[UEFI Practice] OsLoader Code Analysis in SBL

原创 jiangwei0512 Posted on 2021-12-07 22:35:10 Read 1.2k Collection 2 Likes

Category Column: UEFI Development Basics Article Tags: slimbootloader uefi bios

Entrance

The corresponding module is PayloadPkg\OsLoader\OsLoader.inf, and the declaration in BootloaderCorePkg\BootloaderCorePkg.dsc is:

bash AI generated projects 登录复制

```bash
PayloadPkg/OsLoader/OsLoader.inf {
    <PcdsFixedAtBuild>
        gPlatformCommonLibTokenSpaceGuid.PcdDebugOutputDeviceMask  | $(DEBUG_OUTPUT_DEVICE_MASK)
    <LibraryClasses>
        MemoryAllocationLib | BootloaderCommonPkg/Library/FullMemoryAllocationLib/FullMemoryAllocationLib.inf
        PayloadEntryLib     | PayloadPkg/Library/PayloadEntryLib/PayloadEntryLib.inf
        PayloadSupportLib   | PayloadPkg/Library/PayloadSupportLib/PayloadSupportLib.inf
        BootloaderLib       | PayloadPkg/Library/PayloadLib/PayloadLib.inf
        PlatformHookLib     | PayloadPkg/Library/PlatformHookLib/PlatformHookLib.inf
        AbSupportLib        | PayloadPkg/Library/AbSupportLib/AbSupportLib.inf
        SblParameterLib     | PayloadPkg/Library/SblParameterLib/SblParameterLib.inf
        TrustyBootLib       | PayloadPkg/Library/TrustyBootLib/TrustyBootLib.inf
}
```

收起

The implementation of OsLoader.inf:

bash AI generated projects 登录复制

```bash
[Defines]
  INF_VERSION                    = 0x00010005
  BASE_NAME                      = OsLoader
  FILE_GUID                      = A257AA67-53F3-491B-8CFF-E9A4E2E2A514
  MODULE_TYPE                    = PEIM
  VERSION_STRING                 = 1.0

#
#  This flag specifies whether HII resource section is generated into PE image.
#
  UEFI_HII_RESOURCE_SECTION      = TRUE

#
# The following information is for reference only and not required by the build tools.
#
#  VALID_ARCHITECTURES           = IA32 X64 IPF EBC
#

[Sources]
  BlockIoTest.h
  OsLoader.h
  OsLoader.c
  BootOption.c
  BootConfig.c
  LoadImage.c
  PerformanceData.c
  BootParameters.c
  BlockIoTest.c
  KeyManagement.c
  PreOsSupport.c
  ModService.c
  ExtraModSupport.c
```

收起

What's strange is that there is no entry, but it seems that none of the SBL modules have an entry.

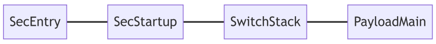
The entry of OsLoader can be found through the code:

c AI generated projects 登录复制 run

```c
/**
  Payload main entry.

  This function will continue Payload execution with a new memory based stack.

  @param  Param           parameter passed from SwitchStack().
  @param  PldBase         payload base passed from SwitchStack().

**/
VOID
EFIAPI
PayloadMain (
  IN  VOID             *Param,
  IN  VOID             *PldBase
  )
```

收起

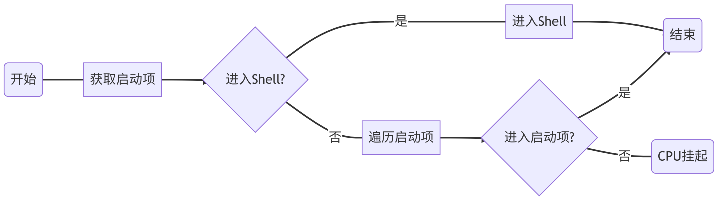
In fact, the module entry of SBL follows the following calling logic:



The first one `SecEntry` comes from the general Lib: `ModuleEntryLib` .

process

The basic process of OsLoader:



Eventually it will hang if no startup item is found .

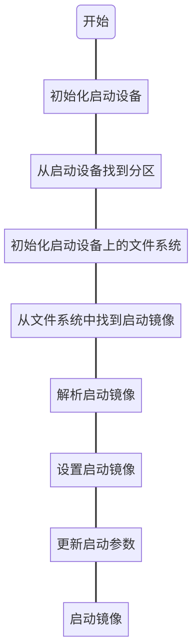
遍历启动项 This is done with a while loop:

c AI generated projects 登录复制 run

```c
while  (BootIdx < OsBootOptionList->OsBootOptionCount) {
  mCurrentBoot = CurrIdx;
  DEBUG ((DEBUG_INFO, "\n======== Try Booting with Boot Option %d ========\n", CurrIdx));

  // Get current boot option and try boot
  CopyMem ((VOID *)&OsBootOption, (VOID *)&OsBootOptionList->OsBootOption[CurrIdx], sizeof (OS_BOOT_OPTION));
  BootOsImage (&OsBootOption);

  // De-init the current boot devices
  // If USB keyboard console is used, don't DeInit USB yet at this moment.
  // It will be handled just before transfering to OS.
  if (!((OsBootOption.DevType == OsBootDeviceUsb) &&
      ((PcdGet32 (PcdConsoleInDeviceMask) & ConsoleInUsbKeyboard) != 0))) {
    MediaInitialize (0, DevDeinit);
  }

  if (OsBootOptionList->RestrictedBoot != 0) {
    // Restricted boot should not try other boot option
    break;
  } else {
    // Move to next boot option
    CurrIdx = GetNextBootOption (OsBootOptionList, CurrIdx);
    if (CurrIdx >= OsBootOptionList->OsBootOptionCount) {
      CurrIdx = 0;
    }
    BootIdx++;
  }
}
```

收起

The most important function here is `BootOsImage()` , its process is as follows:



If an error occurs during execution, it will exit to execute the next startup item.

It should be noted here that, unlike UEFI, SBL first has a startup item, then initializes the corresponding device, and searches for the startup image from the device, and starts if there is one.

The following are introduced according to the key steps:

1. Initialize the boot device. Currently supported devices are defined in BootloaderCommonPkg\Include\Guid\OsBootOptionGuid.h:

c AI generated projects 登录复制 run

```c
// Define OS boot media devices
typedef enum {
  OsBootDeviceSata,
  OsBootDeviceSd,
  OsBootDeviceEmmc,
  OsBootDeviceUfs,
  OsBootDeviceSpi,
  OsBootDeviceUsb,
  OsBootDeviceNvme,
  OsBootDeviceMemory,
  OsBootDeviceMax
} OS_BOOT_MEDIUM_TYPE;
```

收起

The relevant initialization interface will be assigned values in the code, and then the actual initialization will be performed `MediaSetInterfaceType()` later . `MediaInitialize()`

2. Find the partition. SBL, like UEFI, also supports GPT and MBR partitions. After the device is initialized, it finds the partition from the device `FindGptPartitions()` and uses `FindMbrPartitions()` two functions to find the corresponding partition.
3. The next step is to find the file system from the partition. The supported file systems can also be found in BootloaderCommonPkg\Include\Guid\OsBootOptionGuid.h:

c AI generated projects 登录复制 run

```c
typedef enum  {
  EnumFileSystemTypeFat,
  EnumFileSystemTypeExt2,
  EnumFileSystemTypeAuto,
  EnumFileSystemMax
} OS_FILE_SYSTEM_TYPE;
```

In fact, it is FAT and EXT2 file systems.

4. After finding the file system, the boot image can be read and loaded on the file system, and finally the image can be started.