UEFI Development Exploration 55 – UEFI and Network 5 (IPv6)

原创 luobing4365 Posted on 2020-04-10 22:33:21 Read 1.8k Collection 2 Likes 2 copyright
Category Column: UEFI Development Article Tags: UEFI UEFI Programming IPv6 UEFI and Network Low-level application development

(Please keep it-> Author: Luo Bing https://blog.csdn.net/luobing4365)

This article starts writing IPV6 code. Because the network debugging assistant I have been using does not support IPV6 communication, I had to spend some time to implement the relevant IPV6 code (under Windows and Linux) myself.

1Build an IPV6 communication environment

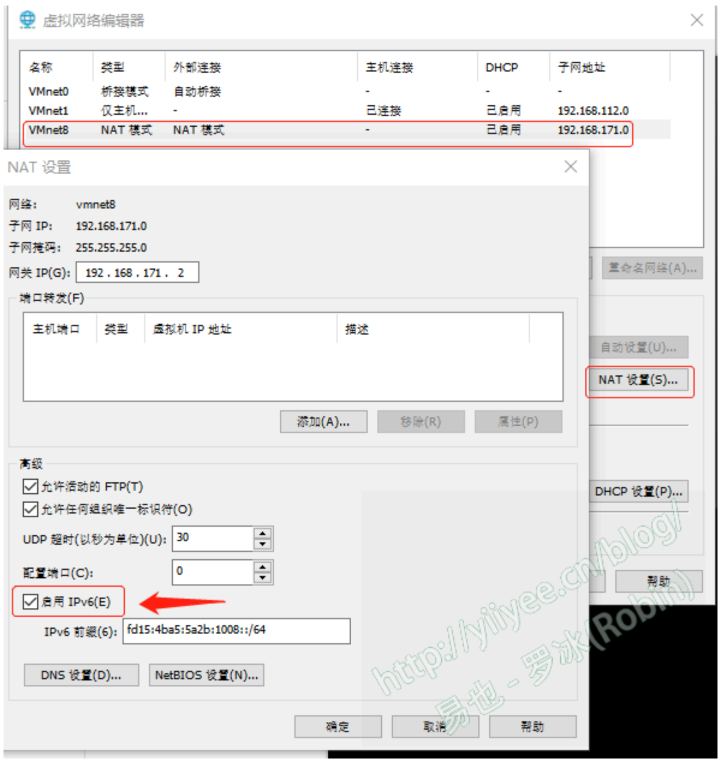
Since the Internet is still basically using IPV4, especially in homes, routers that support IPV6 are rarely used. Therefore, if you need to debug IPV6 code between two machines, it is best to buy a router that supports IPV6 and set it up according to the router manual.

Of course, we can also use virtual machines to build a communication environment. The virtual machines I use all use NAT mode. Before turning on the IPV6 support option, the network card on the virtual machine operating system does not have an external IPV6 address assigned, resulting in the inability to communicate with the host machine.

To make an analogy, turning on the IPV6 support option for a virtual machine is equivalent to adding an IPV6 router to the virtual machine. The network card device on the virtual machine's operating system will be assigned an IPV6 address, and virtual machines can communicate with each other and with the host machine via IPV6.

Record the method of enabling IPV6 in Vmware and VirtualBox.

1) Vmware: Open the menu bar - Edit - Virtual Machine Network Editor - Change Device. In the pop-up menu, select Vmnet8, which is the virtual network card in NAT mode. Open "NAT Settings", check the "Enable IPv6" option, and confirm. As shown in the figure:


虚拟网络编辑器

名称	类型	外部连接	主机连接	DHCP	子网地址
VMnet0	桥接模式	自动桥接	-	-	-
VMnet1	仅主机...	-	已连接	已启用	192.168.112.0
VMnet8	NAT 模式	NAT 模式	-	已启用	192.168.171.0

NAT 设置

网络: vmnet8
子网 IP: 192.168.171.0
子网掩码: 255.255.255.0
网关 IP(G): 192 . 168 . 171 . 2

