RT-THREAD File System Application Notes

RT-THREAD Documentation Center

WWW.RT-THREAD.ORG

Saturday 13th October, 2018

Machine Translated by Google

Table of contents

Table of contents i

1 Purpose and structure of this paper. . 1

1.1 Purpose and background of this paper. . 1

1.2 Structure of this paper. . 1

2 Problem statement . . . 1

3 Problem Solving. . 1

3.1 Introduction to DFS framework. 1

3.1.1. Origin of the DFS framework. . 2

3.1.2. Description of each level of the DFS framework. . . 2

3.1.2.1. Top layer: the POSIX file interface layer . . . 2

3.1.2.2. Middle layer: file system implementation layer. 2

3.1.2.3. Bottom layer: storage device driver layer. 2

3.2 File system migration. 3

3.2.1. Preparing the project. 3

3.2.2. Introduction to the migration process. 3

3.2.3. File system configuration. . 4

3.2.4. Storage device initialization. . 7

3.2.4.1. Enable SPI device driver. 7

3.2.4.2. Check the storage device driver. 8

3.2.4.3. Create a storage device . . . 9

3.3 Use of the file system. 10

3.3.1. Initialization of the file system. 10

3.3.1.1. Initialization of DFS framework. 11

3.3.1.2. Initialization of the intermediate file system. 11

3.3.1.3. Initialization of storage devices. 11

3.3.2. Create a file system. 12

3.3.3. Mounting the file system. 12

3.3.4. File and directory manipulation shell commands. 13

3.4 File operation examples. 15

4 Frequently Asked Questions 17

5 References. 18

5.1 All APIs related to this article. 18

5.1.1. API List. 18

5.1.2. Detailed explanation of core API 19

5.1.2.1. dfs_init() 19

5.1.2.2. elm_init() 19

5.1.2.3. dfs_mount() 19

This application note introduces the basic knowledge and usage of RT-Thread file system to help developers better The RT-Thread file system is used effectively. A code example verified on the Zhengdian Atom STM32F429-apollo development board is given .

1 Purpose and structure of this paper

1.1 Purpose and Background of this Paper

Developers who are new to the RT-Thread file system may feel that the RT-Thread file system is too complicated and don't know where to start. They want to use the file system in their projects but don't know how to do it. This impression is caused by insufficient understanding of the RT-Thread DFS framework. If you understand the DFS framework, you will be able to use the RT-Thread file system with ease.

In order to enable developers to clearly understand the concept of RT-Thread DFS framework and learn to use RT-Thread file system, this application note will introduce the relevant knowledge and implementation principles of RT-Thread DFS framework step by step. By demonstrating shell commands and using examples to operate the file system, developers can learn how to use RT-Thread file system.

1.2 Structure of this paper

This application note will introduce the RT-Thread file system from the following three aspects:

• RT-Thread DFS framework • RT-Thread file system migration • RT-Thread file system usage

2 Problem Statement

This application note will introduce the RT-Thread file system around the following issues.

• How to migrate various types of file systems? • How to operate the file system? • How to operate files and folders in the file system?

To solve these problems, we need to understand the RT-Thread DFS framework.

Use it in a unified way.

3. Problem Solving

3.1 Introduction to DFS Framework

RT-Thread's file system adopts a three-layer structure, which is the RT-Thread DFS framework.

The following figure shows the structure of the **RT-Thread** file system:

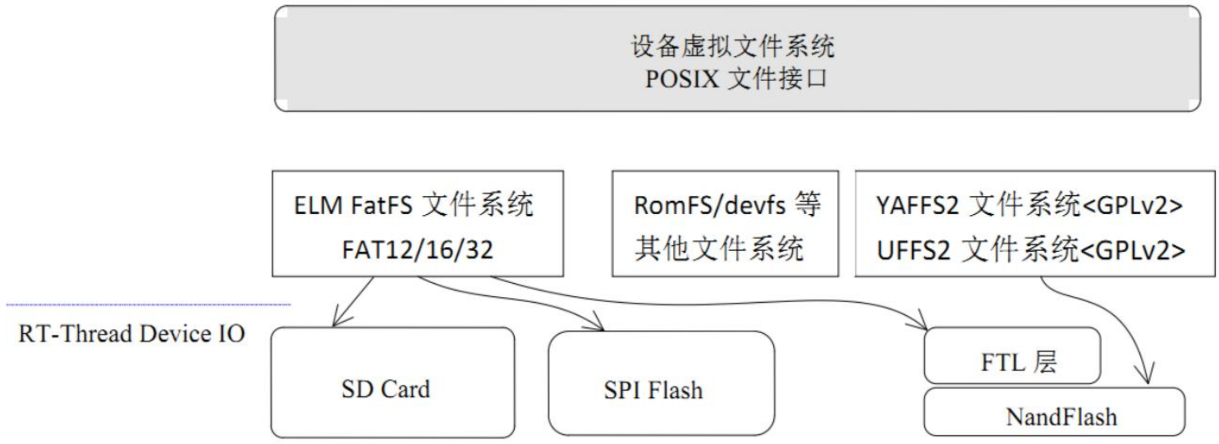


figure 1: File system structure diagram

The top layer of the DFS framework is a set of device virtual file system POSIX file interfaces specially optimized for embedded systems.

The upper layer is the implementation of various file systems, and the bottom layer is the drivers of various storage devices.

3.1.1. Origin of the DFS framework

- In order to support various file systems, RT-Thread has designed a DFS framework, where each layer is implemented independently, improving the scalability of the operating system. Using the DFS framework, various file systems can be matched to this framework with simple modifications, reducing the difficulty of file system porting and giving developers more file system types to choose from.

3.1.2. Description of each level of the DFS framework

3.1.2.1. Top layer: POSIX file interface layer

- This layer is the interface function layer for developers. Developers use the POSIX file interface provided by this layer to perform file-related operations. You don't need to worry about how the file system is implemented or which storage the data is stored in.

3.1.2.2. Middle layer: file system implementation layer

- The middle layer is the implementation of various specific file systems. The file systems mentioned here refer to various types of file systems, such as ELM FatFS, RomFS, devfs, Yaffs2, Uffs2 , etc. It is important to know that different file system types are implemented independently of the storage device driver. Therefore, in order to use these file systems correctly, it is necessary to connect the driver interface of the underlying storage device with the file system.

3.1.2.3. Bottom layer: storage device driver layer

- This layer is the storage device driver layer. Its specific function is to initialize the storage device and provide the storage device driver interface to the upper layer. The device type may be SPI Flash, SD card , etc.

Machine Translated by Google

3.2 File system migration

• This demonstration uses the STM32F429-Apollo development board , and the selected file system type is elm FatFS . Since RT-Thread comes with this file system, the porting work is relatively simple, and only needs to configure the system appropriately through the env tool

The porting process for other file systems supported by RT-Thread is similar. You only need to configure the system appropriately.

use.

