山东大学 计算机科学与技术 学院

操作系统 课程实验报告

学号:201800130005	姓名： 张畅	班级： 2018 计科 3 班
实验题目：线程和管道通信实验		
实验学时：2		实验日期：2020.11.20

实验目的：

通过 Linux 系统中线程和管道通信机制的实验，加深对于线程控制和管道通信概念的理解，观察和体验

并发进程、线程间的通信和协作的效果 ，练习利用无名管道进行进程、线程通信的编程和调试技术。

实验过程中遇到和解决的问题：

（记录实验过程中遇到的问题，以及解决过程和实验结果。可以适当配以关键代码辅助说明，但不要大段贴代码。）

在实验的过程中，管道知识的应用是该实验重要的一部分，需要通过管道的传输功能实现数据的传输和沟通。在学习管道的相关知识的过程中，在管道的实用上出现了一定的问题并解决了该问题。

在阅读了实验报告册中的相应知识之后，了解了管道是一个半双工的数据结构，因此采用了从 0 端写入 1 端读出的方式进行传输，经过代码运行之后发现出现了问题：

```
tang@ubuntu:~/OS-exp/OS/OS-resources/exp2-code$ ./tpipe
Input upper limit:(Attention:upper limit must be upper than 0)
x:5
thread2 write:f(y=1)=1
thread2 write:f(y=2)=1
thread2 write:f(y=3)=2
thread3 read and make answer:f(x,y=1)=0
thread3 read and make answer:f(x,y=2)=0
thread3 read and make answer:f(x,y=3)=0
thread3 read and make answer:f(x,y=4)=0
thread3 read and make answer:f(x,y=5)=0
thread1 write:f(x=1)=1
thread1 write:f(x=2)=2
thread1 write:f(x=3)=6
thread1 write:f(x=4)=24
thread1 write:f(x=5)=120
thread2 write:f(y=4)=3
thread2 write:f(y=5)=5
```

程序无论是读还是写都存在问题发生了阻塞，经过思考尝试将读/写端交换，发现结果是正确的：

```
tang@ubuntu:~/OS-exp/OS/OS-resources/exp2-code$ ./tpipe
Input upper limit:(Attention:upper limit must be upper than 0)
x:5
thread1 write:f(x=1)=1
thread1 write:f(x=2)=2
thread1 write:f(x=3)=6
thread1 write:f(x=4)=24
thread1 write:f(x=5)=120
thread2 write:f(y=1)=1
thread2 write:f(y=2)=1
thread2 write:f(y=3)=2
thread2 write:f(y=4)=3
thread2 write:f(y=5)=5
thread3 read and make answer:f(x,y=1)=2
thread3 read and make answer:f(x,y=2)=3
thread3 read and make answer:f(x,y=3)=8
thread3 read and make answer:f(x,y=4)=27
thread3 read and make answer:f(x,y=5)=125
```

经过分析和资料的查询，由于使用习惯的问题，一般在进行管道的读写时，都遵循从 0 端读 1 端写的方式，反之容易造成代码逻辑上的问题从而导致出现 bug。在后面的管道使用中，需要总结经验，并正确的使用管道来进行数据通讯，方便取得理想的结果。

实验步骤与内容：

1、对线程基本操作的学习

线程间的通讯知识是该实验的基础，因此了解线程的基本知识至关重要：

创建线程：

```
Int pthread_create(pthread_t *thread,pthread_attr_t *attr,
void *(*start_routine)(void *) Void *arg);
```

pthread_create 函数创建一个新的线程。

退出线程：

```
void pthread_exit(void *retval);
```

pthread_exit 函数使用函数 pthread_cleanup_push 调用任何用于该线程的

清除处理函数，然后中止当前进程的执行。

挂起线程：

```
int pthread_join(pthread_t th, void **thread_return);
int pthread_detach(pthread_t th);
```

