# IPV4 over IPV6隧道协议实验

## 1.实验目的

IPV4 over IPV6，简称“4over6”是IPV4向IPV6发展进程中，向纯IPV6主干网过渡提出的一种新技术，可以最大程度地继承基于IPV4网络和应用，实现IPV4向IPV6平滑的过渡。

该实验通过实现IPV4 over IPV6隧道最小原型验证系统，让同学们对4over6隧道的实现原理有更加深刻的认识。

## 2.实验要求

在linux系统下，实现4over6隧道系统服务端程序，内容如下：

1）实现服务端与客户端之间控制通道的建立与维护；

2）实现对客户端网络接口的配置；

3）实现对4over6隧道系统数据报文的封装和解封装。

## 3.实验内容

服务端在linux环境下运行，主要有下面几个功能：

1）创建IPV6 TCP套接字，监听服务器和客户端之间的数据通信；

2）维护虚接口，实现对虚接口的读写操作；

3）维护IPV4地址池，实现为新连接客户端分配IPV4地址；

4）维护客户信息表，保存IPV4地址与IPV6套接字之间的映射关系；

5）读取客户端从IPV6 TCP套接字发送来的数据，实现对系统的控制消息和数据消息的处理；

6）实现对数据消息的解封装，并写入虚接口；

7）实现对虚接口接收到的数据报文进行封装，通过IPV6套接字发送给客户端；

8）实现保活机制，监测客户端是否在线，并且定时给客户端发送keeplive消息。

## 4.实验帮助

### 4.1结构体定义

服务端的消息类型与客户端保持一致，结构体定义如下：

struct Msg

{

int length; //长度

char type; //类型

char data[4096]; //数据段

};

服务端的消息类型如下表：

|  |  |  |  |
| --- | --- | --- | --- |
| 类型(char) | 长度(int) | 数据(char[4096]) | 备注 |
| 100 |  | null | 客户端IP地址请求 |
| 101 |  |  | IP地址回应 |
| 102 |  |  | 上网请求 |
| 103 |  |  | 上网回应 |
| 104 |  | null | 心跳包 |

服务器收到客户端发送来的100类型的IP请求报文，会对其回应101类型报文，该报文的数据段，包含了IP地址、路由、三个DNS，服务器会以字符串的形式把这些信息写入报文数据段，格式为“IP 路由 DNS DNS DNS”，字符串之间以空格隔开，类似“13.8.0.2 0.0.0.0 202.38.120.242 8.8.8.8 202.106.0.20”。

每次连接过来一个客户端，服务器都会把该客户端的信息存储到一个客户信息表里，该信息表的定义如下：

typedef struct User\_Info\_Table //客户信息表

{

int fd; //套接字描述符

int count; //标志位

unsigned long int secs; //上次收到keeplive时间

struct in\_addr v4addr; //服务器给客户端分配的IPV4地址

struct in6\_addr v6addr; //客户端的IPV6地址

struct User\_Info\_Table \* pNext; //链表下一个节点

} User\_Info\_Table;

其中客户信息表中的count字段，是用来当一个计数器，当服务器给客户端发送完IP地址之后，会把该客户端的信息都存到客户信息表里，标志位赋值20（服务器每隔20秒给客户端发送心跳包），然后在keeplive线程中，每隔1秒，对所有客户端的count字段进行减1操作，当该客户端的count为0的时候，服务器才给该客户端发送心跳包消息。

服务端地址池定义：

struct IPADDR

{

char addr[32]; //IP地址

int status; //标志位

};

struct IPADDR ipaddr[128];//全局变量的地址池

服务端的地址池中每个IP地址都有一个标志位status，默认为0，当该地址被分配出去，就把该地址对应的标志位置1，当客户端退出，回收该地址，再把标志位置0。

### 4.2 流程解析

服务端代码主要可以分为三部分：主进程循环、读取虚接口线程、keeplive线程。主框架流程图如图4.1所示：



图4.1 主框架流程图

主框架流程解析：

1）创建IPV6套接字，把该套接字加入Select模型字符集;

