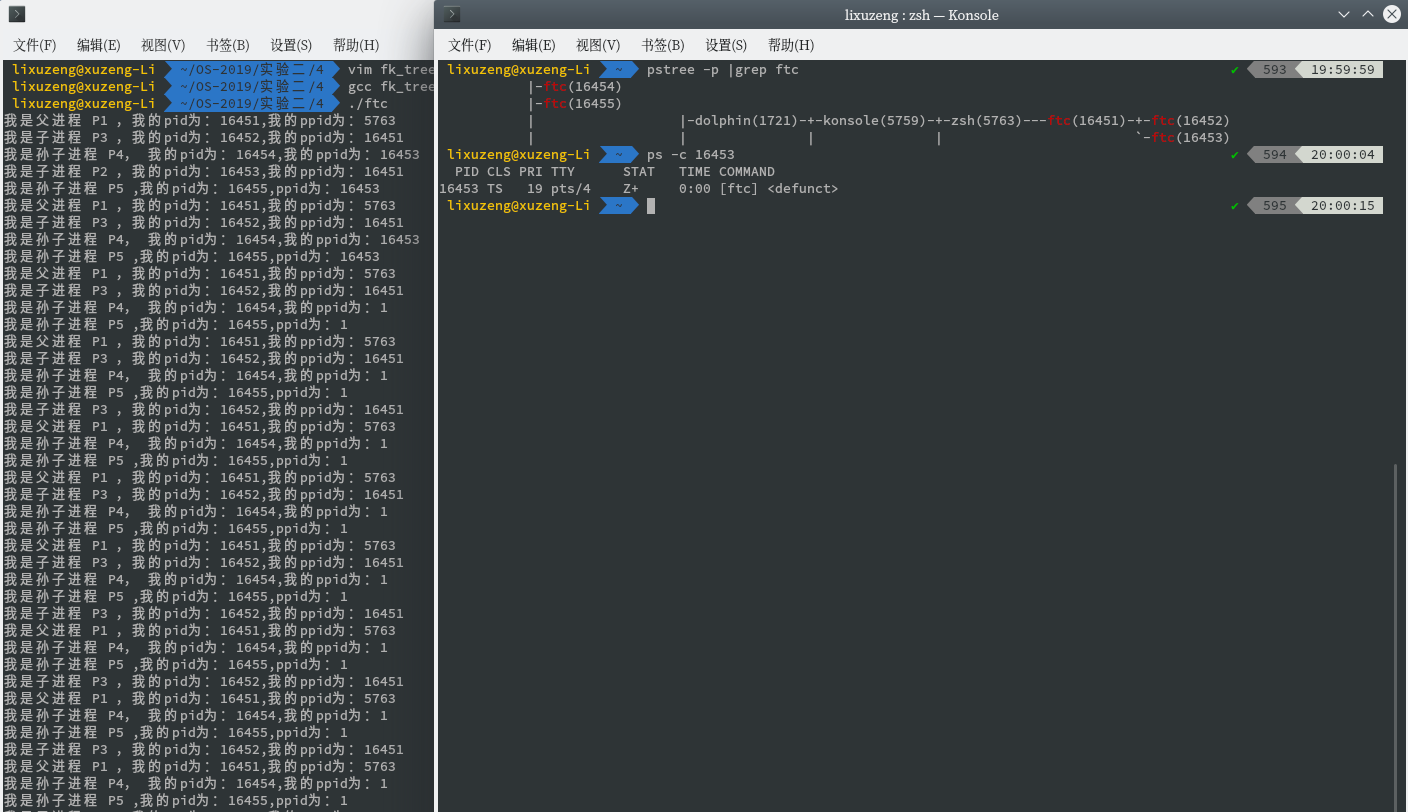
sleep(2);

int \*ptr = (int \*)0;

\*ptr = 100;

}

运行结果：



P1正常运行，同使用前两种方式结束P2进程相同，P2进程变为僵尸进程（stat状态为z，zombie），其子进程P4,P5仍在运行但是变为pid为1的子进程，相对于P1的进程树不显示P4,P5。