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

**RT-THREAD** Document Center




**WWW.RT-THREAD.ORG**

**Friday 28th September, 2018**

Table of contents

This application note describes how to run RT-Thread dynamic modules on Windows using QEMU.

# 1 Purpose and structure of this paper

## 1.1 Purpose and Background of this Paper

The RT-Thread dynamic module component, dlmodule, provides a mechanism for dynamically loading program modules. The dlmodule component is essentially an ELF format loader. It loads the code and data segments of a separately compiled ELF file into memory, resolves the symbols within them, and binds them to API addresses exported by the kernel. Dynamic module ELF files are primarily stored on the RT-Thread file system.

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 parameters on the command line.
- .so is a dynamic library with the suffix .so when compiled . It can be loaded and reside in the memory and 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.

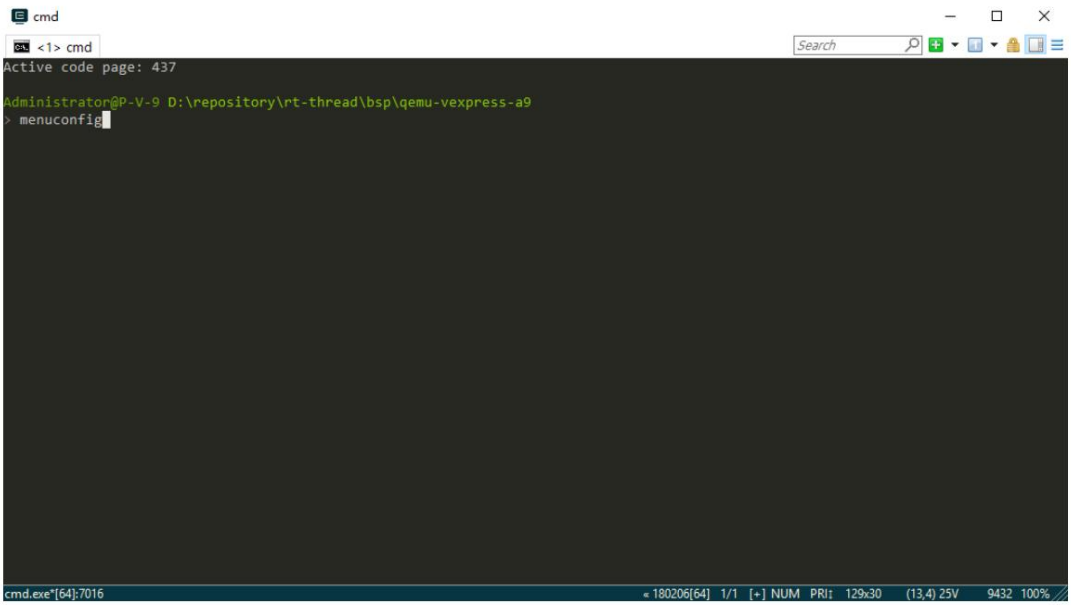


Figure **1:** *menuconfig* Open the configuration menu

Enter the "RT-Thread Components ÿ 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.

