



RT-Thread

中国自主物联网操作系统

RTOS

RT-Thread内核实现 与应用开发实战指南

基于STM32

刘火良 杨森 编著



机械工业出版社
China Machine Press



RT-Thread
中国自主物联网操作系统



**RT-Thread内核实现
与应用开发实战指南**
基于STM32

刘火良 杨森 编著



机械工业出版社
China Machine Press

图书在版编目 (CIP) 数据

RT-Thread 内核实现与应用开发实战指南：基于 STM32/ 刘火良, 杨森编著. —北京: 机械工业出版社, 2019.1

(电子与嵌入式系统设计丛书)

ISBN 978-7-111-61366-4

I. R… II. ①刘… ②杨… III. 微控制器 - 系统开发 - 指南 IV. TP332.3-62

中国版本图书馆 CIP 数据核字 (2018) 第 263077 号

RT-Thread 内核实现与应用开发实战指南：基于 STM32

出版发行：机械工业出版社（北京市西城区百万庄大街 22 号 邮政编码：100037）

责任编辑：赵亮宇

责任校对：殷虹

印刷：北京市兆成印刷有限责任公司

版次：2019 年 1 月第 1 版第 1 次印刷

开本：186mm × 240mm 1/16

印张：26

书号：ISBN 978-7-111-61366-4

定价：99.00 元

凡购本书，如有缺页、倒页、脱页，由本社发行部调换

客服热线：(010) 88379426 88361066

投稿热线：(010) 88379604

购书热线：(010) 68326294 88379649 68995259

读者信箱：hzit@hzbook.com

版权所有·侵权必究

封底无防伪标均为盗版

本书法律顾问：北京大成律师事务所 韩光 / 邹晓东

推 荐 序

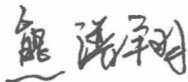
从 2006 年走来，RT-Thread 已经走过了十多个年头。早年就和野火相识于网络，记得那个时候野火电子才开始做开发板，RT-Thread 也只是国内一个小众的 RTOS。直至 2018 年年中，野火来上海我俩才有缘相见，互道年轻。

2017 年年底，欣闻野火有写作 RT-Thread 相关书籍的计划，非常欣喜、感谢。这将是介绍 RT-Thread 的首本书籍，在 RT-Thread 的基础生态推广方面有着举足轻重的作用！不仅如此，作为 RT-Thread 的官方合作伙伴，野火电子也在其全系列的 STM32 开发板上适配了 RT-Thread 例程并配套了手把手的教程。

野火的这本书，第一部分由 0 到 1，从底层的汇编开始，一步一步构造出一个 RT-Thread 操作系统内核，向大家揭示了任务如何定义、如何切换，也讲解了任务的延时如何实现、如何支持多优先级、如何实现定时器以及如何实现时间片等 RT-Thread 操作系统的核心知识点；第二部分则讲解了 RT-Thread 内核设施的应用，使得大家学习和使用 RT-Thread 不再困难。

整本书由浅入深，层层叠加，与初学者的入门路径完全吻合，是学习 RT-Thread 物联网操作系统的不二之选。同时，整本书也兼顾深度，对想要了解操作系统内核原理的读者来说，也非常值得一读。

RT-Thread 创始人





前 言

如何学习本书

本书是首本系统讲解 RT-Thread 的中文书籍，共分为两个部分。第一部分重点讲解 RT-Thread 的原理实现，从 0 开始，不断迭代，教你把 RT-Thread 的内核写出来，让你彻底学会线程是如何定义的、系统是如何调度的（包括底层的汇编代码讲解）、多优先级是如何实现的等操作系统的最深层次的知识。当你拿到本书开始学习的时候，你一定会惊讶，原来 RTOS 的学习并没有那么复杂，反而是那么有趣；原来自己也可以写 RTOS，成就感立马爆棚。

