在 RT-THREAD NANO 上添加控制台与 FINSH

RT-THREAD 文档中心

WWW.RT-THREAD.ORG

Thursday 26th December, 2019

目录

本片文档分为两部分：第一部分是实现 UART 控制台，该部分只需要实现两个函数即可完成 UART 控制台打印功能。第二部分是实现移植 FinSH 组件，实现在控制台输入命令调试系统，该部分实现基于第一部分，只需要添加 FinSH 组件源码并再对接一个系统函数即可实现。下面将对这两部分进行说明。

1 在 Nano 上添加 UART 控制台

在 RT-Thread Nano 上添加 UART 控制台打印功能后，就可以在代码中使用 RT-Thread 提供的打印函数 rt_kprintf() 进行信息打印，从而获取自定义的打印信息，方便定位代码 bug 或者获取系统当前运行状态等。实现控制台打印（需要确认 rtconfig.h 中已使能 `RT_USING_CONSOLE` 宏定义），需要完成基本的硬件初始化，以及对接一个系统输出字符的函数，本小节将详细说明。

1.1 实现串口初始化

使用串口对接控制台的打印，首先需要初始化串口，如引脚、波特率等。uart_init() 需要在 board.c 中的 `rt_hw_board_init()` 函数中调用。

```
/* 实现 1：初始化串口 */
static int uart_init(void);
```

示例代码：如下是基于 HAL 库的 STM32F103 串口驱动，完成添加控制台的示例代码，仅做参考。

```
static UART_HandleTypeDef UartHandle;
static int uart_init(void)
{
    /* 初始化串口参数，如波特率、停止位等等 */
    UartHandle.Instance = USART1;
    UartHandle.Init.BaudRate   = 115200;
    UartHandle.Init.HwFlowCtl  = UART_HWCONTROL_NONE;
    UartHandle.Init.Mode       = UART_MODE_TX_RX;
    UartHandle.Init.OverSampling = UART_OVERSAMPLING_16;
    UartHandle.Init.WordLength = UART_WORDLENGTH_8B;
    UartHandle.Init.StopBits   = UART_STOPBITS_1;
    UartHandle.Init.Parity     = UART_PARITY_NONE;

    /* 初始化串口引脚等 */
    if (HAL_UART_Init(&UartHandle) != HAL_OK)
    {
        while(1);
    }

    return 0;
}
INIT_BOARD_EXPORT(uart_init);
```

```
/* board.c */
void rt_hw_board_init(void)
{
```

```
    ....
    uart_init();              // 在 rt_hw_board_init 函数中调用 串口初始化 函数
    ....
}
```

1.2 实现 rt_hw_console_output

实现 finsh 组件输出一个字符，即在该函数中实现 uart 输出字符：

```
/* 实现 2：输出一个字符，系统函数，函数名不可更改 */
void rt_hw_console_output(const char *str);
```

!!! note "注意事项" 注意：RT-Thread 系统中已有的打印均以 \n 结尾，而并非 \r\n，所以在字符输出时，需要在输出 \n 之前输出 \r，完成回车与换行，否则系统打印出来的信息将只有换行。

示例代码：如下是基于 STM32F103 HAL 串口驱动对接的 rt_hw_console_output() 函数，实现控制台字符输出，示例仅做参考。

```
void rt_hw_console_output(const char *str)
{
    rt_size_t i = 0, size = 0;
    char a = '\r';

    __HAL_UNLOCK(&UartHandle);

    size = rt_strlen(str);
    for (i = 0; i < size; i++)
    {
        if (*(str + i) == '\n')
        {
            HAL_UART_Transmit(&UartHandle, (uint8_t *)&a, 1, 1);
        }
        HAL_UART_Transmit(&UartHandle, (uint8_t *)(str + i), 1, 1);
    }
}
```

1.3 结果验证

在应用代码中编写含有 rt_kprintf() 打印的代码，编译下载，打开串口助手进行验证。如下图是一个在 main() 函数中每隔 1 秒进行循环打印 Hello RT-Thread 的示例效果：

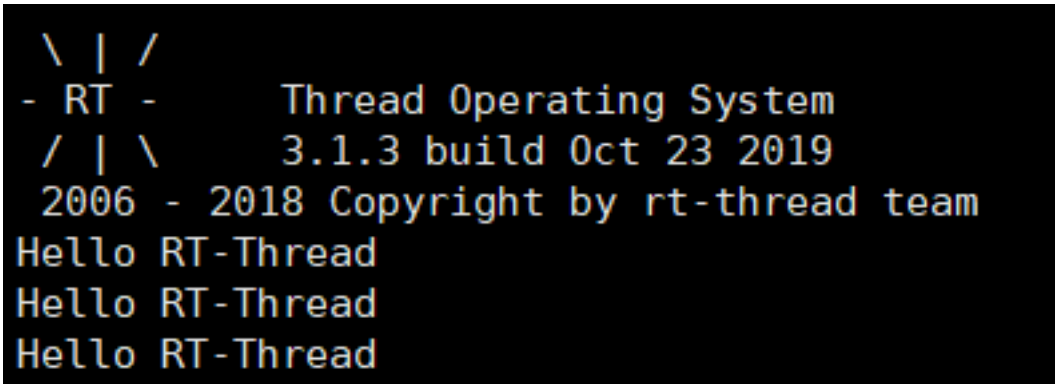

图 1: *示例*

2 在 Nano 上添加 FinSH 组件

RT-Thread FinSH 是 RT-Thread 的命令行组件（shell），提供一套供用户在命令行调用的操作接口，主要用于调试或查看系统信息。它可以使用串口 / 以太网 / USB 等与 PC 机进行通信，使用 FinSH 组件基本命令的效果图如下所示：