2）创建tun虚接口;

3）创建客户信息表和地址池：

1、客户信息表主要包含IPV6套接字描述符、标志位、上次收到心跳包时间、虚接口IPV4地址、物理接口IPV6地址；

2、地址池里面是自己分配的一个网段，可以是13.8.0.2到13.8.0.128，13.8.0.1分配给服务器虚接口，其他地址分配给客户端，每个地址对应一个标志位，初始值为0，当该地址被分配出去，标志位置1，地址被回收，标志位置0。

4）获取服务器DNS地址：

1、读取/etc/resolv.conf文件中的nameserver地址；

2、cat /etc/resolv.conf | grep -i nameserver | cut -c 12-30 > dns.txt；

3、上面的代码可以用system调用，把DNS地址重定向到dns.txt文件中；

4、从文件中把地址解析出来。

5）创建keeplive线程；

6）创建读取虚接口线程；

7）主进程中while循环中数据处理。

#### 4.2.1 主进程循环

主进程while循环中主要是Select模型监听所有套接字，然后根据套接字收到数据类型，做不同的处理。while循环流程如图4.2所示：



图4.2 while循环流程图

1）在while循环中，启用Select模型，对所有的套接字进行监听；

2）假如监听到服务器套接字：

1、accept新的连接；

2、把新连接的客户端的套接字描述符加入Select字符集。

3）假如是客户端套接字：

1、首先用ioctl函数判断一下该描述符：ioctl(fd, FIONREAD, &nread)；

2、假如nread不等于0，客户端正常连接：

A、接收数据；

B、对收到的数据解封装；

C、判断数据类型：

a、100（IP请求）：

（1）遍历地址池；

（2）查找未被分配的地址；

（3）通过sprintf函数把要分配的地址，路由(默认0.0.0.0)，3个DNS 按照101消息类型拼接为一个字符串，拷贝到101类型数据段；

（4）把已经封装好的101(IP地址回应)消息发送给客户端；

（5）把客户端的描述符、分配的IPV4地址、物理物理接口IPV6地址、标志位、以及上次接收keeplive时间，写入客户信息表；

（6）其中标志位赋值为发送keeplive消息的时间间隔(20)，上次接收keeplive消息时间赋值为当前时间。

b、102(上网请求消息)：

（1）对数据进行解封装；

（2）取出data段

（3）写入虚接口。

c、104(keeplive消息)：

（1）获取当前时间；

（2）遍历客户信息表，查找该描述符所在节点；

（3）把当前时间赋值给该节点的secs(上次收到keeplive时间)字段。

3、假如nread等于0，客户端已经断开：

A、遍历客户信息表；

B、从信息表中查找该客户端描述符所在节点；

C、取出该节点的IPV4地址；

D、用该地址与地址池中地址进行匹配；

E、匹配成功，就把地址池中该地址的标志位置0；

F、在客户信息表中把该节点删除；

G、Select字符集中清除该描述符。

#### 4.2.2 读取虚接口线程

读取虚接口线程主要功能是从协议栈读取消息，根据消息的目的地址，发送给相应的客户端，读取虚接口流程如图4.3所示：



图4.3 读取虚接口流程图

1）在while循环中，从虚接口读取消息；

2）取出该消息的ip头；

3）获取目的ip地址；

4）遍历客户信息表，查找该目的ip所在的节点；

5）取出该节点的套接字描述符；

6）把从虚接口读取到的消息封装103(上网回应)报头；

7）通过刚才查找到的套接字描述符发送给相应的客户端。

#### 4.2.3 keeplive线程

keeplive线程主要功能是给所有当前处于连接状态的客户端发送心跳包，该线程的流程如图4.4所示：



图4.4 keeplive流程图

1）sleep一秒钟；

2）遍历客户信息表；

3）链表中的每个节点的count字段减1；

4）当该节点的count字段等于0时；

5）获取该节点的套接字描述符；

6）通过套接字描述符向该节点所在客户端发送104(心跳包)类型消息；

7）发送完成，重新把该节点的count字段赋值为20(每隔20秒发送一次心跳包)；

8）判断每个节点的secs字段的值是否大于60;