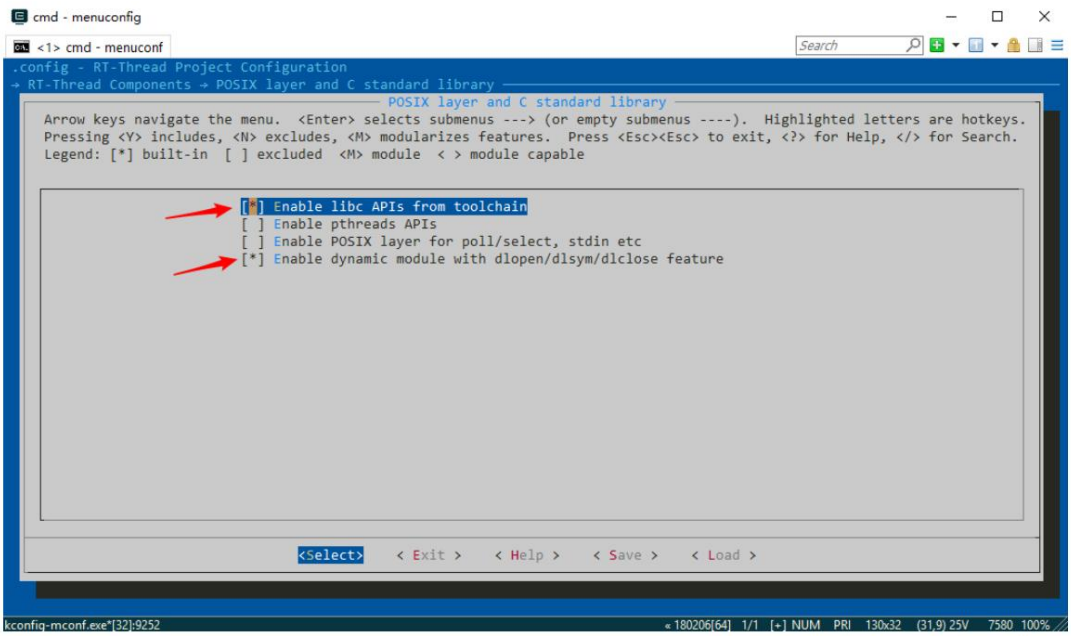


Figure **2:** Open dynamic module

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

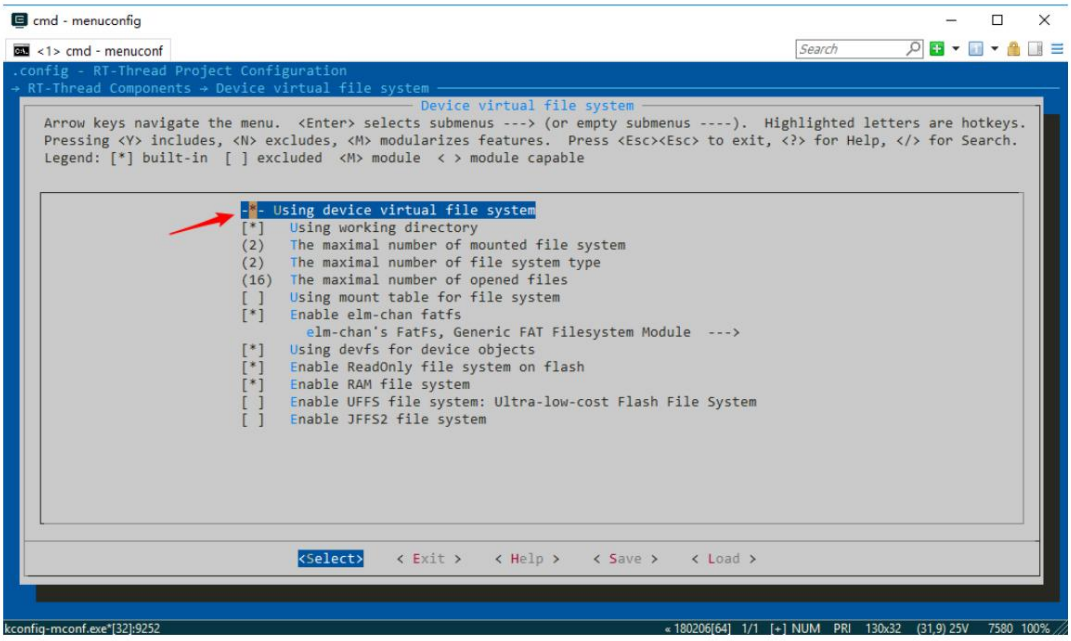


Figure **3:** Open file system

## 3.2 Compile Project

Use the scons command to compile the project.

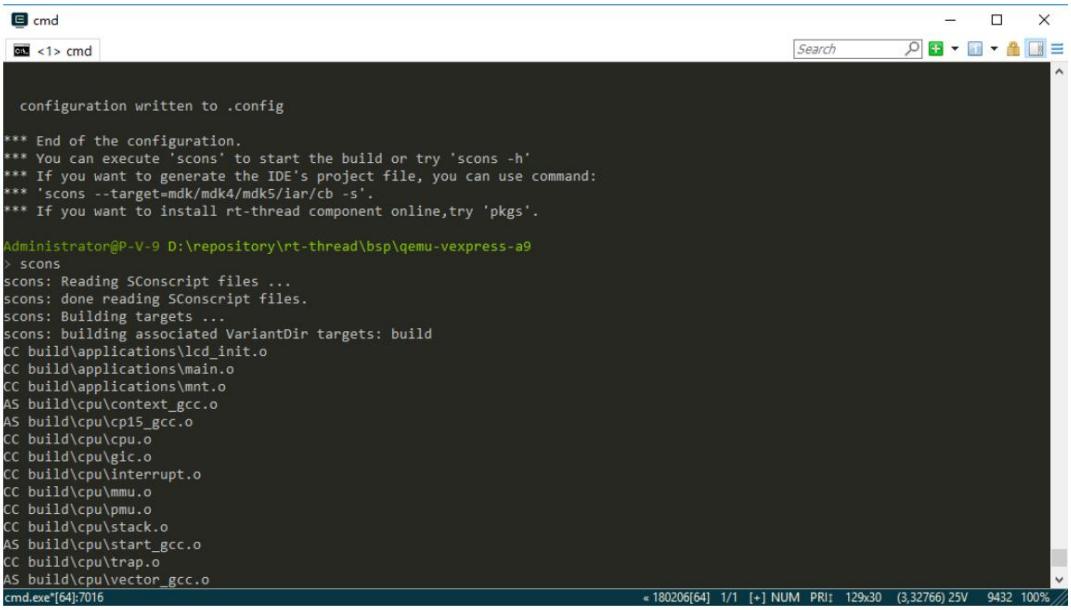


Figure **4:** Compile project

### 3.3 Run dynamic module command

After the compilation is complete, run the project using the qemu.bat command. Press the Tab key to view all commands. You will see the two commands list_module and list_symbols for the dynamic module, indicating that the dynamic module component has been successfully configured.

