
使用 QEMU 运行 RT-THREAD 动态模块

RT-THREAD 文档中心

上海睿赛德电子科技有限公司版权 ©2019



WWW.RT-THREAD.ORG

Friday 28th September, 2018

目录

目录	i
1 本文的目的和结构	1
1.1 本文的目的和背景	1
1.2 本文的结构	1
2 准备工作	1
3 使能动态模块组件	1
3.1 配置工程	1
3.2 编译工程	3
3.3 运行动态模块命令	4
3.4 生成动态模块编译依赖环境	4
4 运行动态模块	5
4.1 运行最简单的动态模块	5
4.1.1. 创建动态模块	5
4.1.1.1. 获取示例	5
4.1.1.2. 设置环境变量	5
4.1.1.3. 编译动态模块	6
4.1.2. 将动态模块放入文件系统	7
4.1.2.1. 新建目录	7
4.1.2.2. 修改配置文件	7
4.1.2.3. 生成映像文件	7
4.1.3. 运行动态模块	8
4.2 动态模块的初始化和清理函数	9
4.2.1. 示例代码	10

	4.2.2. 运行结果	11
5	运行动态库	12
5.1	创建动态库	12
	5.1.1. 获取示例	12
	5.1.2. 编译动态库	12
5.2	运行动态库	13
	5.2.1. 添加示例代码	13
	5.2.2. 运行动态库	15
6	参考	16
7	常见问题	16

!!! abstract “摘要” 本应用笔记描述了在 Windows 平台使用 QEMU 运行 RT-Thread 动态模块。

1 本文的目的和结构

1.1 本文的目的和背景

RT-Thread 动态模块组件 `dlmodule` 提供了动态加载程序模块的机制。`dlmodule` 组件更多的是一个 ELF 格式加载器，把单独编译的一个 elf 文件的代码段，数据段加载到内存中，并对其中的符号进行解析，绑定到内核导出的 API 地址上。动态模块 elf 文件主要放置于 RT-Thread 下的文件系统上。

RT-Thread 的动态模块组件目前支持两种格式：

- `.mo` 则是编译出来时以 `.mo` 做为后缀名的可执行动态模块。它可以被加载，并且系统中会自动创建一个主线程执行这个动态模块中的 `main` 函数；同时这个 `main(int argc, char** argv)` 函数也可以接受命令行上的参数。
- `.so` 则是编译出来时以 `.so` 做为后缀名的动态库。它可以被加载，并驻留在内存中，并提供一些函数集由其他程序（内核里的代码或动态模块）来使用。

本文主要讲解了在 Windows 平台使用 QEMU 运行 RT-Thread 动态模块。

1.2 本文的结构

本文首先讲解了如何使能 RT-Thread 动态模块组件，然后讲解了如何基于 QEMU 运行动态模块及动态库。

2 准备工作

- [下载 RT-Thread 源码](#)，推荐下载 3.1.0 及以上版本。
- [下载 RT-Thread Env 工具](#)，推荐下载 1.0.0 及以上版本。
- [下载 rtthread-apps 源代码](#)。

3 使能动态模块组件

3.1 配置工程

在 Env 控制台切换到 `qemu-vexpress-a9` BSP 根目录，然后输入 `menuconfig` 命令打开配置菜单。

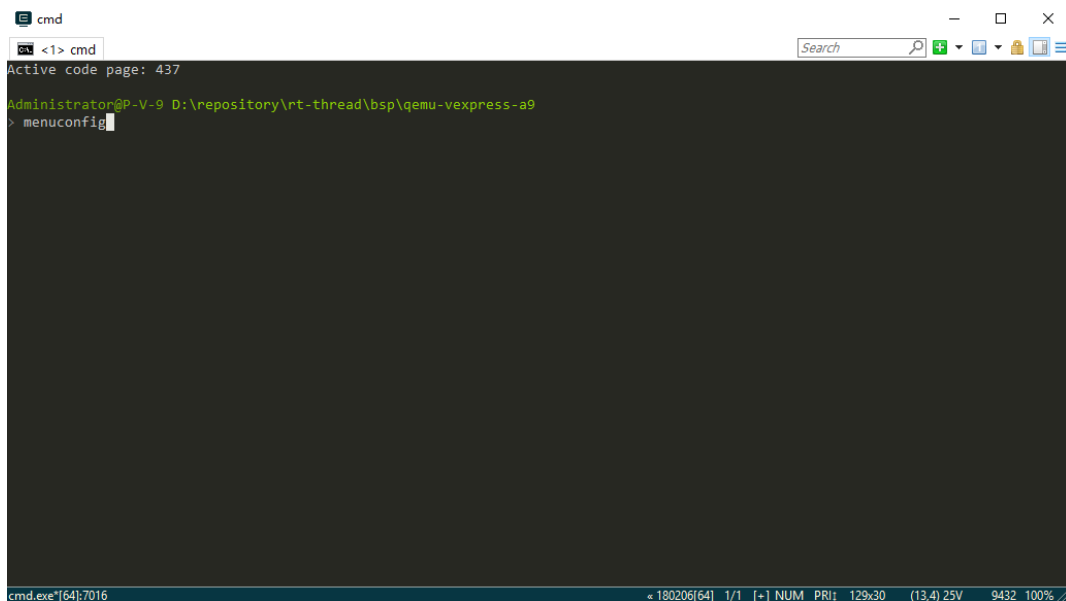


图 1: menuconfig 打开配置菜单

进入“RT-Thread Components → POSIX layer and C standard library”菜单，按下图箭头所示打开 libc 和动态模块的配置选项。

