

# 北 京 林 业 大 学

## 2017 学年—2018 学年第 二 学期 Linux 应用实验报告书

专 业： 计 算 机 科 学 与 技 术 ( 创 新 实 验 班 )

班 级： 计创 16

姓 名： 陈楠 学 号： 161002107

实验地点： 计算中心 N09 任课教师： 李群

实验题目： Linux 进程管理

实验环境： Linux 操作系统

实验目的、实现内容、实验结果及结论分析等：

### 一．实验目的：

1. 掌握 Linux 前后与后台进程控制操作；
2. 掌握利用进程监控工具来维护系统的正常运行；
3. 掌握 Linux 的进程创建，理解进程创建后两个并发进程的执行。

### 二．实验内容：

#### 1. 管理进程：

- (1) 打开终端，输入命令“yes”，然后空格输入“test process”后回车。命令的执行结果是重复打印字符串“test process”。按键“Ctrl+z”中断，暂停前台进程的运行，转为挂起进程。

输入命令“jobs”，记录看到的结果。

- (2) 把挂起的进程转为前台进程执行：“fg 1”

再次按键“Ctrl+z”中断，重新挂起进程

输入命令“jobs -l”，记录看到的结果，进程号是多少？

(3) 把挂起的进程以后台方式运行：“bg 1”

此时能否再次按键“Ctrl+z”中断？为什么？

用鼠标点击关闭中断窗口。

(4) 打开终端，输入命令“xclock &”，记录下时钟时间。

输入命令“xcalc &”

输入命令“jobs”，记录看到的结果；

输入命令“ps”，记录看到的结果；

输入命令“fg %1”将时钟进程转到前台运行；

按键“Ctrl+z”将时钟进程挂起，记录时钟的时间，经过二~三分钟，观察时钟有走动吗？

(5) 输入命令“kill %2”杀死计算器进程，看计算器是否消失？

(6) 查看进程树，并在进程树中显示进程号（pstree -p）。

(7) 杀死 bash 进程，发生了什么？（kill -9 （进程号））

(8) 图形界面下的进程控制和系统维护：

a) 点击“主菜单/系统工具/系统监视器”；

b) 查看标签页“进程列表”，观察各进程的状态；

注意观察进程名、用户、内存、%CPU、ID 等选项；点击各项目旁“▼”或“▲”按钮，降序或升序排列。

c) 点击“进程列表”下的“查看”中的“活动的进程”；

d) 切换到终端键入命令“yes test process”

e) 切换到“进程列表”窗口，观察“活动的进程”有什么变化？

f) 点击标签页“系统监视器”，观察“%CPU 使用历史”、“内存/交换使用历史”、“设备”项目内容；

g) 此时 CPU 的利用率很高，是由哪个进程引起的？

h) 再次查看标签页“进程列表”中的“活动进程”，鼠标选中那个非常“活跃”的进程，右键快捷方式杀死该进程。

## 2. 进程创建

(1) 编写一段程序，实现父进程创建一个子进程，返回后父子进程分别循环输出进程号，例如“I am parent ,my ID is.....”及“I am child ,my ID is .....”循环

3 次，每次输出后使用 `sleep(1)` 延时 1 秒，然后再进入下一次循环，观察程序运行的结果。

(2) 修改上述程序，使用 `exit()` 和 `wait()` 实现父进程和子进程同步，其同步方式为父进程等待子进程的同步，即：子进程循环输出 3 次，然后父进程再循环输出 3 次。观察程序运行的结果。

实验指导：

本实验相关函数：

#### 1. `fork()` 函数

`fork()` 函数创建一个新进程。

其中返回值 `int` 取值：

等于 0：创建子进程，从子进程返回的 ID 值；

大于 0：从父进程返回子进程的 ID 值。

等于 -1：进程创建失败。

#### 2. `sleep(时间)`，作用是延时，程序暂停若干时间。

Linux 下（使用的 `gcc` 的库），`sleep()` 函数是以秒为单位。

#### 3. `wait()` 函数，常用来控制父进程与子进程的同步。父进程中调用 `wait()` 函数，则父进程被阻塞，进入等待队列，等待子进程结束。当子进程结束，会产生一个终止状态字，系统会向父进程发出信号。当接到信号后，父进程提取子进程的终止状态字，从 `wait()` 函数返回继续执行原程序。

