RT-THREAD File System Application Notes

RT-THREAD Document Center

WWW.RT-THREAD.ORG

Saturday 13th October, 2018

Table of contents

This application note introduces the basic knowledge and usage of RT-Thread file system to help developers better

The paper also provides code examples verified on the Zhengdian Atom STM32F429-apollo development board.

1 Purpose and structure of this paper

1.1 Purpose and Background of this Paper

Developers new to the RT-Thread file system may find it too complex and unsure where to begin. They may want to use the file system in their projects but not know how. This impression stems from a lack of understanding of the RT-Thread DFS framework. Understanding the DFS framework will make using the RT-Thread file system a breeze.

To help developers clearly understand the concepts of the RT-Thread DFS framework and learn how to use it, this application note provides a step-by-step introduction to the RT-Thread DFS framework and its implementation principles. By demonstrating shell commands and examples to manipulate the file system, developers can learn how to use the 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 porting • RT-Thread file
system usage

2 Problem Description

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

• How to port various file systems? • How to
operate on the file system? • How to
operate on files and folders in the file system?

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

Use it in a systematic way.

3. Problem Solving

3.1 Introduction to the 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:

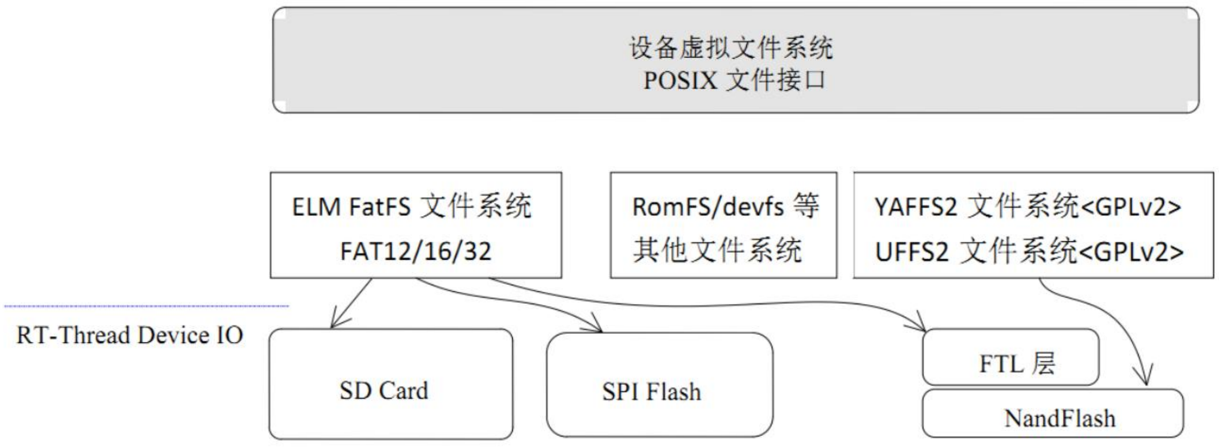


Figure 1: File system structure diagram

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

The top 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

- To support a variety of file systems, RT-Thread has designed a DFS framework, with independent implementations at each level, improving the scalability of the operating system. Using the DFS framework allows various file systems to be easily adapted to the framework, simplifying porting and providing developers with a wider range of 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 care 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 implements various specific file systems. File systems here refer to various types of file systems, such as ELM FatFS, RomFS, devfs, Yaffs2, and Uffs2. It's important to note that different file system types are implemented independently of storage device drivers. Therefore, to correctly use these file systems, you need to connect the underlying storage device driver interface to the file system.

3.1.2.3. Bottom layer: storage device driver layer

- This layer is the storage device driver layer, whose 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.

3.2 File system migration

• This demonstration uses the Zhengdian Atom development board STM32F429-Apollo , 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.

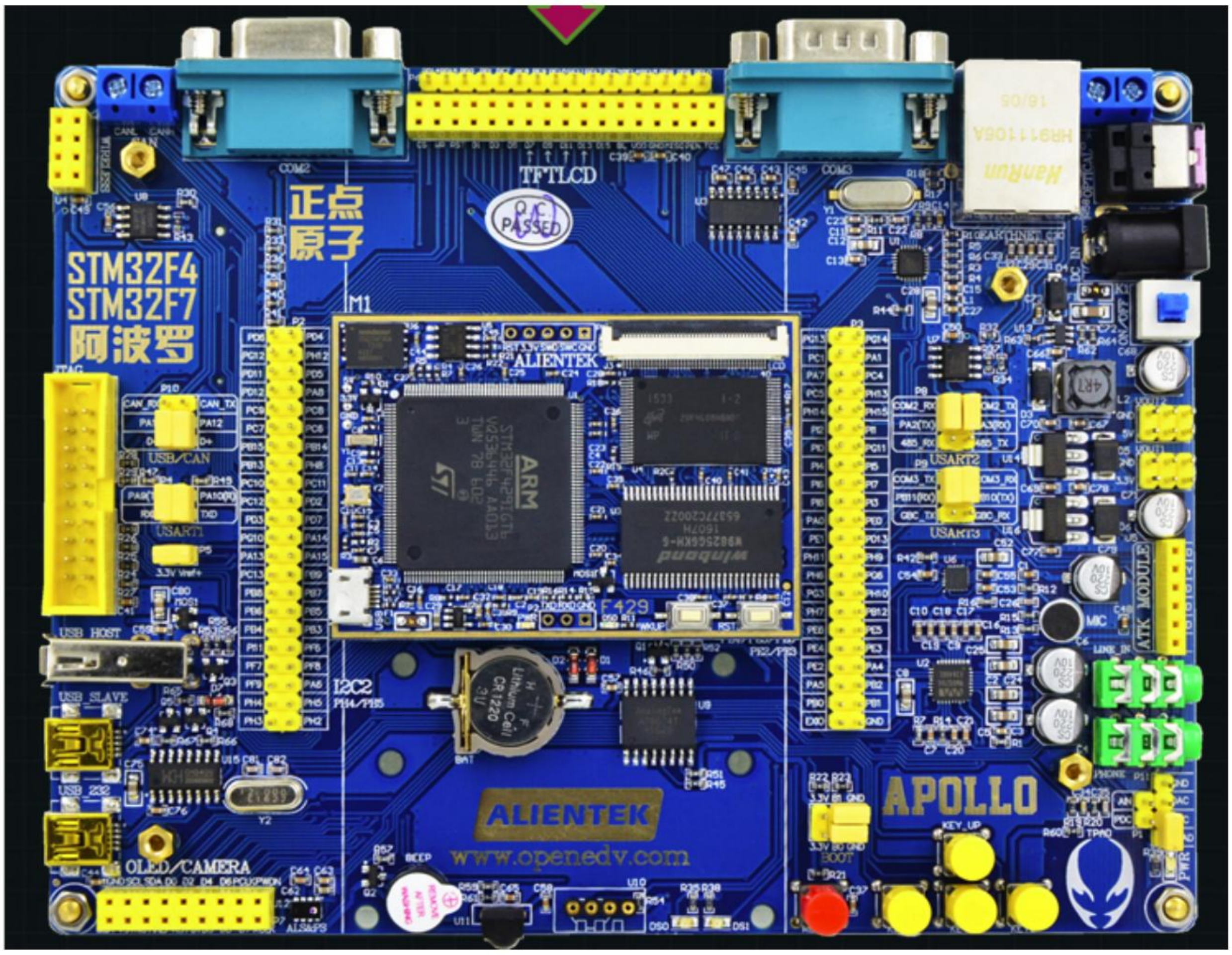


Figure 2: *stm32f429-apollo* Development Board

3.2.1. Preparing the project

• Download RT-Thread source code .

• env tool

3.2.2. Introduction to the transplant process

The file system transplantation mainly includes the following aspects:

• Enable/configure the DFS framework

• Open/configure the specified file system

• Ensure that the storage device driver on the development board is working 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.