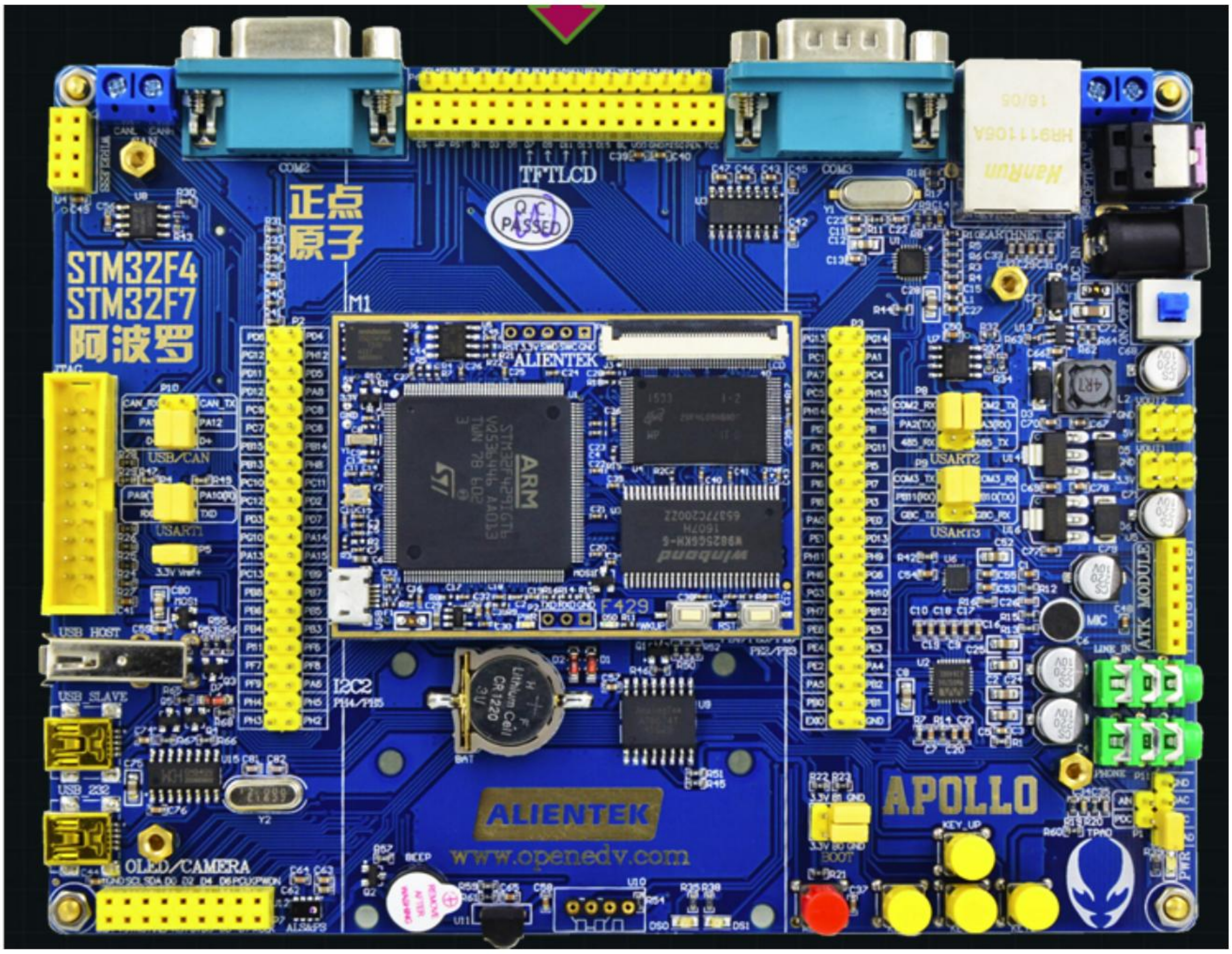


Figure 2: *stm32f429-apollo* Development Boards

3.2.1. Preparing the project

• Download RT-Thread source code .

• env tool

3.2.2. Introduction to the transplant process

The migration of the file system mainly includes the following aspects:

• Enable/configure the DFS framework

• Enable/configure the specified file system

• Ensure that the storage device driver on the development board works properly

The env tool can be used to easily enable the file system and add the required file system type to the project.

Performing a functional test on the storage device can ensure that the storage device driver is working properly. The stable operation of the driver is the key to the normal operation of the file system.

Commonly used basis.

3.2.3. File system configuration

Use the env tool to enter the rt-thread\bsp\stm32f429-apollo directory and enter the menuconfig command in the command line to enter the configuration interface.

• In the menuconfig configuration interface, select RT-Thread Components ÿ Device virtual file system, as shown below:

Show:

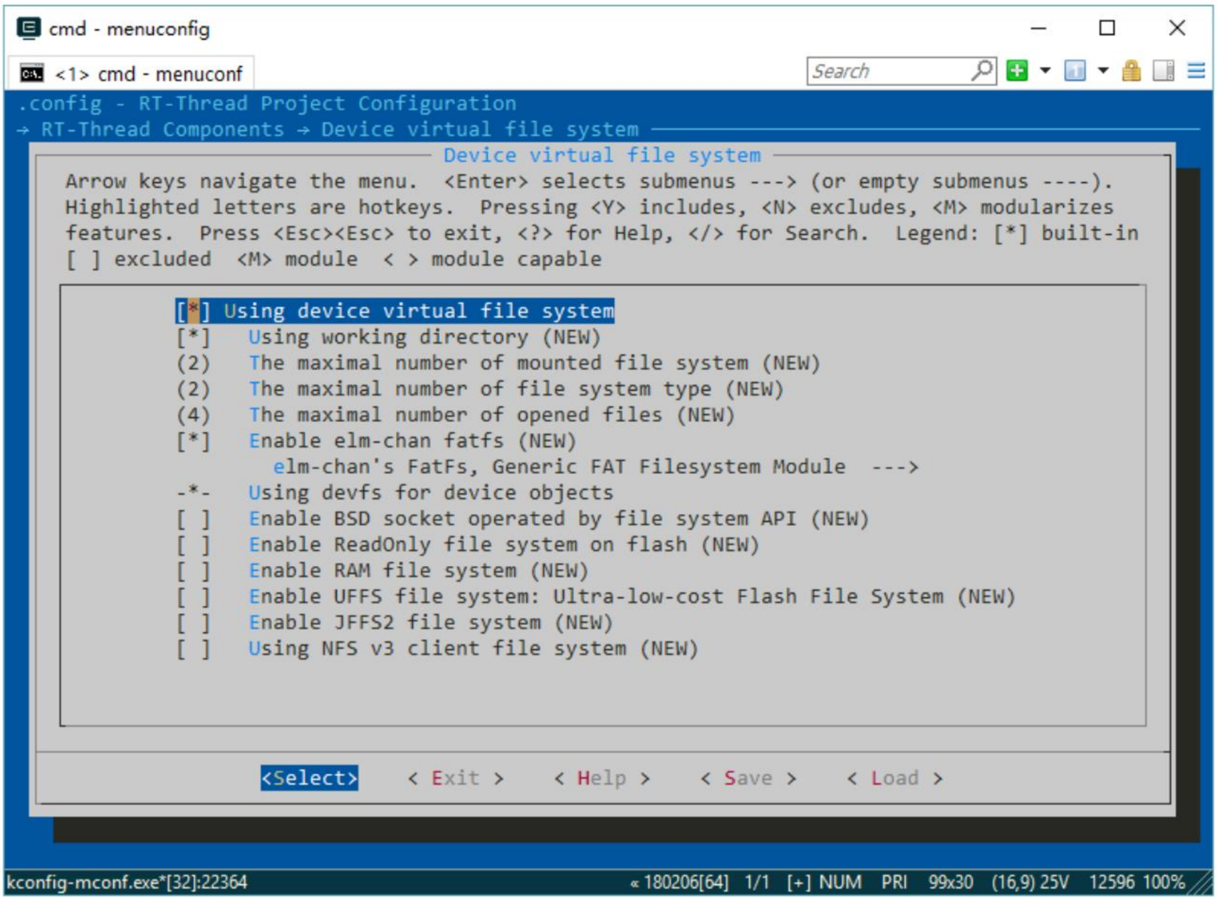


Figure 3: *menuconfig* Configuration interface

• The following describes the DFS configuration items:

– Using device virtual file system: Use the device virtual file system, that is, the RT-Thread file system.

– Using working directory: Turn on this option to use relative commands based on the current working directory in finsh/msh. path.

– The maximal number of mounted file system: The maximum number of mounted file systems.

– The maximal number of file system type: The maximum number of supported file system types.

– The maximal number of opened files: The maximum number of opened files.

– Enable elm-chan fatfs: Use elm-chan FatFs.

– elm-chan’s FatFs, Generic FAT Filesystem Module: Configuration items of elm-chan file system.

– Using devfs for device objects: Enable the devfs file system.

– Enable BSD socket operated by file system API: Enable BSD socket to use the file system API

Management, such as read and write operations and select/poll POSIX API calls.

– Enable ReadOnly file system on flash: Use read-only file system on Flash.

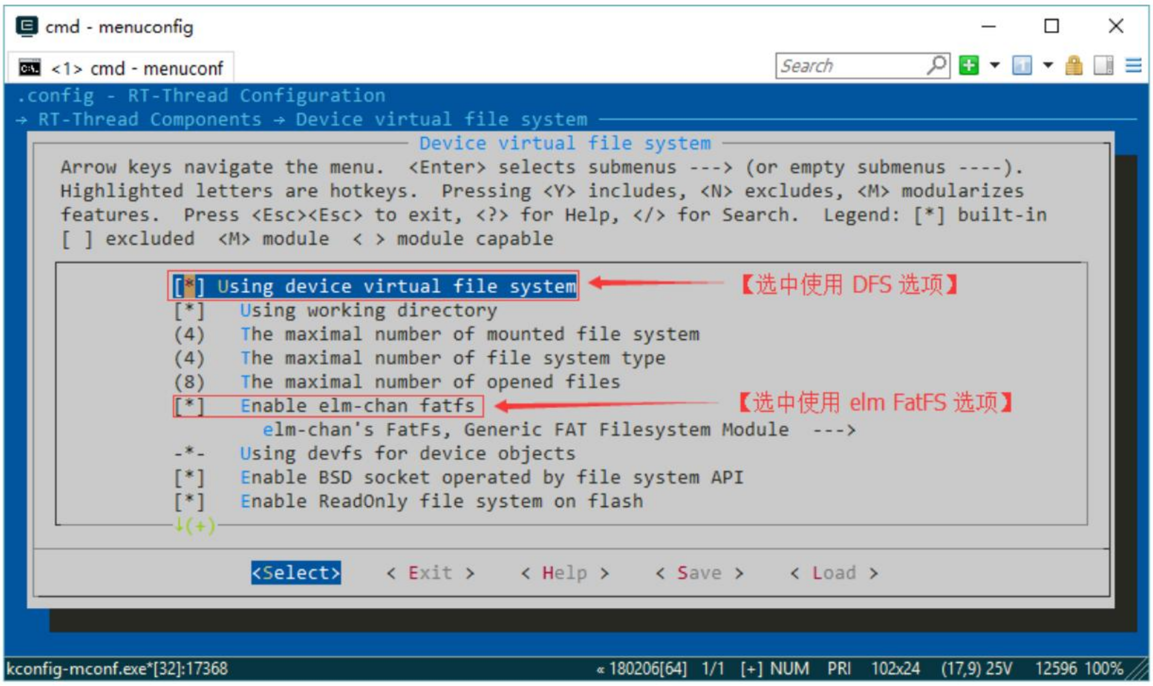
– Enable RAM file system: Use the RAM file system.

– Enable UFFS file system: Ultra-low-cost Flash File System: Use UFFS.

– Enable JFFS2 file system : Use JFFS2 file system.

– Using NFS v3 client file system: Use NFS file system.

• Enter the DFS configuration interface, turn on the options shown in the figure below, and then you can add FatFS to the system. As shown in the figure:



• It should be noted that you need to enter elm-chan's FatFs, Generic FAT Filesystem Module option to modify the option for long file name support, otherwise, when using the file system later, the name of the created file or folder cannot exceed 8 characters. The modification method is shown in the figure below:

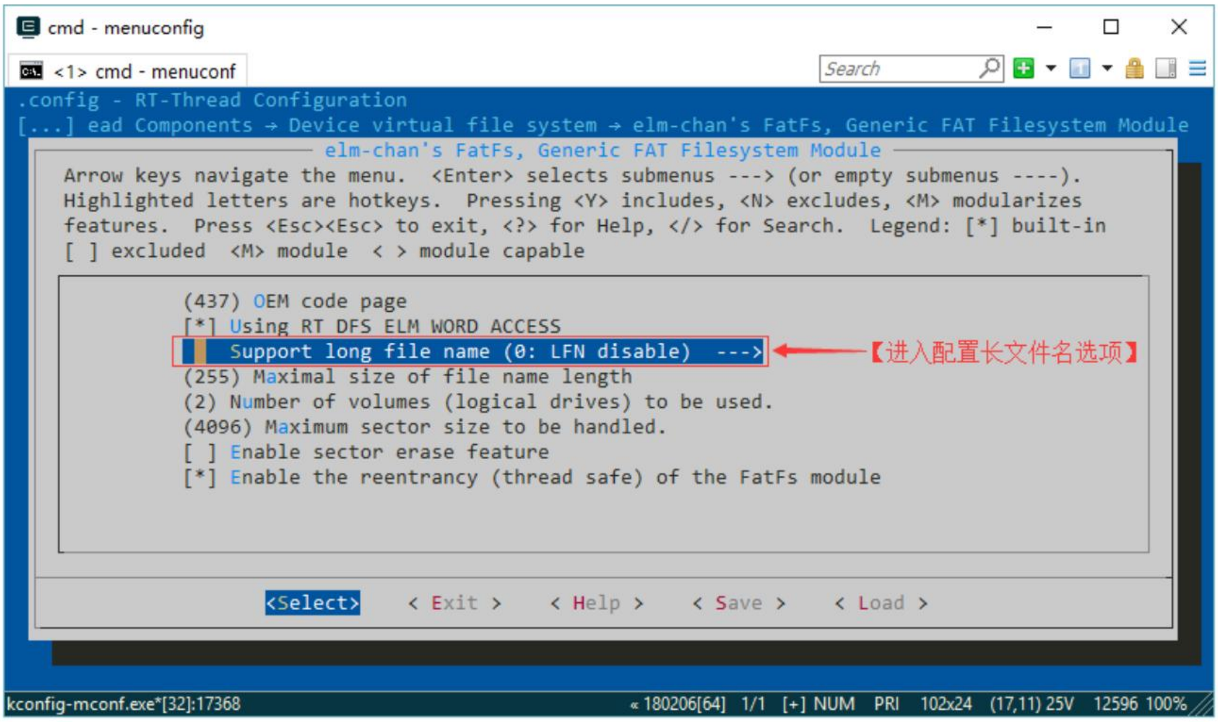


Figure 4: Configuring long file name options

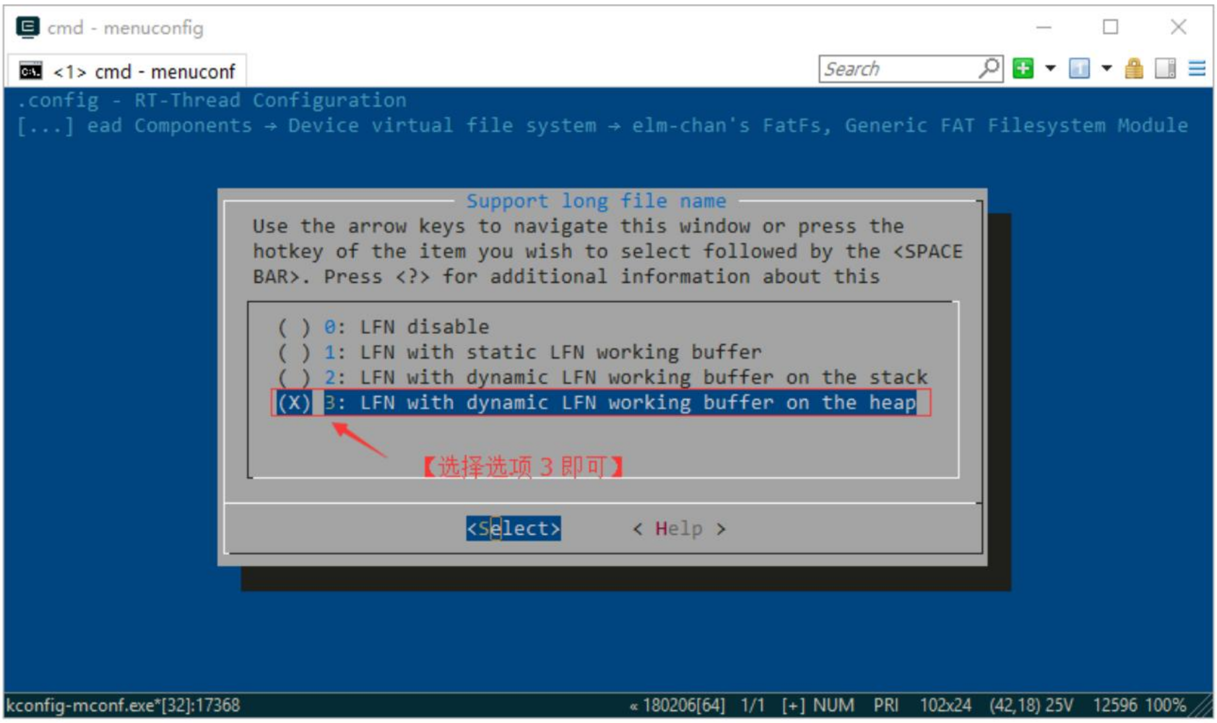


Figure 5: Select Options 3

• Because some C library functions are to be used, the libc function needs to be enabled:

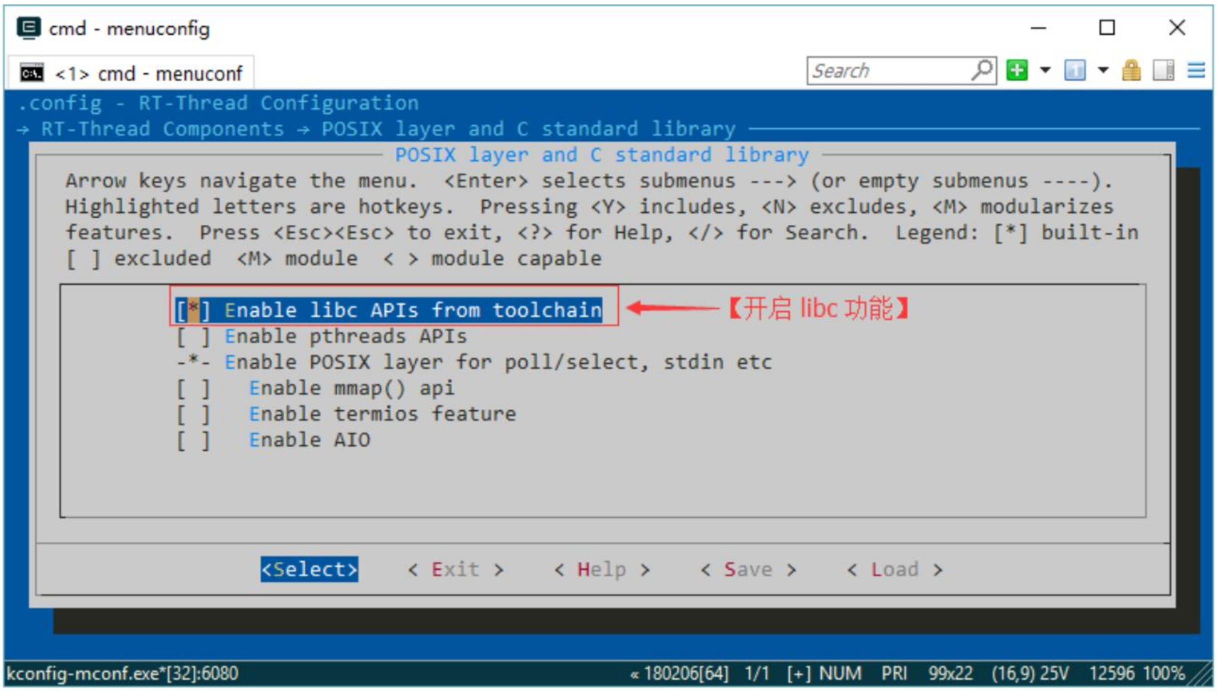


Figure 6: Open *libc*

- Save the options and exit. At this point, elm FatFS has been added to the project.

3.2.4. Storage device initialization

3.2.4.1. Enable SPI device driver

- The file system implementation layer of the DFS framework requires the storage device driver layer to provide a driver interface for docking. The storage device used in this article is SPI Flash, the initialization process of the underlying device can refer to "SPI Device Application Notes" .

