实验2 socket封装

互联网络程序设计实验

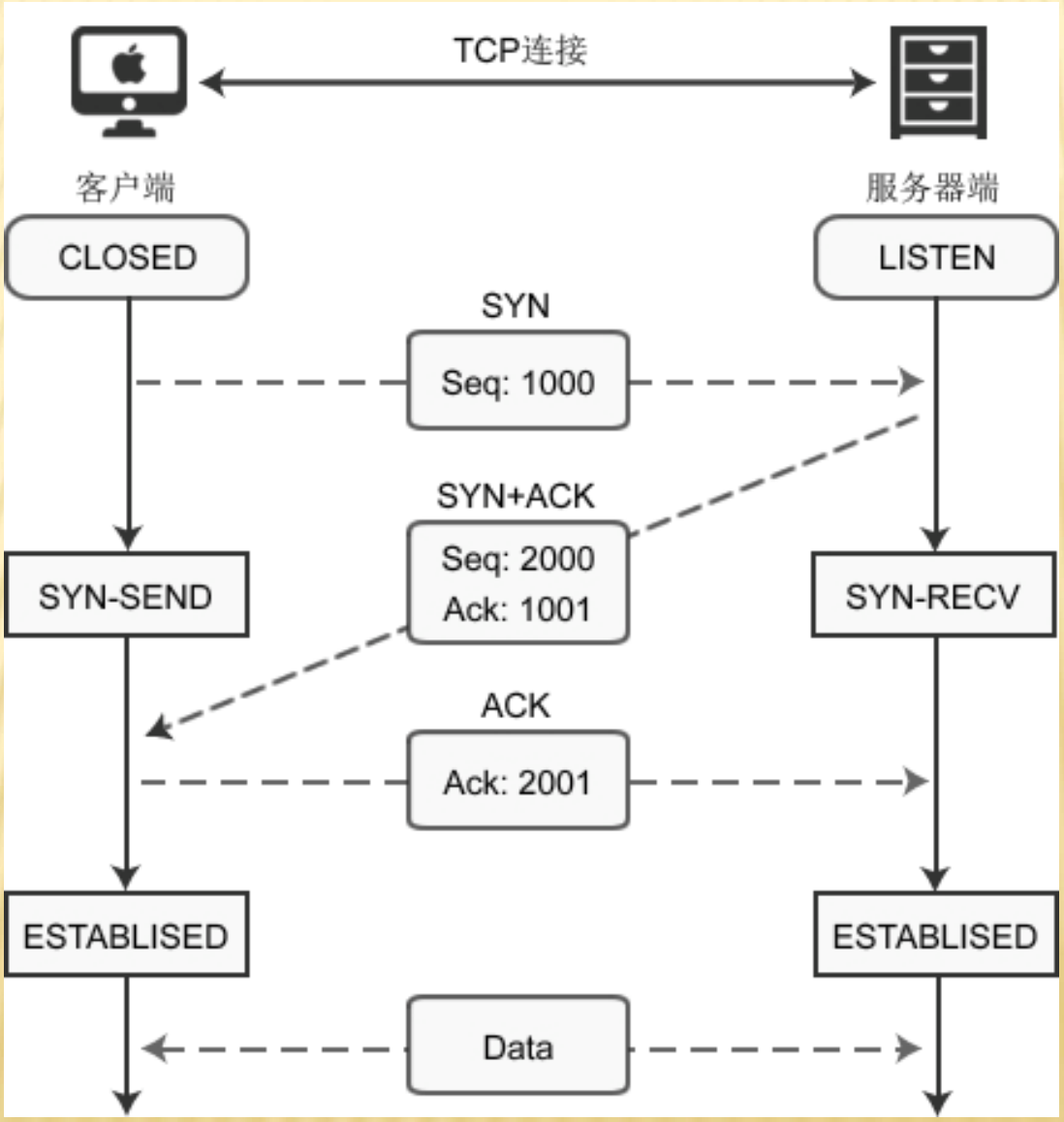
实验目标

- 以C++类封装socket

实验原理

- Socket是用于进程之间通信的一种机制，是网络编程的核心，本实验主要关心三类socket，TCP、UDP和UNIX域。
- 本实验在linux上对以上三种类型的socket进行封装，使其更加简单易用，socket编程根据当前程序是客户端还是服务器流程不尽相同。

