



操作系统实验二

进程管理

班级：2015211314 学号：2015211527 姓名：罗暄澍

目录

实验二 进程管理.....	2
一、实验目的与实验环境.....	2
1、实验目的.....	2
2、实验环境.....	2
二、实验内容.....	3
1、进程的创建.....	3
2、进程的控制.....	4
3、调试并运行下面的程序，给出运行结果.....	6

实验二 进程管理

一、实验目的与实验环境

1、实验目的

- (1) 加深对进程概念的理解，明确进程和程序的区别
- (2) 进一步认识并发执行的实质

2、实验环境

首先打开 CMD，输入 bash，进入 Linux 子系统

通过 `lsb_release -a` 命令和 `gcc -v` 命令查看 Ubuntu 版本和 gcc 编译器版本

```
maurice@SurfaceBook-Mre: /mnt/c/Users/Maurice Luo
Microsoft Windows [版本 10.0.16299.64]
(c) 2017 Microsoft Corporation。保留所有权利。

C:\Users\Maurice Luo>bash
maurice@SurfaceBook-Mre: /mnt/c/Users/Maurice Luo$ lsb_release -a
No LSB modules are available.
Distributor ID: Ubuntu
Description:    Ubuntu 16.04.3 LTS
Release:        16.04
Codename:       xenial
maurice@SurfaceBook-Mre: /mnt/c/Users/Maurice Luo$ gcc -v
Using built-in specs.
COLLECT_GCC=gcc
COLLECT_LTO_WRAPPER=/usr/lib/gcc/x86_64-linux-gnu/5/lto-wrapper
Target: x86_64-linux-gnu
Configured with: ../src/configure -v --with-pkgversion='Ubuntu 5.4.0-6ubuntu1~16.04.5' --with-bugurl=file:///usr/share/doc/gcc-5/README.Bugs --enable-languages=c,ada,c++,java,go,d,fortran,objc,obj-c++ --prefix=/usr --program-suffix=-5 --enable-shared --enable-linker-build-id --libexecdir=/usr/lib --without-included-gettext --enable-threads=posix --libdir=/usr/lib --enable-nls --with-sysroot=/ --enable-clocale=gnu --enable-libstdcxx-debug --enable-libstdcxx-time=yes --with-default-libstdcxx-abi=new --enable-gnu-unique-object --disable-vtable-verify --enable-libmpx --enable-plugin --with-system-zlib --disable-browser-plugin --enable-java-awt=gtk --enable-gtk-cairo --with-java-home=/usr/lib/jvm/java-1.5.0-gcj-5-amd64/jre --enable-java-home --with-jvm-root-dir=/usr/lib/jvm/java-1.5.0-gcj-5-amd64 --with-jvm-jar-dir=/usr/lib/jvm-export/java-1.5.0-gcj-5-amd64 --with-arch-directory=amd64 --with-ecj-jar=/usr/share/java/eclipse-ecj.jar --enable-objc-gc --enable-multiarch --disable-werror --with-arch-32=i686 --with-abi=m64 --with-multilib-list=m32,m64,mx32 --enable-multilib --with-tune=generic --enable-checking=release --build=x86_64-linux-gnu --host=x86_64-linux-gnu --target=x86_64-linux-gnu
Thread model: posix
gcc version 5.4.0 20160609 (Ubuntu 5.4.0-6ubuntu1~16.04.5)
maurice@SurfaceBook-Mre: /mnt/c/Users/Maurice Luo$
```

实验环境如下：

操作系统	Windows 10 Pro 1703 版本的 Linux 子系统 Linux 子系统为 Ubuntu 16.04.3 LTS
编译器	gcc version 5.4.0

