国际知名大学原版教材 —— 信息技术学科与电气工程学科系列 9

Real-time Systems

实时系统

C. M. Krishna
Kang G. Shin

清华大学出版社
http://www.tup.tsinghua.edu.cn
麦格劳-希尔教育出版集团
http://www.mheducation.com

REAL-TIME SYSTEMS

C. M. Krishna
University of Massachusetts

Kang G. Shin
The University of Michigan

Tsinghua University Press **McGraw-Hill**

京新登字 158 号

北京市版权局著作权合同登记号：图字 01-2001-3180

书　名：Real-time Systems
　　实时系统
作　者：C. M. Krishna　Kang G. Shin
出版者：清华大学出版社(北京清华大学学研大厦，邮编 100084)
　　http://www.tup.tsinghua.edu.cn
印刷者：北京密云胶印厂
发行者：新华书店总店北京发行所
开　本：787×960 1/16　印张：29.25
版　次：2001 年 9 月第 1 版　2001 年 9 月第 1 次印刷
书　号：ISBN 7-302-04748-0/TP·28 13
印　数：0001~3000
定　价：39.00 元

国际知名大学原版教材

——信息技术学科与电气工程学科系列

出 版 说 明

郑大钟

清华大学信息技术科学与技术学院

当前，在我国的高等学校中，教学内容和课程体系的改革已经成为教学改革中的一个非常突出的问题，而为数不少的课程教材中普遍存在“课程体系老化，内容落伍时代，本研层次不清”的现象又是其中的急需改变的一个重要方面。同时，随着科教兴国方针的贯彻落实，要求我们进一步转变观念扩大视野，使教学过程适应以信息技术为先导的技术革命和我国社会主义市场经济的需要，加快教学过程的国际化进程。在这方面，系统地研究和借鉴国外知名大学的相关教材，将会对推进我们的课程改革和推进我国大学教学的国际化进程，乃至对我们一些重点大学建设国际一流大学的努力，都将具有重要的借鉴推动作用。正是基于这种背景，我们决定在国内推出信息技术学科和电气工程学科国外知名大学原版系列教材。

本系列教材的组编将遵循如下的几点基本原则。(1)书目的范围限于信息技术学科和电气工程学科所属专业的技术基础课和主要的专业课。(2)教材的范围选自于具有较大影响且为国外知名大学所采用的教材。(3)教材属于在近 5 年内所出版的新书或新版书。(4)教材适合于作为我国大学相应课程的教材或主要教学参考书。(5)每本列选的教材都须经过国内相应领域的资深专家审看和推荐。(6)教材的形式直接以英文原版形式印刷出版。

本系列教材将按分期分批的方式组织出版。为了便于使用本系列教材的相关教师和学生从学科和教学的角度对其在体系和内容上的特点和特色有所了解，在每本教材中都附有我们所约请的相关领域资深教授撰写的影印版序言。此外，出于多样化的考虑，对于某些基本类型的课程，我们还同时列选了多于一本的不同体系、不同风格和不同层次的教材，以供不同要求和不同学时的同类课程的选用。

本系列教材的读者对象为信息技术学科和电气工程学科所属各专业的本科生，同时兼顾其他工程学科专业的本科生或研究生。本系列教材，既可采用作为相应课程的教材或教学参考书，也可提供作为工作于各个技术领域的工程师和技术人员的自学读物。

组编这套国外知名大学原版系列教材是一个尝试。不管是书目确定的合理性，教材选择的恰当性，还是评论看法的确切性，都有待于通过使用和实践来检验。感谢使用本系列教材的广大教师和学生的支持。期望广大读者提出意见和建议。

Real-time Systems

影 印 版 序

《实时系统》（Real-time Systems）一书由美国 Massachusetts 大学的 C. M. Krishna 教授和 Michigan 大学的 Kang G. Shin 教授共同编写。两位教授从事实时系统及相关领域的教学与研究已经很多年，发表了大量学术论文，取得了很多项研究成果，也出版过多本实时系统方面的书籍。

“实时系统”的发展非常迅速，特别是嵌入式计算机的出现并广泛应用，推动了实时系统的发展。在以往出版的相关书籍中，通常以一二个章节介绍一些具体的实时系统，如实时控制系统、实时数据库、实时操作系统、实时通信系统等，本书则全面分析并系统阐述了多种实时系统的工作原理、设计方法和性能评价方法。

本书可作为理工科相关专业研究生或高年级本科生的教材，也可作为参考书或研究用资料。另外，本书也是相关领域实习工程师必备的一本很好的工具书。本书的主要特点如下：

1．理论性强。对许多问题用数学语言进行了形式化描述，给出了相关的公式、定义、定理，并进行了推导或证明。本书主要是根据两位作者从事实时系统方面的教学和研究工作近 20 年所取得的成果写成的，同时也包含了该领域国际上许多最新的研究成果。