9）假如secs大于60，则说明该客户端已经超过60秒没有给服务器发送心跳包；

10）获取该节点的IPV4地址；

11）遍历地址池，找到该地址在地址池中所在位置；

12）把该地址的状态字段置为0，回收该地址；

13）从Select字符集中把该节点的套接字描述符清除；

14）关闭该套接字描述符；

15）删除该节点。

### 4.3 实验环境

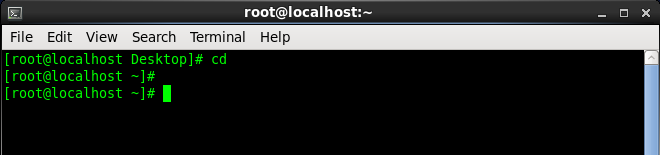
操作系统：CentOS 6.3

内核版本：2.6.32

Gcc版本：4.4.7 20120313 (Red Hat 4.4.7-4) (GCC)

#### 4.3.1安装Gcc

1）以root用户登录系统，打开终端;

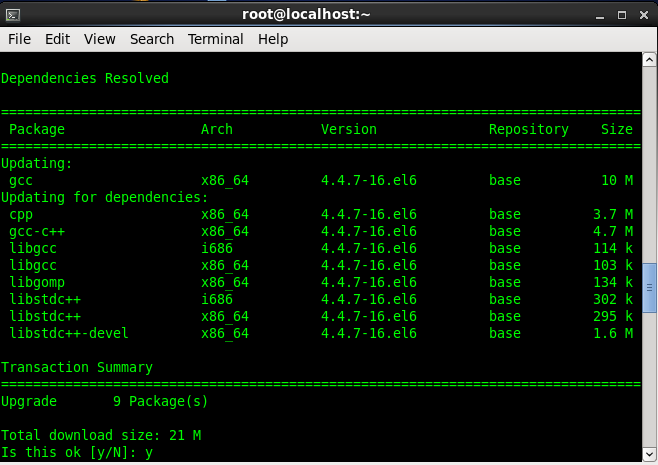


2）查看系统是否已经安装了gcc，输入命令：gcc –version，回车;

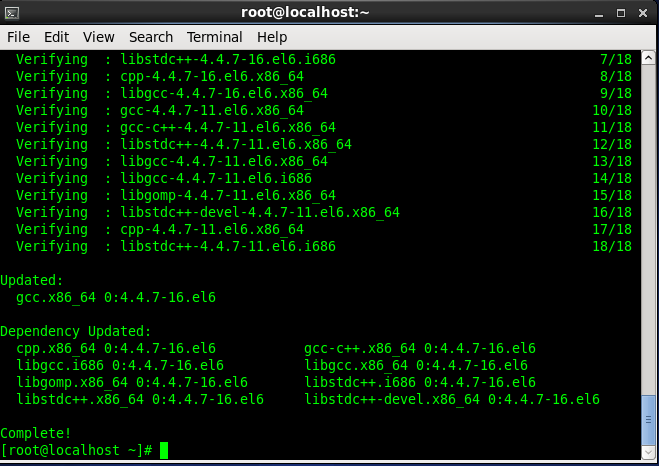
3）假如没有安装，则输入命令：yum install gcc，注意中间有空格，然后回车，如下图：