- 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.

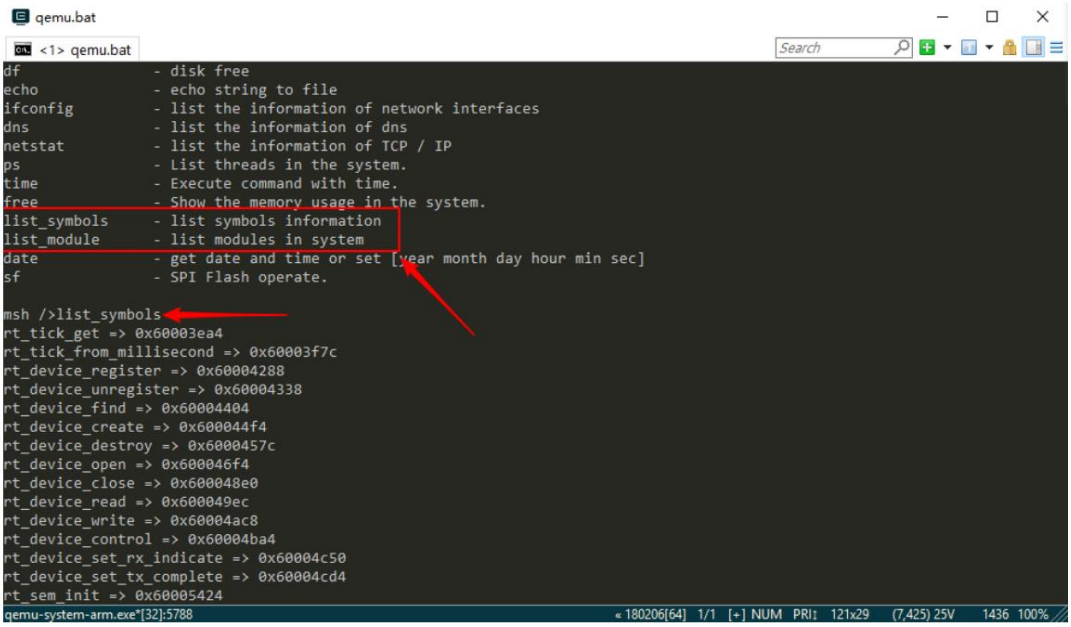


Figure **5:** Run dynamic module command

### 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.

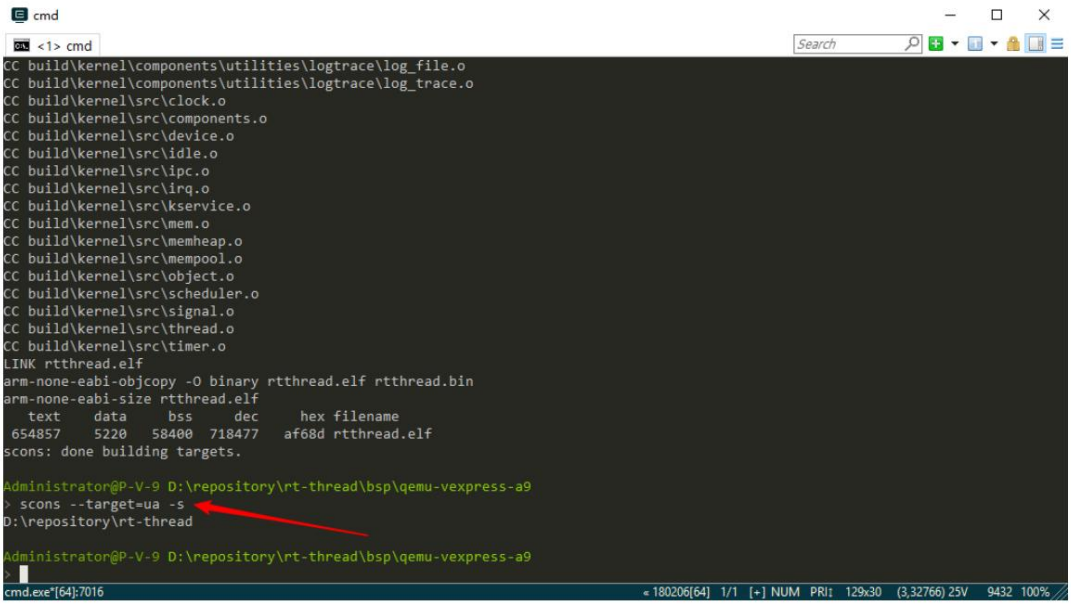


Figure 6: Generate dynamic module compilation dependency environment

# 4Running dynamic modules

## 4.1 Running the simplest dynamic module

### 4.1.1. Creating dynamic modules

**4.1.1.1.** Obtaining Examples 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. The main.c file in the hello directory is a simple example of using a dynamic module. The source code is shown below.

```c
#include <stdio.h>

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

        printf("Hello, world\n");

        return 0;
}
```

This code implements a simple main function that 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

  This is the root directory of the qemu-vexpress-a9 BSP.

