实验3-TCP/UDP通信程序设计

1 实验目的

1. 深入了解 TCP 和 UDP 的区别与联系。
2. 掌握网络 Socket 编程的基本概念和基本编程方法。
3. 掌握 TCP 和 UDP 的 Client/Server 结构的程序的基本编程方法。

2 预备知识

1. 了解 Windows、Linux/Unix 等不同操作系统下的 TCP、UDP 编程环境。
2. 学习 TCP、UDP 编程接口 socket 函数调用。

3 实验原理

Socket 编程基本原理参见参考文档。

传输层和应用层之间进行交换的数据称为报文(Message)，而在传输层和网络层之间进行交换的数据被称为数据报(Datagram)。传输层可以使用传输控制协议(TCP)来封装数据，也可以使用用户数据报协议(UDP)来封装数据。TCP 协议面向连接，使用字节流传送服务，是可靠的；而 UDP 协议面向非连接，使用数据报服务，是非可靠的。TCP 协议提供高可靠性的传输，UDP 协议提供高效的传输。在实际应用中，它们有其各自所适应的场合。

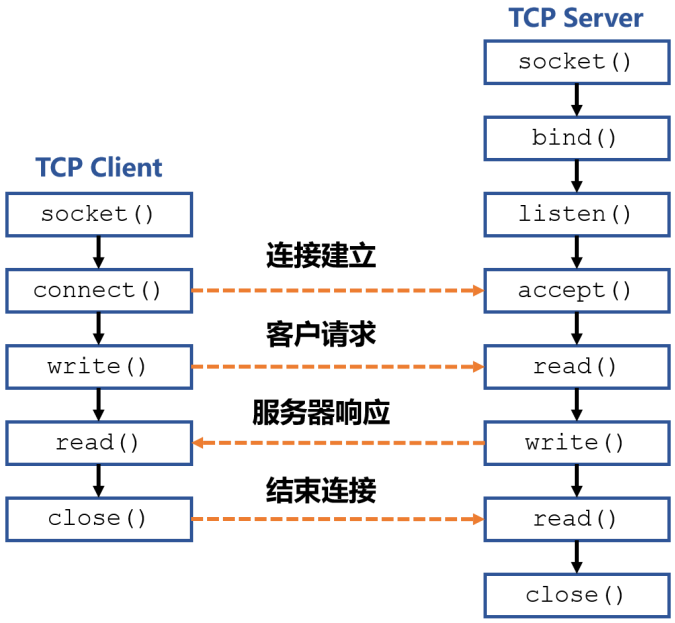
在面向连接的 Client/Server 结构中：服务器首先启动，通过调用 socket() 建立一个套接口，然后调用 bind() 将该套接口和本地网络地址联系在一起，再调用 listen() 使套接口做好侦听的准备，并规定它的请求队列的长度，之后就调用 accept() 来接收连接。客户在建立套接口后就可调用 connect() 和服务器建立连接。连接一旦建立，客户机和服务器之间就可以通过调用 read() 和 write() 来发送和接收数据。最后，待数据传送结束后，双方调用 close() 关闭套接口。

在无连接的 Client/Server 结构中，服务器使用 socket() 和 bind() 函数调用建立和连接 socket。由于此时的 socket 是无连接的，服务器使用 recvfrom() 函数从 socket 接收数据。客户端也只调用 bind() 函数而不调用 connect() 函数。注意：无连接的协议不在两个端口之间建立点到点的连接，因此 sendto() 函数要求程序在一个参数中指明目的地址。recvfrom() 函数不需要建立连接，它对到达相连协议端口的任

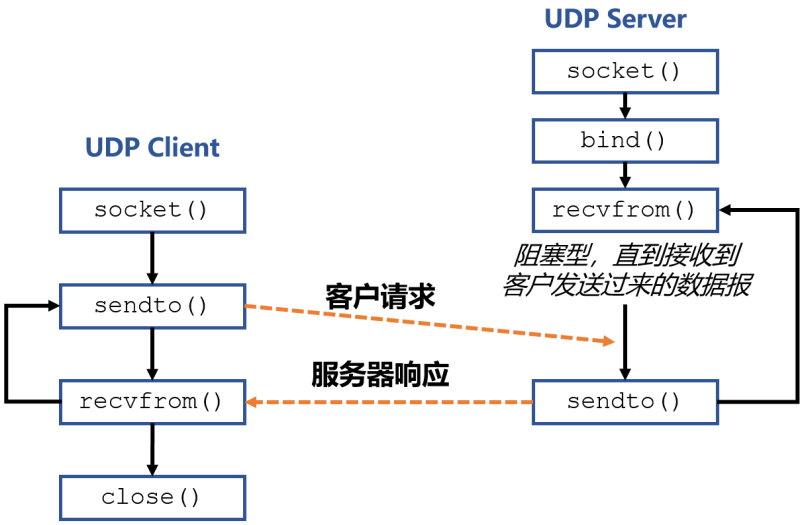
何数据作出响应。当 recvfrom() 函数从 socket 收到一个数据报时，它将保存发送此数据包的进程的网络地址以及数据包本身。程序(服务器和客户)用保存的地址去确定发送(客户)进程。在必要的条件下，服务器将其应答数据报送到从 recvfrom()函数调用中所得到的网络地址中去。

