EFI Basic Tutorial (VI) - Simple Use of PROTOCOL

xiaopangzi313 Modified on 2024-10-30 22:53:51 Read 4.8k Collection 14 Likes 8

Category columns: 15_Firmware Development Article Tags: UEFI FIrmWare BIOS

摘要 This article describes how to use the Protocol mechanism to make UEFI applications easily access DXE driver resources. The specific steps include w ing UEFI application and DXE driver code, compiling to generate EFI files, and running them in a virtual environment. With the help of gMyHelloWorldPEIGUI , the application can locate and call the driver service.

ne summary is generated in C Know , supported by DeepSeek-R1 full version, go to experience>

Write source code

Write UEFI Application code C:\edkii\OvmfPkg\MyHelloWorldAppProtocol\MyHelloWorldAppProtocol.c,

C AI generated projects 登录复制 ru

```c
...
static EFI_MYHELLOWORLD_PROTOCOL * gMyHelloWorldProtocol = NULL;

EFI_STATUS
EFIAPI
MyHelloWorldAppProtocolEntry(
  IN EFI_HANDLE        ImageHandle,
  IN EFI_SYSTEM_TABLE  *SystemTable
)
{
  EFI_STATUS  Status = EFI_SUCCESS;
  DEBUG ((EFI_D_ERROR , "[MyHelloWorldProtocol] MyHelloWorldAppProtocolEntry Start..\n"));
  Print (L"[MyHelloWorldProtocol]  MyHelloWorldAppProtocolEntry Start..\n");

  Status = gBS->LocateProtocol(&gMyHelloWorldPEIGUID,NULL,(VOID **)&gMyHelloWorldProtocol);
  if (EFI_ERROR(Status)){
     Print(L"[MyHelloWorldProtocol] Locate Protocol gMyHelloWorldProtocol %r \n",Status);
     return Status;
  }
  gMyHelloWorldProtocol->PrintMsg(gMyHelloWorldProtocol,L"Hello World App.....\n");
  DEBUG ((EFI_D_ERROR, "[MyHelloWorldProtocol] MyHelloWorldAppProtocolEntry End..\n"));
  Print (L"[MyHelloWorldProtocol] MyHelloWorldAppProtocolEntry  End ... \n");

  return Status;
}
```

收起 ^

Write `UEFI DXE Driver` the code C:\edkii\OvmfPkg\MyHelloWorldDXEProtocol\MyHelloWorldDXEProtocol.c,

C AI generated projects 登录复制 ru

```c
...
static EFI_MYHELLOWORLD_PROTOCOL  gMyHelloWorldProtocol ;

EFI_STATUS
EFIAPI
MyHelloWorldDXEProtocolEntry(
  IN EFI_HANDLE        ImageHandle,
```

```
  IN EFI_SYSTEM_TABLE  *SystemTable
)
{
  EFI_STATUS                              Status = EFI_SUCCESS;
  DEBUG ((EFI_D_ERROR , "[MyHelloWorldProtocol]  MyHelloWorldDXEProtocolEntry Start..\n"));

  gMyHelloWorldProtocol.PrintMsg = PrintHelloWorldMsg;
  gMyHelloWorldProtocol.Revision = 1;

  Status = gBS->InstallProtocolInterface(&ImageHandle,
                                         &gMyHelloWorldPEIGUID,
                                         EFI_NATIVE_INTERFACE,
                                         &gMyHelloWorldProtocol
                                         );
  if (!EFI_ERROR(Status)){
    DEBUG ((EFI_D_ERROR ,"[MyHelloWorldProtocol] MyHelloWorldDXEProtocolEntry Installed Protocol Successfully..\n"))
  }else{
    DEBUG ((EFI_D_ERROR ,"[MyHelloWorldProtocol] MyHelloWorldDXEProtocolEntry Installed Protocol Failly..\n"));
  }

  DEBUG ((EFI_D_ERROR , "[MyHelloWorldProtocol]  MyHelloWorldDXEProtocolEntry End..\n"));
  return Status;
}
```

