



# 操作系统原理 实验报告

## (实验一) 进程的创建实验

学 院 名 称 : 数据科学与计算机学院

专业 (班级) : 17 级软件工程 2 班

学 生 姓 名 : 许滨楠

学 号 : 17343131

时 间 : 2019 年 3 月 27 日

## 一. 实验目的和要求

- 加深对进程概念的理解，明确进程和程序的区别。进一步认识并发执行的实质。
- 认识进程生成的过程，学会使用fork生成子进程，并知道如何使子进程完成与父进程不同的工作。

## 二. 实验内容

### 1.将下面的程序编译运行，并解释现象。

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

#include < stdio.h >

#include < unistd.h >

int main(){

    int pid1=fork();

    printf( "***1**\n" );

    int pid2=fork();

    printf( "***2**\n" );

    if(pid1==0){int pid3=fork();printf( "***3**\n" );}

    else printf( "***4**\n" );

    return 0;

}
```

### 2.通过实验完成第三章习题3.4。

3.编写一段程序，使用系统调用fork()创建两个子进程。当此程序运行时，在系统中有一个父进程和两个子进程活动。让每一个进程在屏幕上显示一个字符；父进程显示字符

“a”；子进程分别显示字符“b”和字符“c”。试观察记录屏幕上的显示结果，并分析原因。

### 三. 实验方案

实验原理：Linux环境基本配置和使用；程序的编译、运行、调试原理等

实验工具和环境：Mac平台Parallels Desktop软件——用于安装配置和运行虚拟机；

Ubuntu Linux虚拟机——镜像文件作为虚拟机安装的源文件；Linux下基本应用（如

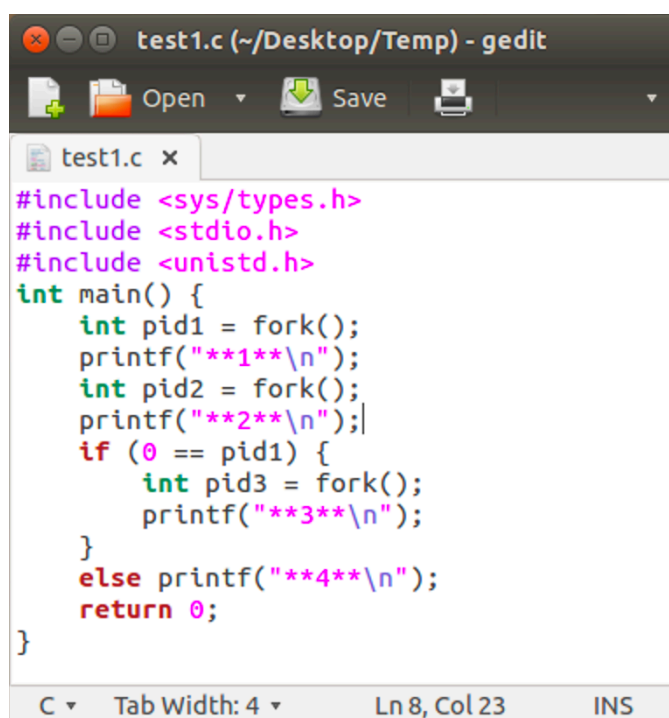
Terminal、Emac编辑器等）——完成代码的编辑和编译、链接、运行；

按照资料指导流程完成相关实验，具体过程见四.实验过程和结果。

### 四. 实验过程和结果

#### 1. 实验过程：

- 在Ubuntu Linux平台下，用gedit编辑实验内容中的代码：



```
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>
int main() {
    int pid1 = fork();
    printf("**1**\n");
    int pid2 = fork();
    printf("**2**\n");
    if (0 == pid1) {
        int pid3 = fork();
        printf("**3**\n");
    }
    else printf("**4**\n");
    return 0;
}
```

- 在终端对该代码文件进行编译、运行，查看结果：

