蜂鸟 E203 移植 FreeRTOS

Content

0 Preface

0.1 Revision History

Date	Version	Author	Change Summary
Oct 20,2018	0.1	Bob Hu	Initial version
Nov 7,2018	0.2	Bob Hu	Fixed some typo

1 RTOS 简述

实时操作系统（RTOS）是指当外界事件或者数据产生时，能够接受并以足够快的速度予以处理，处理的结果又能在规定的时间内来控制生产过程或对处理系统能够做出快速响应，调度一切可利用的资源完成实时任务，并控制所有实时任务协调一致运行的操作系统。主要特点是提供及时响应和考可靠性。

在服务器、个人电脑、手机上运行的操作系统，譬如 Windows 和 Linux，强调在一处理器上能运行更多任务。此类操作系统的代码均具有一定规模，并且不一定能保证实时性。而对于处理器硬件资源有限，对实时性又有特殊要求的嵌入式应用领域，就需要一种代码规模适中，实时性好的操作系统。

实时性可以分为硬实时和软实时。硬实时的功能是必须在给定时间内完成操作，如果不能完成将可能导致严重后果。比如汽车安全气囊触发机制就是一个很好的硬实时的例子，在撞击后安全气囊必须在给定时间内弹出，如果响应时间超出给定时间，可能使驾驶员受到严重伤害。

对于软实时，一个典型的实例是 IPTV 数字电视机顶盒，需要实时的解码视频流，如果丢失了一个或几个视频帧，视频品质也不会相差多少。软实时系统从统计角度来说，一个任务有确定的执行时间，事件在截止时间到来之前也能得到处理，即使违反截止时间也不会带来致命的错误。

2 常用实时操作系统概述

常用的实时操作系统（RTOS）有以下几种：FreeRTOS、VxWorks、uc/os-II、uclinuxeCos、RT-Thread 和 SylixOS 等。下面分别对这几种 RTOS 进行介绍说明。

- SylixOS：
  - 翼辉 SylixOS 实时操作系统是一款功能全面、稳定可靠、易于开发的国产实时系统平台。其解决方案覆盖网络设备、国防安全、工业自动化、轨道交通、电力、医疗、航空航天等诸多领域。SylixOS 是国内唯一一款支持 SMP 的大型实时操作系统。翼辉开发嵌入式操作系统 SylixOS 始于 2006 年，至今在军工领域已有众多项目或产品基于 SylixOS 进行开发，例如雷达、弹载飞控系统、星载任务计算机、机载火控系统、计重收费与超限检测仪、火灾报警系统、特种车辆与船用发动机状态显示器、潜艇蓄电池监控系统、轮式装甲车实时监控系统等，其中大部分产品都要求 7*24 小时不间断运行，当前很多 SylixOS 系统节点已不间断运行超过 5 万小时（6 年）。
- RT-Thread：
  - RT-Thread 是一款主要由中国开源社区主导开发的开源实时操作系统（许可证 GPLv2）。实时线程操作系统不仅是一个单一的实时操作系统内核，它也是一个完整的应用系统，包含了实时、嵌入式系统相关的各个组件：TCP/IP 协议栈、文件系统、libc 接口、图形用户界面等。RT-Thread 拥有良好的软件生态，支持市面上所有主流的编译工具如 GCC、Keil、IAR 等，工具链完善、友好，支持各类标准接口，如 POSIX、CMSIS、C++应用环境、Javascript 执行环境等，方便开发者移植各类应用程序。商用支持所有主流 MCU 架构，如 ARM Cortex-M/R/A、MIPS、X86、Xtensa、Andes、C-Sky、RISC-V，几乎支持市场上所有主流的 MCU 和 Wi-Fi 芯片
- FreeRTOS：
  - 有关 FreeRTOS 见第 3 章。
- VxWorks：
  - 由美国 WindRiver 公司于 1983 年推出的一款实时操作系统。由于其良好的持续发展能力，高性能内核以及友好的开发环境，因此在嵌入式系统领域占有一席之地。VxWorks 由 400 多个相对独立、短小精悍的目标模块组成，用户可根据需要进行配置和裁剪，在通信、军事、航天、航空等领域应用广泛。
- uc/os-II：
  - 前身是 uc/os，最早由 1992 年美国嵌入式专家 Jean J.Labrosse 在《嵌入式系统编程》杂志上发表，其主要特点有公开源代码，代码结构清晰明了，注释详尽，组织有条理，可移植性好，可裁剪，可固化。