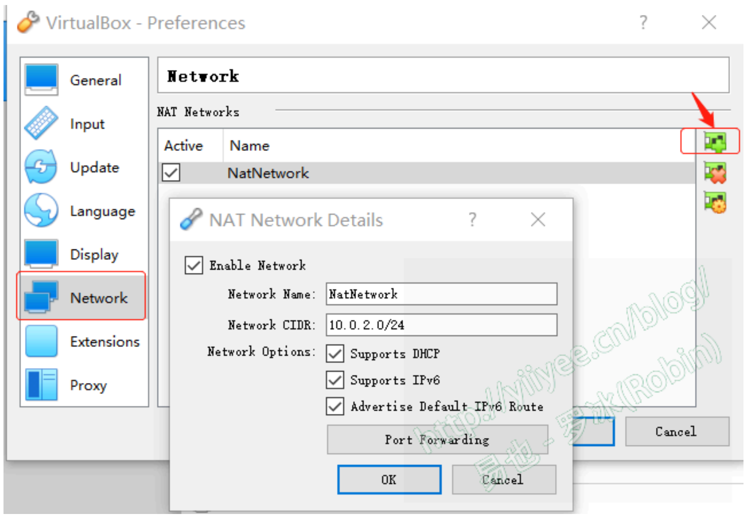
端口转发(F)

主机端口	类型	虚拟机 IP 地址	描述

高级
☑ 允许活动的 FTP(T)
☑ 允许任何组织唯一标识符(O)
UDP 超时(以秒为单位)(U): 30
配置端口(C): 0
☑ 启用 IPv6(E)
IPv6 前缀(6): fd15:4ba5:5a2b:1008::/64

Figure 1 VMware enabling IPv6

2) VirtualBox: Open File-Prefence in the menu bar, select Network, and add NAT network, as shown in the figure:


VirtualBox - Preferences

Network

NAT Networks

Active	Name
✓	NatNetwork

NAT Network Details

☑ Enable Network

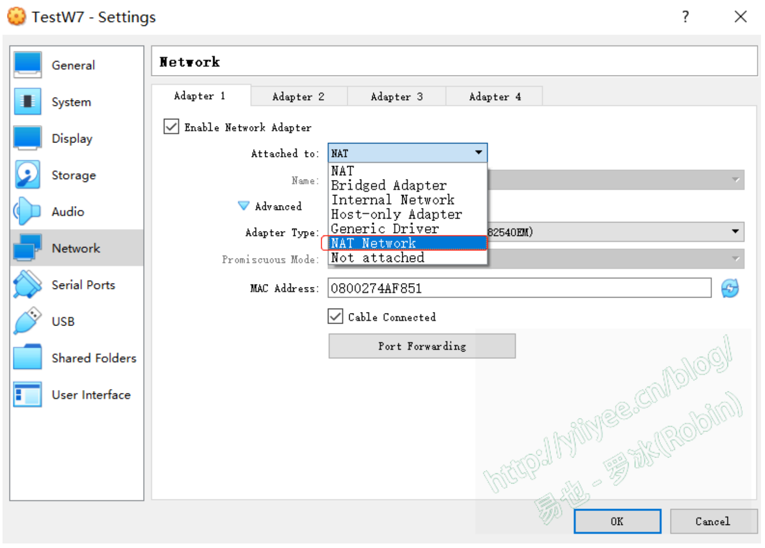
Network Name: NatNetwork

Network CIDR: 10.0.2.0/24

Network Options: ☑ Supports DHCP ☑ Supports IPv6 ☑ Advertise Default IPv6 Route

Figure 2 Adding NatNetwork to VirtualBox

Check all options that support IPv6. Select the virtual machine that needs to support IPv6, open its Setting option, select Network, and change the network card device to NatNetwork, as shown in the figure:


TestW7 - Settings

Network

Adapter 1 | Adapter 2 | Adapter 3 | Adapter 4

☑ Enable Network Adapter

Attached to: NAT (NAT, Bridged Adapter, Internal Network, Host-only Adapter, Generic Driver, [✓ NAT Network], Not attached)

MAC Address: 0800274AF851

☑ Cable Connected

Figure 3 Modify the network card type

After the modification is completed, the virtual machine can support IPv6.

2 Support IPv6 in Nt32 emulator

The basic steps for setting it up are similar to the steps in the previous blog to enable the Nt32 simulator to support IPv4. Note that the network protocol stack in Figure 4 is very similar to IPv4, except that the configuration tools have changed from ping and Ifconfig to ping6 and Ifconfig6.

