【UEFI Basics】UEFI SCSI Stack Introduction

原创 jiangwei0512 Posted on 2018-02-22 21:23:40 Read 3.2k Collection 7 Likes 1

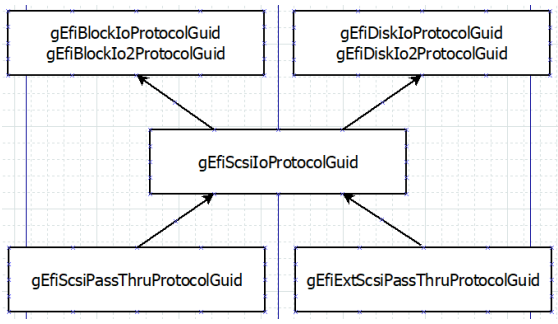
Category Column: UEFI Development Basics Article Tags: uefi

摘要 This article introduces the working principle of the SCSI protocol stack in the UEFI environment and the interaction process between various components, including the functions and roles of SCSIPassThruDriver, SCSIBusDriver and SCSIDeviceDriver. It also explains in detail the content of the SCSI_DISK_DEV structure and the difference between BlockIOProtocol and DiskIOProtocol.

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

EFI SCSI Driver Stack

The SCSI protocol stack under UEFI is roughly as shown below:



The bottom layer involves access to specific hardware devices, and above the top layer are other drivers or applications, which can directly call Block/Disk IO Protocol to complete access to devices such as hard disks.

There are mainly three drivers involved (excerpted from "UEFI Spec"):

> **SCSI Pass Thru Driver:** A SCSI Pass Through Driver manages a SCSI Host Controller that contains one or more SCSI Buses. It creates SCSI Bus Controller Handles for each SCSI Bus, and attaches Extended SCSI Pass Thru Protocol and Device Path Protocol to each handle the driver produced. Please refer to Section 15.7 and Appendix G for details about the Extended SCSI Pass Thru Protocol.
>
> **SCSI Bus Driver:** A SCSI Bus Driver manages a SCSI Bus Controller Handle that is created by SCSI Pass Thru Driver. It creates SCSI Device Handles for each SCSI Device Controller detected during SCSI Bus Enumeration, and attaches SCSI I/O Protocol and Device Path Protocol to each handle the driver produced.
>
> **SCSI Device Driver:** A SCSI Device Driver manages one kind of SCSI Device. Device handles for SCSI Devices are created by SCSI Bus Drivers. A SCSI Device Driver could be a bus driver itself, and may create child handles. But most SCSI Device Drivers will be device drivers that do not create new handles. For the pure device driver, it attaches protocol instance to the device handle of the SCSI Device. These protocol instances are I/O abstractions that allow the SCSI Device to be used in the pre-boot environment. The most common I/O abstractions are used to boot an EFI compliant OS.

Here are some instructions:

1. SCSI Pass Thru Driver is used to control the SCSI Host Controller, which contains one or more SCSI buses.

2. The SCSI Pass Thru Driver will create a Handle (called SCSI Bus Controller Handle) for each SCSI bus. This Handle also has Device Path Protocol, SCSI Pass Thru Protocol and Extended SCSI Pass Thre Protocol.

3. The SCSI Bus Driver processes the SCSI Bus Controller Handle created by the SCSI Pass Thru Driver and creates a new Handle (SCSI Device Handle). This Handle is used to represent the hard disk scanned during the SCSI bus scan (the scanning action is also in this SCSI Bus Driver), and it also has the SCSI I/O Protocol and Device Path Protocol.