TCP连接的建立（三次握手）



- 客户端调用 socket() 函数创建套接字后，因为没有建立连接，所以套接字处于CLOSED状态；服务器端调用 listen() 函数后，套接字进入LISTEN状态，开始监听客户端请求。

- 这个时候，客户端开始发起请求：
- 1) 当客户端调用 connect() 函数后，TCP协议会组建一个数据包，并设置 SYN 标志位，表示该数据包是用来建立同步连接的。同时生成一个随机数字 1000，填充“序号（Seq）”字段，表示该数据包的序号。完成这些工作，开始向服务器端发送数据包，客户端就进入了SYN-SEND状态。

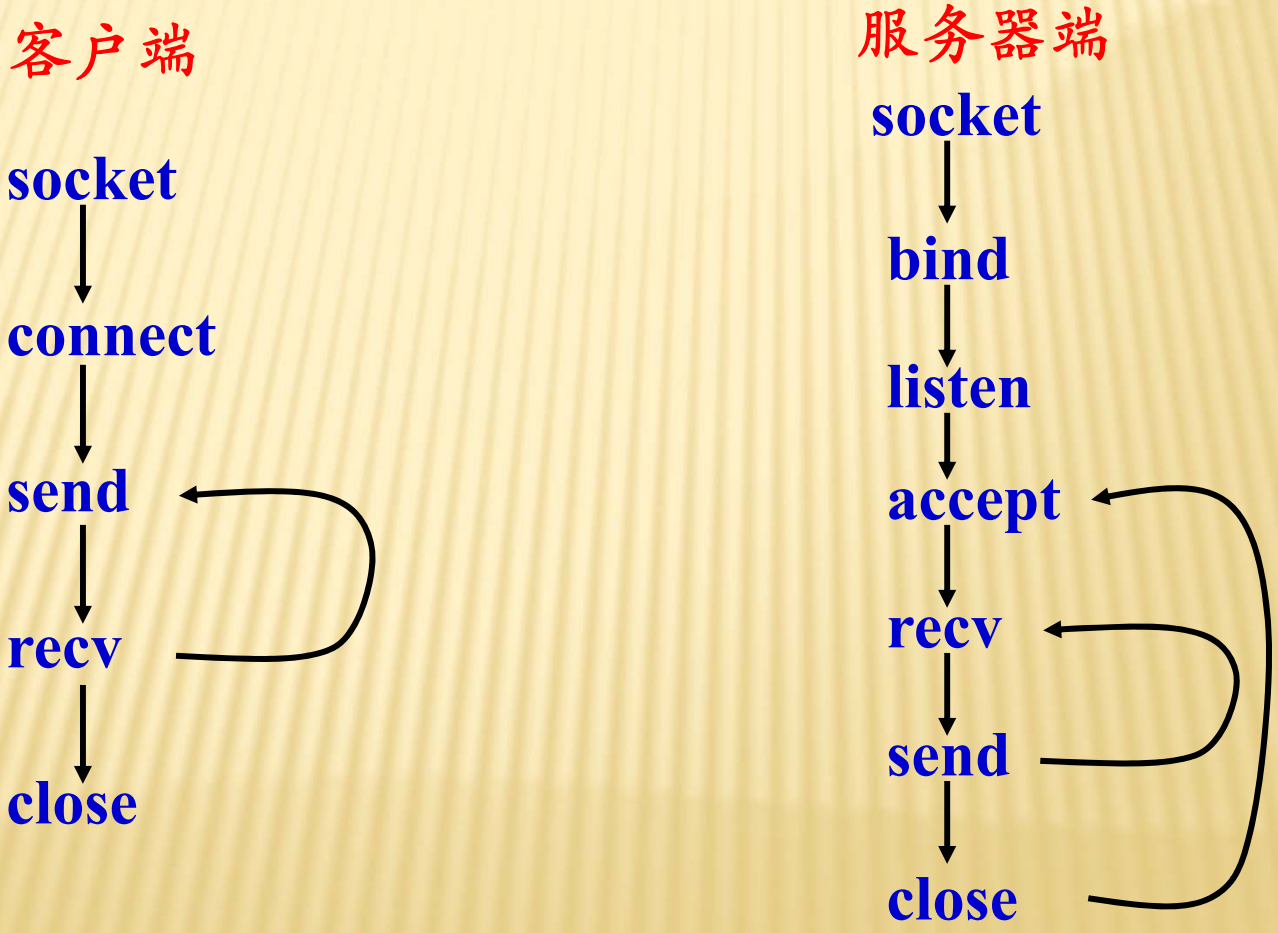
- 2) 服务器端收到数据包，检测到已经设置了 SYN 标志位，就知道这是客户端发来的建立连接的“请求包”。服务器端也会组建一个数据包，并设置 SYN 和 ACK 标志位，SYN 表示该数据包用来建立连接，ACK 用来确认收到了刚才客户端发送的数据包。
- 服务器生成一个随机数 2000，填充“序号（Seq）”字段。2000 和客户端数据包没有关系。
- 服务器将客户端数据包序号（1000）加1，得到1001，并用这个数字填充“确认号（Ack）”字段。
- 服务器将数据包发出，进入SYN-RECV状态。