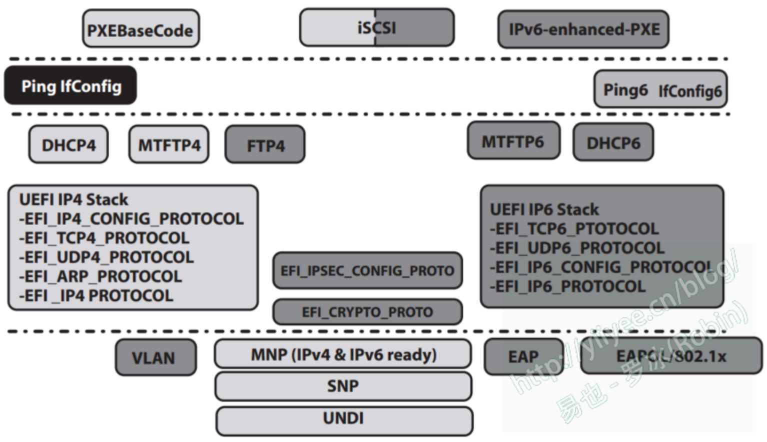


Figure 4 UEFI network protocol stack (from Harnessing the uefi shell p107)

From the previous blog, we already know that the IPv6 protocol driver is in NetworkPkg. Therefore, we need to compile these drivers first. The compilation command is as follows:

```
C:\MyWorkspace>build -p NetworkPkg\NetworkPkg.dsc -a IA32
```

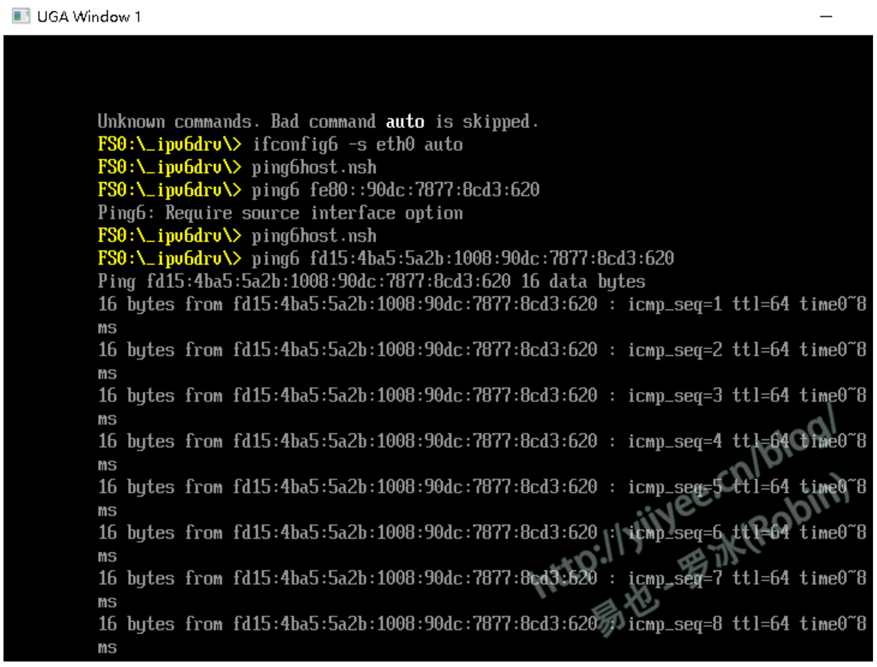
Copy the corresponding driver to the Nt32 directory, start the simulation environment (the network card has been completed when IPv4 was configured before), and load the IPv6 network card driver. The command is as follows:

```
FS0:\>load Mtftp6Dxe.efi Ip6Dxe.efi VlanConfigDxe.efi Udp6Dxe.efi Dhcp6Dxe.efi Mtftp6Dxe.efi TcpDxe.efi
```

To configure the network card device to automatically obtain the IPv6 address, the command is:

```
FS0:\>ifconfig6 -s eth0 auto
```

You can use the command *ifconfig6 -l eth0* to check whether the address is allocated successfully. After the configuration is complete, test the communication between it and the host:


UGA Window 1

```
Unknown commands. Bad command auto is skipped.
FS0:\_ipv6drv\> ifconfig6 -s eth0 auto
FS0:\_ipv6drv\> ping6host.nsh
FS0:\_ipv6drv\> ping6 fe80::90dc:7877:8cd3:620
Ping6: Require source interface option
FS0:\_ipv6drv\> ping6host.nsh
FS0:\_ipv6drv\> ping6 fd15:4ba5:5a2b:1008:90dc:7877:8cd3:620
Ping fd15:4ba5:5a2b:1008:90dc:7877:8cd3:620 16 data bytes
16 bytes from fd15:4ba5:5a2b:1008:90dc:7877:8cd3:620 : icmp_seq=1 ttl=64 time0~8
ms
16 bytes from fd15:4ba5:5a2b:1008:90dc:7877:8cd3:620 : icmp_seq=2 ttl=64 time0~8
ms
16 bytes from fd15:4ba5:5a2b:1008:90dc:7877:8cd3:620 : icmp_seq=3 ttl=64 time0~8
ms
16 bytes from fd15:4ba5:5a2b:1008:90dc:7877:8cd3:620 : icmp_seq=4 ttl=64 time0~8
ms
16 bytes from fd15:4ba5:5a2b:1008:90dc:7877:8cd3:620 : icmp_seq=5 ttl=64 time0~8
ms
16 bytes from fd15:4ba5:5a2b:1008:90dc:7877:8cd3:620 : icmp_seq=6 ttl=64 time0~8
ms
16 bytes from fd15:4ba5:5a2b:1008:90dc:7877:8cd3:620 : icmp_seq=7 ttl=64 time0~8
ms
16 bytes from fd15:4ba5:5a2b:1008:90dc:7877:8cd3:620 : icmp_seq=8 ttl=64 time0~8
ms
```

Figure 5: Ping6 connects to the host

3 Writing TCP6 Code

In the previous article, I introduced the process of writing TCP4 code. The process of writing TCP6 code is the same. However, I didn't find a ready-made TCP6 test tool, so I had to write one myself.

We will design an "echo" program, where the server acts as an echo wall and returns all messages sent by the client. Since most of my UEFI programs are compiled on Windows, I can just write a Windows server.

3.1 Windows server code writing

Open Vs2015 and create a new project of Win32 Console Application type. Add the following header files:

```
#include <stdio.h>
#include <stdlib.h>
#include <WinSock2.h>
#include <Ws2tcpip.h>   //InetPton() header file
```

And introduce Microsoft's socket library:

```
#pragma comment(lib,"ws2_32") //Introduce library function
```

Unlike TCP4, the address type is changed to SOCKADDR_IN6, and the rest of the code is almost the same as TCP4.

Since this is server code, IPv6 addresses do not need to be processed. However, in the client that needs to obtain the server address, it is different from the TCP4 writing and the Linux writing.

To demonstrate the difference, I also wrote a client for Windows. (Later, I wrote the IPv6 UDP server and client as well.)

For details, please refer to the document on the Microsoft website: https://docs.microsoft.com/en-us/windows/win32/winsock/ipv6-enabled-client-code-2?redirectedfrom=MSDN

The getaddrinfo() function provided by Microsoft can resolve the machine name, IPv4, and IPv6 addresses. Together with getnaminfo(), it can obtain the IPv6 address of the server. Then use connect() to connect to the server for communication.

Only this part is slightly different from the TCP4 writing, everything else is the same.

3.2 UEFI Client Writing

The programming method of writing TCP6 UEFI client is very similar to TCP6 under Linux. In fact, my code is also ported from the previous Linux code, just slightly modifying the variable types.

First, establish a TCP6 socket. The code is as follows:

```
socketID = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
```

Then convert the IPv6 address passed in the command line and connect to the server through the IP address:

```
inet_pton(AF_INET6, Argv[1], &ServerIp.sin6_addr);
```

```
connect(sockfd, (struct sockaddr *) &ServerIp, sizeof(ServerIp));
```