图 2: *效果图*

本文以串口 UART 作为 FinSH 的输入输出端口与 PC 进行通信，描述如何在 Nano 上实现 FinSH shell 功能。

在 RT-Thread Nano 上添加 FinSH 组件，实现 FinSH 功能的步骤主要如下：

1. 添加 FinSH 源码到工程。
2. 实现函数对接。

2.1 添加 FinSH 源码到工程

2.1.1. KEIL 添加 FinSH 源码

点击 Manage Run-Environment：

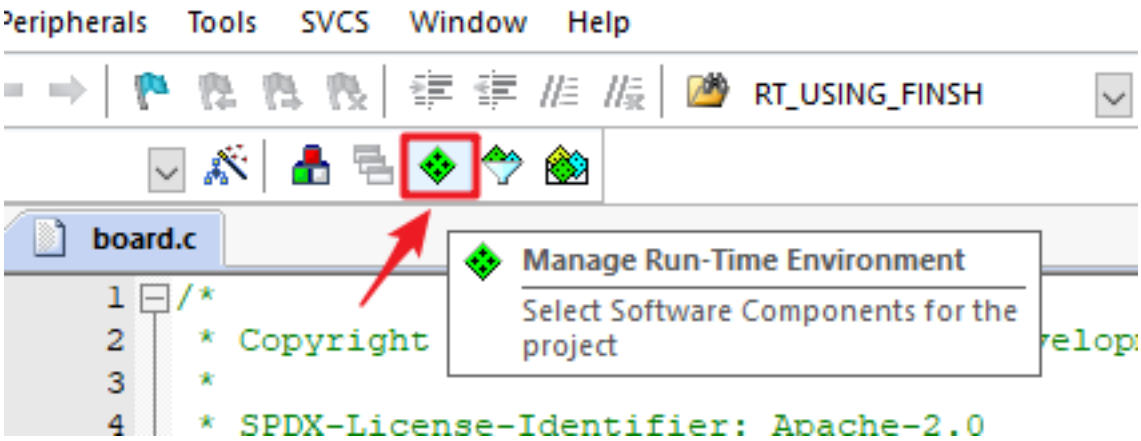

图 3: *Manage Run-Environment*

勾选 shell，这将自动把 FinSH 组件的源码到工程：

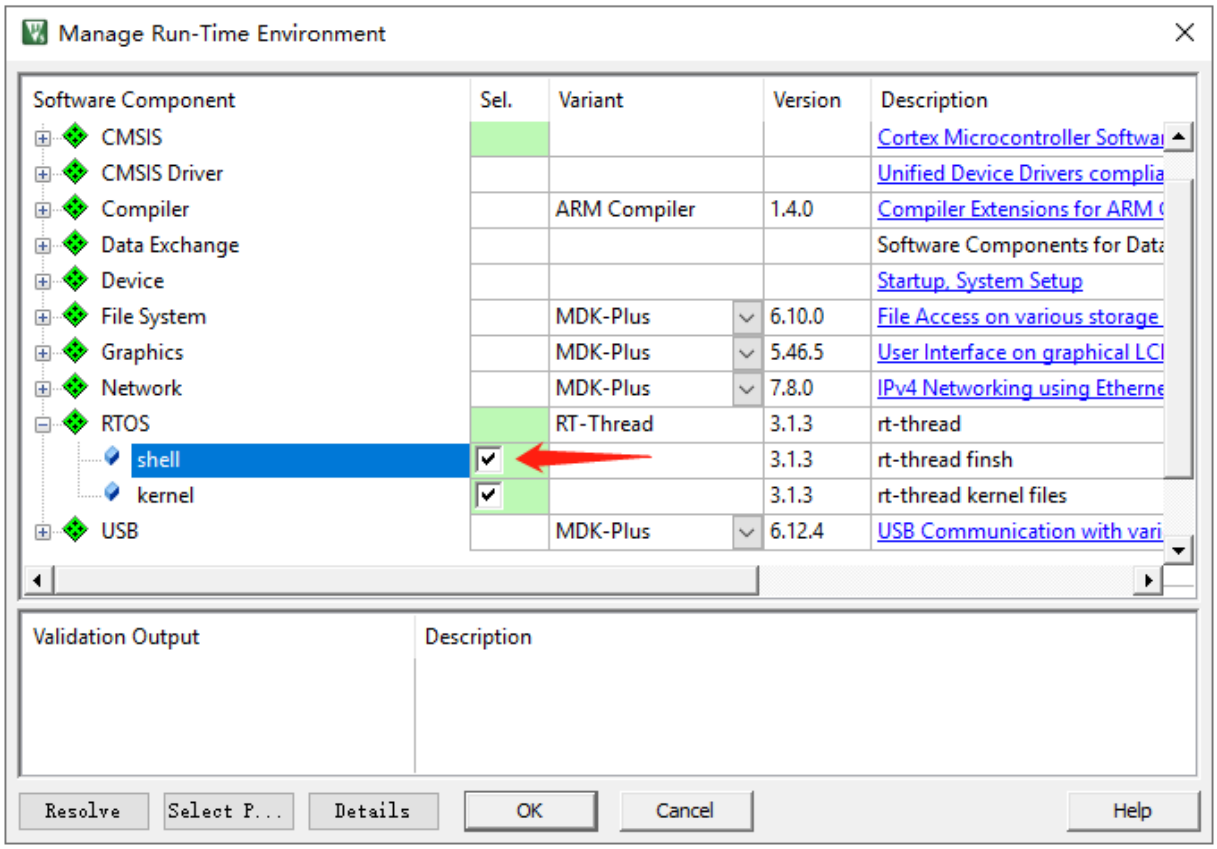

图 4: *勾选 shell*