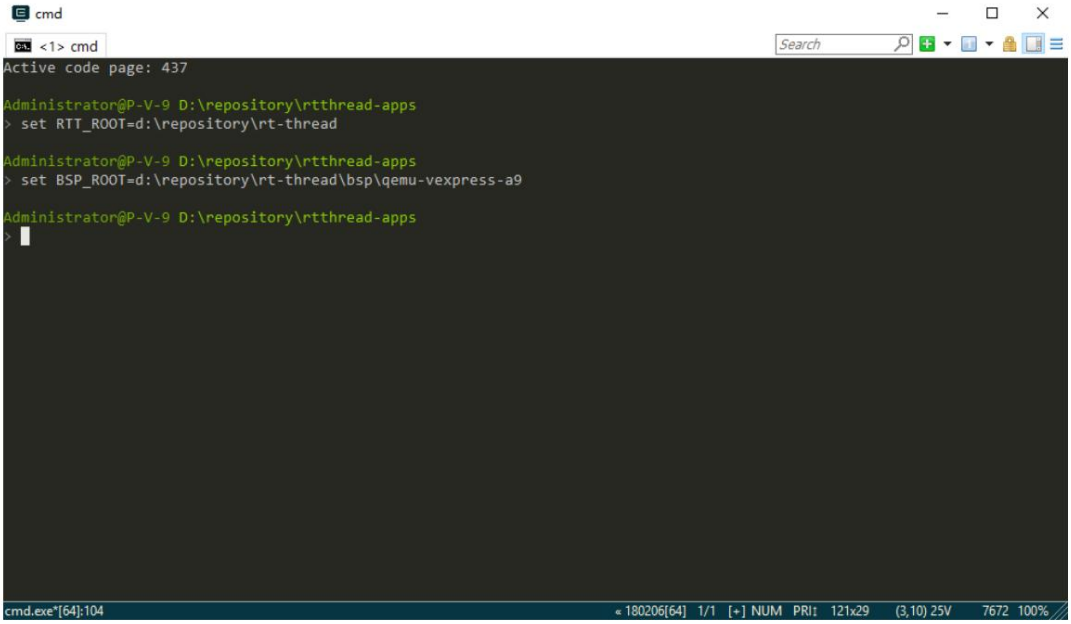


Figure 7: Setting environment variables

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

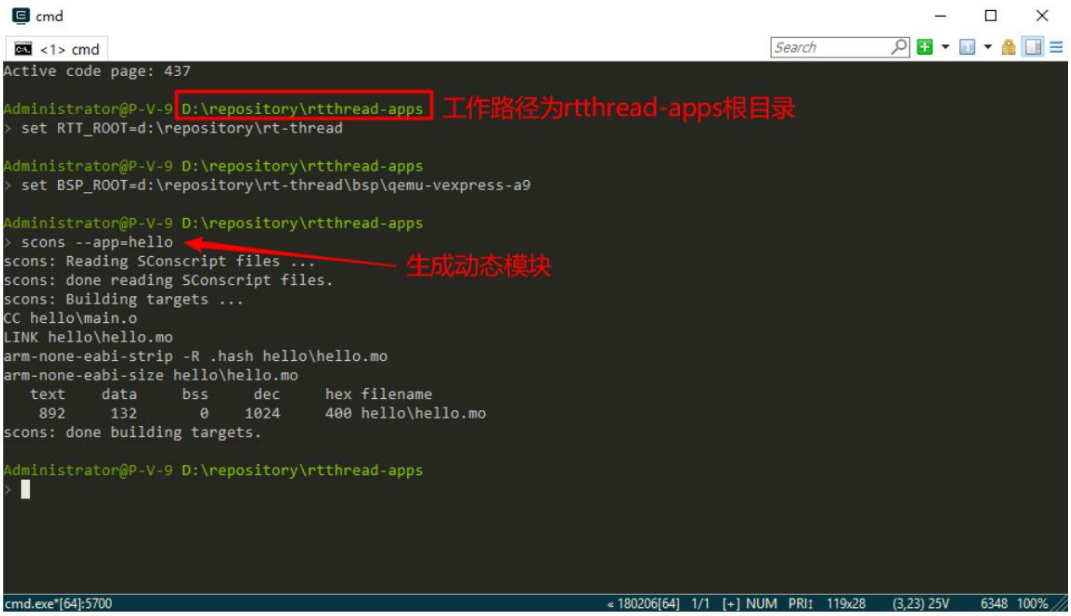


Figure 8: Compiling dynamic modules

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. It is located in the tools directory of Env, and a fatdisk usage manual is also provided. Fatdisk is used here to convert a local directory on the PC into an sd.bin image file. This image file 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.

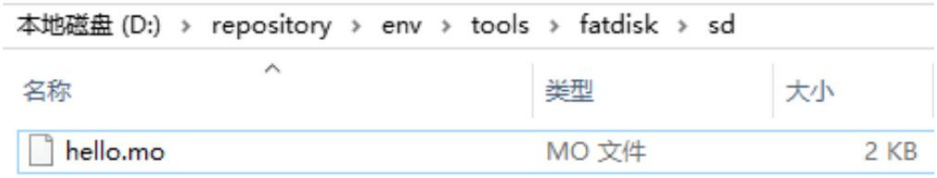


Figure **9:** Increase *fatdisk* Configuration File

