**课 程 实 验 报 告**

**课程名称： 操作系统**

**实验名称： 实验一Linux进程创建**

**专业班级： 计算机Ⅱ类2103班**

**学 号： 8008121077**

**姓 名： 陈映融**

**指导教师： 胡勇**

**报告日期： 2023/03/27-2023/04/04**

## 1.1 实验概述

实验目的：掌握进程的概念，明确进程和程序的区别

实验目标：创建进程、控制进程

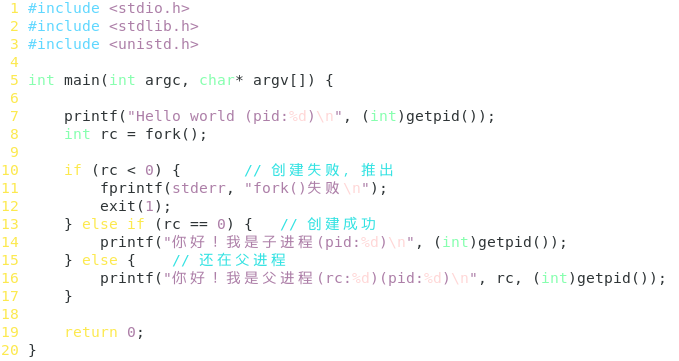
实验要求：通过验证实验内容掌握进程创建，解决实验设计问题，提交报告

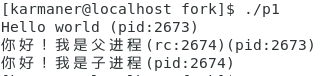
实验语言：c

实验环境：linux、gcc

## 1.2 实验过程及结果

**1.2.1 fork()创建进程**

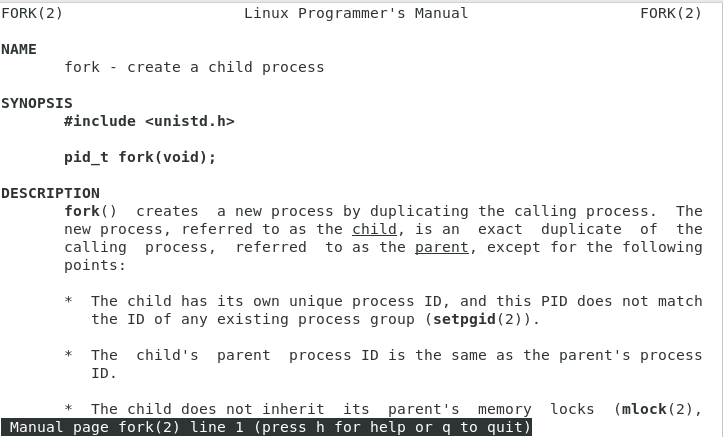




从上我们可以看到程序代码的选择结构”看起来”执行了两次”好像是程序出错“。但是Hello world只输出了一次。究其原因是fork函数。fork是怎么运行的有什么样的返回值。键入以下



得到以下结果



1. 得出解释fork是对父进程代码的拷贝，拷贝代码给到创建的一个子进程。fork()在子进程中返回整形0，在父进程中返回子进程的pid所以我们看到rc的值为子进程ID。这就引出了一个新问题，当子进程被创建后，父进程和子进程竞争CPU，完全有这样一种可能：CPU先执行子进程再执行父进程。所以才有实验指导的两种结果。

