

**[进程控制实验]**

**操作系统**

**实验报告**

**学院：计算机学院**

**2020211376 马天成**

**2022年10月9日**

**北京邮电大学《操作系统》课程实验报告**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **实验**  **名称** | 进程控制 | | **学 院** | 计算机 | **指导教师** | 李文生 |
| **班 级** | **班内序号** | **学 号** | | **学生姓名** | **成绩** | |
| **2020211305** | **2** | **2020211376** | | **马天成** |  | |
| **实**  **验**  **内**  **容** | 实验内容一： 采用系统调用 fork()，编写一个 C 程序，以便在子进程中生成这个序列。 要求： （1）从命令行提供启动数字 （2）由子进程输出数字序列 （3）父进程等子进程结束后再退出。 实验内容二： 以共享内存技术编程实现 Collatz 猜想。 要求在父子进程之间建立一个共享内存对象，允许子进程将序列内容写入共 享内存对象，当子进程完成时，父进程输出序列。 父进程包括如下步骤： 建立共享内存对象（shm\_open(), ftruncate(), mmap()） 建立子进程并等待他终止 输出共享内存的内容 删除共享内存对象。 实验内容三：普通管道通信 设计一个程序，通过普通管道进行通信，让一个进程发送一个字符串消息给 第二个进程，第二个进程收到此消息后，变更字母的大小写，然后再发送给第一 个进程。比如，第一个进程发消息：“I am Here”，第二个进程收到后，将它改变 为：“i AM hERE”之后，再发给第一个进程。 提示： （1）需要创建子进程，父子进程之间通过普通管道进行通信。 （2）需要建立两个普通管道 | | | | | |
| **学生实验报告** | （详见“实验报告和源程序”册） | | | | | |
| **课**  **程**  **设**  **计**  **成**  **绩**  **评**  **定** | **评语**:  **成绩**:  指导教师签名：  年 月 日 | | | | | |

注：评语要体现每个学生的工作情况，可以加页。

目录

[1. 实验目的和要求 4](#_Toc116225691)

[1.1 实验目的 4](#_Toc116225692)

[1.2 实验要求 4](#_Toc116225693)

[2. 实验内容 5](#_Toc116225694)

[2.1 准备工作 5](#_Toc116225695)

[2.2 父子进程 5](#_Toc116225696)

[2.3 share memory 通信 6](#_Toc116225697)

[2.4 pipe通信 7](#_Toc116225698)

[3. 实验总结 9](#_Toc116225699)

# 1. 实验目的和要求

## 1.1 实验目的

详细了解父子进程以及进程间通信的2种方式，并通过代码实现加深理解和运用

1. 学会使用fork函数

2. 学会共享空间通信

3. 学会管道通信

## 1.2 实验要求

* 题面

进程控制 Collatz 猜想：任意写出一个正整数 N，并且按照以下的规律进行变换： 如果是个奇数，则下一步变成 3N+1；如果是个偶数，则下一步变成 N/2。 无论 N 是怎样的一个数字，最终都无法逃脱回到谷底 1。 例如：如果 N=35，则有序列 35, 106, 53, 160, 80, 40, 20, 10, 5, 16, 8, 4, 2, 1。

* 实验内容一

采用系统调用 fork()，编写一个 C 程序，以便在子进程中生成这个序列。

要求：

（1）从命令行提供启动数字

（2）由子进程输出数字序列

（3）父进程等子进程结束后再退出。

* 实验内容二

以共享内存技术编程实现 Collatz 猜想。 要求在父子进程之间建立一个共享内存对象，允许子进程将序列内容写入共 享内存对象，当子进程完成时，父进程输出序列。 父进程包括如下步骤： 建立共享内存对象（shm\_open(), ftruncate(), mmap()） 建立子进程并等待他终止 输出共享内存的内容 删除共享内存对象。

* 实验内容三

普通管道通信 设计一个程序，通过普通管道进行通信，让一个进程发送一个字符串消息给 第二个进程，第二个进程收到此消息后，变更字母的大小写，然后再发送给第一 个进程。比如，第一个进程发消息：“I am Here”，第二个进程收到后，将它改变 为：“i AM hERE”之后，再发给第一个进程。

提示：

（1）需要创建子进程，父子进程之间通过普通管道进行通信。

（2）需要建立两个普通管道。

# 2. 实验内容

## 2.1 准备工作

首先准备好linux平台。我这里使用的是CentOS-7-x86\_64-DVD-2009

此外，也要准备好gcc环境。这里使用yum直接一键解决了。

代码编写的思路也很简单。照着PPT上的思路复现一遍基本上就可以了。

## 2.2 父子进程

基本上就是如何使用fork函数。

int main(int argc ,char\* argv[])

{

pid\_t pid=fork();

if(pid==-1) {

printf("Child process create failed\n");

return 0;

}

//child process

else if(pid == 0) {

···

}

//parent process

else if(pid >0) {

···

exit(0);

}

return 0;

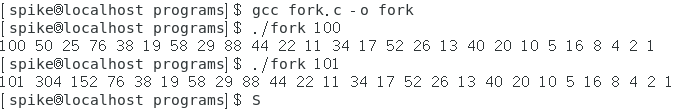
}

上述代码展现了如何使用fork函数的框架。

若pid申请得到的返回值<0，则为错误申请，直接报错。

若pid申请得到的值是>0，则为申请成功，执行子进程，执行结束后返回父进程。

执行结果：



## 2.3 share memory 通信

其实就是对shm框架的使用;

int main(int argc ,char\* argv[])

{

// create share memory file

const char\* name = "share memory";

int shm\_size = 4096;

