南京邮电大学

实 验 报 告

（ 2025/ 2026学年 第 2 学期）

课程名称	操作系统基础
实验名称	Linux 系统及进程创建
实验时间	2026 年 4 月 9 日
指导单位	计算机学院
指导教师	

		班级学号	
学院(系)	计算机学院	专 业	软件工程

实 验 报 告

实验名称	Linux系统及进程创建			指导教师	
实验类型	验证	实验学时	2	实验时间	2026.4.9

一. 实验目的和要求

1. 掌握Linux操作系统的操作和使用;
2. 掌握Linux下C语言的编辑、编译、运行的全过程;
3. 掌握进程创建系统调用的使用。

二、实验环境(实验设备)

Windows + VMWare + Ubuntu

三、实验原理及内容

本次实验主要内容如下:

(1) 熟悉 Ubuntu 操作系统的桌面、终端和文本编辑器,练习 pwd、ls、cd、ps、cp、kill 等常用命令。

(2) 在 Linux 下使用 gedit 或 vi 编辑 C 语言源程序,使用 gcc 编译,并运行可执行文件。

(3) 编写程序,使用 fork() 系统调用创建子进程,分别观察进程顺序输出和加入 sleep 后的并发交叉输出。

(4) 编写多次 fork() 的程序,使用 getpid() 和 getppid() 分析进程关系树。

(5) 观察并理解孤儿进程和僵尸进程的产生过程。

实 验 报 告

2. 编辑一个简单的 C 语言程序 hello.c,并编译运行。

(1) 使用 gedit 文本编辑器创建源文件 hello.c,代码如下:

```
#include <stdio.h>

int main()
{
    printf("hello, world!\n");
    return 0;
}
```

(2) 在终端中使用 gcc 编译,然后执行生成的可执行文件。其中 -o 选项指定输出文件名,不加 -o 时默认输出 a.out。

```
root@duanwh-virtual-machine:~# pwd
/root
root@duanwh-virtual-machine:~# ls
a.out      fork1.c   fork2.c   hello.c   count2.c
root@duanwh-virtual-machine:~# ls -l hello.c
-rw-r--r-- 1 root root 74 2025-10-20 09:59 hello.c
root@duanwh-virtual-machine:~# cd /home
root@duanwh-virtual-machine:~# pwd
/home
root@duanwh-virtual-machine:~#
```

(3) 编译运行 hello.c,屏幕输出 "hello, world!"。

```
root@duanwh-virtual-machine:~# cat hello.c
#include <stdio.h>
int main()
{
    printf("hello, world!\n");
    return 0;
}
root@duanwh-virtual-machine:~# gcc hello.c -o hello.out
root@duanwh-virtual-machine:~# ls -l hello.out
-rwxr-xr-x 1 root root 8520 2025-10-20 10:05 hello.out
root@duanwh-virtual-machine:~# ./hello.out
hello, world!
root@duanwh-virtual-machine:~#
```

3. 使用 fork() 创建子进程,观察并发执行。

(1) 编写 fork1.c,在 main 函数中两次调用 fork() 创建一个女儿进程(daughter)和一个儿子进程(son),三个进程各自打印 8 次循环信息。

```
/* fork1.c */
#include <stdio.h>
int main()
{
    int p1, p2, i;
    while ((p1 = fork()) == -1);
    if (p1 == 0) {
        for (i = 0; i < 8; i++) printf("daughter %d \n", i);
    } else {
        while ((p2 = fork()) == -1);
        if (p2 == 0) {
            for (i = 0; i < 8; i++) printf("son %d \n", i);
        } else {
            for (i = 0; i < 8; i++) printf("parent %d \n", i);
        }
    }
    return 0;
}
```

运行结果:

```
root@duanwh-virtual-machine:~# gcc fork1.c -o fork1.out
root@duanwh-virtual-machine:~# ./fork1.out
parent 0
parent 1
parent 2
parent 3
parent 4
parent 5
parent 6
parent 7
son 0
son 1
son 2
son 3
son 4
son 5
son 6
son 7
daughter 0
daughter 1
daughter 2
daughter 3
daughter 4
daughter 5
daughter 6
daughter 7
root@duanwh-virtual-machine:~#
```

可以看到三个进程的输出基本上是 parent、son、daughter 各自连成一段,没有交叉。这是因为每次循环很短,某个进程在它的时间片内就把 8 行打印完了,看上去像顺序执行。

(2) 修改程序,在每次 printf 后加入 sleep(4),强迫进程让出 CPU,使得三个进程的输出明显交叉,这才真正体现进程并发执行的特性。

```
/* fork2.c —— 仅展示子进程关键修改 */
for (i = 0; i < 8; i++) {
    printf("daughter %d \n", i);
    sleep(4);
}
```

运行结果:

```
root@duanwh-virtual-machine:~# gcc fork2.c -o fork2.out
root@duanwh-virtual-machine:~# ./fork2.out
daughter 0
parent 0
son 0
daughter 1
parent 1
son 1
daughter 2
parent 2
son 2
daughter 3
parent 3
son 3
daughter 4
parent 4
son 4
daughter 5
parent 5
son 5
daughter 6
parent 6
son 6
daughter 7
parent 7
son 7
root@duanwh-virtual-machine:~#
```

结果与 fork1 完全不同, daughter、parent、son 三个进程的输出依次交替出现,这才是进程并发执行的真实表现。这说明进程的执行顺序具有不确定性,即操作系统的异步性。

