前面概念中简单提到，UDP是无连接的不可靠的数据报协议，非常不同于TCP面向连接的可靠字节流。然而，相比TCP有些场合更适合使用UDP，例如常见的应用程序有：DNS(域名系统)、NFS(网络文件系统)和SNMP(简单网络文件管理系统)。下面根据应用场景，给出不同的UDP服务端代码、客户端代码：

1. 服务端、客户端都接受、发送一次数据
2. 服务端可不断接收客户端的，并能区分多客户端
3. 服务端echo服务，客户端发送并回显
4. **简单的一次性UDP通信**

当前的例子，仅给出最简单最少的代码实现一个服务端、客户端收、发消息的流程。

（1）服务端

在UDP通信中，不需要知道客户端的地址结构信息时，可以选用简单的read、recv，也可以使用参数4,5为NULL的recvfrom。但是，需要知道客户端信息时，可以使用recvfrom获取，具体细节看下一个例子。

/\* udp\_server \*/

#include <stdio.h>

#include <sys/socket.h>

#include <arpa/inet.h> // sockaddr\_in, inet\_addr

#include <unistd.h> // close

constexpr const char \*SRV\_ADDR = "127.0.0.1";

constexpr int SRV\_PORT = 8080;

int main()

{

/// 1、创建socket

int socket\_fd = ::socket(AF\_INET, SOCK\_DGRAM, 0); // udp

/// 2、绑定到本地端口（服务端必须）

sockaddr\_in servaddr;

servaddr.sin\_family = AF\_INET; // Ipv4

servaddr.sin\_addr.s\_addr = inet\_addr(SRV\_ADDR); // 仅支持IPv4

servaddr.sin\_port = htons(SRV\_PORT); // 网路序port

::bind(socket\_fd, (const sockaddr \*)&servaddr, sizeof(servaddr));

/// 3、等待接收并打印

char buf[1024];

//int len = ::read(socket\_fd, buf, sizeof(buf));

//int len = ::recv(socket\_fd, buf, sizeof(buf), 0); //同上

int len = ::recvfrom(socket\_fd, buf, sizeof(buf), 0, NULL, NULL); // 同上

printf("%s: recv %2d: %s\n", \_\_func\_\_, len, buf);

/// 4、关闭

::close(socket\_fd);

}

（2）客户端

这里的客户端不使用bind函数（也是可以是，但通常不这么做），直接使用sendto函数发送数据到指定的服务端地址。**当然，也是可以使用connect，成功后可以直接使用write、send以及不带目标地址参数的sendto，具体见后面分析（单独一章说明客户端使用bind、connet函数）链接**。

/\* udp\_client \*/

#include <stdio.h>

#include <sys/socket.h>

#include <arpa/inet.h> // sockaddr\_in, inet\_addr

#include <unistd.h> // close

#include <string.h> // sizeof

constexpr const char \*SRV\_ADDR = "127.0.0.1";

constexpr int SRV\_PORT = 8080;

int main()

{

/// 1、创建socket

int socket\_fd = ::socket(AF\_INET, SOCK\_DGRAM, 0); // udp

// 2、 发送到服务端

sockaddr\_in servaddr;

servaddr.sin\_family = AF\_INET;

servaddr.sin\_addr.s\_addr = inet\_addr(SRV\_ADDR);

servaddr.sin\_port = htons(SRV\_PORT);

char buf[] = "hello sockte!";

::sendto(socket\_fd, buf, strlen(buf), 0, (sockaddr\*)&servaddr, sizeof(servaddr));

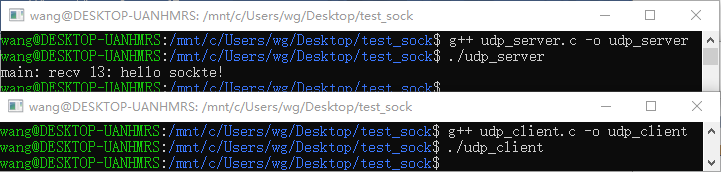
// 3、关闭退出

::close(socket\_fd);

}

(3) 测试

使用g++编译后，运行即可，结果如下



1. **UDP服务端**

在上一节基础上，修改UDP服务端可以不断接收客户端的消息，并且能够区分不同客户端，，另外添加出错的处理。客户端不做修改。

/\* udp\_server \*/

#include <stdio.h>

#include <sys/socket.h>

#include <arpa/inet.h> // sockaddr\_in, inet\_addr

#include <unistd.h> // close

#include <cstring>

#include <errno.h>

constexpr const char \*SRV\_ADDR = "127.0.0.1";

constexpr int SRV\_PORT = 8080;