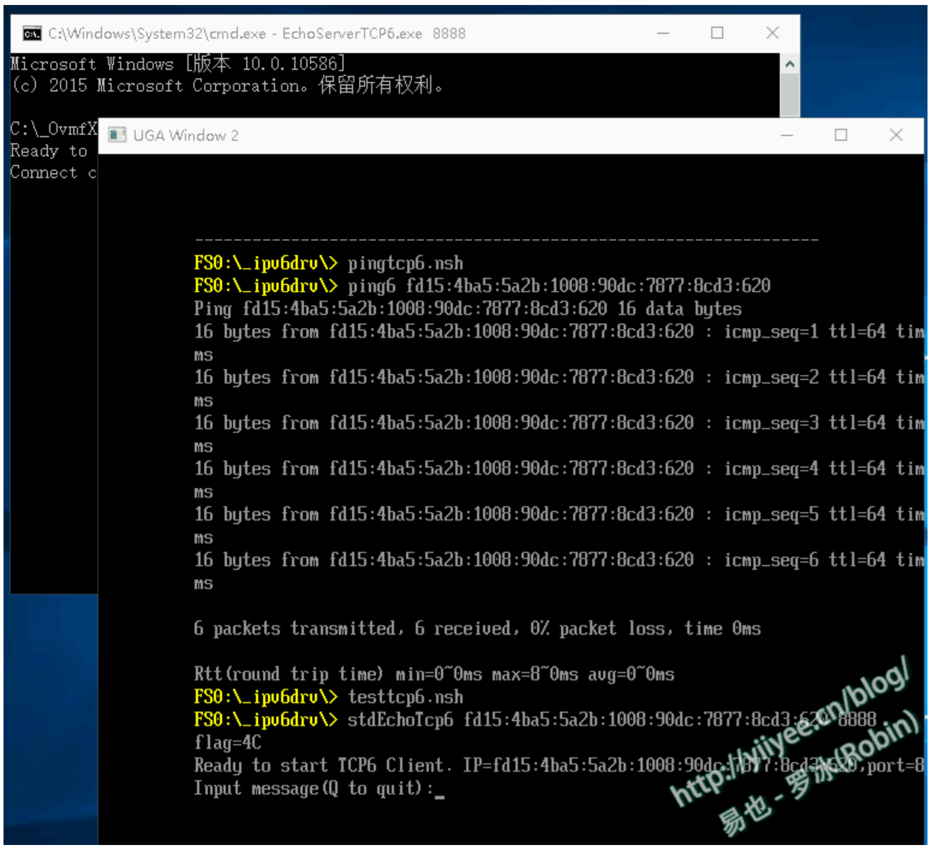
Finally, build a loop for sending and receiving data, sending data to the server and receiving data from the server.

4 Compile test

The compilation command is as follows:

```
C:\MyWorkspace> build -p RobinPkg\RobinPkg.dsc -a IA32 -m RobinPkg\Applications\stdEchoTcp6\stdEchoTcp6.inf
```

After compiling, copy the executable file to the directory where the Nt32 simulation environment is located. According to the method described in Section 2, build an IPv6 test environment and perform the test:



```
Microsoft Windows [版本 10.0.10586]
(c) 2015 Microsoft Corporation。保留所有权利。

C:\_OvmfX
Ready to
Connect c
```

```
-----------------------------------------------------------------
FS0:\_ipv6drv\> pingtcp6.nsh
FS0:\_ipv6drv\> ping6 fd15:4ba5:5a2b:1008:90dc:7877:8cd3:620
Ping fd15:4ba5:5a2b:1008:90dc:7877:8cd3:620 16 data bytes
16 bytes from fd15:4ba5:5a2b:1008:90dc:7877:8cd3:620 : icmp_seq=1 ttl=64 tim
ms
16 bytes from fd15:4ba5:5a2b:1008:90dc:7877:8cd3:620 : icmp_seq=2 ttl=64 tim
ms
16 bytes from fd15:4ba5:5a2b:1008:90dc:7877:8cd3:620 : icmp_seq=3 ttl=64 tim
ms
16 bytes from fd15:4ba5:5a2b:1008:90dc:7877:8cd3:620 : icmp_seq=4 ttl=64 tim
ms
16 bytes from fd15:4ba5:5a2b:1008:90dc:7877:8cd3:620 : icmp_seq=5 ttl=64 tim
ms
16 bytes from fd15:4ba5:5a2b:1008:90dc:7877:8cd3:620 : icmp_seq=6 ttl=64 tim
ms

6 packets transmitted, 6 received, 0% packet loss, time 0ms

Rtt(round trip time) min=0~0ms max=8~0ms avg=0~0ms
FS0:\_ipv6drv\> testtcp6.nsh
FS0:\_ipv6drv\> stdEchoTcp6 fd15:4ba5:5a2b:1008:90dc:7877:8cd3:620 8888
flag=4C
Ready to start TCP6 Client. IP=fd15:4ba5:5a2b:1008:90dc:7877:8cd3:620,port=8
Input message(Q to quit):_
```

Figure 6 Test procedure

As of this article, all blogs related to the network have been completed. As for the application layer protocols, including pxe, ftp, etc., they have not been used much in the project yet. If I encounter similar projects in the future, I will summarize them in depth. The next article will start to study UEFI's support for USB.

TCP6 uefi client code:

Gitee address: https://gitee.com/luobing4365/uefi-explorer
Project code is located in: /FF RobinPkg/ RobinPkg /Applications/stdEchoTCP6

Windows-side IPv6 code, including TCP server and client, UDP server and client (compiled in VS2015):

Gitee address: https://gitee.com/luobing4365/uefi-explorer
Project code is located at: /55blog-WindowsIPv6