使用 finsh

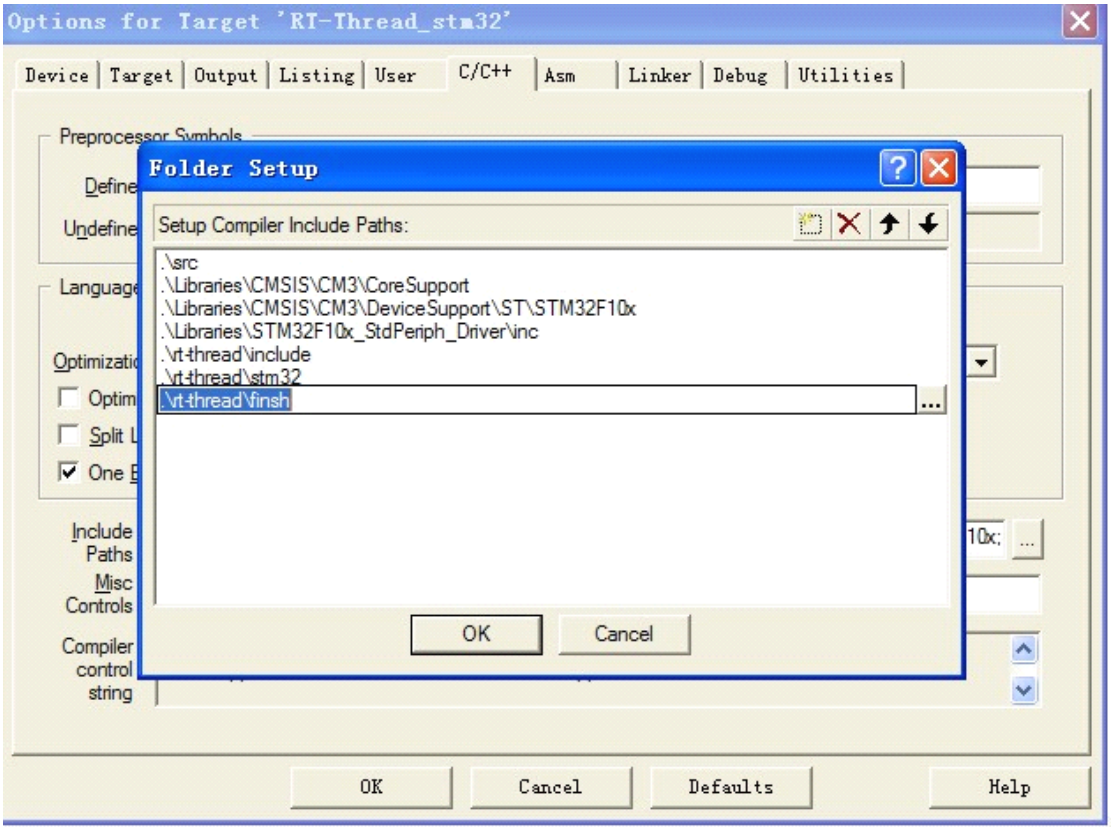
本例程和目标 CPU 配置无关,已把芯片内部 SRAM 容量配置为 20KB,C8T6~ZET6 通用.
除使用 USART1 外不使用其它外设.

Finsh 是一个很有用的调试-交互组件,使用 Finsh 可以查看所有线程,信号量,设备等信息.是调试系统的得力助手,下面介绍下如何使用 finsh.

第一步: 添加 Finsh 源文件

本例程基于 base_kernel 例程修改
既然要使用 Finsh,首先是添加 Finsh 的全部源文件.见:rt-thread/finsh.

然后更新头文件搜索路径:



第二步: 配置使用 Finsh 并初始化 Finsh.