- Reopen the menuconfig configuration interface and select Using SPI in the RT-Thread Components ÿ Device Drivers interface. Bus/Device device drivers and Using Serial Flash Universal Driver options, as shown below:

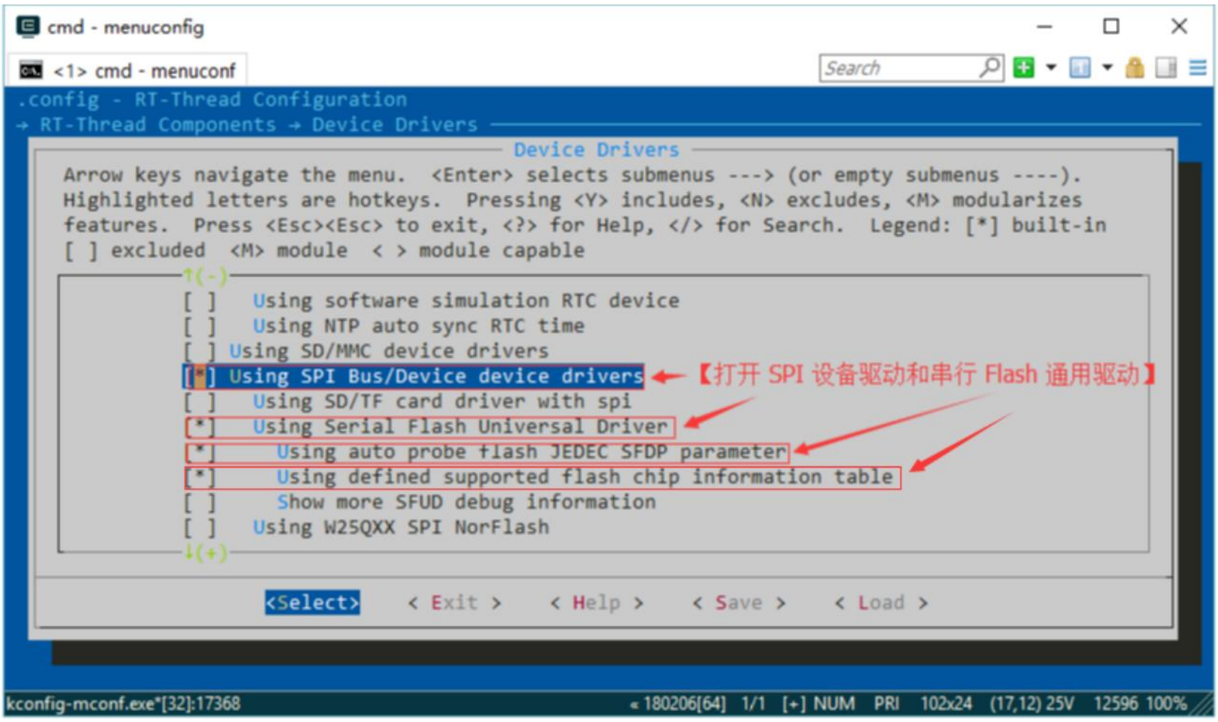


Figure 7: Open *SPI* drive

• To use shell commands conveniently, we enable Using shell commands in RT-Thread Components ÿ Command shell option.

The module shell options are shown below:

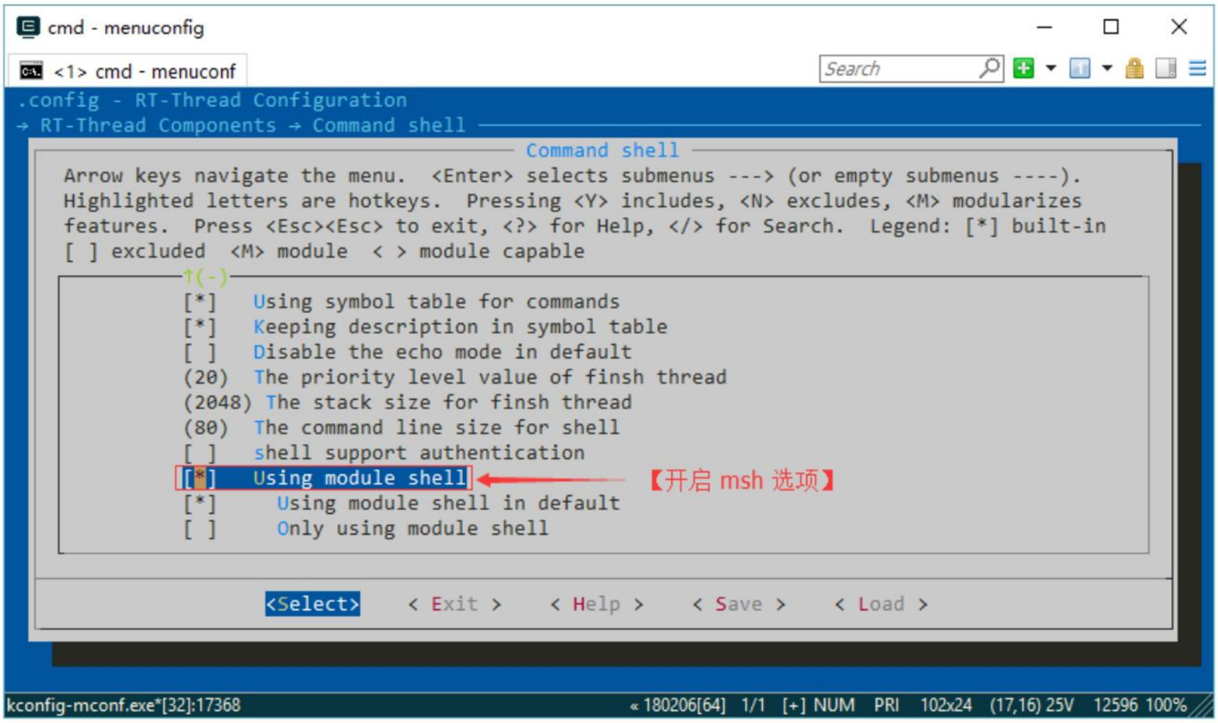


Figure 8: Open *msh* Options

• Save the options and exit, enter the command scons --target=mdk5 -s in env to generate the mdk5 project, compile and download the program.

3.2.4.2. Check storage device drivers

• On the stm32f429-apollo development board, the SPI Flash is connected to the SPI5 bus, and the corresponding SPI Device is named spi50. Enter the list_device command in the terminal and you can see that the device type named spi50 is SPI Device, which means that the SPI device has been added successfully. If the corresponding device does not appear, you need to check the driver and find the error.

```
msh />list_device
device           type            ref count
------- -------------------- ----------
sf_cmd  Block Device         0
e0      Network Interface    0
W25Q256 Block Device         0
spi50   SPI Device           0
spi5    SPI Bus              0
i2c0    I2C Bus              1
nand0   MTD Device           0
sd0     Block Device         0
rtc     RTC                  0
uart3   Character Device     0
uart2   Character Device     0
uart1   Character Device     2
```

Figure 9: View device list

• To ensure that the driver works properly, you can use the sf command to perform a benchmark test on the device . This function is provided by the sfud component. You can determine whether the storage device driver is normal by checking the read, write, and erase functions of the storage device. If it is successful as shown in the figure below, it is considered that the driver works properly. If the test fails, you need to check the driver and use a logic analyzer to analyze the interface waveform of the storage device. The test process is shown in the figure below:

```
msh /spi>sf probe spi50
[SFUD]Find a Winbond flash chip. Size is 33554432 bytes.
[SFUD]sf_cmd flash device is initialize success.
32 MB sf_cmd is current selected device.      【检测 spi50 接口下是否连接了 SPI Flash】
msh /spi>sf bench yes                          【对存储设备进行测试】
Erasing the sf_cmd 33554432 bytes data, waiting...
Erase benchmark success, total time: 82.768S.
Writing the sf_cmd 33554432 bytes data, waiting...
Write benchmark success, total time: 131.073S.  【测试成功】
Reading the sf_cmd 33554432 bytes data, waiting...
Read benchmark success, total time: 16.320S.
```

Figure 10: *benchmark* test

3.2.4.3. Create storage devices

• Since only block devices can be connected to the file system, you need to find the SPI Flash device based on the SPI Device . And create the corresponding Block Device.

