# 实验七：UDP 网络应用程序设计实验

## 一、实验目的

1.学习基于mbed操作系统的网络应用程序的设计方法；

2.掌握UDP协议的通讯技术；

3.掌握调试程序的方法

## 二、实验仪器与设备

Mbed 开发板、连接线、计算机。

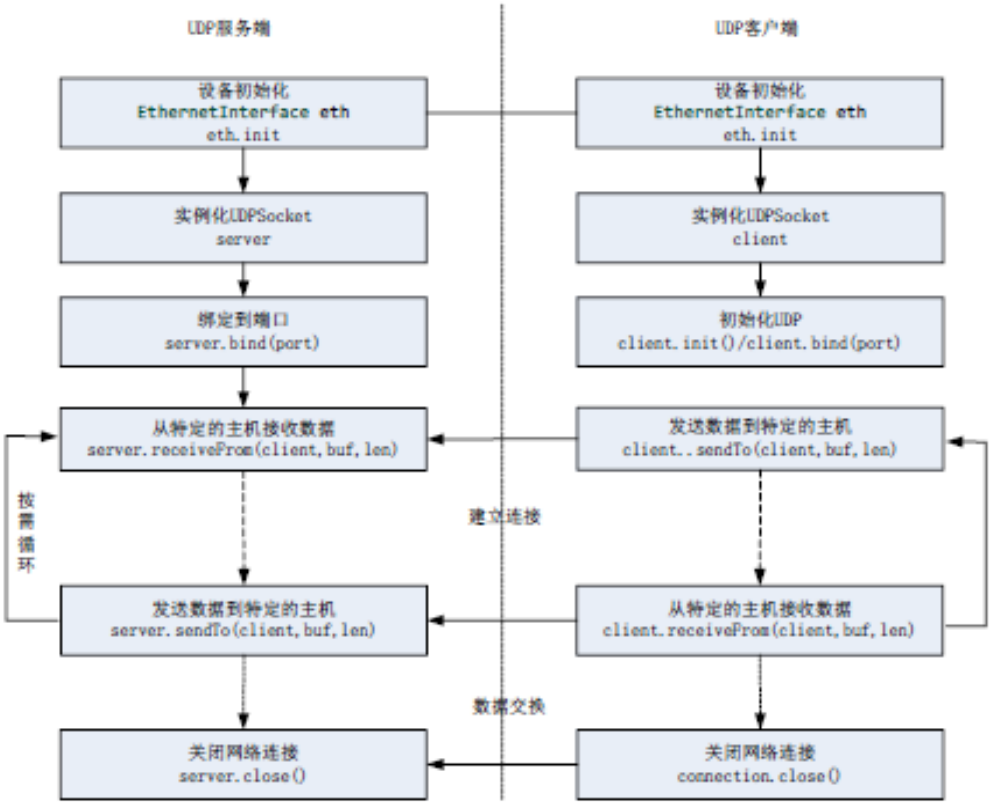
## 三、预备知识

### （1）mbed UDP 程序设计基础

UDP协议工作在TCP/IP层次模型的传输层，它是一个面向快速数据传输而设计传输层协议，它无需建立连接就可以直接发送数据，但并不保证数据能够被正确接收。mbed 提 供了 UDPSocket 类用于 UDP 数据传输，它提供的主要方法有：

| 类名 | 方法 | 用途 |
|---|---|---|
| UDPSocket | UDPSocket(); | 构造函数，实例化 UDPSocket 对象 |
| | int init(void); | 启动 UDP 客户端，它和具体的端口无关，返回 0 表示成功，-1 表示失败 |
| | int bind(int port); | 启动 UDP 服务端,并绑定到 port 端口，返回 0 表示成功，-1 表示失败 |
| | int join_multicast_group(const char* address); | 加入特定地址的组播分组，返回 0 表示成功，-1 表示失败 |
| | int set_broadcasting(bool broadcast=true); | 让 UDP 工作在广播模式 |
| | int sendTo(Endpoint &remote, char *packet, int length); | 发送数据给特定的地址，remote 为一个具体的传输层地址，packet 为要发送的数据缓冲区,length 为要发送的数据大小，返回-1 表示失败，其它则表示成功 |
| | int receiveFrom(Endpoint &remote, char *buffer, int length); | 从特定的地址接收数据，remote 为一个具体的传输层地址,buffet 为数据接收缓冲区,length 为数据接收缓冲区的大小，返回-1 表示失败，其 |
| | | 它值表示接收到数据的大小 |
| | void set_blocking(bool blocking, unsigned int | 设置当前连接的工作方式，可以是阻塞模式，也可以是超时模式 |
| | int close(bool shutdown=true); | 关闭当前连接 |