作为 RT-Thread 的一个组件,要使用 Finsh,需要在 rtconfig.h 中开启使用 Finsh.

```
/* SECTION: finsh, a C-Express shell */
#define RT_USING_FINSH                  //定义使用 FINSH
/* Using symbol table */
#define FINSH_USING_SYMTAB
#define FINSH_USING_DESCRIPTION
```

然后需要在初始化 RT-Thread 时把 Finsh 也一并初始化.

```
    /* init application */
    rt_application_init();

#ifdef RT_USING_FINSH
     /* init finsh */
     finsh_system_init();               //初始化 FINSH 线程
     finsh_set_device("uart1");         //设置 FINSH 使用的输入输出设备为 "uart1"
#endif
```

在以前,FINSH 需要自己实现输入和输出,现改进为使用 RT-Thread 的设备管理框架来实现输入输出,这样可以很方便地使用"uart2"来操作 Finsh,甚至是网络,类似于 telnet.
极大地增强了灵活性,

所以,在下面,我们还需要实现这个 "uart1"设备.

第三步: 实现统一的设备管理

在 base_kernel 例程中,为了使例程变成简单,我们实现了一个最简单的通过查询方式的串口输出程序.缺点很明显:没有灵活性,性能也只能做来做演示.

RT-Thread 提供了一个完整的设备管理方案,可以把不同类型的外设都按统一的接口进行操作,通过设备名称就可以找到设备,然后就可以直接访问了.

因为 STM32 的各型号外设兼容性极高,所以 RT-Thread 为 STM32 的 USART 提供了一个通用框架.
rt-thread/stm32/serial.c
rt-thread/stm32/serial.h
之所以写一个框架是因为 STM32 有多达 5 个串口.这样是为了提高程序的重用性.

然后在 /src/usart.c 把每个串口注册成设备.因 STM32 有多个串口,所以,根据需要来使用.
1.rtconfig.h 打开使用 RT-Thread 的设备管理

```
/* SECTION: Device System */
/* Using Device System */
#define RT_USING_DEVICE                 //定义使用 RT-Thread 的设备管理
#define RT_USING_UART1                  //使用 UART1 在 /src/usart.c 中有用到.
```

2. 取消原来的简单串口输出程序
把原来 board.c 中的 rt_hw_console_init(); 函数取消掉,换成 usart 中的 rt_hw_usart_init();
RT-Thread 中的 rt_kprinf() 也可以使用设备来输出,只需要
rt_console_set_device("uart1");
就可以让 rt_kprinf() 通过"uart1"来输出,原来的 rt_hw_console_output() 也可以取消掉.
因为设置 rt_kprinf() 通过设备来输出后,就不会再通过 rt_hw_console_output() 来输出数据了.

3.添加 USART1 中断服务程序,在前面我们仅实现输出,通过查询方式,没有使用中断。

```
#include <rtthread.h>
void USART1_IRQHandler(void)
{
    extern struct rt_device uart1_device;
     extern void rt_hw_serial_isr(struct rt_device *device);

    /* enter interrupt */
     rt_interrupt_enter();
     rt_hw_serial_isr(&uart1_device);

    /* leave interrupt */
    rt_interrupt_leave();
}
```

第四步: 写一个简单应用来使用 Finsh

因为原来 app.c 中有两个线程,都有使用 rt_kprintf 来打印信息,因 console 和 finsh 现都使用 "uart1",这两个线程打印的信息可能会影响观察,我们先注释掉,然后写个最简单函数.

```
#include <finsh.h>
void fun_a(int input)
{
    rt_kprintf("input : %d \r\n",input);
    rt_kprintf("fun_a done.\r\n");
}
FINSH_FUNCTION_EXPORT(fun_a, fun_a desc);

void fun_b(int input)
{
    rt_kprintf("input : %d \r\n",input);
    rt_kprintf("fun_b done.\r\n");
}
FINSH_FUNCTION_EXPORT(fun_b, fun_b desc); //同上
```