2.1.2. Cube MX 添加 FinSH 源码

打开一个 cube 工程，点击 Additional Software，在 Pack Vendor 中可勾选 RealThread 快速定位 RT-Thread 软件包，然后在 RT-Thread 软件包中勾选 shell，即可添加 FinSH 组件的源码到工程中。

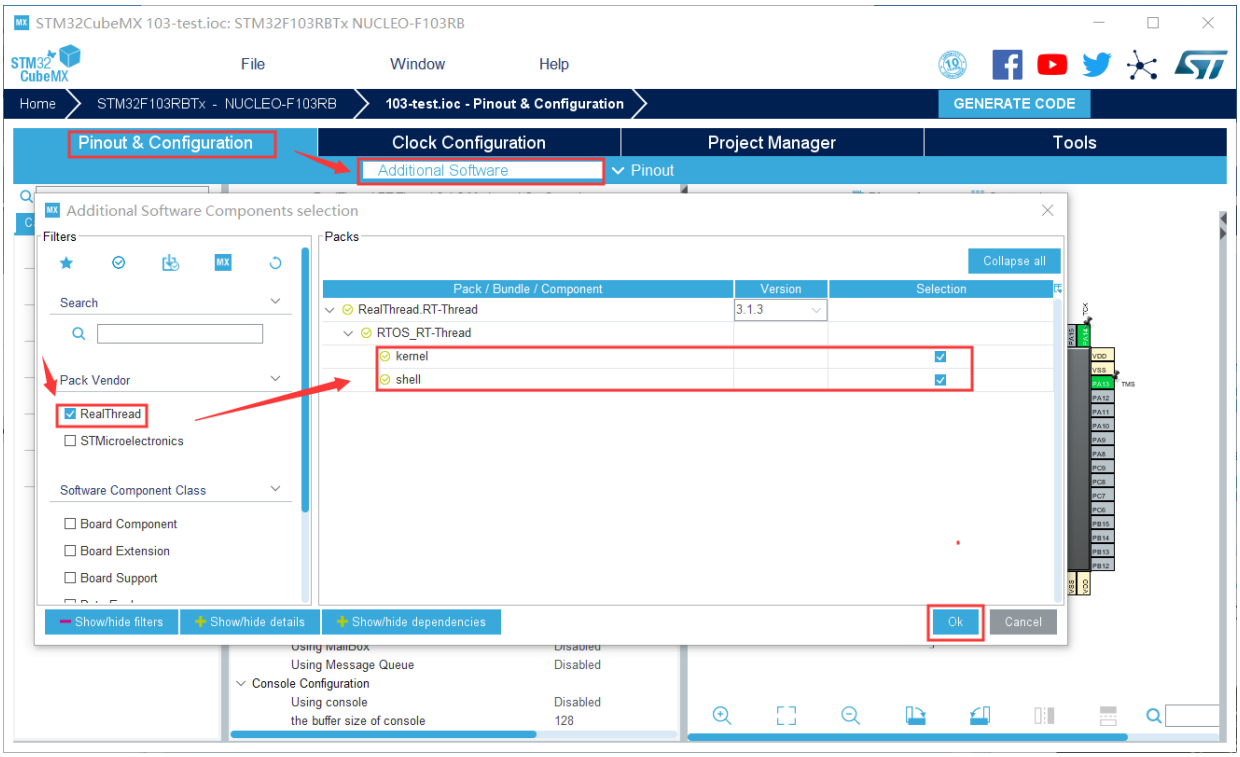

图 5: *Cube MX 添加 FinSH 源码*

2.1.3. 其他 IDE 添加 FinSH 源码

其他 IDE 添加 FinSH 源码，需要手动添加 FinSH 源码以及头文件路径到工程中，以 IAR IDE 为例进行结介绍。

1、复制 FinSH 源码到目标裸机工程：直接复制 Nano 源码中 rtthread-nano/components 文件夹下的 `finsh` 文件夹到工程中，如图：

