前言

0.1 这套书讲的是什么？

这套书旨在为开发和生产实时嵌入式系统提供坚实的知识和技能基础。其内容主要分为两类：

- 奠定基础原理的部分：讲解实时嵌入式系统所依赖的核心概念与基本原理。
- 面向实践技能的部分：展示如何使用、应用具体的设计方法与开发技术。

在电子、机械、航空航天等传统工程学科中，工程师们都清楚地理解“基础知识”与“应用技能”之间的区别。此外，有经验的工程师深知：只有真正掌握基础，才能有效地应用技能解决实际问题。遗憾的是，在软件工程领域，这种对基础的重视却常常被忽视。本书希望能弥补这一缺失，为读者提供扎实的理论基础与可实践的技术能力。

0.2 谁应该阅读这套书？

本书旨在满足那些在实时嵌入式系统软件开发领域工作，或计划从事相关工作的人员的需求。本书针对四类读者而撰写：

- 学生。
- 转入软件系统领域的工程师、科学家及数学家。
- 进入嵌入式领域的专业且有经验的软件工程师。
- 对基于软件的实时系统的基本原理几乎没有正式教育或经验的程序员。

0.3 这本书是关于什么的？

本书主要介绍该领域的基础知识。简单来说，它试图解答以下几个主要问题：

1. 什么是实时操作系统（RTOS）？
2. 为什么在设计中应该使用 RTOS？
3. 使用 RTOS 会有不利之处吗？
4. 嵌入式实时操作系统的组成模块有哪些？
5. 现代嵌入式系统可能采用单处理器、多处理器或多台计算机。我们应如何在这些不同平台上使用 RTOS？
6. 如何评估 RTOS 的性能，并在必要时进行改进？
7. 如何对基于 RTOS 的设计进行调试？

目录部分会更详细地展示本书内容安排；此外，每章开头都会明确说明本章的目标。我建议你先浏览这些内容，以对本书整体范围和目的有一个清晰的了解。原书已重新命名为《实时操作系统第 1 册——理论篇》。之所以这样，是因为本书有一部配套书籍《实时操作系统第 2 册——实践篇》。该配套书包含了一系列练习（当然，你可以选择是否完成），以帮助你更好地理解相关主题。这些练习主要涉及第 1 至第 5 章中讲解的核心内容。我建议你在学习理论材料的同时进行实践练习，这样可以让你更好地应对实际基于 RTOS 的设计问题。

0.4 应该如何阅读这本书？

请大家——无论有经验与否——务必阅读第 1 章。不仅要读，还要真正理解其中的内容。因为如果你没有真正理解这里讨论的问题，你将很难设计出好方案。

第 2 章到第 6 章讲述了该领域的基础知识。它们不仅展示了多任务设计是如何实现的，还解释了为什么要以某种方式去实现。这里的目标读者是那些刚接触实时嵌入式系统任务设计与实现的人。但需要强调的是，本节主要将工作放在单处理器单元的背景下。第 7 章和第 8 章则将视角拓宽到多处理器和分布式系统（需要说明的是，这两者之间的界限有时并不十分明确）。

第 9 章本质上偏理论，但带有实用倾向，从更广的角度讨论任务调度技术。之所以把这一章放在后面，是为了让读者更容易吸收内容。只要读者熟悉本领域的基本概念，这一章应该相对容易理解。

第 10 章到第 12 章侧重于该领域的实际操作。如果你刚接触 RTOS 领域，第 10 章会帮助你扎实理解不同操作系统结构之间的差异，这在你选择第一个 RTOS 时尤其有用。相比之下，第 11 章和第 12 章的内容在你搭建系统之后会特别有帮助。本质上，它们关注的是软件在运行时的行为、质量和可靠性。

0.5 致谢

一大堆地址和人名，不想翻译，略了:)