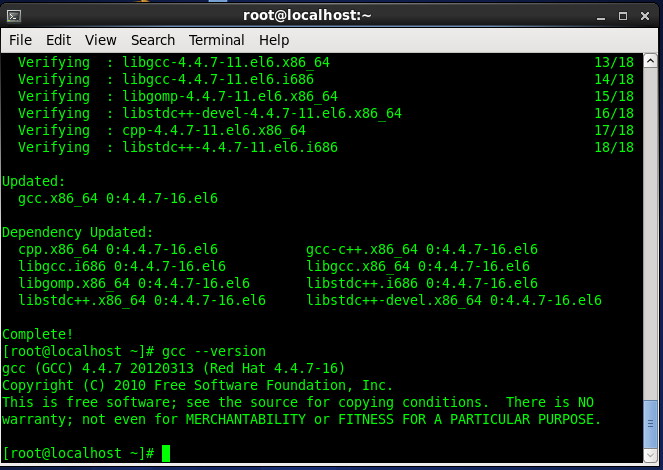
4）系统自动搜索安装包，提示需要下载21兆，输入y，回车，如下图：



5）系统就开始自动下载安装，等待几分钟，最后出现终端上出现Complete提示，就说明安装成功了，如下图：



6）此时再次查看gcc版本，输入gcc –version，回车，出现版本号，说明安装成功。

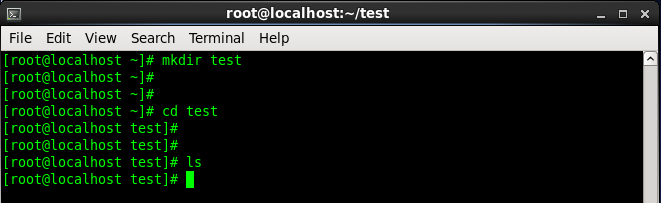


#### 4.3.2使用Gcc

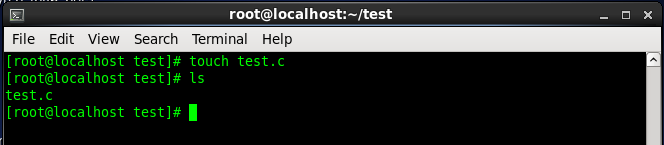
1）首先新建一个文件夹，命名为test，输入命令：mkdir test，然后回车；

2）进入该文件夹，输入命令：cd test，回车；

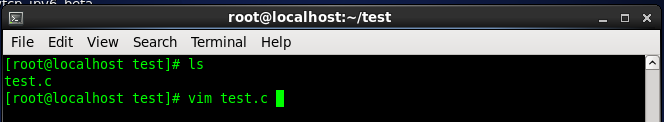
3）输入命令：ls，查看该文件夹下内容，因为是新建的，所以为空，如下图：



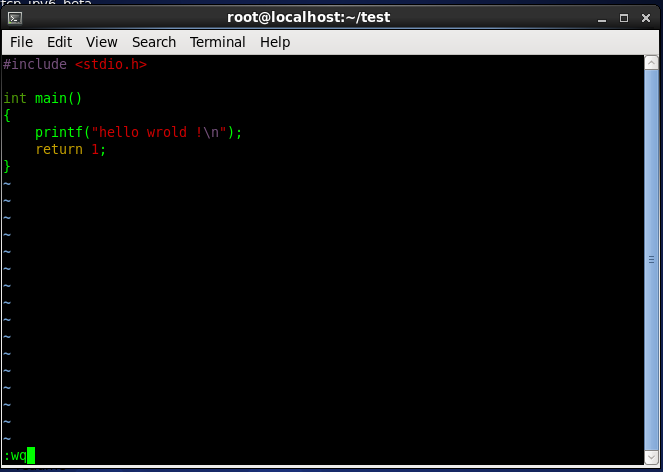
2）然后新建一个C文件，输入命令：touch test.c，回车，然后查看一下，文件已经建好，如下图：



3）然后编辑test.c文件，输入命令：vim test.c，回车，如下图：



4）打开文件后，点击键盘i，进入编辑模式，然后输入下图代码，点击Esc，输入“:wq”，保存代码，然后回车，如下图：



5）接下来编译刚才的文件，输入命令：gcc test.c ，回车，然后输入：ls，发现多了一个a.out的文件，这个就是编译test.c生成的可执行文件，输入命令：./a.out，回车，执行该文件，下面就打印出了我们代码里写的“hello world ！”，如下图：