收起 ^

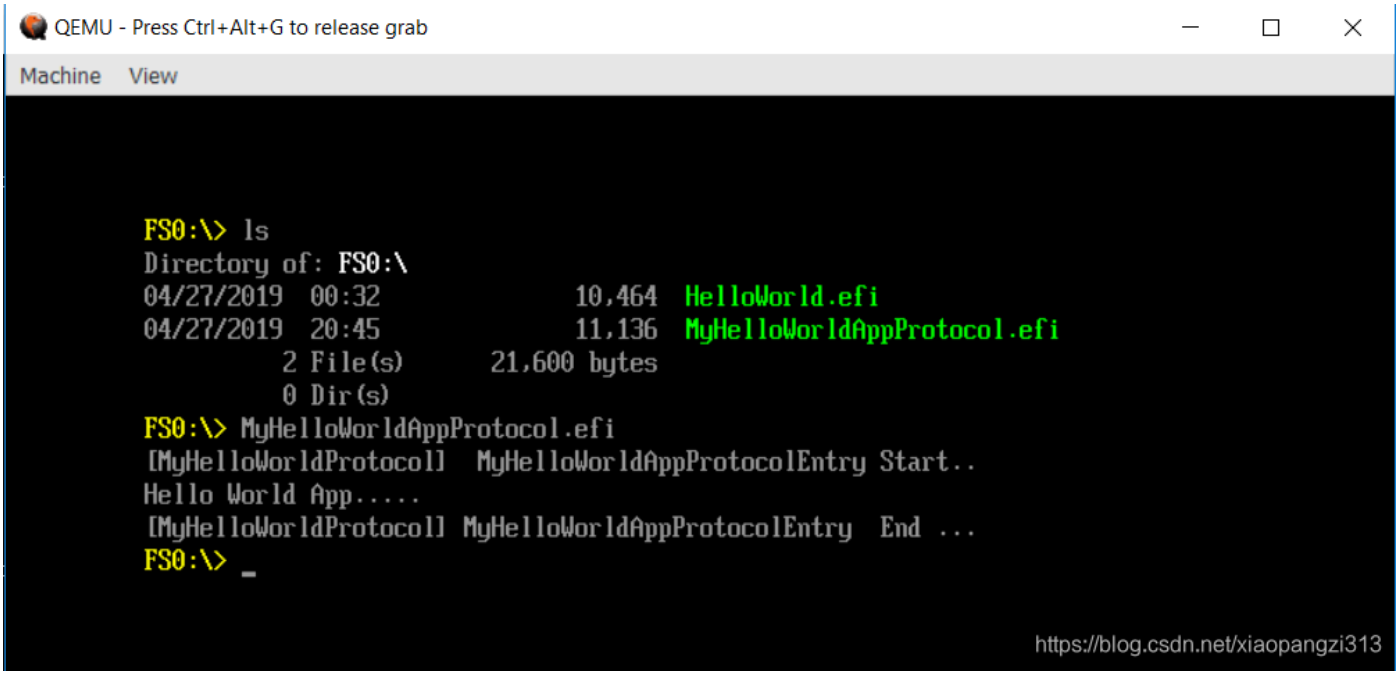
ompile and generate EFI files

and `edksetup.bat` compile the entire OvmfPkg Package

un DXE Driver `MyHelloWorldDXEProtocol` and UEFI APP `MyHelloWorldAppProtocol`

Copy `C:\edkii\Build\OvmfX64\DEBUG_VS2013x86\FV\OVMF.fd` to `C:\qemu` ; Copy `C:\edkii\Build\OvmfX64\DEBUG_VS2013x86\X64\OvmfPkg\MyHelloWorldAppProtocol\MyHelloWorldAppProtocol\OUTPUT\MyHelloWorldAppProtocol.efi` to virtual disk `HDD_BOOT.img`