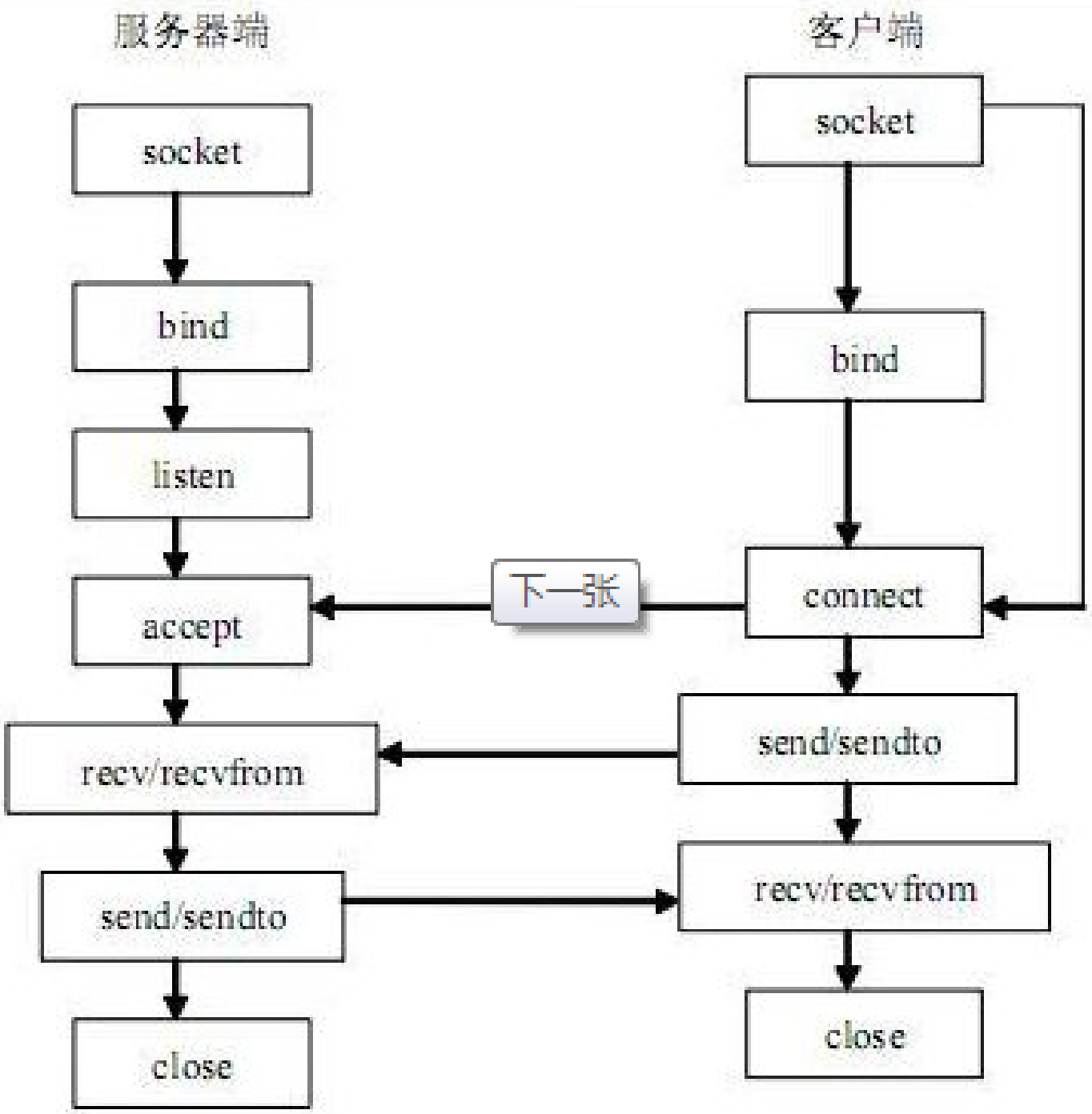
- 3) 客户端收到数据包，检测到已经设置了 SYN 和 ACK 标志位，就知道这是服务器发来的“确认包”。客户端会检测“确认号（Ack）”字段，看它的值是否为 1000+1，如果是就说明连接建立成功。