• Here you need to use the universal SPI Flash driver library: SFUD , RT-Thread has integrated this component, and we have enabled this function in the above configuration process. At this time, we only need to use the rt_sfud_flash_probe function provided by SFUD. This function will perform the following operations:

– Find the corresponding Flash storage device based on the SPI Device named spi50 .

– Initialize the Flash device.

– Create a Block Device named W25Q256 on the Flash storage device .

• If the component automatic initialization function is turned on, this function will be executed automatically, otherwise it needs to be called and run manually.

```
static int rt_hw_spi_flash_with_sfud_init(void) {

    if (RT_NULL == rt_sfud_flash_probe("W25Q256", "spi50")) {

        return RT_ERROR;
    };

    return RT_EOK;
}
INIT_COMPONENT_EXPORT(rt_hw_spi_flash_with_sfud_init)
```

• Enter the list_device command in the terminal. If you see a device named W25Q256 with the type Block Device, this means that the block device has been

If the creation is successful, if it fails, you need to check the spi50 device.

As shown below:

```
msh />list_device
device           type           ref count
------- -------------------- ----------
e0      Network Interface    0
W25Q256 Block Device         0      【根据 SPI Flash 创建 Block Device 成功】
spi50   SPI Device           0      【SPI Flash 设备初始化成功】
spi5    SPI Bus              0
i2c0    I2C Bus              1
nand0   MTD Device           0
sd0     Block Device         1
rtc     RTC                  0
uart3   Character Device     0
uart2   Character Device     0
uart1   Character Device     2
msh />
```

Figure 11: View block devices

• Once a block type device is available for mounting, the porting is complete.

3.3 Use of the file system

3.3.1. Initialization of the file system

The RT-Thread file system initialization process generally follows the following process:

1. Initialize the DFS framework

2. Initialize the specific file system

3. Initialize the storage device

Let's explain the initialization process of the file system step by step in this order:

3.3.1.1. Initialization of DFS Framework The initialization of DFS framework mainly involves initialization of internal data structures and resources. This process includes

This includes the data tables and mutexes required to initialize the file system. This function is completed by the following function. If the component automatic initialization function is turned on,

This function will be executed automatically, otherwise you need to call it manually.

```
int dfs_init(void)
{
    /* clear filesystem operations table */
    memset((void *)filesystem_operation_table, 0, sizeof(filesystem_operation_table));
    /* clear filesystem table */
    memset(filesystem_table, 0, sizeof(filesystem_table));
    /* clean fd table */
    memset(fd_table, 0, sizeof(fd_table));

    /* create device filesystem lock */
    rt_mutex_init(&fslock, "fslock", RT_IPC_FLAG_FIFO);

#ifdef DFS_USING_WORKDIR
    /* set current working directory */
    memset(working_directory, 0, sizeof(working_directory));
    working_directory[0] = '/';
#endif

#ifdef RT_USING_DFS_DEVFS
    {
        extern int devfs_init(void);

        /* if enable devfs, initialize and mount it as soon as possible */
        devfs_init();

        dfs_mount(NULL, "/dev", "devfs", 0, 0);
    }
#endif

    return 0;
}
INIT_PREV_EXPORT(dfs_init);
```

Figure 12: *DFS* Initialization of the framework

3.3.1.2. Initialization of the middle-layer file system The initialization of this step is mainly to register the operation function of elm FatFS to the DFS framework

This function is completed by the following function. If the component automatic initialization function is turned on, this function will be automatically executed, otherwise it needs to be called manually

run.

```
int elm_init(void)
{
    /* register fatfs file system */
    dfs_register(&dfs_elm);

    return 0;
}
INIT_COMPONENT_EXPORT(elm_init);
```

Figure 13: File system initialization

3.3.1.3. Initialization of storage devices For the initialization of storage devices, please refer to the "Creating Storage Devices" section.

3.3.2. Create a file system

- When using SPI Flash as the storage device for the file system for the first time , if we directly restart the development board to mount the file system, we will see the prompt "spi flash mount to /spi failed!" This is because the corresponding type of file system has not been created in the SPI Flash at this time, so the shell command to create a file system is used : mkfs.

- The mkfs command is used to create a file system of a specified type on a specified storage device. The usage format is: mkfs [-t type] device . Before mounting a file system for the first time, you need to use the mkfs command to create a corresponding file system on the storage device, otherwise the mount will fail. If you want to create an elm type file system on a W25Q256 device , you can use the mkfs -t elm W25Q256 command. The usage is as follows:

```
msh />list_device
device           type         ref count
------- -------------------- ----------
e0      Network Interface    0
W25Q256 Block Device         0
spi50   SPI Device           0
spi5    SPI Bus              0
i2c0    I2C Bus              1
nand0   MTD Device           0
sd0     Block Device         0
rtc     RTC                  0
uart3   Character Device     0
uart2   Character Device     0
uart1   Character Device     2
msh />mkfs -t elm W25Q256
msh />
```

【在 W25Q256 设备中初始化 elm 文件系统】

Figure 14: Creating a file system

- After the file system is created, the device needs to be restarted.

3.3.3. Mounting the file system

Mounting a file system refers to associating a file system with a specific storage device and mounting it to a mount point, which is the root directory of the file system. In the following example, we associate the elm FatFS file system with a storage device named W25Q256 and mount it to the /spi folder. (The reason why it can be mounted to the /spi folder here is that the root directory of the file system of the stm32f429-apollo BSP has already mounted RomFS, and the /spi folder has been created . If there are no special circumstances, the file system can be mounted directly to the root directory / .)