2．可读性好。每章都有许多例题，有些章有几十个例题，通过大量的例题阐述了相关实时系统的设计方法和性能评价方法，通过例题分析了目前许多先进的实时系统的各个方面。因此，本书不仅理论性强，同时也非常注重理论联系实际，便于读者阅读。

3．注重系统设计与性能评价。全书以实时系统的设计方法和性能评价方法为主线展开，讲述了多种实时系统的设计方法和性能评价方法，同时也介绍了几种进行实时系统设计和性能评价的工具，读者学会了这些设计方法和评价工具，对研究和开发其他实时系统很有帮助。

4．硬件与软件结合。大多数实时系统是由硬件和软件共同组成的，本书在分别介绍硬件部分和软件部分的工作原理、设计方法和性能分析方法的同时，也给出了在设计实时系统时软件与硬件如何做到平衡。

每章后面都给出进一步阅读的建议，告诉读者如何深入学习，并附有参考书和相关论文，其中有许多论文是本书作者的研究成果。另外每章后面都有练习题，便于读者系统掌握本书的内容。

汤志忠　教授
清华大学计算机科学与技术系
2001 年 7 月

ABOUT THE AUTHORS

C. M. KRISHNA

C. M. Krishna has been on the faculty of the University of Massachusetts since 1984. He has published in the areas of distributed processing, real-time systems, and fault tolerance, edited two volumes of readings for the IEEE Computer Society Press, and been Co-Guest-Editor of special issues of *IEEE Computer* and the *Proceedings of the IEEE* on real-time systems. Professor Krishna's current research deals with the reliability and performance modeling of real-time systems, fault-tolerant synchronization, distributed real-time operating systems, and real-time networks.

KANG G. SHIN

Kang G. Shin is Professor and Director of the Real-Time Computing Laboratory, Department of Electrical Engineering and Computer Science, The University of Michigan, Ann Arbor. He has authored and coauthored over 360 technical papers (about 150 of these in archival journals) and numerous book chapters in the areas of distributed real-time computing and control, fault-tolerant computing, computer architecture, robotics and automation, and intelligent manufacturing. Professor Shin is an IEEE Fellow, was the Program Chairman of the 1986 IEEE Real-Time Systems Symposium (RTSS), the General Chairman of the 1987 RTSS, the Guest Editor of the 1987 August special issue of *IEEE Transactions on Computers* on real-time systems, and an Editor of IEEE Transactions on Parallel and Distributed Systems from 1991–1995.

PREFACE

Real-time systems have proliferated over the past few years. Today, computers are found embedded in almost everything, from toasters to cars to fly-by-wire aircraft. The computational workload of such embedded systems varies widely, from machines expected to do a few arithmetic operations every second to computers executing complex calculations at tremendous rates. The consequences of computer failure also vary widely, from burnt toast at the one extreme to the loss of life in an air crash or a chemical plant explosion at the other.

The objective of this book is to introduce readers to design and evaluation issues in such systems. We cover a wide range of topics; both hardware and software issues are treated in some detail. We expect this book to be used by both practicing engineers and graduate or final-year undergraduate students.

Some of the discussion is mathematical. Wherever possible, we have separated the more mathematical portions from the descriptive. This enables the text to be read at multiple levels. The more advanced sections are starred ($\star$); these require additional perseverance or ability to understand them, and can be skipped if necessary. However, we urge the reader to avoid skipping the mathematical portions. Most often, avoidance of mathematics is grounded on nothing more substantial than a primitive fear of and negative associations with mathematical symbols. A true understanding of many of the issues covered here cannot be achieved without understanding their mathematical underpinnings.

This book contains far more material than can comfortably be covered in a one-semester course. Instructors may decide to concentrate on particular topics, for example, task assignment and scheduling, or fault-tolerance. Alternatively, they may decide to present a wide-ranging survey of the various topics of interest to the real-time systems engineer. To enable both approaches to be used, we have tried to make the chapters as independent of one another as possible. In addition, this allows the book to be used as a reference handbook by the practicing engineer. Typographical or other errors should be reported to the authors at

`rtbook@tikva.ecs.umass.edu`

We plan to maintain a page of errata on the World Wide Web at

`http://www.ecs.umass.edu/ece/gradfac/krishna.html`

ACKNOWLEDGMENTS

Many people have contributed to making this book a reality. We would like to thank Eric Munson of McGraw-Hill for commissioning it, and for being willing to countenance a delay of over a year beyond our original deadline. It is perhaps ironic that the authors of a book that deals largely with tasks meeting deadlines were themselves unable to meet their contracted deadline!

A number of our colleagues and students have read through this book, either in part on in its entirety, and provided valuable suggestions or pointed out mistakes. We list them below in random order.