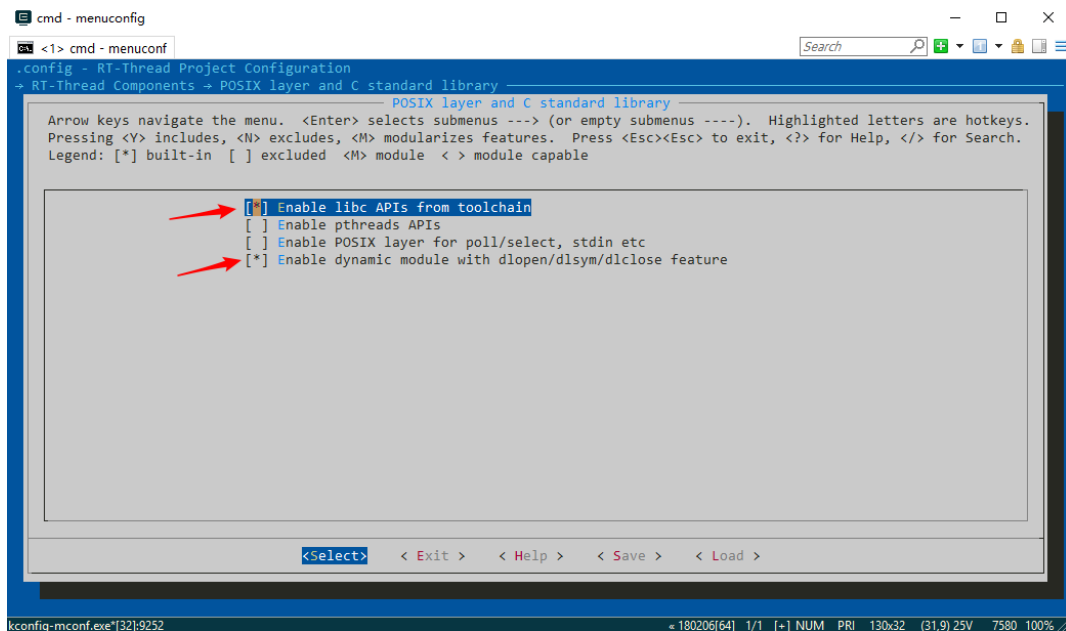


图 2: 开启动态模块

进入“RT-Thread Components → Device virtual file system”菜单打开文件系统的配置选项。退出 menuconfig 并保存配置。

