UEFI Development

UEFI Driver Model, Protocols and Apps

By Elvis M. Teixeira [elvismtt@gmail.com]

UEFI Images

- UEFI applications and drivers are images
- An UEFI image is the compiled code (PEA/COFF) of an app or a driver
- Images can be loaded into memory and unloaded from there (removed)
- A loaded image can be started (The entry point is called)

Drivers VS Applications

Applications

- An application is executed from the beginning of its entry point to its end
- Possibly with side effects (I/O, etc)

Drivers

- A driver exposes a service to be used asynchronously by others.
- Others may be apps, drivers or timer events

Protocols

- Protocols are data structures that contain function pointers
- They can also have data members (e.g. version numbers)
- These pointers should point to the implementation provided by some driver

Example 1: EFI_SIMPLE_FILE_SYSTEM_PROTOCOL

```
struct _EFI_SIMPLE_FILE_SYSTEM_PROTOCOL {
    UINT64          Revision;
    EFI_VOLUME_OPEN OpenVolume;
};


typedef
EFI_STATUS
(EFIAPI *EFI_VOLUME_OPEN) (
    IN EFI_SIMPLE_FILE_SYSTEM_PROTOCOL    * This,
    OUT EFI_FILE                          **Root
);
```

Handles

- The handle database is the most important data structure in the DXE phase
- In each handle there may be any number of protocols and images installed
- A GUID uniquely identifies a resource within a handle
- In a given handle there can be only one resource with a given GUID

The Boot Services Table

Is a set of functions that is globally accessible.

They can be used to:

- Find resources in the handle database
- Load, start and unload images
- Create and start timers
- Many other things

Header UefiBootServicesTableLib.h declares a global pointer `gBS` to this table

Example 2: Using the EFI_SIMPLE_FILE_SYSTEM_PROTOCOL

```
EFI_HANDLE Handle = NULL;
EFI_SIMPLE_FILE_SYSTEM_PROTOCOL *FSProtocol = NULL;
EFI_FILE_PROTOCOL *RootDir = NULL;
EFI_FILE_PROTOCOL *File = NULL;

EFI_STATUS Status = gBS->LocateHandle (
    AllHandles,
    &gEfiSimpleFileSystemProtocol,
    NULL,
    &BufferSize,
    &Handle
    );

Status = gBS->OpenProtocol (
    Handle,
    &gEfiSimpleFileSystemProtocol,
    (VOID **) &FSProtocol,
    ImageHandle,
    NULL,
    EFI_OPEN_PROTOCOL_GET_PROTOCOL
    );
```

```
Status = FSProtocol->OpenVolume (
    FSProtocol,
    &RootDir
    );

Status = RootDir->Open (
    RootDir,
    &File,
    L”FileName.txt”,
    EFI_FILE_MODE_READ,
    EFI_FILE_VALID_ATTR
    );

Status = File->Read (
    File,
    &BufferSize,
    Buffer
    );
```

Driver development

A driver that follows the “UEFI driver model” exposes an entry point,

an unload function (optional but recommended) and installs at least:

- The EFI_DRIVER_BINDING_PROTOCOL
- The EFI_SUPPORTED_EFI_VERSION_PROTOCOL
- The EFI_COMPONENT_NAME_PROTOCOL
- The EFI_COMPONENT_NAME2_PROTOCOL

Installing the protocols

The driver’s entry point:

```
EFI_STATUS
EFIAPI
MyDriverEntry (
    IN EFI_HANDLE ImageHandle,
    IN EFI_SYSTEM_TABLE *SystemTable
)
{
    EFI_STATUS Status = gBS->InstallMultipleProtocolInterfaces (
        &ImageHandle,
        &gEfiDriverSupportedEfiVersionProtocolGuid,
        &gMyDriverSupportedEfiVersion,

        &gEfiDriverBindingProtocolGuid,

        &gMyDriverDriverBinding,

        &gEfiComponentNameProtocolGuid,

        &gMyDriverComponentName,

        &gEfiComponentName2ProtocolGuid,

        &gMyDriverComponentName2,

        NULL

    );


    return Status;

}
```

The EFI_DRIVER_BINDING_PROTOCOL

Contains 3 functions:

Supported():

- Should check if the a handle provides access to a supported device

Start():

- Should install the protocols that make the driver’s services available

Stop():

- Should undo everything Start() does