4. SCSI Device Driver is used to process a certain type of SCSI Device Handle and add a corresponding SCSI Device instance (SCSI_DISK_DEV in UEFI code) to this Handle. This instance is an I/O abstraction of this type of SCSI Device. The structure is as follows:

```
typedef struct {
  UINT32                    Signature;

  EFI_HANDLE                Handle;

  EFI_BLOCK_IO_PROTOCOL     BlkIo;
  EFI_BLOCK_IO2_PROTOCOL    BlkIo2;
  EFI_BLOCK_IO_MEDIA        BlkIoMedia;
  EFI_ERASE_BLOCK_PROTOCOL  EraseBlock;
  EFI_SCSI_IO_PROTOCOL      *ScsiIo;
  UINT8                     DeviceType;
  BOOLEAN                   FixedDevice;
  UINT16                    Reserved;

  EFI_SCSI_SENSE_DATA       *SenseData;
  UINTN                     SenseDataNumber;
  EFI_SCSI_INQUIRY_DATA     InquiryData;

  EFI_UNICODE_STRING_TABLE  *ControllerNameTable;

  EFI_DISK_INFO_PROTOCOL    DiskInfo;

  //
  // The following fields are only valid for ATAPI/SATA device
  //
  UINT32                    Channel;
  UINT32                    Device;
  ATAPI_IDENTIFY_DATA       IdentifyData;

  //
  // Scsi UNMAP command parameters information
  //
  SCSI_UNMAP_PARAM_INFO     UnmapInfo;
  BOOLEAN                   BlockLimitsVpdSupported;

```

```
  //
  // The flag indicates if 16-byte command can be used
  //
  BOOLEAN                   Cdb16Byte;

  //
  // The queue for asynchronous task requests
  //
  LIST_ENTRY                AsyncTaskQueue;
} SCSI_DISK_DEV;
```

收起 ^

5. Above the above Protocol are Block IO (2) Protocol and Disk IO (2) Protocol, which are interfaces that ordinary drivers or applications will call. Both BlockIo and DiskIo have two versions, one is the ordinary version and the other is the extended version. Here we take the ordinary version as an example. The following are their interfaces:

Block IO Protocol:

cpp AI generated projects 登录复制 run

```cpp
///
///  This protocol provides control over block devices.
///
struct _EFI_BLOCK_IO_PROTOCOL {
  ///
  /// The revision to which the block IO interface adheres. All future
  /// revisions must be backwards compatible. If a future version is not
  /// back wards compatible, it is not the same GUID.
  ///
  UINT64              Revision;
  ///
  /// Pointer to the EFI_BLOCK_IO_MEDIA data for this device.
  ///
  EFI_BLOCK_IO_MEDIA  *Media;

  EFI_BLOCK_RESET     Reset;
  EFI_BLOCK_READ      ReadBlocks;
  EFI_BLOCK_WRITE     WriteBlocks;
  EFI_BLOCK_FLUSH     FlushBlocks;
};
```

收起 ^

Disk IO Protocol:

cpp AI generated projects 登录复制 run

```cpp
///
/// This protocol is used to abstract Block I/O interfaces.
///
struct _EFI_DISK_IO_PROTOCOL {
  ///
  /// The revision to which the disk I/O interface adheres. All future
  /// revisions must be backwards compatible. If a future version is not
  /// backwards compatible, it is not the same GUID.
  ///
  UINT64          Revision;
  EFI_DISK_READ   ReadDisk;
  EFI_DISK_WRITE  WriteDisk;
};
```

收起 ^

The difference between the two is described in the UEFI Spec:

Block IO Protocol:

> This protocol is used to abstract mass storage devices to allow code running in the EFI boot services environment to access them without specific knowledge of the type of device or controller that manages the device. Functions are defined to read and write data at a block level from mass storage devices as well as to manage such devices in the EFI boot services environment.

Disk IO Protocol:

> This protocol is used to abstract the block accesses of the Block I/O protocol to a more general offset-length protocol. The firmware is responsible for adding this protocol to any Block I/O interface that appears in the system that does not already have a Disk I/O protocol. File systems and other disk access code utilize the Disk I/O protocol.

Simply put, Block IO Protocol is at a lower level, and Disk IO Protocol is a further abstraction of Block IO Protocol. If we want to access disk storage devices, we should use Disk IO Protocol.