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

RT-THREAD Documentation Center

WWW.RT-THREAD.ORG

Friday 28th September, 2018

Table of contents

!!! 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.

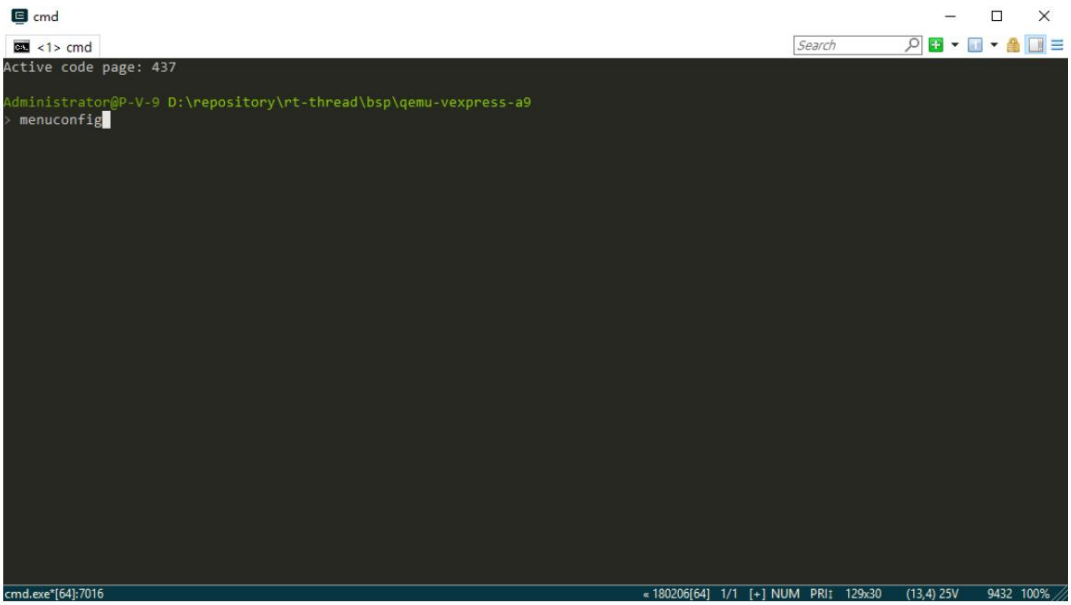

Figure **1:** *menuconfig* Open the configuration menu

Enter the "RT-Thread Components 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.

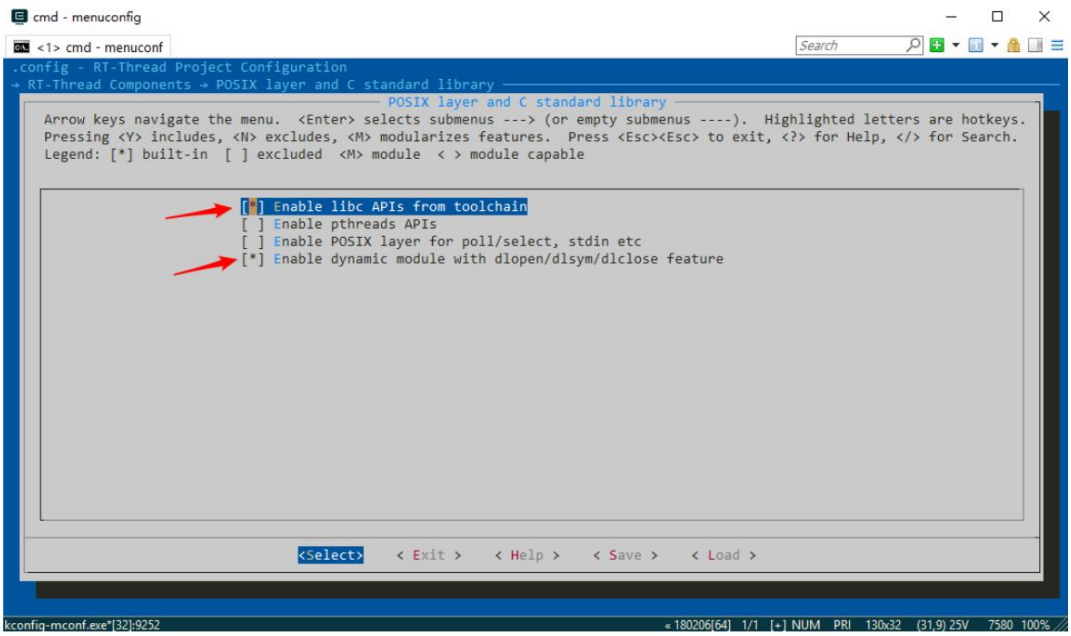

figure **2:** Open dynamic module

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.

