[UEFI Basics] BIOS module execution priority

原创 jiangwei0512 Posted on 2024-09-15 14:48:06 Read 2.1k Collection 12 Likes 13

Category Column: UEFI Development Basics Article Tags: uefi

Overview

There are two main ways to determine the priority of general modules in BIOS : one is the priority specified in the fdf file, and the other is the priority specified in the inf file. It should be noted that the term "general module" is used here, because some modules (especially PEI_CORE, DXE_CORE type modules) are always executed first. In fact, it is because these priority modules control the priority of general modules.

Priority in fdf

APRIORI

The priorities in fdf are indicated by special identifiers. Here is an example:

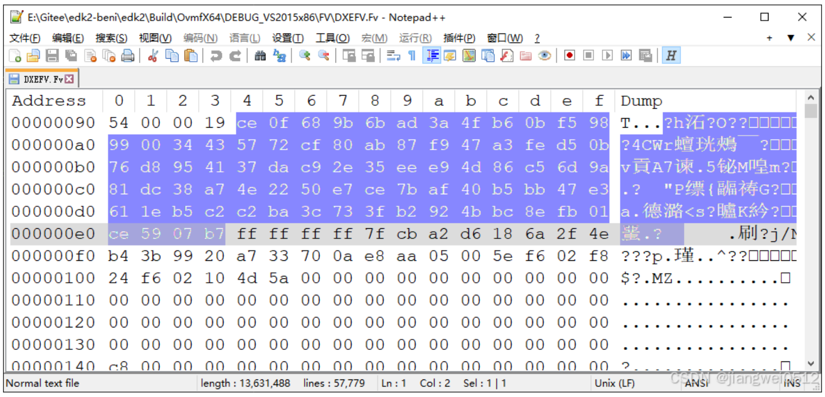
bash AI generated projects 登录复制

```bash
[FV.DXEFV]
# 中间略

APRIORI DXE {
  INF  MdeModulePkg/Universal/DevicePathDxe/DevicePathDxe.inf
  INF  MdeModulePkg/Universal/PCD/Dxe/Pcd.inf
  # AmdSevDxe must be loaded before TdxDxe. Because in SEV guest AmdSevDxe
  # driver performs a MemEncryptSevClearMmioPageEncMask() call against the
  # PcdPciExpressBaseAddress range to mark it shared/unencrypted.
  # Otherwise #VC handler terminates the guest for trying to do MMIO to an
  # encrypted region (Since the range has not been marked shared/unencrypted).
  INF  OvmfPkg/AmdSevDxe/AmdSevDxe.inf
  INF  OvmfPkg/TdxDxe/TdxDxe.inf
!if $(SMM_REQUIRE) == FALSE
  INF  OvmfPkg/QemuFlashFvbServicesRuntimeDxe/FvbServicesRuntimeDxe.inf
!endif
}
```

Here `APRIORI` we specify the modules that need to be executed first.

During compilation, this part will be composed into a Firmware File. In the above example, this Firmware File can be found from DXEFV. The following is the actual data in the file:



E:\Gitee\edk2-beni\edk2\Build\OvmfX64\DEBUG_VS2015x86\FV\DXEFV.Fv - Notepad++

文件(F) 编辑(E) 搜索(S) 视图(V) 编码(N) 语言(L) 设置(T) 工具(O) 宏(M) 运行(R) 插件(P) 窗口(W) ?
DXEFV.Fv

```
Address   0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f  Dump
00000090 54 00 00 19 ce 0f 68 9b 6b ad 3a 4f b6 0b f5 98 T...?h洉?O??□□□□□
000000a0 99 00 34 43 57 72 cf 80 ab 87 f9 47 a3 fe d5 0b ?4CWr蟺珖鴂  ?□□□
000000b0 76 d8 95 41 37 da c9 2e 35 ee e9 4d 86 c5 6d 9a v貢A7谏.5铋M喤m?□
000000c0 81 dc 38 a7 4e 22 50 e7 ce 7b af 40 b5 bb 47 e3 .?  "P缥{疈祷G?□□
000000d0 61 1e b5 c2 c2 ba 3c 73 3f b2 92 4b bc 8e fb 01 a.德潞<s?矑K紟?□□
000000e0 ce 59 07 b7 ff ff ff ff 7f cb a2 d6 18 6a 2f 4e 蝵.?      .刷?j/N
000000f0 b4 3b 99 20 a7 33 70 0a e8 aa 05 00 5e f6 02 f8 ???p.瑾..^??□□□□□
00000100 24 f6 02 10 4d 5a 00 00 00 00 00 00 00 00 00 00 $?.MZ..........□
00000110 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00000120 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00000130 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00000140 c8 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ?..............□
```

