Lab 8：网络LED矩阵显示器

控制 MAX7219

前一次实验已经为 MAX7219 写好了设备驱动，只需要向 `/dev/MAX7219` 中写入字符即可在 LED 上显示。

这一次要在程序中写入这个设备文件，需要注意的是，C 语言的文件函数是具有缓冲区的，因此写入文件的内容无法实时地写入，因此在每次写入后要强制清空缓冲区：

```
int fflush( FILE *stream );
```

这个函数的作用是，将 stream 流的缓冲区中的所有内容写回到与之相关联的外部设备。

下面我们编写一段程序测试用 C 语言将字符串输出到 LED 显示：

```
int main(int argc, const char * argv[]) {
    FILE *MAX7219;
    if ((MAX7219 = fopen("/dev/MAX7219", "w")) == NULL) {
        cout << "MAX7219 open error!" << endl;
        return 0;
    }
    fprintf(MAX7219, "1234\n");
    fflush(MAX7219);
}
```

Socket 编程

Socket 接口是 TCP/IP 网络的 API，该接口定义了许多函数或例程，可以用它们来开发 TCP/IP 网络上的应用程序。网络的 Socket 数据传输是一种特殊的 I/O，Socket 也是一种文件描述符。

Socket 函数库

socket() 函数

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

socket() 用于创建一个 socket 描述符，它唯一标识一个 socket。

bind() 函数

```
int bind(int sockfd, const struct sockaddr *addr, socklen_t addrlen);
```

当我们调用 socket 创建一个 socket 时，返回的 socket 描述字它存在于协议族空间中，但没有一个具体的地址。如果想要给它赋值一个地址，就必须调用 bind() 函数，否则就当调用connect()、listen() 时系统会自动随机分配一个端口。

bind() 函数把一个地址族中的特定地址赋给 socket。例如对应 AF_INET、AF_INET6 就是把一个 ipv4 或 ipv6 地址和端口号组合赋给 socket。

函数的三个参数分别为：

sockfd：即socket描述字，它唯一标识一个socket。bind()函数就是将给这个描述字绑定一个名字。

addr: 一个const struct sockaddr *指针，指向要绑定给sockfd的协议地址。这个地址结构根据地址创建socket时的地址协议族的不同而不同，如ipv4对应的是：

```
struct sockaddr_in {
    sa_family_t    sin_family; /* address family: AF_INET */
    in_port_t      sin_port;   /* port in network byte order */
    struct in_addr sin_addr;   /* internet address */
}

/* Internet address. */
struct in_addr {
    uint32_t       s_addr;     /* address in network byte order */
}
```

addrlen：对应的是地址的长度。

通常服务器在启动的时候都会绑定一个地址，用于提供服务，客户就可以通过它来接连服务器；而客户端就不用指定，有系统自动分配一个端口号和自身的 ip 地址组合。通常服务器端在 listen 之前会调用 bind()，而客户端就不会调用，而是在 connect() 时由系统随机生成一个。

listen()/connect()函数

```
int listen(int sockfd, int backlog);
int connect(int sockfd, const struct sockaddr *addr, socklen_t addrlen);
```

如果作为一个服务器，在调用 socket()、bind() 之后就会调用 listen() 来监听这个 socket，如果客户端这时调用 connect() 发出连接请求，服务器端就会接收到这个请求。

listen 函数的第一个参数即为要监听的 socket 描述字，第二个参数为相应 socket 可以排队的最大连接个数。socket() 函数创建的 socket 默认是一个主动类型的，listen函数将 socket 变为被动类型的，等待客户的连接请求。

connect 函数的第一个参数即为客户端的 socket 描述字，第二参数为服务器的 socket 地址，第三个参数为 socket 地址的长度。客户端通过调用 connect 函数来建立与 TCP 服务器的连接。

accept()函数

```
int accept(int sockfd, struct sockaddr *addr, socklen_t *addrlen);
```

TCP服务器端依次调用 socket()、bind()、listen() 之后，就会监听指定的 socket 地址了。TCP 客户端依次调用 socket()、connect() 之后就向 TCP 服务器发送了一个连接请求。TCP 服务器监听到这个请求之后，就会调用 accept() 函数取接收请求，这样连接就建立好了。之后就可以开始网络 I/O 操作了，即类同于普通文件的读写I/O操作。

如果 accept 成功返回，则服务器与客户已经正确建立连接了，此时服务器通过 accept 返回的套接字来完成与客户的通信。

read()/write()等函数

```
#include <unistd.h>
ssize_t read(int fd, void *buf, size_t count);
ssize_t write(int fd, const void *buf, size_t count);

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

ssize_t send(int sockfd, const void *buf, size_t len, int flags);
ssize_t recv(int sockfd, void *buf, size_t len, int flags);

ssize_t sendto(int sockfd, const void *buf, size_t len, int flags,
        const struct sockaddr *dest_addr, socklen_t addrlen);
ssize_t recvfrom(int sockfd, void *buf, size_t len, int flags,
        struct sockaddr *src_addr, socklen_t *addrlen);

ssize_t sendmsg(int sockfd, const struct msghdr *msg, int flags);
ssize_t recvmsg(int sockfd, struct msghdr *msg, int flags);
```