- Uclinux：
  - 是由 Lineo 公司主推的开放源代码的操作系统，主要针对目标处理器没有存储管理单元的嵌入式系统而设计的。Uclinux 从 Linux2.0/2.4 内核派生而来，拥有 Linux 的绝大部分特性，通常用于内存很少的嵌入式操作系统。其主要特点有体积小、稳定、良好的移植性、优秀的网络功能等。
- eCos：
  - 含义为嵌入式可配置操作系统，主要用于消费电子、电信、车载设备、手持设备等低成本和便携式应用。其最显著的特点为可配置性，可以在源码级别实现对系统的配置和裁剪，还可安装第三方组件扩展系统功能。

3 FreeRTOS 简介

由于 RTOS 需要占用一定系统资源，只有少数 RTOS 支持在小内存的 MCU 上运行，FreeRTOS 是一款迷你型实时操作系统内核，功能包括：任务管理、时间管理、信号量、消息队列、内存管理等功能，可基本满足较小系统的需要。相对于 VxWorks、uc/os-II 等商业操作系统，FreeRTOS 完全免费，具有源码公开、可移植、可裁剪、任务调度灵活等特点，可以方便地移植到各种 MCU 上运行，其突出的特性如下。

- 免费开源。完全可以放心作为商业用途。
- 文档资源齐全。在 FreeRTOS 官网上能下载到内核文件及详细的介绍资料。
- 安全性高。SafeRTOS 基于 FreeRTOS 而来，经过安全认证的 RTOS，近年来在欧美较为流行，支持抢占式和合作式任务切换模式，代码精简，核心由 3 个 C 文件组成，可支持 65536 个任务。因此其开源免费版本 FreeRTOS 在安全性方面也应该拥有一定保障。
- 市场使用率高。从 2011 年开始，FreeRTOS 市场使用率持续高速增长，根据 EEtimes 杂志市场报告显示，FreeRTOS 使用率名列前茅，如图 3-1 所示，2017 年 FreeRTOS 市场占有率为 20%，排名第二。
- 内核文件简单。内核相关文件仅由 3 个 C 文件组成，全部围绕任务调度展开，功能专一，便于理解与学习。

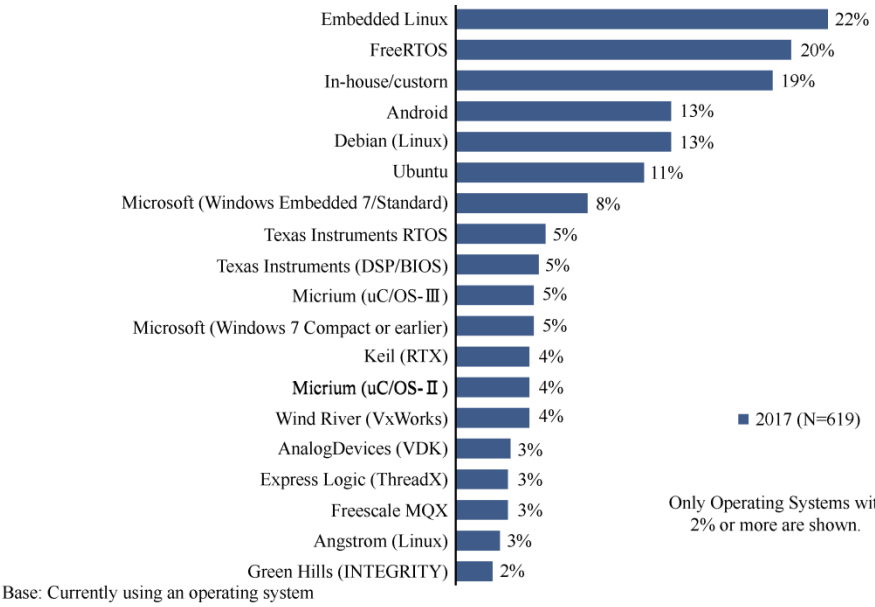

图 3-1　2017 年各种操作系统的使用数量统计

4 蜂鸟 E203 移植 FreeRTOS

本节介绍如何在 HBird-E-SDK 环境下移植一个简单的 FreeROTS 示例。用户可以以此为基础进行丰富和完善，开发更多的复杂示例。

有关 HBird-E-SDK 环境的快速上手请参见《蜂鸟 E203 快速上手介绍》，有关 HBird-E-SDK 系统性的详细介绍请参见中文书籍《RISC-V 架构与嵌入式开发入门指南》的第 11 章。