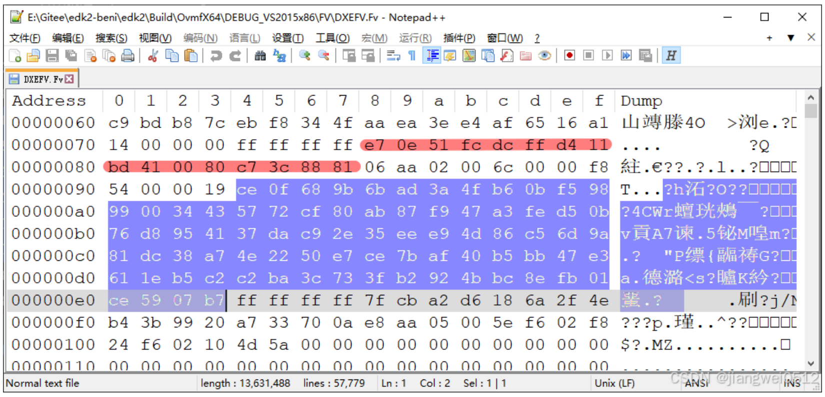
Normal text file length : 13,631,488 lines : 57,779 Ln : 1 Col : 2 Sel : 1 | 1 Unix (LF) ANSI INS

These data are actually GUIDs from the inf file in the included module `FILE_GUID` :

bash AI generated projects 登录复制

```bash
[Defines]
  INF_VERSION                    = 0x00010005
  BASE_NAME                      = DevicePathDxe
  MODULE_UNI_FILE                = DevicePathDxe.uni
  FILE_GUID                      = 9B680FCE-AD6B-4F3A-B60B-F59899003443 # Firmware File中包含的GUID
  MODULE_TYPE                    = DXE_DRIVER
  VERSION_STRING                 = 1.0
  ENTRY_POINT                    = DevicePathEntryPoint
```

And this Firmware File itself also has a GUID:



E:\Gitee\edk2-beni\edk2\Build\OvmfX64\DEBUG_VS2015x86\FV\DXEFV.Fv - Notepad++

文件(F) 编辑(E) 搜索(S) 视图(V) 编码(N) 语言(L) 设置(T) 工具(O) 宏(M) 运行(R) 插件(P) 窗口(W) ?
DXEFV.Fv

```
Address   0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f  Dump
00000060 c9 bd b8 7c eb f8 34 4f aa ea 3e e4 af 65 16 a1 山竱膝4O  >浏e.?□
00000070 14 00 00 00 ff ff ff ff e7 0e 51 fc dc ff d4 11 ....          ?Q
00000080 bd 41 00 80 c7 3c 88 81 06 aa 02 00 6c 00 00 f8 紏.€??.?.l..?□□□□
00000090 54 00 00 19 ce 0f 68 9b 6b ad 3a 4f b6 0b f5 98 T...?h洉?O??□□□□□
000000a0 99 00 34 43 57 72 cf 80 ab 87 f9 47 a3 fe d5 0b ?4CWr蟺珖鴂  ?□□□
000000b0 76 d8 95 41 37 da c9 2e 35 ee e9 4d 86 c5 6d 9a v貢A7谏.5铋M喤m?□
000000c0 81 dc 38 a7 4e 22 50 e7 ce 7b af 40 b5 bb 47 e3 .?  "P缥{疈祷G?□□
000000d0 61 1e b5 c2 c2 ba 3c 73 3f b2 92 4b bc 8e fb 01 a.德潞<s?矑K紟?□□
000000e0 ce 59 07 b7 ff ff ff ff 7f cb a2 d6 18 6a 2f 4e 蝵.?      .刷?j/N
000000f0 b4 3b 99 20 a7 33 70 0a e8 aa 05 00 5e f6 02 f8 ???p.瑾..^??□□□□□
00000100 24 f6 02 10 4d 5a 00 00 00 00 00 00 00 00 00 00 $?.MZ..........□
00000110 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
```

Normal text file length : 13,631,488 lines : 57,779 Ln : 1 Col : 2 Sel : 1 | 1 Unix (LF) ANSI INS

This GUID is actually fixed and is defined in MdePkg\Include\Guid\Apriori.h:

c AI generated projects 登录复制 run

```c
#define EFI_APRIORI_GUID \
  { \
    0xfc510ee7, 0xffdc, 0x11d4, {0xbd, 0x41, 0x0, 0x80, 0xc7, 0x3c, 0x88, 0x81 } \

  }
extern EFI_GUID gAprioriGuid;
```

This `gAprioriGuid` will be used further in the code to get `APRIORI` the GUIDs from the above mentioned files to determine which modules need to be executed first.