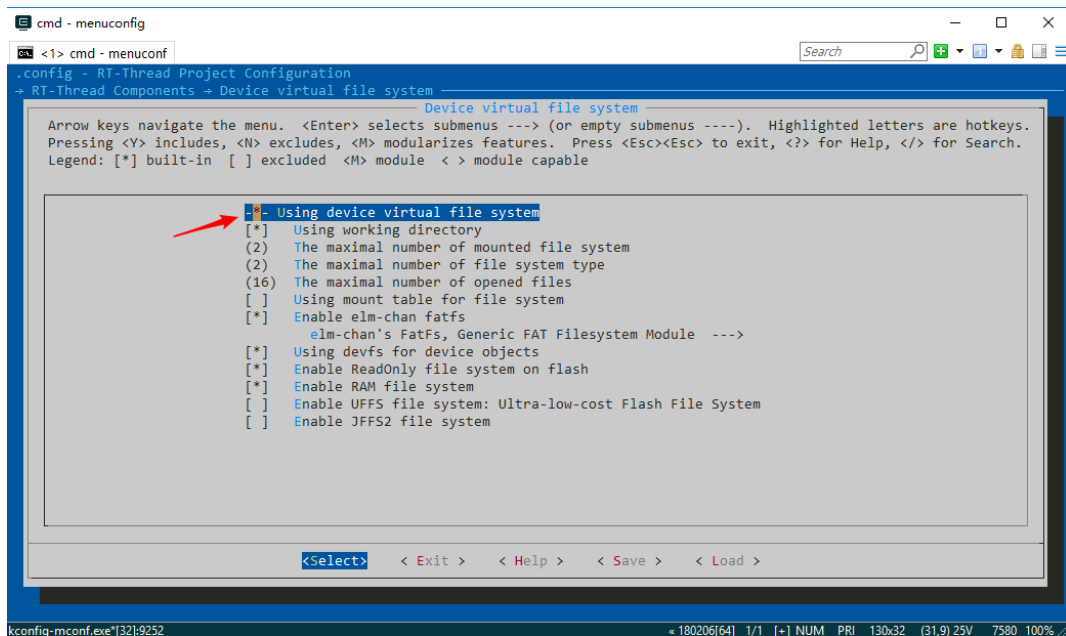


图 3: 开启文件系统

3.2 编译工程

使用 `scons` 命令编译工程。

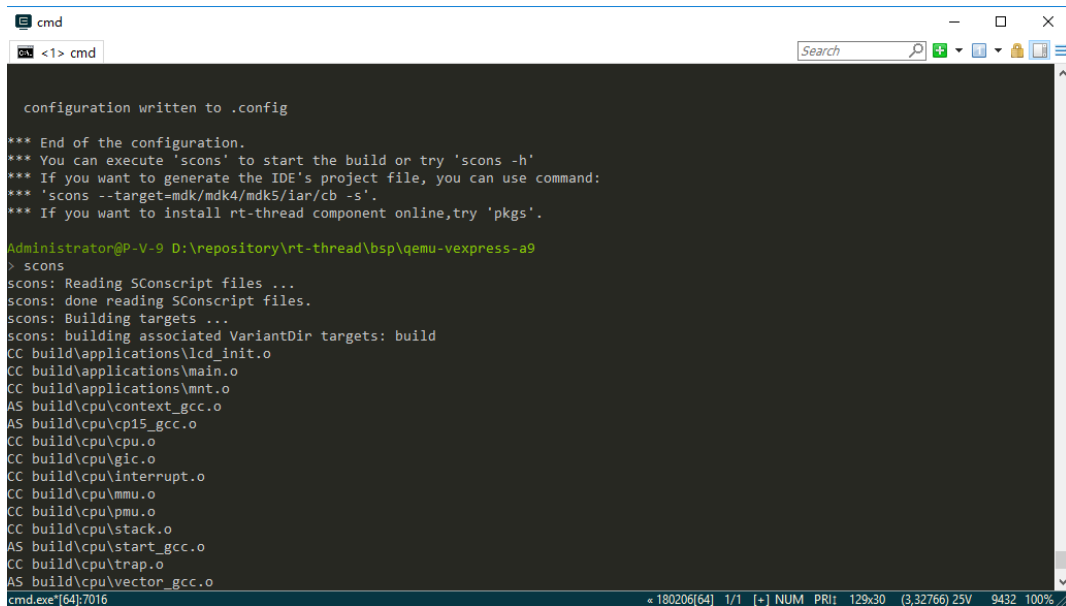
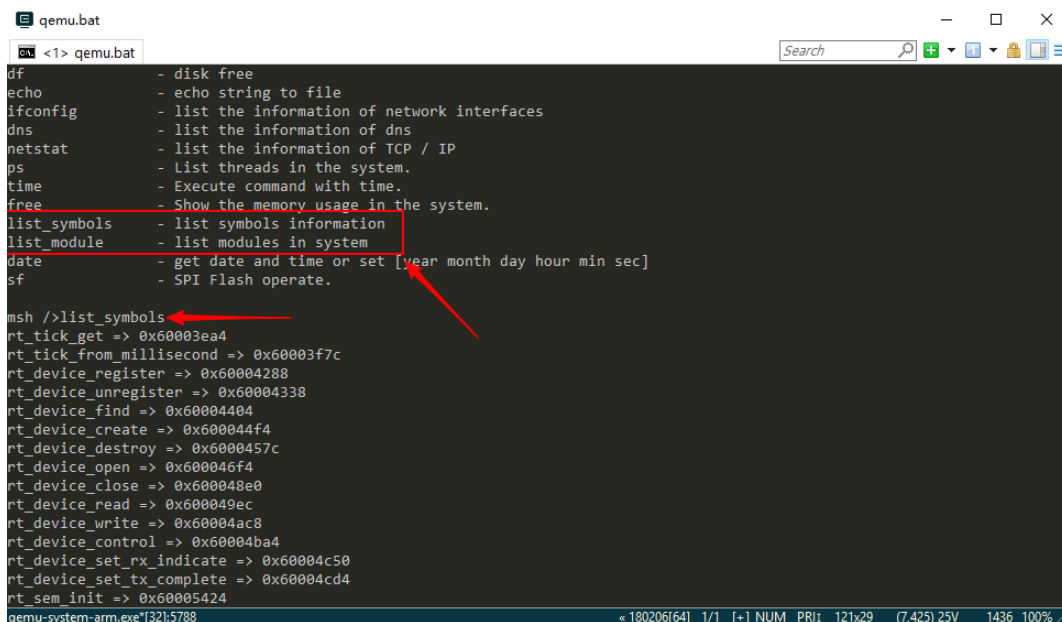


图 4: 编译工程

3.3 运行动态模块命令

编译完成后使用 `qemu.bat` 命令运行工程。按 Tab 键查看所有命令可以看到动态模块的两个命令 `list_module` 和 `list_symbols`，表明动态模块组件配置成功。

- `list_module` 命令可以查看当前正在运行的动态模块。
- `list_symbols` 命令可以查看动态模块可以使用的函数及其对应的内存地址。加载动态模块的时候会对其中的符号进行解析，并绑定到对应的函数地址上。



```

qemu.bat
<1> qemu.bat
df - disk free
echo - echo string to file
ifconfig - list the information of network interfaces
dns - list the information of dns
netstat - list the information of TCP / IP
ps - list threads in the system.
time - Execute command with time.
free - Show the memory usage in the system.
list_symbols - list symbols information
list_module - list modules in system
date - get date and time or set [year month day hour min sec]
sf - SPI Flash operate.

msh />list_symbols
rt_tick_get => 0x60003ea4
rt_tick_from_millisecond => 0x60003f7c
rt_device_register => 0x60004288
rt_device_unregister => 0x60004338
rt_device_find => 0x60004404
rt_device_create => 0x600044f4
rt_device_destroy => 0x6000457c
rt_device_open => 0x600046f4
rt_device_close => 0x600048e0
rt_device_read => 0x600049ec
rt_device_write => 0x60004ac8
rt_device_control => 0x60004ba4
rt_device_set_rx_indicate => 0x60004c50
rt_device_set_tx_complete => 0x60004cd4
rt_sem_init => 0x60005424
qemu-system-arm.exe[32]:5788
  
```

图 5: 运行动态模块命令

3.4 生成动态模块编译依赖环境

关闭运行的程序，在 Env 控制台使用 `scons --target=ua -s` 命令生成编译动态模块时需要包括的内核头文件搜索路径及全局宏定义。

```

cmd
<1> cmd
CC build\kernel\components\utilities\logtrace\log_file.o
CC build\kernel\components\utilities\logtrace\log_trace.o
CC build\kernel\src\clock.o
CC build\kernel\src\components.o
CC build\kernel\src\device.o
CC build\kernel\src\idle.o
CC build\kernel\src\ipc.o
CC build\kernel\src\irq.o
CC build\kernel\src\kservice.o
CC build\kernel\src\mem.o
CC build\kernel\src\memheap.o
CC build\kernel\src\mempool.o
CC build\kernel\src\object.o
CC build\kernel\src\scheduler.o
CC build\kernel\src\signal.o
CC build\kernel\src\thread.o
CC build\kernel\src\timer.o
LINK rtthread.elf
arm-none-eabi-objcopy -O binary rtthread.elf rtthread.bin
arm-none-eabi-size rtthread.elf
  text    data    bss     dec     hex filename
 654857   5220   58400  718477  af68d rtthread.elf
scons: done building targets.

Administrator@P-V-9 D:\repository\rt-thread\bsp\qemu-vexpress-a9
> scons --target=ua -s
D:\repository\rt-thread

Administrator@P-V-9 D:\repository\rt-thread\bsp\qemu-vexpress-a9
>
cmd.exe [64]: 7016
  
