



# 网络编程基础

使用RT-Thread文件系统

# 目录

- 背景介绍
- SD卡挂载操作代码
- 常用命令展示
- 运行文件系统示例程序
- QEMU SD卡的读写
- 使用 RomFS
- 使用 RamFS



# 背景介绍

# 背景介绍

- 在早期的嵌入式系统中，需要存储的数据比较少，数据类型也比较单一，往往使用直接在存储设备中的指定地址写入数据的方法来存储数据。然而随着嵌入式设备功能的发展，需要存储的数据越来越多，也越来越复杂，这时仍使用旧方法来存储并管理数据就变得非常繁琐困难。因此我们需要新的数据管理方式来简化存储数据的组织形式，这种方式就是我们接下来要介绍的文件系统。

# 背景介绍

- 文件系统是一套实现了数据的存储、分级组织、访问和获取等操作的抽象数据类型(**Abstract data type**), 是一种用于向用户提供底层数据访问的机制。文件系统通常存储的基本单位是文件, 即数据是按照一个个文件的方式进行组织。当文件比较多时, 将导致文件繁多, 不易分类、重名的问题。而文件夹则作为一个容纳多个文件的容器而存在。

# 背景介绍

- DFS(Device virtual file system)是一种抽象的文件机制，RT-Thread中对文件系统的相关操作实际上都通过操作DFS实现，也就是说DFS是对各具体文件系统的抽象。DFS使得其他部分无须关心不同文件系统之间的差异，使得RT-Thread可以支持多种类型的文件系统。



# SD卡挂载操作代码

# SD卡挂载操作代码

- 挂载文件系统的源代码位于 `qemu-vexpress-a9\applications\mnt.c` 中。在实际代码中会将块设备 `sd0` 中的文件系统挂载到根目录 `/` 上。

```
1  #include <rtthread.h>
2
3  #ifdef RT_USING_DFS
4  #include <dfs_fs.h>
5
6  int mnt_init(void)
7  {
8      rt_thread_delay(RT_TICK_PER_SECOND);
9
10     if (dfs_mount("sd0", "/", "elm", 0, 0) == 0)
11     {
12         rt_kprintf("file system initialization done!\n");
13     }
14
15     return 0;
16 }
17 INIT_ENV_EXPORT(mnt_init);
18 #endif
```





# 常用命令展示

# ls: 查看当前目录信息

- 在挂载文件系统成功之后，就可以在 msh 中使用一些常用命令体验文件系统了。
- 我们在 QEMU 运行起来之后输入 **ls** 查看当前目录信息

```
\ | /  
- RT -   Thread Operating System  
/ | \  
3.1.1 build Sep 14 2018  
2006 - 2018 Copyright by rt-thread team  
lwIP-2.0.2 initialized!  
[I/SAL_SOC] Socket Abstraction Layer initialize success.  
[I/[SDIO]] SD card capacity 65536 KB.  
[I/[SDIO]] probe mmcblk0 block device!  
found part[0], begin: 32256, size: 63.992MB  
file system initialization done!  
hello rt-thread  
msh />ls  
Directory /:  
msh />
```

可以看到已经存在根目录了

# mkdir: 创建文件夹

- 我们输入命令 `mkdir rt-thread` 创建 `rt-thread` 文件夹
- 然后输入 `ls` 查看是否创建成功

```
msh />mkdir rt-thread  
msh />ls  
Directory /:  
rt-thread      <DIR>  
msh />
```

# echo: 将输入的字符串输出到指定输出位置

- 输入命令 `echo "hello rt-thread!!!"` 将输入的字符串输出到标准输出
- 输入命令 `echo "hello rt-thread!!!" hello.txt` 将输入的字符串输出到 `hello.txt`
- 然后输入命令 `ls` 查看是否存储成功

```
msh /: echo "hello rt-thread!!!"  
hello rt-thread!!!  
msh /: echo "hello rt-thread!!!" hello.txt  
msh />ls  
Directory /:  
rt-thread      <DIR>  
hello.txt      18  
msh />
```

# cat: 查看文件内容

- 输入命令 `cat hello.txt` 查看 `hello.txt` 文本文件的内容

```
msh />cat hello.txt  
hello rt-thread!!!msh />
```