//void udp\_server()

int main()

{

/// 1、创建socket

int socket\_fd = ::socket(AF\_INET, SOCK\_DGRAM, 0); // udp

if (socket\_fd == -1){

printf("%s: create socket failed. %s\n", \_\_func\_\_, strerror(errno));

return 1;

}

else{

printf("%s: create socket (fd = %d) success.\n", \_\_func\_\_, socket\_fd);

}

/// 2、绑定到本地端口

sockaddr\_in servaddr;

//memset(&servaddr,0, sizeof(servaddr));

//bzero(&servaddr, sizeof(servaddr)); // 可以根据实际情况选用

servaddr.sin\_family = AF\_INET; // IPv4

//servaddr.sin\_addr.s\_addr = inet\_addr(SRV\_ADDR); //仅支持IPv4

inet\_pton(servaddr.sin\_family, SRV\_ADDR, &servaddr.sin\_addr); // 新函数，通用IPv4/6

servaddr.sin\_port = htons(SRV\_PORT);

int ret = ::bind(socket\_fd, (const sockaddr \*)&servaddr, sizeof(servaddr));

if (ret == -1){

printf("%s: bind %s:%d failed. %s \n", \_\_func\_\_, SRV\_ADDR, SRV\_PORT,strerror(errno));

return 1;

}

else{

printf("%s: bind %s:%d success.\n", \_\_func\_\_, SRV\_ADDR, SRV\_PORT);

}

/// 3、等待接收和响应

char buf[1024];

// 用于获取客户端信息

sockaddr\_in clientaddr;

socklen\_t socklen;

while (1)

{

// int len = ::read(socket\_fd, buf, sizeof(buf));

// int len = ::recv(socket\_fd, buf, sizeof(buf), 0);

// 不关心数据发送者的协议地址,后面两个参数指定为空指针， 效果同上

// int len = ::recvfrom(socket\_fd, buf, sizeof(buf), 0, NULL, NULL);

int len = ::recvfrom(socket\_fd, buf, sizeof(buf), 0, (struct sockaddr \*)&clientaddr, &socklen);

if (len < 0){

printf("%s: recv failed. err %s \n", \_\_func\_\_, strerror(errno));

break;

}

else{

// 获取客户端的ip、和port

//char \*ip = inet\_ntoa(clientaddr.sin\_addr); // 仅支持IPv4，不可重入

char ip[INET6\_ADDRSTRLEN];

inet\_ntop(clientaddr.sin\_family, &clientaddr.sin\_addr, ip, socklen);

int port = ntohs(clientaddr.sin\_port);

printf("%s: client [%s:%d] recv %2d: %s\n", \_\_func\_\_, ip, port, len, buf);

}

}

/// 4、关闭连接

::close(socket\_fd);

}

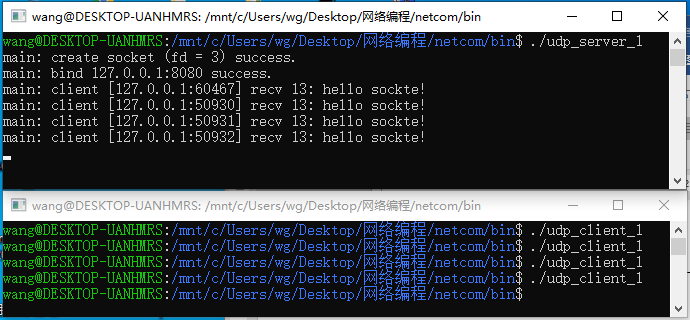
代码::recvfrom(socket\_fd, buf, sizeof(buf), 0, (struct sockaddr \*)&clientaddr, &socklen);不同与之前仅接受数据，还能获取到客户端的地址信息。

根据socklen的值，确定clientaddr对象应该是sockaddr\_in还是sockaddr\_in6，这里明确是IPv4，所以就直接从地址对象中通过调用地址转换相关函数inet\_ntop、ntohs获取ip和port。

**参看 recvfrom 链接。。。**

**参看 地址转换 链接。。。**

下图为演示结果，服务端启动后，客户端启动多次。



1. **UDP echo服务**

在上一节的基础上，服务端接收来自个客户端的消息，并将接收到的消息发给客户端。客户端仅处理一次对客户端发送、接收消息。

（1）服务端

接收之后，增加一段发送数据的代码

#include <stdio.h>

#include <sys/socket.h>

#include <arpa/inet.h> // sockaddr\_in, inet\_addr

#include <unistd.h> // close

#include <cstring>

#include <errno.h>

constexpr const char \*SRV\_ADDR = "127.0.0.1";

constexpr int SRV\_PORT = 8080;

int main()