当彻底掌握第一部分的知识之后，再学习其他 RTOS，可以说十分轻松。纵观现在市面上流行的几种 RTOS，它们的内核实现差异不大，只需要深入研究其中一种即可，没有必要对每一种 RTOS 都深入地研究源码，但如果时间允许，看一看也并无坏处。第二部分重点讲解 RT-Thread 的移植、内核中每个组件的应用，比起第一部分，这部分内容掌握起来应该比较容易。

全书内容循序渐进，不断迭代，尤其在第一部分，前一章是后一章的基础，必须从头开始阅读，不能进行跳跃式的阅读。在学习时务必做到两点：一是不能一味地看书，要把代码和书本结合起来学习，一边看书，一边调试代码。如何调试代码呢？即单步执行每一条程序，看程序的执行流程和执行的效果与自己所想的是否一致。二是在每学完一章之后，必须将配套的例程重写一遍（切记不要复制，即使是一个分号，但可以照书录入），做到举一反三，确保真正理解。在自己写的时候难免错误百出，要珍惜这些错误，好好调试，这是你提高编程能力的最好机会。记住，程序不是一气呵成写出来的，而是一步一步调试出来的。

本书的编写风格

本书第一部分主要以 RT-Thread Nano 3.0.3 官方源码为蓝本，抽丝剥茧，不断迭代，教你如何从 0 开始把 RT-Thread 内核写出来。书中涉及的数据类型、变量名称、函数名称、文件名称、文件存放的位置都完全按照 RT-Thread 官方的方式来实现。学完这本书之后，你可

以无缝地切换到原版的 RT-Thread 中使用。要注意的是，在实现的过程中，某些函数中会去掉一些形参和冗余的代码，只保留核心的功能，但这并不会影响我们学习。

本书第二部分主要介绍 RT-Thread 的移植和内核组件的使用，不会再去深入讲解源码，而是着重讲解如何应用，如果对第一部分不感兴趣，也可以跳过第一部分，直接进入第二部分的学习。

本书还有姊妹篇——《FreeRTOS 内核实现与应用开发实战指南：基于 STM32》，两本书的编写风格、内容框架和章节命名与排序基本一致，语言阐述类似，且涉及 RTOS 抽象层的理论部分也相同，不同之处在于 RTOS 的实现原理、内核源码的讲解和上层 API 的使用，这些内容才是重点部分，是读者学习的核心。例如，虽然两本书的第一部分的章节名称基本类似，但内容不同，因为针对的 RTOS 不一样。其中，关于新建 RT-Thread 工程和裸机系统与多线程（任务）系统的描述属于 RTOS 抽象层的理论部分，不具体针对某个 RTOS，所以基本一样。第二部分中，对于什么是线程（任务）、阻塞延时和信号量的应用等 RTOS 抽象层的理论讲解也基本类似，但是具体涉及这两个 RTOS 的原理实现和代码讲解时则完全不同。

如果读者已经学习了其中一本书，再学习另外一本的话，那么涉及 RTOS 抽象层的理论部分可跳过，只需要把精力放在 RTOS 内核的实现和源码 API 的应用方面。因为现有的 RTOS 在理论层基本都是相通的，但在具体的代码实现上各有特点，所以可以用这两本书进行互补学习，掌握了其中一本书的知识，再学习另外一本书定会得心应手，事半功倍。

本书的参考资料和配套硬件

关于本书的参考资料和配套硬件的信息，请参考本书附录部分。

本书的技术论坛

如果在学习过程中遇到问题，可以到野火电子论坛 www.firebbs.cn 发帖交流，开源共享，共同进步。

鉴于水平有限，书中难免有错漏之处，热心的读者也可把勘误发送到论坛上以便改进。祝你学习愉快，RT-Thread 的世界，野火与你同行。

引 言

为什么学习 RTOS

当我们进入嵌入式系统这个领域时，首先接触的往往是单片机编程，单片机编程又首选 51 单片机来入门。这里说的单片机编程通常都是指裸机编程，即不加入任何 RTOS (Real Time Operation System, 实时操作系统) 的程序。常用的 RTOS 有国外的 FreeRTOS、 $\mu\text{C}/\text{OS}$ 、RTX 和国内的 RT-Thread、Huawei LiteOS、AliOS-Things 等，其中，开源且免费的 FreeRTOS 的市场占有率最高。如今国产的 RT-Thread 经过 10 余年的迅猛发展，在国产 RTOS 中占据鳌头。