4.1 HBird-E-SDK 中 FreeRTOS 程序代码结构

在 HBird-E-SDK 环境中，FreeRTOS 示例程序的相关代码结构如下所示。

```
hbird-e-sdk             // 存放 hbird-e-sdk 的目录
          |----software           // 存放示例程序的源代码
               |----FreeRTOSv9.0.0     // FreeRTOS 示例程序目录
                    |----Source        //FreeRTOS 内核源代码
                    |----Demo          //FreeRTOS 的示例 Demo 程序代码
                    |----Makefile      //Makefile 脚本
```

Makefile 为主控制脚本，其代码片段如下：

```
    //指明生成的 elf 文件名
TARGET = FreeRTOSv9.0.0
    //指明 FreeRTOS 程序所需要的特别的 GCC 编译选项，默认选择 Code Size 优化（-Os）
CFLAGS += -Os

BSP_BASE = ../../bsp

    //指明 FreeRTOS 程序所需要的 C 源文件
C_SRCS += Source/croutine.c
C_SRCS += Source/list.c
C_SRCS += Source/queue.c
C_SRCS += Source/tasks.c
C_SRCS += Source/timers.c
C_SRCS += Source/event_groups.c
C_SRCS += Source/portable/MemMang/heap_4.c
C_SRCS += Source/portable/GCC/E203/port.c

C_SRCS += Demo/RISCV_E203_GCC/main.c
C_SRCS += $(BSP_BASE)/$(BOARD)/drivers/plic/plic_driver.c

INCLUDES += -ISource/include
INCLUDES += -IDemo/RISCV_E203_GCC
INCLUDES += -ISource/portable/GCC/E203


ASM_SRCS += Source/portable/GCC/E203/portasm.S
```

```
    //调用板级支持包（bsp）目录下的 common.mk
include $(BSP_BASE)/$(BOARD)/env/common.mk
```

4.2 FreeRTOS 原理和移植介绍

由于 RTOS 需要占用一定系统资源，只有少数 RTOS 支持在小内存的 MCU 上运行，FreeRTOS 是一款迷你型实时操作系统内核，功能包括：任务管理、时间管理、信号量、消息队列、内存管理等功能，可基本满足较小系统的需要。相对于 VxWorks、uc/os-II 等商业操作系统，FreeRTOS 完全免费，具有源码公开、可移植、可裁剪、任务调度灵活等特点、可以方便地移植到各种 MCU 上运行。

4.2.1 RTOS 操作系统的基本原理

传统裸机程序是一个大 while 循环，将所有事情看作一个任务，顺序执行代码，遇到中断发生则响应中断（可能发生中断嵌套），响应完中断后会继续之前被中断的任务，其过程如图 4-1 所示。

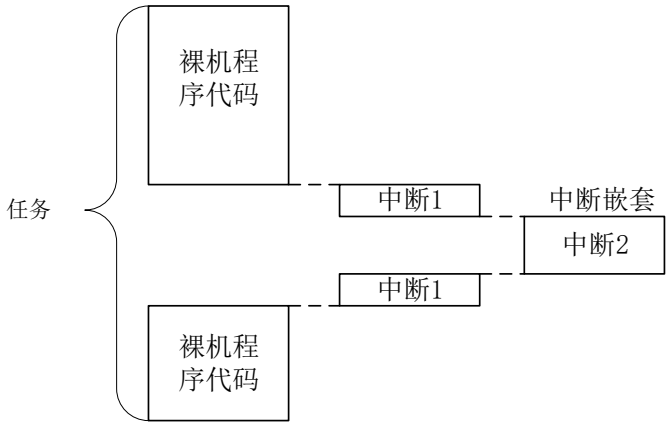

图 4-1 裸机程序的过程

而在 RTOS 中，将所有事情分成各个模块，每一个模块的内容看作一个任务，任务的执行顺序是灵活的，根据相应的调度算法管理任务的运行，灵活性比裸机程序强，其过程如图 4-2 所示。

