
Run **RT-THREAD** dynamic module using **QEMU**

RT-THREAD Documentation Center

Copyright ©2019 Shanghai Ruiside Electronic Technology Co., Ltd.



WWW.RT-THREAD.ORG

Friday 28th September, 2018

[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 Preparation.	1
3 Enable dynamic module components.	1
3.1 Configuration project.	1
3.2 Compile the project.	3
3.3 Running dynamic module commands	4
3.4 Generate dynamic module compilation dependency environment.	4
4 Run the dynamic module.	5
4.1 Run the simplest dynamic module.	5
4.1.1. Creating dynamic modules.	5
4.1.1.1. Get the example.	5
4.1.1.2. Setting environment variables.	5
4.1.1.3. Compiling dynamic modules.	6
4.1.2. Putting dynamic modules into the file system.	7
4.1.2.1. Create a new directory.	7
4.1.2.2. Modify the configuration file.	7
4.1.2.3. Generate image file.	7
4.1.3. Running dynamic modules.	8
4.2 Initialization and cleanup functions for dynamic modules.	9
4.2.1. Sample code	10

- 4.2.2. Operation results 11
- 5 Run the dynamic library. 12
 - 5.1 Creating a dynamic library 12
 - 5.1.1. Get the example 12
 - 5.1.2. Compile dynamic library. 12
 - 5.2 Running the dynamic library 13
 - 5.2.1. Add sample code. 13
 - 5.2.2. Run the dynamic library. 15
- 6 References. 16
- 7 Frequently asked questions. 16

!!! abstract "Abstract" This application note describes running RT-Thread dynamic modules on Windows platform using QEMU.

1 Purpose and structure of this paper

1.1 Purpose and Background of this Paper

RT-Thread dynamic module component `dlmodule` provides a mechanism for dynamically loading program modules. The `dlmodule` component is more of an ELF format loader, which loads the code segment and data segment of a separately compiled elf file into memory, parses the symbols in it, and binds them to the API address exported by the kernel. The dynamic module elf file is mainly placed on the file system under RT-Thread.

RT-Thread's dynamic module components currently support two formats:

- `.mo` is an executable dynamic module with the suffix `.mo` when compiled. It can be loaded, and the system will automatically create a main thread to execute the `main` function in this dynamic module; at the same time, this `main (int argc, char** argv)` functions can also accept arguments on the command line.
- `.so` is a dynamic library with the suffix `.so` when compiled. It can be loaded and reside in memory. Provides a set of functions to be used by other programs (code in the kernel or dynamic modules).

This article mainly explains how to use QEMU to run RT-Thread dynamic modules on the Windows platform.

1.2 Structure of this paper

This article first explains how to enable RT-Thread dynamic module components, and then explains how to run dynamic modules and dynamic libraries based on QEMU.

2. Preparation

- [Download RT-Thread](#) Source code, it is recommended to download version 3.1.0 or above.
- [Download RT-Thread Env](#) Tools, it is recommended to download version 1.0.0 or above.
- [Download the rtthread-apps source code.](#)

3. Enable dynamic module components

3.1 Configuration Project

In the Env console, switch to the `qemu-vexpress-a9` BSP root directory and enter the `menuconfig` command to open the configuration menu.