在裸机系统中，所有的程序基本都是用户自己写的，所有的操作都是在一个无限的大循环里面实现。现实生活的很多中小型电子产品中用的都是裸机系统，而且能够满足需求。但是为什么还要学习 RTOS 编程，并会涉及一个操作系统呢？一是因为项目需求，随着产品要实现的功能越来越多，单纯的裸机系统已经不能完美地解决问题，反而会使编程变得更加复杂，如果想降低编程的难度，可以考虑引入 RTOS 实现多线程管理，这是使用 RTOS 的最大优势；二是出于学习的需要，必须学习更高级的技术，实现更好的职业规划，为将来能有更好的职业发展做准备，而不是一味拘泥于裸机编程。作为一个合格的嵌入式软件工程师，学习是永远不能停歇的，时刻都得为将来做准备。书到用时方恨少，希望当机会来临时，读者不要有这种感觉。

为了帮大家厘清 RTOS 编程的思路，本书会在第 2 章简单地分析这两种编程方式的区别，这个区别被笔者称为“学习 RTOS 的‘命门’”，只要掌握这一关键内容，以后的 RTOS 学习可以说是易如反掌。在讲解这两种编程方式的区别时，我们主要讲解方法，不会涉及具体的代码编程，即主要还是通过伪代码来讲解。

如何学习 RTOS

裸机编程和 RTOS 编程的风格有些不一样，而且有很多人说学习 RTOS 很难，这就导致

想要学习 RTOS 的人一听到 RTOS 编程就在心里忌惮三分，结果就是“出师未捷身先死”。

那么到底如何学习 RTOS 呢？最简单的方法就是在别人移植好的系统上，先看看 RTOS 中 API 的使用说明，然后调用这些 API 实现自己想要的功能即可，完全不用关心底层的移植，这是最简单、快速的入门方法。这种方法有利有弊，如果是做产品，好处是可以快速实现功能，将产品推向市场，赢得先机；弊端是当程序出现问题时，因对 RTOS 不够了解，会导致调试困难。如果想系统地学习 RTOS，那么只会简单地调用 API 是不可取的，我们应该深入学习其中一款 RTOS。

目前市场上现有的 RTOS，其内核实现方式差异不大，我们只需要深入学习其中一款即可。万变不离其宗，即使以后换到其他型号的 RTOS，使用起来自然也是得心应手。那么，如何深入地学习一款 RTOS 呢？这里有一个非常有效但也十分难的方法，就是阅读 RTOS 的源码，深入研究内核和每个组件的实现方式。这个过程枯燥且痛苦，但为了能够学到 RTOS 的精华，还是很值得一试的。

市面上虽然有一些讲解 RTOS 源码的图书，但如果基础知识掌握得不够，且先前没有使用过该款 RTOS，那么只看源码还是会非常枯燥，并且不能从全局掌握整个 RTOS 的构成和实现。

现在，我们采用一种全新的方法来教大家学习一款 RTOS，既不是单纯地介绍其中的 API 如何使用，也不是单纯拿里面的源码一句句地讲解，而是从 0 开始，层层叠加，不断完善，教大家如何把一个 RTOS 从 0 到 1 写出来，让你在每一个阶段都能享受到成功的喜悦。在这个 RTOS 实现的过程中，只需要具备 C 语言基础即可，然后就是跟着本书笃定前行，最后定有所成。

选择什么 RTOS

用来教学的 RTOS，我们不会完全从头写一个，而是选取目前国内很流行的 RT-Thread 为蓝本，将其抽丝剥茧，从 0 到 1 写出来。在实现的过程中，数据类型、变量名、函数名称、文件类型等都完全按照 RT-Thread 里面的写法，不会再重新命名。这样学完本书之后，就可以无缝地过渡到 RT-Thread 了。