- The operation of mounting the file system is completed by the dfs_mount() function. The parameters of the dfs_mount() function are: block device name, file system mount point path, mount file system type, read-write flag, and file system private data. The usage is shown in the figure below:

```c
/* mount sd card fat partition 0 as root directory */
if (dfs_mount("W25Q256", "/spi", "elm", 0, 0) == 0)
{
    rt_kprintf("spi flash mount to /spi !\n");
}
else
{
    rt_kprintf("spi flash mount to /spi failed!\n");
}
```

Figure 15: Mounting the file system

• After the above file system creation operation, we restart the development board (which will automatically re-execute the mount function) and then we can successfully mount the

The file system is now installed. You can see the prompt spi flash mount to /spi !. Then use the list_device command again to see

The W25Q256 device has been mounted successfully. As shown in the following figure:

```
msh /spi>list_device
device           type         ref count
------- -------------------- ----------
e0      Network Interface    0
W25Q256 Block Device         1
spi50   SPI Device           0
spi5    SPI Bus              0
i2c0    I2C Bus              1
nand0   MTD Device           0
sd0     Block Device         0
rtc     RTC                  0
uart3   Character Device     0
uart2   Character Device     0
uart1   Character Device     2
```

【ref 的值由 0 变为 1，说明挂载成功了】

Figure 16: View Mounts

• At this point, the file system has been initialized and you can now operate on files and directories.

3.3.4. Shell commands for file and directory operations

This section introduces the commonly used shell commands for file and directory operations:

• **ls**

Function: Display information about files and directories, as shown in the following figure:

```
msh />ls
Directory /:
readme.txt          12
sdcard              <DIR>
spi                 <DIR>
```

Figure 17: *ls* Order

• **cd**

Function: Switch to the specified working directory, as shown in the following figure:

```
msh />cd spi
msh /spi>
```

Figure 18: *cd* Order

• **cp**

Function: copy file, the example is as follows:

```
msh /spi>cp ../readme.txt .
msh /spi>ls
Directory /spi:
readme.txt             12
msh /spi>
```

Figure 19: *cp* Order

- **rm**

Function: Delete files or directories. The example is as follows:

```
msh /spi>rm readme.txt
msh /spi>ls
Directory /spi:
msh /spi>
```

Figure 20: *rm* Order

- **mv**

Function: Move or rename the file, as shown in the following figure:

```
msh /spi>cp ../readme.txt .
msh /spi>ls
Directory /spi:
readme.txt             12
msh /spi>mv r
msh /spi>mv readme.txt hello.txt        【文件名自动补全】
readme.txt => hello.txt
msh /spi>ls
Directory /spi:
hello.txt              12
msh /spi>
```

Figure 21: *mv* Order

- **echo**

Function: Write the specified content to the file:

```
msh /spi>echo "RT-Thread" hello.txt
msh /spi>
```

Figure 22: *echo* Order

- **cat**

Function: Display the contents of the file, as shown in the following figure:

```
msh /spi>cat hello.txt
RT-Thread
msh /spi>
```

Figure 23: *cat* Order

- **pwd**

Function: Print out the current directory address, as shown in the following figure:

```
msh /spi>pwd
/spi
msh /spi>
```

Figure **24:** *pwd* Order

- **mkdir**

Function: Create a folder, the example is as follows:

```
msh /spi>mkdir hello_rt_thread
msh /spi>ls
Directory /spi:
hello.txt           12
hello_rt_thread     <DIR>
msh /spi>
```

Figure **25:** *mkdir* Order

3.4 File Operation Examples

This section takes the folder creation operation as an example to introduce how to use the RT-Thread file system Sample to operate the file system.

- In the menuconfig configuration interface, select RT-Thread online packages ÿ miscellaneous packages ÿ filesystem sample options, select the [filesystem] mkdir option, as shown in the following figure:

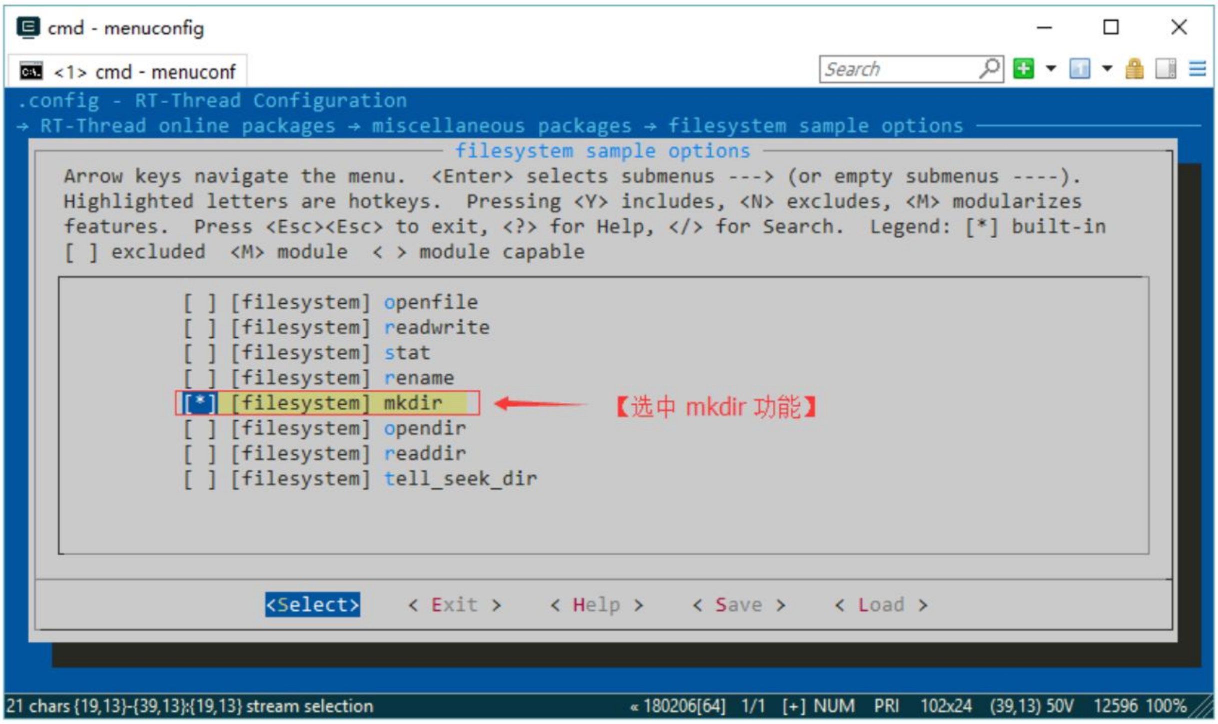


Figure **26:** Select *mkdir* Function

Machine Translated by Google

• After saving and exiting, use the pkgs --update command to update the software package, and then use the scons --target=mdk5 -s command to re-run the package.

