

UEFI Development Exploration 42 – Protocol Usage 1

原创

luobing4365

Posted on 2020-02-28 11:28:41

Read 1.8k

Collection 6

Likes

Category columns: 

UEFI Development

Article Tags: 

UEFI Programming

UEFI Protocol

Low-level programming

UEFI Programming Practice

UEFI Development

This column includes this content

503 Subscribe104 articles

Subscribe to our column

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

Although I have been using various protocols to implement the required program functions, the principles and implementation methods behind them have always been vague. I adhere to the pragmatism of "use it first and then talk about it". I happened to have some free time on the weekend, so I explored the vague parts of my understanding of the protocol.

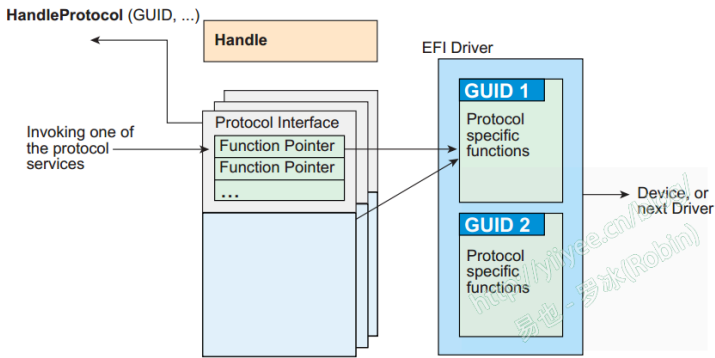


Figure 1 Protocol structure

The following figure shows the structure of the Protocol, which is taken from UEFI Spec 2.8 page 45.

The questions I want to clarify are:

- 1) How to use the Protocol service?
- 2) How is the Protocol mechanism implemented in UEFI?
- 3) How to implement a Protocol?

1. Function Learning

The functions related to Protocol processing are listed in the Spec:

名称	类型	描述
InstallProcotocolInterface	Boot	在设备句柄上安装一个 Protocol 接口
UninstallProtocolInterface	Boot	移除设备句柄上的 Protocol 接口
ReinstallProtocolInterface	Boot	重新安装设备句柄上的 Protocol
RegisterProtocolNoitfy	Boot	注册一个 event, 当指定的 Protocol 接口安装时, 会被触发
LocateHandle	Boot	返回支持指定 Protocol 的句柄列表
HandleProtocol	Boot	查询句柄是否支持指定的 Protocol
LocateDevicePath	Boot	对支持指定 Protocol 的设备路径, 枚举其所有设备, 并且返回最靠近路径的句柄
OpenProtocol	Boot	查询句柄是否支持指定的 Protocol, 如果支持, 则打开该 Protocol。这是 HandleProtocol 的扩展版本
CloseProtocol	Boot	关闭打开的 Protocol
OpenProtocolInformation	Boot	获得指定设备上指定 Protocol 的打开信息
ConnectController	Boot	将一个或多个 drivers 连接到控制器上
DisconnectController	Boot	从一个控制器上将一个或多个 drivers 断开
ProtocolsPerHandle	Boot	通过 protocol GUID 检索安装在设备句柄上的接口
LocateHandleBuffer	Boot	返回支持指定 Protocol 的句柄列表
LocateProtocol	Boot	返回支持指定 Protocol 句柄中的第一个
InstallMultipleProtocolInterfaces	Boot	在 handle 上安装一个或多个 Protocol 接口
UninstallMultipleProtocolInterfaces	Boot	从 handle 上移除一个或多个 Protocol 接口

Figure 2 Related functions

If you only use Protocol, you don't need to pay attention to many of the above functions. Here is a brief description of the functions you need to use:

**1-1 OpenProtocol()**

```
typedef EFI_STATUS (EFIAPI *EFI_OPEN_PROTOCOL) (
    IN EFI_HANDLE Handle, // Specify the Protocol interface installed by this Handle to be opened
    IN EFI_GUID *Protocol, // Protocol to be opened (pointer to the Protocol GUID)
    OUT VOID **Interface OPTIONAL, // Return the opened Protocol object, or NULL if none
    IN EFI_HANDLE AgentHandle, // Open the Image of this Protocol (for UEFI Application)
    IN EFI_HANDLE ControllerHandle, // If the opened Protocol is a driver that complies with the UEFI Driver Model, this parameter is the controller that controls the Protocol interface, otherwise it is optional and may be NULL
    IN UINT32 Attributes // Parameters for opening the Protocol);
```

**Function description:**

Among the above parameters, Handle is the object of the device in UEFI, which is the provider of Protocol. If the Protocol is in the Protocol list of Handle, the pointer of the Protocol object will be written to \*Interface (note that Interface is a pointer to a pointer).

