# 操作系统实验二报告

# 问题：

1、练习管道通信

2、练习基于CS模式的进程通信并对进程通信的代码进行说明。总结每种通信的特点，使用场景

# 一、解决方案：

## 1.1管道通信：

管道是链接两个实体之间单项数据传输的导管。



以消费者生产者为例，创建管道输入输出流，将管道输入输出流相连接，生产者与消费者分别与管道输出和输入流相关联。

### 1.1.1管道流特点：

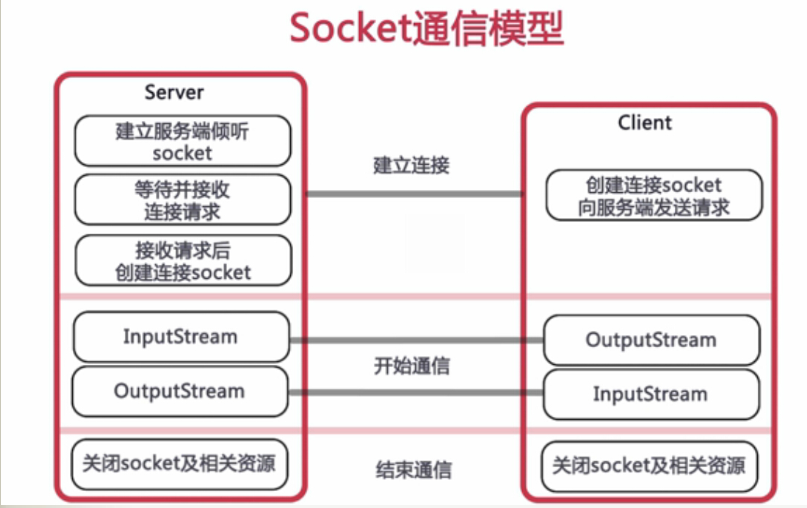
1. 管道数据流向的单向的，数据只能从一个进程(线程)流向另一个进程(线程)，如果要进行双向通信，必须建立两个管道。

2. 管道的读数据是一次性操作，数据一旦被读，它就从管道中被抛弃，释放空间以便写更多数据。

3. 当管道输出流write()导致管道缓冲区变满时，管道的write()调用将默认的被阻塞，等待缓冲区的数据被读取。同样的读进程也可能工作得比写进程块。当所有当前进程数据被读取时，管道变空。当这种情况发生时，一个随后的read()调用将默认被阻塞，等待缓冲区数据，这解决了read()调用返回文件结束的问题。

4. 管道输出流或者管道输入流的提前关闭，不会影响到对端流。比如输出流提前结束，输入流不会产生异常；输入流的提前结束也不会影响到输出流。

## 1.2 CS模式的进程通信：



Socket通信步骤：

1.建立服务端ServerSocket和客户端Socke

2.打开连接到Socket的输出输入流

3.按照协议进行读写操作

4.关闭相对应的资源

### 2.1.1socket通讯特点：

服务器端

（1）建立服务器端的Socket，开始监听整个网络中的连接请求。

（2）当检测到来自客户端的连接请求时，向客户端发送收到连接请求的信息，并建立与客户端之间的连接。

（3）当完成通信后，服务器端关闭与客户端的Socket连接。

服务器端

（1）建立客户端的Socket，确定要连接的服务器的主机名和端口。

（2）发送连接请求到服务器，并等待服务器的反馈信息。

（3）连接成功后，与服务器进行数据的交互。

（4）数据处理完毕后，关闭自身的Socket连接。

# 二、代码：

## 2.1 管道通信：

import java.io.IOException;

import java.io.PipedInputStream;

import java.io.PipedOutputStream;

public class testPipeConnection {

public static void main(String[] args) {

/\*\*\* 创建管道输出流\*/

PipedOutputStream pos = new PipedOutputStream();

/\*\*\* 创建管道输入流\*/

PipedInputStream pis = new PipedInputStream();

try {

/\*\*\* 将管道输入流与输出流连接\*/

pos.connect(pis);

} catch (IOException e) {

e.printStackTrace();

}

/\*\*\* 创建生产者线程\*/

Producer p = new Producer(pos);

/\*\*\* 创建消费者线程\*/

Consumer1 c1 = new Consumer1(pis);

/\*\*\* 启动线程\*/

p.start();

c1.start();

}

}

## 2.2 Socket通信:

public class Server {

/\*\*

\* Socket服务端

\*/

public static void main(String[] args) {

try {

ServerSocket serverSocket=new ServerSocket(8888);

System.out.println("服务端已启动，等待客户端连接..");

Socket socket=serverSocket.accept();//侦听并接受到此套接字的连接,返回一个Socket对象

//根据输入输出流和客户端连接

InputStream inputStream=socket.getInputStream();//得到一个输入流，接收客户端传递的信息

InputStreamReader inputStreamReader=new InputStreamReader(inputStream);//提高效率，将自己字节流转为字符流

BufferedReader bufferedReader=new BufferedReader(inputStreamReader);//加入缓冲区

String temp=null;

String info="";

while((temp=bufferedReader.readLine())!=null){

info+=temp;

System.out.println("已接收到客户端连接");

System.out.println("服务端接收到客户端信息："+info+",当前客户端ip为："+socket.getInetAddress().getHostAddress());

}

OutputStream outputStream=socket.getOutputStream();//获取一个输出流，向服务端发送信息

PrintWriter printWriter=new PrintWriter(outputStream);//将输出流包装成打印流

printWriter.print("你好，服务端已接收到您的信息");

printWriter.flush();

socket.shutdownOutput();//关闭输出流

//关闭相对应的资源

printWriter.close();

outputStream.close();

bufferedReader.close();

inputStream.close();

socket.close();

} catch (IOException e) {

e.printStackTrace();

} }}

public class Client {

/\*\*

\* Socket客户端

\*/

public static void main(String[] args) {

try {

//创建Socket对象

Socket socket=new Socket("localhost",8888);

//根据输入输出流和服务端连接

OutputStream outputStream=socket.getOutputStream();//获取一个输出流，向服务端发送信息

PrintWriter printWriter=new PrintWriter(outputStream);//将输出流包装成打印流

printWriter.print("服务端你好，我是Balla\_兔子");

printWriter.flush();

socket.shutdownOutput();//关闭输出流

InputStream inputStream=socket.getInputStream();//获取一个输入流，接收服务端的信息

InputStreamReader inputStreamReader=new InputStreamReader(inputStream);//包装成字符流，提高效率

BufferedReader bufferedReader=new BufferedReader(inputStreamReader);//缓冲区

String info="";

String temp=null;//临时变量

while((temp=bufferedReader.readLine())!=null){

info+=temp;

System.out.println("客户端接收服务端发送信息："+info);

}

//关闭相对应的资源

bufferedReader.close();

inputStream.close();

printWriter.close();

outputStream.close();

socket.close();

} catch (UnknownHostException e) {

e.printStackTrace();

} catch (IOException e) {

e.printStackTrace();

}

}

}