有了fork对于代码是怎么拷贝的，那么对于变量呢？对于子进程父进程打开同一文件到底是怎么样的？

代码结构基本相同我们可以用

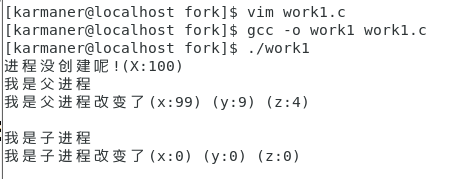


来减少操作。

**实验设计一**我们这里设计三个变量一个是main内局部变量，一个是全局变量，一个是静态全局变量。代码如下



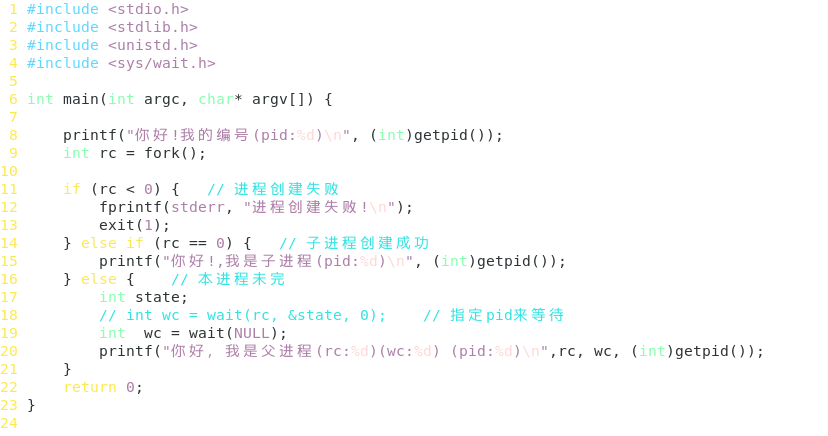
结果如下



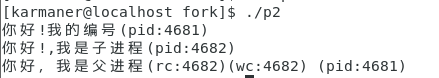
1. 从结果可以看出来x,y,z三个变量好像一点而也不相关了，其实事实上也是如此不管是局部变量，全局变量，静态全局变量子进程都会对他进行拷贝(不管位置)拷贝的内容将分配新的内存，这是个有趣的现象一个程序可以创建多个进程。
2. 接下来我们对父子进程访问文件做出判断，父进程写写入文件(保证先访问)，子进程读出文件(保证后访问)。其实在fork()以后这已经是形成了两个进程，每个进程都有自己的进程实体，所以对于文件的访问只和CPU先执行那个进程挂钩。好！接下来我们怎么让CPU执行父进程还是子进程变的确定！下面是实验1.2.2

**1.2.2 wait()、waitpid()、sleep()系统调用**

为了让进程的运行变的确定，Linux提供了wait(),waitpid()两个系统调用的。我们先来看看wait()



结果如下：



我们可以看到子进程在父进程之前完成了,并且wc(wait的返回值)同fork()一样返回了子进程的pid。如果在子进程中使用wait函数则会返回-1。waitpid()是wait的增强版waitpid()有三个参数



Man手册是这样描述waitpid的用法的

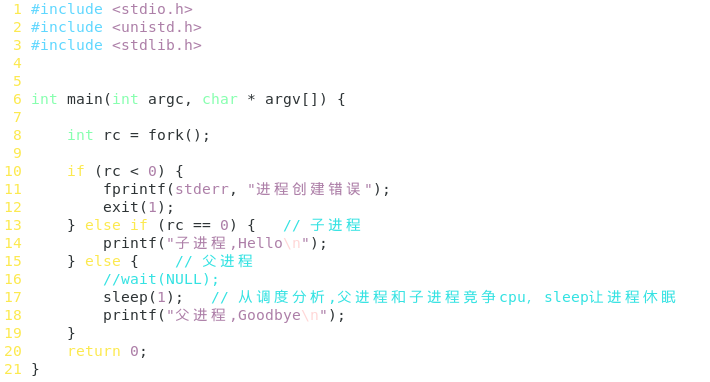


具体用法在上图注释部分

**实验设计的第二题**不使用wait(),waitpid()子进程先打印，父进程的后打印。有两个思路一是竞争失败，二是进程堵塞等来某个进程完成。显然wait和waitpid是通过二来实现的。要不用wait(),waitpid().我们可以使用sleep()函数在man手册 中的描述是这样的



sleep函数可以让进程挂起(放弃竞争时间片)也可以说是暂停进程。很显然有sleep(num)的或者num大的则排序在后竞争失败，后运行。以下是代码



结果如下:



实现了程序要求这只是实现了子进程排在了父进程的前面但是如果子进程调用进入阻塞态，父进程同样也会先执行。

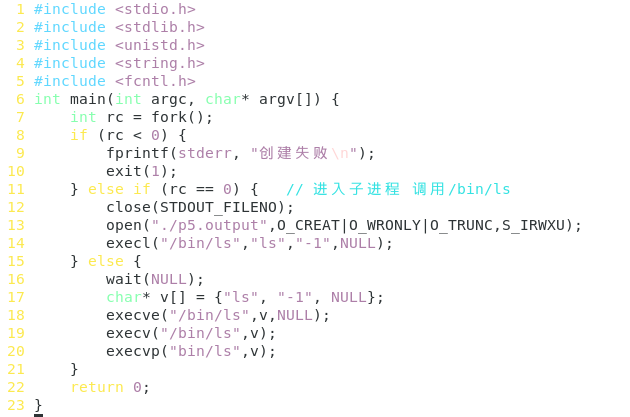
通过思想二也可以实现以上功能。也就是父进程等待子进程的消息。通过pipe来实现。代码如下