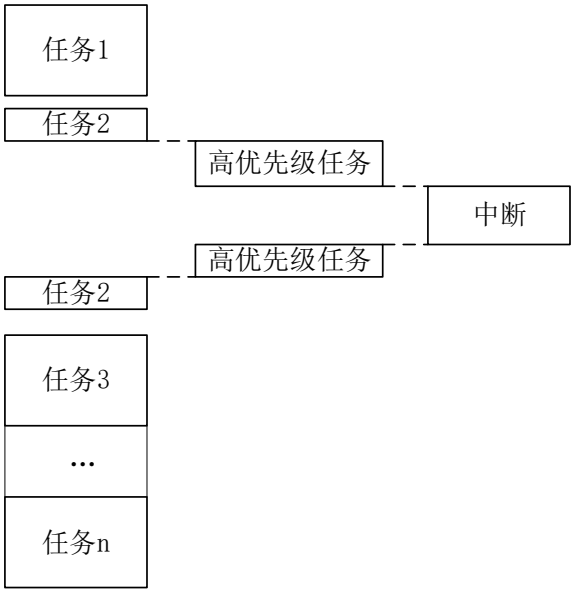

图 4-2 RTOS 程序的过程

FreeRTOS 中的调度算法分为时间片调度算法和抢占式调度，在 FreeRTOS 的 FreeRTOSConfig.h 文件中配置如下：

```
#define configUSE_PREEMPTION 1//使能抢占式调度器
#define configUSE_TIME_SLICING 1//使能时间片调度器
```

即使不配置 configUSE_TIME_SLICING 为 1，FreeRTOS 也会默认开启时间片调度。时间片调度算法和抢占式调度的特点简述如下：

- 时间片调度算法：每一个任务给予固定的执行时间，时间结束后进入调度器，由调度器切换到下一个任务，在默认所有任务优先级相同情况下，轮流执行所有任务。
- 抢占式调度需要设置任务优先级，在进入调度器后，调度器选择处于就绪态中优先级最高的任务作为下一个执行的任务。高优先级任务可以抢占低优先级任务，发生抢占时需要有能进入调度器的操作，调度器是任务切换的唯一实体。

4.2.2 FreeRTOS 源码解析和移植介绍

在 HBird-E-SDK 环境中，FreeRTOS 源代码和蜂鸟 E203 移植相关的代码结构如下所示。

```
hbird-e-sdk             // 存放 hbird-e-sdk 的目录
          |----software           // 存放示例程序的源代码
               |----FreeRTOSv9.0.0     // FreeRTOS 示例程序目录
                    |----Source        //FreeRTOS 内核源代码
                         |----list.c//与内核相关文件
                         |----queue.c//与内核相关文件
```

```
                    |----task.c//与内核相关文件
                    |----include
                         |----FreeRTOS.h//头文件
                         |----list.h//头文件
                         |----其它头文件
                    |----Portable
                         |----GCC
                              |----E203  //蜂鸟 E203 移植的相关代码，用户只需要修改此处三个
                                         // 代码即可完成对 FreeRTOS 的移植
                                   |----port.c//与移植相关的代码
                                   |----portasm.S//与移植相关的代码
                                   |----portamacro.h//与移植相关的代码
                         |----MemMang
                              |----heap_1.c//与内存分配有关的文件
                              |----heap_2.c//与内存分配有关的文件
                              |----heap_3.c//与内存分配有关的文件
                              |----heap_3.c//与内存分配有关的文件
                              |----heap_4.c//与内存分配有关的文件
                              |----heap_5.c//与内存分配有关的文件
               |----Demo          //FreeRTOS 的示例 Demo 程序代码
               |----Makefile      //Makefile 脚本
```

如上所示，FreeRTOS 的代码层次结构分明，用户只需要修改三个文件名为“port*”的源代码，完成基本的中断和异常的底层移植，即可完成对于 FreeRTOS 的移植。

