[UEFI Practice] Support more file systems under BIOS

原创 jiangwei0512 Posted on 2025-02-23 15:38:42 Views: 663 Collection 9 Likes 6

Copyright CC 4.0 BY-SA

Category Column: UEFI Development Basics Article Tags: uefi File System

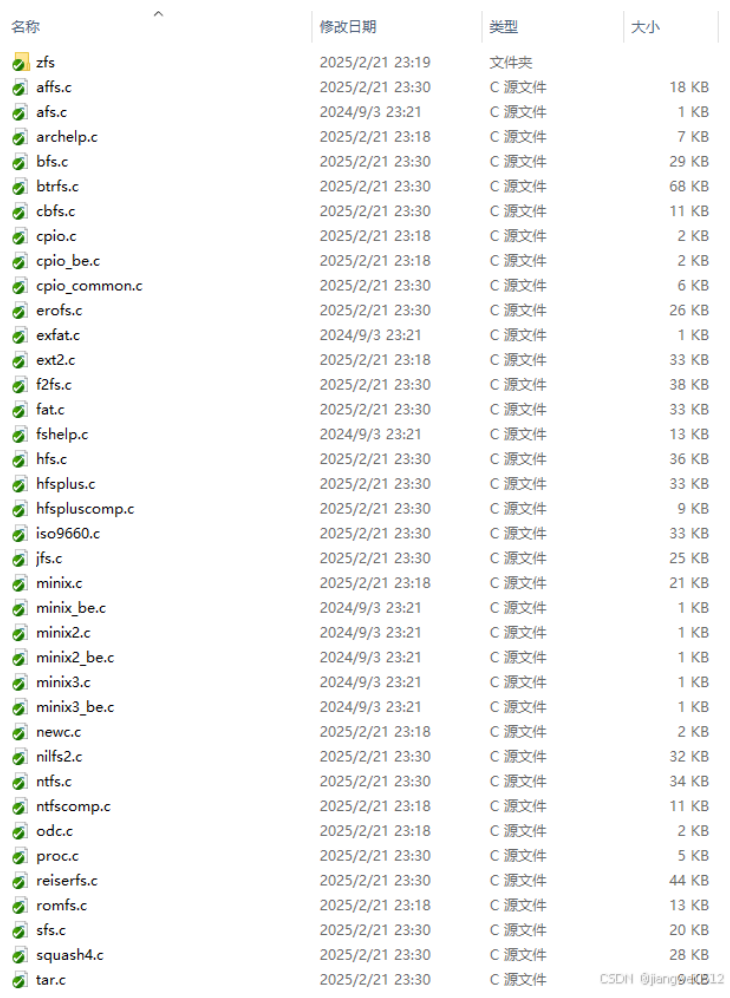
UEFI Development ... This column includes this content 136 articles Subscribe to our column

Overview

The open source EDK code only supports the FAT32 file system, including read and write operations.

In actual use, there are many file systems that need to be supported , such as NTFS, EXT4, etc. Usually BIOS does not need to support write operations, but it is best to support general read operations.

GRUB is a BootLoader called by BIOS. It actually supports many file system read operations, as shown in the following figure:



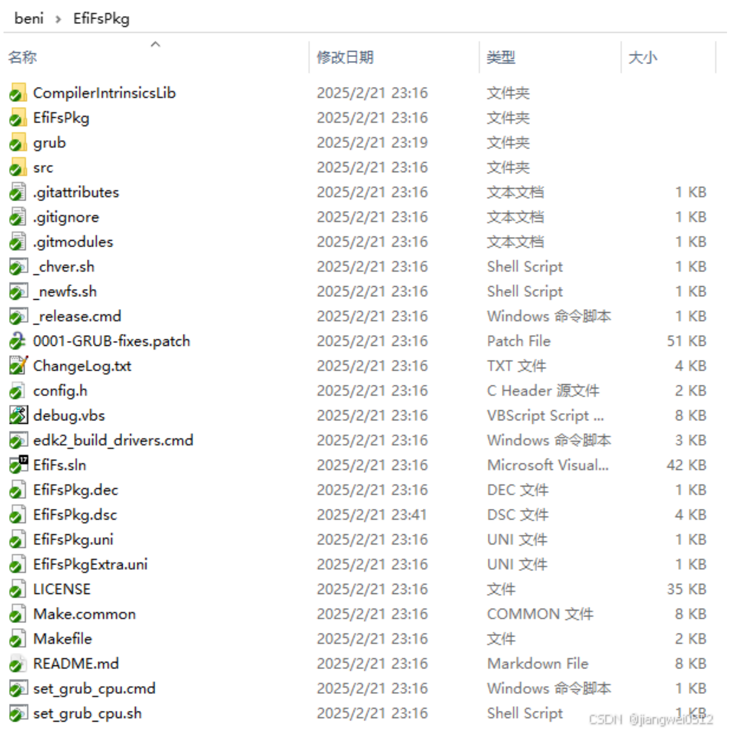
名称	修改日期	类型	大小
zfs	2025/2/21 23:19	文件夹	
affs.c	2025/2/21 23:30	C 源文件	18 KB
afs.c	2024/9/3 23:21	C 源文件	1 KB
archelp.c	2025/2/21 23:18	C 源文件	7 KB
bfs.c	2025/2/21 23:30	C 源文件	29 KB
btrfs.c	2025/2/21 23:30	C 源文件	68 KB
cbfs.c	2025/2/21 23:30	C 源文件	11 KB
cpio.c	2025/2/21 23:18	C 源文件	2 KB
cpio_be.c	2025/2/21 23:18	C 源文件	2 KB
cpio_common.c	2025/2/21 23:30	C 源文件	6 KB
erofs.c	2025/2/21 23:30	C 源文件	26 KB
exfat.c	2024/9/3 23:21	C 源文件	1 KB
ext2.c	2025/2/21 23:18	C 源文件	33 KB
f2fs.c	2025/2/21 23:30	C 源文件	38 KB
fat.c	2025/2/21 23:30	C 源文件	33 KB
fshelp.c	2024/9/3 23:21	C 源文件	13 KB
hfs.c	2025/2/21 23:30	C 源文件	36 KB
hfsplus.c	2025/2/21 23:30	C 源文件	33 KB
hfspluscomp.c	2025/2/21 23:30	C 源文件	9 KB
iso9660.c	2025/2/21 23:30	C 源文件	33 KB
jfs.c	2025/2/21 23:30	C 源文件	25 KB
minix.c	2025/2/21 23:18	C 源文件	21 KB
minix_be.c	2024/9/3 23:21	C 源文件	1 KB
minix2.c	2024/9/3 23:21	C 源文件	1 KB
minix2_be.c	2024/9/3 23:21	C 源文件	1 KB
minix3.c	2024/9/3 23:21	C 源文件	1 KB
minix3_be.c	2024/9/3 23:21	C 源文件	1 KB
newc.c	2025/2/21 23:18	C 源文件	2 KB
nilfs2.c	2025/2/21 23:30	C 源文件	32 KB
ntfs.c	2025/2/21 23:30	C 源文件	34 KB
ntfscomp.c	2025/2/21 23:18	C 源文件	11 KB
odc.c	2025/2/21 23:18	C 源文件	2 KB
proc.c	2025/2/21 23:30	C 源文件	5 KB
reiserfs.c	2025/2/21 23:30	C 源文件	44 KB
romfs.c	2025/2/21 23:18	C 源文件	13 KB
sfs.c	2025/2/21 23:30	C 源文件	20 KB
squash4.c	2025/2/21 23:30	C 源文件	28 KB
tar.c	2025/2/21 23:30	C 源文件	9 KB

So by wrapping GRUB, it is possible to make EDK support these file systems. In fact, there are already open source projects that support this operation, the corresponding code path is GitHub - pbatard/EfiFs: EFI FileSystem drivers .

This article describes how to put the above project into EDK code for compilation.

Compile

The EDK code repository used is edk2-beni: used to learn and verify UEFI BIOS. Corresponding to EfiFsPkg, the GRUB code it depends on is also included.


beni › EfiFsPkg

名称	修改日期	类型	大小
CompilerIntrinsicsLib	2025/2/21 23:16	文件夹	
EfiFsPkg	2025/2/21 23:16	文件夹	
grub	2025/2/21 23:19	文件夹	
src	2025/2/21 23:16	文件夹	
.gitattributes	2025/2/21 23:16	文本文档	1 KB
.gitignore	2025/2/21 23:16	文本文档	1 KB
.gitmodules	2025/2/21 23:16	文本文档	1 KB
_chver.sh	2025/2/21 23:16	Shell Script	1 KB
_newfs.sh	2025/2/21 23:16	Shell Script	1 KB
_release.cmd	2025/2/21 23:16	Windows 命令脚本	1 KB
0001-GRUB-fixes.patch	2025/2/21 23:16	Patch File	51 KB
ChangeLog.txt	2025/2/21 23:16	TXT 文件	4 KB
config.h	2025/2/21 23:16	C Header 源文件	2 KB
debug.vbs	2025/2/21 23:16	VBScript Script ...	8 KB
edk2_build_drivers.cmd	2025/2/21 23:16	Windows 命令脚本	3 KB
EfiFs.sln	2025/2/21 23:16	Microsoft Visual...	42 KB
EfiFsPkg.dec	2025/2/21 23:16	DEC 文件	1 KB
EfiFsPkg.dsc	2025/2/21 23:41	DSC 文件	4 KB
EfiFsPkg.uni	2025/2/21 23:16	UNI 文件	1 KB
EfiFsPkgExtra.uni	2025/2/21 23:16	UNI 文件	1 KB
LICENSE	2025/2/21 23:16	文件	35 KB
Make.common	2025/2/21 23:16	COMMON 文件	8 KB
Makefile	2025/2/21 23:16	文件	2 KB
README.md	2025/2/21 23:16	Markdown File	8 KB
set_grub_cpu.cmd	2025/2/21 23:16	Windows 命令脚本	1 KB
set_grub_cpu.sh	2025/2/21 23:16	Shell Script	1 KB