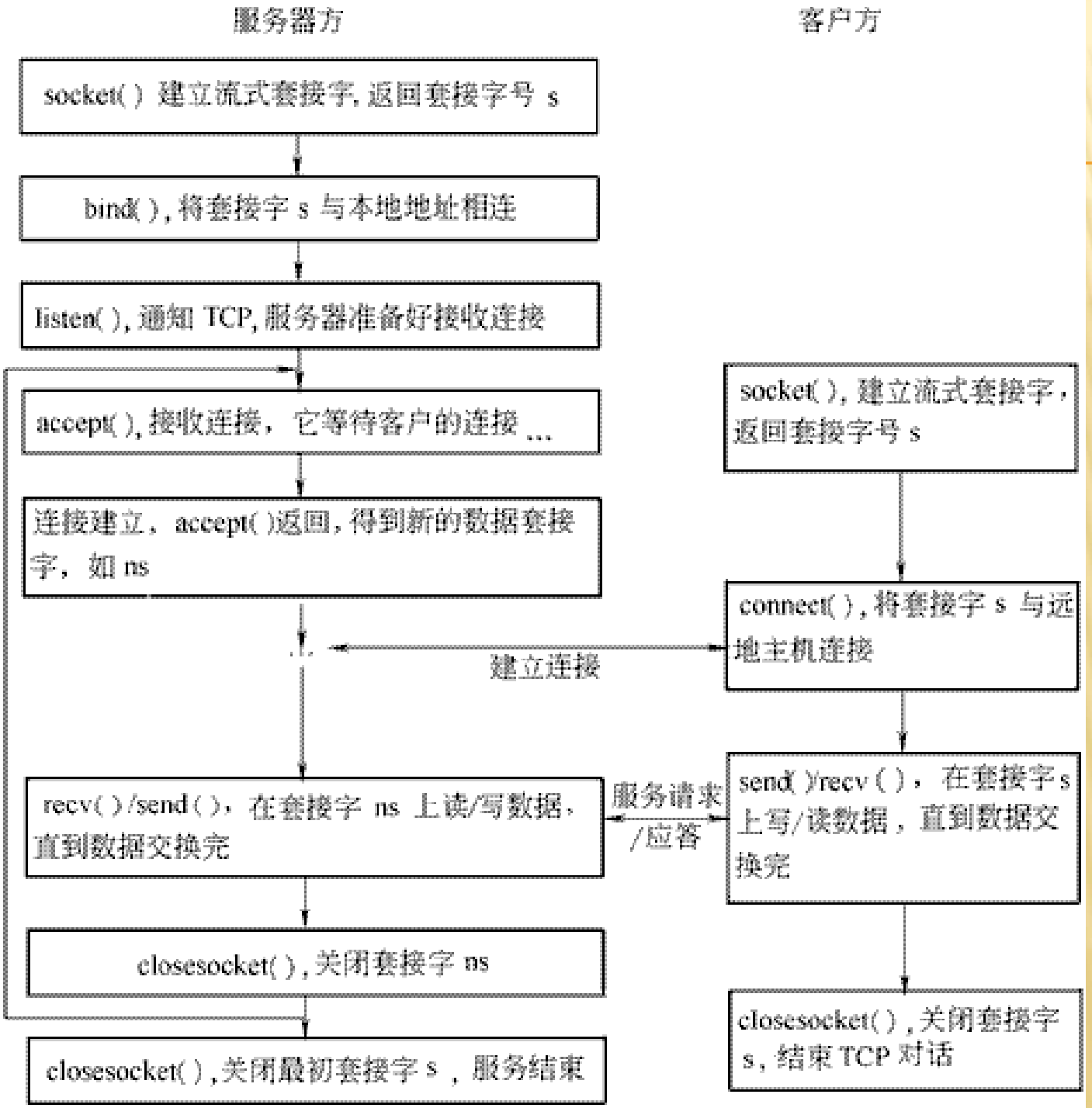
- 接下来，客户端会继续组建数据包，并设置 ACK 标志位，表示客户端正确接收了服务器发来的“确认包”。同时，将刚才服务器发来的数据包序号（2000）加1，得到 2001，并用这个数字来填充“确认号（Ack）”字段。