系统调用格式：

```
int wait(status)
```

```
int *status;
```

正确返回：大于 0：子进程的进程 ID 值；

等于 0：其它。

错误返回：等于 -1：调用失败。

#### 4. `exit()`，终止进程的执行。

调用方式：

```
exit(status)
```

```
int status;
```

其中，`status` 是返回给父进程的一个整数，以备查考。

1. 通常父进程在创建子进程时，应在进程的末尾安排一条 `exit()`，使子进程自我终止。`exit(0)`表示进程正常终止，`exit(1)`表示进程运行有错，异常终止。

### 三. 实验结果:

### 1. 管理进程:

- (1) 打开终端，输入命令“yes”，然后空格输入“test process”后回车。命令的执行结果是重复打印字符串“test process”。按键“Ctrl+z”中断，暂停前台进程的运行，转为挂起进程。

输入命令“jobs”，记录看到的结果。

A screenshot of a Linux terminal window. The title bar at the top reads "root@Linux:". Below it is a menu bar with options: "文件(F)", "编辑(E)", "查看(V)", "终端(T)", "转到(G)", and "帮助(H)". The main area of the terminal shows a series of commands and their outputs:

```
test process  
test process  
test process  
test process  
test process  
test process  
test process  
test process  
test process  
test process  
test process  
test process  
test process  
test process  
test process  
test process  
test process  
test process  
test process  
[1]+ Stopped yes test process  
[root@linux root]# jobs  
[1]+ Stopped yes test process  
[root@linux root]#
```

The cursor is visible at the end of the last command line.

- (2) 把挂起的进程转为前台进程执行：“fg 1”

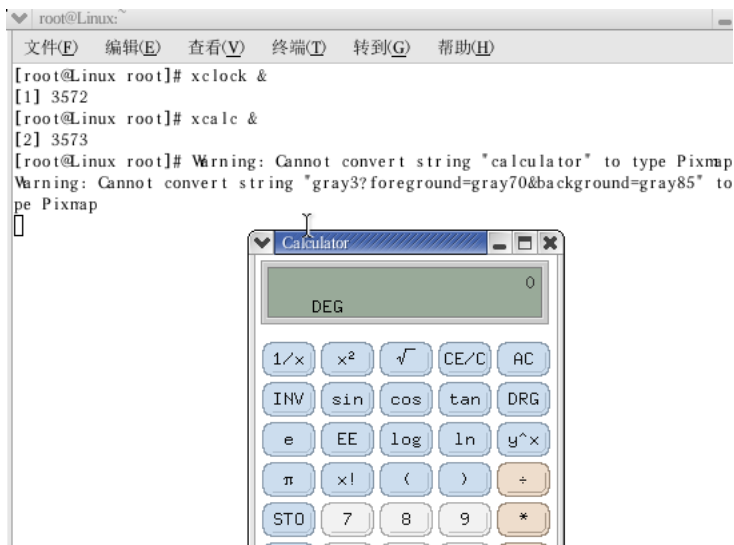
再次按键“Ctrl+z”中断，重新挂起进程

输入命令“jobs -l”，记录看到的结果，进程号是多少？





输入命令“xcalc &”



输入命令“jobs”，记录看到的结果：

```
[root@Linux root]# jobs
[1]-  Running                  xclock &
[2]+  Running                  xcalc &
[root@Linux root]#
```

输入命令“ps”，记录看到的结果：

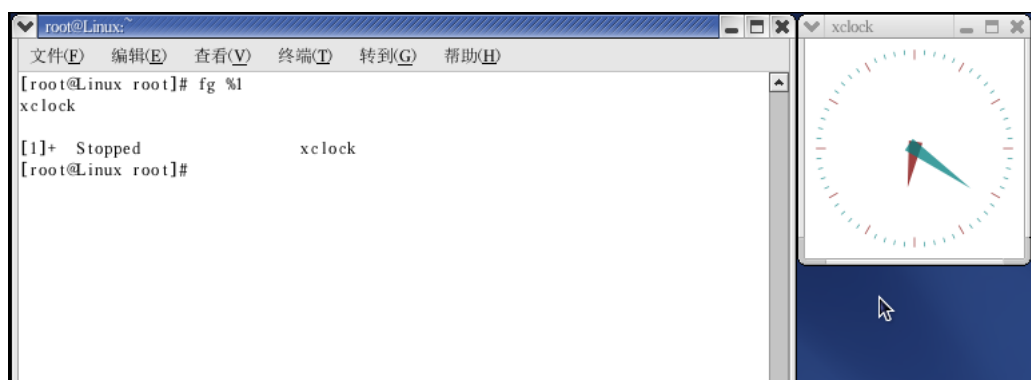
```
[root@Linux root]# ps
  PID TTY          TIME CMD
 3548 pts/0    00:00:00 bash
 3572 pts/0    00:00:00 xclock
 3573 pts/0    00:00:00 xcalc
 3575 pts/0    00:00:00 ps
[root@Linux root]#
```