EfiFsPkg has a separate dsc file that can be compiled, which contains the libraries and modules that need to be supported:

bash AI generated projects 登录复制

```bash
[LibraryClasses]
  #
  # Entry Point Libraries
  #
  UefiDriverEntryPoint|MdePkg/Library/UefiDriverEntryPoint/UefiDriverEntryPoint.inf
  #
  # Common Libraries
  #
  BaseLib|MdePkg/Library/BaseLib/BaseLib.inf
  BaseMemoryLib|MdePkg/Library/BaseMemoryLib/BaseMemoryLib.inf
  UefiLib|MdePkg/Library/UefiLib/UefiLib.inf
  PrintLib|MdePkg/Library/BasePrintLib/BasePrintLib.inf
  # 中间略

[Components]
  EfiFsPkg/EfiFsPkg/Afs.inf
  EfiFsPkg/EfiFsPkg/Affs.inf
  # 后面略
```

收起 ^

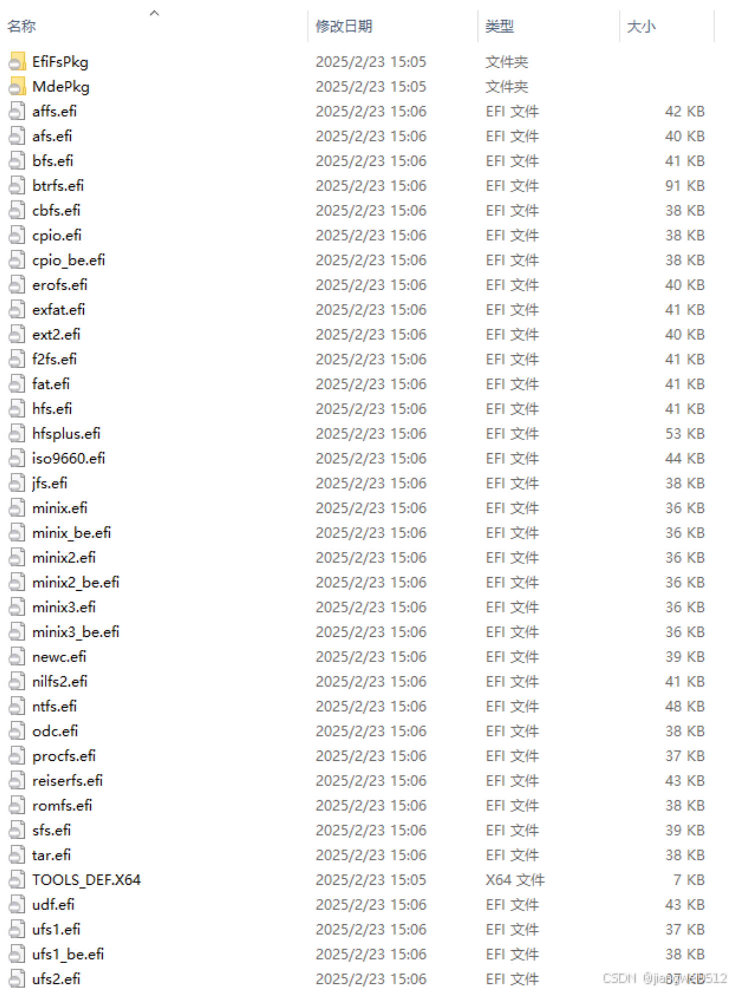
Since compilation requires some specific syntax, there are requirements for the version of Visual Studio. Here we directly use the latest VS2022 Community version.