{

/// 1、创建socket

int socket\_fd = ::socket(AF\_INET, SOCK\_DGRAM, 0); // udp

if (socket\_fd == -1){

printf("%s: create socket failed. %s\n", \_\_func\_\_, strerror(errno));

return 1;

}

else{

printf("%s: create socket (fd = %d) success.\n", \_\_func\_\_, socket\_fd);

}

/// 2、绑定到本地端口

sockaddr\_in servaddr;

servaddr.sin\_family = AF\_INET; // IPv4

//servaddr.sin\_addr.s\_addr = inet\_addr(SRV\_ADDR); //仅支持IPv4

inet\_pton(servaddr.sin\_family, SRV\_ADDR, &servaddr.sin\_addr); // 新函数，通用IPv4/6

servaddr.sin\_port = htons(SRV\_PORT);

int ret = ::bind(socket\_fd, (const sockaddr \*)&servaddr, sizeof(servaddr));

if (ret == -1){

printf("%s: bind %s:%d failed. %s \n", \_\_func\_\_, SRV\_ADDR, SRV\_PORT,strerror(errno));

return 1;

}

else{

printf("%s: bind %s:%d success.\n", \_\_func\_\_, SRV\_ADDR, SRV\_PORT);

}

/// 3、等待接收和响应

char buf[1024]; // 发送

sockaddr\_in clientaddr;

socklen\_t socklen;

while (1)

{

// 接收客户端数据

int len = ::recvfrom(socket\_fd, buf, sizeof(buf), 0, (struct sockaddr \*)&clientaddr, &socklen);

if (len < 0){

printf("%s: recv failed. err %s \n", \_\_func\_\_, strerror(errno));

break;

}

else{

// 获取客户端的ip、和port

char ip[INET6\_ADDRSTRLEN];

inet\_ntop(clientaddr.sin\_family, &clientaddr.sin\_addr, ip, socklen);

int port = ntohs(clientaddr.sin\_port);

printf("%s: client [%s:%d] recv %2d: %s\n", \_\_func\_\_, ip, port, len, buf);

}

// 接收到的数据发送给客户端

len = ::sendto(socket\_fd, buf, sizeof(buf), 0, (struct sockaddr \*)&clientaddr, socklen);

if (len < 0){

printf("%s: send failed. err %s \n", \_\_func\_\_, strerror(errno));

break;

}

}

/// 4、关闭连接

::close(socket\_fd);

}

（2）客户端

#include <stdio.h>

#include <sys/socket.h>

#include <arpa/inet.h> // sockaddr\_in, inet\_addr

#include <unistd.h> // close

#include <cstring>

#include <errno.h>

constexpr const char \*SRV\_ADDR = "127.0.0.1";

constexpr int SRV\_PORT = 8080;

int main()

{

/// 1、创建socket

int socket\_fd = ::socket(AF\_INET, SOCK\_DGRAM, 0); // udp

if(socket\_fd == -1){

printf("%s: create socket failed.\n", \_\_func\_\_);

return -1;

}else{

printf("%s: create socket (fd = %d) success.\n", \_\_func\_\_, socket\_fd);

}

// 2、 发送到服务端

sockaddr\_in servaddr;

servaddr.sin\_family = AF\_INET;

servaddr.sin\_addr.s\_addr = inet\_addr(SRV\_ADDR);

servaddr.sin\_port = htons(SRV\_PORT);

int len ;

char buf[1024] = "hello sockte!";

// 发送

len = ::sendto(socket\_fd, buf, strlen(buf), 0, (sockaddr\*)&servaddr, sizeof(servaddr));

if(len < 0){

printf("%s: send failed. %s \n", \_\_func\_\_, strerror(errno));

return 1;

}

// 接收

len = ::read(socket\_fd, buf, strlen(buf));

//len = ::recv(socket\_fd, buf, strlen(buf), 0);

if(len < 0){

printf("%s: recv failed. %s \n", \_\_func\_\_, strerror(errno));

return 1;

}

else{

printf("recv %2d: %s\n", len, buf);

}

// 3、关闭退出

::close(socket\_fd);

}

（3）测试

发开服务端后，多次打开客户端，可以看到服务端能够正常工作，结果如下