UDP应用的流程图如下：




从图中可以看出，对于UDP来说，服务端和客户端的处理几乎是一样的，唯一不同的是使用init方法还是bind方法，这可以通过后面的示例代码加以理解。

## 五、实验内容

### 1、 mbed UDP 程序初步设计

由于 UDP 在使用时无需连接就可以收发数据，所以在代码的编写上比较简单，但也带 来了问题，由于 UDP 在数据通讯的过程中并没有一个切实存在的连接，所以让服务端返回 数据给客户端就变得复杂了，我们先看下面的代码，这里用的是 init 方法：

```
#include "EthernetInterface.h"
#include
"rtos.h"
EthernetInterf
ace eth;
UDPSocket
socket;
const char* REMOTEIP = "192.168.1.11";
const int REMOTEPORT = 8088;
  Endpoint echo_server;
void net_task(void const *)
{
  eth.init("192.168.1.100","255.255.255.0","192.168.1.1");
  eth.connect();
  printf("IP Address is %s\n", eth.getIPAddress());
```

```
        printf("MAC Address is %s\n", eth.getMACAddress());
        printf("GateWay Address is %s\n", eth.getGateway());
        socket.set_blocking(false,100);
        socket.init();
    }
void send_task(void const *)
    {
        char out_buffer[] = "Hello World\n";
        while (1)
        {
            socket.sendTo(echo_server, out_buffer, sizeof(out_buffer));
            Thread::wait(1000);
        }
    }
    int main()
        {
        echo_server.set_address(REMOTEIP, REMOTEPORT);
        Thread netTask(net_task, NULL, osPriorityNormal, 1024 * 4);
        Thread sendTask(send_task, NULL, osPriorityNormal, 1024 * 4);
        while (1)
        {
            char in_buffer[256];
            int n = socket.receiveFrom(echo_server, in_buffer,
            sizeof(in_buffer)); if (n>0)
            {
                in_buffer[n] = '\0';
                    printf("%s\n",
                      in_buffer);
            }
            Thread::yield();
        }
        socket.close();
        eth.disconnect();
    }
```

（2）**mbed UDP** 广播程序设计

UDP 支持广播数据的发送和接收。计算机网络里面的广播指的是从一个终端向同一个网络即网络好相同的每个计算机发送消息。Mbed 在广播的实现上也非常简单，只要用广播地址代替确定的地址即可，下面是一个广播方式数据收发的示例：

```
#include "EthernetInterface.h"
#include "rtos.h"
EthernetInterface eth;
UDPSocket socket;
const char* REMOTEIP = "255.255.255.255";
const int REMOTEPORT = 8088;
Endpoint echo_server;
void net_task(void const *)
```

```
{
    eth.init("192.168.1.100","255.255.255.0","192.168.1.1");
    eth.connect();
    printf("IP Address is %s\n", eth.getIPAddress());
    printf("MAC Address is %s\n", eth.getMACAddress());
    printf("GateWay Address is %s\n", eth.getGateway());
    socket.set_blocking(false,100);
    socket.bind(1000);
    socket.set_broadcasting();
}
void send_task(void const *)
{
    char out_buffer[] = "Hello World\n";
    while (1)
    {
        socket.sendTo(echo_server, out_buffer, sizeof(out_buffer));
        Thread::wait(1000);
    }
}
int main() {
    echo_server.set_address(REMOTEIP, REMOTEPORT);
    Thread netTask(net_task, NULL, osPriorityNormal, 1024 * 4);
    Thread sendTask(send_task, NULL, osPriorityNormal, 1024 * 4);
    while (1)
    {
        char in_buffer[256];
        int n = socket.receiveFrom(echo_server, in_buffer, sizeof(in_buffer));
        if (n>0)
        {
            in_buffer[n] = '\0';
            printf("%s\n", in_buffer);
        }
        Thread::yield();
    }
    socket.close();
    eth.disconnect();
    }
```