Execute , and then execute in , the result is as follows, `setup-qemu-x64.bat | findstr MyHelloWorldProtocol` `UEFI SHELL MyHelloWorldAppProtocol.efi`




```
C:\qemu>setup-qemu-x64.bat | findstr MyHelloWorldProtocol
WARNING: Image format was not specified for 'HDD_BOOT.img' and probing guessed raw.
         Automatically detecting the format is dangerous for raw images, write operation
s on block 0 will be restricted.
         Specify the 'raw' format explicitly to remove the restrictions.
[MyHelloWorldProtocol]  MyHelloWorldDXEProtocolEntry Start..
[MyHelloWorldProtocol] MyHelloWorldDXEProtocolEntry Installed Protocol Successfully..
[MyHelloWorldProtocol]  MyHelloWorldDXEProtocolEntry End..
[MyHelloWorldProtocol] MyHelloWorldAppProtocolEntry Start..
[MyHelloWorldProtocol]  MyHelloWorldAppProtocolEntry Start..
[MyHelloWorldProtocol] PrintHelloWorldMsg Hello World App.....
[MyHelloWorldProtocol] MyHelloWorldAppProtocolEntry End..
[MyHelloWorldProtocol] MyHelloWorldAppProtocolEntry  End ...
https://blog.csdn.net/xiaopangzi313
```

Summary

ı the help of Protocol mechanism, the application can easily access the resources of the driver (DXE Driver). In this article, the driver module `lelloWorldDXEProtocol` registers the Protocol service at the program entry `gMyHelloWorldProtocol` , and the service is identified as `'HelloWorldPEIGUID` ; then the application can obtain the service registered in the driver `MyHelloWorldAppProtocol` in its entry function `GUID` , an ı call the function in the service. Among them, there are multiple interfaces for creating Protocol, and you can create a single or multiple Protocols a ɜ. There are also multiple interfaces for obtaining Protocol, such as LocateProtocol, HandleProtocol, and OpenProtocol. The specific differences ca eferred to in the spec.

ıe Dxe/gRT/BDS stage, protocol plays a very important role, such as PCIe , Oprom, SMM, MultiProcessor, Variable, driver model, Smbios, ACPI other subsystems or services. Specifically, if you want to operate a device SATA/NVME/CXL/VGA/Serial port in the UEFI environment, you don't e d to directly care about the library implementation of these domains. You only need to get the corresponding Protocol instance through the guid giv he spec, and then call the corresponding function. On the other hand, the protocol mechanism protects the interests of manufacturers. For example ↄ, Intel, IBV, and OEM release some core drivers in the form of xxx.efi when releasing code. After users get them, they only need to locate the esponding Protocol to complete the access to the device.

hort, the meaning of Protocol is as follows:

Achieve code isolation and remove dependencies on calling libraries

To protect the interests of manufacturers, for example, in the BIOS code released by Intel, ASPPED VGA driver, VMD driver, SATA controller drive PFR driver, and network card driver are all distributed in the form of Oprom.

To realize the linkage of driver events,

1) For example, if you want driver B to execute after driver A, you can add a dependency to driver B as follows,

[depex]

gDriverAProtocol # The gDriverAProtocol is installed in driver A

2) For example, if you want a function in driver B to be triggered by a protocol in driver A

