



华南理工大学

实 验 报 告

课程名称：操作系统

学生姓名：陈卓文

学生学号：201936380215

学生专业：软件工程

开课学期：2020-2021 第二学期

2021 年 04 月

实验二 进程创建及进程间通信

地 点： B7 楼 331 房； 评 分： _____
实验日期与时间： 2021.04.11 13: 50-17: 25 实验教师： _____

一、实验目标

1. 掌握 Linux 进程的创建方法，加深对进程概念的理解，明确进程和程序的区别。
2. 认识进程并发执行的实质。
3. 学习控制进程同步的方法。
4. 分析进程竞争资源的现象，学习解决进程互斥的方法。
5. 了解管道通信的特点，掌握管道通信的使用方法。

二、实验内容

1. 运行Linux进程的创建程序，观察运行结果。
2. 利用fork函数，编写程序。
3. 用fork()创建一个进程，再调用exec()用新的程序替换该子进程的内容。
利用wait()来控制进程执行顺序。
4. 用lockf()来给每一个进程加锁，以实现进程之间的互斥。
5. 用pipe()来实现进程的管道通信。

三、实验设备及环境

PC（ubuntu 操作系统）； C/C++等编程语言。

四、实验主要步骤

1. 进程创建程序示例：

```

#include<stdio.h>
int main(){
    int p1;
    while((p1=fork())== -1);
    if(p1==0)
        printf("This is a child process.");/*在子进程中*/
    else{
        printf("This is a parent process.");/*在父进程中*/
    }
    return 0;
}

```

2. 进程的创建。编写一段 C/C++ 程序，使用系统功能调用 `fork()` 创建两个子进程。要求：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“c”。试观察记录屏幕上的显示结果，并分析原因。提示：实验结果需要运行多次找出不同结果并分析原因。
3. 运行以下程序，并分析 `switch` 语句中各个 `case` 所做的事和产生原因。`wait()` 给我们提供了一种实现进程同步的简单方法，它是如何实现进程同步的？

```

#include<stdio.h>
#include<unistd.h>
int main(){
    int pid;
    pid = fork();
    switch(pid){
        case -1:
            printf("Error in fork()\n");
            exit(1);
        case 0:
            execl("/bin/ls", "ls", "-l", "-color", NULL);
            printf("execl fail!\n");
            exit(1);
        default:
            wait(NULL);
            printf("is completed!\n");
            exit(0);
    }
    return 0;
}

```

4. 分析以下程序的输出结果。可以使用 `cat to_be_locked.txt` 查看输出结果。多次验证看是否有不同结果，为什么？

```
#include <stdio.h>
#include <unistd.h>

int main(){
    int p1, p2, i;
    int *fp;
    fp = fopen("to_be_locked.txt", "w+");
    if(fp == NULL){
        printf("Fail to create File\n");
        exit(-1);
    }
    while((p1 = fork()) == -1){ /*创建子进程p1*/
        if(p1 == 0){
            lockf(*fp, 1, 0); /*加锁*/
            for(i = 0; i < 10; i++)
                fprintf(fp, "daughter %d\n", i);
            lockf(*fp, 0, 0); /*解锁*/
        }else{
            while((p2 = fork()) == -1){ /*创建子进程p2*/
                if(p2 == 0){
                    lockf(*fp, 1, 0); /*加锁*/
                    for(int i = 0; i < 10; i++)
                        fprintf(fp, "son %d\n", i);
                    lockf(*fp, 0, 0); /*解锁*/
                }else{
                    wait(NULL);
                    lockf(*fp, 1, 0); /*加锁*/
                    for(i = 0; i < 10; i++)
                        fprintf(fp, "parent %d\n", i);
                    lockf(*fp, 0, 0); /*解锁*/
                } // end if p2 == 0
            }; // end p2
        } // end if p1 == 0
    } // end p1
    fclose(fp);
    return 0;
}
```

5. 编写程序：实现进程的管道通信，用系统调用 `pipe()` 建立一管道，二个子进程 P1 和 P2 分别向管道各写一句话：

Child 1 is sending a message!

Child 2 is sending a message!

父进程从管道中读出二个来自子进程的信息并显示（要求先接收 P1，后 P2）。

五、问题与算法

1. 问题描述

这次实验主要是学习 `fork`, `wait` 和 `pipe` 的用法。

2. 算法思路

`Fork` 的作用是将运行着的程序分成2个几乎一样的进程，每个进程都启动一个从代码的同一位置开始执行的线程。但是父线程的 `fork` 函数会返回

子进程pid，子进程fork函数会返回零。

Wait是等待子进程的完成

Pipe是返回两个fd，使用write和read便可以实现进程间通讯。

3. 算法实现关键点

通过判断fork的返回值便可以判断父子进程，从而执行两份不同的代码。

六、实验结果与分析

1. 实验数据及结果

实验步骤一：

```
simon@Aliyun:~/code/study$ ./a.out
This is a parent process.
simon@Aliyun:~/code/study$ This is a child process.
```

图1 步骤一输出

实验步骤二：

```
C++ 2.cpp > ...
1  #include <assert.h>
2  #include <unistd.h>
3  #include <iostream>
4  using namespace std;
5
6  int main() {
7      for (char ch = 'a'; ch <= 'c'; ch++) {
8          pid_t child = fork();
9          assert(child != -1);
10         if (child) {
11             cout << ch << endl;
12             break;
13         };
14     }
15 }
```