**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 size (disk_size) is configured to 5120 KB (the size can be adjusted as needed). • The image file sector size (sector_size) needs to be configured to 512 KB. • To convert the directory name, configure root_dir to sd, indicating the sd directory under the current directory. • Specify the generated image file name (output) as sd.bin. • The strip flag needs to be set 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.

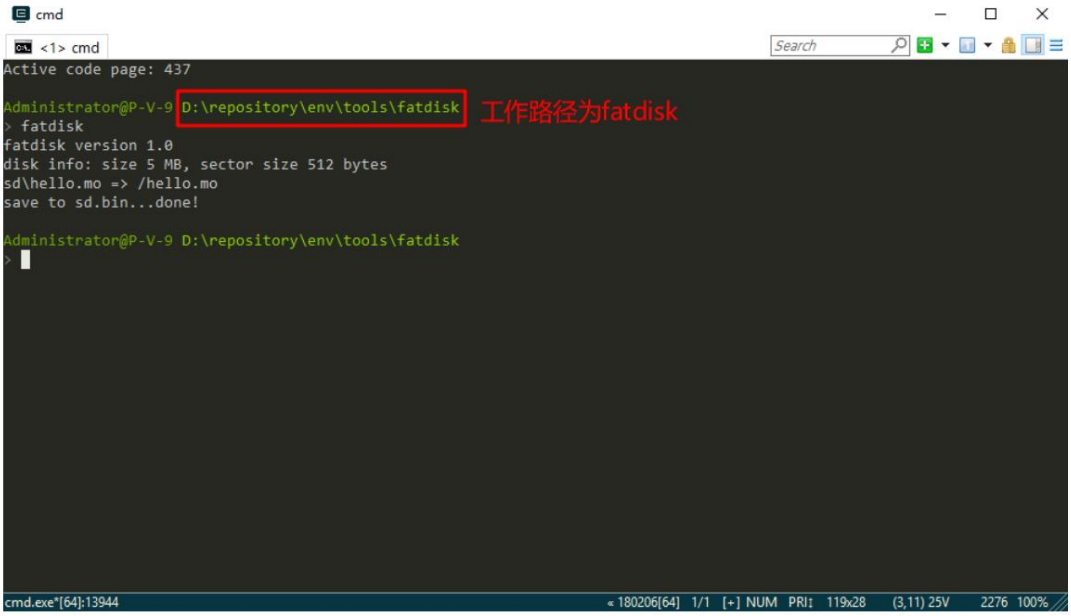


Figure **10:** run *fatdisk* Order

If the command is run successfully, a 5MB sd.bin file will be generated in the fatdisk directory.


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.