- 客户端将数据包发出，进入ESTABLISED状态，表示连接已经成功建立。
- 4）服务器端收到数据包，检测到已经设置了ACK 标志位，就知道这是客户端发来的“确认包”。服务器会检测“确认号（Ack）”字段，看它的值是否为 2000+1，如果是就说明连接建立成功，服务器进入ESTABLISED状态。
- 至此，客户端和服务器都进入了ESTABLISED状态，连接建立成功，接下来就可以收发数据了。

实验流程


客户端
socket
connect
send
recv
close
服务器端
socket
bind
listen
accept
recv
send
close


服务器端
客户端
socket
bind
listen
accept
recv/recvfrom
send/sendto
close
socket
bind
connect
send/sendto
recv/recvfrom
close
下一张


服务器方
客户方
socket( ) 建立流式套接字, 返回套接字号 s
bind( ), 将套接字 s 与本地地址相连
listen( ), 通知 TCP, 服务器准备好接收连接
accept( ), 接收连接， 它等待客户的连接 ...
连接建立， accept( )返回, 得到新的数据套接字， 如 ns
建立连接
recv( )/send( )， 在套接字 ns 上读/写数据, 直到数据交换完
服务请求
/应答
closesocket( ), 关闭套接字 ns
closesocket( ), 关闭最初套接字 s ， 服务结束
socket( ), 建立流式套接字，返回套接字号 s
connect( ), 将套接字 s 与远地主机连接
send( )/recv( )， 在套接字 s 上写/读数据， 直到数据交换完
closesocket( ), 关闭套接字 s, 结束 TCP 对话

实验步骤

- *服务器：*
- *首先使用socket函数创建一个套接口文件描述符，文件描述符是linux中通用的文件句柄，用于描述一个文件（unix把大部分IO资源统称为文件），原型如下：*
- #include <sys/types.h>
- #include <sys/socket.h>
- int socket(int domain, int type, int protocol);

- 之后使用bind函数绑定该套接口用于监听的IP地址和端口，之后，客户端可以使用该IP地址和端口进行连接，已达到通信的目的，原型如下：
- #include <sys/types.h>
- #include <sys/socket.h>
- int bind(int sockfd, const struct sockaddr *addr, socklen_t addrlen);

- bind完成之后，程序使用listen函数使socket开始监听来自外部的连接，使用listen之后，外部的程序就可以连接到该程序了：
- #include <sys/types.h>
- #include <sys/socket.h>
- int listen(int sockfd, int backlog);