Compilation can be done directly using the Build.cmd script:

bash AI generated projects 登录复制

```bash
Build.cmd efifs
```

Finally generate the EFI binary:



名称	修改日期	类型	大小
EfiFsPkg	2025/2/23 15:05	文件夹	
MdePkg	2025/2/23 15:05	文件夹	
affs.efi	2025/2/23 15:06	EFI 文件	42 KB
afs.efi	2025/2/23 15:06	EFI 文件	40 KB
bfs.efi	2025/2/23 15:06	EFI 文件	41 KB
btrfs.efi	2025/2/23 15:06	EFI 文件	91 KB
cbfs.efi	2025/2/23 15:06	EFI 文件	38 KB
cpio.efi	2025/2/23 15:06	EFI 文件	38 KB
cpio_be.efi	2025/2/23 15:06	EFI 文件	38 KB
erofs.efi	2025/2/23 15:06	EFI 文件	40 KB
exfat.efi	2025/2/23 15:06	EFI 文件	41 KB
ext2.efi	2025/2/23 15:06	EFI 文件	40 KB
f2fs.efi	2025/2/23 15:06	EFI 文件	41 KB
fat.efi	2025/2/23 15:06	EFI 文件	41 KB
hfs.efi	2025/2/23 15:06	EFI 文件	41 KB
hfsplus.efi	2025/2/23 15:06	EFI 文件	53 KB
iso9660.efi	2025/2/23 15:06	EFI 文件	44 KB
jfs.efi	2025/2/23 15:06	EFI 文件	38 KB
minix.efi	2025/2/23 15:06	EFI 文件	36 KB
minix_be.efi	2025/2/23 15:06	EFI 文件	36 KB
minix2.efi	2025/2/23 15:06	EFI 文件	36 KB
minix2_be.efi	2025/2/23 15:06	EFI 文件	36 KB
minix3.efi	2025/2/23 15:06	EFI 文件	36 KB
minix3_be.efi	2025/2/23 15:06	EFI 文件	36 KB
newc.efi	2025/2/23 15:06	EFI 文件	39 KB
nilfs2.efi	2025/2/23 15:06	EFI 文件	41 KB
ntfs.efi	2025/2/23 15:06	EFI 文件	48 KB
odc.efi	2025/2/23 15:06	EFI 文件	38 KB
procfs.efi	2025/2/23 15:06	EFI 文件	37 KB
reiserfs.efi	2025/2/23 15:06	EFI 文件	43 KB
romfs.efi	2025/2/23 15:06	EFI 文件	38 KB
sfs.efi	2025/2/23 15:06	EFI 文件	39 KB
tar.efi	2025/2/23 15:06	EFI 文件	38 KB
TOOLS_DEF.X64	2025/2/23 15:05	X64 文件	7 KB
udf.efi	2025/2/23 15:06	EFI 文件	43 KB
ufs1.efi	2025/2/23 15:06	EFI 文件	37 KB
ufs1_be.efi	2025/2/23 15:06	EFI 文件	38 KB
ufs2.efi	2025/2/23 15:06	EFI 文件	37 KB

use

Here we test the ext2.efi file, which is included in the BIOS binary:

bash AI generated projects 登录复制

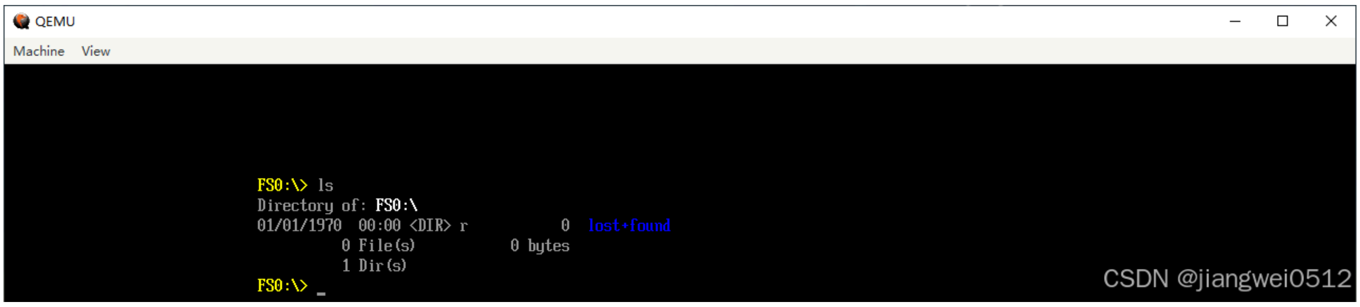
```bash
FILE DRIVER = 7DDA7772-B8F5-4859-9DBA-0D6F2DBA4AF1 {
  SECTION PE32 = Build/EfiFs/$(COMPILE_DIR)/X64/ext2.efi
}
```