通过pipe。父进程等待子进程要写入的数据来实现wait功能。就这样可以实现父进程等待子进程。

**1.2.3exec()系统调用及其变种和输出重定向**

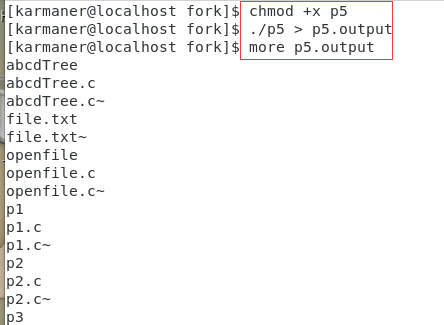
我参考《操作系统导论》调用bash/bin/ls程序。以下是程序代码



编译程序以后键入./p5开始运行提示



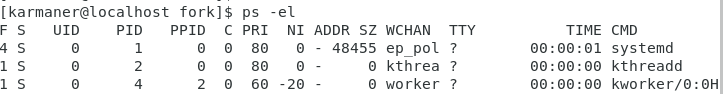
Linux默认不提供运行权限键入chmod +x p5



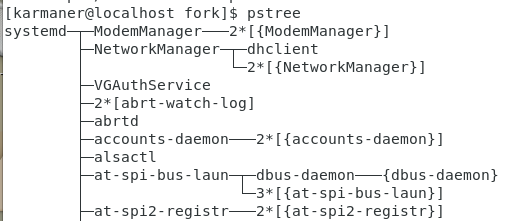
实现了输出重定向。”好像”只执行了子进程。我们先前就知道fork()是形成了两个进程互不相关为什么父进程没有运行呢。原因是出在了exec()函数和wait()上。父进程wait(NULL)需要等待子进程完成，但是exec()函数开始运行，就接管了这个进程可以运行其他的程序**不返回**所以等待子进程完成的父进程不会运行。exec有多个变种，用法大体相同略有区别。

**1.2.4 进程树**

在Linux中提供了,ps命令来查看进程列表。如ps -el 部分进程如下



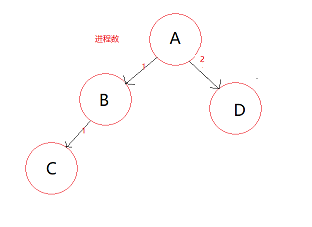
我们也可以通过pstree来查看进程之间的关系也就是系统的进程数。结果如下



以下是一个程序代码，请画出他的进程图



对代码进行分析原进程为A在第9行第一此调用fork()创建了子进程B，在第28行调用创建了子进程D。在子进程B中B在16行又创建了C。所以按照我的推理来讲进程树应该是这样的