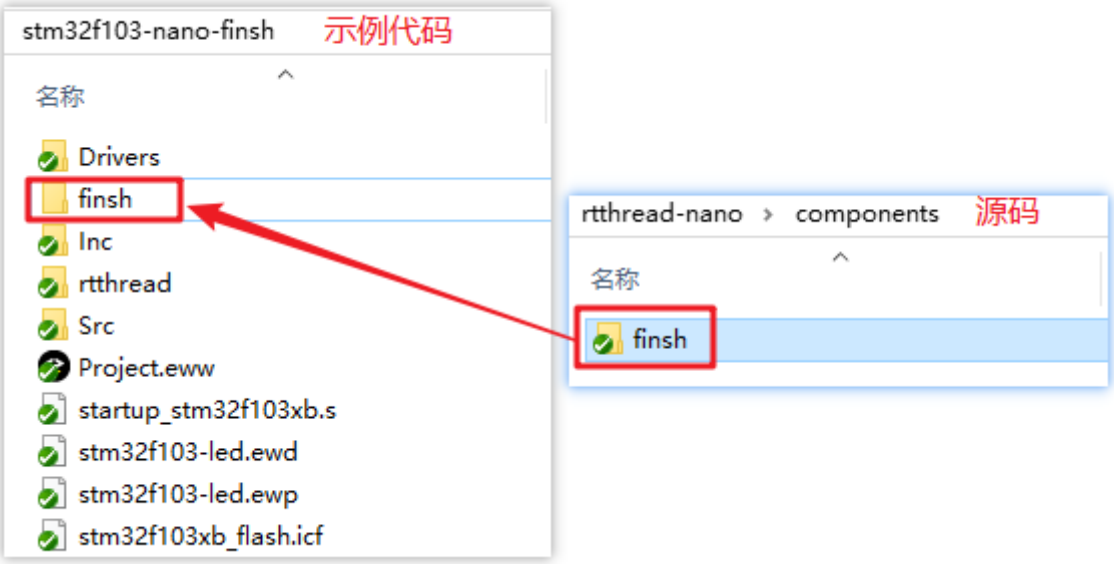

图 6: *复制 finsh 源码*

2、目标工程添加 FinSH 源码：

- 打开工程，新建 `finsh` 分组，添加工程中 `finsh` 文件夹下的所有. c 文件，如下图；
- 添加 `finsh` 文件夹的头文件路径（点击 `Project -> Options...` 进入弹窗进行添加，如下图）；
- 在 rtconfig.h 中添加 `##define RT_USING_FINSH` 宏定义，这样 FinSH 将生效，如下图。