## 五、实验要求

1、认真阅读预备知识和实验内容；

2、编写代码，编译、下载到 mbed 开发板，验证实验效果。

3、撰写实验报告。

# 实验八：HTTP 网络应用程序设计实验

## 一、实验目的

1.学习基于mbed操作系统的网络应用程序的设计方法；

2.掌握HTTP协议的通讯技术；

3.掌握调试程序的方法

## 二、实验仪器与设备

Mbed 开发板、连接线、计算机。

## 三、预备知识

### 1. HTTP 协议基础

超文本传输协议(HTTP，HyperText Transfer Protocol)是互联网上应用最为广泛的一种网络协议。所有的WWW文件都必须遵守这个标准。设计HTTP最初的目的是为了提供一种发布和接收HTML页面的方法。HTTP协议是一种面向无连接的应用层协议，客户端发送一次请求，服务器端接收请求，经过处理返回给客户端信息，然后客户端和服务器端的链接就断开了。服务器和客户端的交互仅限于请求/响应过程，结束之后便断开，在下一次请求服务器会认为新的客户端。HTTP协议的请求消息和响应消息有固定的格式，具体列表如下：

----------请求头格式----------

HTTP请求行

（请求）头

空行

可选的消息体

----------响应头格式----------

HTTP状态行

（应答）头

空行

可选的消息体

我们首先需要关心出现在HTTP请求行中的请求方法，HTTP规范共定义了8种可能的请求方法，其中常用的是：

- GET：检索URI中标识资源的一个简单请求；
- HEAD：与GET方法相同，服务器只返回状态行和头标，并不返回请求文档；
- POS：服务器接受被写入客户端输出流中的数据的请求；
- PUT：服务器保存请求数据作为指定URI新内容的请求；
- DELETE：服务器删除URI中命名的资源的请求；
- OPTIONS：关于服务器支持的请求方法信息的请求；
- TRACE：Web服务器反馈Http请求和其头标的请求。

我们最常用的也就是get和post方法，get方法的请求方式比较简单，所有请求的参数都显示追加在请求的url后面，而且请求长度有限制，post方式的请求参数都追加在请求体当中，消息长度没有限制而且以隐式的方式进行发送。另外我们还需要关心一下出现在HTTP状态行中的状态吗，它是表示HTTP服务器响应状态的3位数字代码，其中最常用的是：

- 200 –服务器成功返回网页；
- 404 –请求的网页不存在；

●503 –服务不可用。

对于其它的状态吗，我们分类如下：

●1xx（临时响应）：表示临时响应并需要请求者继续执行操作的状态代码；

●2xx （成功）：表示成功处理了请求的状态代码。

●3xx （重定向）：表示要完成请求，需要进一步操作。通常，这些状态代码用来重定向。

●4xx（请求错误）：这些状态代码表示请求可能出错，妨碍了服务器的处理。

●5xx（服务器错误）：这些状态代码表示服务器在尝试处理请求时发生内部错误。这些错误可能是服务器本身的错误，而不是请求出错。

## 2.Mbed 提供的方法

# 四、实验内容

## 1、HTTP 客户端程序设计

mbed官网推荐了一个HTTPCLIENT扩展库，HTTPCLIENT及其辅助类提供的主要方法有：

| 类名 | 方法 | 用途 |
|---|---|---|
| HTTPText | HTTPClient(); | 构造函数,实例化 httpclient 对象 |
| | HTTPResult get(const char* url, IHTTPDataIn* pDataIn, int timeout = HTTP_CLIENT_DEFAULT_TIMEOUT); | 使用 get 请求方法获取 http 资源，传入的参数分别是 url 地址，用于存储数据的 IHTTPDataIn 对象地址，网络超时返回时间，默认是 15 秒，返回 0 表示成功，小于 0 表示错误 |
| | HTTPResult get(const char* url, char* result, size_t maxResultLen, int timeout = HTTP_CLIENT_DEFAULT_TIMEOUT); | 使用 get 请求方法获取 http 资源，传入的参数分别是 url 地址，接收缓冲区地址，允许接收的最大字节数，网络超时返回时间，默认是 15 秒，返回 0 表示成功，小于 0 表示错误 |
| | HTTPResult post(const char* url, const IHTTPDataOut& dataOut, IHTTPDataIn* pDataIn, int timeout = HTTP_CLIENT_DEFAULT_TIMEOUT); | 使用 post 请求方法上报数据，传入的参数分别是 url 地址，用于存储上报数据的 IHTTPDataOut 对象地址，网络超时返回时间，默认是 15 秒，返回 0 表示成功，小于 0 表示错误 |
| | HTTPResult put(const char* url, const IHTTPDataOut& dataOut, IHTTPDataIn* pDataIn, int timeout = HTTP_CLIENT_DEFAULT_TIMEOUT); | 使用 put 请求方法上报数据，传入的参数分别是 url 地址，用于存储上报数据的 IHTTPDataOut 对象地址，网络超时返回时间，默认是 15 秒，返回 0 表示成功，小于 0 表示错误 |
| | HTTPResult del(const char* url, IHTTPDataIn* pDataIn, int timeout = HTTP_CLIENT_DEFAULT_TIMEOUT); | 使用 del 请求方法删除远程服务器数据，传入的参数分别是 url 地址，用于存储返回数据的 IHTTPDataIn 对象地址，网络超时返回时间，默认是 15 秒，返回 0 表示成功，小于 0 表示错误 |
| | | |
| | HTTPText(char* str); | 构造函数,创建一个 HTTPText 实例，str 为要发送的字符串 |
| | HTTPText(char* str, size_t size); | 构造函数,创建一个 HTTPText 实例，str 为用于接收字符串的缓冲区，size |