Y.-H. Lee	C. Ravishankar	N. Soparkar
A. Ansari	J. Rexford	S. H. Son
N. Suri	J. Strosnider	F. Zhou
A. Mehra	W. Feng	A. Shaikh
T. Abdelzaher	A. Indiresan	E. Atkins
P. Ramanathan	S. Daniel	T. Koprowski
S. Wilson	K. Ramamritham	W. Preska

Beverly Monaghan of The University of Michigan and June Daehler of the University of Massachusetts provided valuable secretarial assistance. Julie F. Nemer of ETP Harrison was the copyeditor, and her comments helped improve the readability of the text. Thanks are also due to Michael J. Kolibaba for coordinating our interactions with the copyeditor and to the following reviewers: Wei Zhao, In-Sup Lee, William Marcy, and Borko Furht.

C. M. Krishna
Kang G. Shin

CONTENTS

CHAPTER 1

INTRODUCTION

After writing a book on real-time systems, you might think that we would be able to give you a precise, cogent statement of what a real-time system is. Unfortunately, we cannot. If pressed for a definition, we might say something like this:

> Any system where a timely response by the computer to external stimuli is vital is a real-time system.

The above statement is true, but only because it is almost content-free. If you take a second look at it, you will find that it raises as many questions as it answers. If you decide to dig a little deeper by taking apart the above definition, you might have the following dialogue with us:

You: What do you mean by "timely"?

Us: It means a real-time system runs tasks that have deadlines.

You: By "deadlines" do you mean that the task *must* be done by then?

Us: Not necessarily. Sometimes, yes: If you are controlling an aircraft by computer and you miss a sequence of deadlines as the aircraft comes in to land, you risk crashing the plane. Sometimes, no: If you are playing a video game and the response takes a mite longer than specified, nothing awful will happen.

You: What do you mean by a task being "done"? Is there a sharp distinction between when a task is "done" and when it is not?

Us: Not necessarily. Sometimes, yes: If you have a banking application that needs to total some figures before it will let you draw a million dollars from your checking account, then yes. Sometimes, no: If your application needs to calculate the value of π, it can decide either to stop early and accept a less accurate value, or to continue calculating and make the estimate more and more accurate.

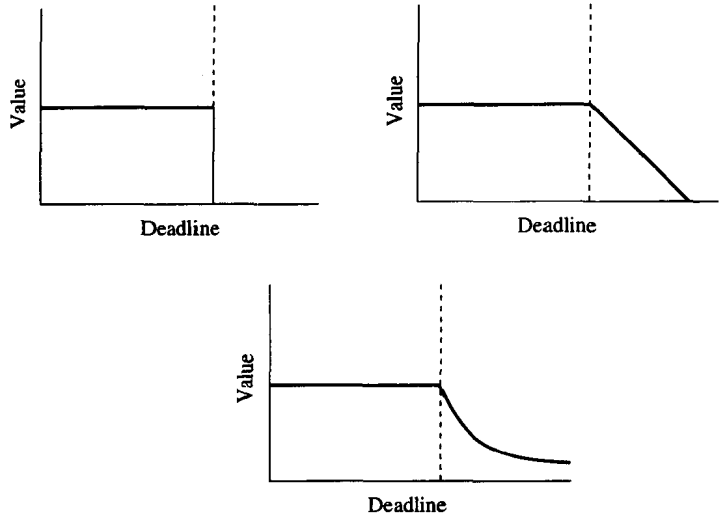

FIGURE 1.1
Examples of task value functions.

You: What do you do with a real-time task that misses its deadline? Do you drop it or complete it anyway?

Us: It depends. If you are on the aircraft that has crashed because a series of deadlines has been missed, neither you nor the computer is in a position to care. If, on the other hand, you have a video-conferencing application that encounters a minor delay in processing a voice packet, you may decide not to drop that packet. In any case, a task's value will drop to a certain level after the deadline has been missed. In some cases, it will be reduced abruptly to zero; in others, it will decline more gradually. Figure 1.1 shows some examples.

You: Does this make every computer a real-time computer by your definition?

Us: Unfortunately, yes. If you read our definition too legalistically, the general-purpose workstation or personal computer is also a real-time system: if you hit the key and the computer takes an hour to echo the character onto the screen, you will not be very happy. Everything is "real-time" in the sense of our needing the result within a finite time. So, our definition covers all computers and is therefore worthless.

You: Do you want to change your definition of real-time systems?

Us: Yes. The new definition is fuzzier, less sweeping, and not as clear-cut, but it has the inestimable virtue of ending this argument.