```

图 6: 生成动态模块编译依赖环境

4 运行动态模块

4.1 运行最简单的动态模块

4.1.1. 创建动态模块

4.1.1.1. 获取示例 下载 RT-Thread 动态模块工具库 [rtthread-apps](#), rtthread-apps 的 tools 目录放置了编译动态模块需要使用到的 Python 和 SConscript 脚本。hello 目录下的 main.c 是一个简单的动态模块使用示例, 源代码如下所示。

```

#include <stdio.h>

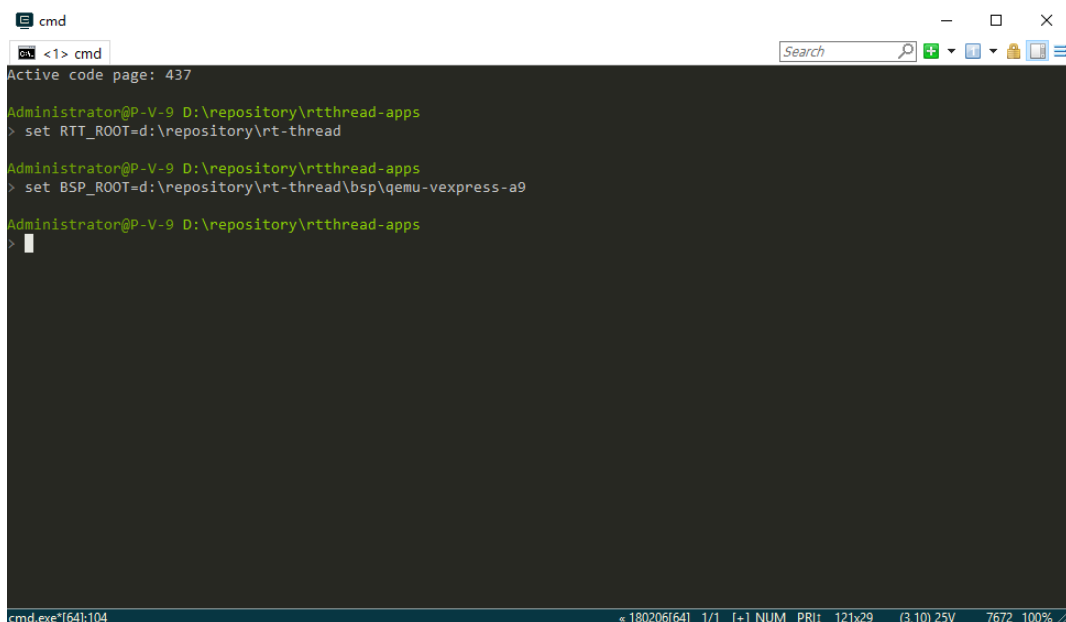
int main(int argc, char *argv[])
{
    printf("Hello, world\n");

    return 0;
}
  
```

这段代码实现了一个最简单的 main 函数, 打印字符串 “Hello world”。

4.1.1.2. 设置环境变量 在 Env 控制台切换到 rtthread-apps 根目录 (目录所在全路径不包含空格和中文字符), 然后通过下面 2 条命令设置环境变量。

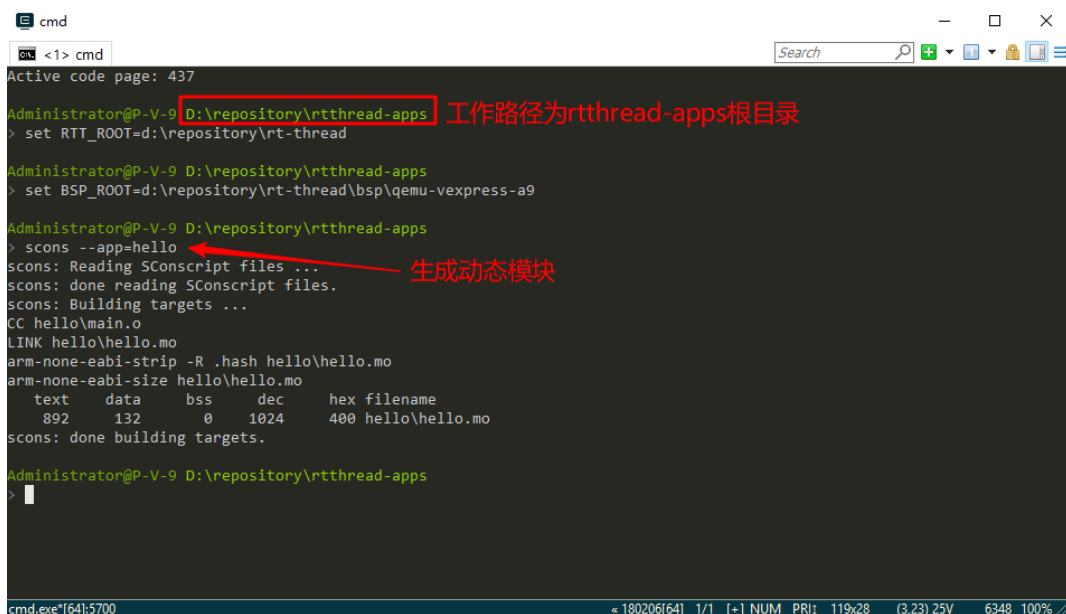
- set RTT_ROOT=d:\repository\rt-thread, 设置 RTT_ROOT 为 RT-Thread 源代码根目录。
- set BSP_ROOT=d:\repository\rt-thread\bsp\qemu-vexpress-a9, 设置 BSP_ROOT 为 qemu-vexpress-a9 BSP 根目录。



```
cmd
Active code page: 437
Administrator@P-V-9 D:\repository\rtthread-apps
> set RTT_ROOT=d:\repository\rt-thread
Administrator@P-V-9 D:\repository\rtthread-apps
> set BSP_ROOT=d:\repository\rt-thread\bsp\qemu-vexpress-a9
Administrator@P-V-9 D:\repository\rtthread-apps
>
```

图 7: 设置环境变量

4.1.1.3. 编译动态模块 使用 `scons --app=hello` 命令编译动态模块。



```
cmd
Active code page: 437
Administrator@P-V-9 D:\repository\rtthread-apps 工作路径为rtthread-apps根目录
> set RTT_ROOT=d:\repository\rt-thread
Administrator@P-V-9 D:\repository\rtthread-apps
> set BSP_ROOT=d:\repository\rt-thread\bsp\qemu-vexpress-a9
Administrator@P-V-9 D:\repository\rtthread-apps
> scons --app=hello 生成动态模块
scons: Reading SConscript files ...
scons: done reading SConscript files.
scons: Building targets ...
CC hello\main.o
LINK hello\hello.mo
arm-none-eabi-strip -R .hash hello\hello.mo
arm-none-eabi-size hello\hello.mo
  text  data   bss   dec    hex filename
   892   132     0  1024    400 hello\hello.mo
scons: done building targets.
Administrator@P-V-9 D:\repository\rtthread-apps
>
```