```

pie@ubuntu: ~/Desktop/Temp
pie@ubuntu ~ ➤ cd Desktop/Temp
pie@ubuntu Desktop/Temp ➤ g++ test1.c -o test1.o
pie@ubuntu Desktop/Temp ➤ ./test1.o
**1**
**2**
**4**
**2**
**4**
**1**
**2**
**3**
**2**
**3**
**3**
**3**
pie@ubuntu Desktop/Temp ➤

```

- 结果分析：

- I. 我们称这个程序运行时的进程为主进程，也为进程1。在主进程的第一个fork()函数执行之后，该进程生成了一个子进程，我们称为进程2。此时主进程和其子进程 - 进程1 & 2都从第一个fork()之后执行；（当前活跃进程：1 2）
- II. 进程1继续占用CPU，**输出"1"**，之后进程1通过第二个fork()语句创建了另外一个子进程，称为进程3，进程1 & 3都从第二个fork语句之后开始执行；（当前活跃进程：1 2 3）
- III. 进程1继续占用CPU，**输出"2"**并进入条件判断语句，由于主进程中，fork()返回的值为其创建的子进程的进程id，所以主进程中的pid1不为0，跳转到else段，**输出"4"**，进程1基本代码执行结束；（当前活跃进程：2 3）
- IV. 进程3占用CPU，从第二次fork()之后开始执行，**输出"2"**并进入条件判断语句，由于进程3是进程1的子进程且未执行过第一次fork()，所以进程3中pid1的值同进程1，为进程2的进程id，不为0，故亦跳转到else段，**输出"4"**，进程3基本代

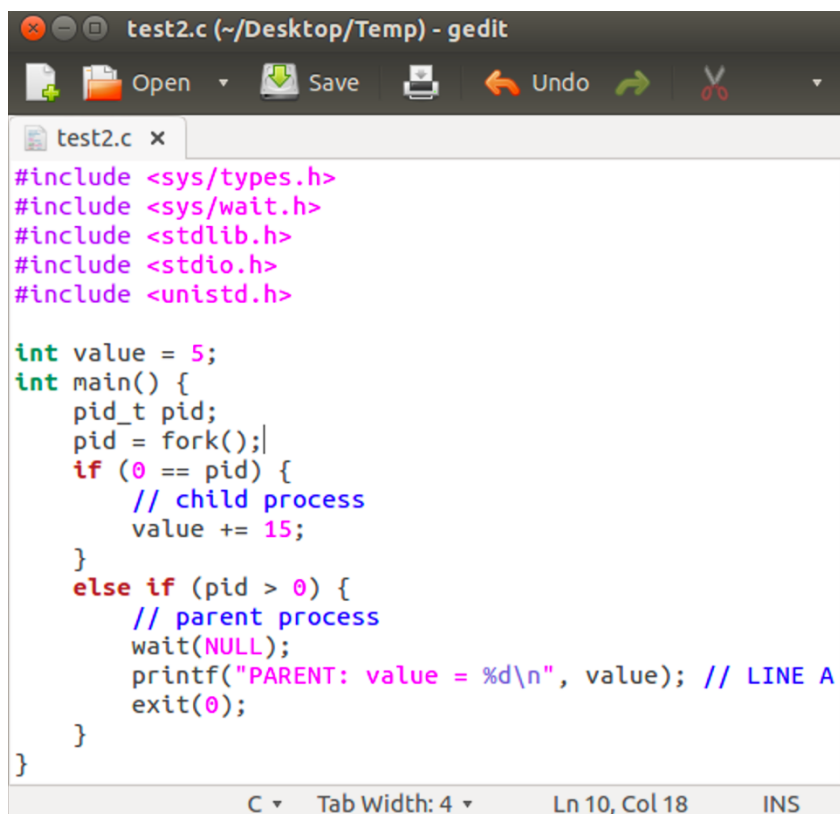
- 码执行结束；（当前活跃进程：2）
- V. 进程2占用CPU，从第一次fork()之后开始执行，**输出"1"**并进行下一次fork()，创建了另一个子进程，称为进程4，进程2 & 4都从第二次fork()之后开始执行；（当前活跃进程：2 4）
- VI. 进程2继续占用CPU，**输出"2"**并进入条件判断语句，由于进程2是进程1创建的子进程，该fork()函数的返回值为0，此时pid1 == 0成立，进入if代码段，通过第三个fork()语句创建了另外一个子进程，称为进程5，进程2 & 5都从第三个fork()语句之后开始执行，且进程2继续占用CPU，**输出"3"**后完成if代码段的执行，进程2基本代码段执行结束；（当前活跃进程：4 5）
- VII. 进程4占用CPU，**输出"2"**并进入条件判断语句，与前述同理，进程4的pid1与其父进程进程2相同，为0，此时pid1 == 0成立，进入if代码段，通过第三个fork()创建了另一个子进程，称为进程6，进程4 & 6都从第三个fork()语句之后开始执行，且进程4继续占用CPU，**输出"3"**后完成if代码段的执行，进程4基本代码段执行结束；（当前活跃进程：5 6）
- VIII. 进程5 & 6都从第三次fork()代码段开始执行，分别在之后，**输出"3"**并结束执行，因此最后输出了两次**"3"**。

## 2. 第三章习题3.4:

· 推测：父进程在fork之后继续占用CPU，因其pid为子进程id不为0，进入else代码段，执行wait语句时将自身进程转为等待态，让子进程占用CPU，子进程中pid为0，进入if代码段，value值自增15，变为20，但父子进程只共享代码段，子进程中的value变量在改变时已经完成拷贝，不再影响父进程value的值，其代码结束执行。接着父进程重新占用CPU，输出

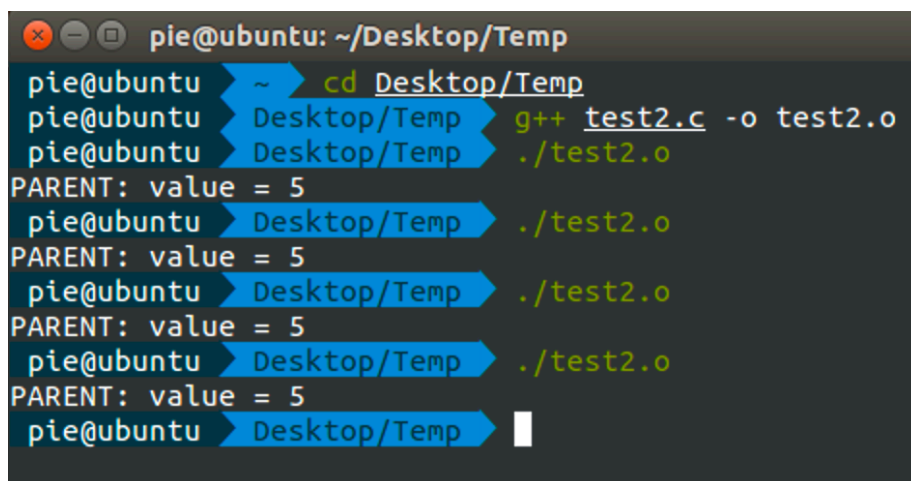
value值，仍然为进程中全局定义的5，输出"PARENT: value = 5"。

- 与1.中相同，在Linux平台下编辑、编译、多次运行代码，查看结果：



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

int value = 5;
int main() {
    pid_t pid;
    pid = fork();
    if (0 == pid) {
        // child process
        value += 15;
    }
    else if (pid > 0) {
        // parent process
        wait(NULL);
        printf("PARENT: value = %d\n", value); // LINE A
        exit(0);
    }
}
```