RT-Thread 简介

RT-Thread 的版权属于上海睿赛德电子科技有限公司，它于 2006 年 1 月首次发布，初始版本号为 0.1.0，经过 10 余年的发展，如今主版本号已经升级到 3.0，累计开发者人数达数

百万，在各行各业产品中装机量达到了 2000 多万，占据国产 RTOS 的鳌头。

RT-Thread 是一款开源免费的实时操作系统，遵循的是 GPLv2+ 许可协议。这里所说的开源，指的是你可以免费获取 RT-Thread 的源代码，而且当你的产品使用了 RT-Thread 且没有修改 RT-Thread 内核源码时，你的产品的全部代码都可以闭源，但是当你修改了 RT-Thread 内核源码时，就必须将修改的这部分开源，反馈给社区，其他应用部分不用开源。无论是个人还是公司，都可以免费使用 RT-Thread。

RT-Thread 的意义

不知你是否发现，在 RTOS 领域，我们能接触到的实时操作系统基本都来自国外，很少见到有国内厂家的产品。从早年很流行的 $\mu\text{C}/\text{OS}$ ，到如今市场占有率很高的 FreeRTOS，到获得安全验证非常多的 RTX (KEIL 自家的)，再到盈利能力领先的 ThreadX，均来自国外。可在十几年前，在中国，出现了一个天赋异禀、倔强不屈的极客——熊谱翔。他编写了 RT-Thread 初代内核，并联合中国开源社区的极客不断完善，推陈出新，经过十几年的发展，如今占据国产 RTOS 的鳌头，且每年能递增数十万名开发者，加上如今 AI 和物联网等技术发展的机遇，让 RT-Thread 有“一统江湖”之势，从 2018 年完成 A 轮数百万美元的融资就可以看出，在未来不出 5 年，RT-Thread 将是你学习和做产品的不二之选。

那么 RT-Thread 的意义究竟是什么？RT-Thread 来自中国，让我们看到了国内的技术开发者也能写出如此优秀的 RTOS，技术方面并不逊色于其他国家。以我们 10 多年从事电子行业的经验来看，RT-Thread 无疑增强了我们在这一领域的自信，这是我们认为的 RT-Thread 的最大意义。当然，作为一款国产的物联网操作系统，RT-Thread 简单易用、低功耗设计、组件丰富等特性也将让其大放异彩。野火，作为一个国内嵌入式领域的教育品牌，能为国产 RTOS 出一份力也是我们的荣幸，希望本书能够帮助大家快速地入门并掌握 RT-Thread。

目 录

推荐序

前言

引言

第一部分 从 0 到 1 教你写 RT-Thread 内核

第 1 章 新建 RT-Thread 工程——软件仿真 2

- 1.1 新建本地工程文件夹 2
- 1.2 使用 KEIL 新建工程 3
 - 1.2.1 New Project 3
 - 1.2.2 Select Device for Target 3
 - 1.2.3 Manage Run-Time Environment 4
- 1.3 在 KEIL 工程中新建文件组 5
- 1.4 在 KEIL 工程中添加文件 6
- 1.5 调试配置 7
 - 1.5.1 设置软件仿真 7
 - 1.5.2 修改时钟大小 8
 - 1.5.3 添加头文件路径 9

第 2 章 裸机系统与多线程系统 10

- 2.1 裸机系统 10
 - 2.1.1 轮询系统 10
 - 2.1.2 前后台系统 11

2.2 多线程系统 12

第 3 章 线程的定义与线程切换的实现 15

- 3.1 什么是线程 16
- 3.2 创建线程 17
 - 3.2.1 定义线程栈 17
 - 3.2.2 定义线程函数 19
 - 3.2.3 定义线程控制块 20
 - 3.2.4 实现线程创建函数 20
- 3.3 实现就绪列表 30
 - 3.3.1 定义就绪列表 30
 - 3.3.2 将线程插入就绪列表 30
- 3.4 实现调度器 31
 - 3.4.1 调度器初始化 32
 - 3.4.2 启动调度器 33
 - 3.4.3 第一次线程切换 34
 - 3.4.4 系统调度 41
- 3.5 main() 函数 44
- 3.6 实验现象 47