4. 多次 fork 与进程关系树。

连续调用三次 fork() 会产生 $2^3 = 8$ 个进程。下面的程序通过 getpid() 和 getppid() 打印每个进程及其父进程的进程号。

```
/* count2.c */
#include <stdio.h>
#include <stdlib.h>
int main()
{
    fork();
    fork();
    fork();
    printf("Hello, world! from process id %d, it's parent id %d\n",
            getpid(), getppid());
    wait(0); wait(0); wait(0);
    return 0;
}
```

某一次运行的结果如下:

```
root@duanwh-virtual-machine:~# gcc count2.c -o count2.out
root@duanwh-virtual-machine:~# ./count2.out
Hello, world! from process id 17361, it's parent id 16421
Hello, world! from process id 17362, it's parent id 17361
Hello, world! from process id 17363, it's parent id 17361
Hello, world! from process id 17364, it's parent id 17361
Hello, world! from process id 17365, it's parent id 17362
Hello, world! from process id 17366, it's parent id 17362
Hello, world! from process id 17367, it's parent id 17363
Hello, world! from process id 17368, it's parent id 17365
root@duanwh-virtual-machine:~#
```

整理得到进程号 -> 父进程号的关系:

17361 -> 16421(bash);17362、17363、17364 -> 17361;17365、17366 -> 17362;17367 -> 17363;17368 -> 17365。

据此可以画出一棵三层的进程关系树,共 8 个进程,符合预期。父进程末尾使用 wait(0) 是为了等待所有子进程结束,避免产生僵尸或孤儿进程。

5. 孤儿进程。

当父进程先于子进程结束时,子进程被托管给 init 进程(进程号为 1),即成为孤儿进程。下面程序在子进程中 sleep 2 秒后再打印一次 ppid,通常可以看到第二次打印的 ppid 已变为 1。

```
/* orphan.c */
pid = fork();
if (pid == 0) {
    printf("child pid=%d, ppid=%d\n", getpid(), getppid());
    sleep(2);
    printf("child pid=%d, ppid=%d\n", getpid(), getppid());
} else {
    printf("parent pid=%d, ppid=%d\n", getpid(), getppid());
}
```

运行结果:

```
root@duanwh-virtual-machine:~# ./orphan.out
parent pid = 23331, ppid = 23162
root@duanwh-virtual-machine:~# child pid = 23332, ppid = 1
child pid = 23332, ppid = 1
root@duanwh-virtual-machine:~#
```

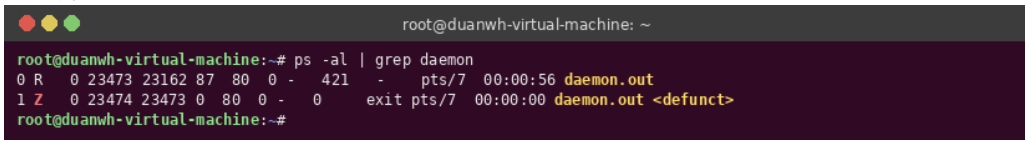
父进程先返回,子进程 sleep 后被 init 进程接管,所以第二次打印时 ppid 已是 1。

6. 僵尸进程。

子进程已经结束,但父进程没有调用 wait/waitpid 回收其退出状态,这时子进程就以僵尸(zombie)状态留在进程表里。下面的程序在父进程中故意写一个空的死循环来制造僵尸子进程。

```
/* daemon.c */
pid = fork();
if (pid > 0) { while (1); }        /* 父进程一直空转 */
else { printf("child pid=%d\n", getpid()); }
```

在另一个终端中执行 ps -al | grep daemon,可以看到子进程的状态(STAT)为 Z,即僵尸进程,命令后面跟有 <defunct> 标记。



实 验 报 告

四、实验小结（包括问题和解决方法、心得体会、意见与建议等）

本次实验是操作系统课程的第一个上机实验。通过在 VMware 虚拟机上的 Ubuntu 环境中进行各项操作,熟悉了 Linux 的基本使用,并掌握了进程创建的相关系统调用。主要收获如下:

1. 熟悉了 Ubuntu 桌面、终端和 gedit 文本编辑器的使用,练习了 pwd、ls、cd、ps、cp、kill 等常用 Shell 命令。

2. 掌握了 Linux 下 C 程序的开发流程:用 gedit 编写源代码,用 gcc 编译,用 ./a.out 执行,理解了 -o 选项以及当前目录前缀 ./ 的含义。

3. 通过 fork1.c 和 fork2.c 直观看到了进程并发执行的特点。 fork1 的输出几乎是顺序的,但加入 sleep 后, daughter / parent / son 的打印明显交错,体现了进程执行顺序的异步性和不确定性。

4. 通过三次 fork 的 count2 程序,亲手画出了 8 个进程的进程关系树,理解了 fork 的指数式生成规律。

5. 编写了制造孤儿进程和僵尸进程的小程序,观察到孤儿进程被 init(pid=1)接管, 以及僵尸进程在 ps 中显示为 Z 状态加 <defunct>。这让我对父子进程的生命周期、wait 调用的作用有了更具体的认识。

问题及解决:

1. 编译时提示 "command not found":刚开始没有用 ./ 前缀,直接输入 a.out,系统在 PATH 中找不到。加上 ./ 后正常执行。

2. fork1 程序输出顺序看上去是顺序执行的,一开始误以为是单进程。后来在每次 printf 后加 sleep(4),输出立刻变成交错的,才理解 "顺序" 只是因为每个进程时间片足够长,并非真的串行。

意见与建议:

1. 希望后续的进程实验能配合调度算法一起练习,看到调度的实际效果。

2. 实验文档中如能补充 ps、kill 等命令的常用选项速查表,会更方便上手。

五、指导教师评语

成　绩		批阅人		日　期	