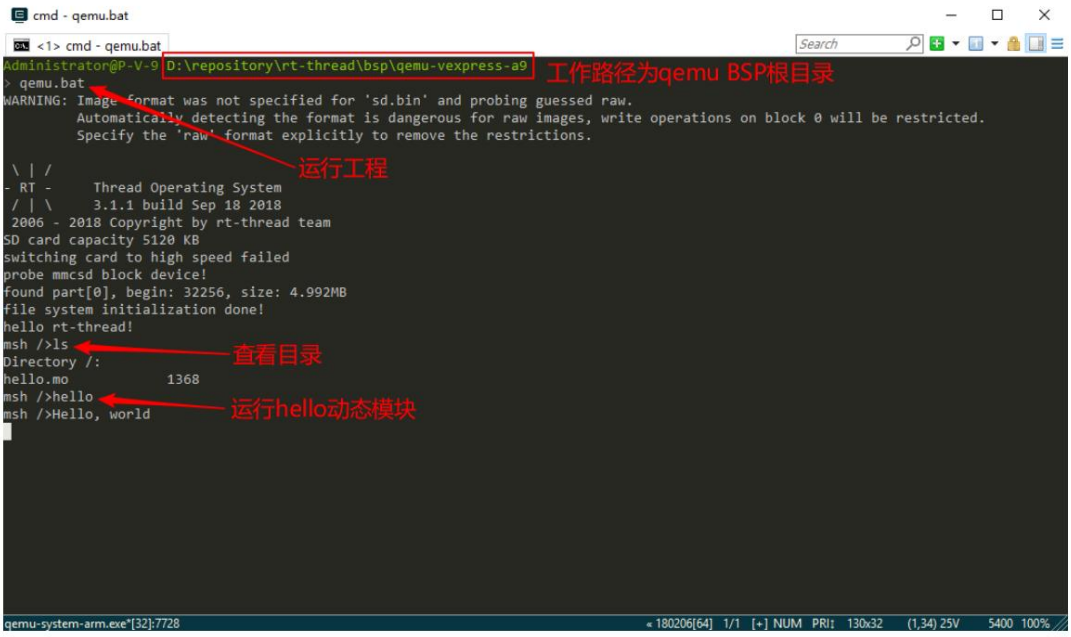


Figure **12:** run *qemu* project

- 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 command to run the dynamic module hello.mo. You can see the string printed by the main function of the dynamic module "Hello, world"

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

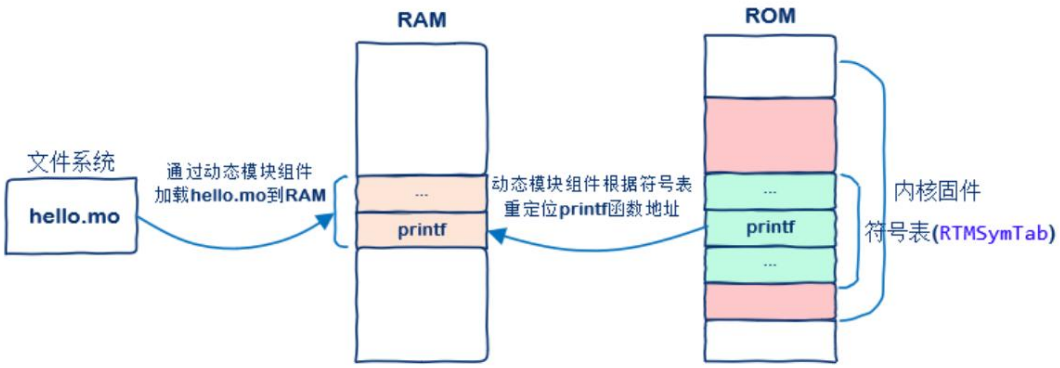


Figure **13:** How dynamic modules work

## 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, and 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 automatically creates a thread to execute the main function in the dynamic module. This main( int argc, **char*** argv[] ) function also accepts command-line arguments. The default priority of this thread is the same as the idle thread priority, and the thread stack defaults to 2048 bytes. Users can modify this thread priority and stack 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:

• The priority and stack of the thread can be set in the initialization function of the dynamic module. • The cleanup function simply prints information. • The main function parses the command line arguments 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 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.