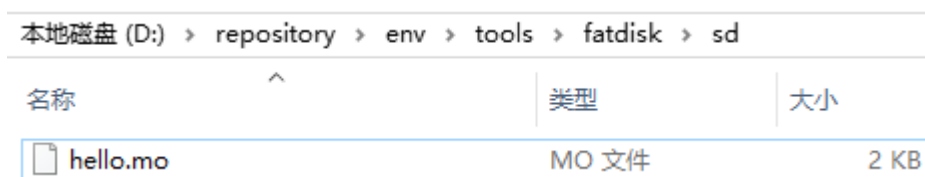
图 8: 编译动态模块

在 rtthread-apps/hello 目录下会生成动态模块文件 hello.mo。

4.1.2. 将动态模块放入文件系统

编译好的动态模块 hello.mo 需要放到文件系统下。qemu-vexpress-a9 BSP 会使用一个虚拟的 sd 卡设备 sd.bin，我们需要把动态模块放到这个虚拟的 sd 卡里面。对于物理设备来说，直接将动态模块添加到文件系统管理的存储设备中就可以。这里需要使用到 Env 工具里面的一个小工具 fatdisk，它位于 Env 的 tools 目录下，里面也提供了一份 fatdisk 的使用说明。这里使用 fatdisk 用于把 PC 上本地的一个目录转换成 sd.bin 映像文件，这个映像文件是做为一个 fat 文件系统而存在。

4.1.2.1. 新建目录 在 fatdisk 目录下新建一个 sd 目录，并复制刚刚编译的动态模块 hello.mo 文件到 sd 目录。



名称	类型	大小
hello.mo	MO 文件	2 KB

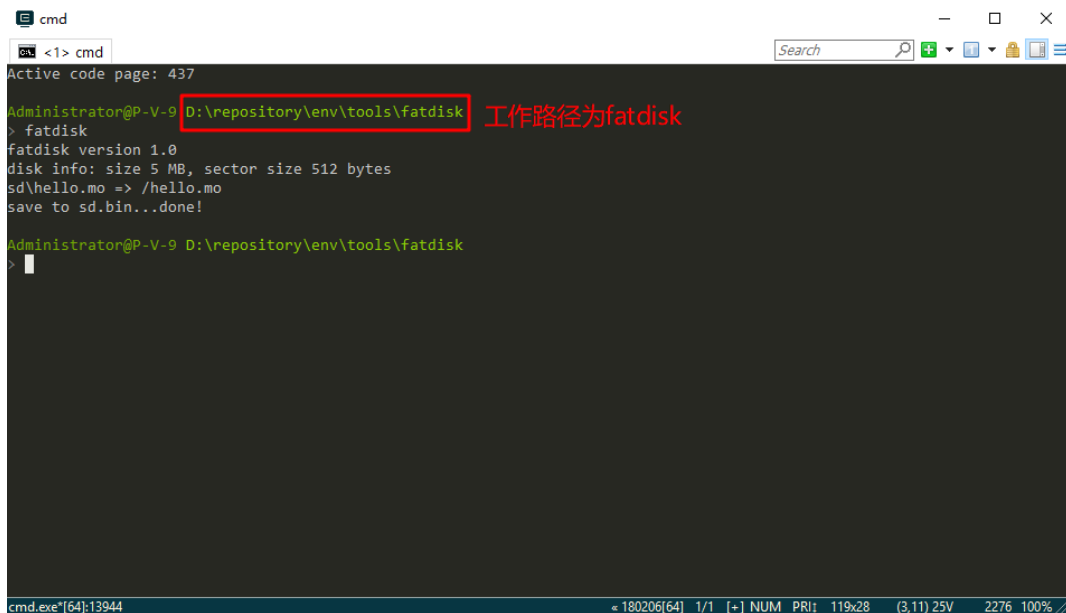
图 9: 增加 fatdisk 配置文件

4.1.2.2. 修改配置文件 按照下面的配置修改 fatdisk 目录下的配置文件 fatdisk.xml。

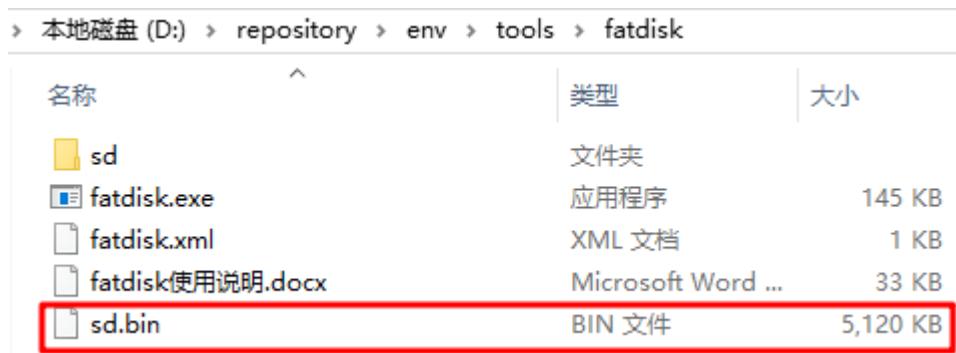
- 映像文件空间大小 disk_size 配置为了 5120Kbytes（大小可根据需要配置）。
- 映像文件的扇区大小 sector_size 需要配置为 512 KBytes。
- 要转换目录名 root_dir 配置为 sd，表示当前目录下的 sd 目录。
- 指定生成的映像文件名称 output 配置为 sd.bin。
- strip 需要配置为 0。

```
<?xml version="1.0" encoding="UTF-8"?>
<fatdisk>
  <disk_size>5120</disk_size>
  <sector_size>512</sector_size>
  <root_dir>sd</root_dir>
  <output>sd.bin</output>
  <strip>0</strip>
</fatdisk>
```

4.1.2.3. 生成映像文件 在 Env 控制台切换到 fatdisk 根目录，运行 fatdisk 命令则会按照配置文件 fatdisk.xml 中的配置，把里面指定的目录转换成 flash 映像文件。