Then create a disk.img file, format it into ext4 format, and execute the following script:

bash AI generated projects 登录复制

```bash
Build.cmd start disk
```

Execute QEMU and you can see that the ext4 formatted disk.img can be recognized:



```
FS0:\> ls
Directory of: FS0:\
01/01/1970  00:00 <DIR> r           0  lost+found
          0 File(s)           0 bytes
          1 Dir(s)
FS0:\> _
```

Implementation principle

The implementation of the file system under BIOS mainly depends on two important structures, one is `EFI_SIMPLE_FILE_SYSTEM_PROTOCOL` :

c AI generated projects 登录复制 run

```c
struct _EFI_SIMPLE_FILE_SYSTEM_PROTOCOL {
  ///
  /// The version of the EFI_SIMPLE_FILE_SYSTEM_PROTOCOL. The version
  /// specified by this specification is 0x00010000. All future revisions
  /// must be backwards compatible.
  ///
  UINT64                                     Revision;
  EFI_SIMPLE_FILE_SYSTEM_PROTOCOL_OPEN_VOLUME    OpenVolume;
};
```

`OpenVolume()` Used to open the root directory of a file system and get another important structure:

c AI generated projects 登录复制 run

```c
struct _EFI_FILE_PROTOCOL {
  ///
  /// The version of the EFI_FILE_PROTOCOL interface. The version specified
  /// by this specification is EFI_FILE_PROTOCOL_LATEST_REVISION.
  /// Future versions are required to be backward compatible to version 1.0.
  ///
  UINT64                   Revision;
  EFI_FILE_OPEN            Open;
  EFI_FILE_CLOSE           Close;
  EFI_FILE_DELETE          Delete;
  EFI_FILE_READ            Read;
  EFI_FILE_WRITE           Write;
  EFI_FILE_GET_POSITION    GetPosition;
  EFI_FILE_SET_POSITION    SetPosition;
  EFI_FILE_GET_INFO        GetInfo;
  EFI_FILE_SET_INFO        SetInfo;
  EFI_FILE_FLUSH           Flush;
  EFI_FILE_OPEN_EX         OpenEx;
  EFI_FILE_READ_EX         ReadEx;
  EFI_FILE_WRITE_EX        WriteEx;
  EFI_FILE_FLUSH_EX        FlushEx;
};
```

收起 ^

This includes file-related operations, so it is the key point that needs to be implemented, and it includes GRUB's file operation code.

c AI generated projects 登录复制 run

```c
    This->RootFile->EfiFile.Open = FileOpen;
    This->RootFile->EfiFile.Close = FileClose;
    This->RootFile->EfiFile.Delete = FileDelete;
    This->RootFile->EfiFile.Read = FileRead;
    This->RootFile->EfiFile.Write = FileWrite;
    This->RootFile->EfiFile.GetPosition = FileGetPosition;
    This->RootFile->EfiFile.SetPosition = FileSetPosition;
    This->RootFile->EfiFile.GetInfo = FileGetInfo;
    This->RootFile->EfiFile.SetInfo = FileSetInfo;
    This->RootFile->EfiFile.Flush = FileFlush;
    This->RootFile->EfiFile.OpenEx = FileOpenEx;
    This->RootFile->EfiFile.ReadEx = FileReadEx;
    This->RootFile->EfiFile.WriteEx = FileWriteEx;
    This->RootFile->EfiFile.FlushEx = FileFlushEx;
```

收起 ^

Take `FileOpen()` for example:

c AI generated projects 登录复制 run

```c
static EFI_STATUS EFIAPI
FileOpen(EFI_FILE_HANDLE This, EFI_FILE_HANDLE *New,
        CHAR16 *Name, UINT64 Mode, UINT64 Attributes)
{
  // 其它略

    /* Finally we can call on GRUB open() if it's a regular file */
    if (!NewFile->IsDir) {
        Status = GrubOpen(NewFile);
    }
}
```

收起 ^

You can see that it contains a lot of `Grub` beginning codes, all of which are located in EfiFsPkg\src\grub_file.c, where the GRUB code will be called:

c AI generated projects 登录复制 run

```c
EFI_STATUS
GrubOpen(EFI_GRUB_FILE *File)
{
    grub_fs_t p = grub_fs_list;
    grub_file_t f = (grub_file_t) File->GrubFile;
    grub_err_t rc;

    grub_errno = 0;
    rc = p->fs_open(f, File->path);
    return GrubErrToEFIStatus(rc);
}
```

收起 ^