# rm: 删除文件夹或文件

- 输入命令 `rm rt-thread` 删除 `rt-thread` 目录
- 输入命令 `rm hello.txt` 删除 `hello.txt` 文件
- 然后输入 `ls` 查看当前目录下的文件或目录

```
msh />ls
Directory /:
rt-thread      <DIR>
hello.txt      18
msh />rm rt-thread
msh />ls
Directory /:
hello.txt      18
msh />rm hello.txt
msh />ls
Directory /:
msh />
```



# 运行文件系统示例程序

# 运行文件系统示例程序

- 了解了文件系统的一些常用命令之后，下面带领大家通过运行文件系统的一些示例程序，来熟悉文件系统的基本操作。示例程序通过使用一些 **DFS** 的 **API** 接口来实现，并将示例导出到 **msh** 命令，通过运行示例程序并对照示例程序源码，有利于我们尽快上手操作文件系统。
- 文件系统的示例代码包含在 **RT-Thread samples** 软件包中，可以通过 **env** 配置将示例代码加入到项目中，路径如下所示。

```
RT-Thread online packages --->
miscellaneous packages --->
  samples: RT-Thread kernel and components samples --->
    [*] a filesystem_samples package for rt-thread --->
```



# 获取示例代码

- 将示例代码全部选中，然后退出保存并更新软件包即可将示例代码加入到工程里。

```
.config - RT-Thread Project Configuration
[...] kages → samples: kernel and components samples → a filesystem_samples package for rt-thread
a filesystem_samples package for rt-thread
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

--- a filesystem_samples package for rt-thread
    Version (latest) --->
    [*] [filesystem] openfile
    [*] [filesystem] readwrite
    [*] [filesystem] stat
    [*] [filesystem] rename
    [*] [filesystem] mkdir
    [*] [filesystem] opendir
    [*] [filesystem] readdir
    [*] [filesystem] tell_seek_dir
```

# 运行示例代码

- 在运行示例代码之前需要先输入 `scons` 编译一遍工程。
- 然后输入 `.\qemu.bat` 运行工程
- RT-Thread 启动完成之后，按 `TAB` 键查看 `msh` 命令，文件系统 `samples` 命令已经导出到 `msh`：

# 运行示例代码

- 在运行示例代码之前需要先输入 **scons** 编译一遍工程。

- 然后输入

- RT-Thread**

已经导出

```
\ | /
- RT -   Thread Operating System
/ | \   3.1.0 build Jul 19 2018
2006 - 2018 Copyright by rt-thread team
lwIP-2.0.2 initialized!
SD card capacity 65536 KB
probe mmcscd block device!
found part[0], begin: 32256, size: 63.992MB
file system initialization done!
hello rt-thread
msh />
RT-Thread shell commands:
openfile_sample - open file sample
readwrite_sample - readwrite sample
stat_sample     - show text.txt stat sample
rename_sample   - rename sample
mkdir_sample    - mkdir sample
opendir_sample  - open dir sample
readdir_sample  - readdir sample
telldir_sample  - telldir sample
memcheck        - check memory data
qemu-system-arm.exe*[32]:6140
```

mples 命令

# 运行示例代码

- 然后就可以输入命令运行相应的示例代码了。
- 例如：执行命令 `mkdir_sample` 的运行结果是

```
msh />mkdir_sample
mkdir ok!
msh />ls
Directory /:
dir_test      <DIR>
```

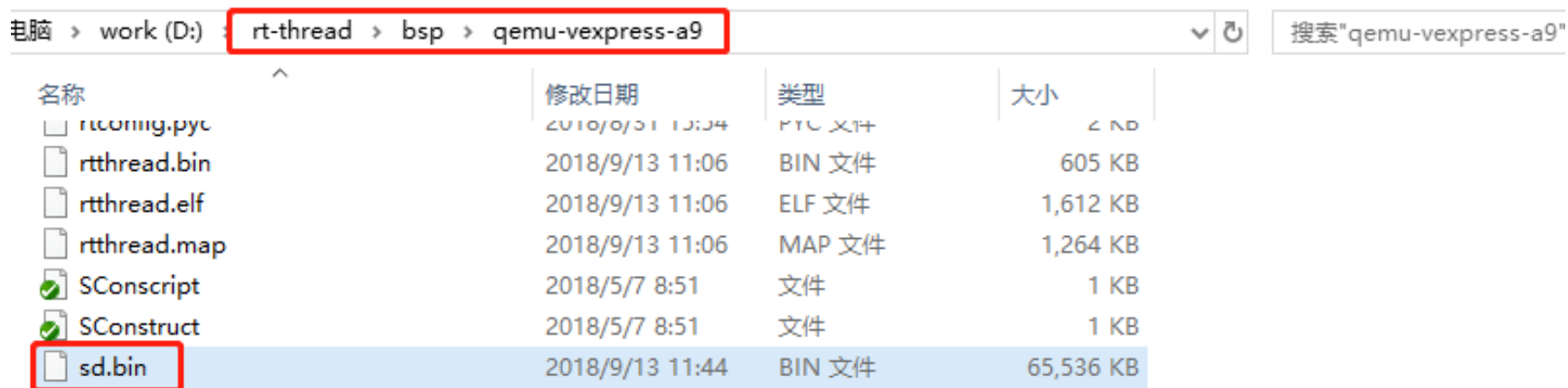
- 然后我们就可以对照这几个示例代码的源码来详细的了解文件系统 **API** 的用法了。