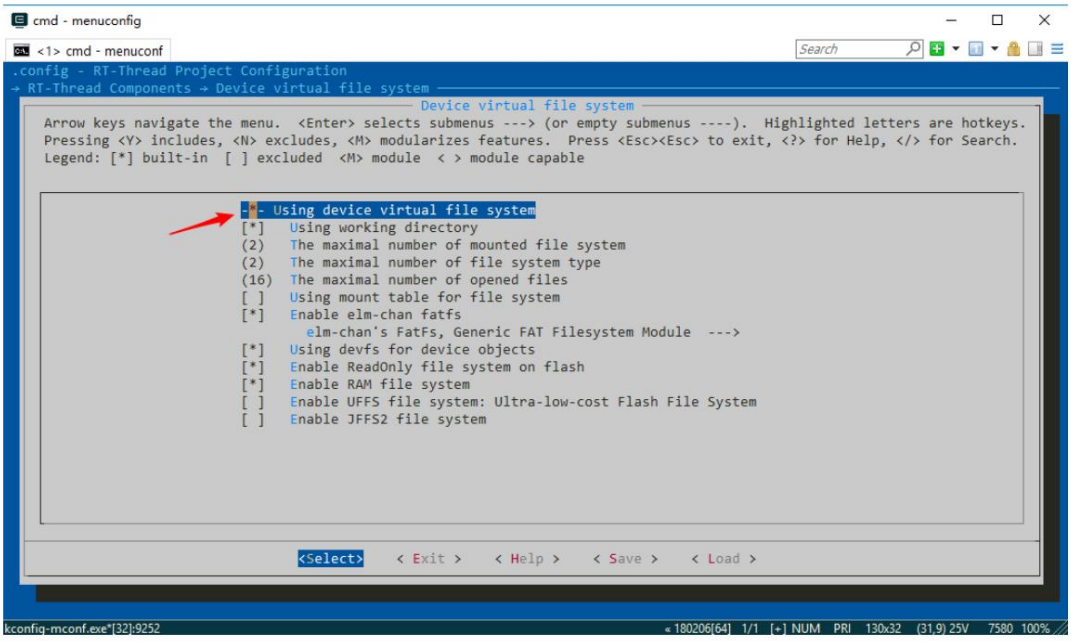

image **3:** Open file system

3.2 Compile Project

Use the scons command to compile the project.

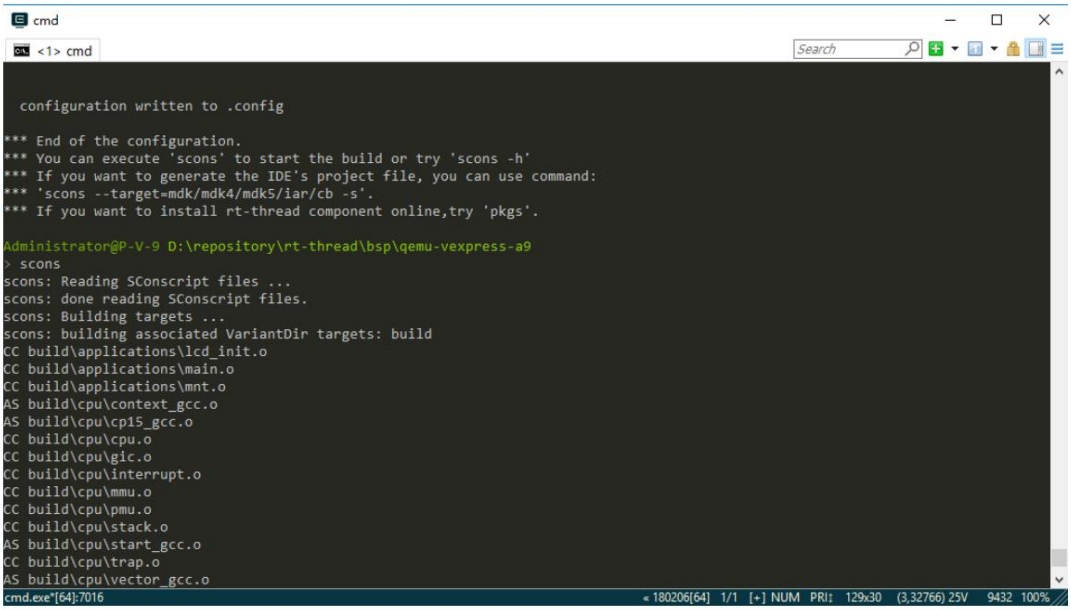

Figure **4:** Compile project

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.