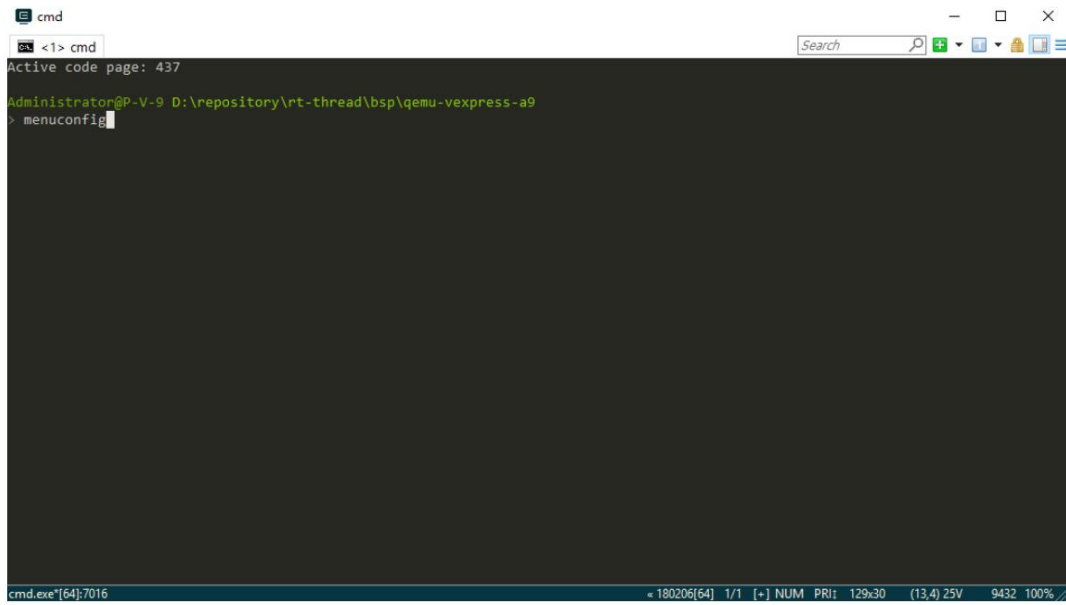


Figure 1: menuconfig Open the configuration menu

Enter the "RT-Thread Components → POSIX layer and C standard library" menu and click on the picture below

Turn on the configuration options for libc and dynamic modules as shown by the arrows.

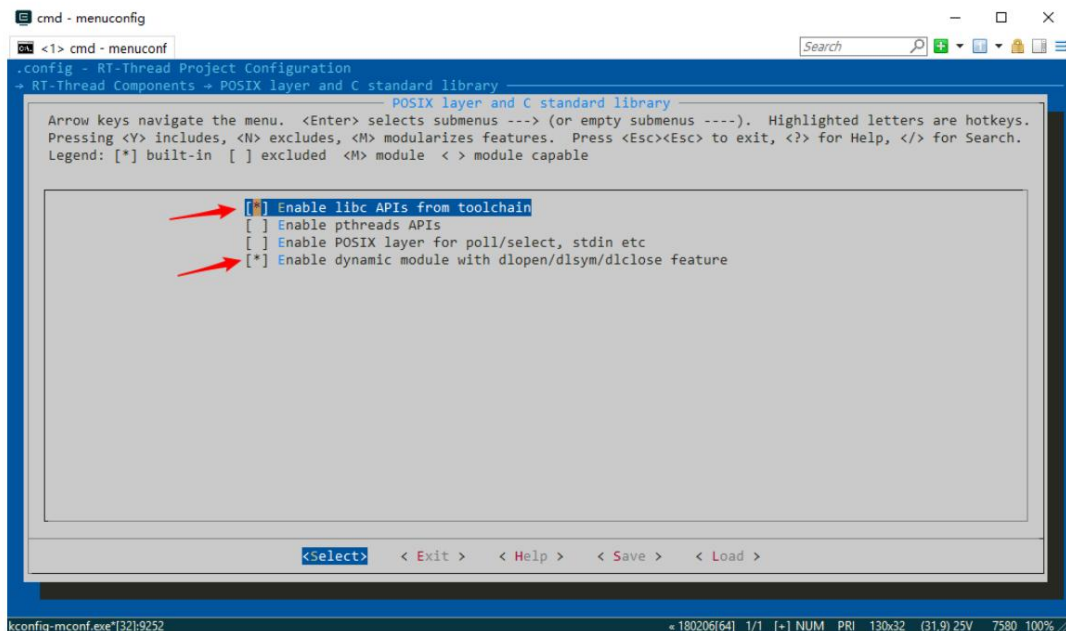


figure 2: Open dynamic module

Go to the "RT-Thread Components → Device virtual file system" menu to open the file system configuration option. Exit menuconfig and save the configuration.