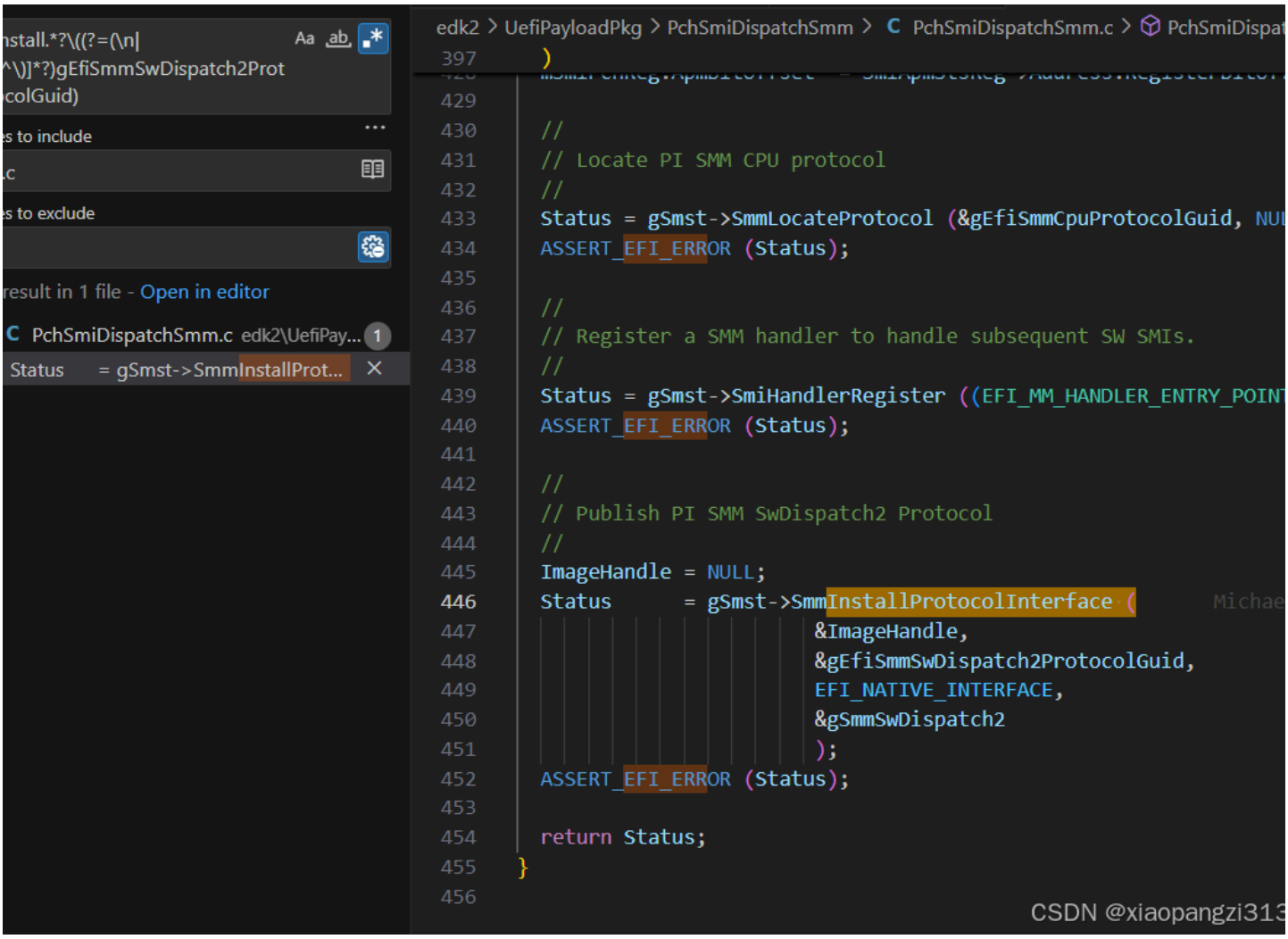
gBS->RegisterProtocolNotify (&gDriverAProtocol, …callback) //Driver B