# QEMU SD卡的读写

# QEMU SD卡的读写

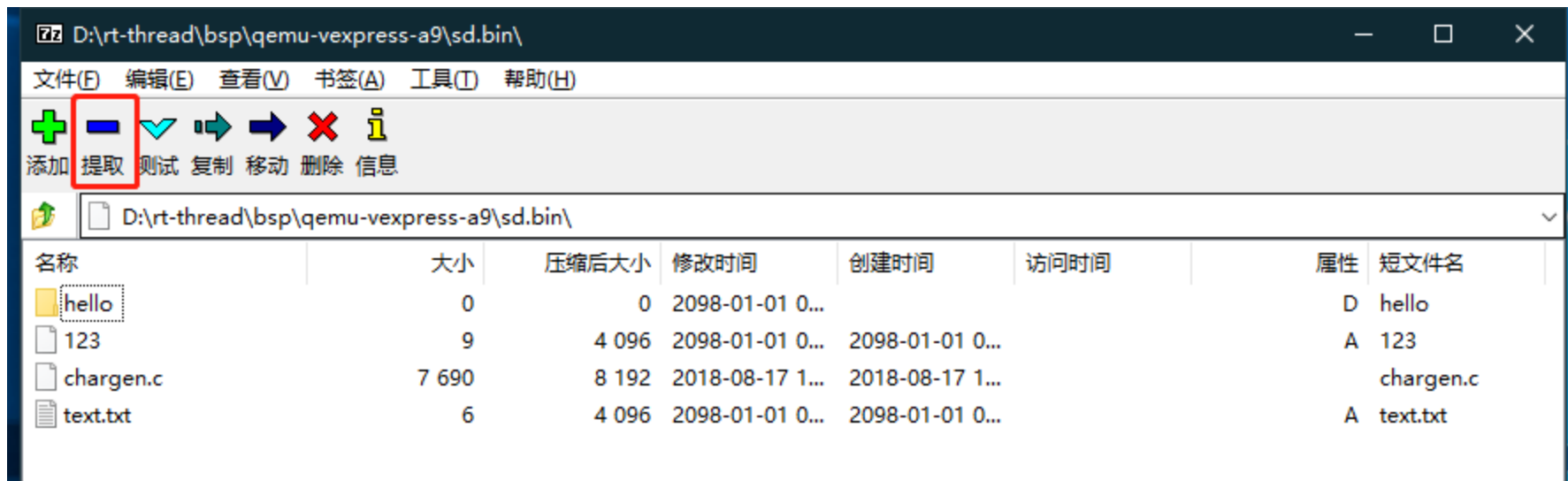
- QEMU 运行起来之后会在 bsp\qemu-vexpress-a9 目录下创建一个 sd.bin 文件。这是一个虚拟的 SD 卡，RT-Thread 默认的文件系统就是搭建在这个里面的。



名称	修改日期	类型	大小
rtconfig.py	2018/9/13 13:34	Python 文件	2 KB
rtthread.bin	2018/9/13 11:06	BIN 文件	605 KB
rtthread.elf	2018/9/13 11:06	ELF 文件	1,612 KB
rtthread.map	2018/9/13 11:06	MAP 文件	1,264 KB
SConscript	2018/5/7 8:51	文件	1 KB
SConstruct	2018/5/7 8:51	文件	1 KB
sd.bin	2018/9/13 11:44	BIN 文件	65,536 KB

# 读取 QEMU SD 卡的内容

- 因 sd.bin 本质上就是一个 FAT 文件系统的镜像文件，所以我们利用支持提取 FAT 格式的解压软件 7-Zip 就可以将 sd.bin 里的文件读取出来。