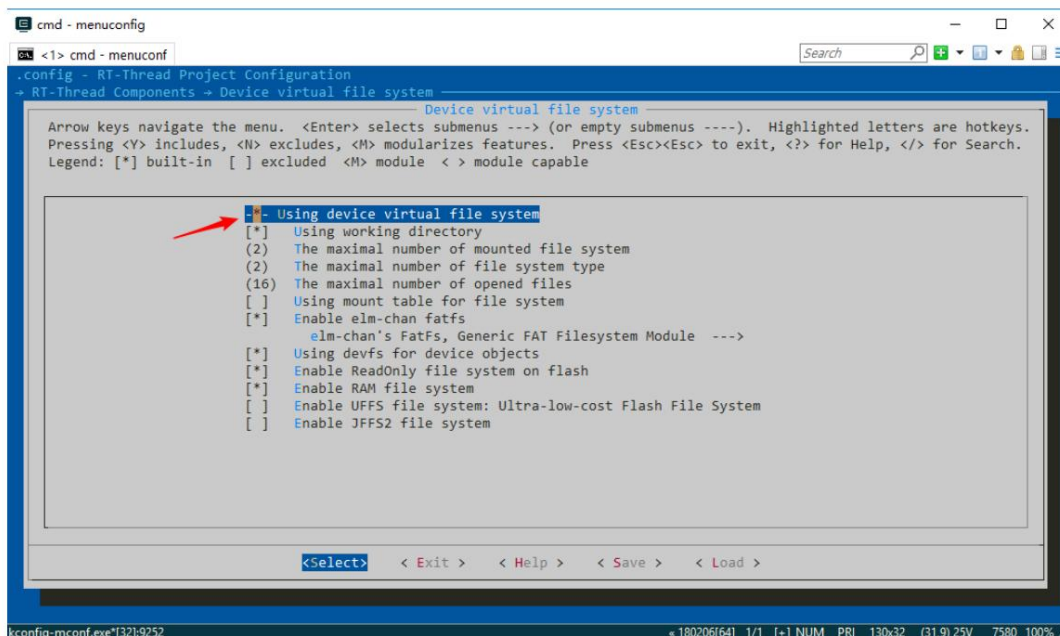


image 3: Open file system

3.2 Compile Project

Use the `scons` command to compile the project.