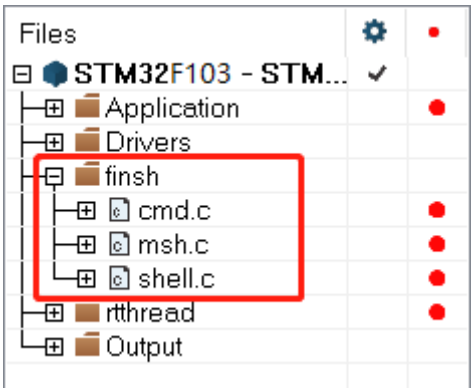

图 7: *添加 finsh 源码*

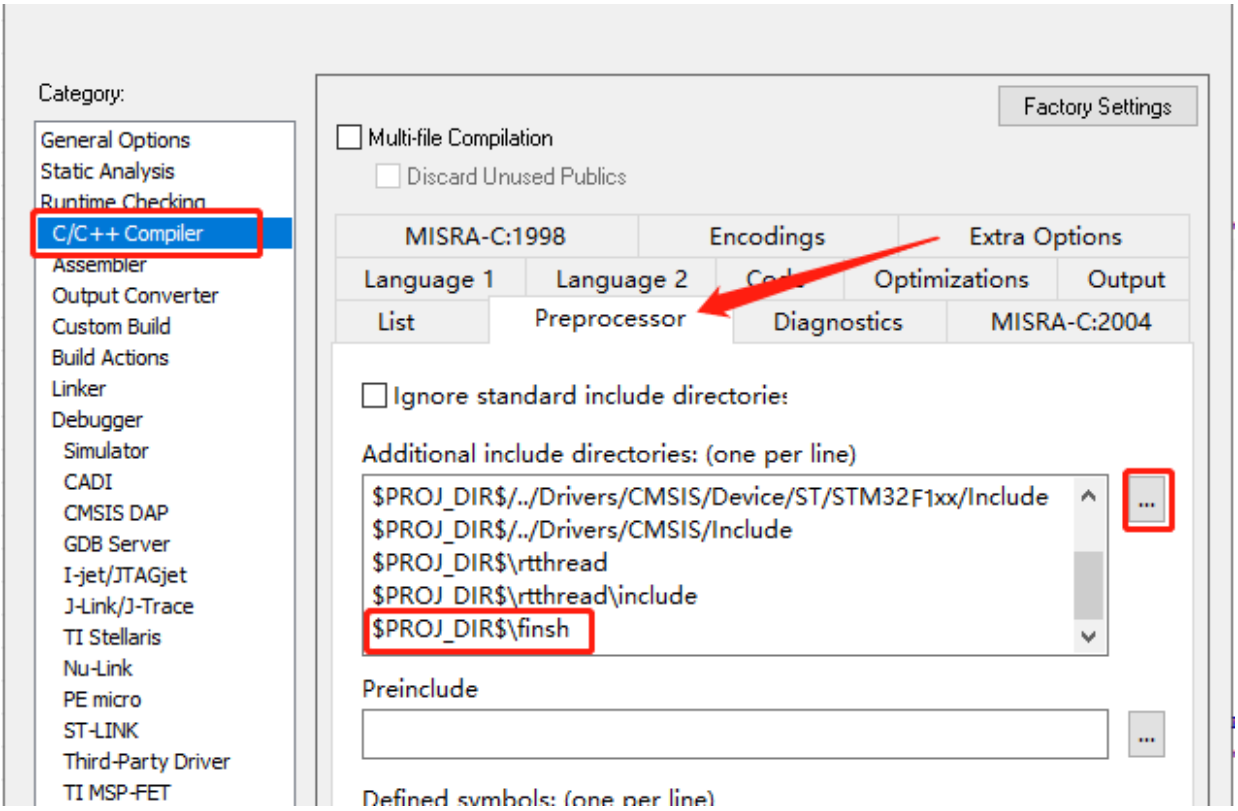

图 8: 添加 *finsh* 头文件路径

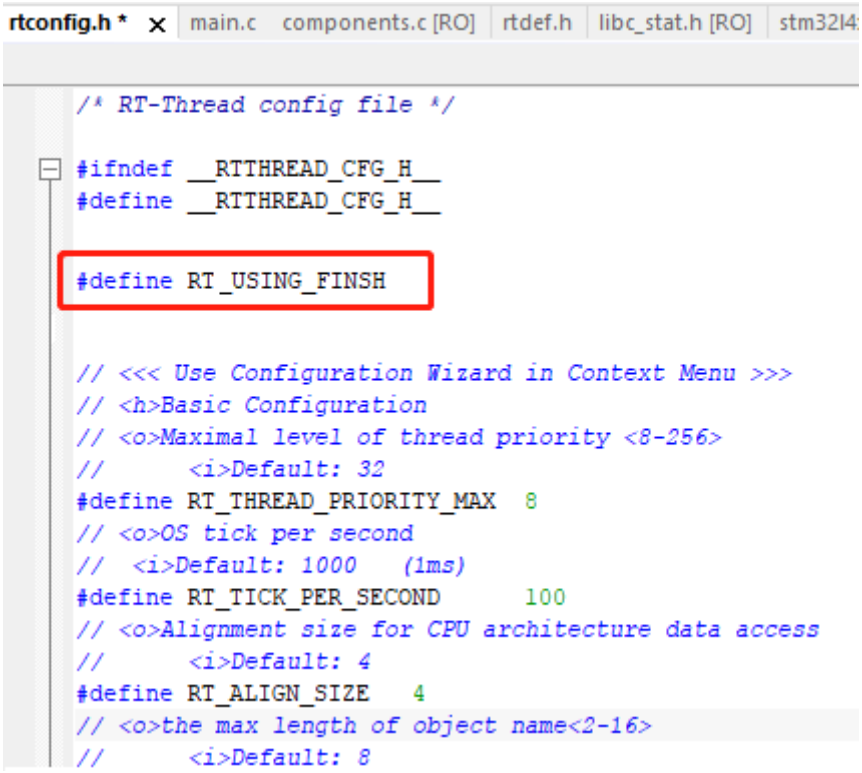

图 9: 添加 *finsh* 所需宏定义

2.2 实现 rt_hw_console_getchar

要实现 FinSH 组件功能：既可以打印也能输入命令进行调试，控制台已经实现了打印功能，现在还需要在 board.c 中对接控制台输入函数，实现字符输入：

```
/* 实现 3：finsh 获取一个字符，系统函数，函数名不可更改 */
char rt_hw_console_getchar(void);
```

- rt_hw_console_getchar()：控制台获取一个字符，即在该函数中实现 uart 获取字符，可以使用查询方式获取（注意不要死等，在未获取到字符时，需要让出 CPU），也可以使用中断方式获取。

示例代码：如下是基于 STM32F103 HAL 串口驱动对接的 rt_hw_console_getchar()，完成对接 FinSH 组件，其中获取字符采用查询方式，示例仅做参考。

```
char rt_hw_console_getchar(void)
{
    int ch = -1;

    if (__HAL_UART_GET_FLAG(&UartHandle, UART_FLAG_RXNE) != RESET)
    {
        ch = UartHandle.Instance->DR & 0xff;
    }
    else
    {
        if(__HAL_UART_GET_FLAG(&UartHandle, UART_FLAG_ORE) != RESET)
        {
            __HAL_UART_CLEAR_OREFLAG(&UartHandle);
        }
        rt_thread_mdelay(10);
    }
    return ch;
}
```