# 制作 QEMU SD 卡

- 在 env 工具的 tools/fatdisk 目录下有一个打包 FAT 格式文件的工具 fatdisk.exe，我们可以利用这个工具将我们要存储到QEMU SD卡里的文件打包成 sd.bin 文件。然后用新生成的 sd.bin 替换掉 bsp\qemu-vexpress-a9 目录下原来的 sd.bin 文件即可。
- 打开路径 env/tools/fatdisk 并在该目录下新建文件夹 sd

电脑 > work (D:) > env > tools > fatdisk >			
名称	修改日期	类型	大小
sd	2018/9/13 11:07	文件夹	
fatdisk.exe	2018/8/1 17:00	应用程序	145 KB
fatdisk.xml	2018/9/13 11:23	XML 文档	1 KB
fatdisk使用说明.docx	2018/8/1 17:00	Microsoft Word	33 KB
rofs.bin	2018/9/11 10:06	BIN 文件	276 KB



# 制作 QEMU SD 卡

- 打开上一步我们创建好的文件夹，放入我们需要存储到QEMU SD卡里的文件或文件夹

电脑 > work (D:) > env > tools > fatdisk > sd			
名称	修改日期	类型	大小
RT_Thread	2018/9/12 16:05	文件夹	
hello.txt	2018/9/12 16:06	文本文档	1 KB

# 制作 QEMU SD 卡

- 修改 env/tools/fatdisk 目录下 fatdisk.xml 文件为下面的内容

```
<?xml version="1.0" encoding="UTF-8"?>
<fatdisk>
  <disk_size>65536</disk_size>
  <sector_size>512</sector_size>
  <root_dir>sd</root_dir>
  <output>sd.bin</output>
  <strip>0</strip>
</fatdisk>
```

# 制作 QEMU SD 卡

- 在 `env/tools/fatdisk` 目录下右键打开 `env` 工具，输入命令 `fatdisk` 运行，就会在当前目录下生成 `sd.bin` 文件了。

```
11714@DESKTOP-EDPCUC0 D:\env\tools\fatdisk
> fatdisk
fatdisk version 1.0
disk info: size 64 MB, sector size 512 bytes
sd\hello.txt => /hello.txt
mkdir /RT_Thread
save to sd.bin...done!
```

- 然后用新生成的 `sd.bin` 替换掉 `bsp\qemu-vexpress-a9` 目录下原来的 `sd.bin` 文件

# 制作 QEMU SD 卡

- 运行 qemu 输入 ls 可以看到我们存储到QEMU SD卡里的文件和文件夹了。

```
\ | /  
- RT -   Thread Operating System  
/ | \  
2006 - 2018 Copyright by rt-thread team  
lwIP-2.0.2 initialized!  
[I/SAL_SOC] Socket Abstraction Layer initialize success.  
SD card capacity 65536 KB  
probe mmcblk0 block device!  
found part[0], begin: 32256, size: 63.992MB  
file system initialization done!  
hello rt-thread  
msh />ls  
Directory /:  
hello.txt          16  
RT_Thread          <DIR>  
msh />
```



# 使用 RomFS

# 使用 RomFS

- **RomFS** 是在嵌入式设备上常用的一种文件系统，具备体积小，可靠性高，读取速度快等优点，常用来作为系统初始文件系统。但也具有其局限性，**RomFS** 是一种只读文件系统。
- 不同的数据存储方式对文件系统占用空间，读写效率，查找速度等主要性能影响极大。**RomFS** 使用顺序存储方式，所有数据都是顺序存放的。因此 **RomFS** 中的数据一旦确定就无法修改，这是 **RomFS** 是一种只读文件系统的原因。也由于这种顺序存放策略，**RomFS** 中每个文件的数据都能连续存放，读取过程中只需要一次寻址操作，就可以读入整块数据，因此 **RomFS** 中读取数据效率很高。

