REAL-TIME SYSTEMS AND THEIR PROGRAMMING LANGUAGES

REAL-TIME SYSTEMS AND THEIR PROGRAMMING LANGUAGES

Alan Burns
University of Bradford

Andy Wellings
University of York

ADDISON-WESLEY
PUBLISHING
COMPANY

Wokingham, England · Reading, Massachusetts · Menlo Park, California
New York · Don Mills, Ontario · Amsterdam · Bonn
Sydney · Singapore · Tokyo · Madrid · San Juan

Cover designed by Crayon Design of Henley-on-Thames
and printed by The Riverside Printing Co. (Reading) Ltd.
Typeset by CRB Typesetting Services, Ely, Cambs.
Printed and bound in Great Britain by T.J. Press, Padstow, Cornwall.

First printed 1989.

British Library Cataloguing in Publication Data
Burns, Alan
Real-time systems and their programming languages.
1. Real time computer systems. Programming languages
I. Title II. Wellings, Andrew
005.13

ISBN 0-201-17529-0

Library of Congress Cataloging-in-Publication Data
Burns, Alan
Real-time systems and their programming languages / Alan Burns, Andrew J. Wellings.
p. cm.
Includes bibliographical references.
ISBN 0-201-17529-0
1. Real-time data processing. 2. Real-time programming. 3. Programming languages (Electronic computers) I. Wellings, Andrew J. II. Title.
QA76.54.B87 1990
004'.33--dc20

89-17930
CIP

Preface

In 1981 a software error caused a stationary robot to move suddenly, with impressive speed, to the edge of its operational area. A nearby worker was crushed to death.

This is just one example of the hazards of embedded real-time systems. Unfortunately, it is not an isolated incident. Every month the newsletter *Software Engineering Notes* has pages of examples of events in which the malfunctioning of real-time systems has put the public or the environment at risk. What these sobering descriptions illustrate is that there is a need to take a system-wide view of embedded systems. Indeed, it can be argued that there is a requirement for real-time systems to be recognized as a distinct engineering discipline. This book is a contribution towards the development of this discipline. It cannot, of course, cover all the topics that are apposite to the study of real-time systems engineering; it does, however, present a comprehensive description and assessment of the programming languages used in this domain. Particular emphasis is placed on language primitives and their role in the production of reliable, safe and dependable software.

Audience

The book is aimed at Final Year and Masters students in Computer Science and related disciplines. It has also been written with the professional software engineer, and real-time systems engineer, in mind. Readers are assumed to have knowledge of a sequential programming language, such as Pascal, and to be familiar with the basic tenets of software engineering. The material presented reflects the content of courses developed over a number of years by the authors at their respective universities. These courses specifically address real-time systems and their programming languages.

Structure and content

In order to give the chapters continuity three programming languages are considered in detail: Ada, Modula-2 and occam 2. These languages have been chosen because they are actually used for software production. Other theoretical or experimental languages are discussed when they offer primitives not available within the core languages. Practitioners who are primarily interested in only one of these language should find sufficient material for their needs. The authors believe that a full appreciation of a language like Ada (say) can only be obtained through a comparative study of its facilities.

In all, the book contains 17 chapters, the first 11 of which are loosely organized into the following four groups. Chapters 1 through 4 represent an extended introduction. The characteristics and requirements of real-time systems are presented, then an overview of the design of such systems is given. Design is not the primary focus of this book; nevertheless, it is important to discuss implementation within an appropriate context – this chapter attempts to provide such a context. Also considered in this chapter are general criteria by which languages can be assessed. Chapters 3 and 4 consider basic language structures through discussions on *programming in the small* and *programming in the large*. These chapters also serve to introduce Ada, Modula-2 and occam 2. Readers familiar with these languages, and the basic properties of real-time systems, can progress quickly through these four opening chapters. For other readers the material presented will help to make the book more self contained.

Chapters 5 and 6 concern themselves with the production of reliable software components. Although consideration is given to fault prevention, attention is primarily focused on fault tolerance. Both forward and backward error recovery techniques are considered. The use of an exception handling facility is discussed in Chapter 6. Both resumption and termination models are described, as are the language primitives found in Ada, CHILL and Mesa.