第 4 章 临界段的保护 50

- 4.1 什么是临界段 50
- 4.2 Cortex-M 内核快速关中断指令 50
- 4.3 关中断 51
- 4.4 开中断 51
- 4.5 临界段代码的应用 52

4.6 实验现象	56	7.2.3 修改线程初始化函数 <code>rt_</code> <code>thread_init()</code>	91
第 5 章 对象容器	57	7.2.4 添加线程启动函数 <code>rt_thread_</code> <code>startup()</code>	92
5.1 什么是对象	57	7.2.5 修改空闲线程初始化函数 <code>rt_thread_idle_init()</code>	93
5.1.1 对象枚举的定义	57	7.2.6 修改启动系统调度器函数 <code>rt_system_scheduler_start()</code>	94
5.1.2 对象数据类型的定义	57	7.2.7 修改系统调度函数 <code>rt_schedule()</code>	95
5.1.3 在线程控制块中添加对象 成员	58	7.2.8 修改阻塞延时函数 <code>rt_thread_</code> <code>delay()</code>	98
5.2 什么是容器	59	7.2.9 修改时基更新函数 <code>rt_tick_</code> <code>increase()</code>	98
5.3 容器的接口实现	63	7.3 <code>main()</code> 函数	99
5.3.1 获取指定类型的对象信息	64	7.4 实验现象	102
5.3.2 对象初始化	64	第 8 章 定时器	103
5.3.3 调用对象初始化函数	67	8.1 实现定时器	103
5.4 实验现象	67	8.1.1 系统定时器列表	103
第 6 章 空闲线程与阻塞延时	68	8.1.2 系统定时器列表初始化	104
6.1 实现空闲线程	68	8.1.3 定义定时器结构体	104
6.1.1 定义空闲线程的栈	68	8.1.4 在线程控制块中内置 定时器	105
6.1.2 定义空闲线程的线程 控制块	69	8.1.5 定时器初始化函数	106
6.1.3 定义空闲线程函数	69	8.1.6 定时器删除函数	108
6.1.4 空闲线程初始化	69	8.1.7 定时器停止函数	108
6.2 实现阻塞延时	70	8.1.8 定时器控制函数	109
6.3 <code>SysTick_Handler()</code> 中断服务 函数	73	8.1.9 定时器启动函数	110
6.4 <code>main()</code> 函数	77	8.1.10 定时器扫描函数	115
6.5 实验现象	80	8.2 修改代码以支持定时器	118
第 7 章 多优先级	82	8.2.1 修改线程初始化函数	118
7.1 就绪列表	82	8.2.2 修改线程延时函数	119
7.1.1 线程就绪优先级组	82	8.2.3 修改系统时基更新函数	121
7.1.2 线程优先级表	86	8.2.4 修改 <code>main.c</code> 文件	122
7.2 修改代码以支持多优先级	88		
7.2.1 修改线程控制块	88		
7.2.2 修改调度器初始化函数 <code>rt_</code> <code>system_scheduler_init()</code>	90		