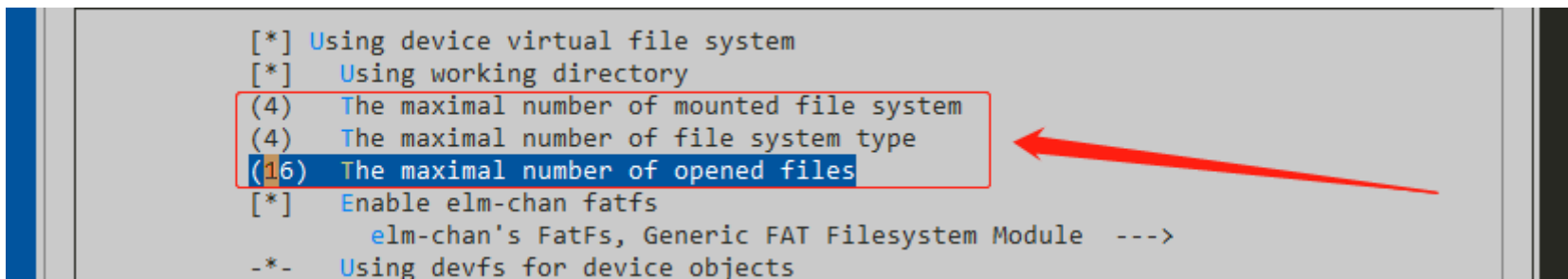
# 配置使能RomFS

- 开启 RT-Thread 对 RomFS 的支持，并调整最大支持的文件系统类型数目。
- 打开 menuconfig 菜单，保证 “RT-Thread Components” → “Device virtual file system” → “Enable ReadOnly file system on flash” 为开启状态：

```
-*- Using devfs for device objects
[*] Enable BSD socket operated by file system API
[*] Enable ReadOnly file system on flash
[*] Enable RAM file system
[ ] Enable UFFS file system: Ultra-low-cost Flash File System
[ ] Enable JFFS2 file system
[ ] Using NFS v3 client file system
```

# 配置使能RomFS

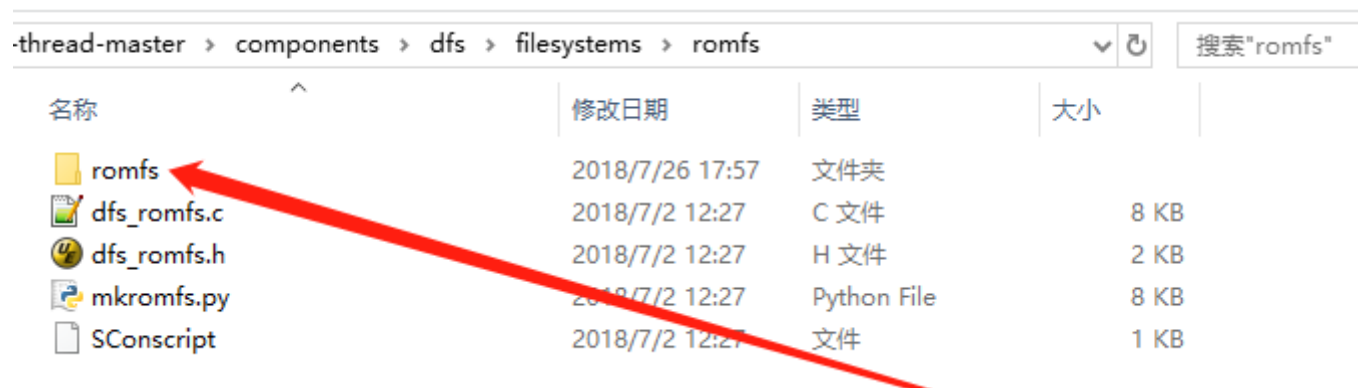
- 打开子菜单 "RT-Thread Components" → "Device virtual file system" 调整最大支持文件系统系统类型数:





# 生成 romfs.c 文件

- 由于 RomFS 是只读文件系统，所以需要放入到 RomFS 的文件都需要在系统运行前加入。我们可以将要存入 RomFS 中的文件数据放在 romfs.c 文件中，RT-Thread提供了制作 romfs.c 的 Python 脚本文件，根据用户需要加入到 RomFS 的文件和目录生成对应的数据结构。
- 打开路径 `rt-thread\components\dfs\filesystems\romfs` 并在该目录下新建文件夹 romfs