Code handling gAprioriGuid

The relevant code can be found in edk2\MdeModulePkg\Core\Dxe\DxeMain.inf `CoreDispatcher()` :

c AI generated projects 登录复制 run

```c
    //
    // Read the array of GUIDs from the Apriori file if it is present in the firmware volume
    //
    AprioriFile = NULL;
    Status      = Fv->ReadSection (
                        Fv,
                        &gAprioriGuid,
                        EFI_SECTION_RAW,
                        0,
                        (VOID **)&AprioriFile,
                        &SizeOfBuffer,
                        &AuthenticationStatus
                        );
    if (!EFI_ERROR (Status)) {
      AprioriEntryCount = SizeOfBuffer / sizeof (EFI_GUID);
    } else {
      AprioriEntryCount = 0;
    }

    //
    // Put drivers on Apriori List on the Scheduled queue. The Discovered List includes
    // drivers not in the current FV and these must be skipped since the a priori list
    // is only valid for the FV that it resided in.
    //

    for (Index = 0; Index < AprioriEntryCount; Index++) {
      for (Link = mDiscoveredList.ForwardLink; Link != &mDiscoveredList; Link = Link->ForwardLink) {
        DriverEntry = CR (Link, EFI_CORE_DRIVER_ENTRY, Link, EFI_CORE_DRIVER_ENTRY_SIGNATURE);
        if (CompareGuid (&DriverEntry->FileName, &AprioriFile[Index]) &&
            (FvHandle == DriverEntry->FvHandle))
        {
          CoreAcquireDispatcherLock ();
          DriverEntry->Dependent = FALSE;
          DriverEntry->Scheduled = TRUE;
          InsertTailList (&mScheduledQueue, &DriverEntry->ScheduledLink);
          CoreReleaseDispatcherLock ();
          DEBUG ((DEBUG_DISPATCH, "Evaluate DXE DEPEX for FFS(%g)\n", &DriverEntry->FileName));
          DEBUG ((DEBUG_DISPATCH, "  RESULT = TRUE (Apriori)\n"));
          break;
        }
      }
    }
```

The code is also very simple:

- Get the GUID.
- Iterate over the GUIDs.
- Traverse all modules found and match them with the specified GUID. If a match is found, put it in `mScheduledQueue` .

The above is just the first step, that is, to store the priority module. The header of the edk2\MdeModulePkg\Core\Dxe\Dispatcher\Dispatcher.c file corresponds to the following description:

> Step #1 - When a FV protocol is added to the system every driver in the FV
> is added to the mDiscoveredList. The SOR, Before, and After Depex are
> pre-processed as drivers are added to the mDiscoveredList. If an Apriori
> file exists in the FV those drivers are added to the
> mScheduledQueue. The mFvHandleList is used to make sure a
> FV is only processed once.

Mainly this sentence:

> If an Apriori file exists in the FV those drivers are added to the mScheduledQueue.

At execution time:

c AI generated projects 登录复制 run

```c
EFI_STATUS
EFIAPI
CoreDispatcher (
  VOID
  )
{
  // 其它次要代码已经略去
  do {
    //
    // Drain the Scheduled Queue
    //
    while (!IsListEmpty (&mScheduledQueue)) {
      // 获取模块
      DriverEntry = CR (
                      mScheduledQueue.ForwardLink,
                      EFI_CORE_DRIVER_ENTRY,
                      ScheduledLink,
                      EFI_CORE_DRIVER_ENTRY_SIGNATURE
                      );
      // 加载模块
      Status = CoreLoadImage (
                  FALSE,
                  gDxeCoreImageHandle,
                  DriverEntry->FvFileDevicePath,
                  NULL,
                  0,
                  &DriverEntry->ImageHandle
                  );
      // 执行之后移除魔魁啊
      DriverEntry->Scheduled   = FALSE;
      DriverEntry->Initialized = TRUE;
      RemoveEntryList (&DriverEntry->ScheduledLink);
      if (DriverEntry->IsFvImage) {
        //

        // Produce a firmware volume block protocol for FvImage so it gets dispatched from.
        //
        Status = CoreProcessFvImageFile (DriverEntry->Fv, DriverEntry->FvHandle, &DriverEntry->FileName);
      } else {
        // 执行模块
        Status = CoreStartImage (DriverEntry->ImageHandle, NULL, NULL);
      }

      ReturnStatus = EFI_SUCCESS;
    }
  } while (ReadyToRun);
```