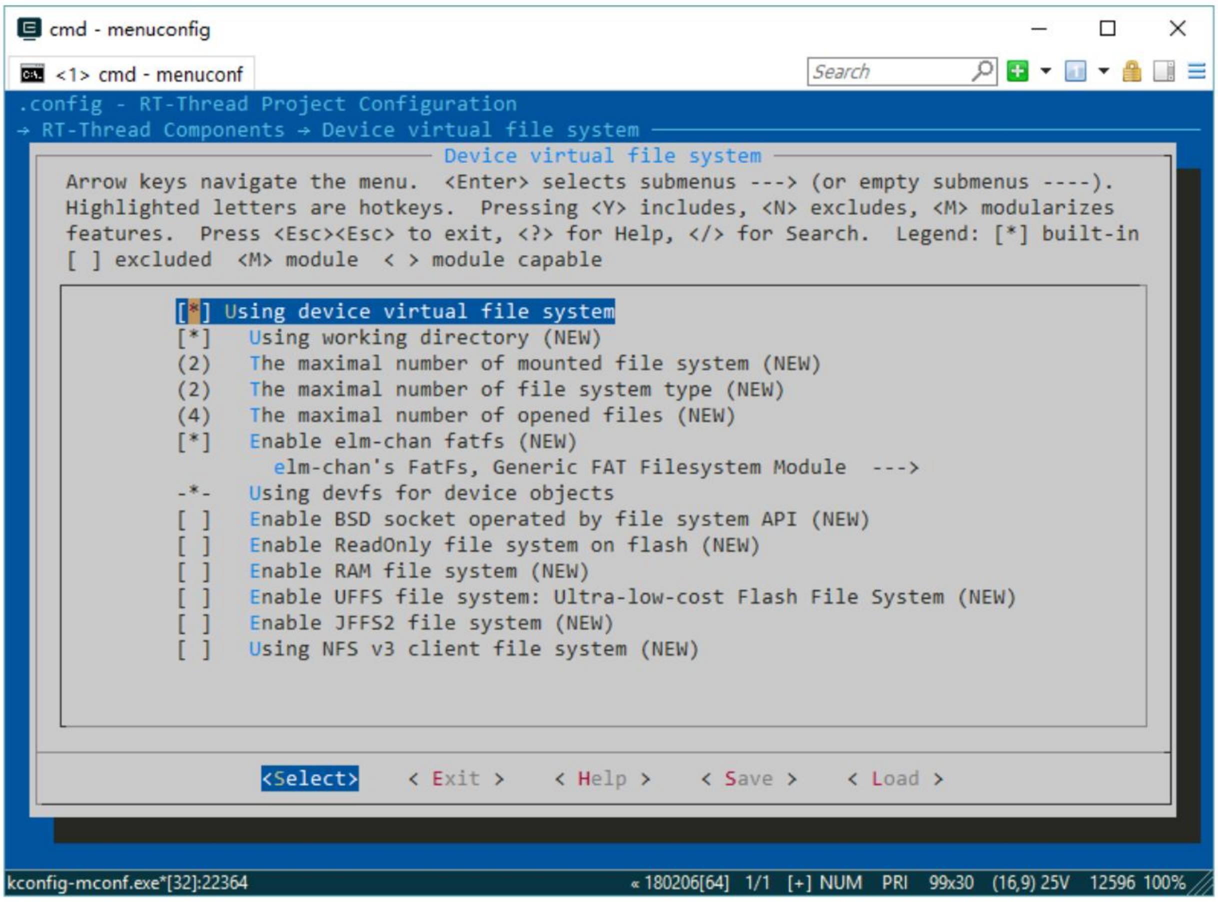
Commonly used foundation.

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:



```
.config - RT-Thread Project Configuration
→ RT-Thread Components → Device virtual file system
                        Device virtual file system
  Arrow keys navigate the menu.  <Enter> selects submenus ---> (or empty submenus ----).
  Highlighted letters are hotkeys.  Pressing <Y> includes, <N> excludes, <M> modularizes
  features.  Press <Esc><Esc> to exit, <?> for Help, </> for Search.  Legend: [*] built-in
  [ ] excluded  <M> module  < > module capable

          [*] Using device virtual file system
          [*]   Using working directory (NEW)
          (2)   The maximal number of mounted file system (NEW)
          (2)   The maximal number of file system type (NEW)
          (4)   The maximal number of opened files (NEW)
          [*]   Enable elm-chan fatfs (NEW)
                  elm-chan's FatFs, Generic FAT Filesystem Module  --->
          -*-   Using devfs for device objects
          [ ]   Enable BSD socket operated by file system API (NEW)
          [ ]   Enable ReadOnly file system on flash (NEW)
          [ ]   Enable RAM file system (NEW)
          [ ]   Enable UFFS file system: Ultra-low-cost Flash File System (NEW)
          [ ]   Enable JFFS2 file system (NEW)
          [ ]   Using NFS v3 client file system (NEW)

         <Select>    < Exit >    < Help >    < Save >    < Load >
```

cmd - menuconfig
<1> cmd - menuconf
Search
kconfig-mconf.exe*[32]:22364
« 180206[64] 1/1 [+] NUM PRI 99x30 (16,9) 25V 12596 100%

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 and you can 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 open files.

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

Machine Translated by Google

– 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: Enables read-only file system on Flash.

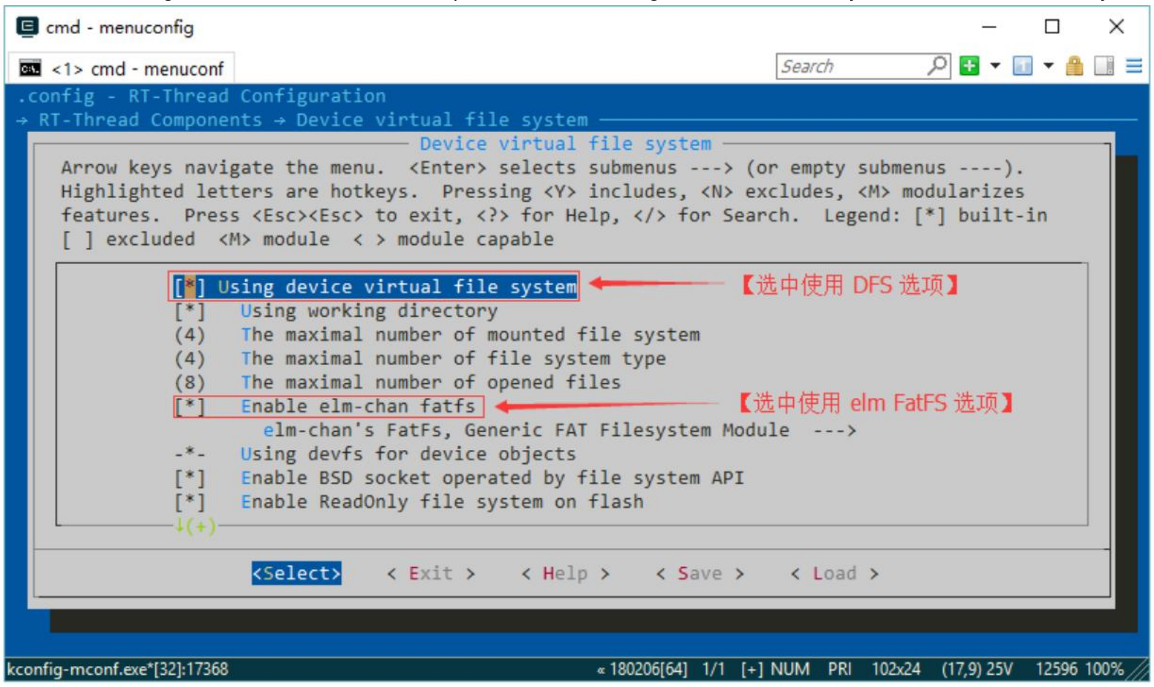
– Enable RAM file system: Use the RAM file system.

– 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.

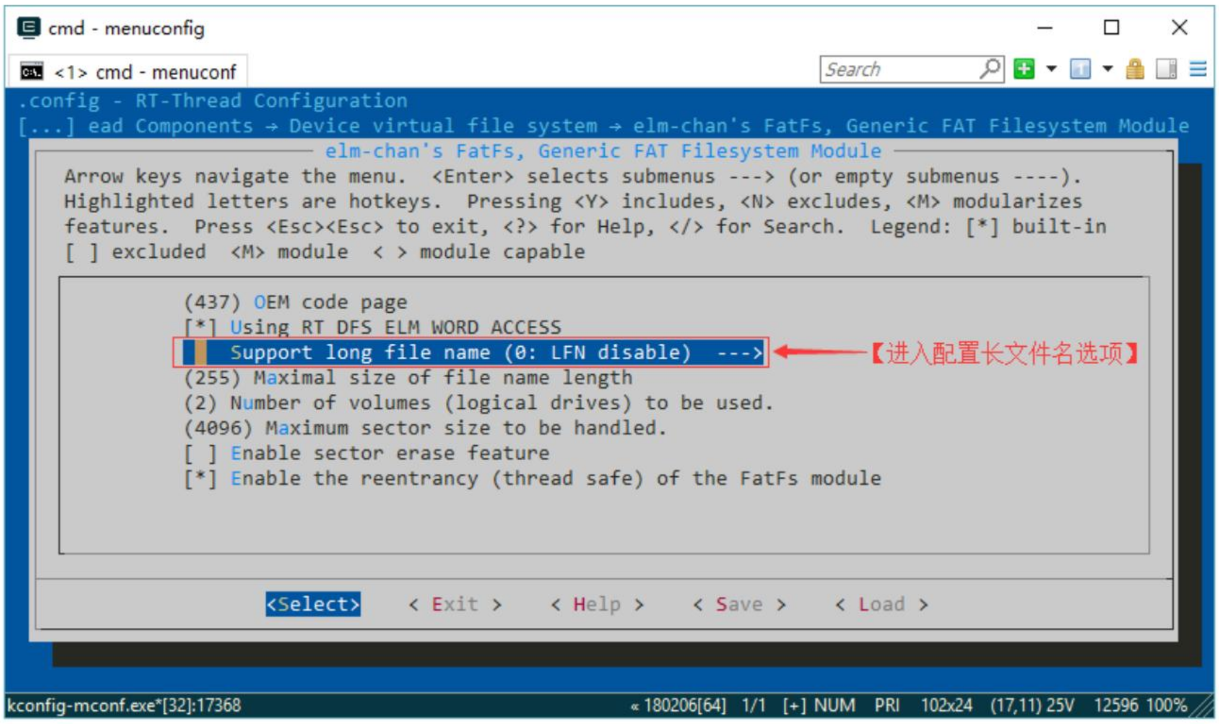


```
.config - RT-Thread Configuration
→ RT-Thread Components → Device virtual file system
                          Device virtual file system
  Arrow keys navigate the menu.  <Enter> selects submenus ---> (or empty submenus ----).
  Highlighted letters are hotkeys.  Pressing <Y> includes, <N> excludes, <M> modularizes
  features.  Press <Esc><Esc> to exit, <?> for Help, </> for Search.  Legend: [*] built-in
  [ ] excluded  <M> module  < > module capable

          [*] Using device virtual file system
          [*]   Using working directory
          (4)   The maximal number of mounted file system
          (4)   The maximal number of file system type
          (8)   The maximal number of opened files
          [*]   Enable elm-chan fatfs
                  elm-chan's FatFs, Generic FAT Filesystem Module  --->
          -*-   Using devfs for device objects
          [*]   Enable BSD socket operated by file system API
          [*]   Enable ReadOnly file system on flash
          ↓(+)

            <Select>    < Exit >    < Help >    < Save >    < Load >
```

cmd - menuconfig
<1> cmd - menuconf
Search
【选中使用 DFS 选项】
【选中使用 elm FatFS 选项】
kconfig-mconf.exe*[32]:17368
« 180206[64] 1/1 [+] NUM PRI 102x24 (17,9) 25V 12596 100%