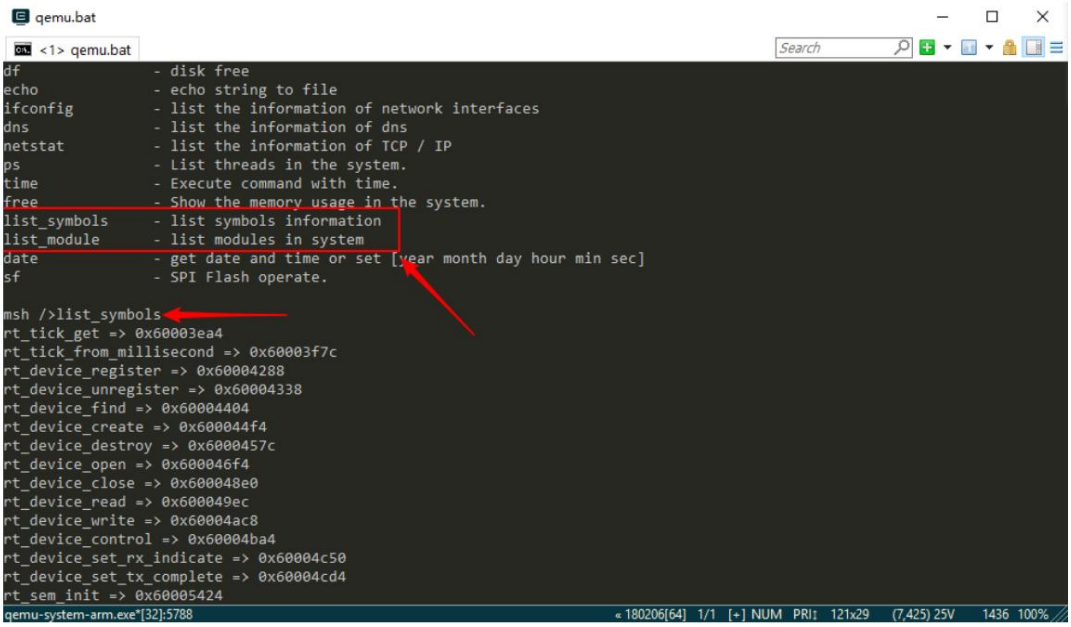

Figure **5:** Run dynamic module command

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.