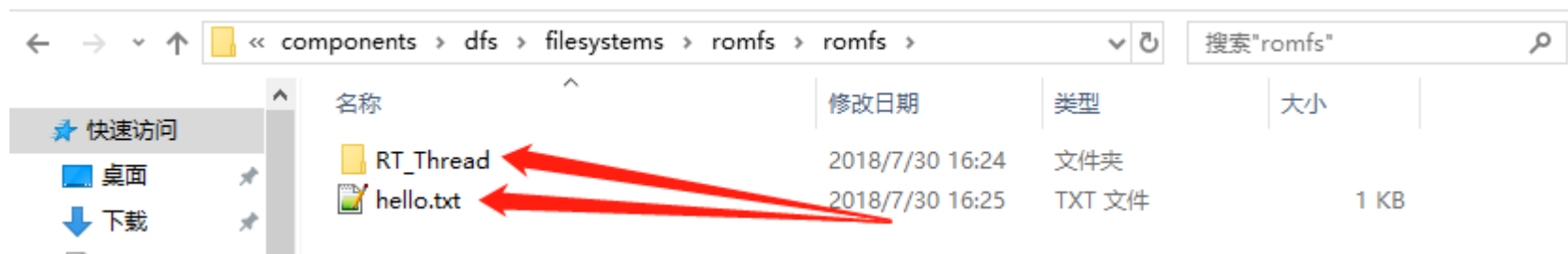


The screenshot shows a file explorer window with the path `rt-thread-master > components > dfs > filesystems > romfs`. The search bar contains "搜索"romfs"". The table below lists the files and folders in the current directory.

名称	修改日期	类型	大小
romfs	2018/7/26 17:57	文件夹	
dfs_romfs.c	2018/7/2 12:27	C 文件	8 KB
dfs_romfs.h	2018/7/2 12:27	H 文件	2 KB
mkromfs.py	2018/7/2 12:27	Python File	8 KB
SConscript	2018/7/2 12:27	文件	1 KB

# 生成 romfs.c 文件

- 打开上一步我们创建好的文件夹，放入我们需要在 RomFS 中放置的文件或文件夹。

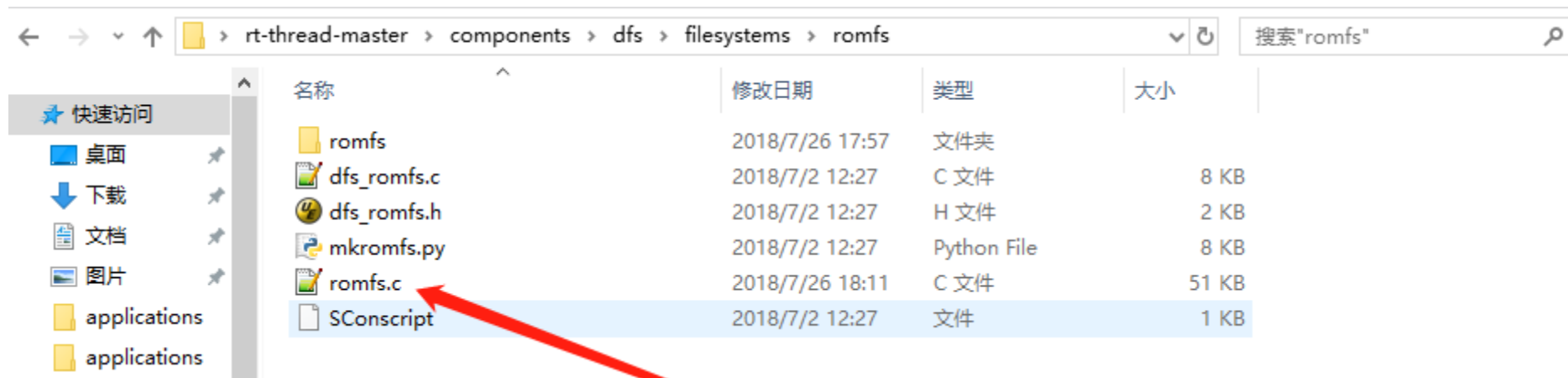


- 回到上一级目录 `rt-thread\components\dfs\filesystems\romfs`，在该目录下打开 `env` 工具，并运行命令

```
python mkromfs.py romfs romfs.c
```

# 生成 romfs.c 文件

- 可以看到目录下成功生成 romfs.c 文件：



# 挂载RomFS

- 在系统任务调度开始之后，通过 `dfs_mount()` 函数挂载 RomFS ， 在添加挂载函数的文件中需添加头文件 `#include "dfs_romfs.h"`
- 我们将 `qemu-vexpress-a9\applications\mnt.c` 文件中的内容替换成下面的代码，即可将 RomFS 挂载到根目录。