输入命令“fg %1” 将时钟进程转到前台运行；



按键“Ctrl+z” 将时钟进程挂起，记录时钟的时间，经过二~三分钟，观察

时钟有走动吗？



没有走动

(5) 输入命令“kill %2”杀死计算器进程，看计算器是否消失？

```
[root@Linux root]# kill %2  
[2]- Stopped                  xcalc  
[root@Linux root]#
```

消失了

(6) 查看进程树，并在进程树中显示进程号（pstree -p）。



(7) 杀死 bash 进程，发生了什么？（kill -9 （进程号））

```
文件(F) 编辑(E) 查看(V) 终端(T) 转到(G) 帮助(H)
[root@Linux root]# kill -9 3548
```

终端消失了

(8) 图形界面下的进程控制和系统维护：

- a) 点击 “主菜单/系统工具/系统监视器”；
- b) 查看标签页 “进程列表”，观察各进程的状态；

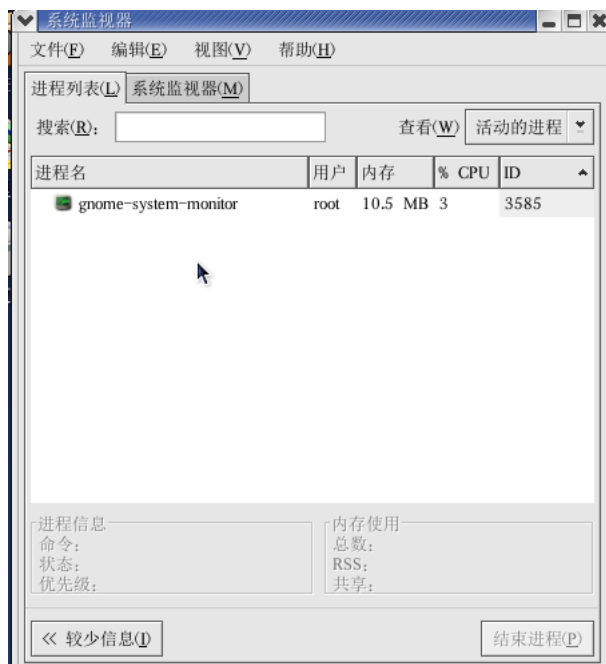
注意观察进程名、用户、内存、%CPU、ID 等选项；点击各项目旁 “▼” 或 “▲” 按钮，降序或升序排列。