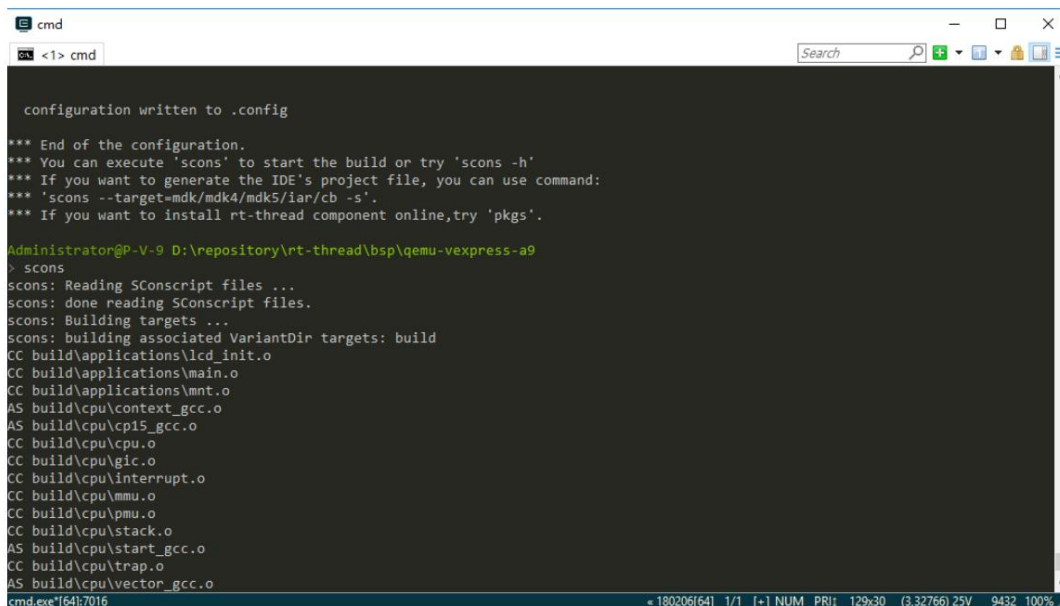


Figure 4: Compile project

3.3 Run dynamic module commands

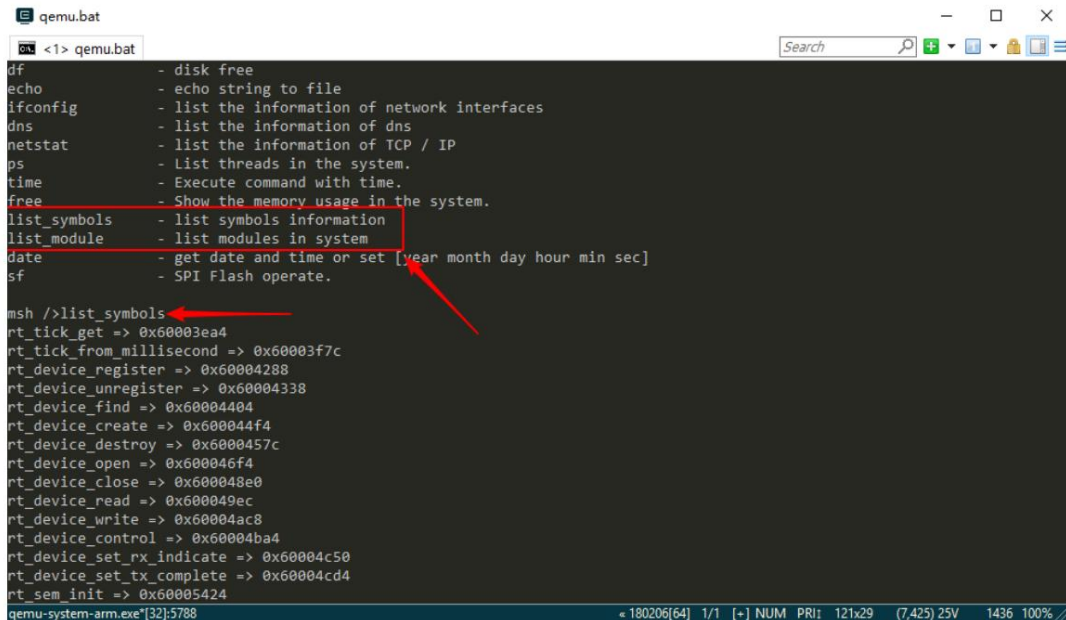
After the compilation is complete, use the `qemu.bat` command to run the project. Press the Tab key to view all commands and you can see the two commands

`list_module` and `list_symbols` of the dynamic module, indicating that the dynamic module component has been configured successfully.

- The `list_module` command can be used to view the currently running dynamic module. • The

`list_symbols` command can be used to view the functions that can be used by the dynamic module and their corresponding memory addresses.

When building a module, the symbols in it will be parsed and bound to the corresponding function addresses.



```

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
  
```

Figure 5: Run dynamic module command

3.4 Generate dynamic module compilation dependency environment

Close the running program and use the `scons -target=ua -s` command in the Env console to generate and compile dynamic modules.

Kernel header file search paths and global macro definitions that need to be included.

```

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
  
```

Figure 6: Generate dynamic module compilation dependency environment

4Running dynamic modules

4.1 Running the simplest dynamic module

4.1.1. Creating dynamic modules

4.1.1.1. Get the example Download the RT-Thread dynamic module tool library rtthread-apps. The tools directory of rtthread-apps contains the Python and SConscript scripts needed to compile dynamic modules. main.c in the hello directory is a simple example of using a dynamic module. The source code is shown below.

```

#include <stdio.h>

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

    printf("Hello, world\n");

    return 0;

}
  
```

This code implements the simplest main function and prints the string "Hello world".

4.1.1.2. Set environment variables Switch to the rtthread-apps root directory in the Env console (the full path of the directory does not contain spaces and Chinese characters), and then set the environment variables using the following two commands.

- set `RTT_ROOT=d:\repository\rt-thread`, set `RTT_ROOT` to the RT-Thread source code root

Table of contents.