There are two loops here. The second while loop is `mScheduledQueue` the module that is executed first. The corresponding description of the header of the edk2\MdeModulePkg\Core\Dxe\Dispatcher\Dispatcher.c file is:

> Step #2 - Dispatch. Remove driver from the mScheduledQueue and load and
> start it. After mScheduledQueue is drained check the
> mDiscoveredList to see if any item has a Depex that is ready to be
> placed on the mScheduledQueue.

Mainly corresponds to the first sentence:

> Dispatch. Remove driver from the mScheduledQueue and load and start it.

Priority in inf

Not all modules can contain dependencies. Some modules' dependencies will be ignored even if they are written. The following description is given in edk-ii-inf-specification.pdf:

> - If the Module is a Library, then a [Depex] section is optional.
>   If the Module is a Library with a MODULE_TYPE of BASE, the generic (ie, [Depex]) and generic with only architectural modifier entries (ie, [Depex.IA32]) are not permitted. It is permitted to have a Depex section if one ModuleType modifier is specified (ie, [Depex.common.PEIM).
> - If the ModuleType is USER_DEFINED , then a [Depex] section is optional. If a PEI, SMM or DXE DEPEX section is required, the user must specify a ModuleType of PEIM to generate a PEI_DEPEX section, a ModuleType of DXE_DRIVER to generate a DXE_DEPEX section, or a ModuleType of DXE_SMM_DRIVER to generate an SMM_DEPEX section.
> - If the ModuleType is SEC, UEFI_APPLICATION, UEFI_DRIVER, PEI_CORE, SMM_CORE or DXE_CORE, no [Depex] sections are permitted and all library class [Depex] sections are ignored.
> - Module types PEIM, DXE_DRIVER, DXE_RUNTIME_DRIVER, DXE_SMM_DRIVER require a DXE_SAL_DRIVER and [Depex] section unless the dependencies are specified by a PEI_DEPEX , DXE_DEPEX or SMM_DEPEX in the [Binaries] section.

Generate depex file

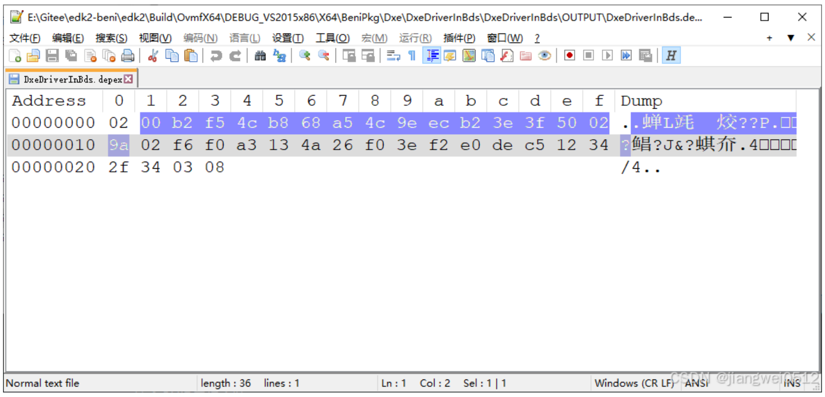
There is a Section in the inf that contains dependencies. Here is an example (from beni\BeniPkg\Dxe\DxeDriverInBds\DxeDriverInBds.inf):

bash   AI generated projects   登录复制

```bash
[Depex]
  gEfiPciIoProtocolGuid
```

In other words, to execute this module, the prerequisite is that `gEfiPciIoProtocolGuid` this GUID has been installed.