图 10: 运行 *fatdisk* 命令

运行成功则会在 *fatdisk* 目录生成一个 *sd.bin* 文件，大小为 5MB。

图 11: 生成 *sd.bin* 文件

生成的映像文件 *sd.bin* 需要复制到 *qemu-vexpress-a9* BSP 目录。

4.1.3. 运行动态模块

在 Env 控制台切换到 *qemu-vexpress-a9* BSP 根目录输入 *qemu.bat* 命令运行工程。

```

cmd - qemu.bat
Administrator@V-9: D:\repository\rt-thread\bsp\qemu-vexpress-a9
> qemu.bat
WARNING: Image format was not specified for 'sd.bin' and probing guessed raw.
Automatically detecting the format is dangerous for raw images, write operations on block 0 will be restricted.
Specify the 'raw' format explicitly to remove the restrictions.

\ | /
- RT -      Thread Operating System
/ | \      3.1.1 build Sep 18 2018
2006 - 2018 Copyright by rt-thread team

SD card capacity 5120 KB
switching card to high speed failed
probe mmsd block device!
found part[0], begin: 32256, size: 4.992MB
file system initialization done!
hello rt-thread!
msh />ls
Directory /:
hello.mo      1368
msh />hello
msh />Hello, world

```

图 12: 运行 qemu 工程

- 系统运行起来后会看到文件系统初始化成功信息 “file system initialization done!”。
- 使用 `ls` 命令可以看到根目录下的动态模块文件 `hello.mo`。
- 输入 `hello` 命令运行动态模块 `hello.mo`。可以看到动态模块 `main` 函数打印的字符串 “Hello,world”

使用动态模块组件运行动态模块的主要原理如下图所示：

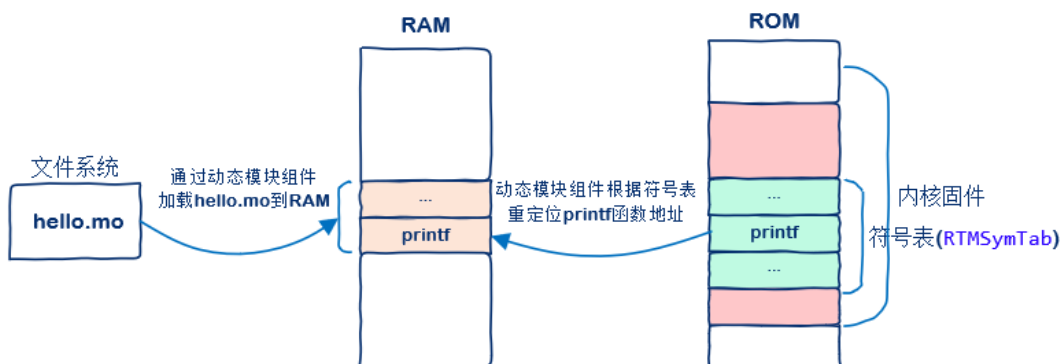


图 13: 动态模块的运行原理

4.2 动态模块的初始化和清理函数

动态模块组件提供了 2 个扩展的函数供用户使用，分别是 `module_init()` 和 `module_cleanup()`。

- `module_init()` 函数会在动态模块运行前被执行，用户可以根据需要做一些初始化工作。

- `module_cleanup()`函数会在动态模块运行结束后在 idle 线程里回调一次，执行用户设置的清理工作。

RT-Thread 系统会自动创建一个线程执行动态模块中的 `main` 函数，同时这个 `main(int argc, char* argv[])` 函数也可以接受命令行上的参数。这个线程默认的优先级等同空闲线程的优先级，线程堆栈默认为 2048 字节。用户可以在 `module_init()` 函数里修改这个线程的优先级和堆栈。

4.2.1. 示例代码

基于前面简单的动态模块示例代码 `main.c` 增加 `module_init()` 和 `module_cleanup()` 函数的使用，示例代码如下所示。

```
#include <stdio.h>
#include <dlmodule.h>

/* 动态模块的初始化函数 */
void module_init(struct rt_dlmodule *module)
{
    module->priority = 8;
    module->stack_size = 4096;

    printf("this is module %s initial function!\n", module->parent.name);
}

/* 动态模块的清理函数 */
void module_cleanup(struct rt_dlmodule *module)
{
    printf("this is module %s cleanup function!\n", module->parent.name);
}

int main(int argc, char *argv[])
{
    int i;

    printf("hello world from RTT::dynamic module!\n");

    /* 打印命令行参数 */
    for(i = 0; i < argc; i++)
    {
        printf("argv[%d]:%s\n", i, argv[i]);
    }

    return 0;
}
```

```
}

```

示例代码主要实现了如下功能：

- 在动态模块的初始化函数里可以设置这个线程的优先级和堆栈。
- 清理函数简单的打印信息。
- main 函数解析命令行参数并打印出来。

请参考前面小节将此示例代码生成的动态模块文件放到文件系统里，并将生成的映像文件 sd.bin 复制到 qemu-vexpress-a9 BSP 目录。

4.2.2. 运行结果

在 Env 控制台切换到 qemu-vexpress-a9 BSP 根目录输入 `qemu.bat` 命令运行工程。

```

Administrator@P-V-9 D:\repository\rt-thread\bsp\qemu-vexpress-a9
> qemu.bat
WARNING: Image format was not specified for 'sd.bin' and probing guessed raw.
Automatically detecting the format is dangerous for raw images, write operations on block 0 will be restricted.
Specify the 'raw' format explicitly to remove the restrictions.

\ | /
- 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.
SD card capacity 5120 KB
switching card to high speed failed
probe mmcblk0 block device!
found part[0], begin: 32256, size: 4.992MB
file system initialization done!
hello rt-thread
msh />ls
Directory /:
hello.mo      1864
msh />hello this is rt-thread!
this is module hello initial function!
hello world from RTT::dynamic module!
argv[0]:hello
argv[1]:this
argv[2]:is
argv[3]:rt-thread!
msh />this is module hello cleanup function!

qemu-system-arm.exe[32]:11148  = 180206[64] 1/1 [+] NUM PRI: 121x31 (1,486) 25V 1436 100%