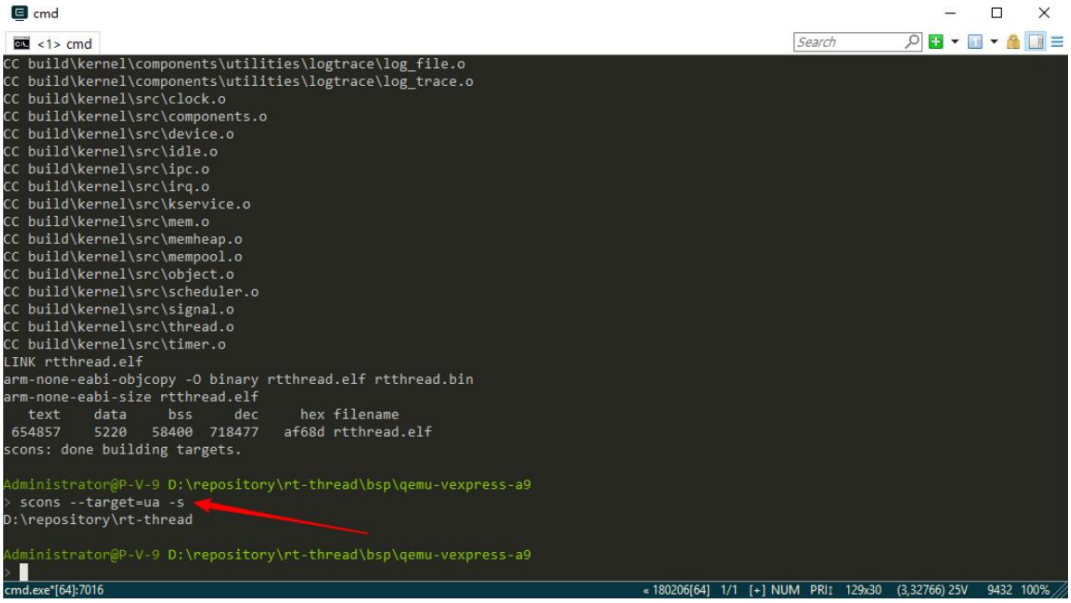

Figure 6: Generate dynamic module compilation dependency environment

4Running dynamic modules

4.1 Running the simplest dynamic module

4.1.1. Creating dynamic modules

4.1.1.1. 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.