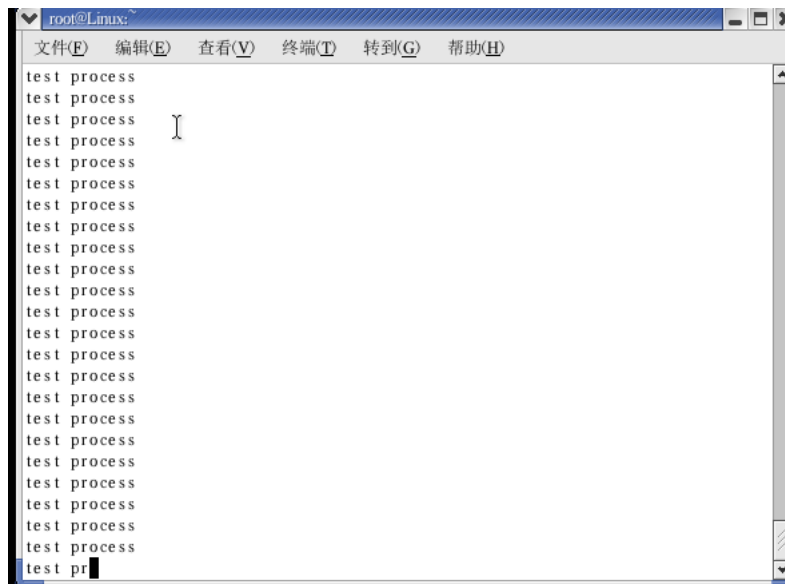




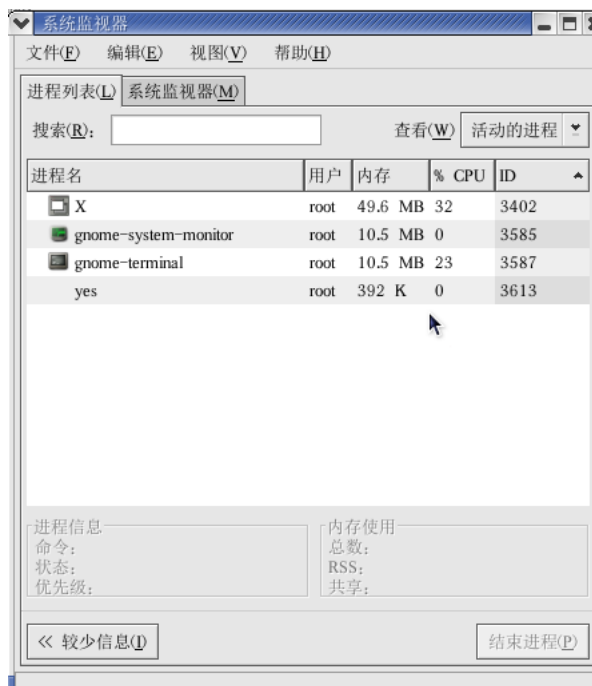
c) 点击“进程列表”下的“查看”中的“活动的进程”；



d) 切换到终端键入命令“yes test process”

