UEFI Development Exploration 54 – UEFI and Network 4 (IPv4)

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

This article mainly discusses how to use StdLib library functions to write TCP4 code.

In the previous article, we used the TCP4 Protocol provided by UEFI to write code, which was complicated and required us to rebuild all the code. We still hope to use the standard Socket library to program directly, so that we can reuse the previously written Socket code and reduce the programming workload.

1 BSD Socket Interface

StdLib provides the standard BSD Socket interface, which allows communication between different hosts or different processes on the same computer. It is the de facto standard interface for connecting to the Internet.

The API interfaces required by BSD Socket are basically provided in the UEFI implementation. The main header files are as follows (under StdLib\Include):

<sys/socket.h> socket core functions and data structures;
<netinet/in.h> Internet address families, AF_INET and AF_INET6 address families and their corresponding protocol families PF_INET and PF_INET6, including IP addresses and TCP and UDP port numbers;
<arpa/inet.h> Some functions related to IP addresses, such as inet_pton() for address translation, etc.;

Since UEFI is not a complete operating system, many APIs in BSD Socket are not implemented, such as un.h for local communication between programs on the computer. Most of these unimplemented functions are not related to network communication and have little impact on our UEFI network programming.

Most programmers should have written Socket code, and the method of writing under UEFI is actually similar. If necessary, the previously written code can be directly ported over.

2 Function Introduction

The library functions provided in Socket are introduced as follows (the following content is only valid for UEFI platform).

int socket(IN INT32 domain, IN INT32 type, IN INT32 protocol);
- ●Function Create a socket of a certain type and allocate system resources.
- ●Parameter domain: Determine the protocol family, PF_INET is IPv4, PF_INET6 is IPv6
  type: type, supports SOCK_STREAM, SOCK_DGRAM and SOCK_RAW;
  protocol: Specifies the protocol used by the transport layer. IPPROTO_TCP, type must be specified as SOCK_STREAM; IPPROTO_UDP, type must be specified as SOCK_DGRAM; 0-254, type is SOCK_RAW, the meaning of the value can be found at: https://www.iana.org/assignments/protocol-numbers/protocol-numbers.xml
- ●Return value Returns the socket handle. If -1 is returned, it means an error has occurred

int bind (IN int s, IN const struct sockaddr * name, IN socklen_t namelen);
- ●Function Connect the socket handle and the socket address structure.
- ●Parameter s: socket handle, returned by the socket() function;
  name: pointer to the sockaddr structure, representing the address to be bound;
  namelen: the size of the sockaddr structure
- ●Return value 0 means the operation is successful, -1 means failure

int listen (IN int s, IN int backlog);
- ●Function Listen for connections, applicable to connection-oriented mode, such as sockets of type SOCK_STREAM.
- ●Parameter s: socket handle, returned by the socket() function;
  backlog: the maximum number of connections that can be waited for;
- ●Return value 0 indicates successful operation, -1 indicates failure

int connect (int s, const struct sockaddr * address, socklen_t address_len);
- ●Function: Establish a connection through the network
- ●Parameter s: socket handle, returned by the socket() function;
  address: network address;
  namelen: the byte length of the network address
- ●Return value 0 indicates successful operation, -1 indicates failure

int accept (int s, struct sockaddr * address, socklen_t * address_len);
- ●Function: Receive network connection, generally used by server program
- ●Parameter: s: socket handle, returned by socket() function;
  address: network address;
  namelen: length of network address in bytes
- ●Return value: 0 indicates successful operation, -1 indicates failure

ssize_t recv (int s, void * buffer, size_t length, int flags);

- ●Function Receive data
- ●Parameter s: socket handle, returned by socket() function;
  buffer: buffer used to receive data;
  flags: flag used to control the receiving method, such as receiving OOB (out-of-band data). Usually set to 0;
- ●Return value -1 indicates failure, and other integer values greater than -1 indicate the number of data received

ssize_t send (int s, CONST void * buffer, size_t length, int flags);

- ●Function Send data
- ●Parameter s: socket handle, returned by the socket() function;
  buffer: buffer used to send data;
  flags: flags used to control the sending method, such as sending OOB (out-of-band data). Generally set to 0;
- ●Return value -1 indicates failure, and other integer values greater than -1 indicate the number of data sent

Other functions, including recvfrom(), sendto(), setsockopt(), etc., are not used in this example. You can check the relevant information.

3. Write the program

In the example I prepared, the server uses the network debugging assistant and the client uses the UEFI NT32 simulation environment. Of course, you can also write a server code to better demonstrate the process of network programming. I will write it when I have time.

The general process of writing TCP network server code is as follows:

1) Use socket() to create a Socket;
2) Use bind() to bind the Socket to the local IP and TCP port;
3) Create a waiting queue for the client's connection;
4) Process the connection in a loop and receive the client's connection through accept();
5) Use recv() or send() to receive and send data;
6) Use close() to close the Socket and terminate the communication.