> A real-time system is anything that we, the authors of this book, consider to be a real-time system. This includes embedded systems that control things like aircraft, nuclear reactors, chemical power plants, jet engines, and other objects where Something Very Bad will happen if the computer does not

deliver its output in time. These are called *hard* real-time systems. There is another category called (not surprisingly) *soft* real-time systems, which are systems such as multimedia, where nothing catastrophic happens if some deadlines are missed, but where the performance will be degraded below what is generally considered acceptable. In general, a real-time system is one in which a substantial fraction of the design effort goes into making sure that task deadlines are met.

1.1 A CAR-AND-DRIVER EXAMPLE

To understand the major issues of real-time computing, consider a familiar problem of human control—driving a car. The driver is the real-time controller, the car is the controlled process, and the other cars together with the road conditions make up the operating environment. By understanding what is required of the driver, we can understand something of the major issues in real-time computing.

The main actuators in a car are the wheels, the engine, and the brakes. The controls are the accelerator, the steering wheel, the brake-pedal, and other elements (such as the switches for the lights, radio, wipers, etc.).

What are the constraints that the driver must meet? She must get from her starting point to the destination without colliding with other vehicles or stationary objects and keep her speed within designated limits. Let us translate this into the terms used in real-time computing.

The driver may be said to be performing a *mission*—that of getting to her destination while satisfying the constraints mentioned above. How can we quantify the driver's performance? We measure the outcome of the driver's actions, taken in the context of the operating environment. One obvious outcome is getting to the destination; a secondary consideration is the time it takes to get there. However, our assessment of the time taken must be weighted by a consideration of the road conditions: A driver may be performing very well if she maintains an average speed of 15 mph in a snowstorm (and gets to her destination without killing or maiming herself or anyone else); if the weather is dry and the roads are clear, however, the same speed of 15 mph represents an abject failure.

Suppose the driver fails to get to her destination and instead ends up unhurt in a ditch by the side of the road. Once again, our assessment of the driver's performance depends not just on the outcome—her landing in the ditch—but also on what caused her to get there. If she was forced into the ditch to avoid a head-on collision with someone driving on the wrong side of the road, we count that as a success; if she went into the ditch by taking a turn too quickly and skidding, we count that as a failure.

The point is that performance is not an absolute commodity. Performance must be measured instead in terms of what the conditions allow. In other words, *performance* measures the goodness of the outcome relative to the best outcome possible under the circumstances. This is an important point, and we shall return to it in Chapter 2.

Let us consider now the tasks that the driver must perform. Some of them are critical to the success of the mission, and some are not. Steering and braking

are critical tasks; tuning the radio is a noncritical task. Steering and braking are real-time tasks with varying deadlines that depend on the operating environment. The deadlines associated with driving down a quiet street at 6 o'clock on a Sunday morning are very different from those associated with driving along a busy highway at the height of the rush hour. *Task deadlines* in real-time systems are not constant; they vary with the operating environment.

What information do we need about the driver's physical condition in order to predict her performance? It is not sufficient to know that the driver will be awake for 99.99% of her trip. If she falls asleep for more than a second or two at a stretch, she will fail catastrophically. If, on the other hand, there are many periods of "micro-sleep" along the way, each lasting no more than half a second, the chances are high that she will not fail. Simple average measures are useless to predict the performance of a real-time controller.

Even if she gets to her destination safely, it is possible that the micro-sleeping driver has cause to brake and accelerate abruptly. This can increase her fuel consumption; so even if she completes her mission successfully, her micro-sleep has not been free of cost. It is possible to discriminate on the basis of secondary factors, such as fuel consumption or time taken, among missions that have been successfully completed.

Suppose we try to precisely specify the responsibilities of the driver. The overall goal of the mission is clear: get to the destination safely, without committing any traffic violations. But what of the details? Suppose we are to include in the specifications precisely what the driver must do in each conceivable eventuality. How would we write such specifications and then ensure that the specifications were complete? The reader might, as an exercise, try writing out a set of specifications in plain English. A few minutes of this activity will be sufficient to convince him that to write a complete and correct set of specifications for even so well understood a task as driving a car is extremely difficult. Writing out formal specifications and validating them are perhaps the most difficult tasks in real-time systems. They are also the tasks about which researchers know the least.

With this background, we now turn to listing some major issues in real-time computing.

1.2 ISSUES IN REAL-TIME COMPUTING

A real-time computer must be much more reliable than its individual hardware and software components. It must be capable of working in harsh environments, rich in electromagnetic noise and elementary-particle radiation, and in the face of rapidly changing computation loads.

The field of real-time computing is especially rich in research problems because all problems in computer architecture, fault-tolerant computing, and operating systems are also problems in real-time computing, with the added complexity that real-time constraints must be met.

For example, take task scheduling. The purpose of task scheduling in a general-purpose system is fairness, by which we mean that the computer's resources must be shared out equitably among the users. This end is usually achieved