蜂鸟 E203 移植 FreeRTOS 操作系统时，实现固定时间切换任务的操作由内核自带的 mtime 计时器中断支持，可以设置为每隔一个固定的时间段发生一次计时器中断（称之为 System Tick），在中断处理函数中进入调度器切换下一个任务。在 port.c 文件中 mtime 计时器设置代码如下：

```
//mtime 计时器设置函数
void vPortSetupTimer() {
//获取当前时间 mtime 的地址
volatile uint64_t * mtime= (uint64_t*) (CLINT_CTRL_ADDR + CLINT_MTIME);

//获取 mtimecmp 的地址
volatile uint64_t * mtimecmp = (uint64_t*) (CLINT_CTRL_ADDR + CLINT_MTIMECMP);

uint64_t now = *mtime;//获取当前时间
uint64_t then = now + (configRTC_CLOCK_HZ / configTICK_RATE_HZ);//计算下一次时间

//设置中断触发时间
*mtimecmp = then;

//使能计时器中断
   set_csr(mie, MIP_MTIP);
}

mtime 计时器中断处理函数如下：
void vPortSysTickHandler()
{
     static uint64_t then = 0;
     clear_csr(mie, MIP_MTIP);
     volatile uint64_t * mtime= (uint64_t*) (CLINT_CTRL_ADDR + CLINT_MTIME); //获取当前
时间的地址
     volatile uint64_t * mtimecmp= (uint64_t*) (CLINT_CTRL_ADDR + CLINT_MTIMECMP); //
```

```
定义设置比较值的地址
if(then != 0)
{
     then += (configRTC_CLOCK_HZ / configTICK_RATE_HZ);
}
Else
{
     //first time setting the timer
     uint64_t now = *mtime;
     then = now + (configRTC_CLOCK_HZ / configTICK_RATE_HZ);
}
*mtimecmp = then;//设置下一次触发中断时间


/* Increment the RTOS tick. */
if( xTaskIncrementTick() != pdFALSE )
{
     vTaskSwitchContext();//进入调度器切换任务
}
set_csr(mie, MIP_MTIP);
}
```

更多移植代码详情，请用户自行参见三个文件名为“port*”的源代码。

4.2.3 任务与中断的关系

FreeRTOS 的任务和中断的优先级关系是移植 FreeRTOS 的难点，需要被正确的理解，否则程序会运行出错：

- 任务总是可以被中断打断，任务之间具有的优先级，但是与“中断的优先级”没有关系，这两种优先级是相互独立的。
- 不调用任何 FreeRTOS API 函数的中断，可以设置为任意的“中断优先级”，并且允许嵌套。
- 在 FreeRTOSConfig.h 中预先定义 configMAX_SYSCALL_INTERRUPT_PRIORITY 的值，调用 API 函数的中断优先级只能设置为不大于该值，支持嵌套，但是会被内核延迟。

关于 FreeRTOS 的任务优先级和中断优先级如何设置，以及 FreeRTOS 的更多详细信息，请用户自行查阅相关 FreeRTOS 手册学习。

4.3 运行 FreeRTOS

FreeRTOS 示例可运行于 HBird-E-SDK 环境中，使用《蜂鸟 E203 快速上手介绍》中描述的运行方法按照如下步骤运行：

```
// 注意：确保在 HBIRD-E-SDK 中正确的安装了 RISC-V GCC 工具链，请参见《蜂鸟 E203 快速上手介绍》了解其详情。

// 步骤一：参照《蜂鸟 E203 快速上手介绍》中描述的方法，编译 FreeRTOS 示例程序，使用如下命令：
```

```
make software PROGRAM=FreeRTOSv9.0.0 NANO_PFLOAT=0
```

// 步骤二：参照《蜂鸟 E203 快速上手介绍》中描述的方法，将编译好的 FreeRTOS 程序下载至 FPGA 原型开发板中，使用如下命令：

```
make upload  PROGRAM=FreeRTOSv9.0.0
```

// 步骤三：参照《蜂鸟 E203 快速上手介绍》中描述的方法，在 FPGA 原型开发板上运行 FreeRTOS 程序：

```
// 由于示例程序将需要通过 UART 打印结果到主机 PC 的显示屏上。参考《蜂鸟 E203 快速上手介绍》
// 所述方法将串口显示电脑屏幕设置好，使得程序的打印信息能够显示在电脑屏幕上。
//
// 由于步骤二已经将程序烧写进 FPGA 开发板的 Flash 之中，因此每次按 MCU 开发板的
// RESET 按键，则处理器复位开始执行 FreeRTOS 程序，并将 RISC-V 字符串打印至主
//机 PC 的串口显示终端上。
```