# 挂载RomFS

- 在系统任务调度函数的文件中
- 我们将 qemu 即可将 Romf

C mnt.c

```
1  #include <rtthread.h>
2
3  #ifdef RT_USING_DFS
4  #include <dfs_fs.h>
5  #include "dfs_romfs.h"
6
7  int mnt_init(void)
8  {
9      if (dfs_mount(RT_NULL, "/", "rom", 0, &(romfs_root)) == 0)
10     {
11         rt_kprintf("ROM file system initialized!\n");
12     }
13     else
14     {
15         rt_kprintf("ROM file system initialize failed!\n");
16     }
17
18     return 0;
19 }
20 INIT_ENV_EXPORT(mnt_init);
21 #endif
```

，在添加挂载

换成下面的代码，

# 预期结果

- 编译并运行工程之后，可以看到 RomFS 文件系统挂载成功，使用 ls 命令可以看到 RomFS 文件系统里面的文件夹和文件：

```
\ | /  
- RT -   Thread Operating System  
/ | \  
3.1.0 build Jul 30 2018  
2006 - 2018 Copyright by rt-thread team  
lwIP-2.0.2 initialized!  
ROM file system initialized!  
hello rt-thread  
msh />ls  
Directory /:  
RT_Thread      <DIR>  
hello.txt      21  
msh />
```



# 使用 RamFS

# 使用 RamFS

- **RamFS** 顾名思义是内存文件系统，它不能格式化，可以同时创建多个，在创建时可以指定其最大能使用的内存大小。其优点是读写速度很快，但存在掉电丢失的风险。如果一个进程的性能瓶颈是硬盘的读写，那么可以考虑在 **RamFS** 上进行大文件的读写操作。
- **RamFS** 使用链式存储方式，并且数据存储在内存空间，因此 **RamFS** 具备了可读写文件系统的特征，同时也拥有较快的读写速度。



# 配置使能RamFS

- 打开 menuconfig 菜单，保证 “RT-Thread Components” → “Device virtual file system” → “Enable RAM file system” 为开启状态：

```
[*] 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
[*] Enable RAM file system
[ ] Enable UFFS file system: Ultra-low-cost Flash File System
[ ] Enable JFFS2 file system
```

# 挂载RamFS

- 由于 RamFS 是在系统运行过程中动态创建的，所以在挂载之前我们应该先创建 RamFS，RT-Thread 提供了创建 RamFS 的 API 接口：

```
struct dfs_ramfs* dfs_ramfs_create(rt_uint8_t *pool, rt_size_t size)
```

参数	描述
pool	文件系统内存池地址
size	文件系统大小
返回	——
> 0	文件系统根目录对应的数据结构
< = 0	失败

# 挂载RamFS

- 在系统任务调度开始之后，通过 `dfs_mount()` 函数挂载 RamFS
- 我们将 `qemu-vexpress-a9\applications\mnt.c` 文件中的内容替换成下面的代码，即可将 RamFS 挂载到根目录。

# 挂载RamFS

- 在系统任务调度开始之后，通过 `dfs_mount()` 函数挂载 RamFS

- 我们将 `qe`  
即可将 `Ra`

下面的代码，

```
C mnt.c
1  #include <rtthread.h>
2
3  #ifdef RT_USING_DFS
4  #include <dfs_fs.h>
5
6  int mnt_init(void)
7  {
8      if (dfs_mount(RT_NULL, "/", "ram", 0, dfs_ramfs_create(rt_malloc(1024), 1024)) != 0)
9      {
10         rt_kprintf("RAM file system initialized!\n");
11     }
12     else
13     {
14         rt_kprintf("RAM file system initialize failed!\n");
15     }
16
17     return 0;
18 }
19 INIT_ENV_EXPORT(mnt_init);
20 #endif
--
```

# 预期结果

- 编译并运行工程之后, 可以看到 **RamFS** 文件系统挂载成功了。然后我们使用 **echo** 命令创建一个文件, 可以看到创建成功了。

```
\ | /
- RT -   Thread Operating System
/ | \    3.1.0 build Jul 31 2018
2006 - 2018 Copyright by rt-thread team
lwIP-2.0.2 initialized!
RAM file system initialized!
hello rt-thread
msh />ls
Directory /:
msh />echo "hello rt-thread!" hello.txt
msh />ls
Directory /:
hello.txt          16
msh />
```

qemu-system-arm.exe\*[32]:7344      α 180206[64] 1/1 [+] NUM PRI: 104x27 (7,32766) 25V 7928 100%