- set `BSP_ROOT=d:\repository\rt-thread\bsp\qemu-vexpress-a9`, set `BSP_ROOT`

The root directory of the qemu-vexpress-a9 BSP.

```

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
>
  
```

Figure 7: Setting Environment Variables

4.1.1.3. Compile dynamic modules Use `scons --app=hello` command to compile dynamic modules.

```

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
>
  
```

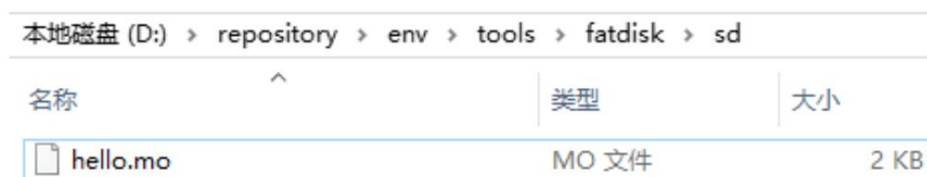
Figure 8: Compiling dynamic modules

The dynamic module file `hello.mo` will be generated in the `rtthread-apps/hello` directory.

4.1.2. Putting dynamic modules into the file system

The compiled dynamic module `hello.mo` needs to be placed in the file system. The `qemu-vexpress-a9` BSP will use a virtual sd card device `sd.bin`, and we need to put the dynamic module in this virtual sd card. For physical devices, you can directly add the dynamic module to the storage device managed by the file system. Here you need to use a small tool `fatdisk` in the `Env` tool, which is located in the `tools` directory of `Env`, and a `fatdisk` usage manual is also provided. Here `fatdisk` is used to convert a local directory on the PC into an `sd.bin` image file, which exists as a fat file system.

4.1.2.1. Create a new directory Create a new directory `sd` under the `fatdisk` directory and copy the dynamic module `hello.mo` file just compiled to the `sd` directory.



名称	类型	大小
hello.mo	MO 文件	2 KB

Figure 9: Increase *fatdisk* Configuration Files

4.1.2.2. Modify the configuration file Modify the configuration file `fatdisk.xml` in the `fatdisk` directory according to the following configuration.

- The image file space size `disk_size` is configured to 5120Kbytes (the size can be configured as needed).
- The sector size `sector_size` of the image file needs to be configured to 512 KBytes.
- To convert the directory name `root_dir`, configure it to `sd`, indicating the `sd` directory under the current directory.
- Specify the generated image file name output to `sd.bin`.
- `Strip` needs to be configured to 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. Generate image file Switch to the `fatdisk` root directory in the `Env` console and run the `fatdisk` command to convert the specified directory into a flash image file according to the configuration in the configuration file `fatdisk.xml`.

```

cmd
Active code page: 437
Administrator@P-V-9 D:\repository\env\tools\fatdisk 工作路径为fatdisk
> fatdisk
fatdisk version 1.0
disk info: size 5 MB, sector size 512 bytes
sd\hello.mo => /hello.mo
save to sd.bin...done!

Administrator@P-V-9 D:\repository\env\tools\fatdisk
>

```

Figure 10: run fatdisk Order

If the operation is successful, a sd.bin file with a size of 5MB will be generated in the fatdisk directory.

本地磁盘 (D:) > repository > env > tools > fatdisk			
名称	类型	大小	
sd	文件夹		
fatdisk.exe	应用程序	145 KB	
fatdisk.xml	XML 文档	1 KB	
fatdisk使用说明.docx	Microsoft Word ...	33 KB	
sd.bin	BIN 文件	5,120 KB	

Figure 11: generate sd.bin document

The generated image file sd.bin needs to be copied to the qemu-vexpress-a9 BSP directory.

4.1.3. Running dynamic modules

In the Env console, switch to the qemu-vexpress-a9 BSP root directory and enter the qemu.bat command to run the project.

```

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 18 2018
2006 - 2018 Copyright by rt-thread team
SD card capacity 5120 KB
switching card to high speed failed
probe mmcsd 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

```

Figure 12: run *qemu* project

- After the system is running, you will see the message "file system initialization done!" indicating successful file system initialization.
- Use the `ls` command to see the dynamic module file `hello.mo` in the root directory .
- Enter the `hello` command to run the dynamic module `hello.mo`. You can see the string printed by the dynamic module main function.