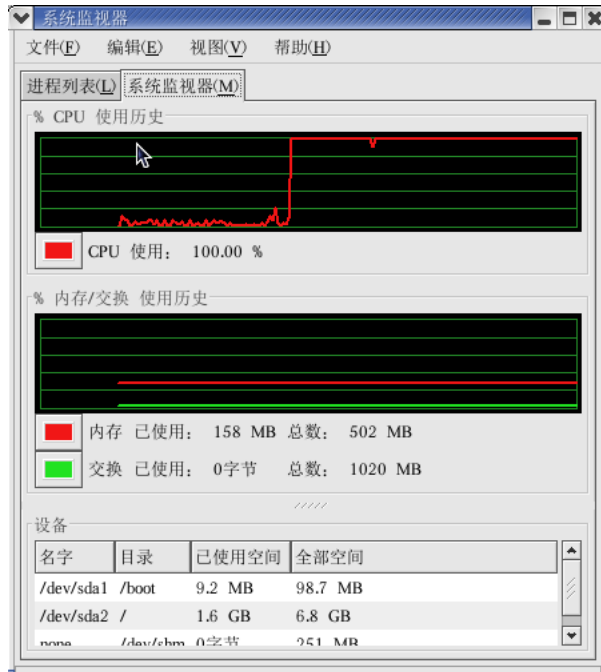


e) 切换到“进程列表”窗口，观察“活动的进程”有什么变化？

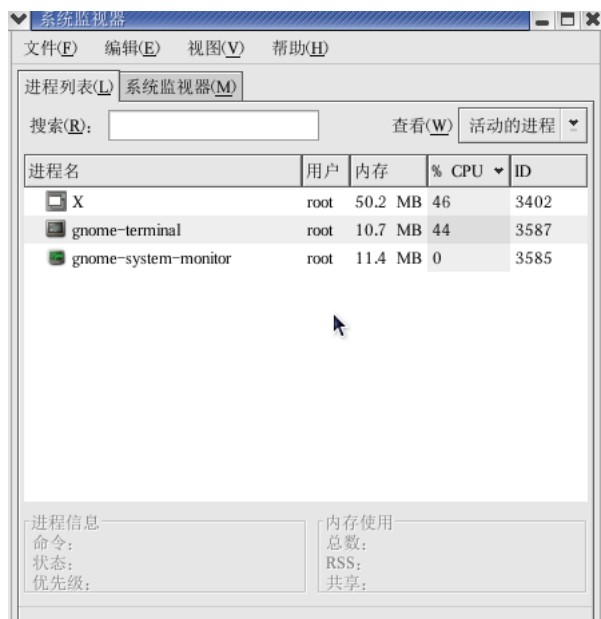


多了几个进程

f) 点击标签页“系统监视器”，观察“%CPU 使用历史”、“内存/交换使用历史”、“设备”项目内容；



g) 此时 CPU 的利用率很高，是由哪个进程引起的？

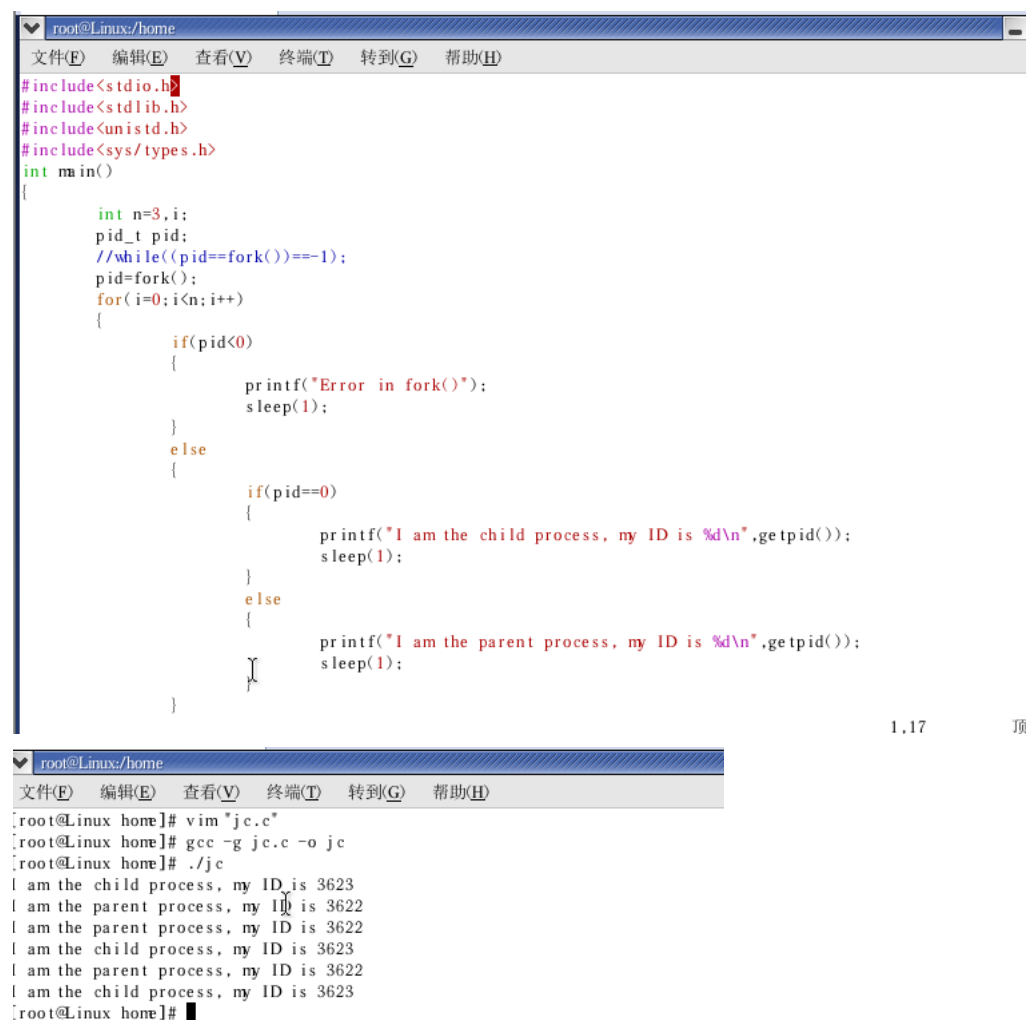


h) 再次查看标签页“进程列表”中的“活动进程”，鼠标选中那个非常“活跃”的进程，右键快捷方式杀死该进程。

杀死后 test process 窗口消失

## 2. 进程创建

(1) 编写一段程序，实现父进程创建一个子进程，返回后父子进程分别循环输出进程号，例如 “I am parent ,my ID is.....” 及 “I am child ,my ID is .....” 循环 3 次，每次输出后使用 sleep(1)延时 1 秒，然后再进入下一次循环，观察程序运行的结果。



```
root@Linux:/home
文件(F) 编辑(E) 查看(V) 终端(T) 转到(G) 帮助(H)

#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<sys/types.h>
int main()
{
    int n=3,i;
    pid_t pid;
    //while((pid==fork())==1);
    pid=fork();
    for(i=0;i<n;i++)
    {
        if(pid<0)
        {
            printf("Error in fork()");
            sleep(1);
        }
        else
        {
            if(pid==0)
            {
                printf("I am the child process, my ID is %d\n",getpid());
                sleep(1);
            }
            else
            {
                printf("I am the parent process, my ID is %d\n",getpid());
                sleep(1);
            }
        }
    }
}
```