- 当有连接到达时，监听的socket变为可读状态，这是使用accept函数读取该socket，即可得到一个新的socket描述符，该描述符代表了新进来的连接，之后就可以对该描述符进行读写操作了，这时的读写操作其实是调用网络设备进行网络收发：
- #include <sys/types.h>
- #include <sys/socket.h>
- int accept(int sockfd, struct sockaddr *addr, socklen_t *addrlen);

- 之后便是调用read和write函数进行网络IO了：
- #include <unistd.h>
- ssize_t read(int fildes, void *buf, size_t nbyte);
- ssize_t write(int fildes, const void *buf, size_t nbyte);

- 在使用完socket时候，我们需要断开连接，这时可以调用close或者shutdown函数：
- #include <unistd.h>
- int close(int fd);
- #include <sys/socket.h>
- int shutdown(int socket, int how);

- **客户端：**
- 首先使用socket函数创建一个套接口文件描述符，文件描述符是linux中通用的文件句柄，用于描述一个文件（unix把大部分IO资源统称为文件），原型如下：
- #include <sys/types.h>
- #include <sys/socket.h>
- int socket(int domain, int type, int protocol);

- 客户端只需要使用临时的端口进行通信即可，因为它没有必要让用户知道该临时端口是多少，所以没有必要使用bind函数，在得到socket描述符之后，可以直接使用connect函数连接服务器，connect函数发现之前的套接口没有bind端口时会自动选择一个临时端口号：
- #include <sys/types.h>
- #include <sys/socket.h>
- int connect(int sockfd, const struct sockaddr *serv_addr, socklen_t addrlen);

- 连接成功之后，就可以调用上面说过的read和write函数进行和服务器的通信了。并在通信完成之后关闭socket。

设计思路

- socket表现为文件描述符
  - 将int socket作为类的成员，类向外暴露各种socket操作

```
class Socket
{
  private:
      int socket_;
  public:
      int create();     // 创建一个socket
      int connect();  // 连接
      int listen();     // 侦听
      ...
};
```

- 但是socket的类型很多

	AF_INET	AF_INET6	AF_LOCAL	AF_ROUTE
SOCK_STREAM	TCP|SCTP	TCP|SCTP	yes	
SOCK_DGRAM	UDP	UDP	yes	
SOCK_SEQPACKET	SCTP	SCTP	yes	
SOCK_RAW	IPv4	IPv6		yes

  - 问题1：如何以一种统一的形式封装各种不同类型的socket?

- 考察socket的连接函数

```
#include <sys/socket.h>

int connect(int sockfd, const struct sockaddr *servaddr, socklen_t addrlen);
```

  - socket的地址族不一样，导致sockaddr结构是不一样的
  - 虽然tcp/sctp这样的面向连接的协议可能都会有connect这样的函数，并且连接的动作都差不多，但是connect函数的servaddr参数是不一样的
  - 问题2：如何解决相似动作，但是参数类型不同?

- 采用虚函数?
- 采用模板?

```
/**
 * @class Socket
 * @brief
 * wrap of socket
 *
 * @params
 * A - Address4 or AddressUn
 * T - SOCK_STREAM, SOCK_DGRAM
 * P - PROTOCOL type
 */
template <typename A, int T, int P>
class Socket : public BasicFile
{
    ...
```

```
#include <sys/socket.h>

int socket (int family, int type, int protocol);
```