二、实验内容

1、进程的创建

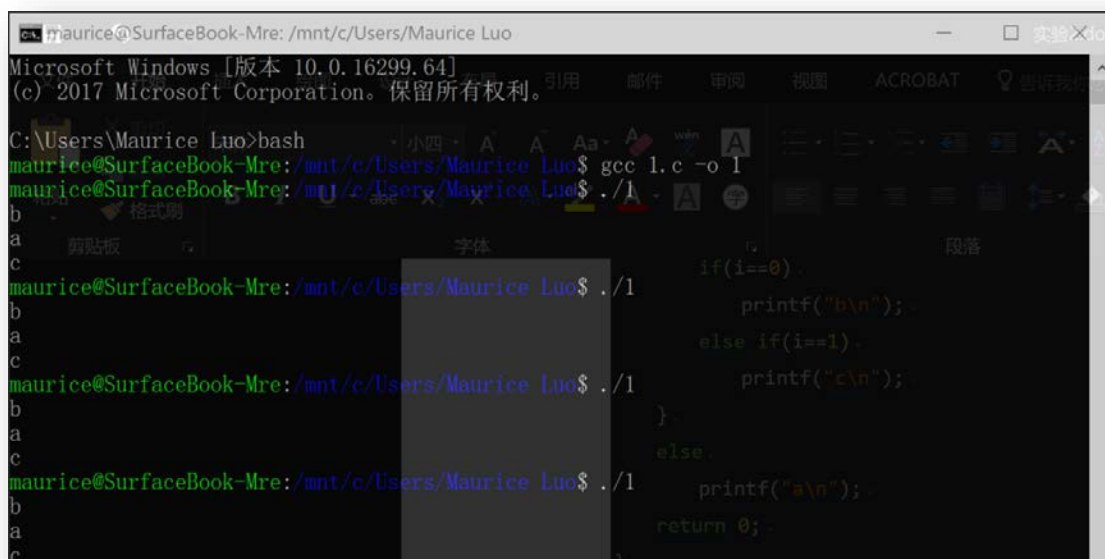
描述

编写一段程序，使用系统调用 *fork()* 或 *createprocess()* 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“c”。试观察记录屏幕上的显示结果，并分析原因。

代码

```
#include <unistd.h>
#include <stdio.h>
#include <sys/types.h>
#include <stdlib.h>
int main(int argc ,char argv[])
{
    int i;
    pid_t status;
    for (i=0;i<2;i++){
        status = fork();
        if(status == 0||status == -1)
            break; //每次循环时，假设发现是子进程就直接从创建子进程的循环中跳出来。
                //保证了每次仅仅有父进程来做循环创建子进程的工作
    }
    if (status == -1)
    {
        printf("error in fork!");
        exit(-1);
    }
    else if (status == 0) //每一个子进程都会运行的代码
    {
        if(i==0)
            printf("b\n");
        else if(i==1)
            printf("c\n");
    }
    else
        printf("a\n");
    return 0;
}
```

输出



```
Microsoft Windows [版本 10.0.16299.64]
(c) 2017 Microsoft Corporation. 保留所有权利。

C:\Users\Maurice Luo>bash
maurice@SurfaceBook-Mre:/mnt/c/Users/Maurice Luo$ gcc 1.c -o 1
maurice@SurfaceBook-Mre:/mnt/c/Users/Maurice Luo$ ./1
b
a
c
maurice@SurfaceBook-Mre:/mnt/c/Users/Maurice Luo$ ./1
b
a
c
maurice@SurfaceBook-Mre:/mnt/c/Users/Maurice Luo$ ./1
b
a
c
```

可以看到每次的输出均为 bac 或 bca

原因

在代码中，我利用循环每次都利用主进程调用 *fork()* 方法来生成子进程。一个可能的流程如下：主进程生成子进程 1——>子进程 1 输出 b——>回到主进程输出 a——>主进程生成子进程 2——>子进程 2 输出 c。出现不确定输出的原因是因为父进程与两个子进程之间没有同步措施，所以父进程和两个子进程的输出次序带有随机性；同时系统对父子进程在不同时刻的管理方法也不一样。这些都是导致随机的原因

2、进程的控制

描述

修改已经编写的程序，将每个进程输出一个字符改为每个进程输出一句话，再观察程序执行时屏幕上出现的现象，并分析原因。

代码

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

int main(int argc ,char argv[])
{
    int i;
    pid_t status;

    for (i=0;i<2;i++)
```

```

{
    status = fork();
    if(status == 0 || status == -1)
        break; //每次循环时，假设发现是子进程就直接从创建子进程的循环中跳出来。
                //不让你进入循环，这样就保证了每次仅仅有父进程来做循环创建子进程的工作
}

if (status == -1)
{
    printf("error in fork 1!");
    exit(-1);
}

else if (status == 0) //每一个子进程都会运行的代码
{
    if(i==0)
    {
        printf("I'm child 1.\n");
        exit(0);
    }

    else if(i==1)
    {
        printf("I'm child 2.\n");
        exit(0);
    }

}

else
    printf("I'm parent.\n");

return 0;
}

```

输出

```

maurice@SurfaceBook-Mre: /mnt/c/Users/Maurice Luo
Microsoft Windows [版本 10.0.16299.64]
(c) 2017 Microsoft Corporation. 保留所有权利。