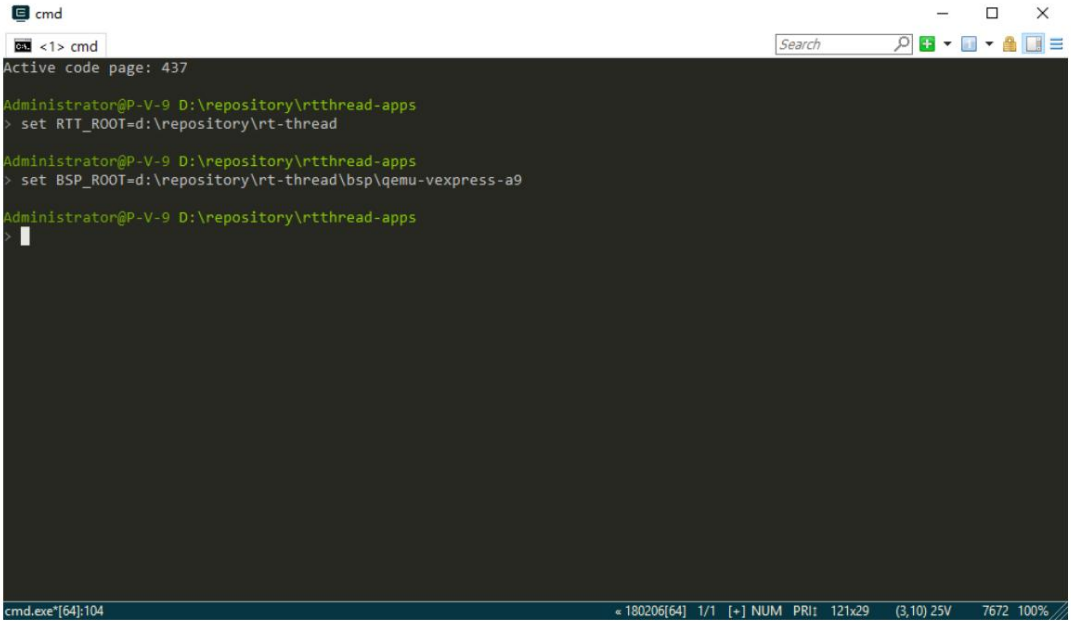

Figure 7: Setting Environment Variables

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

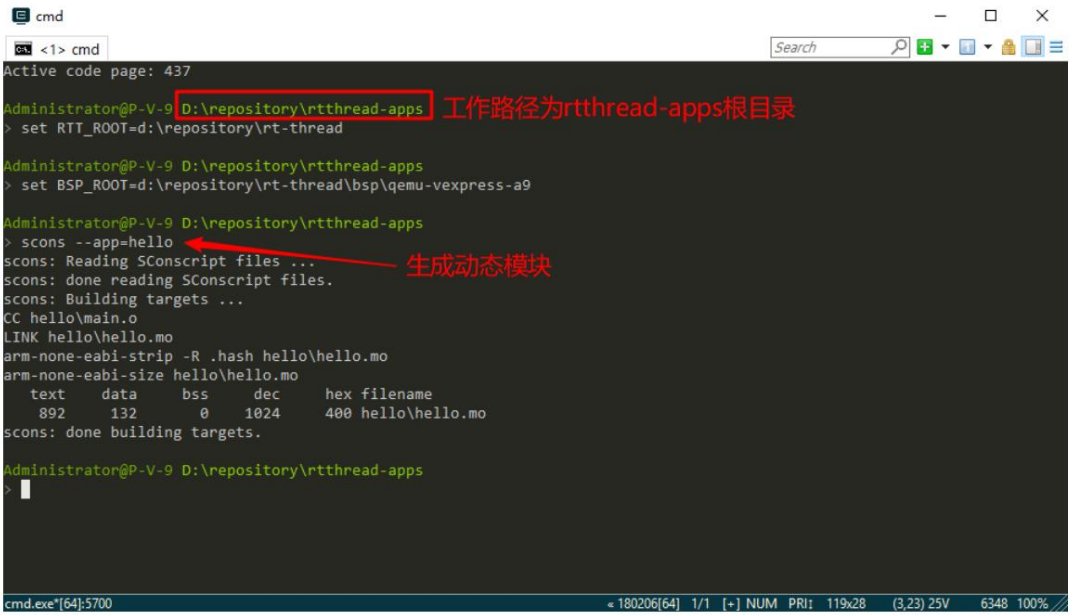

Figure 8: Compiling dynamic modules

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, 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.

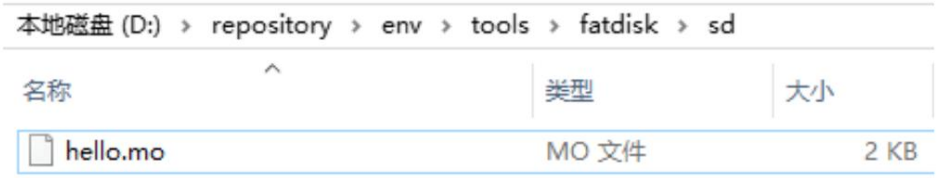

Figure **9:** Increase *fatdisk* Configuration Files

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.

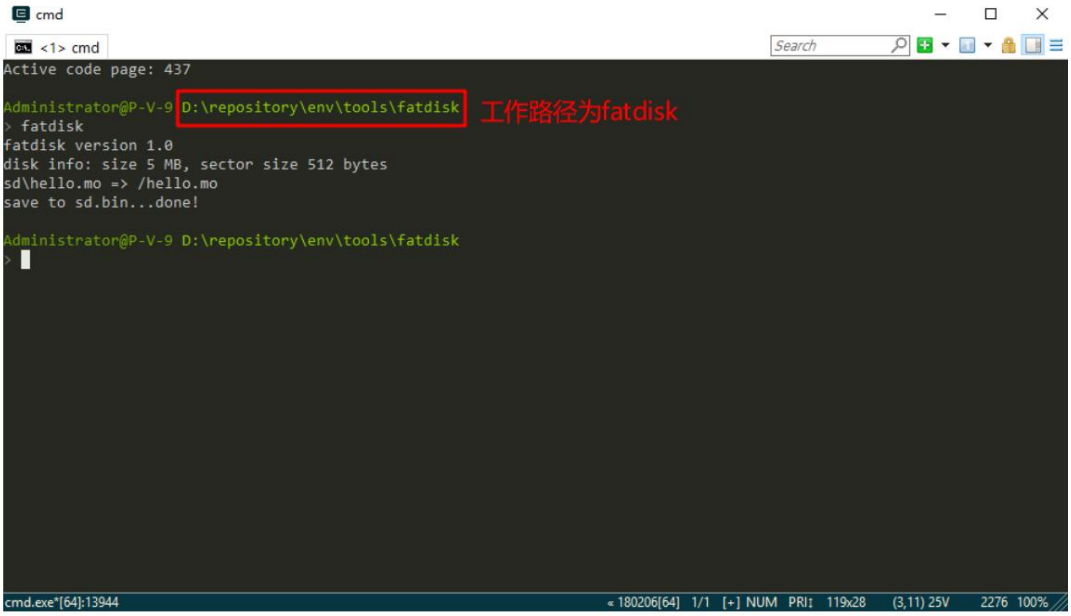

Figure **10:** run *fatdisk* Order

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

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.