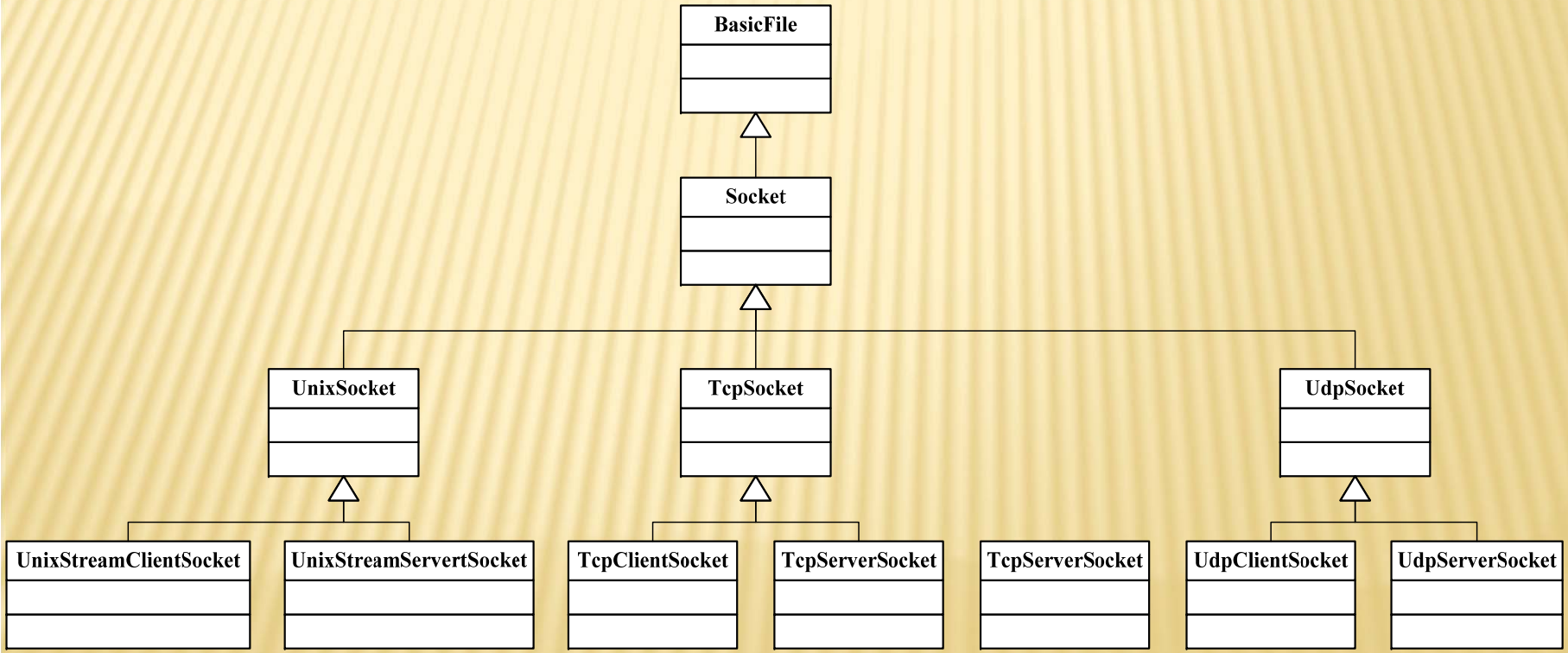
虚函数

- 在面向对象程序设计领域，C++、Object Pascal 等语言中有虚函数（英语：virtual function）或虚方法（英语：virtual method）的概念。这种函数或方法可以被子类继承和覆盖，通常使用动态调度实现。这一概念是面向对象程序设计中（运行时）多态的重要组成部分。简言之，虚函数可以给出目标函数的定义，但该目标的具体指向在编译期可能无法确定。

STL编程

- STL（Standard Template Library，即标准模版库）是一个具有工业级强度的，高效的C++程序库。它被容纳于C++标准程序库（C++ Standard Library）中，是ANSI/ISO C++标准中最新的也是极具革命性的一部分。该库包含了诸多在计算机科学领域所常用的基本数据结构和基本算法。为广大C++程序员提供了一个可扩展的应用框架，高度体现了软件的可复用性。这种现象有些类似于Microsoft Visual C++中的MFC（Microsoft Foundation Class Library），或者是Borland C++ Builder中的VCL（Virsual Component Library）。

2. 类继承体系


BasicFile
Socket
UnixSocket
TcpSocket
UdpSocket
UnixStreamClientSocket
UnixStreamServertSocket
TcpClientSocket
TcpServerSocket
TcpServerSocket
UdpClientSocket
UdpServerSocket

The End