Embedded Real-Time Operating System Design

RTOS.X: Theory, Analysis, Performance and Portability

First Edition

Greg P. Semeraro, PhD
Fairport, NY USA

GPS Engineering Consulting, LLC
18 Royal Birkdale Court
Penfield, NY 14526

Preface

This text is the result of more than thirty years of designing hardware, Field Programmable Gate Array (FPGA) logic, System-on-Chip (SoC) components, low-level firmware, and software for embedded, real-time systems, as well as experience in advanced microarchitecture research and design. That unique experience provides the context for the real-time operating system (RTOS) design presented and analyzed in this text. The text brings together all these perspectives resulting in a comprehensive description of not only what an RTOS is and how to design it, but also why decisions should be made in one way or another. The results is that this text provides a comprehensive guide through the design of an RTOS with detailed explanations and justifications for all design decisions along the way. In this way, the reader can appropriately adapt the RTOS concepts presented to real-time applications, and, more importantly, design additional features and capabilities into the RTOS which leverage the underlying knowledge of the design trade-offs presented.

This text will focus on the RTOS kernel, that is, the core components and fundamental internal data structures of a complete and fully functional RTOS. The RTOS design presented in this text can be used as-is for many real-time applications and embedded products on the four hardware platforms presented. The RTOS can also be used as a foundation upon which a more feature-rich RTOS can be designed, if the principles and concepts presented are carried through to the RTOS extensions as well. Additionally, the RTOS can be used as a basis for porting to a hardware platform other than the four described in the text. The source code and RTOS architecture are specifically designed to support different hardware platforms by localizing and paying special attention all hardware-specific aspects of the RTOS design.

Objectives

Simply put, the objectives of this text are to provide an understanding of the principles guiding the design of an RTOS and an understanding of the detailed software design and implementation of an RTOS adhering to these principles. These principles are also essential to the design of real-time applications, therefore providing a level of understanding that can be leveraged in many application domains. A common misconception is that having the knowledge and skills to design and develop software or even embedded firmware is all that is needed to design real-time software or firmware. The issue is that real-time applications include all the logic and complexity of other software but also include temporal constraints and requirements on top of all that. In addition, real-time applications include the complexities of multi-threaded software. These additional issues make real-time application design and development challenging. Understanding the underlying design of the RTOS significantly reduces this complexity by providing an additional perspective. In the process of learning about the design of an RTOS, the reader will be able to design better, more efficient and, (more importantly) more correct real-time applications as well as analyze and debug real-time applications more efficiently and easily.

An important distinction between a traditional desktop or server operating system and a real-time operating system is that the former must be general-purpose and must do many things well, whereas an RTOS is, to a certain degree, specialized. That is, an RTOS contains features and capabilities that allow it to support a wide range of special-purpose, real-time applications but it does not attempt to support general-purpose applications. Real-time applications do not need all the features available in a general-purpose operating system, and even for those features that are common, real-time applications need guarantees in temporal performance that only an RTOS can provide. This text describes the features of an RTOS that are necessary to support the vast majority of real-time applications and explains how additional features can broaden the range of real-time applications that the RTOS can support even further while still maintaining real-time characteristics. This is important because the temptation is to add general-purpose operating system features to the RTOS to make its use broader; it is important to understand the characteristics of an RTOS to know when adding features can make the result no longer a real-time operating system. This is especially true when it comes to input / output peripherals; nearly all real-time applications will need to interact with peripheral devices, but not all peripherals can be readily supported by an RTOS without the RTOS losing the ability to meet temporal guarantees.

In this text four microprocessor / microcontroller hardware platforms and development environments will be fully described, and the RTOS developed can be used on each of these platforms. An interesting feature of the RTOS is that very little of the software that comprises the RTOS is specific to the platform on which it is running. This means that although the development is specific to the platforms, it is a simple matter for the RTOS to be migrated or ported to another microprocessor or microcontroller. To achieve this port, only the hardware-specific code (clearly discussed in the text) needs to be ported. The rest of the RTOS code simply needs to be compiled for the target device.

Although most real-time applications can be adequately supported by an

RTOS with simple scheduling and process management algorithms, there are some applications which can benefit from more sophisticated approaches. In this text those advanced algorithms will be described and developed as well. In most cases, the overhead associated with these advanced techniques is not significant, allowing these sophisticated algorithms to be included in the RTOS even for applications which do not initially require them but might require them in the future. That said, there are instances where a trade-off must be made to include advanced scheduling or advanced process management algorithms when they are not always needed, simply because of the additional code space that the algorithms require. It is therefore important to understand how these advanced algorithms can be used and when they can provide a benefit to a real-time application.

For critical software development (which an RTOS certainly is) an important aspect is the ability to validate that the design and implementation is correct and to be able to perform regression testing whenever that critical software is changed. To that end, a validation methodology and test suite will be developed so that full confidence in the RTOS can be achieved. This is especially important when developing something as complex, critical and largely obscured from the real-time application development that it supports. It is important that the development, testing, and debugging of the real-time application be done knowing that the underlying RTOS is known to be correct independent of the application that it is running.

Lastly, the temporal performance of an RTOS must be known for it to be properly used in a real-time application. Proper design approaches result in best performance but it must also be possible to quantify the RTOS temporal performance accurately. In this text the philosophy, means, and techniques to perform this performance analysis will be discussed so that any RTOS (even a port to another hardware platform) can be assessed quantitatively, objectively and accurately.

Skills Required

This text is intended for upper-division or graduate students of computer science, software engineering, computer engineering, or practicing engineers with similar background and education. It is assumed that the reader has full knowledge of the C programming language and can successfully design and develop moderately complex C applications that contain multiple modules with many related and disparate functions. It is also assumed that the reader understands how parameters are passed to functions in C applications as well as global, local, and automatic variable semantics.

In addition, the reader should be familiar with at least one assembly language. It is not necessary to know any of the specific assembly languages used in this text as only small assembly language fragments will need to be written. It is important that the reader have a solid understanding of how assembly language programs are written, as well as knowledge of at least one processor organization and understanding of hardware stacks and how they are used. All other hardware-specific information will be presented in the text and will pose no difficulties for most readers.

A working knowledge of pointer-based, high-level data structures (lists, queues, trees, etc.) is also required. These data structures will be developed, and are integral to the operation of the RTOS; prior knowledge of data structure design and development is not required but a solid understanding of how each of these structures is used and an understanding of their advantages and disadvantages is required. It is also important that the reader understand multi-threaded software concepts, such as, having experience with moderately complex application development using Java, POSIX or .NET threads.

Finally, the reader is expected to be able to debug moderately complex `C` code without the use of a debugger using only code walk-through analysis and simple `printf()` debugging output. Access to and the ability to use a logic analyzer or oscilloscope can be beneficial (depending on the hardware platform being used) but is not required (especially if code walk-through analysis skills are finely honed). The ability to debug `C` code without a debugger is necessary due to the very nature of operating system, and especially, RTOS development.

How To Read This Book

This book is organized in a way that allows the reader to choose between four supported hardware platforms. Once a platform is chosen, the common chapters and sections of the book should be read along with the chosen platform sections. The sections pertaining to the other three platforms should be bypassed. If it is desirable to subsequently study another platform, those sections should be read at that time. Attempting to study multiple platforms at the same time is *not* advised. Until all of the concepts presented are understood in the context of a single hardware platform, an understanding of other platforms is unlikely to be easily achieved.

Contents