close()函数

```
int close(int fd);
```

在服务器与客户端建立连接之后，会进行一些读写操作，完成了读写操作就要关闭相应的 socket 描述字，好比操作完打开的文件要调用 fclose 关闭打开的文件。

Socket 编程流程

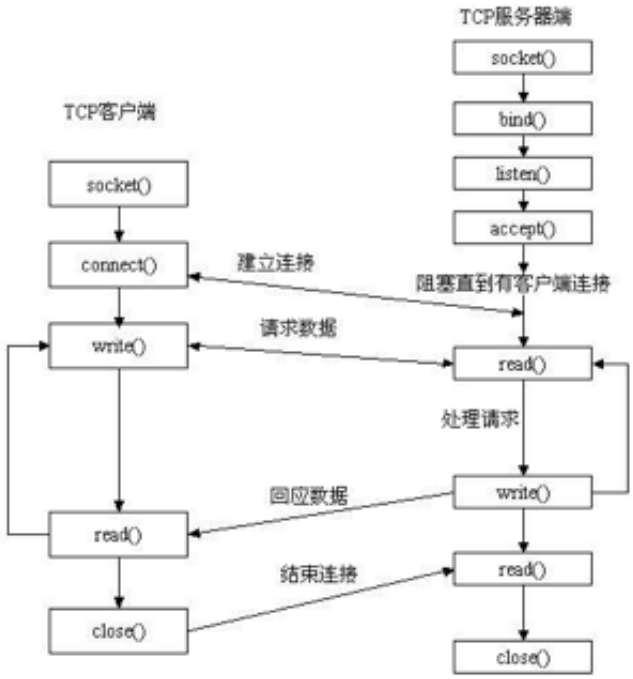
服务器端编程的步骤：

1. 创建套接字：socket()
2. 绑定套接字到一个IP地址和一个端口上：bind()
3. 将套接字设置为监听模式等待连接请求：listen()
4. 请求到来后，接受连接请求，返回一个新的对应于此次连接的套接字：accept()
5. 用返回的套接字和客户端进行通信：send()/recv()
6. 返回，等待另一连接请求；
7. 关闭套接字：closesocket()

客户端编程的步骤：

1. 创建套接字：socket()
2. 向服务器发出连接请求：connect()
3. 和服务器端进行通信：send()/recv()
4. 关闭套接字：closesocket()

流程图：



网络控制 MAX7219

服务器编程

利用 Socket 编程原理，在树莓派上创建一个 socket，监听指定端口，接受到请求后开启一个线程处理连接，并将客户端发送的字符串写入 MAX7219 的设备驱动文件。

```
//
//  main.cpp
//  MAX7219Server
//
//  Created by Yym on 16/6/9.
//  Copyright © 2016年 me.helloyym. All rights reserved.
//

#include <iostream>
#include <string>
#include <pthread.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include <unistd.h>
#include <arpa/inet.h>

using std::cout;
using std::endl;

#define SERVER_PORT 6666 //侦听端口
#define BUFFER_SIZE 1000
#define SOCKET_ERROR -1
```

```
FILE *MAX7219;

void* ClientThread(void *lpParameter) {
    int sServer = *(int*) lpParameter;
    char buffer[BUFFER_SIZE + 1];

    while (1) {
        if (recv(sServer, buffer, BUFFER_SIZE, 0) == SOCKET_ERROR) {
            cout << "recv failed" << endl;
            break;
        }
        fprintf(MAX7219, "%s\n", buffer);
        fflush(MAX7219);
    }

    close(sServer);
    return 0;
}

int main(int argc, const char * argv[]) {

    //打开设备文件
    if ((MAX7219 = fopen("/dev/MAX7219", "w")) == NULL) {
        cout << "MAX7219 open error!" << endl;
        return 0;
    }

    int sListen, sServer; //侦听套接字，连接套接字
    struct sockaddr_in saServer, saClient; //地址信息

    //创建socket，使用TCP协议：
    sListen = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
    if (sListen == SOCKET_ERROR) {
        cout << "socket() failed!" << endl;
        return 0;
    }

    //构建本地地址信息：
    memset(&saServer, 0, sizeof(saServer));
    saServer.sin_family = AF_INET; //地址家族
    saServer.sin_port = htons(SERVER_PORT); //注意转化为网络字节序
    saServer.sin_addr.s_addr = htonl(INADDR_ANY); //使用INADDR_ANY指示任意地址

    //绑定：
    bind(sListen, (struct sockaddr *) &saServer, sizeof(saServer));

    //侦听连接请求：
    if (listen(sListen, 5) == SOCKET_ERROR) {
        cout << "listen() failed!" << endl;
```

```
        close(sListen); //关闭套接字
        return 0;
    }

    cout << "Waiting for client connecting!" << endl;
    cout << "tips : Ctrl+c to quit!" << endl;

    while (true) {
        //阻塞等待接受客户端连接:
        socklen_t length = sizeof(saClient);
        sServer = accept(sListen, (struct sockaddr *) &saClient, &length);
        if (sServer == SOCKET_ERROR) {
            cout << "accept() failed!" << endl;
            continue;
        }
        cout << "new client" << endl;
        cout << "Accepted client: " << inet_ntoa(saClient.sin_addr) << ":"
<< ntohs(saClient.sin_port) << endl;

        pthread_t hThread;
        pthread_create(&hThread, NULL, ClientThread, &sServer);
        if (hThread < 0) {
            printf("Create Thread Failed!\n");
            break;
        }

        char title[] = "welcome to yym's MAX7219 server.\r\n";
        send(sServer, title, strlen(title), 0);
    }
    close(sListen);
}
```

客户端连接

在 Mac 上利用 telnet 连接服务器的指定端口，连接成功后写入字符串，可以看到 LED 矩阵显示了发送的字符。

```
➜  ~ telnet 192.168.1.10 6666
Trying 192.168.1.10...
Connected to 192.168.1.10.
Escape character is '^]'.
welcome to yym's MAX7219 server.
```