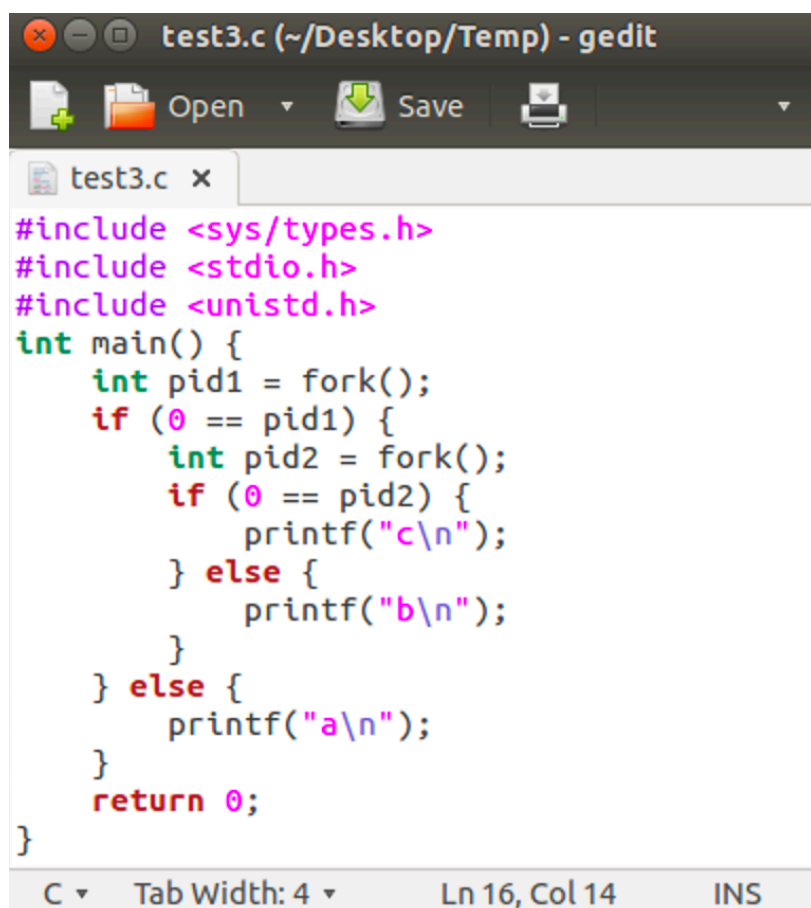


```
pie@ubuntu: ~/Desktop/Temp
pie@ubuntu ~$ cd Desktop/Temp
pie@ubuntu Desktop/Temp$ g++ test2.c -o test2.o
pie@ubuntu Desktop/Temp$ ./test2.o
PARENT: value = 5
pie@ubuntu Desktop/Temp$ ./test2.o
PARENT: value = 5
pie@ubuntu Desktop/Temp$ ./test2.o
PARENT: value = 5
pie@ubuntu Desktop/Temp$ ./test2.o
PARENT: value = 5
pie@ubuntu Desktop/Temp$
```