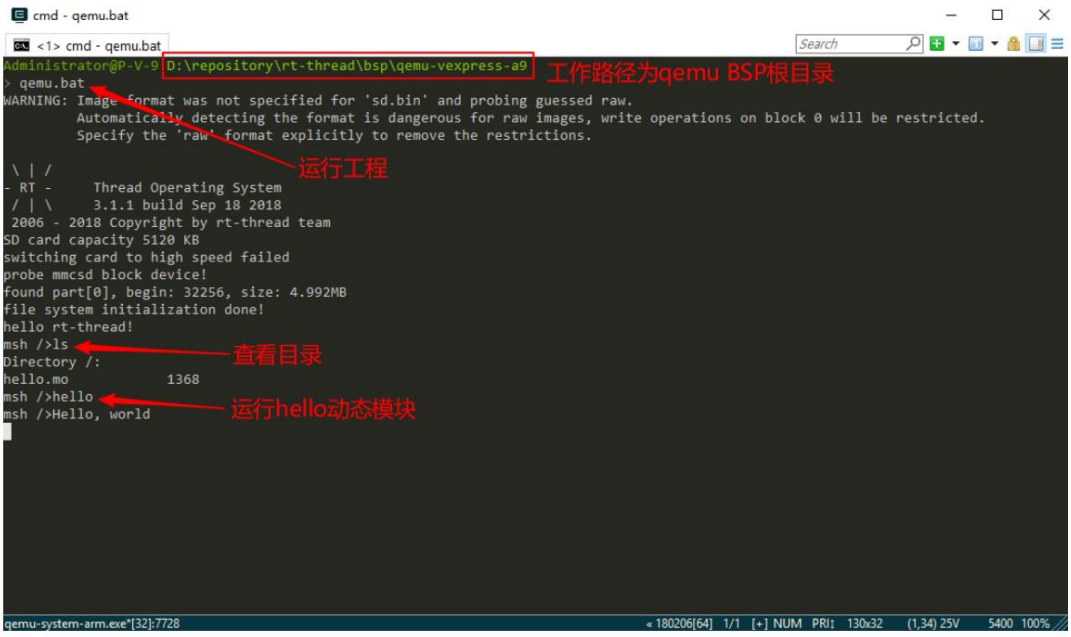

Figure **12:** run *qemu* project

- After the system is running, you will see the 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:

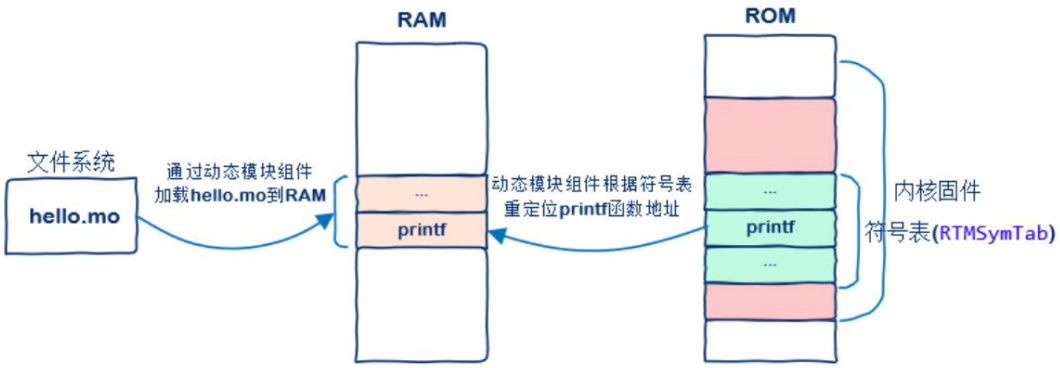

Figure **13:** How dynamic modules work

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.