"Hello, world"

The main principle of running dynamic modules using dynamic module components is shown in the following figure:

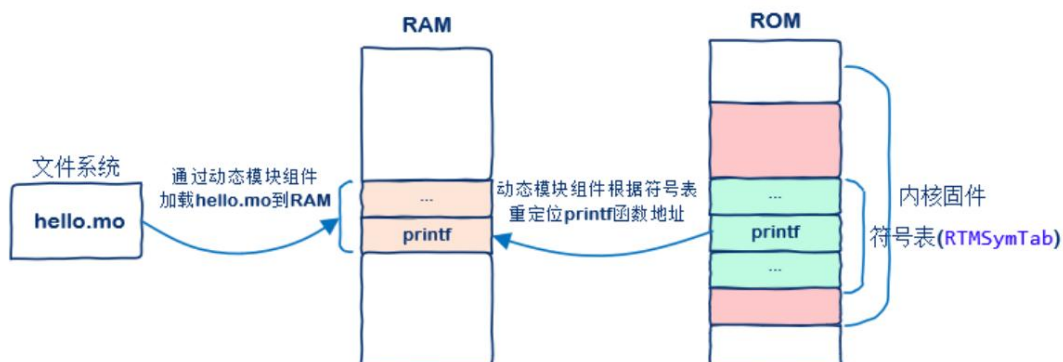


Figure 13: How dynamic modules work

4.2 Dynamic module initialization and cleanup functions

The dynamic module component provides two extended functions for users to use, namely `module_init()` and `module_cleanup()`.

- The `module_init()` function will be executed before the dynamic module runs. Users can do some initialization work as needed.
- do.

- The `module_cleanup()` function will be called back once in the idle thread after the dynamic module is finished running , and execute the user-defined

Cleaning work of the equipment.

The RT-Thread system will automatically create a thread to execute the main function in the dynamic module. At the same time, this `main(int argc, char* argv[])` function can also accept parameters on the command line. The default priority of this thread is equal to the priority of the idle thread, and the thread stack defaults to 2048 bytes. Users can modify the priority and stack of this thread in the `module_init()` function .

4.2.1. Example code

Based on the previous simple dynamic module example code `main.c`, add `module_init()` and `module_cleanup()` functions. The sample code for usage is shown below.

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

/* Initialization function of dynamic module*/
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);
}

/* Dynamic module cleanup
function*/ 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");

    /* Print command line
    arguments */ for(i = 0;i < argc;i +
    +) {
        printf("argv[%d]:%s\n",i,argv[i]);
    }