Real-time systems are inherently concurrent, and, therefore, the study of this aspect of programming languages is fundamental. Chapter 7 introduces the notion of process and reviews the many different models that are used by language designers. Communication between processes is considered in the following two chapters. Shared-variable methods are described including the use of semaphores and monitors. Message-based models are, however, more popular in modern languages; combining as they do communication and synchronization. These models are covered in Chapter 9. Particular attention is given to the primitives of Ada and occam 2.

It is debatable whether issues of reliability or concurrency should have been considered first within the book. Both authors have experimented with reversing the order and have found little to choose between

the two possible approaches. The book can in fact be used in either mode with only one or two topics being 'out of place'. The decision to cover reliability first, reflects the authors' belief that safety is the predominant requirement of real-time systems.

The final grouping incorporates Chapters 10 and 11. In general, the relationship between system processes can be described as either cooperating (to achieve a common goal) or competing (to acquire a shared resource). Chapter 10 extends the earlier discussions on fault tolerance by describing how reliable process cooperation can be programmed. Central to this discussion is the notion of an *atomic action*. Competing processes are considered in the following chapter. An assessment is given of different language features. One important topic here is the distinction between conditional and avoidance synchronization within the concurrency model.

The remaining chapters are, essentially, self contained. Temporal requirements constitute the distinguishing characteristic of real-time systems. Chapter 12 presents a detailed discussion of these requirements and of the language facilities and implementation strategies that are used to satisfy them. Hard real-time systems have timing constraints that must be satisfied; soft systems can occasionally fail to perform adequately. Both are considered within the context of deadline scheduling. The notion of *priority* is discussed at length.

Recent advances in hardware and communications technology have made distributed computer systems a viable alternative to uniprocessor and centralized systems in many embedded application areas. Although, in some respects, distribution can be thought of as an implementation consideration, issues which arise when applications are distributed raise fundamental questions that go beyond mere implementation details. Chapter 13 considers four areas of interest: partitioning and configuration, reliability in the presence of processor and communication failure, algorithms for distributed control, and multiprocessor and distributed deadline scheduling. This chapter is specifically designed to be self-contained and can be omitted by students on shorter courses.

One important requirement of many real-time systems is that they incorporate external devices that must be programmed (that is, controlled) as part of the application software. This low-level programming is at variance with the abstract approach to software production that characterizes software engineering. Chapter 14 considers ways in which low-level facilities can be successfully incorporated into high-level languages.

A popular misconception surrounding real-time systems is that they must be highly efficient. This is not in itself true. Real-time systems must satisfy timing constraints (and reliability requirements); efficient implementation is one means of extending the realms of possibility, but it is not an end in itself. Chapter 15 reviews some of the strategies that can be used to improve the performance of language implementations.

The final major chapter of the book is a case study programmed in Ada. An example from a mine control system is used. Inevitably, a single scaled-down study cannot illustrate all the issues covered in the previous chapters: in particular, factors such as size and complexity are not addressed. Nevertheless, the case study does cover many important aspects of real-time systems.

All chapters have summaries and further reading lists. Most also have lists of exercises. These have been chosen to help readers consolidate their understanding of the material presented in each chapter. They mostly represent exercises that have been used by the authors for assessment purposes.

Ada, Modula-2 and occam 2

Currently, Ada is under review. The examples in this book conform to the ANSI/MIL-STD 1815A standard. Modula-2 is defined informally by Wirth's textbooks, however, there is a move to produce an ISO standard. The examples in this book conform to the language defined in the second edition of *Programming in Modula-2*. The occam 2 language is still in its infancy and will no doubt mature over the coming years. The occam 2 examples presented in this book conform to the occam 2 definition given by INMOS.

To facilitate easy identification of the three languages, different presentation styles are used. Ada is presented with keywords in **lower case bold**; program identifiers are given in UPPER CASE. Both Modula-2 (and Modula-1) and occam 2 require keywords to be upper case. As these languages are easily distinguished the same style of presentation has been adopted: namely keywords in UPPER CASE (UPPER CASE for occam 2) and Mixed-Case (Mixed-Case for occam 2) identifiers. All other languages have keywords in lower case.

Braille copies

Braille copies of this book, on paper or Versabraille cassette, can be made available. Enquiries should be addressed to Dr Alan Burns, Department of Computing, University of Bradford, Bradford, West Yorkshire, BD7 1DP, UK.