When OpenProtocol() is called in the driver, ControllerHandle is the controller that owns the driver, that is, the controller that requests to use this Protocol; AgentHandle is the Handle that owns the EFI\_DRIVER\_BINDING\_PROTOCOL object. If the application (UEFI Application) is opened, AgentHandle is the Handle of the program, that is, the first parameter of the UefiMain function, and ControllerHandle can be ignored.

If this function returns an error, there are many possibilities. Please refer to the Spec (UEFI spec 2.8 page 187) for details.

Attributes has 6 values that can be used:

```
#define EFI_OPEN_PROTOCOL_BY_HANDLE_PROTOCOL 0x00000001
#define EFI_OPEN_PROTOCOL_GET_PROTOCOL 0x00000002
#define EFI_OPEN_PROTOCOL_TEST_PROTOCOL 0x00000004
#define EFI_OPEN_PROTOCOL_BY_CHILD_CONTROLLER 0x00000008
#define EFI_OPEN_PROTOCOL_BY_DRIVER 0x00000010
#define EFI_OPEN_PROTOCOL_EXCLUSIVE 0x00000020
```

The above parameters can be freely selected according to actual conditions.

**1-2 HandleProtocol()**

```
typedef EFI_STATUS (EFIAPI *EFI_HANDLE_PROTOCOL) (
    IN EFI_HANDLE Handle, // Handle to be queried to see if it supports the specified Protocol
    IN EFI_GUID *Protocol, // The only protocol flag published, that is, a pointer to a valid Protocol GUID
    OUT VOID **Interface // Return the protocol to be queried);
```

**Function description:**

It is a simplified version of OpenProtocol(), and does not require the values of AgentHandle, ControllerHandle, and Attributes (the three parameters in OpenProtocol()). In fact, OpenProtocol() is still used inside the function, with AgentHandle as gDxeCoreImageHandle, ControllerHandle using NULL, and Attributes using EFI\_OPEN\_PROTOCOL\_BY\_HANDLE\_PROTOCOL.

**1-3 LocateHandleBuffer()**

```
typedef EFI_STATUS (EFIAPI *EFI_LOCATE_HANDLE_BUFFER) (
    IN EFI_LOCATE_SEARCH_TYPE SearchType, // Specify which handle to return, i.e. set the search method
    IN EFI_GUID *Protocol OPTIONAL, // Specify the Protocol (GUID), this parameter is only valid when SearchType is ByProtocol
    IN VOID *SearchKey OPTIONAL, // Depending on the SearchType, provide the search keyword
    IN OUT UINTN *NoHandles, // Return the number of found handles
    OUT EFI_HANDLE **Buffer // Allocate and return the handle array);
```

**Function description:**

This function returns one or more Handles to match the request specified in SearchType. The memory required by the parameter Buffer is allocated by the function using EFI\_BOOT\_SERVICES.AllocatePool(). After use, the caller must call FreePool() to release it.

There are three types of SearchType:

AllHandles: The parameters Protocol and SearchKey will be ignored, and all Handles in the system will be returned;

ByRegisterNotify: Find the Handle that matches the SearchKey from RegisterProtocolNotify, and the Protocol parameter will be ignored;

ByProtocol: Find the Handle that supports the specified Protocol from the system Handle database, and the SearchKey parameter will be ignored.

**1-4 LocateProtocol()**

```
typedef EFI_STATUS (EFIAPI *EFI_LOCATE_PROTOCOL) (
    IN EFI_GUID *Protocol, // Protocol to be queried
    IN VOID *Registration OPTIONAL, // Optional parameter, get the registration key from RegisterProtocolNotify()
    OUT VOID **Interface // Return the first matching Protocol interface in the system);
```

Function description:

This function finds the first device Handle that supports the Protocol and returns its Protocol Interface. It does not need to specify the Handle when calling, which is simpler than OpenProtocol() and HandleProtocol().

#### 1-5 OpenProtocolInformation()

```
typedef EFI_STATUS (EFI_API *EFI_OPEN_PROTOCOL_INFORMATION) (
    IN EFI_HANDLE Handle, // Device handle of the queried Protocol
    IN EFI_GUID *Protocol, // Protocol (GUID) to be queried
    OUT EFI_OPEN_PROTOCOL_INFORMATION_ENTRY **EntryBuffer, // The queried information is returned
    OUT UINTN *EntryCount // Number of elements in the EntryBuffer array
);
```

Function description:

This function is used to obtain the opening information of the specified Protocol on the specified device. It allocates memory for EntryBuffer and stores the returned information in it. The caller is responsible for releasing the memory. The data structure of EntryBuffer is as follows:

```
typedef struct {
    EFI_HANDLE AgentHandle;
    EFI_HANDLE ControllerHandle;
    UINT32 Attributes;
    UINT32 OpenCount;
} EFI_OPEN_PROTOCOL_INFORMATION_ENTRY
```

The returned information includes AgentHandle, ControllerHandle, opened properties and number of opened. In many cases, the information obtained by this function is used to close the opened Protocol.