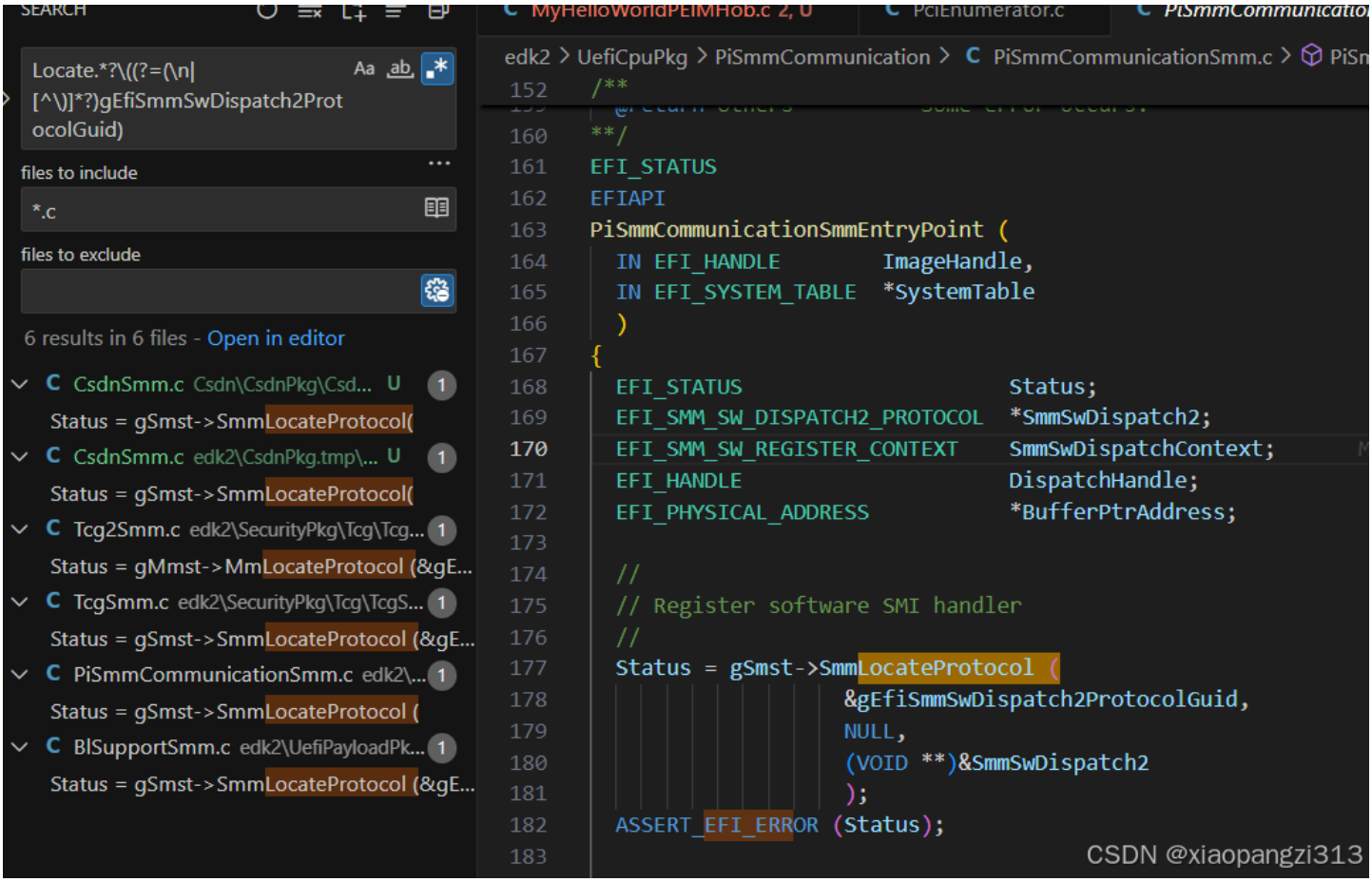
gBS->InstallProtocolInterface (&gDriverAProtocol, xxx) //Driver A, once the protocol is installed, the callback in B will be called

Protocol is widely used in edk2. If you want to quickly find out the usage of the protocol in the current code, you can use the following regex to sea in VSCODE,

ıking gEfiSmmSwDispatch2ProtocolGuid regex search as an example, the following match can quickly search all related Protocol operations from t k


```
.*?\((?=(\n|[^\)]*?)gEfiSmmSwDispatch2ProtocolGuid)          //查询所有的gEfiSmmSwDispatch2ProtocolGuid操作
Install.*?\((?=(\n|[^\)]*?)gEfiSmmSwDispatch2ProtocolGuid)   //查询所有的gEfiSmmSwDispatch2ProtocolGuid Install操作
Locate.*?\((?=(\n|[^\)]*?)gEfiSmmSwDispatch2ProtocolGuid)    //查询所有的gEfiSmmSwDispatch2ProtocolGuid Locate操作
Register.*?\((?=(\n|[^\)]*?)gEfiSmmSwDispatch2ProtocolGuid)  //查询所有的gEfiSmmSwDispatch2ProtocolGuid Register操作
```





e: The protocol is managed by gBS. So once the system boots to the bootloader (grub/lilo) when the ExitBootService Event is triggered, all gBS vices will be discarded (DXE protocol will not be available). If you still want to use the service provided by BIOS, you can only use gRT or gSmm.

EProtocol DEMO source code]