When compiling a module, a specific file is generated (through edk2\BaseTools\Source\Python\AutoGen\GenDepex.py, which is also part of AutoGen) with the name format "module name.depex", in this case DxeDriverInBds.depex:



E:\Gitee\edk2-beni\edk2\Build\OvmfX64\DEBUG_VS2015x86\X64\BeniPkg\Dxe\DxeDriverInBds\DxeDriverInBds\OUTPUT\DxeDriverInBds.de...

文件(F) 编辑(E) 搜索(S) 视图(V) 编码(N) 语言(L) 设置(T) 工具(O) 宏(M) 运行(R) 插件(P) 窗口(W) ?
DxeDriverInBds.depex

```
Address   0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f  Dump
00000000 02 00 b2 f5 4c b8 68 a5 4c 9e ec b2 3e 3f 50 02 ..蝉L琵  烄??P.□□
00000010 9a 02 f6 f0 a3 13 4a 26 f0 3e f2 e0 de c5 12 34 ?鲳?J&?蜞夰.4□□□□
00000020 2f 34 03 08                                     /4..
```

Normal text file   length : 36   lines : 1   Ln : 1   Col : 2   Sel : 1 | 1   Windows (CR LF)   ANSI   INS

The GUID is highlighted in the image above `gEfiPciIoProtocolGuid` . However, there are a few points to note:

- First, there is a 02 in front of the GUID, which represents the OPCODE. More OPCODEs can be seen in edk2\BaseTools\Source\Python\AutoGen\GenDepex.py:

Python   AI generated projects   登录复制   run

```python
    Opcode = {
        "PEI"   : {
            DEPEX_OPCODE_PUSH  :   0x02,
            DEPEX_OPCODE_AND   :   0x03,
            DEPEX_OPCODE_OR    :   0x04,
            DEPEX_OPCODE_NOT   :   0x05,
            DEPEX_OPCODE_TRUE  :   0x06,
            DEPEX_OPCODE_FALSE :   0x07,
            DEPEX_OPCODE_END   :   0x08
        },

        "DXE"   : {
            DEPEX_OPCODE_BEFORE:   0x00,
            DEPEX_OPCODE_AFTER :   0x01,
            DEPEX_OPCODE_PUSH  :   0x02,
            DEPEX_OPCODE_AND   :   0x03,
            DEPEX_OPCODE_OR    :   0x04,
            DEPEX_OPCODE_NOT   :   0x05,
            DEPEX_OPCODE_TRUE  :   0x06,
            DEPEX_OPCODE_FALSE :   0x07,
            DEPEX_OPCODE_END   :   0x08,
            DEPEX_OPCODE_SOR   :   0x09
        },
```

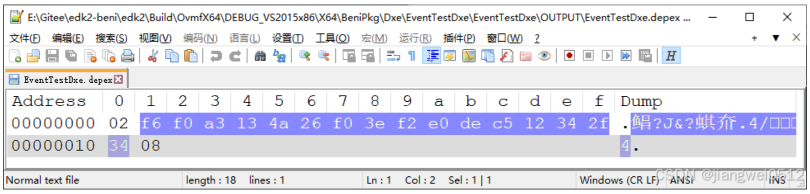
02 means yes `DEPEX_OPCODE_PUSH` , 03 means yes `DEPEX_OPCODE_AND` , and 08 means yes `DEPEX_OPCODE_END` .

- There is also a second GUID corresponding to `gEfiPcdProtocolGuid` :

bash   AI generated projects   登录复制

```bash
  ## Include/Protocol/PiPcd.h
  gEfiPcdProtocolGuid = { 0x13a3f0f6, 0x264a, 0x3ef0, { 0xf2, 0xe0, 0xde, 0xc5, 0x12, 0x34, 0x2f, 0x34 } }
```

Not sure why you are including this GUID, and `TRUE` also have this GUID when under [Depex] there is only:



E:\Gitee\edk2-beni\edk2\Build\OvmfX64\DEBUG_VS2015x86\X64\BeniPkg\Dxe\EventTestDxe\EventTestDxe\OUTPUT\EventTestDxe.depex ...

EventTestDxe.depex

```
Address   0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f  Dump
00000000 02 f6 f0 a3 13 4a 26 f0 3e f2 e0 de c5 12 34 2f .鲳?J&?蜞疥.4/□□□
00000010 34 08                                           4.
```

Normal text file   length : 18   lines : 1   Ln : 1   Col : 2   Sel : 1 | 1   Windows (CR LF)   ANSI   INS

文件(F) 编辑(E) 搜索(S) 视图(V) 编码(N) 语言(L) 设置(T) 工具(O) 宏(M) 运行(R) 插件(P) 窗口(W) ?
CSDN @jiangwei0512

Another point worth noting is that the PCD module will also be included in the fdf file `APRIORI` , because PCD is a basic mode. In order for all modules to support PCD, it is understandable to have such a dependency.