1,17 顶

```
root@Linux:/home
文件(F) 编辑(E) 查看(V) 终端(T) 转到(G) 帮助(H)

[root@Linux home]# vim "jc.c"
[root@Linux home]# gcc -g jc.c -o jc
[root@Linux home]# ./jc
I am the child process, my ID is 3623
I am the parent process, my ID is 3622
I am the parent process, my ID is 3622
I am the child process, my ID is 3623
I am the parent process, my ID is 3622
I am the child process, my ID is 3623
[root@Linux home]#
```

(2) 修改上述程序，使用 exit()和 wait()实现父进程和子进程同步，其同步方式为父进程等待子进程的同步，即：子进程循环输出 3 次，然后父进程再循环输出 3 次。观察程序运行的结果。

```
root@Linux:/home
文件(F) 编辑(E) 查看(V) 终端(T) 转到(G) 帮助(H)
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<sys/types.h>
int main()
{
    int n=3,i;
    pid_t pid;
    pid=fork();
    for(i=0;i<n;i++)
    {
        if(pid<0)
        {
            printf("Error in fork()");
            sleep(1);
        }
        else
        {
            if(pid==0)
            {
                printf("I am the child process, my ID is %d\n",getpid());
                //exit(0);
            }
            else
            {
                wait();
                printf("I am the parent process, my ID is %d\n",getpid());
            }
        }
    }
}
*jc2.c* [已转换] 33L, 443C 27,5-33
```

```
root@Linux:/home
文件(F) 编辑(E) 查看(V) 终端(T) 转到(G) 帮助(H)
[root@Linux home]# vim jc2.c
[root@Linux home]# gcc -g jc2.c -o jc2
[root@Linux home]# ./jc2
I am the child process, my ID is 3652
I am the child process, my ID is 3652
I am the child process, my ID is 3652
I am the parent process, my ID is 3651
I am the parent process, my ID is 3651
I am the parent process, my ID is 3651
[root@Linux home]#
```

实验指导：

本实验相关函数：

## 5. fork()函数

fork()函数创建一个新进程。

其中返回值 int 取值：

等于 0：创建子进程，从子进程返回的 ID 值；

大于 0：从父进程返回子进程的 ID 值。

等于-1：进程创建失败。

## 6. sleep（时间），作用是延时，程序暂停若干时间。

Linux 下（使用的 gcc 的库），sleep()函数是以秒为单位。

## 7. wait（）函数，常用来控制父进程与子进程的同步。父进程中调用 wait()函数，则父进程被阻塞，进入等待队列，等待子进程结束。当子进程结束，会产生一个终止状态字，系统会向父进程发出信号。当接到信号后，父进程提

取子进程的终止状态字，从 `wait()` 函数返回继续执行原程序。

系统调用格式：

```
int wait(status)
```

```
int *status;
```

正确返回：大于 0：子进程的进程 ID 值；

等于 0：其它。

错误返回：等于 -1：调用失败。

8. `exit()`，终止进程的执行。

调用方式：

```
exit(status)
```

```
int status;
```

其中，`status` 是返回给父进程的一个整数，以备查考。

2. 通常父进程在创建子进程时，应在进程的末尾安排一条 `exit()`，使子进程自我终止。`exit(0)` 表示进程正常终止，`exit(1)` 表示进程运行有错，异常终止。

#### 四. 结论分析：

通过本次 Linux 进程管理实验，我对 Linux 下的进程管理有了更加深入地理解。掌握了 Linux 前后与后台进程控制操作；初步掌握并能够利用进程监控工具来维护系统的正常运行；尝试了 Linux 的进程创建，初步理解了进程创建后两个并发进程的执行过程。在 Linux 中，相关操作的具体实现只会更加复杂，因为要考虑的东西很多，所以需要我们更加努力地学习。