8.3 实验现象	126	10.6 board.c 文件	156
第 9 章 时间片	127	10.6.1 board.c 文件内容讲解	156
9.1 实现时间片	127	10.6.2 board.c 文件修改	160
9.1.1 在线程控制块中添加 时间片相关成员	127	10.7 添加 core_delay.c 和 core_delay.h 文件	167
9.1.2 修改线程初始化函数	128	10.8 修改 main.c	171
9.1.3 修改空闲线程初始化函数	129	10.9 下载验证	172
9.1.4 修改系统时基更新函数	129	第 11 章 线程	173
9.2 修改 main.c 文件	131	11.1 硬件初始化	173
9.3 实验现象	135	11.2 创建单线程——SRAM 静态 内存	175
第二部分 RT-Thread 内核 应用开发		11.2.1 定义线程函数	175
第 10 章 移植 RT-Thread 到 STM32	138	11.2.2 定义线程栈	176
10.1 获取 STM32 的裸机工程模板	138	11.2.3 定义线程控制块	176
10.2 下载 RT-Thread Nano 源码	138	11.2.4 初始化线程	176
10.3 安装 RT-Thread Package	139	11.2.5 启动线程	177
10.4 向裸机工程中添加 RT-Thread 源码	140	11.2.6 main.c 文件内容	177
10.4.1 复制 RT-Thread Package 到裸机工程根目录	140	11.3 下载验证 SRAM 静态内存 单线程	179
10.4.2 复制 rtconfig.h 文件到 User 文件夹	141	11.4 创建单线程——SRAM 动态 内存	179
10.4.3 复制 board.c 文件到 User 文件夹	141	11.4.1 动态内存空间堆的来源	180
10.4.4 rt-thread 文件夹内容 简介	141	11.4.2 定义线程函数	181
10.4.5 添加 RT-Thread 源码到 工程组文件夹	143	11.4.3 定义线程栈	181
10.5 rtconfig.h 文件	145	11.4.4 定义线程控制块指针	181
10.5.1 rtconfig.h 文件内容讲解	145	11.4.5 创建线程	181
10.5.2 rtconfig.h 文件修改	152	11.4.6 启动线程	182
		11.4.7 main.c 文件内容	182
		11.5 下载验证 SRAM 动态内存 单线程	184
		11.6 创建多线程——SRAM 动态 内存	185
		11.7 下载验证 SRAM 动态内存 多线程	187

第 12 章 重映射串口到 rt_kprintf() 函数	188	14.8 实验现象	215
12.1 rt_kprintf() 函数定义	188	第 15 章 消息队列	216
12.2 自定义 rt_hw_console_output() 函数	189	15.1 消息队列的基本概念	216
12.3 测试 rt_kprintf() 函数	191	15.2 消息队列的运作机制	217
12.3.1 硬件初始化	191	15.3 消息队列的阻塞机制	218
12.3.2 编写 rt_kprintf() 测试代码	192	15.4 消息队列的应用场景	218
12.3.3 下载验证	192	15.5 消息队列控制块	218
第 13 章 RT-Thread 的启动流程	194	15.6 消息队列函数	219
13.1 “万事俱备，只欠东风”法	194	15.6.1 消息队列创建函数 rt_mq_create()	219
13.2 “小心翼翼，十分谨慎”法	195	15.6.2 消息队列删除函数 rt_mq_delete()	221
13.3 两种方法的适用情况	197	15.6.3 消息队列发送消息函数 rt_mq_send()	223
13.4 RT-Thread 的启动流程	197	15.6.4 消息队列接收消息函数 rt_mq_recv()	226
13.4.1 \$\$Sub\$\$main() 函数	198	15.7 消息队列使用注意事项	229
13.4.2 rtthread_startup() 函数	199	15.8 消息队列实验	230
13.4.3 rt_application_init() 函数	201	15.9 实验现象	233
13.4.4 \$\$Super\$\$main() 函数	202	第 16 章 信号量	234
13.4.5 main() 函数	203	16.1 信号量的基本概念	234
第 14 章 线程管理	205	16.2 二值信号量的应用场景	235
14.1 线程的基本概念	205	16.3 二值信号量的运作机制	236
14.2 线程调度器的基本概念	205	16.4 计数型信号量的运作机制	236
14.3 线程状态的概念	206	16.5 信号量控制块	237
14.4 线程状态迁移	206	16.6 信号量函数	237
14.5 常用的线程函数	207	16.6.1 信号量创建函数 rt_sem_create()	238
14.5.1 线程挂起函数 rt_thread_suspend()	207	16.6.2 信号量删除函数 rt_sem_delete()	239
14.5.2 线程恢复函数 rt_thread_resume()	209	16.6.3 信号量释放函数 rt_sem_release()	240
14.6 线程的设计要点	210	16.6.4 信号量获取函数 rt_sem_take()	242
14.7 线程管理实验	212		