Acknowledgements

The material in this book has been developed over the last five years and presented to many third year and MSc students at the Universities of Bradford and York, taking Computer Science or Electronics degrees. We

would like to acknowledge their contribution to the end product, for without them this book would never have been written.

Many people have read and commented on a first draft of the book. In particular we would like to thank: Martin Atkins, Chris Hoggarth, Andy Hutcheon, Andrew Lister, and Jim Welsh. We would also like to thank our colleagues at our respective Universities for providing us with a stimulating environment and for many enlightening discussions, particularly Ljerka Beus-Dukic, Geoff Davies, John McDermid, Gary Morgan, Rick Pack, Rob Stone and Hussein Zedan.

During 1988 Alan Burns was on sabbatical at the Universities of Queensland and Houston. We would like to thank all staff at these institutions particularly Andrew Lister, Charles McKay and Pat Rogers.

This book would not have been possible without the use of electronic mail over JANET. We would like to thank the Computer Board of the United Kingdom University Grants Council and the Science and Engineering Research Council for providing this invaluable service.

Finally, we would like to give special thanks to Sylvia Holmes and Carol Burns. Sylvia for the many hours she has spent painstakingly proofreading the final manuscript and Carol for the many evenings she has tolerated our meetings and discussions.

Alan Burns
Andy Wellings

November 1989

Contents

Chapter 1
Introduction to Real-time Systems

As computers become smaller, faster, more reliable and cheaper so their range of application widens. Built initially as equation solvers their influence has extended into all walks of life from washing machines to air traffic control. One of the fastest expanding areas of computer exploitation is that involving applications, whose prime function is *not* that of information processing, but which nevertheless require information processing in order to carry out their prime function. A microprocessor-controlled washing machine is a good example of such a system. Here the prime function is to wash clothes; however, depending on the type of clothes to be washed, different 'wash programs' must be executed. These types of computer applications are generically called **real-time** or **embedded**. They place particular requirements on the computer languages needed to program them – as they have different characteristics from the more traditional information processing systems.

This book is concerned with embedded computer systems and their programming languages. It studies the particular characteristics of these systems and discusses how some modern real-time programming languages have evolved.

1.1 Definition of a real-time system

Before proceeding further it is worth trying to define the phrase 'real-time system' more precisely. There are many interpretations of the exact nature of a real-time system; however, they all have in common the notion of response time – the time taken for the system to generate output from some associated input. The Oxford Dictionary of Computing gives the following definition of a real-time system.

> Any system in which the time at which output is produced is significant. This is usually because the input corresponds to some movement in the physical world, and the output has to relate to that same movement. The lag from input time to output time must be sufficiently small for acceptable timeliness.

Here, the word timeliness is taken in the context of the total system. For example, in a missile guidance system output is required within a few milliseconds, whereas in a computer-controlled car assembly line the response may be required only within a second.

Young (1982) defines a real-time system to be:

> any information processing activity or system which has to respond to externally-generated input stimuli within a finite and specified period.

In their most general sense both these definitions cover a very wide range of computer activities. For example, an operating system like UNIX may be considered real time in that when a user enters a command he/she will expect a response within a few seconds. Fortunately, it is usually not a disaster if the response is not forthcoming. These types of systems can be distinguished from those where *failure* to respond can be considered just as bad as a wrong response. Indeed, for some, it is this aspect that distinguishes a real-time system from others where response time is important but not crucial. Consequently, *the correctness of a real-time system depends not only on the logical result of the computation, but also on the time at which the results are produced*. Practitioners in the field of real-time computer system design often distinguish between hard and soft real-time systems. **Hard real-time systems** are those where it is absolutely imperative that responses occur within the specified deadline. **Soft real-time systems** are those where response times are important but the system will still function correctly if deadlines are occasionally missed. Soft systems can themselves be distinguished from interactive ones in which there are no explicit deadlines. For example, a flight control system of a combat aircraft is a hard real-time system because a missed deadline could lead to a catastrophe, whereas a data acquisition system for a process control application is soft as it may be defined to sample an input sensor at regular

intervals but to tolerate intermittent delays. In this book the term real-time system is used to mean both soft and hard real time. Where discussion is concerned specifically with hard real-time systems the term will be used explicitly.