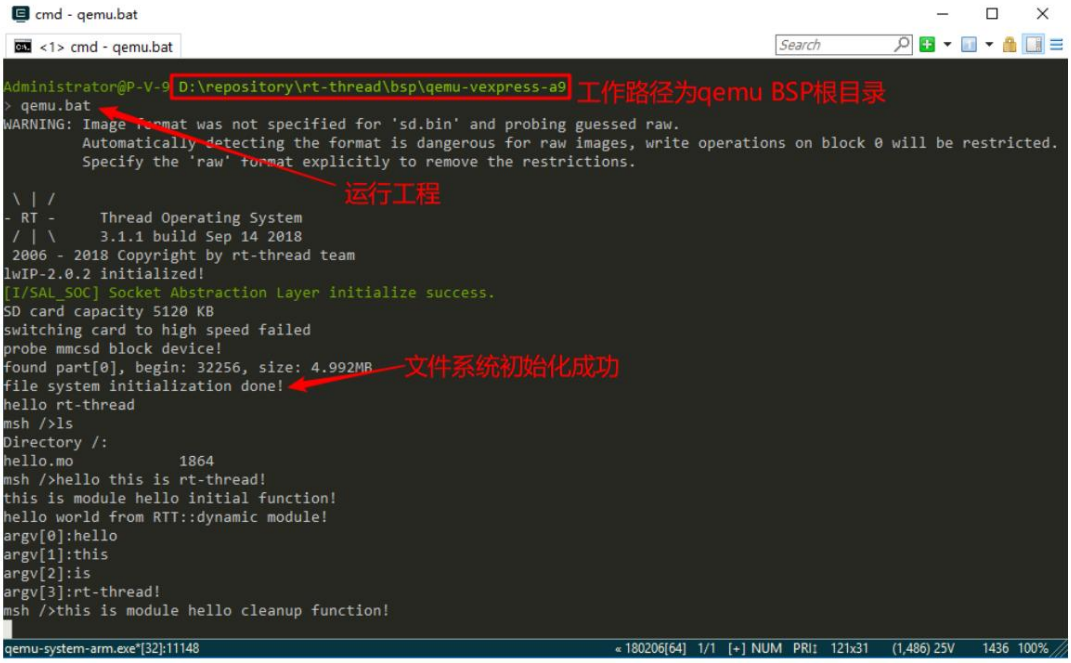


Figure **14:** run *qemu* project

• After the system is running, you will see the message "file system initialization done!" that the file system is initialized successfully. • Use the ls command to see the dynamic module file hello.mo in the root directory . • Enter the command hello this is rt-thread! to run the dynamic module hello.mo. The string after hello is parameter.

• When the dynamic module initialization function module_init is executed, 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 and the string "this is module hello cleanup function!".

# 5Run 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 two 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. Compiling dynamic libraries