函数 pthread_join 用于挂起当前线程，直到 th 指定的线程终止运行为止。

通过在 pthread_create 调用创建一个线程时使用 PTHREAD_CREATE_DETACHED

属性或者使用pthread_detach 可以让线程处于被分离状态。

2、对管道通信知识的学习

管道被应用于两个线程/进程间的信息传输，经过对管道知识的测试，基本掌握了管道传输的方法：

管道的创建：

```
int pipe(int pipe_id[2]);
```

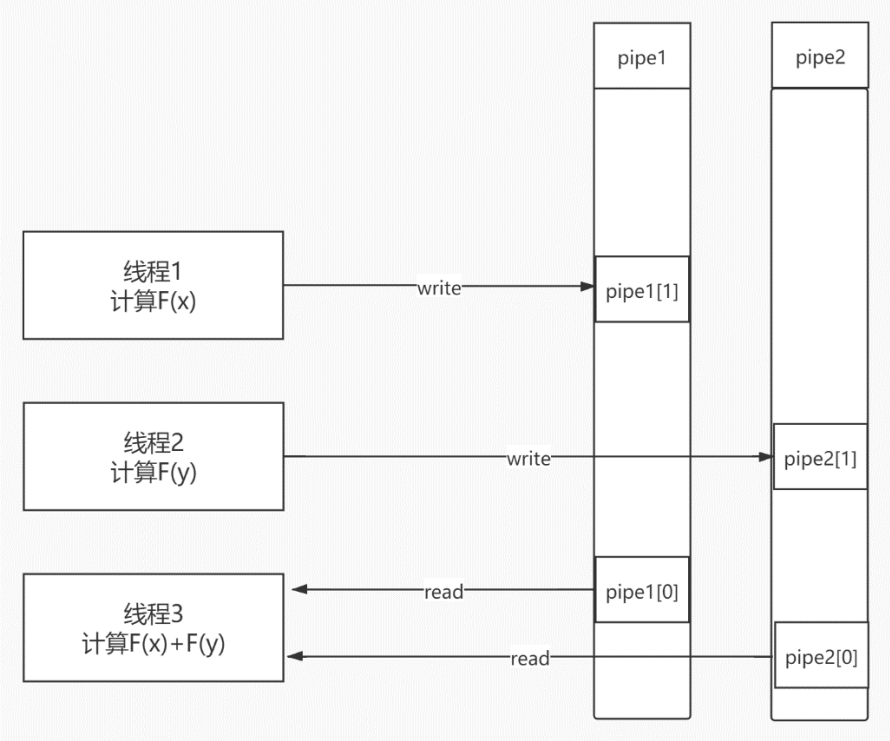
pipe 建立一个无名管道，pipe_id[0] 和 pipe_id[1] 中将放入管道两端的描述符。如果 pipe 执行成功返回 0，出错返回 -1。

管道的内容读写：

```
ssize_t read(int pipe_id,const void *buf,size_t count);
ssize_t write(int pipe_id,const void *buf,size_t count);
```

read 和 write 分别在管道的两端进行读和写。

3、设计实验的基本逻辑结构



4、编写程序，测试结果

根据上述的逻辑结构编写了程序，并且进行了运行测试，验证了设计思路的正确性，测试结果如下图所示：

```
tang@ubuntu:~/OS-exp/OS/OS-resources/exp2-code$ ./tpipe
Input upper limit:(Attention:upper limit must be upper than 0)
x:5
thread1 write:f(x=1)=1
thread1 write:f(x=2)=2
thread1 write:f(x=3)=6
thread1 write:f(x=4)=24
thread1 write:f(x=5)=120
thread2 write:f(y=1)=1
thread2 write:f(y=2)=1
thread2 write:f(y=3)=2
thread2 write:f(y=4)=3
thread2 write:f(y=5)=5
thread3 read and make answer:f(x,y=1)=2
thread3 read and make answer:f(x,y=2)=3
thread3 read and make answer:f(x,y=3)=8
thread3 read and make answer:f(x,y=4)=27
thread3 read and make answer:f(x,y=5)=125
```

实验总结：

通过这次实验，对进程、线程间的通信有了更加深刻的认识：

在多道程序设计系统中，同一时刻可能有许多进程，这些进程之间存在两种基本关系：竞争关系和协作关系。进程的互斥、同步、通信都是基于这两种基本关系而存在的，为了解决进程间竞争关系（间接制约关系）而引入进程互斥；为了解决进程间松散的协作关系(直接制约关系)而引入进程同步；为了解决进程间紧密的协作关系而引入进程通信。竞争关系就是系统中的多个进程之间彼此无关，它们并不知道其他进程的存在，并且也不受其他进程执行的影响，由于这些进程共用了一套计算机系统资源，因而， 必然要出现多个进程竞争资源的问题。当多个进程竞争共享硬设备、存储器、处理器 和文件等资源时，操作系统必须协调好进程对资源的争用；协作关系某些进程为完成同一任务需要分工协作，由于合作的每一个进程都是独立地以不可预知的速度推进，这就需要相互协作的进程在某些协调点上协 调各自的工作。当合作进程中的一个到达协调点后，在尚未得到其伙伴进程发来的消息或信号之前应阻塞自己，直到其他合作进程发来协调信号或消息后方被唤醒并继续执行。这种协作进程之间相互等待对方消息或信号的协调关系称为进程同步。