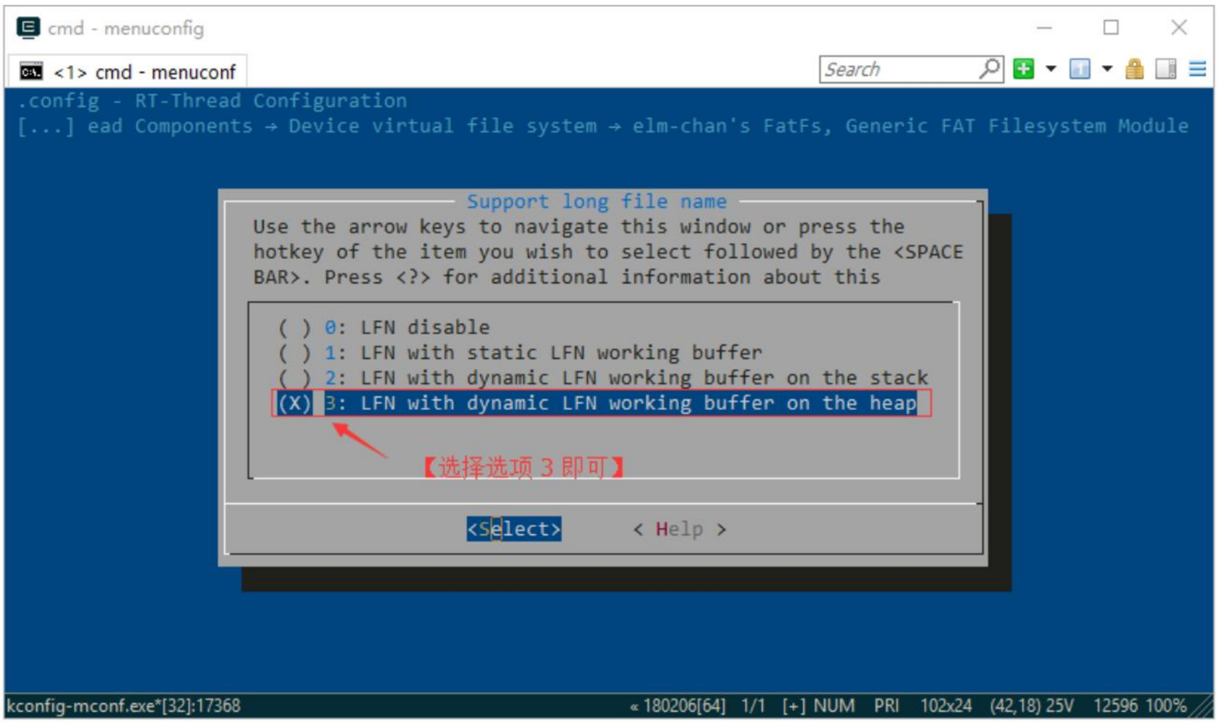
• It is important to note that you also need to modify the long file name support option in elm-chan's FatFs, Generic FAT Filesystem Module . Otherwise, when you use the file system later, the names of the files or folders you create will not exceed 8 characters. The modification method is shown in the figure below:

Machine Translated by Google



```
.config - RT-Thread Configuration
[...] ead Components → Device virtual file system → elm-chan's FatFs, Generic FAT Filesystem Module
                    elm-chan's FatFs, Generic FAT Filesystem Module
  Arrow keys navigate the menu.  <Enter> selects submenus ---> (or empty submenus ----).
  Highlighted letters are hotkeys.  Pressing <Y> includes, <N> excludes, <M> modularizes
  features.  Press <Esc><Esc> to exit, <?> for Help, </> for Search.  Legend: [*] built-in
  [ ] excluded  <M> module  < > module capable

          (437) OEM code page
          [*] Using RT DFS ELM WORD ACCESS
              Support long file name (0: LFN disable)  --->  【进入配置长文件名选项】
          (255) Maximal size of file name length
          (2) Number of volumes (logical drives) to be used.
          (4096) Maximum sector size to be handled.
          [ ] Enable sector erase feature
          [*] Enable the reentrancy (thread safe) of the FatFs module

          <Select>    < Exit >    < Help >    < Save >    < Load >

kconfig-mconf.exe*[32]:17368        « 180206[64]  1/1  [+] NUM  PRI  102x24  (17,11) 25V  12596 100%
```

Figure 4: Configuring long file name options

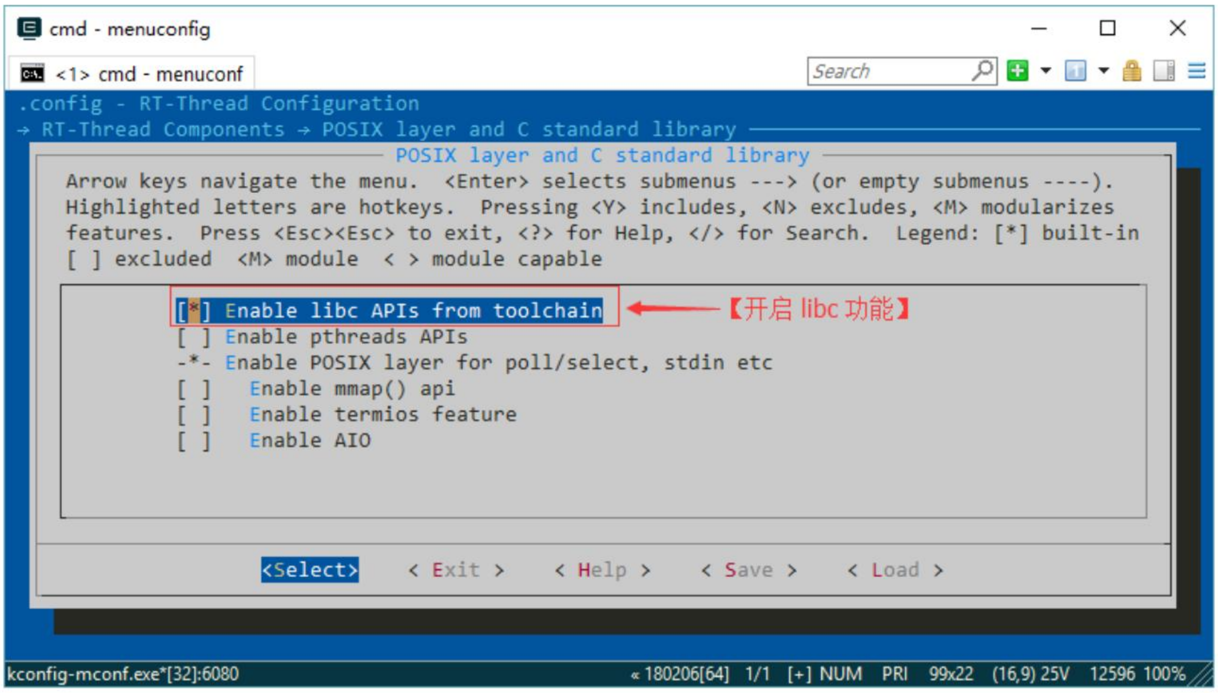


```
.config - RT-Thread Configuration
[...] ead Components → Device virtual file system → elm-chan's FatFs, Generic FAT Filesystem Module

                    Support long file name
  Use the arrow keys to navigate this window or press the
  hotkey of the item you wish to select followed by the <SPACE
  BAR>. Press <?> for additional information about this

    ( ) 0: LFN disable
    ( ) 1: LFN with static LFN working buffer
    ( ) 2: LFN with dynamic LFN working buffer on the stack
    (X) 3: LFN with dynamic LFN working buffer on the heap

                 【选择选项 3 即可】

             <Select>      < Help >

kconfig-mconf.exe*[32]:17368        « 180206[64]  1/1  [+] NUM  PRI  102x24  (42,18) 25V  12596 100%
```

Figure 5: Select Options 3

• Because you need to use some C library functions, you need to enable the libc function:

Machine Translated by Google



```
.config - RT-Thread Configuration
→ RT-Thread Components → POSIX layer and C standard library
                    POSIX layer and C standard library
  Arrow keys navigate the menu.  <Enter> selects submenus ---> (or empty submenus ----).
  Highlighted letters are hotkeys.  Pressing <Y> includes, <N> excludes, <M> modularizes
  features.  Press <Esc><Esc> to exit, <?> for Help, </> for Search.  Legend: [*] built-in
  [ ] excluded  <M> module  < > module capable

         [*] Enable libc APIs from toolchain
         [ ] Enable pthreads APIs
         -*- Enable POSIX layer for poll/select, stdin etc
         [ ]   Enable mmap() api
         [ ]   Enable termios feature
         [ ]   Enable AIO

         <Select>    < Exit >    < Help >    < Save >    < Load >
```

cmd - menuconfig
<1> cmd - menuconf
Search
【开启 libc 功能】
kconfig-mconf.exe*[32]:6080
« 180206[64] 1/1 [+] NUM PRI 99x22 (16,9) 25V 12596 100%

Figure 6: Open *libc*

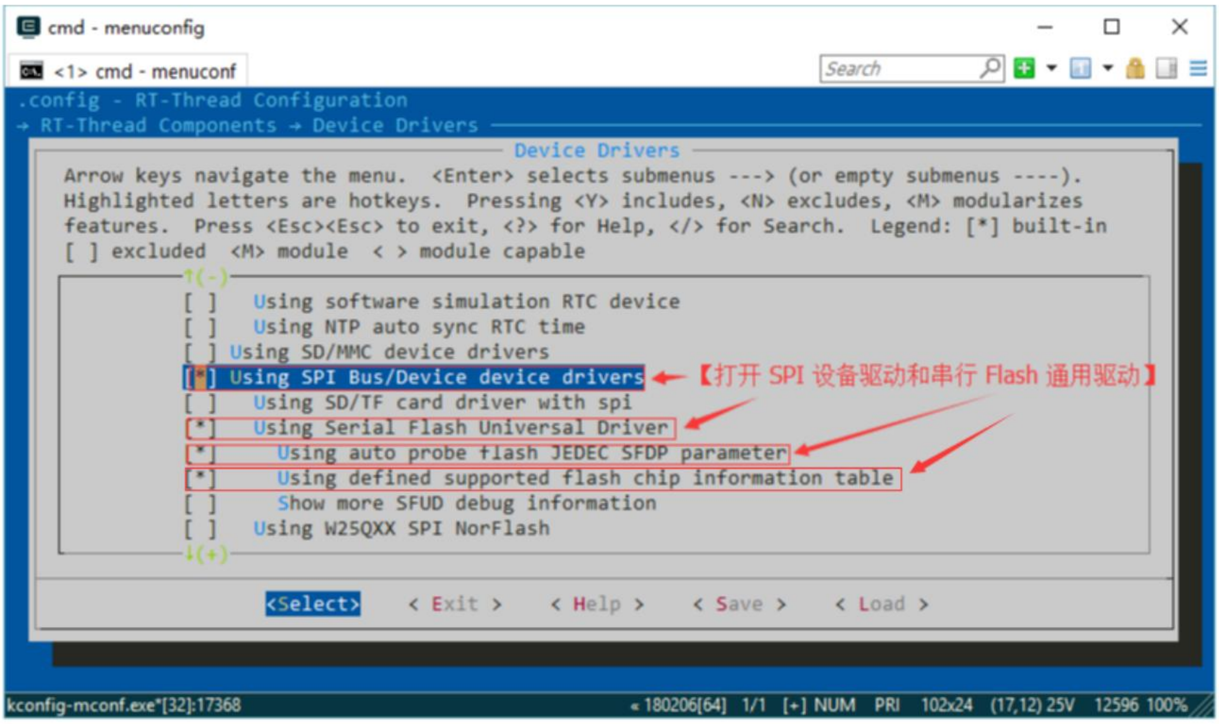
• Save the options and exit. At this point, elm FatFS has been added to the project.

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 the "SPI Device Application Notes" .

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

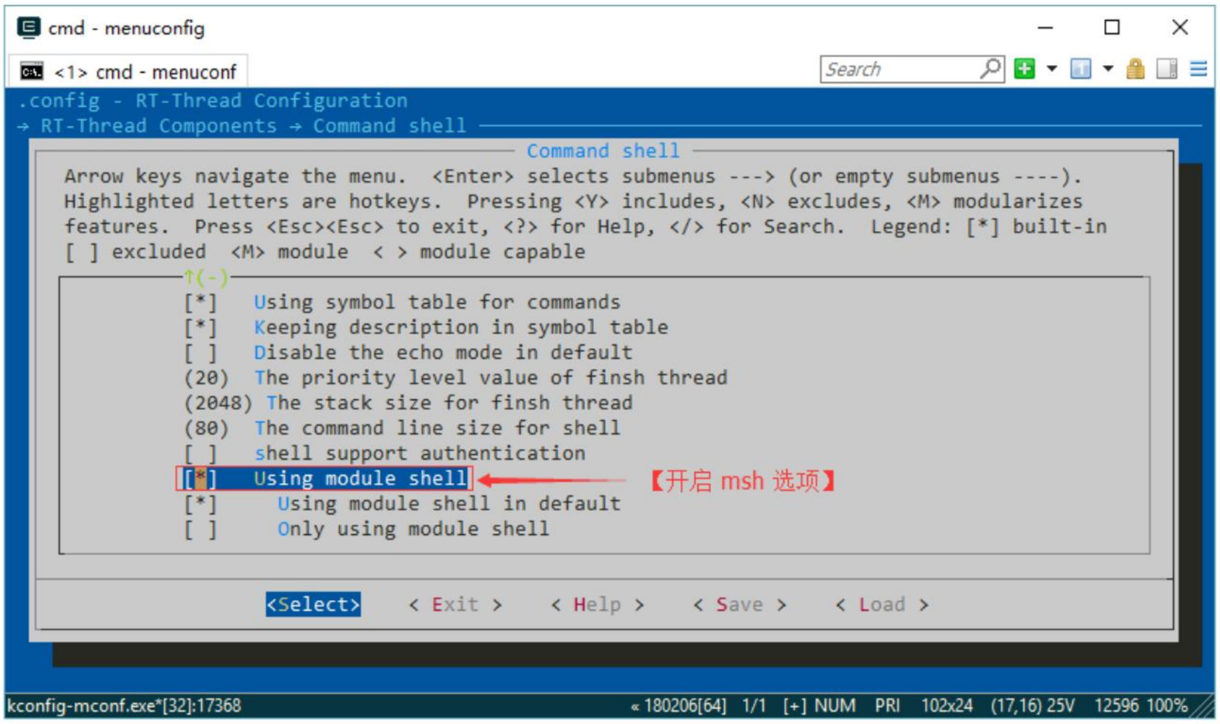


```
cmd - menuconfig
<1> cmd - menuconf
.config - RT-Thread Configuration
→ RT-Thread Components → Device Drivers
                         Device Drivers
  Arrow keys navigate the menu.  <Enter> selects submenus ---> (or empty submenus ----).
  Highlighted letters are hotkeys.  Pressing <Y> includes, <N> excludes, <M> modularizes
  features.  Press <Esc><Esc> to exit, <?> for Help, </> for Search.  Legend: [*] built-in
  [ ] excluded  <M> module  < > module capable
            ↑(-)
            [ ]   Using software simulation RTC device
            [ ]   Using NTP auto sync RTC time
            [ ] Using SD/MMC device drivers
            [*] Using SPI Bus/Device device drivers ← 【打开 SPI 设备驱动和串行 Flash 通用驱动】
            [ ]   Using SD/TF card driver with spi
            [*]   Using Serial Flash Universal Driver
            [*]     Using auto probe flash JEDEC SFDP parameter
            [*]     Using defined supported flash chip information table
            [ ]     Show more SFUD debug information
            [ ]   Using W25QXX SPI NorFlash
            ↓(+)
       <Select>    < Exit >    < Help >    < Save >    < Load >
kconfig-mconf.exe*[32]:17368        « 180206[64]  1/1  [+] NUM  PRI  102x24  (17,12) 25V  12596 100%
```

Figure 7: Open SPI drive

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

The module shell options are shown in the figure below:



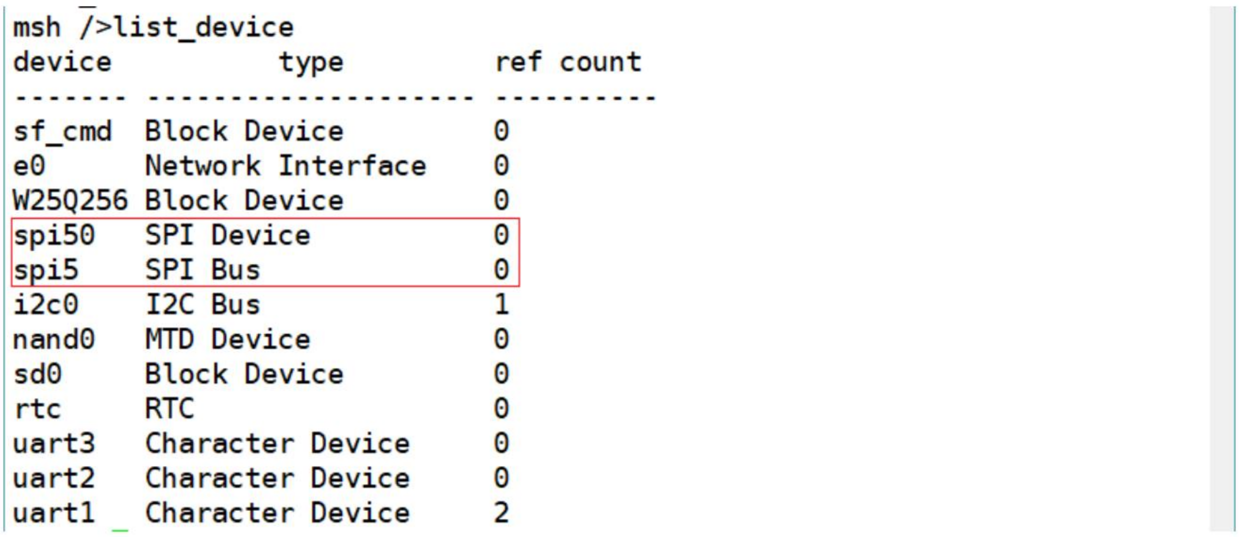
```
cmd - menuconfig
<1> cmd - menuconf
.config - RT-Thread Configuration
→ RT-Thread Components → Command shell
                         Command shell
  Arrow keys navigate the menu.  <Enter> selects submenus ---> (or empty submenus ----).
  Highlighted letters are hotkeys.  Pressing <Y> includes, <N> excludes, <M> modularizes
  features.  Press <Esc><Esc> to exit, <?> for Help, </> for Search.  Legend: [*] built-in
  [ ] excluded  <M> module  < > module capable
            ↑(-)
            [*]   Using symbol table for commands
            [*]   Keeping description in symbol table
            [ ]   Disable the echo mode in default
            (20)  The priority level value of finsh thread
            (2048) The stack size for finsh thread
            (80)  The command line size for shell
            [ ]   shell support authentication
            [*]   Using module shell ←  【开启 msh 选项】
            [*]     Using module shell in default
            [ ]     Only using module shell
       <Select>    < Exit >    < Help >    < Save >    < Load >
kconfig-mconf.exe*[32]:17368        « 180206[64]  1/1  [+] NUM  PRI  102x24  (17,16) 25V  12596 100%
```

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

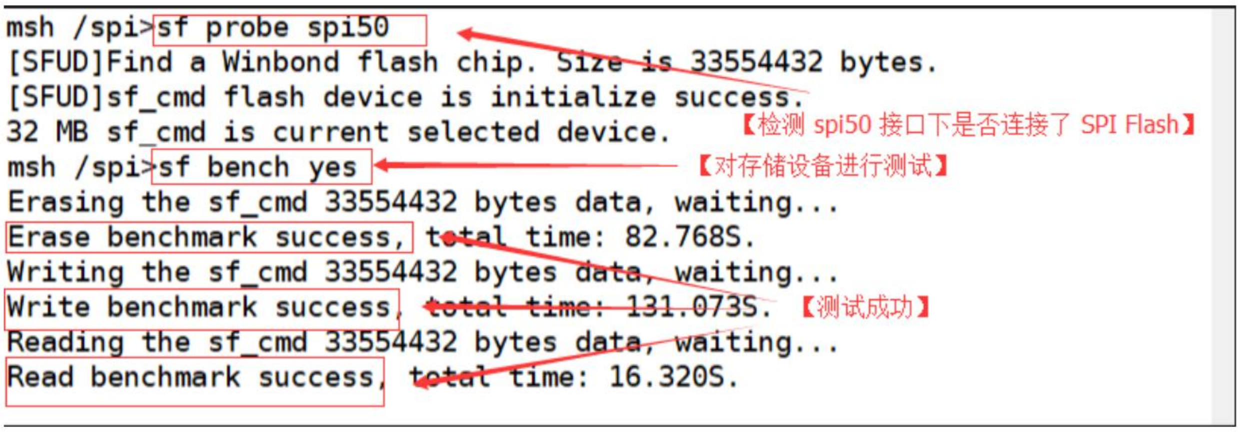
• On the stm32f429-apollo development board, the SPI Flash is connected to the SPI5 bus, and the corresponding SPI device is named spi50. Entering the list_device command in the terminal will show that the device type named spi50 is SPI Device, indicating that the SPI device has been successfully added. If the corresponding device does not appear, check the driver for errors.



```
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 the driver is functioning properly, you can use the sf command to benchmark the device . This function, provided by the sfud component, allows you to determine whether the storage device driver is functioning properly by checking the read, write, and erase functions of the storage device. If the test is successful, as shown in the figure below, the driver is considered to be functioning 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 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.
msh /spi>sf bench yes
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.
Reading the sf_cmd 33554432 bytes data, waiting...
Read benchmark success, total time: 16.320S.
```

Figure **10:** *benchmark* test

3.2.4.3. Create a storage device

• Since only block device type devices can be connected to the file system, it is necessary 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 already integrated this component, and we have enabled this feature in the above configuration process. At this point, 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 .

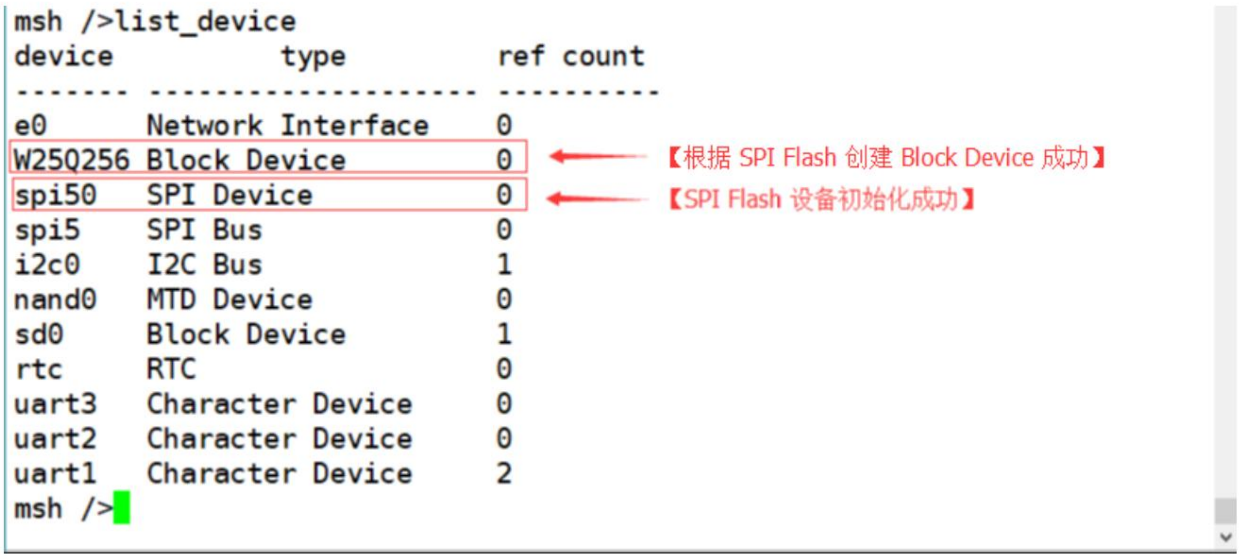
• If the component automatic initialization function is turned on, this function will be executed automatically, otherwise it needs to be called and run manually.

```
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 the block device has been

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

As shown in the figure below:



```
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         1
rtc      RTC                  0
uart3    Character Device     0
uart2    Character Device     0
uart1    Character Device     2
msh />
```

【根据 SPI Flash 创建 Block Device 成功】
【SPI Flash 设备初始化成功】

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. File system initialization

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

1. Initialize the DFS framework

2. Initialize the specific file system

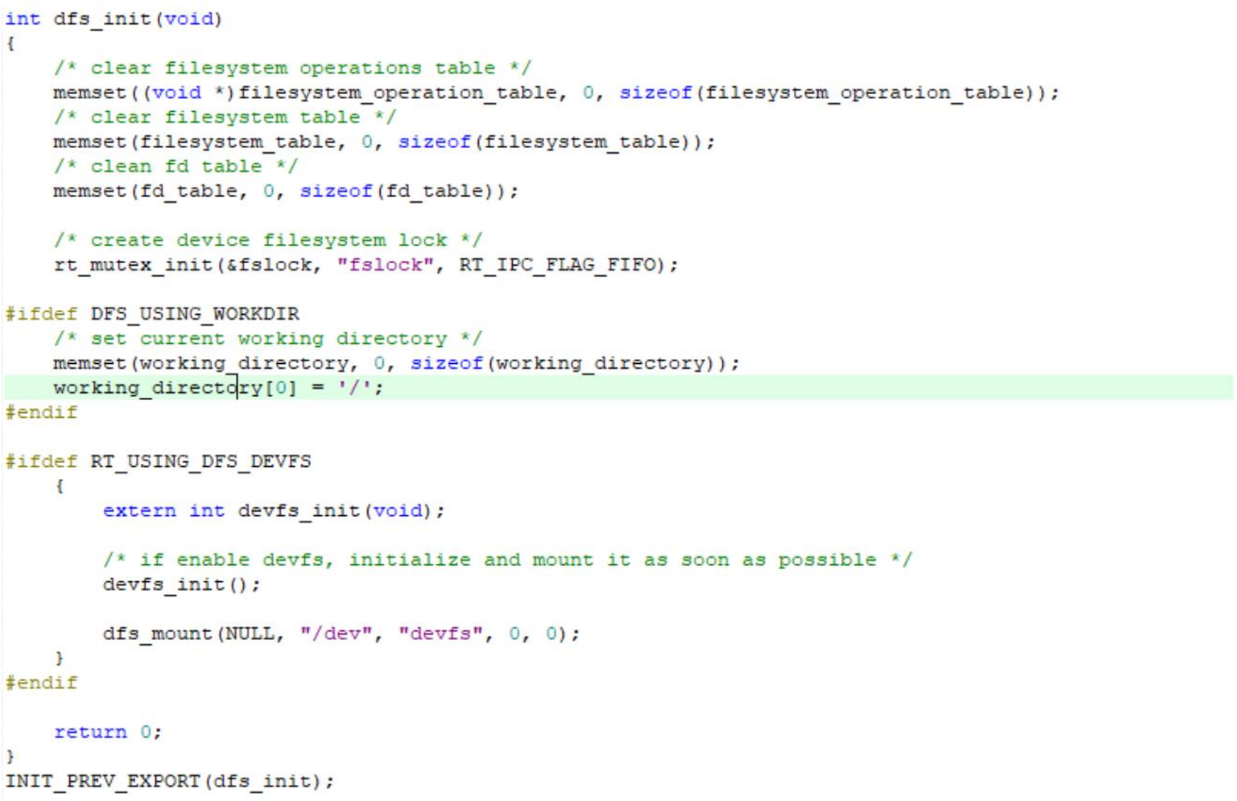
3. Initialize the storage device

Below we will explain the file system initialization process 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 initializing the data tables and mutexes required for 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.



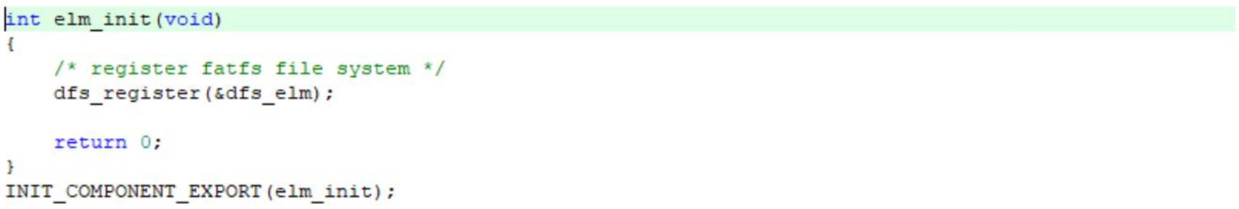
```
int dfs_init(void)
{
    /* clear filesystem operations table */
    memset((void *)filesystem_operation_table, 0, sizeof(filesystem_operation_table));
    /* clear filesystem table */
    memset(filesystem_table, 0, sizeof(filesystem_table));
    /* clean fd table */
    memset(fd_table, 0, sizeof(fd_table));

    /* 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 intermediate 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

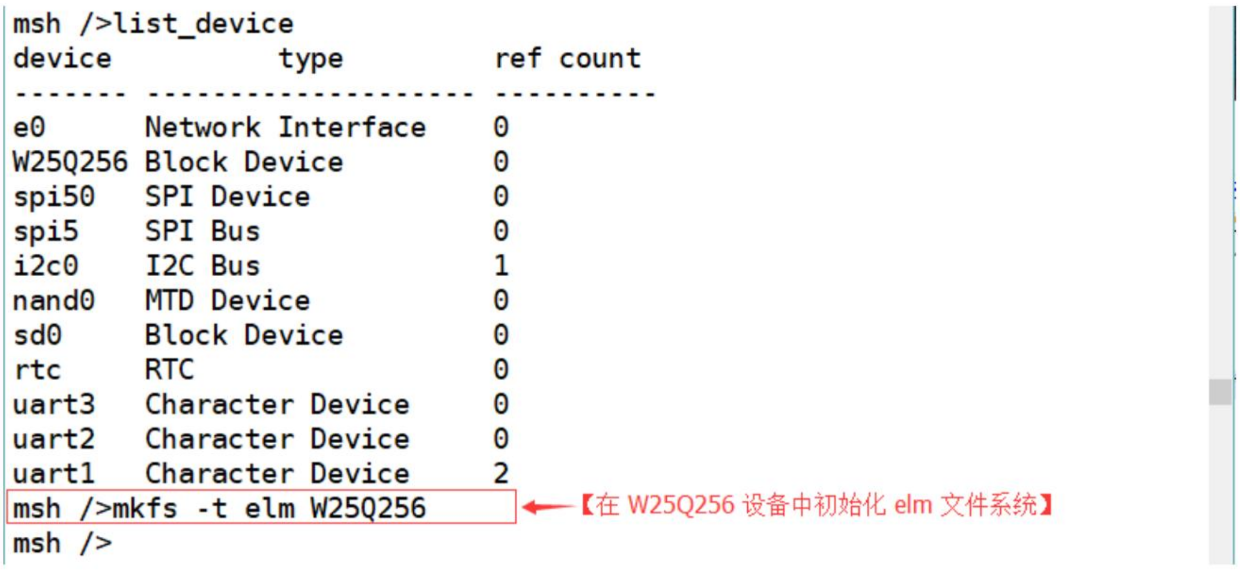
3.3.1.3. Storage device initialization For the initialization of storage devices, please refer to the "Creating Storage Devices" section.

Machine Translated by Google

3.3.2. Create a file system

• When using SPI Flash as a file system storage device 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. This is why we use the shell command "mkfs" to create a file system.

• 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] Before mounting a file system for the first time, you need to use the mkfs command to create the 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 />
```

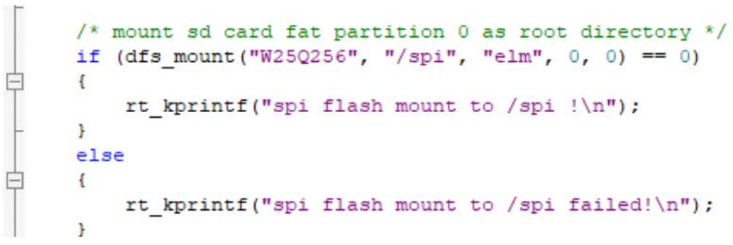
Figure 14: Creating a file system

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

3.3.3. Mounting the file system

Mounting a file system involves associating the 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 /spi folder can be mounted here because the root directory of the stm32f429-apollo BSP 's file system has already been mounted with RomFS and the /spi folder has been created . Unless otherwise specified, 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, mounted file system type, read-write flag, and file system private data. The usage is shown in the figure below:



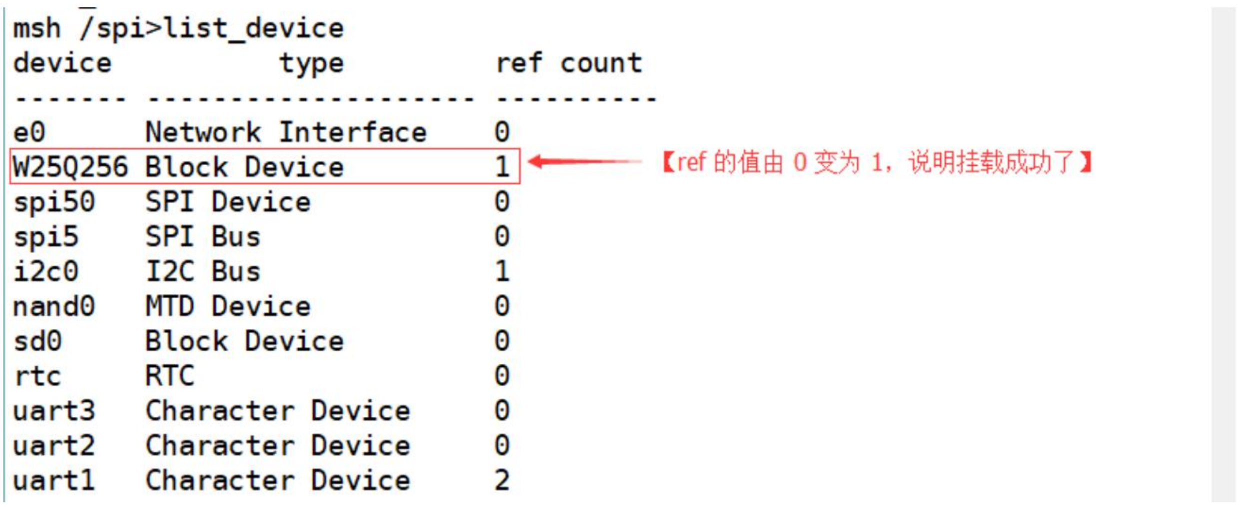
```
/* mount sd card fat partition 0 as root directory */
if (dfs_mount("W25Q256", "/spi", "elm", 0, 0) == 0)
{
    rt_kprintf("spi flash mount to /spi !\n");
}
else
{
    rt_kprintf("spi flash mount to /spi failed!\n");
}
```

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 below:



```
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
```

Figure **16:** View Mount

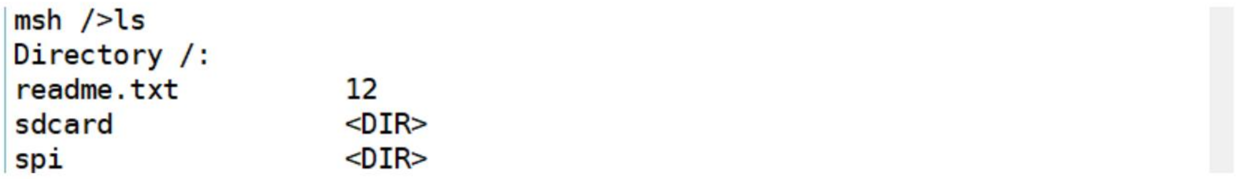
- At this point, the file system has been initialized and you can now operate on files and directories.

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:

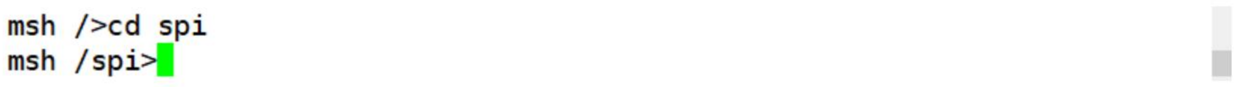


```
msh />ls
Directory /:
readme.txt              12
sdcard                  <DIR>
spi                     <DIR>
```

Figure **17:** *ls* Order

- **cd**

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

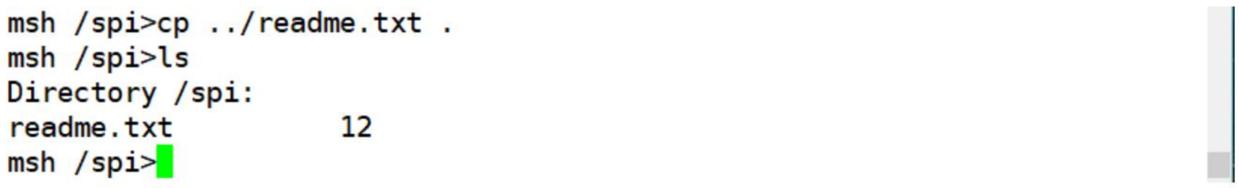


```
msh />cd spi
msh /spi>
```

Figure **18:** *cd* Order

- **cp**

  Function: copy file, the example is as follows:

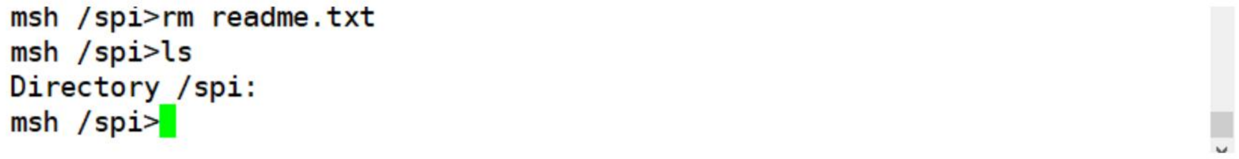


```
msh /spi>cp ../readme.txt .
msh /spi>ls
Directory /spi:
readme.txt          12
msh /spi>
```

Figure 19: *cp* Order

- **rm**

Function: Delete files or directories, as shown in the following figure:

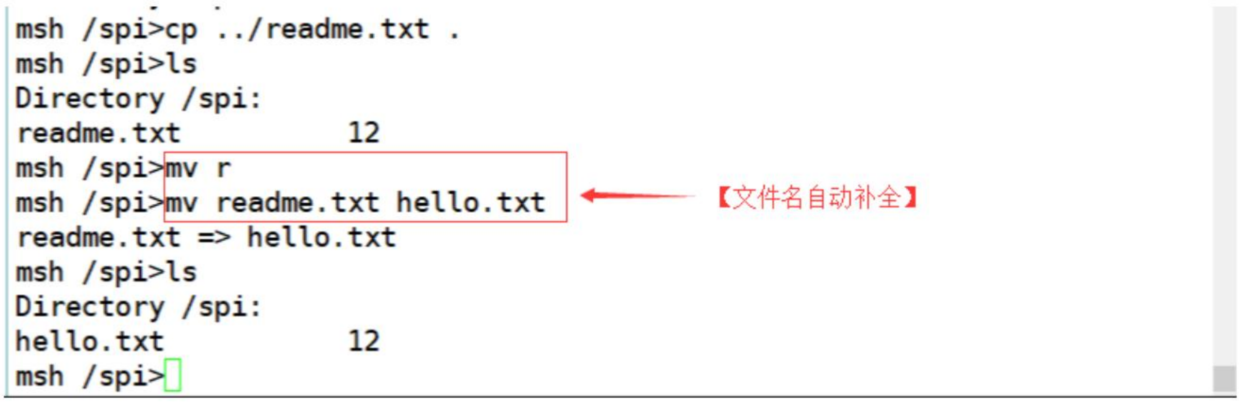


```
msh /spi>rm readme.txt
msh /spi>ls
Directory /spi:
msh /spi>
```

Figure 20: *rm* Order

- **mv**

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

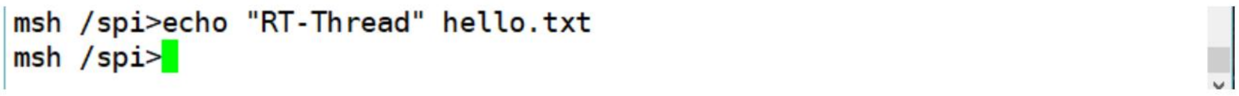


```
msh /spi>cp ../readme.txt .
msh /spi>ls
Directory /spi:
readme.txt          12
msh /spi>mv r
msh /spi>mv readme.txt hello.txt
readme.txt => hello.txt
msh /spi>ls
Directory /spi:
hello.txt           12
msh /spi>
```

Figure 21: *mv* Order

- **echo**

Function: Write the specified content to the file:

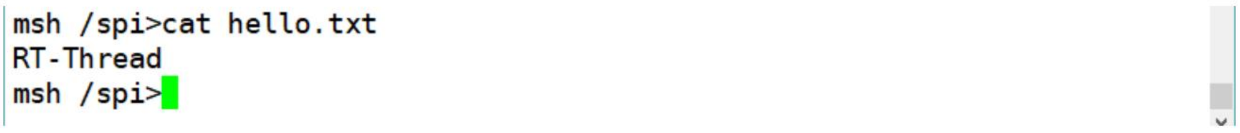


```
msh /spi>echo "RT-Thread" hello.txt
msh /spi>
```

Figure 22: *echo* Order

- **cat**

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

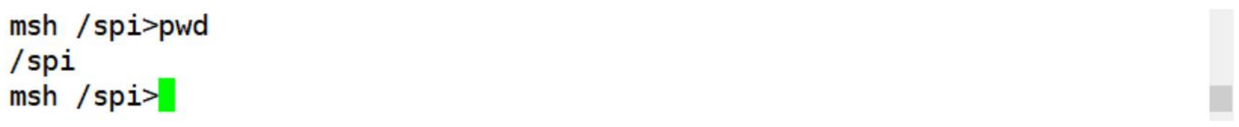


```
msh /spi>cat hello.txt
RT-Thread
msh /spi>
```

Figure 23: *cat* Order

• **pwd**

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

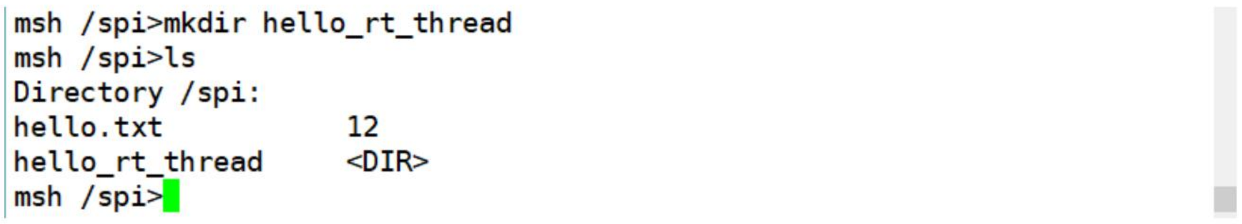


```
msh /spi>pwd
/spi
msh /spi>
```

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>
```