Generate the project. You can see that the Sample has been added to the project:

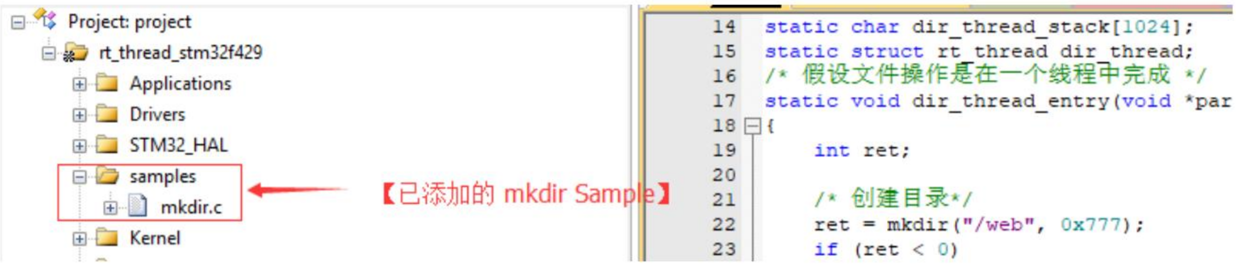


Figure 27: added *mkdir sample*

• It should be noted here that since the root directory of our file system is mounted with RomFS and cannot be modified, we cannot directly

Create a folder. Therefore, we need to make simple modifications to the program, as shown below:

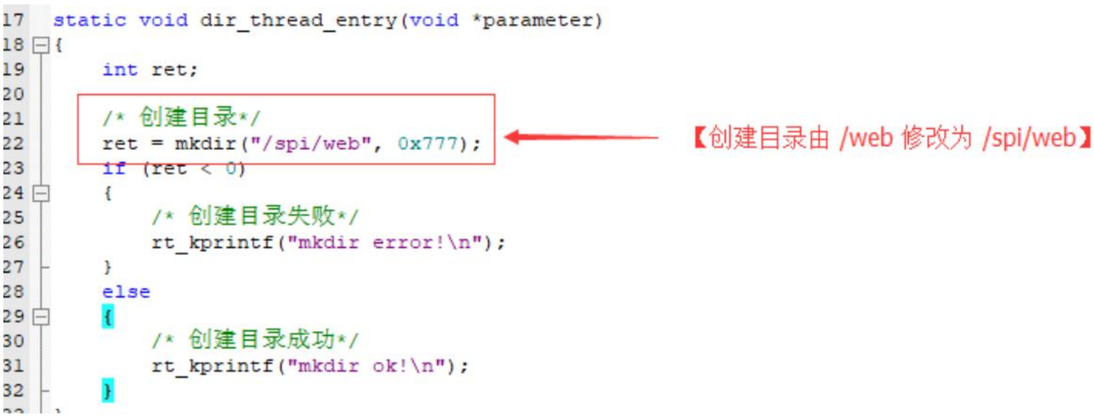


Figure 28: Create a directory

• After recompiling, download and run, you can use the mkdir_sample_init command in msh to create a web folder, the effect is as shown below

As shown:

Figure 29: Create Success

• Now switch to the /spi folder and you can see that the web folder has been created.

Figure 30: View Catalog

• The samples provided by the file system include openfile, readwrite, stat, rename, opendir, readdir , tell_seek_dir,

You can use these functions using the methods above.

4 Frequently Asked Questions

(1) What should I do if the file name or folder name is displayed abnormally?

- Check whether long file name support is enabled. Refer to the "File System Configuration" section of this application note.

(2) What should I do if the file system initialization fails?

- Check whether the types and number of file systems allowed to be mounted in the file system configuration item are sufficient.

(3) What should I do if the mkfs command fails to create the file system?

- Check if the storage device exists. If it does, check if the device driver can pass the functional test. If not, check the driver mistake.
- Check whether the libc function is enabled, see the "File System Configuration" section.

(4) What should I do if the file system fails to mount?

- Check whether the specified mount path exists. The file system can be mounted directly to the root directory ("/"), but if you want to mount it to another path, such as ("/sdcard"), you need to ensure that the ("/sdcard") path exists. Otherwise, you need to create an sdcard folder in the root directory before mounting successfully.
- Check whether a file system has been created on the storage device. If no file system has been created on the storage device, you need to use the mkfs command to create a file system on the storage device. Create a file system on the server.

(5) What should I do if SFUD cannot detect the specific model of Flash used?

- Check if the hardware pin setting is wrong
- Whether the SPI device has been registered
- Whether the SPI device has been mounted on the bus
- Check whether the Using **auto** probe flash JEDEC SFDP parameter and Using defined supported flash chip information table configuration items are selected under the RT-Thread Components ÿ Device Drivers -> Using SPI Bus/Device device drivers -> Using Serial Flash Universal Driver menu. If not, then these two options need to be enabled. For the configuration diagram, please refer to the "Enabling SPI Device Driver" section.
- If the storage device is still not recognized after enabling the above options, you can Raise issues in the project.

(6) How to set the maximum sector size of elm FatFS?

- Depending on the storage device used, it will be slightly different. Generally, it can be set to 4K according to the requirements of the Flash device. The value is 4096.
- The sector size of common TF cards and SD cards is generally set to 512.

(7) Why does the benchmark test of the storage device take too long?

- Compare the benchmark test data when the system tick is 1000 If the time difference between the final test time and the time required for this test is too large, it can be considered that the test work is not running normally.

Machine Translated by Google

- Check the system tick settings, because some delay operations are determined by the tick time, so you need to set it appropriately according to the system situation.

  If the system tick value is not less than 1000, you need to use a logic analyzer to check the waveform .

  The communication rate is normal.

(8) When implementing the elmfat file system with SPI Flash, how can we reserve some sectors from being used by the file system?

- You can use the partition provided by RT-Thread The tool package creates multiple block devices for the entire storage device and multiple blocks for the created

  Just assign different functions to the devices.

(9) What should I do if the program gets stuck while testing the file system?

- Try to use a debugger or print some necessary debugging information to determine where the program is stuck before raising questions.

(10) How to check file system problems step by step?

- A bottom-up approach can be used to troubleshoot the problem step by step.
- First check whether the storage device is successfully registered and functions normally.
- Check whether a file system is created on the storage device.
- Checks that the specified file system type is registered with the DFS framework, and often checks that the allowed file system types and number are sufficient.
- Check whether DFS is initialized successfully. This initialization operation is purely software-based, so the possibility of error is low.

  If automatic component initialization is turned on, there is no need to manually initialize it again.

5 References

5.1 All relevant **APIs** in this article

5.1.1. **API** List

File system initialization related API	Location
dfs_init()	dfs.c
elm_init()	dfs_elm.c
dfs_mount()	dfs_fs.c

Shell command related API	Location
cmd_cat()	msh_cmd.c
cmd_rm()	msh_cmd.c
cmd_cd()	msh_cmd.c
cmd_cp()	msh_cmd.c
cmd_mv()	msh_cmd.c

Shell command related API	Location
cmd_rm()	msh_cmd.c
cmd_pwd()	msh_cmd.c
cmd_mkdir()	msh_cmd.c
cmd_mkfs()	msh_cmd.c

5.1.2. Detailed explanation of core API

5.1.2.1. dfs_init() function function: - Initialize the RT-Thread file system DFS framework.

Function prototype:

```
int dfs_init(void)
```

Function return: Returns RT_EOK if successful.

5.1.2.2. elm_init() Function: - Register FatFS to the DFS framework.

Function prototype:

```
int elm_init(void)
```

Function return: Returns RT_EOK if successful.

5.1.2.3. dfs_mount() Function: - Mount a specific type of file system at the specified path.

Function prototype:

```
int dfs_mount(const char *device_name,
              const char *path,
              const char *filesystemtype,
              unsigned long rwflag,
              const void *data)
```

parameter	describe
device_name	The name of the block device containing the file system
path	File system mount point path
filesystemtype	The type of file system to be mounted
rwflag	Read and write flag
data	Private data of this file system

Function return: Returns RT_EOK if successful, or -1 if failed.