```

图 14: 运行 qemu 工程

- 系统运行起来后会看到文件系统初始化成功信息 “file system initialization done!”。
- 使用 `ls` 命令可以看到根目录下的动态模块文件 `hello.mo`。
- 输入 `hello this is rt-thread!` 命令运行动态模块 `hello.mo`。hello 后面的字符串为参数。
- 执行到动态模块初始化函数 `module_init` 时会打印字符串 “this is module hello initial function!”。
- 执行动态模块的 main 函数时会打印字符串 “hello world from RTT::dynamic module!”, 命令行参数也依次打印了出来。
- 动态模块运行结束后又执行清理函数 `module_cleanup`, 打印字符串 “this is module hello cleanup function!”。

5 运行动态库

5.1 创建动态库

5.1.1. 获取示例

下载 RT-Thread 动态模块工具库 [rtthread-apps](#), rtthread-apps 的 lib 目录下有一个简单的动态库示例的 lib.c, 源代码如下所示, 它实现了 2 个简单的函数供使用。

```
#include <stdio.h>

int lib_func(void)
{
    printf("hello world from RTT::dynamic library!\n");

    return 0;
}

int add_func(int a, int b)
{
    return (a + b);
}
```

5.1.2. 编译动态库

编译动态库之前需要先设置环境变量。然后使用 `scons --lib=lib` 命令编译动态库。

```

cmd
Active code page: 437
Administrator@P-V-9 D:\repository\rtthread-apps 工作路径为rtthread-apps目录
> set RTT_ROOT=d:\repository\rt-thread
Administrator@P-V-9 D:\repository\rtthread-apps
> set BSP_ROOT=d:\repository\rt-thread\bsp\qemu-vexpress-a9
Administrator@P-V-9 D:\repository\rtthread-apps
> scons --lib=lib 生成动态库
scons: Reading SConscript files ...
scons: done reading SConscript files.
scons: Building targets ...
CC lib\lib.o
LINK lib\lib.so
arm-none-eabi-strip -R .hash lib\lib.so
arm-none-eabi-size lib\lib.so
   text    data     bss     dec     hex filename
   485      128        0     613     265 lib\lib.so
scons: done building targets.

Administrator@P-V-9 D:\repository\rtthread-apps
>

```

图 15: 编译动态模块

在 rtthread-apps/lib 目录下会生成动态库文件 lib.so。

请参考前面小节将动态库文件 lib.so 放到文件系统里，并将生成的映像文件 sd.bin 复制到 qemu-vexpress-a9 BSP 目录。

5.2 运行动态库

5.2.1. 添加示例代码

将以下示例代码添加到 qemu-vexpress-a9 BSP applications 目录下的 main.c 里。

```

#include <stdio.h>
#include <dlfcn.h>
#include <rtthread.h>

/* 动态库文件路径 */
#define APP_PATH    "/lib.so"

/* 函数指针类型 */
typedef int (*add_func_t)(int, int);
typedef void (*lib_func_t)(void);

int dlmodule_sample(void)
{
    void* handle;
    lib_func_t lib_function;

```



```

    add_func_t add_function;
    /* 以RTLD_LAZY模式打开动态库文件，并获取动态库操作句柄 */
    handle = dlopen(APP_PATH,RTLD_LAZY);

    if(!handle)
    {
        printf("dlopen %s failed!\n",APP_PATH);
        return -1;
    }

    /* 根据动态库操作句柄handle，返回动态库函数lib_func()对应的地址 */
    lib_function = (lib_func_t)dlsym(handle,"lib_func");
    if(!lib_function)
    {
        printf("dlsym %p failed!\n",handle);
        return -1;
    }
    /* 运行动态库函数 */
    lib_function();
    /* 根据动态库操作句柄handle，返回动态库函数add_func()对应的地址 */
    add_function = (add_func_t)dlsym(handle,"add_func");
    if(!add_function)
    {
        printf("dlsym %p failed!\n",handle);
        return -1;
    }
    /* 运行动态库函数计算 3+4 并打印结果 */
    printf("add_function result is:%d\n",add_function(3,4));
    /* 运行完毕根据操作句柄handle关闭动态库 */
    dlclose(handle);

    return 0;
}

MSH_CMD_EXPORT(dlmodule_sample, dlmodule sample);

int main(void)
{
    printf("hello rt-thread!\n");

    return 0;
}

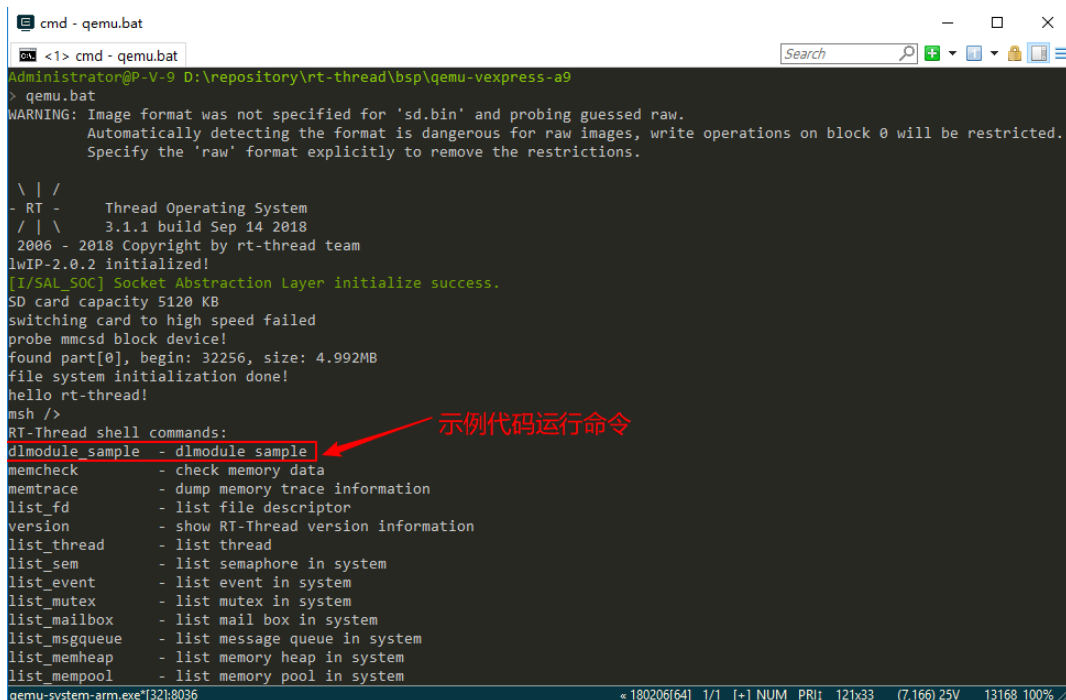
```