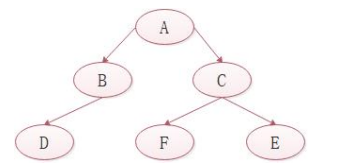


运行验证结果：



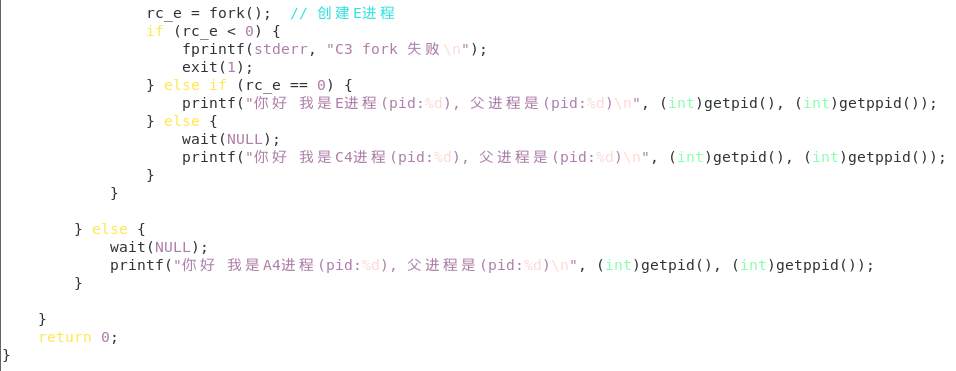
就如同我们推测的一般A的子进程是B和D，C的父进程是B。结果确实是如此。

**实验设计三**我们需要写出下图的代码

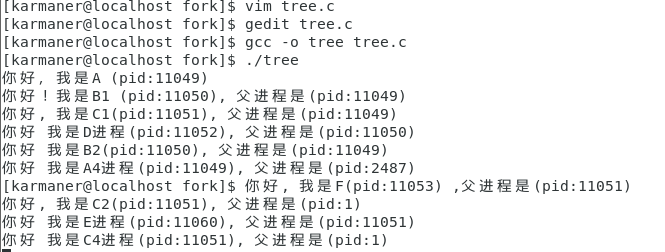


其实就要掌握了fork()函数的返回值这题就很好做了





运行结果如下

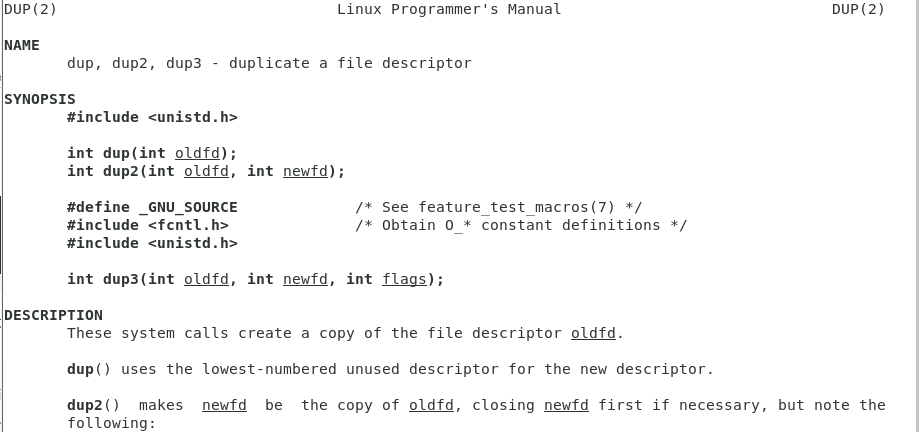


我们来分析一下结果首先A是B C的父进程。 B是D的父进程。C是E F的父进程。上面结果还有有趣的地方。比如C2 C4的父进程为什么是1我认为是在A完成后就被释放了导致的。对于兄弟进程的创建要把我住选择结构。

else if(rc == 0)是子进程范围，else 是父进程范围 在else中创建他的兄弟进程。

**1.2.5进程通讯**

**实验设计四**创建两个子进程，并使用 pipe()系统调用，将一个子进程的 标准输出连接到另一个子进程的标准输入。要有标准的输入输出就不能关闭标准一下是man手册对于dup dup2 dup3的描述



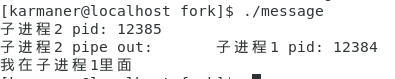
这dup()函数可以把原本STDOUT\_FILENO和作为标准输出和显示器绑定的，现在调用函数dup2，会先解除它们之间的映射关系，然后让STEOUT\_FILENO与文件描述符fd指向的文件表项建立映射关系。也就能通过pipe进行通讯了。

一下是代码



输入是由子进程一来输入的，然后有子进程二来用buffer来读取，并且输出。这就实现了子进程之间的通讯。

以下为通讯结果



子进程二输出了子进程一的内容实现了进程之间的通讯。

以上本次实验就完结了

## 1.3 实验小结及心得

本次实验花费的时间较长。需要学习的项目多，fokr(),wait(),waitpid(),pipe(),exec()，dup()以及他们的变种。当然本次实验也充分了解了进程(Linux)是怎么创建的，怎么控制进程，怎么实现进程间的通讯。和一个很大的概念进程树。

本次实验即是对预实验的加深运用尤其是vim的C代码编写和进程控制更加熟悉。本次实验给我的最大的体验就是，文件要自己管理了每个文件在什么位置，叫什么名字明明就以及在心中有了位置。怎么样对文件处理才能减少重复代码的编写，怎么样对vim设置会有更好的代码风格。本次实验让我有了十足的长进。Man手册同样作为Linux学习和查询的一个重要作用。

本此实验让我对于操作系统的权限概念有了新的理解，实验调用的wc ls(shell程序)如果没有权限是不能直接运行的这也就是操作系统用于区分模式的一个例证吧！

进程树的实验是比较困难的，在vim下写代码一些重复的代码复用很麻烦，在vim命令模式复制粘贴。代码的结构分析比较难，特别是对于进程执行的先后顺序的把控十分有挑战。