#### 1-6 CloseProtocol()

```
typedef EFI_STATUS (EFI_API *EFI_CLOSE_PROTOCOL) (
    IN EFI_HANDLE Handle, // Device handle of the previously opened Protocol interface, ready to be closed
    IN EFI_GUID *Protocol, // The only Protocol flag released, that is, a pointer to a valid Protocol GUID
    IN EFI_HANDLE AgentHandle, // Open the Image of this Protocol (for UEFI Application)
    IN EFI_HANDLE ControllerHandle, // If the opened Protocol is a driver that complies with the UEFI Driver Model, this parameter is the controller that controls the Protocol interface, otherwise it is optional and may be NULL
);
```

Function description:

After using the Protocol, you need to close it through this function. The Protocol opened through HandleProtocol() and LocateProtocol() does not specify an AgentHandle and cannot be closed directly. You need to call OpenProtocolInformation() to obtain the AgentHandle and ControllerHandle, and then close it.

Other functions are not listed here. Read the Spec when needed. You can find the corresponding content in Spec 7.3 Protocol Handler Services.

## 2Using Protocol

In order to become familiar with the use of Protocol, I wrote a small program to determine whether the Protocol exists.

In fact, when using various protocols, the most important point is to provide corresponding support under UEFI. Just like the development under Legacy BIOS before, for example, if the BIOS does not support the mouse interrupt, then it is too much work to support the mouse in your own code.

For the convenience of display, I transplanted the Chinese character display code I wrote before (Exploration Series Blog 18) into the code with UefiMain as the entry point. I also encountered some minor problems during the process, which are also described in the related problem collection of the series of blogs, so I will not repeat them here.

The code is relatively simple. In fact, after reading the description of the above function, the code is not difficult to write.

My main purpose is to locate the incoming Protocol GUID to see if the corresponding Protocol exists in the system and the number of related Handles.

Specifically, taking the serial port as an example, we need to find out whether SerialProtocol exists and how many serial ports exist. The actual code is in the function ListProtocolMsg() in the example TryProtocol.c in this chapter, as follows:

```

169 //Name: ListProtocolMsg
170 //Input:
171 //Output:
172 //Descriptor:
173 EFI_STATUS ListProtocolMsg(EFI_GUID *Protocol)
174 {
175     EFI_STATUS Status;
176     EFI_HANDLE *myHandleBuff = NULL;
177     // UINTN HandleIndex = 0;
178     UINTN HandleCount = 0;
179     UINTN i;
180
181     //get the handles which supports
182     Status = gBS->LocateHandleBuffer(
183         ByProtocol,
184         Protocol,
185         NULL,
186         &HandleCount,
187         &myHandleBuff
188     );
189     // if (EFI_ERROR(Status)) return Status; //unsupport
190
191     //list message
192     Print((const CHAR16*)"=== Protocol信息获取 Begin ===\n");
193     Print((const CHAR16*)" GUID: {0x%08x, 0x%04x, 0x%04x, {", Protocol->Data1, Protocol->Data2, Protocol->Data3);
194     for (i = 0; i < 8; i++)
195         Print((const CHAR16*)"L" 0x%02x", Protocol->Data4[i]);
196     Print((const CHAR16*)"L"}\n");
197     if (EFI_ERROR(Status))
198     {
199         Print((const CHAR16*)"L"没有找到相应的Protocol!\n");
200     }
201     else
202     {
203         Print((const CHAR16*)"L"找到Handle: %d个\n", HandleCount);
204     }
205     Print((const CHAR16*)"L"=== Protocol信息获取 END ===\n");
206     if(myHandleBuff!=NULL)
207         FreePool(myHandleBuff);
208     return Status;
209 }