2.3 结果验证

编译下载代码，打开串口助手，可以在串口助手中打印输入 help 命令，回车查看系统支持的命令：

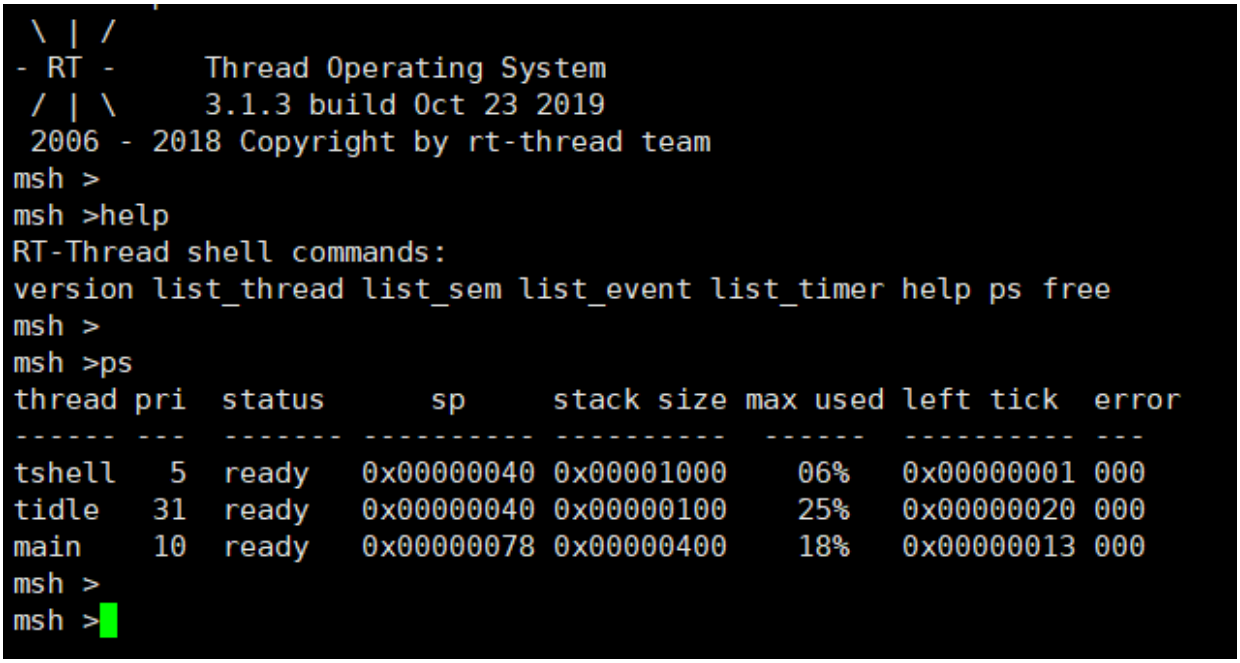

图 10: *下载验证 finsh*

如果没有成功运行，请检查对接的函数实现是否正确。

3 移植示例代码

3.1 轮询示例

如下是基于 STM32F103 HAL 串口驱动，实现控制台输出与 FinSH Shell，其中获取字符采用查询方式，示例仅做参考。

```
/* 初始化串口 */
static UART_HandleTypeDef UartHandle;
static int uart_init(void)
{
    /* 初始化串口参数，如波特率、停止位等等 */
    UartHandle.Instance = USART1;
    UartHandle.Init.BaudRate   = 115200;
    UartHandle.Init.HwFlowCtl  = UART_HWCONTROL_NONE;
    UartHandle.Init.Mode       = UART_MODE_TX_RX;
    UartHandle.Init.OverSampling = UART_OVERSAMPLING_16;
    UartHandle.Init.WordLength = UART_WORDLENGTH_8B;
    UartHandle.Init.StopBits   = UART_STOPBITS_1;
    UartHandle.Init.Parity     = UART_PARITY_NONE;

    /* 初始化串口引脚等 */
    if (HAL_UART_Init(&UartHandle) != HAL_OK)
    {
        while(1);
    }
```

```
    return 0;
}
INIT_BOARD_EXPORT(uart_init);

/* 移植控制台，实现控制台输出，对接 rt_hw_console_output */
void rt_hw_console_output(const char *str)
{
    rt_size_t i = 0, size = 0;
    char a = '\r';

    __HAL_UNLOCK(&UartHandle);

    size = rt_strlen(str);
    for (i = 0; i < size; i++)
    {
        if (*(str + i) == '\n')
        {
            HAL_UART_Transmit(&UartHandle, (uint8_t *)&a, 1, 1);
        }
        HAL_UART_Transmit(&UartHandle, (uint8_t *)(str + i), 1, 1);
    }
}

/* 移植 FinSH，实现命令行交互，需要添加 FinSH 源码，然后再对接 rt_hw_console_getchar
    */
/* 查询方式 */
char rt_hw_console_getchar(void)
{
    int ch = -1;

    if (__HAL_UART_GET_FLAG(&UartHandle, UART_FLAG_RXNE) != RESET)
    {
        ch = UartHandle.Instance->DR & 0xff;
    }
    else
    {
        if(__HAL_UART_GET_FLAG(&UartHandle, UART_FLAG_ORE) != RESET)
        {
            __HAL_UART_CLEAR_OREFLAG(&UartHandle);
        }
        rt_thread_mdelay(10);
    }
    return ch;
}
```