The TCP client writing process is:

1) Use socket() to create a Socket;
2) Use connect() to send a connection request to the server, wait for a response, and execute further;
3) Process the connection in a loop, using recv() or send() to receive and send data;
4) Use close() to close the Socket and terminate communication.

The UEFI code is written basically according to the above process. For details, please refer to the provided routines. In the TCP communication function code, all functions in StdLib are used.

The compilation command is as follows:

```
C:\MyWorkspace> build -p RobinPkg\RobinPkg.dsc -a IA32 -m RobinPkg\Applications\stdEchoTcp4\stdEchoTcp4.inf
```

Unexpectedly, an error message was reported during compilation. I checked the error location and found that it was caused by the statement in line 83 of \StdLib\BsdSocketLib\ns_addr.c:

```
if ((cp = strchr(buf, ':')) &&
```

Because the assignment in the conditional expression caused the C4706 warning, it was easy to fix. However, I didn't want to touch the source code of StdLib, so I solved it by changing the compilation options.

In conf\tool_def.txt, find the assignment line of DEBUG_VS2015x86_IA32_CC_FLAGS (I usually compile with debug, that is, use the -b DEBUG option. If it is Release, modify the assignment statement of DEBUG_VS2015x86_IA32_CC_FLAGS), and change /W4 to /W3.

4 Testing Procedure

The method of setting up a network test environment has been discussed in previous blogs and will not be repeated here.

1) Start the server program

Figure 1 Server

2 Start the UEFI client program

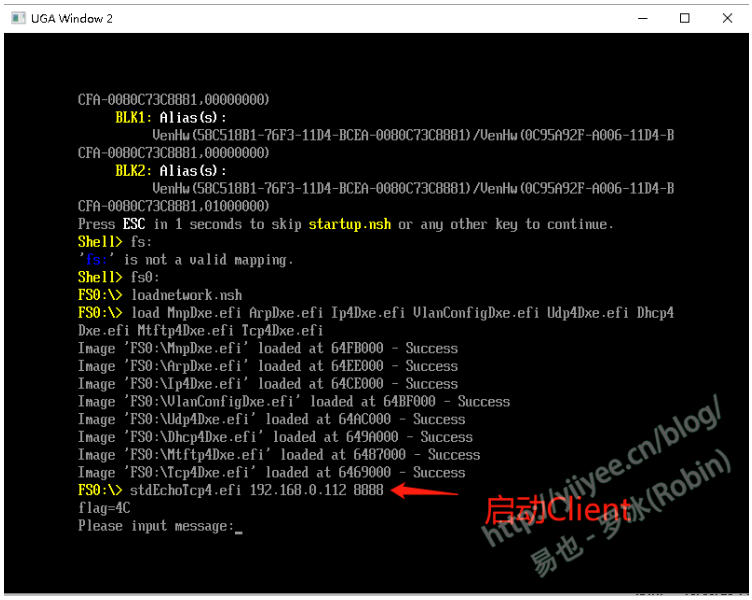

Figure 2 Client

3 Communication test

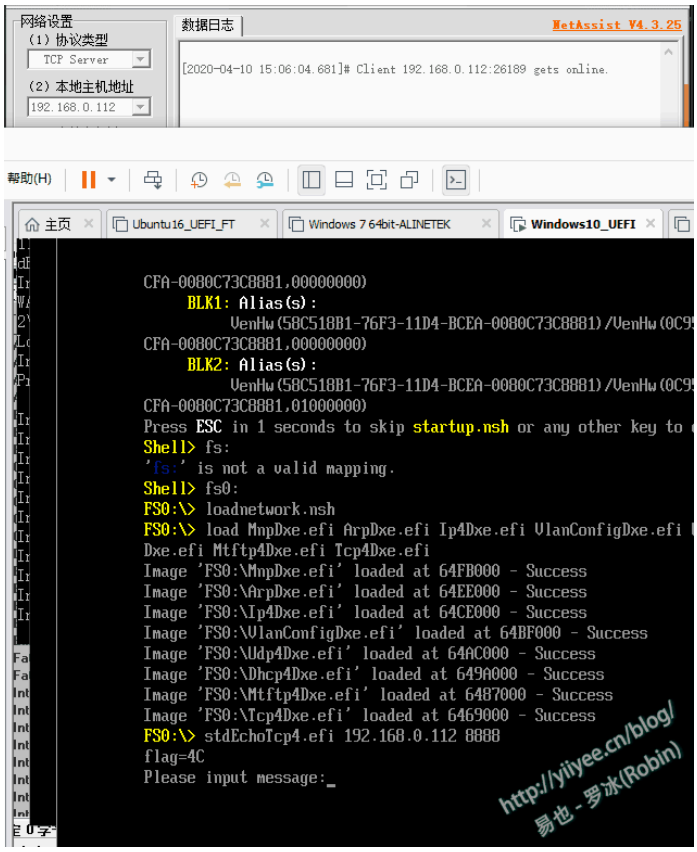

Figure 3 Send and receive test

Gitee address: https://gitee.com/luobing4365/uefi-explorer
Project code is located at: / FF RobinPkg/RobinPkg/Applications/stdEchoTCP4