同时通过这次实验，对管道的通信机制了解地更加深刻：

管道实际上是一种固定大小的缓冲区，管道对于管道两端的进程而言，就是一个文件，但它不是普通的文件，它不属于某种文件系统，而是自立门户，单独构成一种文件系统，并且只存在于内存中。它类似于通信中半双工信道的进程通信机制，一个管道可以实现双向 的数据传输，而同一个时刻只能最多有一个方向的传输，不能两个方向同时进行。管道的容 量大小通常为内存上的一页，它的大小并不是受磁盘容量大小的限制。当管道满时，进程在 写管道会被阻塞，而当管道空时，进程读管道会被阻塞,不能同时进行双向传输。匿名管道只能单向；命名管道可以双向；管道是内存中的，可以有多个进程对其进行读操作；也可以有多个进程写，只不过不能同时写。

通过对管道的使用，总结了管道通信的基本原则：

管道可以实现父子进程之间的通信，并且读取进程会阻塞等到所有的写入进程关闭后，才会收到文件结束符。由此通过一下步骤可以实现父子间同步(1)父进程在创建子进程之前构建管道;(2)每个子进程会继承管道的写入端的文件描述符并在完成动作之后关闭这些描述符;(3)当所有的子进程都关闭了管道的写入端描述符之后，父进程在管道的 read（）就会结束并返回文件结束。

附录：程序源代码

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <pthread.h>
void task1(int *); // 线程 1 执行函数原型
void task2(int *); // 线程 2 执行函数原型
void task3(int *); // 线程 3 执行函数原型
int pipe1[2], pipe2[2]; // 存放的两个无名管道标号
pthread_t thrd1, thrd2, thrd3;    // 存放的两个线程标识
int max_num;
```

```
int main(int argc, char *argv[])
{
    int ret;
    int num1, num2, num3;
    printf("Input upper limit:(Attention:upper limit must be upper than 0)\nx:");
    scanf("%d",&max_num);
    // 使用 pipe() 系统调用建立两个无名管道。建立不成功程序退出,执行终止
    if (pipe(pipe1) < 0)
    {
        perror("pipe1 not create");
        exit(EXIT_FAILURE);
    }
    if (pipe(pipe2) < 0)
    {
        perror("pipe2 not create");
        exit(EXIT_FAILURE);
    }
    // 使用 pthread_create 系统调用建立两个线程。建立不成功程序退出,执行终止
    num1 = 1;
    ret = pthread_create(&thrd1, NULL, (void *)task1, (void *)&num1);
    if (ret)
    {
        perror("pthread_create: task1");
        exit(EXIT_FAILURE);
    }
    num2 = 2;
    ret = pthread_create(&thrd2, NULL, (void *)task2, (void *)&num2);
    if (ret)
    {
        perror("pthread_create: task2");
        exit(EXIT_FAILURE);
    }
    num3 = 3;
    ret = pthread_create(&thrd3, NULL, (void *)task3, (void *)&num3);
    if (ret)
    {
        perror("pthread_create: task3");
        exit(EXIT_FAILURE);
    }
    pthread_join(thrd1, NULL);
    pthread_join(thrd2, NULL);
    pthread_join(thrd3, NULL);
    exit(EXIT_SUCCESS);
```

```
}
// 线程 1 执行函数,它首先向管道写,然后从管道读
void task1(int *num)
{
    int x = 1;
    int ans1=1;

    do
    {
        if(x==1){
            ans1=1;
        }
        else{
            ans1=ans1*x;
        }
        write(pipe1[1], &ans1, sizeof(int));
        printf("thread%d write:f(x=%d)=%d\n",*num,x++,ans1);
    } while (x <= max_num);
    // 读写完成后,关闭管道
    close(pipe1[1]);
}
// 线程 2 执行函数,它首先从管道读,然后向管道写
void task2(int *num)
{
    int y = 1;
    int ans_n1=1;
    int ans_n2=1;
    int ans=1;

    do
    {
        if(y==1 || y==2){
            ans=1;
            ans_n1=1;
            ans_n2=1;
        }
        else{
            ans=ans_n1+ans_n2;
            ans_n2=ans_n1;
            ans_n1=ans;
        }
        write(pipe2[1], &ans, sizeof(int));
        printf("thread%d write:f(y=%d)=%d\n",*num,y++,ans);
    } while (y <= max_num);
```

```
    // 读写完成后,关闭管道
    close(pipe2[1]);
}
void task3(int * num)
{
    int ans=0;
    int ans_fx;
    int ans_fy;
    int flag=1;

    do
    {
        read(pipe1[0], &ans_fx, sizeof(int));
        read(pipe2[0], &ans_fy, sizeof(int));
        ans=ans_fx+ans_fy;
        printf("thread%d read and make answer:f(x,y=%d)=%d\n",*num,flag++,ans);
    } while (flag <= max_num);
    // 读写完成后,关闭管道
    close(pipe1[0]);
    close(pipe2[0]);
}
```