一般来说，大多数 TCP 服务器是并发的，而大多数 UDP 服务器是迭代的。多数 TCP 服务器是与调用 fork 来处理每个客户连接的服务器并发执行的。迭代服务器没有对 fork 的调用，所以单一服务器进程就处理了所有客户。

基本的 TCP 通信流程如下：



基本的 UDP 通信流程如下：



4 实验内容

注意：有时候第一次运行程序没问题，再运行会出现发出端口已占用的错误。这是因为操作系统会给占用的端口号设置一小段冷却时间。等冷却时间结束或者将端口号从常量改成程序输入都行。

4.1 TCP 通信程序设计

选择一个操作系统环境(Linux/Windows，建议 linux)，编制 TCP/IP 通信程序，完成一定的通信功能。例如：两机聊天或单机两终端模拟聊天，使用TCP/IP 协议进行相互直接的发送和接收等。

4.1 UDP 通信程序设计

选择一个操作系统环境(Linux/Windows，建议 linux)，编制 UDP/IP 通信程序，完成一定的通信功能。在发送 UDP 数据包时做一个循环，连续发送100 个数据包；在接收端统计丢失的数据包。

5 实验报告

1. 说明在实验过程中遇到的问题和解决方法。
2. 给出程序详细的流程图和对程序关键函数的详细说明。
3. 使用 socket API 开发通信程序中的客户端程序和服务器程序时，各需要哪些不同的函数。
4. 解释 connect()、bind() 等函数中 struct sockaddr *addr 参数各个部分的含义，并用具体的数据举例说明。
5. 说明面向连接的客户端和非连接的客户端在建立 socket 时有什么区别。
6. 说明面向连接的客户端和面向非连接的客户端在收发数据时有什么区别。在非连接的客户端又是如何判断数据发送结束的。
7. 比较面向连接的通信和无连接通信，它们各有什么优点和缺点?适合在哪种场合下使用?
8. 实验过程中使用 socket 的时候是工作在阻塞方式还是非阻塞方式，通过网络检索阐述这两种操作方式的不同。

附录A 辅助函数介绍

本节内容翻译自 die.net 。

inet_addr

```
#include <arpa/inet.h>
in_addr_t inet_addr(const char *cp);
```

inet_addr() 函数将 Internet 主机地址 `cp` 从 IPv4 的点分八进制表示法转换为网络字节顺序的二进制数据。 如果输入无效，则返回 **INADDR_NONE** (通常为 -1)。 使用此函数存在隐患，因为 -1 是有效地址 (255.255.255.255)。

htons

```
#include <arpa/inet.h>
uint32_t htonl(uint32_t hostlong);
uint16_t htons(uint16_t hostshort);
uint32_t ntohl(uint32_t netlong);
uint16_t ntohs(uint16_t netshort);
```

htonl() 函数将无符号整数 `hostlong` 从主机字节以网络字节顺序。
htons() 函数将无符号短整数 `hostshort` 从主机字节以网络字节顺序。
ntohl() 函数将无符号整数 `netlong` 从网络字节顺序到主机字节顺序。
ntohs()函数将无符号短整数 `netshort` 从网络字节顺序到主机字节顺序。

perror

```
#include <stdio.h>
void perror(const char *s);
```

perror() 函数会在标准错误输出上生成一条消息，描述在调用系统或库函数期间遇到的最后一个错误。首先打印参数字符串 `s` (如果 `s` 不为 `NULL` 且 `*s` 不是空字节 `\0`)，然后打印冒号和空格。 然后错误消息和换行符。

错误号取自外部变量 `errno` ，该值在发生错误时设置，但在成功调用后不会清除。

sprintf

```
#include <stdio.h>
int printf(const char *format, ...);
int fprintf(FILE *stream, const char *format, ...);
int sprintf(char *str, const char *format, ...);
```

printf() 系列中的函数根据 `format` 产生格式化的输出，并将输出写入到不同的地方。函数 **printf** 写入到标准输出 `stdout` ； **fprintf** 写入到给定的输出流`stream` ； **sprintf** 写入到字符串 `str` 。

memset

```
#include <string.h>
void *memset(void *s, int c, size_t n);
```

memset() 函数会将 `s` 所指的内存空间的前 `n` 个比特，用常数 `c` 填充。