计算机学院 操作系统 课程实验报告

实验题目： 线程和管道通信		学号：202200130048
日期：10.16	班级： 6	姓名： 陈静雯
Email：1205037094@qq.com		

实验步骤与现象：

1. 示例实验 tpipe

```
orange@orange-VirtualBox:~/czsystem$ gmake
gcc -g -c tpipe.c
gcc     tpipe.o  -l pthread -o tpipe
orange@orange-VirtualBox:~/czsystem$ ./tpipe
thread2 read: 1
thread1 read: 2
thread2 read: 3
thread1 read: 4
thread2 read: 5
thread1 read: 6
thread2 read: 7
thread1 read: 8
thread2 read: 9
thread1 read: 10
```

线程 1 和线程 2 交替的将整数 X 的值从 1 加到了 10。线程 1 每次循环向管道 1 的 1 端写入变量 X 的值,并从管道 2 的 0 端读一整数写入 X 再对 X 加 1，直到 X 大于 10；线程 2 每次循环从管道 1 的 0 端读一个整数放入变量 X 中,并对 X 加 1 后写入管道 2 的 1 端，直到 X 大于 10

2. 示例实验 ppipe

```
orange@orange-VirtualBox:~/czsystem/test2/pipep$ gmake
gcc -g -c ppipe.c
gcc ppipe.o -o ppipe
orange@orange-VirtualBox:~/czsystem/test2/pipep$ ./ppipe
child 3766 read: 1
parent 3765 read: 2
child 3766 read: 3
parent 3765 read: 4
child 3766 read: 5
parent 3765 read: 6
child 3766 read: 7
parent 3765 read: 8
child 3766 read: 9
parent 3765 read: 10
```

父子进程交替的将整数 X 的值从 1 加到了 10。过程类似 tpipe

3. 独立实验

```
orange@orange-VirtualBox:~/czsystem/test2$ ./test
thread1 write: 1
thread2 write: 1
thread3 read1: 1, read2: 1, f(x)+f(y)= 2
thread2 write: 1
thread1 write: 2
thread3 read1: 2, read2: 1, f(x)+f(y)= 3
thread1 write: 6
thread2 write: 2
thread3 read1: 6, read2: 2, f(x)+f(y)= 8
thread2 write: 3
thread1 write: 24
thread3 read1: 24, read2: 3, f(x)+f(y)= 27
thread1 write: 120
thread2 write: 5
thread3 read1: 120, read2: 5, f(x)+f(y)= 125
thread2 write: 8
thread1 write: 720
thread3 read1: 720, read2: 8, f(x)+f(y)= 728
thread1 write: 5040
thread2 write: 13
thread3 read1: 5040, read2: 13, f(x)+f(y)= 5053
thread1 write: 40320
thread2 write: 21
thread3 read1: 40320, read2: 21, f(x)+f(y)= 40341
thread2 write: 34
thread1 write: 362880
thread3 read1: 362880, read2: 34, f(x)+f(y)= 362914
thread2 write: 55
thread1 write: 3628800
thread3 read1: 3628800, read2: 55, f(x)+f(y)= 3628855
```

f(x,y) = f(x) + f(y)

其中： f(x)=1 (x=1) ， f(x) = f(x-1) * x (x >1) ； f(y)=1 (y=1,2) ， f(y) = f(y-1) + f(y-2) (y> 2)

三个线程 123，线程 1 是 f（x），线程 2 是 f（y），线程 3 是 f（x，y）。x、y 从 1 开始，每次线程 1 在 pipe1 写入 f（x）的值，再从 pipe3 中读数据看是否正确传递，同线程 2 在 pipe2 中写入 f（y），再从 pipe4 读取。Pipe1 和 pipe3 是连接线程 1 和 3，pipe2 和 4 连接线程 2 和 3，线程 3 读到 12 传来的数据，将它俩相加。

结论分析：

1. 反映出操作系统教材中讲解的进/线程协作和进/线程通信概念的哪些特征和功能？

（1）资源共享

线程之间通过管道（pipe）进行通信，这是一种共享内核资源的方式。每个管道实际上是一段内核管理的缓冲区，线程可以通过读写这些缓冲区来交换数据。

```
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 isstop=1;
int pipe1[2],pipe2[2],pipe3[2],pipe4[2]; //13是线程13通信，24是线程23通信
pthread_t thrd1,thrd2,thrd3; //存放第两个线程标识
```

（2）消息传递

线程 1 和线程 2 通过管道向线程 3 发送数据，线程 3 再通过管道将结果返回给线程 1 和线程 2。这种消息传递机制是典型的线程间通信方式，适用于需要异步处理的场景。

```
    write(pipe1[1],&x,sizeof(int));
    printf("thread%d write: %d\n",*num,x);
    read(pipe3[0],&r,sizeof(int));
    n++;
```