第一章
你应该知道的有关实时操作系统的内容

本章的目标是：

- 向你展示使用实时操作系统（RTOS）是有充分理由的。
- 向你说明使用 RTOS 也存在一些缺点（简单来说，就是“得一分，失一分”）。
- 描述 RTOS 支持如何简化软件的功能设计，从而更容易构建高质量系统。
- 强调在基于 RTOS 的设计中，时间和时序的重要性。
- 说明基于任务的设计的基本目标、结构和运行方式。

0.6 设定背景

在大型计算机领域，操作系统（OS）已经伴随我们相当长的时间了。实际上，最初的操作系统可以追溯到 20 世纪 50 年代。到了 60 年代，操作系统取得了重大进展；到 70 年代中期，其概念、结构、功能和接口已经基本确立。

微型计算机大约在 1970 年出现。从逻辑上来看，操作系统似乎应当能够迅速应用于基于微处理器的系统中。然而，到 80 年代中期，真正采用正式设计的实时操作系统（RTOS）的微处理器实现仍然很少。确实，CP/M 于 1975 年发布，并后来被 Intel 制作成硅片产品，但它对实时领域影响不大，其自然应用领域是桌面计算机。

RTOS 的应用受两个因素影响：一方面是机器本身的限制，另一方面是围绕微型计算机的设计文化。早期的微型计算机在计算能力、运行速度和内存容量上都非常有限。想要在这种基础上引入操作系统结构是非常困难的。此外，当时大多数从事嵌入式系统编程的人，对操作系统几乎没有背景知识。

如今情况则大不相同。现代嵌入式设计的核心工作马是 16/32 位复杂微控制器。这些器件低成本、高性能，并且集成了大量片上存储和外设功能。此外，市场上有非常多的商业 RTOS 可供选择。然而，仅仅因为你能够做某件事，并不意味着你就应该去做。那么，为什么在下一个设计中你应该选择使用 RTOS 呢？

在回答这个问题之前，我们需要先考虑一个更基础的问题：在嵌入式系统中，我们到底应该如何开展软件设计？这正是本章的关键点。本章为实用的设计技术奠定基础，并展示了 RTOS 在其中的具体作用。

0.7 生产高质量的软件

乍一看，在本章开头就谈论软件质量似乎有些奇怪。这似乎与操作系统关系不大。但事实并非如此，我们可以在这里学到一些宝贵的经验。

如果让你定义“高质量”软件的含义，你会怎么说？不妨看看以下几点：

- 它应该正确地完成其工作（“功能”正确性）。
- 它应该在正确的时间内完成其工作（“时间”正确性）。
- 其行为应该是可预测的。
- 其行为应该是一致的。
- 代码不应难以维护（低复杂度）。
- 代码的正确性可以被分析（静态分析）。
- 代码的行为可以被分析（覆盖率分析）。
- 运行时性能应该是可预测的。
- 内存需求应该是可预测的。
- 如果需要，代码可以被证明符合相关标准。

当然，你可以根据自己的特定需求扩展这个列表。

现在，考虑将这些原则应用于下面这个小型、相对简单的实时系统（图 1.1）。

这里的要求是通过改变液体的流速来控制其温度，它是通过以下步骤完成的：

- 使用温度传感器测量液体温度。
- 将其与期望的温度值进行比较。
- 生成一个控制信号，以设置控制冷却剂流量的执行器的位置。

软件必须执行：

- 数据采集。
- 信号线性化和缩放。
- 控制计算。
- 执行器驱动。

然而，这是一个核反应堆控制系统中的安全完整性等级 4（SIL4）的子系统。因此，禁止使用中断。

没有唯一的代码解决方案，但它肯定会是以下形式（代码清单 1.1）。

我们所拥有的是一个“应用级”代码的例子，这里它被构造成一个单一的顺序程序单元。还要注意，低级别的细节对我们是隐藏的。

总的来说，低级别操作涉及系统硬件和相关活动。即使使用高级语言，程序员也必须对机器硬件和功能有专业的知识。这也凸显了传统微处理器编程的一个问题：要实现好的设计，需要硬件/软件两方面的专业知识。即使对于这个简单的例子，程序员也需要相当程度的硬件和软件技能。