    return 0;
```

```
}

```

The sample code mainly implements the following functions:

- In the initialization function of the dynamic module, you can set the priority and stack of this thread.
- The cleanup function simply prints information.
- The main function parses the command line parameters and prints them out.

Please refer to the previous section to put the dynamic module file generated by this sample code into the file system and copy the generated image file

Copy the sd.bin file to the qemu-vexpress-a9 BSP directory.

4.2.2. Operation results

In the Env console, switch to the qemu-vexpress-a9 BSP root directory and enter the qemu.bat command to run the project.

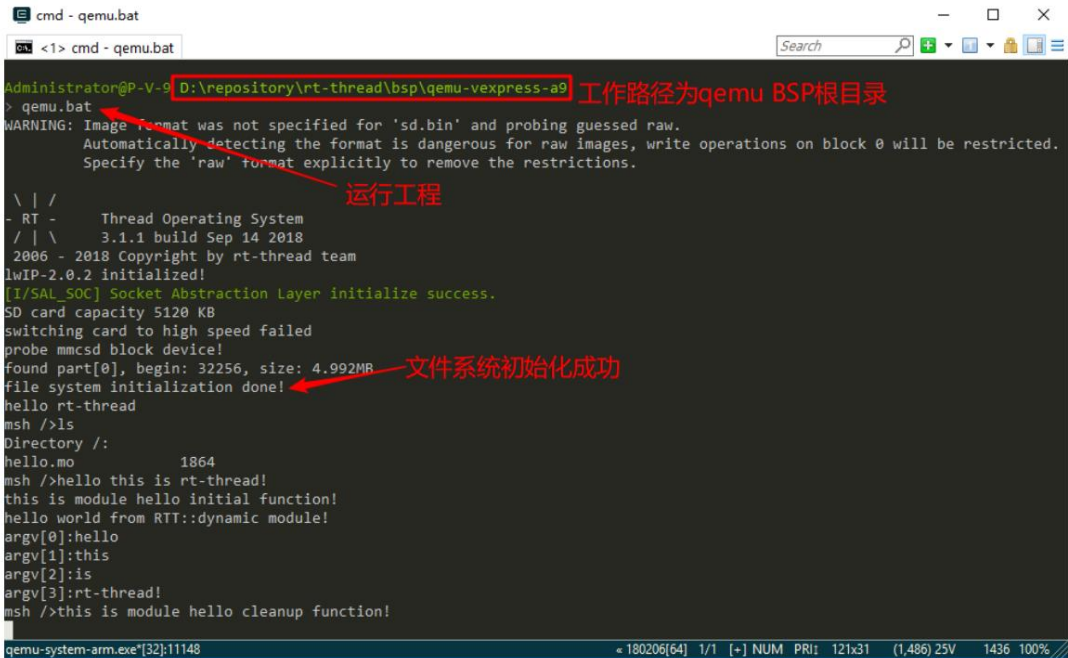


Figure 14: run *qemu* project

- After the system is running, you will see the file system initialization success message "file system initialization done!".
- Use the `ls` command to see the dynamic module file `hello.mo` in the root directory .
- Enter the `hello this is rt-thread!` command to run the dynamic module `hello.mo`. The string after `hello` is parameter.
- When executing the dynamic module initialization function `module_init` , the string "this is module hello initial function!".
- When the main function of the dynamic module is executed, the string "hello world from RTT::dynamic module!" will be printed.
- The command line parameters are also printed out one by one.
- After the dynamic module is finished running, the cleanup function `module_cleanup` is executed, printing the string "this is module hello cleanup function!".

5Running dynamic library

5.1 Creating a dynamic library

5.1.1. Get the example

Download the RT-Thread dynamic module tool library `rtthread-apps`. There is a simple
The source code of `lib.c` of a single dynamic library example is shown below. It implements 2 simple functions for use.

```
#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. Compile dynamic library

Before compiling the dynamic library, you need to set the environment variables. Then use the `scons --lib=lib` command to compile the dynamic library.

```

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
>

```

Figure 15: Compiling dynamic modules

The dynamic library file lib.so will be generated in the rtthread-apps/lib directory.

Please refer to the previous section to put the dynamic library file lib.so into the file system, and copy the generated image file sd.bin

Go to the qemu-vexpress-a9 BSP directory.

5.2 Run dynamic library

5.2.1. Add sample code

Add the following sample code to main.c in the qemu-vexpress-a9 BSP applications directory.

```

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

/* Dynamic library file path
*/ #define APP_PATH      "/lib.so"

/* Function pointer type*/
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; /* Open
the dynamic library file in RTLD_LAZY mode and obtain the dynamic library operation
handle*/ handle = dlopen(APP_PATH,RTLD_LAZY);

if(!handle) {

    printf("dlopen %s failed\n",APP_PATH); return -1;

}

/* According to the dynamic library operation handle handle, return the address corresponding to
the dynamic library function lib_func()*/ lib_function =

(lib_func_t)dlsym(handle,"lib_func"); if(!lib_function) {
    printf("dlsym %p failed\n",handle); return -1;

} /* Run dynamic library
function*/ lib_function(); /
* According to the dynamic library operation handle handle, return the address corresponding to the
dynamic library function add_func()*/ add_function =

(add_func_t)dlsym(handle,"add_func"); if(!add_function) {
    printf("dlsym %p failed\n",handle); return -1;

} /* Run the dynamic library function to calculate 3+4 and print the result*/
printf("add_function result is:%d\n",add_function(3,4)); /* After the operation is
completed, close the dynamic library according to the operation handle*/
dlclose(handle);

return 0;
}

MSH_CMD_EXPORT(dlmodule_sample, dlmodule sample);

int main(void) {

    printf("hello rt-thread\n");

    return 0;
}

```

RT-Thread dynamic module components also support the POSIX standard libdl API. This sample code calls the libdl API to run the dynamic library. The sample code first opens the dynamic library file lib.so according to the path of the dynamic library, then obtains the address of the dynamic library's lib_func() function and runs this function. Then it obtains the address of the dynamic library's add_func() function and passes it in.