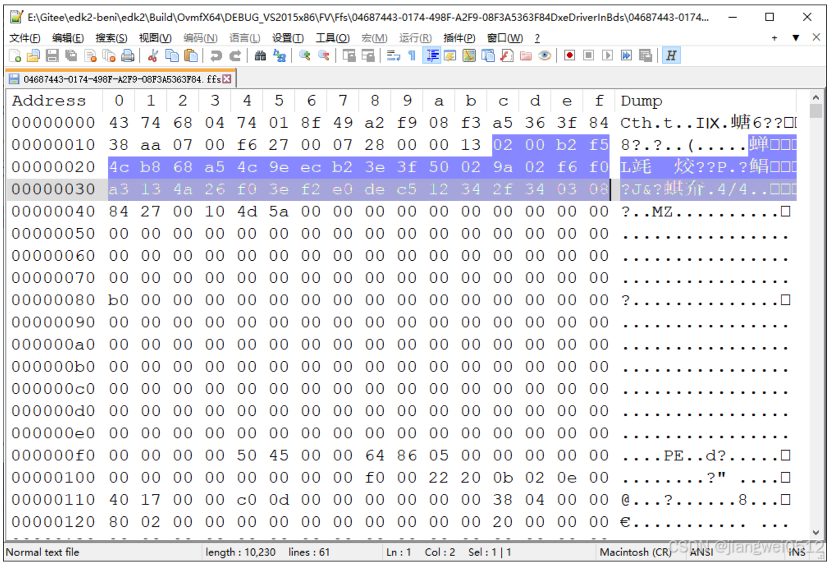
Include the depex file into the BIOS binary

By looking at fdf, you can know how depex is included, mainly through Rules [Section]. For example, a DXE_DRIVER generates a ffs file structure that follows the following Rules:

bash AI generated projects 登录复制

```bash
[Rule.Common.DXE_DRIVER]
  FILE DRIVER = $(NAMED_GUID) {
    DXE_DEPEX    DXE_DEPEX Optional      $(INF_OUTPUT)/$(MODULE_NAME).depex
    PE32     PE32                    $(INF_OUTPUT)/$(MODULE_NAME).efi
    UI       STRING="$(MODULE_NAME)" Optional
    VERSION  STRING="$(INF_VERSION)" Optional BUILD_NUM=$(BUILD_NUMBER)
    RAW ACPI  Optional               |.acpi
    RAW ASL   Optional               |.aml
  }
```

That is, first a depex file, then an efi file, like this:



E:\Gitee\edk2-beni\edk2\Build\OvmfX64\DEBUG_VS2015x86\FV\Ffs\04687443-0174-498F-A2F9-08F3A5363F84DxeDriverInBds\04687443-0174...

04687443-0174-498F-A2F9-08F3A5363F84.ffs

```
Address   0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f  Dump
00000000 43 74 68 04 74 01 8f 49 a2 f9 08 f3 a5 36 3f 84 Cth.t..IIX.螗6??□
00000010 38 aa 07 00 f6 27 00 07 28 00 00 13 02 00 b2 f5 8?.?..(.....蝉□□□
00000020 4c b8 68 a5 4c 9e ec b2 3e 3f 50 02 9a 02 f6 f0 L瓱  烄??P.?鲳□□□
00000030 a3 13 4a 26 f0 3e f2 e0 de c5 12 34 2f 34 03 08 ?J&?蜞疥.4/4..□□□
00000040 84 27 00 10 4d 5a 00 00 00 00 00 00 00 00 00 00 ?..MZ..........□
00000050 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00000060 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00000070 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00000080 b0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ?..............□
00000090 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
000000a0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
000000b0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
000000c0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
000000d0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
000000e0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
000000f0 00 00 00 00 50 45 00 00 64 86 05 00 00 00 00 00 ....PE..d?.....□
00000100 00 00 00 00 00 00 00 00 f0 00 22 20 0b 02 0e 00 ........?" ....□
00000110 40 17 00 00 c0 0d 00 00 00 00 00 00 38 04 00 00 @...?.......8...□
00000120 80 02 00 00 00 00 00 00 00 00 00 00 20 00 00 00 €........... ...
```