0.8 第二节标题

乍一看，在本章开头就谈论软件质量似乎有些奇怪。这似乎与操作系统关系不大。但事实并非如此，我们可以在这里学到一些宝贵的经验。

如果让你定义“高质量”软件的含义，你会怎么说？不妨看看以下几点：

- 它应该正确地完成其工作（“功能”正确性）。
- 它应该在正确的时间内完成其工作（“时间”正确性）。
- 其行为应该是可预测的。
- 其行为应该是一致的。
- 代码不应难以维护（低复杂度）。
- 代码的正确性可以被分析（静态分析）。
- 代码的行为可以被分析（覆盖率分析）。
- 运行时性能应该是可预测的。
- 内存需求应该是可预测的。
- 如果需要，代码可以被证明符合相关标准。

当然，你可以根据自己的特定需求扩展这个列表。

现在，考虑将这些原则应用于下面这个小型、相对简单的实时系统（图 1.1）。

这里的要求是通过改变液体的流速来控制其温度，它是通过以下步骤完成的：

- 使用温度传感器测量液体温度。
- 将其与期望的温度值进行比较。
- 生成一个控制信号，以设置控制冷却剂流量的执行器的位置。

软件必须执行：

- 数据采集。
- 信号线性化和缩放。
- 控制计算。
- 执行器驱动。

然而，这是一个核反应堆控制系统中的安全完整性等级 4（SIL4）的子系统。因此，禁止使用中断。

没有唯一的代码解决方案，但它肯定会是以下形式（代码清单 1.1）。

我们所拥有的是一个“应用级”代码的例子，这里它被构造成一个单一的顺序程序单元。还要注意，低级别的细节对我们是隐藏的。

总的来说，低级别操作涉及系统硬件和相关活动。即使使用高级语言，程序员也必须对机器硬件和功能有专业的知识。这也凸显了传统微处理器编程的一个问题：要实现好的设计，需要硬件/软件两方面的专业知识。即使对于这个简单的例子，程序员也需要相当程度的硬件和软件技能。

0.9 第二节标题

乍一看，在本章开头就谈论软件质量似乎有些奇怪。这似乎与操作系统关系不大。但事实并非如此，我们可以在这里学到一些宝贵的经验。

如果让你定义“高质量”软件的含义，你会怎么说？不妨看看以下几点：

- 它应该正确地完成其工作（“功能”正确性）。
- 它应该在正确的时间内完成其工作（“时间”正确性）。
- 其行为应该是可预测的。
- 其行为应该是一致的。
- 代码不应难以维护（低复杂度）。
- 代码的正确性可以被分析（静态分析）。
- 代码的行为可以被分析（覆盖率分析）。
- 运行时性能应该是可预测的。
- 内存需求应该是可预测的。
- 如果需要，代码可以被证明符合相关标准。

当然，你可以根据自己的特定需求扩展这个列表。

现在，考虑将这些原则应用于下面这个小型、相对简单的实时系统（图 1.1）。

这里的要求是通过改变液体的流速来控制其温度，它是通过以下步骤完成的：

- 使用温度传感器测量液体温度。
- 将其与期望的温度值进行比较。
- 生成一个控制信号，以设置控制冷却剂流量的执行器的位置。

软件必须执行：

- 数据采集。
- 信号线性化和缩放。
- 控制计算。
- 执行器驱动。

然而，这是一个核反应堆控制系统中的安全完整性等级 4（SIL4）的子系统。因此，禁止使用中断。

没有唯一的代码解决方案，但它肯定会是以下形式（代码清单 1.1）。

我们所拥有的是一个“应用级”代码的例子，这里它被构造成一个单一的顺序程序单元。还要注意，低级别的细节对我们是隐藏的。

总的来说，低级别操作涉及系统硬件和相关活动。即使使用高级语言，程序员也必须对机器硬件和功能有专业的知识。这也凸显了传统微处理器编程的一个问题：要实现好的设计，需要硬件/软件两方面的专业知识。即使对于这个简单的例子，程序员也需要相当程度的硬件和软件技能。