RT-Thread 动态模块组件也支持 POSIX 标准的 libdl API，此示例代码调用 libdl API 运行动态库。示例代码首先根据动态库的路径打开动态库文件 lib.so，然后获取动态库的 lib_func() 函数的地址并运行此函数。之后获取动态库的 add_func() 函数的地址，并传入

参数 3 和 4 运行函数计算结果。最后关闭动态库。

5.2.2. 运行动态库

在 Env 控制台切换到 qemu-vexpress-a9 BSP 根目录，输入 `scons` 命令重新编译工程。编译完成后输入 `qemu.bat` 命令运行工程。按 Tab 键可以看到新增的示例代码命令 `dlmodule_sample`。



```

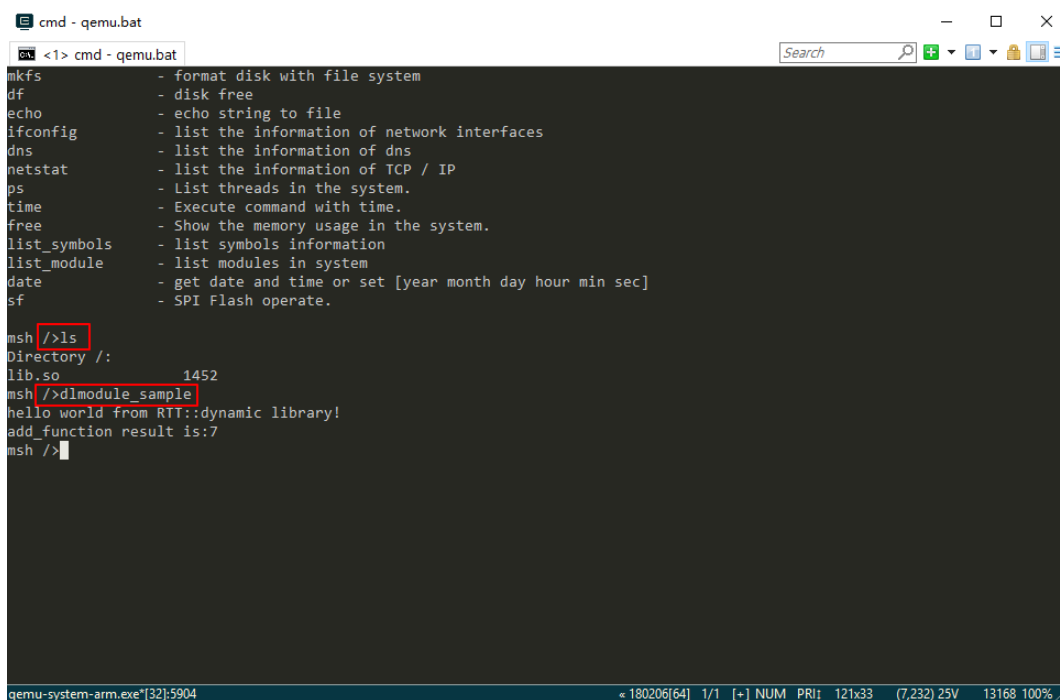
cmd - qemu.bat
Administrator@P-V-9 D:\repository\rt-thread\bsp\qemu-vexpress-a9
> qemu.bat
WARNING: Image format was not specified for 'sd.bin' and probing guessed raw.
Automatically detecting the format is dangerous for raw images, write operations on block 0 will be restricted.
Specify the 'raw' format explicitly to remove the restrictions.

\ | /
- 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_S0C] Socket Abstraction Layer initialize success.
SD card capacity 5120 KB
switching card to high speed failed
probe mmcblk0 block device!
found part[0], begin: 32256, size: 4.992MB
file system initialization done!
hello rt-thread!
msh />
RT-Thread shell commands:
dlmodule sample - dlmodule sample
memcheck       - check memory data
memtrace       - dump memory trace information
list_fd        - list file descriptor
version        - show RT-Thread version information
list_thread    - list thread
list_sem       - list semaphore in system
list_event     - list event in system
list_mutex     - list mutex in system
list_mailbox   - list mail box in system
list_msgqueue  - list message queue in system
list_memheap   - list memory heap in system
list_mempool   - list memory pool in system
qemu-system-arm.exe[32]:8036

```

图 16: 运行 `qemu` 工程

使用 `ls` 命令可以看到根目录下的动态库文件 `lib.so`，输入 `dlmodule_sample` 命令就可以运行动态库示例代码。



```
cmd - qemu.bat
<1> cmd - qemu.bat
mkfs      - format disk with file system
df         - disk free
echo       - echo string to file
ifconfig   - list the information of network interfaces
dns        - list the information of dns
netstat    - list the information of TCP / IP
ps         - List threads in the system.
time       - Execute command with time.
free       - Show the memory usage in the system.
list_symbols - list symbols information
list_module - list modules in system
date       - get date and time or set [year month day hour min sec]
sf         - SPI Flash operate.

msh />ls
Directory /:
lib.so      1452
msh />dlmodule_sample
hello world from RTT::dynamic library!
add_function result is:7
msh />
```

图 17: 运行动态库示例

- 第一行运行了 `lib_func()` 函数打印了字符串 “hello world from RTT::dynamic library!”
- 第二行运行了 `add_func()` 函数计算了 $3+4$ 并打印了相加结果 7。

6 参考

- [使用 QEMU 运行 RT-Thread](#)
- [使用 VS Code + QEMU 调试 RT-Thread](#)
- [编程指南 - 动态模块](#)，关于动态模块使用的更多详细介绍请参考编程指南动态模块章节。
- [Env 工具使用手册](#)

7 常见问题

- Env 工具的相关问题请参考 Env 工具使用手册的常用资料链接小节。
- 根据文档不能成功运行动态模块。

解决方法：请更新 RT-Thread 源代码到 3.1.0 及以上版本。