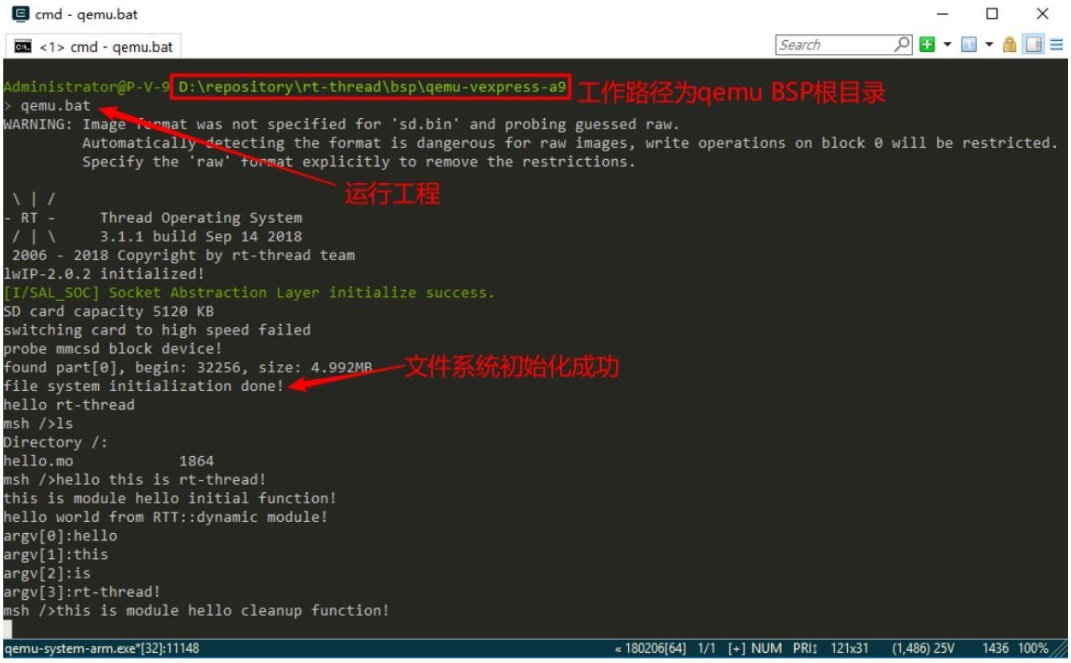

Figure **14:** run *qemu* project

• 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.

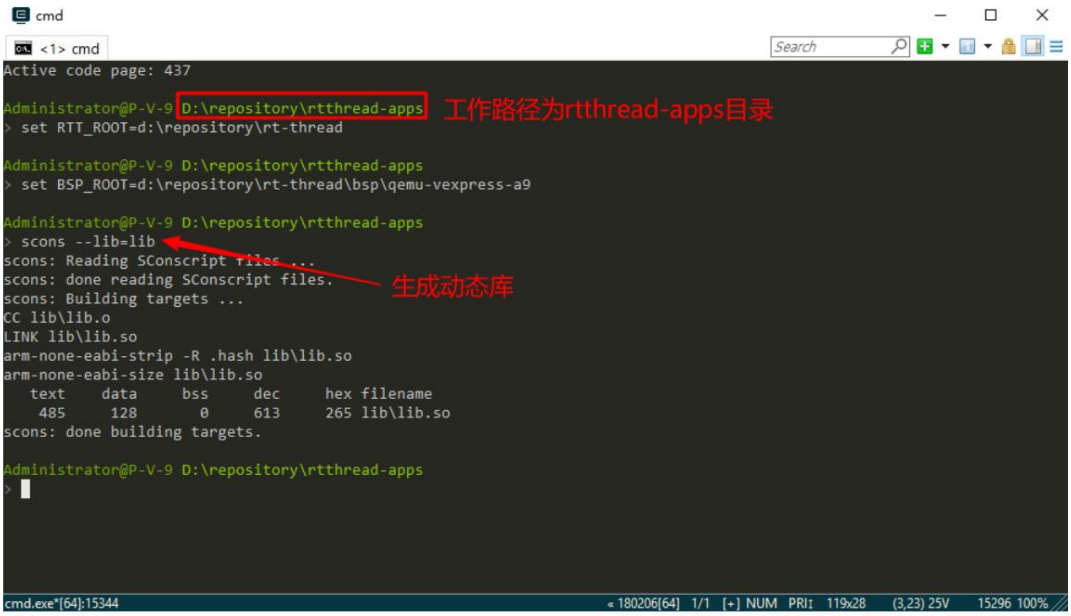

Figure **15:** Compiling dynamic modules

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.