C:\Users\Maurice Luo>bash
maurice@SurfaceBook-Mre: /mnt/c/Users/Maurice Luo$ gcc l.c -o l
maurice@SurfaceBook-Mre: /mnt/c/Users/Maurice Luo$ ./l
I'm child 1.
I'm parent.
I'm child 2.
maurice@SurfaceBook-Mre: /mnt/c/Users/Maurice Luo$ ./l
I'm child 1.
I'm parent.
I'm child 2.
maurice@SurfaceBook-Mre: /mnt/c/Users/Maurice Luo$

```

复制了上一个实验的结果

原因

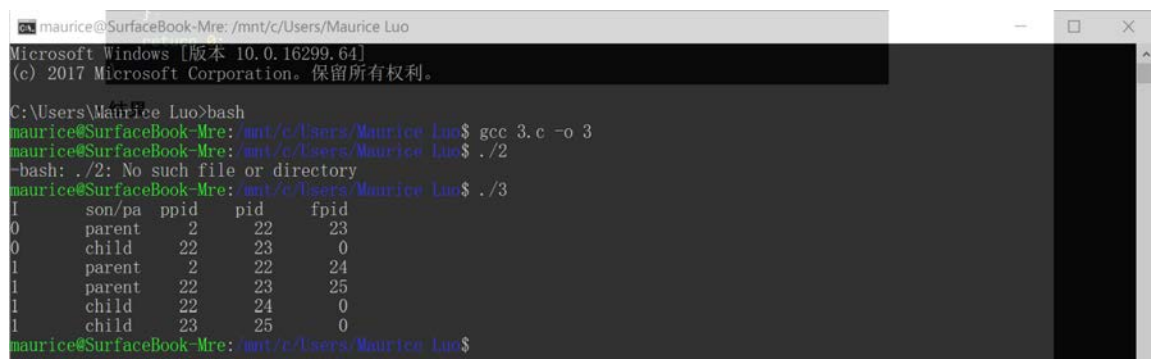
输出结果的随机性分析与上次实验相同

3、调试并运行下面的程序，给出运行结果

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

int main()
{
    int i=0;
    printf("I\\tson\\pa\\tppid\\tpid\\tfpid\\n");
    //ppid 指当前进程的父进程 pid
    //pid 指当前进程的 pid,
    //fpid 指 fork 返回给当前进程的值
    for(i=0;i<2;i++)
    {
        pid_t fpid=fork();
        if(fpid==0)
            printf("%d\\tchild\\t%4d\\t%4d\\t%4d\\n", i, getppid(), getpid(), fpid);
        else
            printf("%d\\tparent\\t%4d\\t%4d\\t%4d\\n", i, getppid(), getpid(), fpid);
    }
    return 0;
}
```

结果



```
maurice@SurfaceBook-Mre: /mnt/c/Users/Maurice Luo
Microsoft Windows [版本 10.0.16299.64]
(c) 2017 Microsoft Corporation。保留所有权利。

C:\Users\Maurice Luo>bash
maurice@SurfaceBook-Mre: /mnt/c/Users/Maurice Luo$ gcc 3.c -o 3
maurice@SurfaceBook-Mre: /mnt/c/Users/Maurice Luo$ ./2
-bash: ./2: No such file or directory
maurice@SurfaceBook-Mre: /mnt/c/Users/Maurice Luo$ ./3
I      son/pa  ppid  pid  fpid
0      parent  2    22  23
0      child   22   23  0
1      parent  2    22  24
1      parent  22   23  25
1      child   22   24  0
1      child   23   25  0
maurice@SurfaceBook-Mre: /mnt/c/Users/Maurice Luo$
```

分析

乍看起来应该只输出 4 个结果，然而最后却输出了六个。这是因为子进程执行完之后并没有进入 terminate 状态，而是继续执行。下一次使用 `fork()` 方法时，主进程和子进程均调用该方法，所以会有六行输出。从输出结果我们也可看出子进程又生成了子进程。

修改后再次运行

这里我们将子进程代码中增加 `return 0;` 得到如下代码。

```

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

int main()
{
    int i=0;
    printf("I\\tson\\pa\\tppid\\tpid\\tfpid\\n");
    //ppid 指当前进程的父进程 pid
    //pid 指当前进程的 pid,
    //fpid 指 fork 返回给当前进程的值
    for(i=0;i<2;i++)
    {
        pid_t fpid=fork();
        if(fpid==0)
        {
            printf("%d\\tchild\\t%4d\\t%4d\\t%4d\\n", i, getppid(), getpid(), fpid);
            return 0;
        }
        else
            printf("%d\\tparent\\t%4d\\t%4d\\t%4d\\n", i, getppid(), getpid(), fpid);
    }
    return 0;
}

```

再次运行可得预期的四个输出的结果，而不是 6 个。

```

maurice@SurfaceBook-Mre: /mnt/c/Users/Maurice Luo
C:\Users\Maurice Luo>bash
maurice@SurfaceBook-Mre: /mnt/c/Users/Maurice Luo$ gcc 3.c -o 3
maurice@SurfaceBook-Mre: /mnt/c/Users/Maurice Luo$ ./2
-bash: ./2: No such file or directory
maurice@SurfaceBook-Mre: /mnt/c/Users/Maurice Luo$ ./3
I      son/pa  ppid  pid    fpid
0      parent  2     22     23
0      child   22    23     0
1      parent  2     22     24
1      parent  22    23     25
1      child   22    24     0
1      child   23    25     0
maurice@SurfaceBook-Mre: /mnt/c/Users/Maurice Luo$ gcc 3.c -o 3
maurice@SurfaceBook-Mre: /mnt/c/Users/Maurice Luo$ ./3
I      son/pa  ppid  pid    fpid
0      parent  2     31     32
0      child   31    32     0
1      parent  2     31     33
1      child   31    33     0
maurice@SurfaceBook-Mre: /mnt/c/Users/Maurice Luo$

```