Normal text file   length : 10,230   lines : 61   Ln : 1   Col : 2   Sel : 1 | 1   Macintosh (CR)   ANSI   INS

文件(F) 编辑(E) 搜索(S) 视图(V) 编码(N) 语言(L) 设置(T) 工具(O) 宏(M) 运行(R) 插件(P) 窗口(W) ?
CSDN @jiangwei0512

Code Processing

There is no specific GUID (image `gAprioriGuid`) in the module to specify the dependency, but depex is originally part of ffs, so it can be read out. There is a member in a structure describing the module to represent this depex (taking DXE as an example):

c AI generated projects 登录复制 run

```c
typedef struct {
  // 其它略
  VOID                              *Depex;       // 描述依赖关系
  UINTN                             DepexSize;    // depex的大小
} EFI_CORE_DRIVER_ENTRY;
```

Therefore, when we get the module, we can already know the GUID it depends on, and these modules `mDiscoveredList` form a linked list through a global variable, and various operations will be performed later by traversing this linked list.

The code that actually handles dependencies is also in `CoreDispatcher()` :

c AI generated projects 登录复制 run

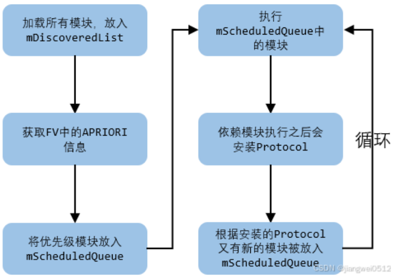
```c
EFI_STATUS
EFIAPI
CoreDispatcher (
  VOID
  )
{
  // 其它次要代码已经略去
  do {
    //
    // Drain the Scheduled Queue
    //
    while (!IsListEmpty (&mScheduledQueue)) {
      // 首次执行的时候，执行fdf中的优先模块
      // 后面的操作又会往mScheduledQueue里面放更多的模块，又会继续执行
    }
    //
    // Search DriverList for items to place on Scheduled Queue
    //
    ReadyToRun = FALSE;
    for (Link = mDiscoveredList.ForwardLink; Link != &mDiscoveredList; Link = Link->ForwardLink) {
      DriverEntry = CR (Link, EFI_CORE_DRIVER_ENTRY, Link, EFI_CORE_DRIVER_ENTRY_SIGNATURE);

      if (DriverEntry->DepexProtocolError) {
        //
        // If Section Extraction Protocol did not let the Depex be read before retry the read
        //
        // 会将满足依赖的模块继续放入mScheduledQueue
        Status = CoreGetDepexSectionAndPreProccess (DriverEntry);
      }

      if (DriverEntry->Dependent) {
        if (CoreIsSchedulable (DriverEntry)) {
```

```
     CoreInsertOnScheduledQueueWhileProcessingBeforeAndAfter (DriverEntry);
     ReadyToRun = TRUE;
    }
   }
  }
 } while (ReadyToRun);
```

In general, dependencies are handled as follows:


加载所有模块，放入mDiscoveredList
获取FV中的APRIORI信息
将优先级模块放入mScheduledQueue
执行mScheduledQueue中的模块
依赖模块执行之后会安装Protocol
根据安装的Protocol又有新的模块被放入mScheduledQueue
循环
CSDN @jiangwei0512

other

At the beginning, I mentioned that there are two main types of dependencies. In fact, there are some derivative methods. For example, the priority in the inf file mentions the depex file, which can be used in different ways. You can generate a depex file by including [Depex] in the inf file, or you can generate it manually (through edk2\BaseTools\Source\Python\AutoGen\GenDepex.py) and then put it directly into the fdf to specify the dependency for a file. This is very useful when you directly include the efi file in the fdf. Here is an example:

bash AI generated projects 登录复制

```
FILE DRIVER = 5BBA83E5-F027-4ca7-BFD0-16358CC9E123 {
    SECTION PE32 = $(PLATFORM_FEATURES_PATH)/Icc/IccOverClocking/IccOverClocking.efi
    SECTION DXE_DEPEX = $(PLATFORM_FEATURES_PATH)/Icc/IccOverClocking/IccOverClocking.depex
    SECTION UI = "IccOverClocking"
  }
```