16.7	信号量实验	245	18.5.2	事件删除函数 <code>rt_event_</code> <code>delete()</code>	276
16.7.1	二值信号量同步实验	245	18.5.3	事件发送函数 <code>rt_event_</code> <code>send()</code>	277
16.7.2	计数型信号量实验	248	18.5.4	事件接收函数 <code>rt_event_</code> <code>recv()</code>	281
16.8	实验现象	251	18.6	事件实验	285
16.8.1	二值信号量同步实验 现象	251	18.7	实验现象	288
16.8.2	计数型信号量实验现象	252			
第 17 章 互斥量 253					
17.1	互斥量的基本概念	253	第 19 章 软件定时器 289		
17.2	互斥量的优先级继承机制	253	19.1	软件定时器的基本概念	289
17.3	互斥量的应用场景	256	19.2	软件定时器的应用场景	290
17.4	互斥量的运作机制	256	19.3	软件定时器的精度	291
17.5	互斥量控制块	257	19.4	软件定时器的运作机制	291
17.6	互斥量函数	258	19.5	定时器超时函数	293
17.6.1	互斥量创建函数 <code>rt_mutex_</code> <code>create()</code>	258	19.6	软件定时器的使用	297
17.6.2	互斥量删除函数 <code>rt_mutex_</code> <code>delete()</code>	260	19.7	软件定时器实验	299
17.6.3	互斥量释放函数 <code>rt_mutex_</code> <code>release()</code>	261	19.8	实验现象	302
17.6.4	互斥量获取函数 <code>rt_mutex_</code> <code>take()</code>	264	第 20 章 邮箱 303		
17.7	互斥量使用注意事项	268	20.1	邮箱的基本概念	303
17.8	互斥量实验	268	20.2	邮箱的运作机制	304
17.9	实验现象	271	20.3	邮箱的应用场景	305
第 18 章 事件 272					
18.1	事件的基本概念	272	20.4	邮箱的应用技巧	305
18.2	事件的应用场景	273	20.5	邮箱控制块	306
18.3	事件的运作机制	273	20.6	邮箱函数	306
18.4	事件控制块	275	20.6.1	邮箱创建函数 <code>rt_mb_</code> <code>create()</code>	306
18.5	事件函数	275	20.6.2	邮箱删除函数 <code>rt_mb_</code> <code>delete()</code>	308
18.5.1	事件创建函数 <code>rt_event_</code> <code>create()</code>	275	20.6.3	邮箱邮件发送函数 <code>rt_mb_</code> <code>send_wait()</code> (阻塞)	310
			20.6.4	邮箱邮件发送函数 <code>rt_mb_</code> <code>send()</code> (非阻塞)	315
			20.6.5	邮箱邮件接收函数 <code>rt_mb_</code> <code>recv()</code>	316