（3）任务调度

三个线程，每个线程执行不同的任务。操作系统负责调度这些线程，使它们并发执行。这种并发执行提高了程序的效率，但也带来了同步和协调的挑战。

```
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 isstop=1;
int pipe1[2],pipe2[2],pipe3[2],pipe4[2]; //13是线程13通信，24是线程23通信
pthread_t thrd1,thrd2,thrd3; //存放第两个线程标识
```

（4）错误处理

检查返回值，并在失败时输出错误信息并退出程序。

```
if(pipe(pipe1) < 0){
    perror("pipe1 not create");
    exit(EXIT_FAILURE);
}
if(pipe(pipe2) < 0)
{
    perror("pipe2 not create");
    exit(EXIT_FAILURE);
}
```

（5）资源管理

在每个线程结束时，关闭使用的管道描述符，确保资源得到正确释放，避免了资源泄露。

```
    }while(n<=10&&isstop);
    isstop=0;
    //读写完成后,关闭管道
    close(pipe1[1]);
    close(pipe3[0]);
}
```

2. 在真实的操作系统中它是怎样**实现**和反映出教材中进/线程通信概念的

（1）共享内存

操作系统提供了一种机制，允许多个进程或线程共享同一块内存区域。这种共享内存区域通常由内核管理，进程或线程可以通过映射这块内存来访问数据。

```
void task3(int *); //线程 3 执行函数原型
int isstop=1;
int pipe1[2],pipe2[2],pipe3[2],pipe4[2]; //13是线程13通信，24是线程23通信
pthread_t thrd1,thrd2,thrd3; //存放第两个线程标识
int main(int argc,char *arg[])
```

（2）管道

管道是一种半双工的通信机制，允许数据在一个方向上流动。操作系统提供了一组系统调用（如 pipe ,read ,write）来创建和操作管道。

```
int num1,num2,num3;
//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);
}
```

```
do{
    write(pipe1[1],&x,sizeof(int));
    printf("thread%d write: %d\n",*num,x);
    read(pipe3[0],&r,sizeof(int));
    n++;
    x=n*x;
```

3. 对于进/线程协作和进/线程通信的概念和实现有哪些新的理解和认识？

（1）通信机制的多样性

共享内存 ， 消息队列 ， 管道和命名管道

（2）线程 vs 进程

线程在同一进程中共享地址空间，通信开销小，但需要更多的同步管理；进程有独立的地址空间，通信开销大，但隔离性好。根据应用场景选择合适的模型。

（3）同步与互斥的必要性

4. 管道机制的机理是什么？

管道是一种半双工的通信机制，数据只能在一个方向上传输。通常用于父子进程之间的通信。默认情况下，管道是匿名的，只能用于具有亲缘关系的进程之间（如父子进程）。

（1） 创建管道：使用 pipe 系统调用来创建管道。0 读 1 写

（2） 读写操作：当一个进程向写端写入数据时，数据被存储到内核缓冲区中。另一个进程可以从读端读取数据，数据从内核缓冲区中移除。

（3） 阻塞模式：默认情况下，管道操作是阻塞的。

读操作：如果读端没有数据可读，读操作会阻塞，直到有数据可用。

写操作：如果写端缓冲区已满，写操作会阻塞，直到有空间可用。

非阻塞模式：可以通过设置文件描述符的属性来启用非阻塞模式。

5. 怎样利用管道完成进/线程间的协作和通信？

（1）创建管道：使用 pipe 系统调用创建一个管道。管道会返回两个文件描述符，一个用于读取（pipe [0]），一个用于写入（pipe [1]）。

（2）创建子进程/线程

（3）关闭不必要的文件描述符

（4）读写操作

```
(pid == 0){
  //子进程负责从管道 1 的 0 端读,管道 2 的 1 端写,
  //所以关掉管道 1 的 1 端和管道 2 的 0 端。
  close(pipe1[1]);
  close(pipe2[0]);
  //每次循环从管道 1 的 0 端读一个整数放入变量 X 中,
  //并对 X 加 1 后写入管道 2 的 1 端，直到 X 大于 10
  do{
      read(pipe1[0],&x,sizeof(int));
      printf("child %d read: %d\n",getpid(),x++);
      write(pipe2[1],&x,sizeof(int));
  }while( x<=9 );
  //读写完成后,关闭管道
  close(pipe1[0]);
  close(pipe2[1]);
  //子进程执行结束
  exit(EXIT_SUCCESS);
```

```
do{
    write(pipe1[1],&x,sizeof(int));
    printf("thread%d write: %d\n",*num,x);
    read(pipe3[0],&r,sizeof(int));
    n++;
    x=n*x;
}while(n<=10&&isstop);
isstop=0;
//读写完成后,关闭管道
close(pipe1[1]);
close(pipe3[0]);
```