```

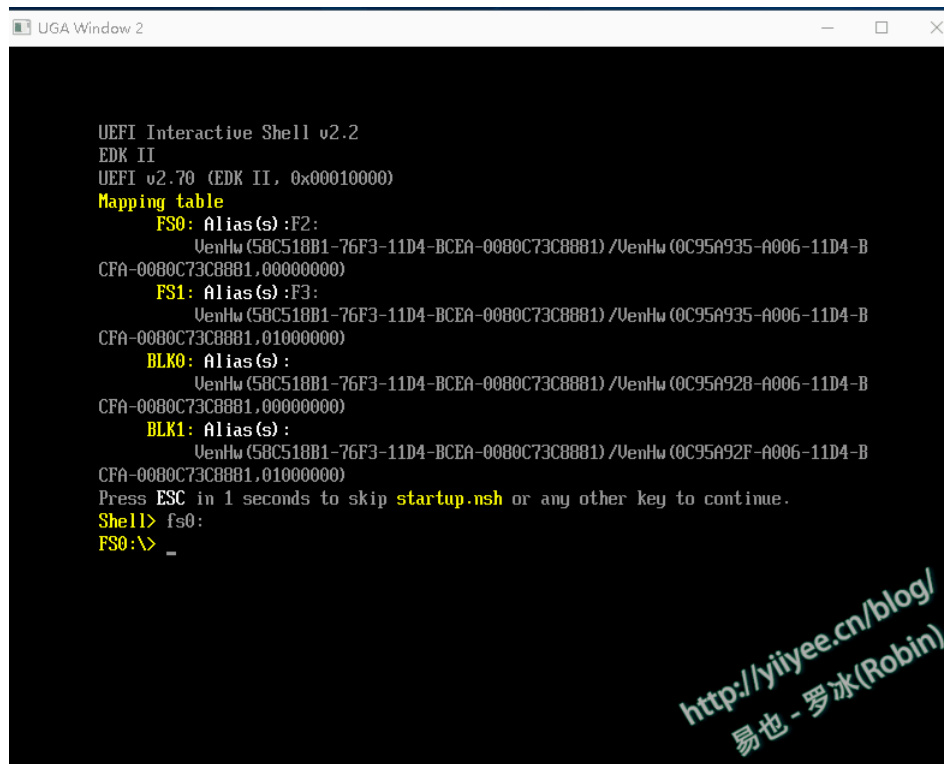
Figure 3 Protocol information enumeration

In addition, in normal use, the code I handle Protocol is in Common.c. It should be noted that some details in the code have not been handled well, such as memory release and closing of Protocol after use.

We'll deal with it when it's commercialized.

### 3 Demonstration

I use the above function to check the protocol of the serial port and the random number generator . After compiling, it is displayed as follows:



```

UEFI Interactive Shell v2.2
EDK II
UEFI v2.70 (EDK II, 0x00010000)
Mapping table
FS0: Alias(s):F2:
VenHw(58C518B1-76F3-11D4-BCEA-0080C73C8881)/VenHw(0C95A935-A006-11D4-B
CFA-0080C73C8881,00000000)
FS1: Alias(s):F3:
VenHw(58C518B1-76F3-11D4-BCEA-0080C73C8881)/VenHw(0C95A935-A006-11D4-B
CFA-0080C73C8881,01000000)
BLK0: Alias(s):
VenHw(58C518B1-76F3-11D4-BCEA-0080C73C8881)/VenHw(0C95A92B-A006-11D4-B
CFA-0080C73C8881,00000000)
BLK1: Alias(s):
VenHw(58C518B1-76F3-11D4-BCEA-0080C73C8881)/VenHw(0C95A92F-A006-11D4-B
CFA-0080C73C8881,01000000)
Press ESC in 1 seconds to skip startup.nsh or any other key to continue.
Shell> fs0:
FS0:\>

```

Figure 4 Program demonstration

In TianCore's simulation environment, there are two serial ports and no protocol for providing a random number generator, as shown in the test results.

There are a lot of examples for other functions on the Internet, but I don't have the energy to try them one by one. Let's move on to the next question: how to generate a Protocol.




Gitee address: <https://gitee.com/luobing4365/uefi-exolorer>

Project code is located in: / FF RobinPkg/RobinPkg/Applications/TryProtocol

探索人工智能前沿，尽在[全球机器学习技术大会云会员](#)

广告

深入探讨大语言模型技术演进、AI 智能代理、ML/LLM Ops 大模型运维、GenAI 产品创新与探索等覆盖人工智能全场景的实践议题，共同探索人工智能的发展前沿。

关于我们 招贤纳士 商务合作 寻求报道  400-660-0108  kefu@csdn.net  在线客服 工作时间 8:30-22:00

公安备案号11010502030143 京ICP备19004658号 京网文〔2020〕1039-165号 经营性网站备案信息 北京互联网违法和不良信息举报中心

家长监护 网络110报警服务 China Internet Reporting Center Chrome Store Download Account Management Specifications

Copyright and Disclaimer Copyright Complaints Publication License Business license

©1999-2025 Beijing Innovation Lezhi Network Technology Co., Ltd.