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

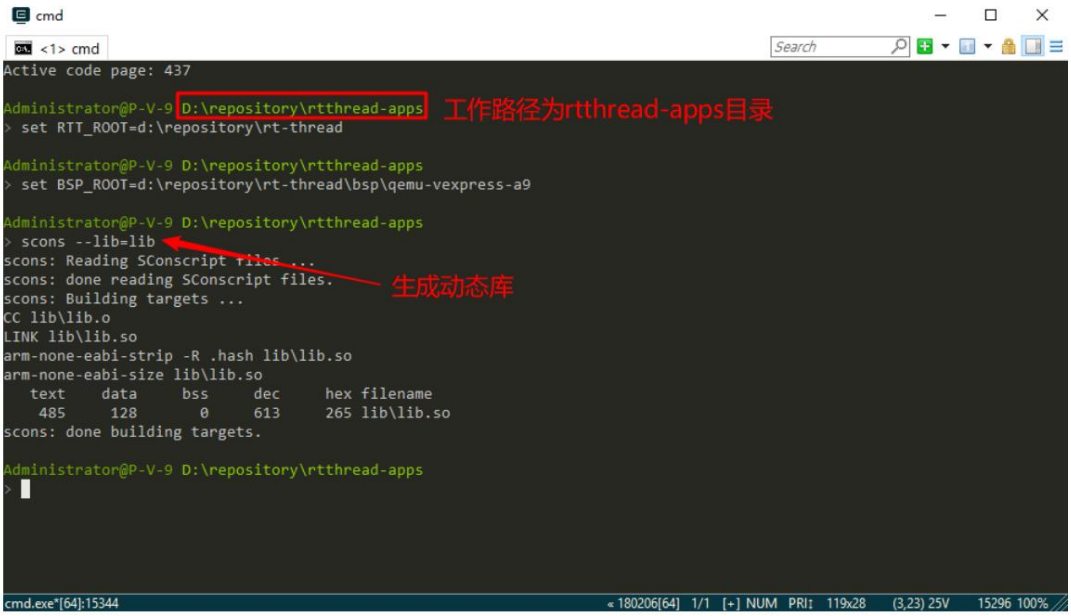


Figure **15:** Compiling dynamic modules

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 Running Dynamic Library

### 5.2.1. Add sample code

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

```
#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 to

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

#### 5.2.2. Running dynamic libraries

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

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.

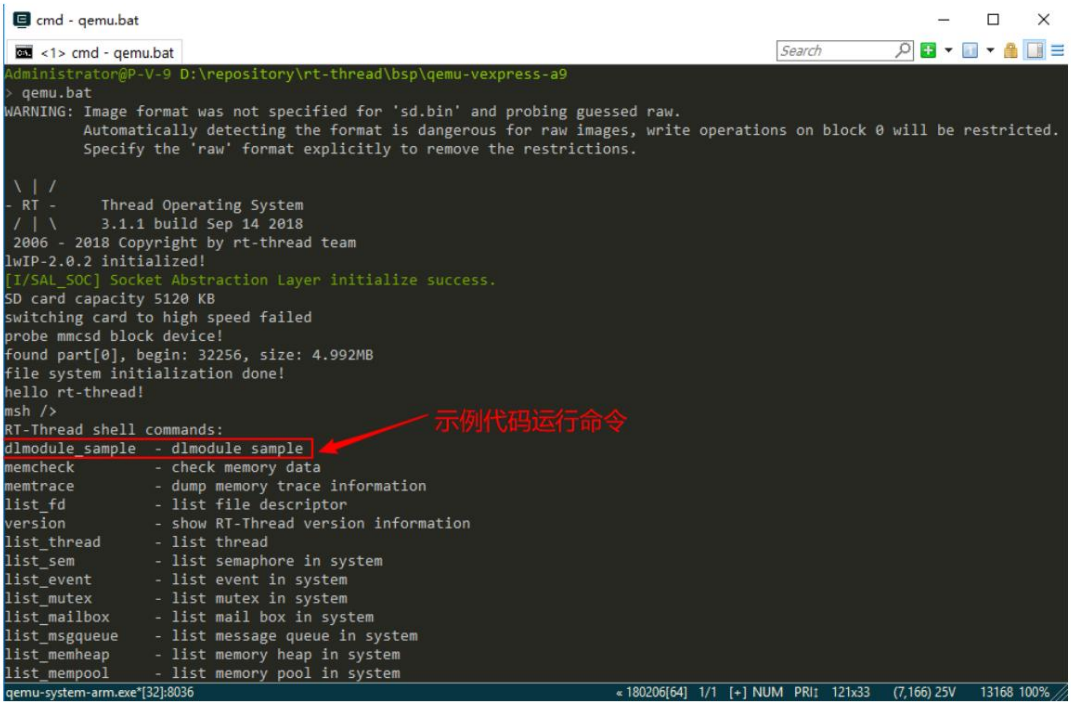


Figure **16:** run *qemu* project

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.

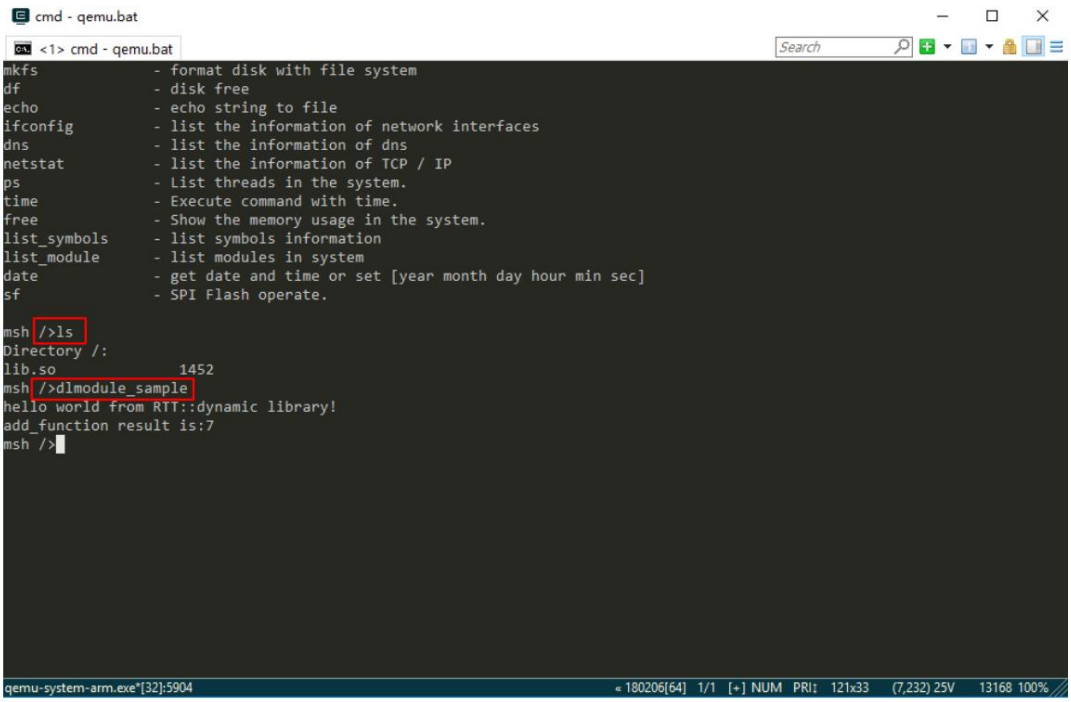


Figure **17:** Running the dynamic library example

• 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.

## 6References

• Using 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.

Festival.

• 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 successfully run according to the documentation.

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