In a hard or soft real-time system the computer is usually interfaced directly to some physical equipment and is dedicated to monitoring or controlling the operation of that equipment. A key feature of all these applications is the role of the computer as an information processing component within a larger engineering system. It is for this reason that such applications have become known as **embedded computer systems**. The terms 'real-time' and 'embedded' will be used interchangeably in this book.

1.2 Examples of real-time systems

Having defined what is meant by embedded systems some examples of their use are now given.

1.2.1 Process control

The first use of a computer as a component in a larger engineering system occurred in the process control industry in the early 1960s. Nowadays, the use of microprocessors is the norm. Consider the simple example, shown in Figure 1.1, where the computer performs a single activity: that of ensuring an even flow of liquid in a pipe by controlling a valve. On detecting an increase in flow the computer must respond by altering the valve angle; this response must occur within a finite period if the equipment at the receiving

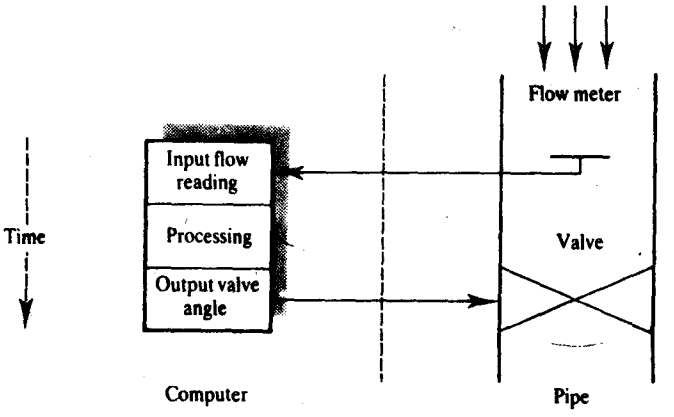

Figure 1.1 A fluid control system.

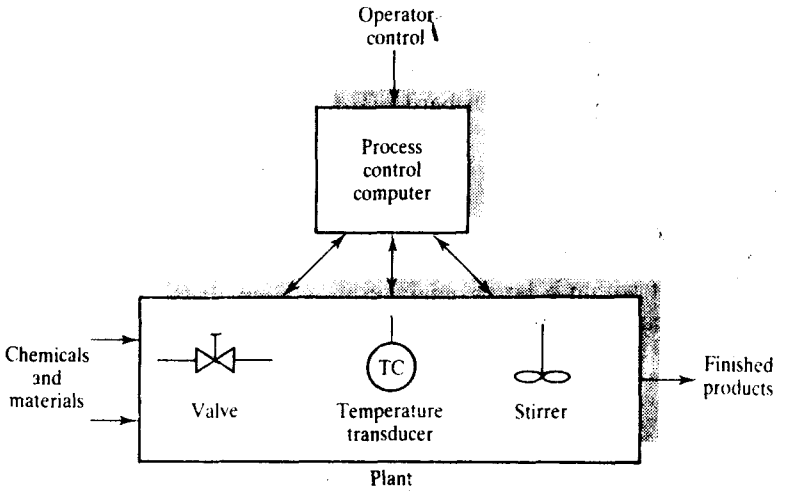

Figure 1.2 A process control system.

end of the pipe is not to become overloaded. Note that the actual response may involve quite a complex computation in order to calculate the new valve angle.

This example shows just one component of a larger control system. Figure 1.2 illustrates the role of a real-time computer embedded in a complete process-control environment. The computer interacts with the equipment using sensors and actuators. A valve is an example of an actuator and a temperature or pressure transducer is an example of a sensor. (A transducer is a device that generates an electrical signal that is proportional to the physical quantity being measured.) The computer controls the operation of the sensors and actuators to ensure that the correct plant operations are performed at the appropriate times. Where necessary, analogue to digital and digital to analogue converters must be inserted between the controlled process and the computer.

1.2.2 Manufacturing

The use of computers in manufacturing has become essential over the last few years in order that production costs can be kept low and productivity increased. Computers have enabled the integration of the entire manufacturing process from product design to fabrication. It is in the area of production control that embedded systems are best illustrated. Figure 1.3 represents, diagrammatically, the role of the production control computer