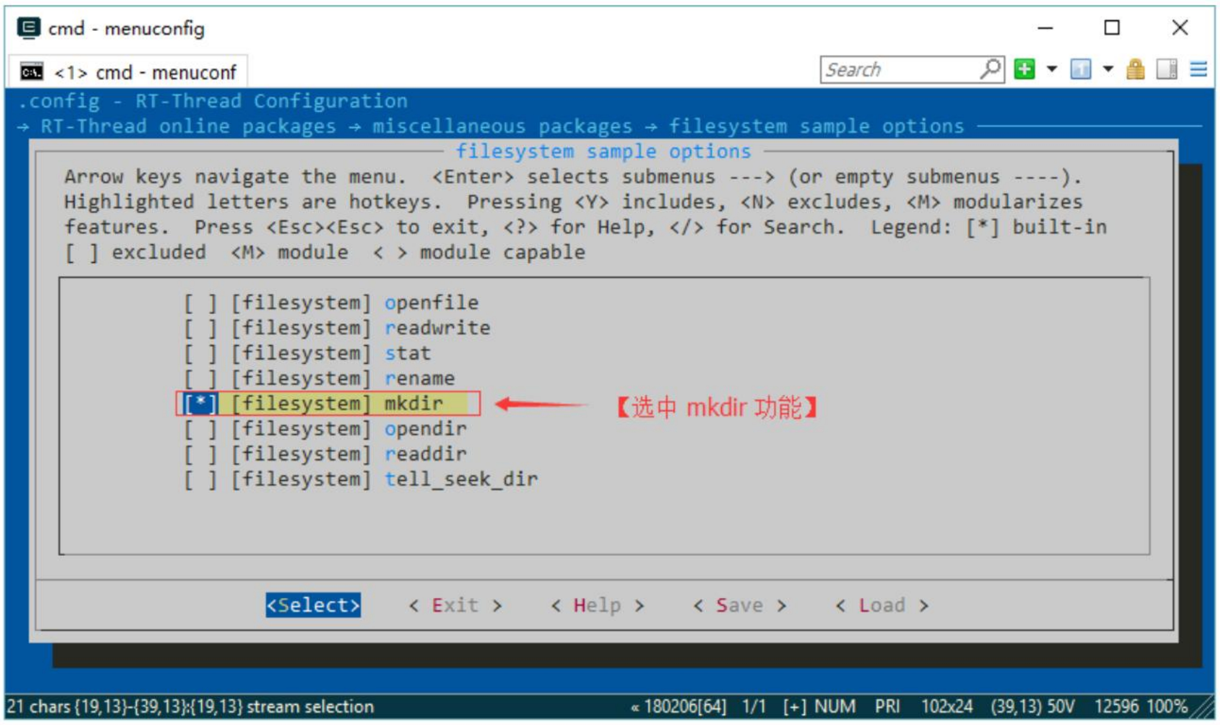
Figure **25:** *mkdir* Order

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:



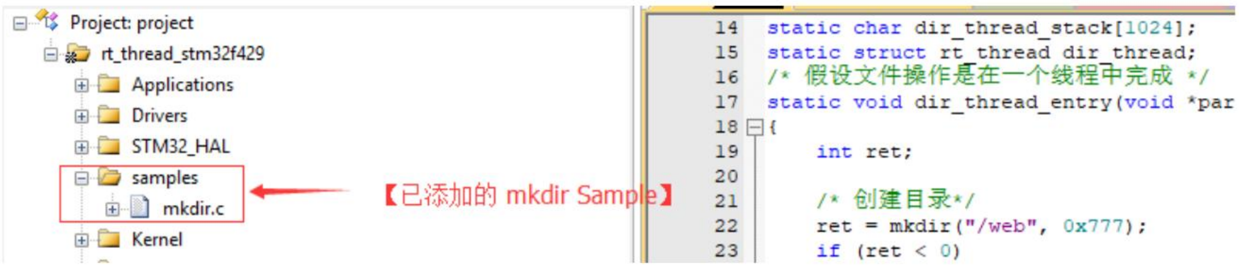
```
.config - RT-Thread Configuration
→ RT-Thread online packages → miscellaneous packages → filesystem sample options
                         filesystem sample options
  Arrow keys navigate the menu.  <Enter> selects submenus ---> (or empty submenus ----).
  Highlighted letters are hotkeys.  Pressing <Y> includes, <N> excludes, <M> modularizes
  features.  Press <Esc><Esc> to exit, <?> for Help, </> for Search.  Legend: [*] built-in
  [ ] excluded  <M> module  < > module capable

          [ ] [filesystem] openfile
          [ ] [filesystem] readwrite
          [ ] [filesystem] stat
          [ ] [filesystem] rename
          [*] [filesystem] mkdir
          [ ] [filesystem] opendir
          [ ] [filesystem] readdir
          [ ] [filesystem] tell_seek_dir

     <Select>    < Exit >    < Help >    < Save >    < Load >
```

cmd - menuconfig
<1> cmd - menuconf
Search
【选中 mkdir 功能】
21 chars {19,13}-{39,13}:{19,13} stream selection
« 180206[64] 1/1 [+] NUM PRI 102x24 (39,13) 50V 12596 100%

Figure **26:** Selected *mkdir* Function

• 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:

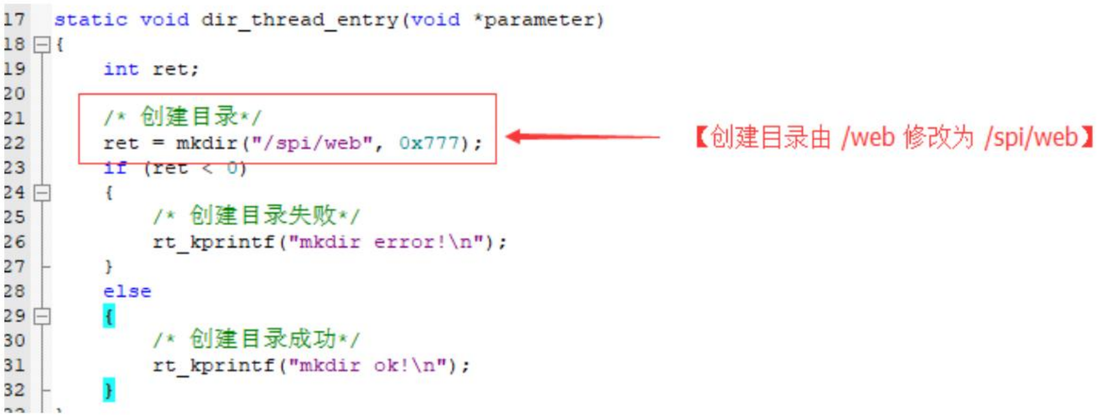


```
static char dir_thread_stack[1024];
static struct rt_thread dir_thread;
/* 假设文件操作是在一个线程中完成 */
static void dir_thread_entry(void *par
{
    int ret;

    /* 创建目录*/
    ret = mkdir("/web", 0x777);
    if (ret < 0)
```

Figure 27: Added *mkdir sample*

• It should be noted 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:

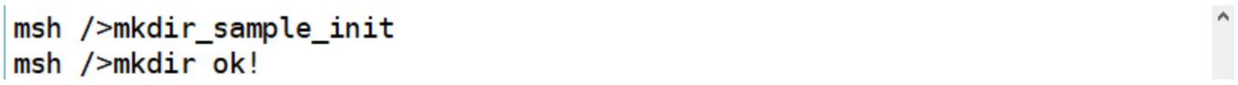


```
static void dir_thread_entry(void *parameter)
{
    int ret;

    /* 创建目录*/
    ret = mkdir("/spi/web", 0x777);
    if (ret < 0)
    {
        /* 创建目录失败*/
        rt_kprintf("mkdir error!\n");
    }
    else
    {
        /* 创建目录成功*/
        rt_kprintf("mkdir ok!\n");
    }
```

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

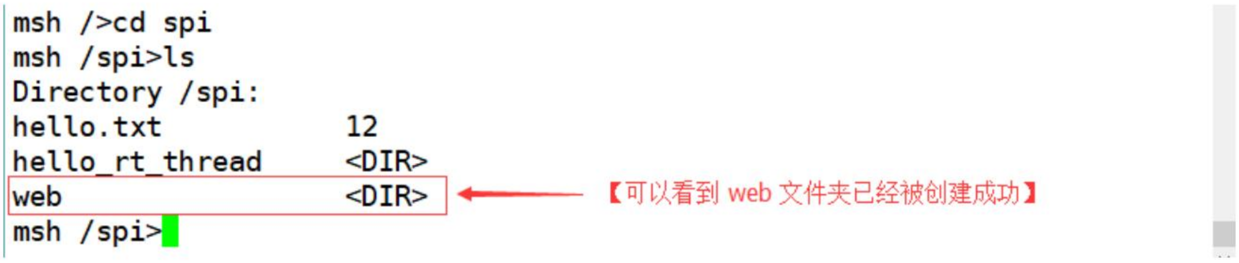
As shown:



```
msh />mkdir_sample_init
msh />mkdir ok!
```

Figure 29: Created successfully

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



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

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 above methods.

Machine Translated by Google

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. Refer to 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 to another path, such as ("/sdcard"), you need to ensure that the ("/sdcard") path exists. Otherwise, you need to create the sdcard folder in the root directory before the mount is successful.

• Check whether a file system has been created on the storage device. If there is no file system 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 for hardware pin configuration errors. • Check that the SPI device is registered. • Check that the SPI device is attached to the bus. • Check that the " Using **auto** probe flash JEDEC SFDP parameter" and "Using defined supported flash chip information table" options are selected under RT-Thread Components ÿ Device Drivers -> Using SPI Bus/Device device drivers -> Using Serial Flash Universal Driver . If not, enable these two options. For configuration diagrams, refer to the "Enabling the SPI Device Driver" section.

• If the storage device is still not recognized after turning on 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, there will be some differences. Generally, it can be set to 4K according to the requirements of the Flash device. Please fill in 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.

- 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 on SPI Flash, how can we keep some sectors unused 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

  Just assign different functions to the devices.

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

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

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

- You can use a bottom-up approach to troubleshoot the problem step by step.
- First check whether the storage device is successfully registered and functions normally.
- Check whether a file system has been created on the storage device.
- Checks that the specified file system types are registered with the DFS framework, and always checks that the number and types of allowed file systems are sufficient.
- Check whether DFS is initialized successfully. This initialization step 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.

5References

5.1 All related APIs in this article

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 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 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 flags
data	Private data of this file system

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