3.2 中断示例

如下是基于 STM32F103 HAL 串口驱动，实现控制台输出与 FinSH Shell，其中获取字符采用中断方式。原理是，在 uart 接收到数据时产生中断，在中断中把数据存入 ringbuffer 缓冲区，然后释放信号量，tshell 线程接收信号量，然后读取存在 ringbuffer 中的数据。示例仅做参考。

```
/* 第一部分：ringbuffer 实现部分 */
#include <rtthread.h>
#include <string.h>

#define rt_ringbuffer_space_len(rb) ((rb)->buffer_size - rt_ringbuffer_data_len(rb))

struct rt_ringbuffer
{
    rt_uint8_t *buffer_ptr;

    rt_uint16_t read_mirror : 1;
    rt_uint16_t read_index : 15;
    rt_uint16_t write_mirror : 1;
    rt_uint16_t write_index : 15;

    rt_int16_t buffer_size;
};

enum rt_ringbuffer_state
{
    RT_RINGBUFFER_EMPTY,
    RT_RINGBUFFER_FULL,
    /* half full is neither full nor empty */
    RT_RINGBUFFER_HALFFULL,
};

rt_inline enum rt_ringbuffer_state rt_ringbuffer_status(struct rt_ringbuffer *rb)
{
    if (rb->read_index == rb->write_index)
    {
        if (rb->read_mirror == rb->write_mirror)
            return RT_RINGBUFFER_EMPTY;
        else
            return RT_RINGBUFFER_FULL;
    }
    return RT_RINGBUFFER_HALFFULL;
}

/**
 * get the size of data in rb
 */
rt_size_t rt_ringbuffer_data_len(struct rt_ringbuffer *rb)
{
```

```
    switch (rt_ringbuffer_status(rb))
    {
    case RT_RINGBUFFER_EMPTY:
        return 0;
    case RT_RINGBUFFER_FULL:
        return rb->buffer_size;
    case RT_RINGBUFFER_HALFFULL:
    default:
        if (rb->write_index > rb->read_index)
            return rb->write_index - rb->read_index;
        else
            return rb->buffer_size - (rb->read_index - rb->write_index);
    };
}

void rt_ringbuffer_init(struct rt_ringbuffer *rb,
                        rt_uint8_t           *pool,
                        rt_int16_t            size)
{
    RT_ASSERT(rb != RT_NULL);
    RT_ASSERT(size > 0);

    /* initialize read and write index */
    rb->read_mirror = rb->read_index = 0;
    rb->write_mirror = rb->write_index = 0;

    /* set buffer pool and size */
    rb->buffer_ptr = pool;
    rb->buffer_size = RT_ALIGN_DOWN(size, RT_ALIGN_SIZE);
}

/**
 * put a character into ring buffer
 */
rt_size_t rt_ringbuffer_putchar(struct rt_ringbuffer *rb, const rt_uint8_t ch)
{
    RT_ASSERT(rb != RT_NULL);

    /* whether has enough space */
    if (!rt_ringbuffer_space_len(rb))
        return 0;

    rb->buffer_ptr[rb->write_index] = ch;

    /* flip mirror */
    if (rb->write_index == rb->buffer_size-1)
    {
        rb->write_mirror = ~rb->write_mirror;
        rb->write_index = 0;
```

```
    }
    else
    {
        rb->write_index++;
    }

    return 1;
}
/**
 * get a character from a ringbuffer
 */
rt_size_t rt_ringbuffer_getchar(struct rt_ringbuffer *rb, rt_uint8_t *ch)
{
    RT_ASSERT(rb != RT_NULL);

    /* ringbuffer is empty */
    if (!rt_ringbuffer_data_len(rb))
        return 0;

    /* put character */
    *ch = rb->buffer_ptr[rb->read_index];

    if (rb->read_index == rb->buffer_size-1)
    {
        rb->read_mirror = ~rb->read_mirror;
        rb->read_index = 0;
    }
    else
    {
        rb->read_index++;
    }

    return 1;
}


/* 第二部分：finsh 移植对接部分 */
#define UART_RX_BUF_LEN 16
rt_uint8_t uart_rx_buf[UART_RX_BUF_LEN] = {0};
struct rt_ringbuffer  uart_rxcb;          /* 定义一个 ringbuffer cb */
static UART_HandleTypeDef UartHandle;
static struct rt_semaphore shell_rx_sem; /* 定义一个静态信号量 */

/* 初始化串口，中断方式 */
static int uart_init(void)
{
    /* 初始化串口接收 ringbuffer  */
    rt_ringbuffer_init(&uart_rxcb, uart_rx_buf, UART_RX_BUF_LEN);
```

```
    /* 初始化串口接收数据的信号量 */
    rt_sem_init(&(shell_rx_sem), "shell_rx", 0, 0);

    /* 初始化串口参数，如波特率、停止位等等 */
    UartHandle.Instance = USART2;
    UartHandle.Init.BaudRate   = 115200;
    UartHandle.Init.HwFlowCtl  = UART_HWCONTROL_NONE;
    UartHandle.Init.Mode       = UART_MODE_TX_RX;
    UartHandle.Init.OverSampling = UART_OVERSAMPLING_16;
    UartHandle.Init.WordLength = UART_WORDLENGTH_8B;
    UartHandle.Init.StopBits   = UART_STOPBITS_1;
    UartHandle.Init.Parity     = UART_PARITY_NONE;

    /* 初始化串口引脚等 */
    if (HAL_UART_Init(&UartHandle) != HAL_OK)
    {
        while (1);
    }

    /* 中断配置 */
    __HAL_UART_ENABLE_IT(&UartHandle, UART_IT_RXNE);
    HAL_NVIC_EnableIRQ(USART2_IRQn);
    HAL_NVIC_SetPriority(USART2_IRQn, 3, 3);

    return 0;
}
INIT_BOARD_EXPORT(uart_init);

/* 移植控制台，实现控制台输出，对接 rt_hw_console_output */
void rt_hw_console_output(const char *str)
{
    rt_size_t i = 0, size = 0;
    char a = '\r';

    __HAL_UNLOCK(&UartHandle);

    size = rt_strlen(str);
    for (i = 0; i < size; i++)
    {
        if (*(str + i) == '\n')
        {
            HAL_UART_Transmit(&UartHandle, (uint8_t *)&a, 1, 1);
        }
        HAL_UART_Transmit(&UartHandle, (uint8_t *)(str + i), 1, 1);
    }
}

/* 移植 FinSH，实现命令行交互，需要添加 FinSH 源码，然后再对接 rt_hw_console_getchar
    */
```

```
/* 中断方式 */
char rt_hw_console_getchar(void)
{
    char ch = 0;

    /* 从 ringbuffer 中拿出数据 */
    while (rt_ringbuffer_getchar(&uart_rxcb, (rt_uint8_t *)&ch) != 1)
    {
        rt_sem_take(&shell_rx_sem, RT_WAITING_FOREVER);
    }
    return ch;
}

/* uart 中断 */
void USART2_IRQHandler(void)
{
    int ch = -1;
    rt_base_t level;
    /* enter interrupt */
    rt_interrupt_enter();          //在中断中一定要调用这对函数，进入中断

    if ((__HAL_UART_GET_FLAG(&(UartHandle), UART_FLAG_RXNE) != RESET) &&
        (__HAL_UART_GET_IT_SOURCE(&(UartHandle), UART_IT_RXNE) != RESET))
    {
        while (1)
        {
            ch = -1;
            if (__HAL_UART_GET_FLAG(&(UartHandle), UART_FLAG_RXNE) != RESET)
            {
                ch =  UartHandle.Instance->DR & 0xff;
            }
            if (ch == -1)
            {
                break;
            }
            /* 读取到数据，将数据存入 ringbuffer */
            rt_ringbuffer_putchar(&uart_rxcb, ch);
        }
        rt_sem_release(&shell_rx_sem);
    }

    /* leave interrupt */
    rt_interrupt_leave();    //在中断中一定要调用这对函数，离开中断
}

#define USART_TX_Pin GPIO_PIN_2
#define USART_RX_Pin GPIO_PIN_3

void HAL_UART_MspInit(UART_HandleTypeDef *huart)
```

```
{
    GPIO_InitTypeDef GPIO_InitStruct = {0};
    if (huart->Instance == USART2)
    {
        __HAL_RCC_USART2_CLK_ENABLE();

        __HAL_RCC_GPIOA_CLK_ENABLE();
        /**USART2 GPIO Configuration
        PA2     ------> USART2_TX
        PA3     ------> USART2_RX
        */
        GPIO_InitStruct.Pin = USART_TX_Pin | USART_RX_Pin;
        GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
        GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
        HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
    }
}
```