- （因为环境不同，对代码做了一些兼容性的调整，主体内容不变）结果符合预期。

### 3. 编程实现三个进程同时活动与每个进程的相应输出：

- 根据题意，完成程序代码：



```
test3.c (~/Desktop/Temp) - gedit
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>
int main() {
    int pid1 = fork();
    if (0 == pid1) {
        int pid2 = fork();
        if (0 == pid2) {
            printf("c\n");
        } else {
            printf("b\n");
        }
    } else {
        printf("a\n");
    }
    return 0;
}
```

· 编译并多次运行，发现结果符合预期要求：



```
pie@ubuntu: ~/Desktop/Temp
pie@ubuntu Desktop/Temp ➤ g++ test3.c -o test3.o
pie@ubuntu Desktop/Temp ➤ ./test3.o
a
b
c
pie@ubuntu Desktop/Temp ➤ ./test3.o
a
b
c
pie@ubuntu Desktop/Temp ➤ ./test3.o
a
b
c
```

· 代码分析：主要原理是fork()函数返回值在父进程和子进程中的不同表现，父进程中其返回值为所创建的子进程pid，子进程中为0，故在主进程1中，pid1为子进程id，不为零，进入最后else段，输出a；在子进程2中，pid1为0，进入第一个if段，之后fork()创建子进程3，pid2为进程3的pid，进入else段，输出b；子进程3中，pid2为0，进入第二个if段，输出c。

## 五. 实验总结与心得

本次实验并不复杂,主要是通过分析代码结果、预测代码结果、实现程序要求三个方面,让我们对fork函数有一定的理解和掌握,并能够进行基本的分析和利用。虽然其背后定有更为复杂的原理,但是从这几个实例中,如果我的理解和分析是正确的,那也算是基本上了解了fork函数及其返回值、父子进程之间关系和运行情况的基本概念了。

fork函数的返回值表现为:

- 1) 在父进程中,返回新创建子进程的进程ID;
- 2) 在子进程中,返回0;
- 3) 如果出现错误,返回一个负值;

而父子进程之间只共享代码,而不共享数据,子进程在被创建时,拷贝了父进程的数据空间、堆、栈等资源的副本,父子进程之间并不共享这些存储空间,而只有代码段。因此,子进程修改一个全局变量,父进程的该全局变量并不会改变。

总的来说,这是一次比较有练习效果、比较有收获,但又不至于太难太花时间的实验。