| | | |
|---|---|---|
| | | 为缓冲区大小 |
| | void readReset(); | 重置 HTTPText 对象，把缓冲区初始到起始为止，用于发送处理 |
| | int read(char* buf, size_t len, size_t* pReadLen); | 读取数据用于发送，buf 为数据来源,len 为数据大小,pReadLen 为发送缓冲区 |
| | int getDataType(char* type, size_t maxTypeLen); | 获取数据 MIME 媒体类型,type 为获取的媒体类型字符串，maxTypeLen 为字符串长度 |
| | bool getIsChunked(); | 检测接收到的数据是否为 Chunked 编码方式，即无法确定数据的大小，用于 HTTP 数据接收 |
| | size_t getDataLen(); | 非 Chunked 编码下返回数据大小 |
| | void writeReset(); | 重置 HTTPText 对象，把缓冲区初始到起始为止，用于接收处理 |
| | int write(const char* buf, size_t len); | 从 HTTP 接收流中写入数据到 buf 中，len 为 buf 大小 |
| | void setDataType(const char*type); | 设置 MIME 媒体类型 |
| | void setIsChunked(bool chunked); | 设定是否要设为 Chunked 编码方式，用于 HTTP 数据发送 |
| | void setDataLen(size_t len); | 非 Chunked 编码下设定数据大小 |
| HTTPMap | HTTPMap(); | 构造函数,实例化 HTTPMap 对象，用于发送 application/x-www-form-urlencoded encoding 即键值类型的数据 |
| | void put(const char* key, const char* value); | 设置键值 |
| | void clear(); | 清空键值表 |

## 2、实验参考代码

```
#include "EthernetInterface.h"
#include "HTTPClient.h"
EthernetInterface eth;
HTTPClient http;
char str[512];
int main()
{
    eth.init(); //Use DHCP
    eth.connect();
    int ret;
    //POST data
    HTTPMap map;
    HTTPText inText(str, 512);
    map.put("temperature", "27.4");
    map.put("humidity", "78");
    printf("\nTrying to post data...\n");
    ret = http.post("http://192.168.1.139/post", map, &inText);
    if (!ret)
        {
        printf("Executed POST successfully - read %d characters\n", strlen(str));
        printf("Result: %s\n", str);
        }
    else
```

```
        {
        printf("Error - ret = %d - HTTP return code = %d\n", ret,
        http.getHTTPResponseCode());
        }
    //PUT data
    strcpy(str, "This is a PUT test!");
    HTTPText outText(str);
    printf("\nTrying to put resource...\n");
    ret = http.put("http://192.168.1.100/put", outText, &inText);
    if (!ret)
        {
        printf("Executed PUT successfully - read %d characters\n", strlen(str));
        printf("Result: %s\n", str);
        }
    else
        {
        printf("Error - ret = %d - HTTP return code = %d\n", ret,
        http.getHTTPResponseCode());
        }
    //DELETE data
    //HTTPText inText(str, 512);
    printf("\nTrying to delete resource...\n");
    ret = http.del("http://192.168.1.100/delete", &inText);
    if (!ret)
        {
        printf("Executed DELETE successfully - read %d characters\n", strlen(str));
        printf("Result: %s\n", str);
        }
    else
        {
        printf("Error - ret = %d - HTTP return code = %d\n", ret,
        http.getHTTPResponseCode());
        }
    eth.disconnect();
    while(1) {

    }
}
```

## 五、实验要求

1、认真阅读预备知识和实验内容；

2、编写代码，编译、下载到 mbed 开发板，验证实验效果。

3、撰写实验报告。