//Create shared memory file

int shm\_fd = shm\_open(name, O\_CREAT|O\_RDWR, 0777);

ftruncate(shm\_fd, shm\_size); //Reset the file size

// create fork

pid\_t pid=fork();

if(pid==-1) {

…

}

//child process

else if(pid == 0) {

void \*shm\_fptr=mmap(0,shm\_size,PROT\_READ|PROT\_WRITE,MAP\_SHARED,shm\_fd,0);

…

while(n!=1) {

…

// write into share memory

str\_len=sprintf(buff,"%d ",n);

sprintf(shm\_fptr,"%s",buff);

shm\_fptr+=str\_len;

memset(buff,0,str\_len);

}

}

//parent process

else if(pid >0) {

…

// read from share memory

void\* shm\_fptr=mmap(0,shm\_size,PROT\_READ,MAP\_SHARED,shm\_fd,0);

printf("%s\n",(char\*)shm\_fptr);

shm\_unlink(name);//delete share memory

exit(0);

}

return 0;

}

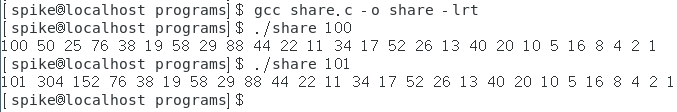
通过父子进程共享一块共享文件空间，使得交换数据非常快，减小了大量的交换时间。

实际上工作原理就是子进程把数据写进gongxaingkongjian1，子进程结束后触发父进程读取，然后输出计算结果。

tips：本处使用的空间是**shm**，相当于直接共享内存，交换速度极快。

dev/shm/是linux下一个非常有用的目录，因为这个目录不在硬盘上，而是在内存里。因此在linux下，就不需要大费周折去建ramdisk，直接使用/dev/shm/就可达到很好的优化效果。

执行结果：



## 2.4 pipe通信

pipe通信很像我们在计算机网络里面学习的通信方式。这个管道的本质是一个半双工系统，有两个管道结合到一起的，一个收一个发。

管道本身有着容量限制，但一般不会使用到极限。

pipe框架如下：

int main()

{

int fd1[2],fd2[2];

int pipe1=pipe(fd1);

int pipe2=pipe(fd2);

char buff[2049]={0};

if (pipe1<0||pipe2<0) {

…

}

// create fork

pid\_t pid=fork();

if(pid==-1) {

…

}

//child process

else if(pid == 0) {

close(fd1[0]); //Turn off pipe1 read

close(fd2[1]); //Turn off pipe1 read

…

}

//parent process

else if(pid >0) {

close(fd1[1]);//Turn off pipe1 write

close(fd2[0]);//Turn off pipe2 read

read(fd1[0],buff,2048);

…

write(fd2[1],buff,strlen(buff));

wait(NULL);

exit(0);

}

return 0;

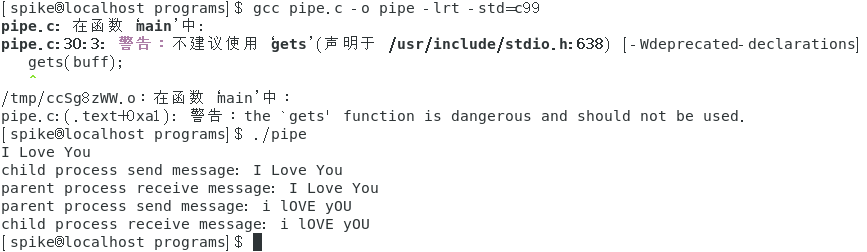
}

在这个框架下，我们就可以使用管道进行相互通信。很简单的read和write就可以实现数据间的交换。这个fd所指向的2空间实际上就是分配给两端的。

pipe函数定义中的fd参数是一个大小为2的一个数组类型的指针。该函数成功时返回0，并将一对打开的文件描述符值填入fd参数指向的数组。失败时返回 -1并设置errno。

通过pipe函数创建的这两个文件描述符 fd[0] 和 fd[1] 分别构成管道的两端，往 fd[1] 写入的数据可以从 fd[0] 读出。并且 fd[1] 一端只能进行写操作，fd[0] 一端只能进行读操作，不能反过来使用。要实现双向数据传输，可以使用两个管道。

执行结果：



# 3. 实验总结

在这个实验里，我们主要进行了代码的编写。在这个过程中，显示的理解了代码的运作原理和设计思路。但是我们在深究的过程中，发现操作系统对于这一系列操作的支持是有迹可循的：

* fork

一个进程，包括代码、数据和分配给进程的资源。fork（）函数通过系统调用创建一个与原来进程几乎完全相同的进程，也就是两个进程可以做完全相同的事，但如果初始参数或者传入的变量不同，两个进程也可以做不同的事。  
    一个进程调用fork（）函数后，系统先给新的进程分配资源，例如存储数据和代码的空间。然后把原来的进程的所有值都复制到新的新进程中，只有少数值与原来的进程的值不同。相当于克隆了一个自己。

* shm

dev/shm/是linux下一个非常有用的目录，因为这个目录不在硬盘上，而是在内存里。因此在linux下，就不需要大费周折去建ramdisk，直接使用/dev/shm/就可达到很好的优化效果。

使用shm\_open来操作共享内存。shm\_open最主要的操作也是默认的操作就是在/dev/shm/下面，建立一个文件。文件名字是用户自己输入的。要点一定要用ftruncate把文件大小于设置为共享内存大小。

* pipe

管道内部传输的数据是字节流，这和TCP字节流的概念相同。但它们又存在细微的差别。应用层程序能往一个TCP连接中写入多少字节的数据，取决于对方接受窗口的大小和本端的拥塞窗口的大小。而管道的话本身拥有一个容量限制，它规定如果管道的写端应用程序不将管道中数据读走的话，该管道最多还能被写入多少字节的数据。管道容量的大小默认是65536字节。我们也可以使用fcntl函数来修改管道容量。