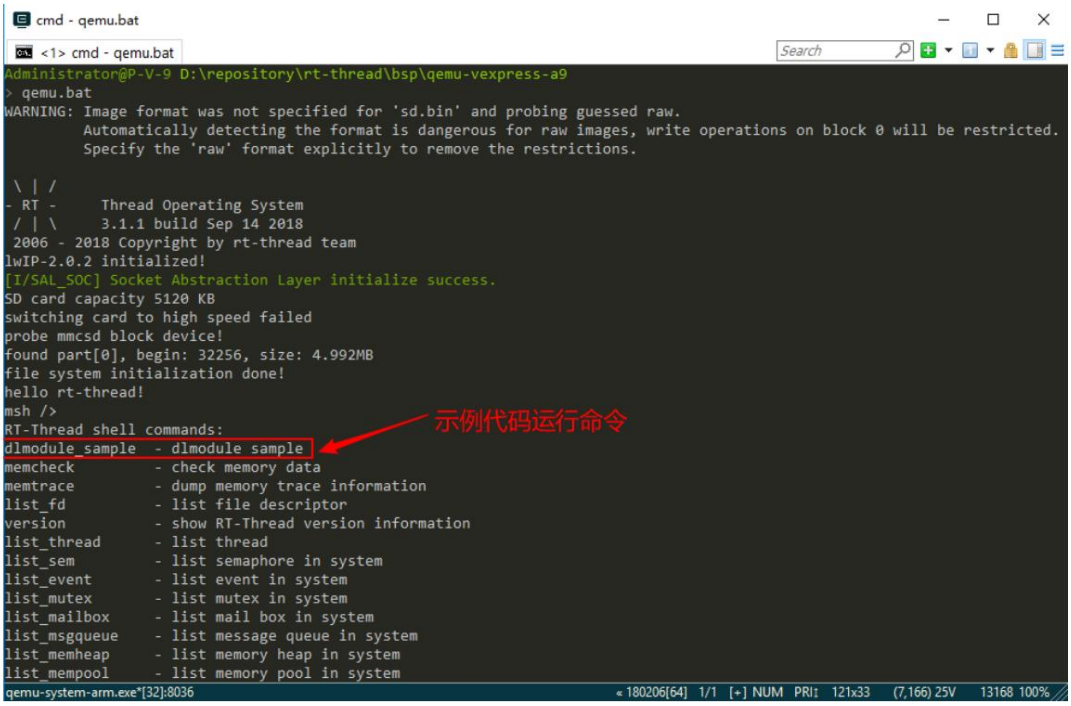

Figure **16:** run *qemu* project

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.

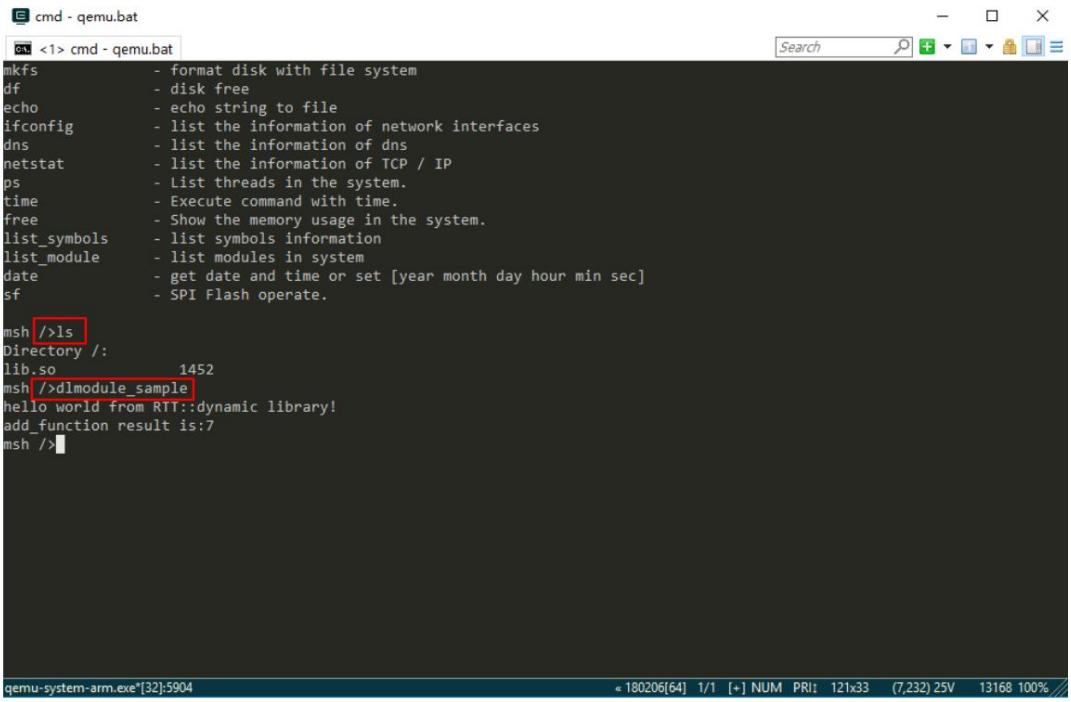

Figure **17:** Run the dynamic library example

• 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.

  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 run successfully according to the documentation.

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