4 常见问题

4.1 Q: rt_kprintf() 不能打印浮点数吗？

A: 不可以。但是可以通过其他方法实现打印浮点数的目的，比如成倍扩大数值后，分别打印整数与小数部分。

4.2 Q: 在实现 FinSH 完整功能时，却不能输入。

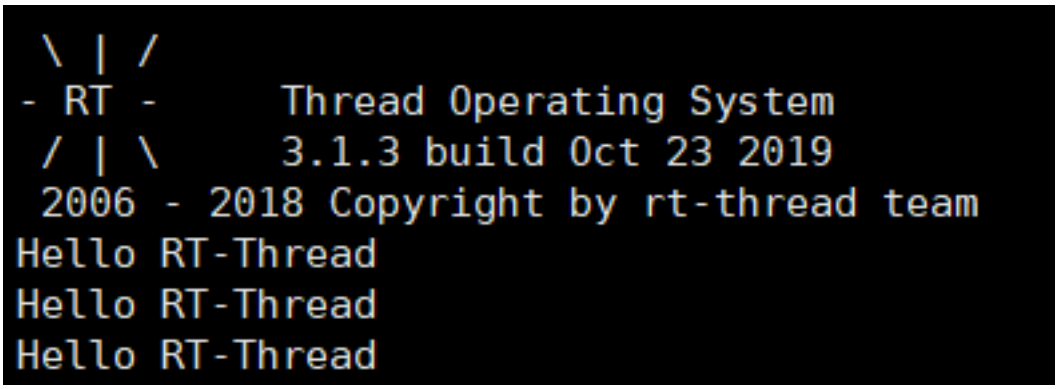

图 11: *示例*

A：可能的原因有：UART 驱动未实现字符输入函数、未打开 FinSH 组件等；如果手动开启了 HEAP，需要确定 HEAP 是否过小，导致 tshell 线程创建失败。

4.3 Q: 出现 hard fault。

A：ps 后关注各个线程栈的最大利用率，若某线程出现 100% 的情况，则表示该线程栈过小，需要将值调大。