图2 步骤二源代码

```
simon@Aliyun:~/code/study$ g++ 2.cpp
simon@Aliyun:~/code/study$ ./a.out
a
simon@Aliyun:~/code/study$ b
c
./a.out
a
simon@Aliyun:~/code/study$ b
c
```

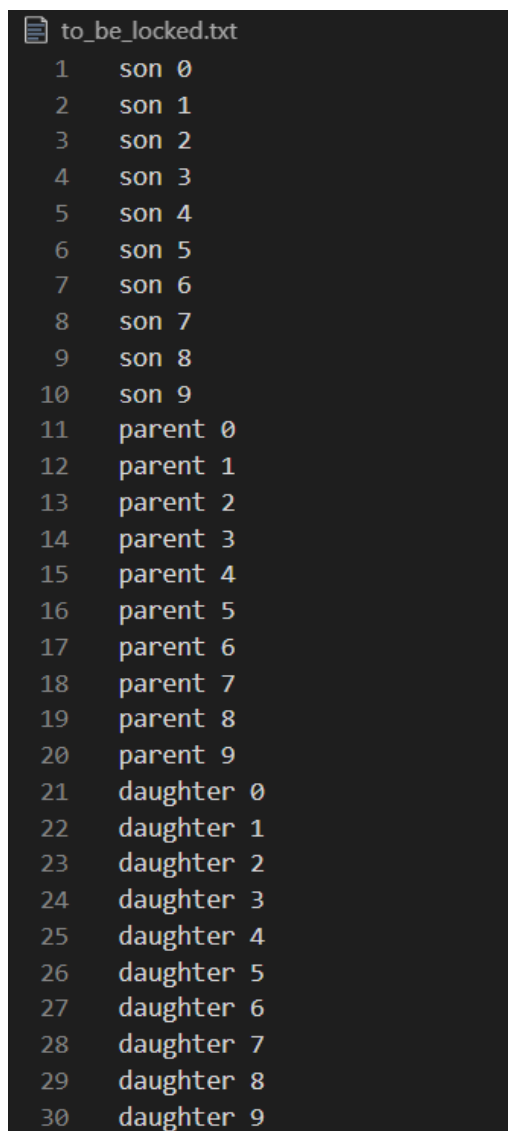
图3 步骤二输出

经过多次执行，输出未发生变化。此处理论上的确有可能乱序输出，原因是fork后操作系统未保证父子进程的执行顺序。

步骤三：case的作用是依据fork的返回值执行不同的事，-1代表创建失败，0代表是子进程，default代表父进程。

Wait的作用是调用后会在子进程执行时挂起父进程的执行。子进程终止时，它将退出状态返回给操作系统，然后将其返回给等待的父进程。然后，父进程恢复执行。以此达到进程同步。

步骤四：这个程序在fork不会失败的情况下，没有输出，原因是两个while循环条件为false。在while后添加分号且再次执行后得到输出：



```
to_be_locked.txt
1 son 0
2 son 1
3 son 2
4 son 3
5 son 4
6 son 5
7 son 6
8 son 7
9 son 8
10 son 9
11 parent 0
12 parent 1
13 parent 2
14 parent 3
15 parent 4
16 parent 5
17 parent 6
18 parent 7
19 parent 8
20 parent 9
21 daughter 0
22 daughter 1
23 daughter 2
24 daughter 3
25 daughter 4
26 daughter 5
27 daughter 6
28 daughter 7
29 daughter 8
30 daughter 9
```

图4 步骤四输出（修改源代码后的）

产生这个输出的原因是,首先父子进程的先后顺序是乱序的，其次文件读写有锁，同一时间有且仅有一个进程写入文件。同时，父进程有一个wait，但是只要有一个子进程结束wait便会返回，故父进程可能出现在第二位和第三位。

步骤五:

```
C++ 5.cpp > main()
1  #include <assert.h>
2  #include <unistd.h>
3  #include <stdio.h>
4  #include <stdlib.h>
5  #include <string.h>
6  #include <sys/wait.h>
7  using namespace std;
8
9  int fd[2] = {};
10
11 pid_t fork_write(const char* ch) {
12     pid_t pid = fork();
13     assert(pid != -1);
14     if (pid)
15         return pid;
16     else {
17         close(fd[0]);
18         write(fd[1], ch, strlen(ch));
19         exit(0);
20     }
21 }
22
23 char buf[200];
24 int main() {
25     pipe(fd);
26
27     pid_t pid1 = fork_write("Child 1 is sending a message!");
28     waitpid(pid1, nullptr, 0);
29     read(fd[0], buf, 200);
30     printf("%s\n", buf);
31
32     pid_t pid2 = fork_write("Child 2 is sending a message!");
33     waitpid(pid2, nullptr, 0);
34     read(fd[0], buf, 200);
35     printf("%s\n", buf);
36 }
```

图5 步骤五源代码

```
simon@Aliyun:~/code/study$ g++ 5.cpp
simon@Aliyun:~/code/study$ ./a.out
Child 1 is sending a message!
Child 2 is sending a message!
simon@Aliyun:~/code/study$
```

图6 步骤五输出

2. 实验分析及结论

能够很好的实现步骤中的要求，或者复现步骤中出现的输出。

七、心得与展望

1. 自我评价及心得体会

能够深刻理解fork, wait, pipe的作用与原理，能够掌握多进程的基础编程。

2. 展望

无

八、附录

1. 主要界面

Ubuntu 18.04, vscode, bash

2. 源程序

详见图2与图5

九、参考文献

[1] 《计算机操作系统教程》张尧学等，清华大学出版社，2006年10月第3版