20.7	邮箱实验	320	21.7.1	静态内存管理实验现象	364
20.8	实验现象	324	21.7.2	动态内存管理实验现象	364
第 21 章 内存管理			第 22 章 中断管理		
21.1	内存管理的基本概念	325	22.1	异常与中断的基本概念	366
21.2	内存管理的运作机制	327	22.1.1	中断	367
21.2.1	静态内存管理	327	22.1.2	和中断相关的术语	367
21.2.2	动态内存管理	328	22.2	中断管理的运作机制	368
21.3	内存管理的应用场景	330	22.3	中断延迟的概念	369
21.4	静态内存管理函数	331	22.4	中断管理的应用场景	370
21.4.1	静态内存控制块	331	22.5	ARM Cortex-M 的中断管理	370
21.4.2	静态内存创建函数 <code>rt_mp_create()</code>	332	22.6	中断管理实验	372
21.4.3	静态内存删除函数 <code>rt_mp_delete()</code>	335	22.7	实验现象	377
21.4.4	静态内存初始化函数 <code>rt_mp_init()</code>	338	第 23 章 双向链表		
21.4.5	静态内存申请函数 <code>rt_mp_alloc()</code>	340	23.1	双向链表的基本概念	378
21.4.6	静态内存释放函数 <code>rt_mp_free()</code>	344	23.2	双向链表函数	378
21.5	动态内存管理函数	346	23.2.1	链表初始化函数 <code>rt_list_init()</code>	378
21.5.1	系统堆内存初始化函数 <code>rt_system_heap_init()</code>	346	23.2.2	向链表中插入节点	379
21.5.2	系统堆内存申请函数 <code>rt_malloc()</code>	350	23.2.3	从链表删除节点函数 <code>rt_list_remove()</code>	381
21.5.3	系统堆内存释放函数 <code>rt_free()</code>	354	23.3	双向链表实验	382
21.6	内存管理实验	357	23.4	实验现象	385
21.6.1	静态内存管理实验	357	第 24 章 CPU 利用率统计		
21.6.2	动态内存管理实验	361	24.1	CPU 利用率的基本概念	387
21.7	实验现象	364	24.2	CPU 利用率的作用	387
			24.3	CPU 利用率统计实现	388
			24.4	CPU 利用率实验	392
			24.5	实验现象	394
			附录 参考资料和配套硬件		
					396

第一部分

从 0 到 1 教你写 RT-Thread 内核

本部分以 RT-Thread Nano 为蓝本，抽丝剥茧，不断迭代，教大家如何从 0 开始写 RT-thread。这一部分的重点在于 RTOS 的实现过程，当学完这部分内容之后，再来重新使用 RT-Thread 或者其他 RTOS 将会得心应手，不仅知其然，而且知其所以然。在源码实现的过程中，涉及的数据类型、变量名称、函数名称、文件名称以及文件的存放目录都会完全按照 RT-Thread 的来实现，一些不必要的代码会被剔除，但这并不会影响我们理解整个操作系统的功能。

本部分中每一章都以前一章为基础，环环相扣，逐渐揭开操作系统的神秘面纱，读起来会有一种令人豁然开朗的感觉。如果你把代码都自己输入一遍，得到的仿真效果如果与书中的一样，那从心里油然而生的成就感简直就要爆棚，恨不得一下子把这本书读完，真是让人看了还想看，读了还想读。

第 1 章

新建 RT-Thread 工程——软件仿真

在开始编写 RT-Thread 内核之前，我们先新建一个 RT-Thread 工程，Device 选择 Cortex-M3（Cortex-M4 或 Cortex-M7）内核的处理器，调试方式选择软件仿真，然后我们开始一步一步地教大家把 RT-Thread 内核写出来，让大家彻底明白 RT-Thread 的内部实现和设计思想，最后把 RT-Thread 移植到野火 STM32 开发板上。最后的移植其实已经非常简单，只需要换一下启动文件并添加 bsp 驱动即可。

1.1 新建本地工程文件夹

在开始新建工程之前，我们先在本地计算机中新建一个文件夹用于存放工程。可将文件夹命名为“新建 RT-Thread 工程——软件仿真”（名字可以随意设置），然后在该文件夹下新建各个文件夹和文件，有关这些文件夹的包含关系和作用如表 1-1 所示。

表 1-1 工程文件夹根目录下的文件夹及作用

文件夹名称	文件夹	文件夹作用
Doc	—	用于存放整个工程的说明文件，如 readme.txt。通常情况下，要对整个工程实现的功能、如何编译、如何使用等做一个简要的说明
Project	—	用于存放新建的工程文件
rtthread/3.0.3	bsp	存放板级支持包，暂时为空
	components\finsh	存放 RT-Thread 组件，暂时为空
	include	存放头文件，暂时为空
	include\libc	
	libcpu\arm\cortex-m0	存放与处理器相关的接口文件，暂时为空
	libcpu\arm\cortex-m3	
	libcpu\arm\cortex-m4	
libcpu\arm\cortex-m7		
src	存放 RT-Thread 内核源码，暂时为空	
User		存放 main.c 和其他用户编写的程序，第一次使用 main.c 时需要用户自行创建