Parameters 3 and 4 run the function to calculate the result. Finally, close the dynamic library.

5.2.2. Run dynamic library

In the Env console, switch to the qemu-vexpress-a9 BSP root directory and enter the `scons` command to recompile the project.

After the compilation is complete, enter the `qemu.bat` command to run the project. Press the Tab key to see the newly added sample code commands.

`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_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 />
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

```

Figure 16: run `qemu` project

Use the `ls` command to see the dynamic library file `lib.so` in the root directory and enter the `dlmodule_sample` command.

to run the dynamic library sample code.

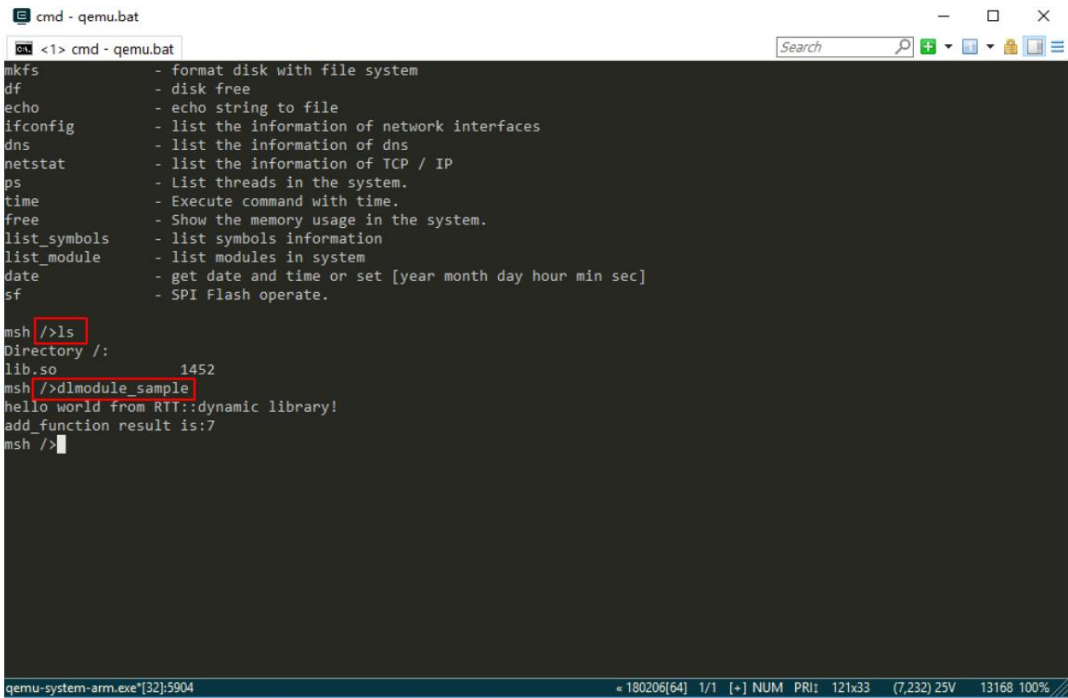


Figure 17: Run the dynamic library example

- The first line runs the lib_func() function and prints the string "hello world from RTT::dynamic library!" • The second line runs the add_func() function which calculates 3+4 and prints the result 7.

6 References

- [Use QEMU to run RT-Thread](#)
- [Use VS Code + QEMU to debug RT-Thread](#)
- [Programming Guide - Dynamic modules](#). For more detailed information about the use of dynamic modules, please refer to the Dynamic Modules chapter of the Programming Guide.
- [Env tool user manual](#)

7 Frequently Asked Questions

- For questions related to the Env tool, please refer to the Common Resource Links section of the Env Tool User Manual.
- Dynamic modules cannot be run successfully according to the documentation.

Solution: Please update the RT-Thread source code to version 3.1.0 or above.