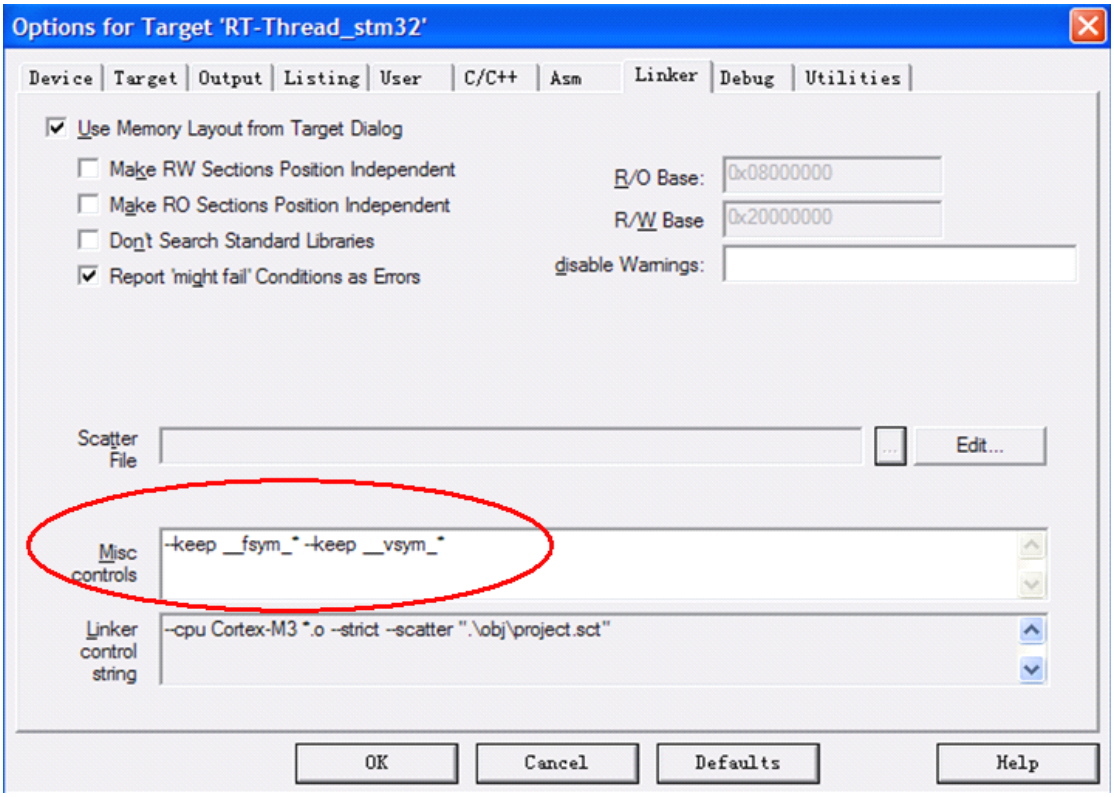
这样,当我们在 finsh 中执行 fun_a() 时,就会运行 fun_a 这个函数(**注意加回车**).

```
finsh>> fun_a(9)
finsh>> fun_b(10)
finsh>> int fun_val = 20
finsh>> fun_a(fun_val)
finsh>> fun_b(++fun_val)
finsh>> fun_a(fun_val *2)
```

Finsh 使用秘籍:
1.finsh>>状态下按 TAB 有惊喜.
2.输入 fun 再按 TAB 依然有惊喜.(可以输入 list 再按 TAB 测试下)
3.执行一个函数退出后再按上,下键可浏览历史.

后记:

因为需要导出一些符号给 finsh shell 使用，也就是那些函数名，变量名.
而这些函数名和变量名统一的放在两个 section 中，这样 finsh shell 能够自动的到这两个段中寻找.
而因为上面的 fun_a() fun_b() 并没有在别处被引用,因此在链接时可能被链接程序自动移除掉.
所以需要添加如下图链接选项让链接器保留这些段.
"--keep __fsym_* --keep __vsym_*" 详细请看 FINSH_FUNCTION_EXPORT 的实现


Options for Target 'RT-Thread_stm32'
Device
Target
Output
Listing
User
C/C++
Asm
Linker
Debug
Utilities
Use Memory Layout from Target Dialog
Make RW Sections Position Independent
Make RO Sections Position Independent
Don't Search Standard Libraries
Report 'might fail' Conditions as Errors
R/O Base:
0x08000000
R/W Base
0x20000000
disable Warnings:
Scatter File
Edit...
Misc controls
--keep __fsym_* --keep __vsym_*
Linker control string
--cpu Cortex-M3 *.o --strict --scatter ".\obj\project.sct"
OK
Cancel
Defaults
Help