
Embedded and Real-Time Operating Systems

K.C. Wang

Embedded and Real-Time Operating Systems

K.C. Wang
School of Electrical Engineering
and Computer Science
Washington State University
Pullman, WA
USA

ISBN 978-3-319-51516-8 ISBN 978-3-319-51517-5 (eBook)
DOI 10.1007/978-3-319-51517-5

Library of Congress Control Number: 2016961664

© Springer International Publishing AG 2017

This work is subject to copyright. All rights are reserved by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed.

The use of general descriptive names, registered names, trademarks, service marks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.

The publisher, the authors and the editors are safe to assume that the advice and information in this book are believed to be true and accurate at the date of publication. Neither the publisher nor the authors or the editors give a warranty, express or implied, with respect to the material contained herein or for any errors or omissions that may have been made. The publisher remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Printed on acid-free paper

This Springer imprint is published by Springer Nature
The registered company is Springer International Publishing AG
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

Dedicated to

Cindy

Preface

Since the publication of my first book on the Design and Implementation of the MTX Operating System by Springer in 2015, I have received inquiries from many enthusiastic readers about how to run the MTX OS on their ARM based mobile devices, such as iPods or iPhones, etc. which motivated me to write this book.

The purpose of this book is to provide a suitable platform for teaching and learning the theory and practice of embedded and real-time operating systems. It covers the basic concepts and principles of operating systems, and it shows how to apply them to the design and implementation of complete operating systems for embedded and real-time systems. In order to do these in a concrete and meaningful way, it uses the ARM toolchain for program development, and it uses ARM virtual machines to demonstrate the design principles and implementation techniques.

Due to its technical contents, this book is not intended for entry-level courses that teach only the concepts and principles of operating systems without any programming practice. It is intended for technically oriented Computer Science/Engineering courses on embedded and real-time systems that emphasize both theory and practice. The book's evolutionary style, coupled with detailed source code and complete working sample systems, make it especially suitable for self-study.

Undertaking this book project has proved to be yet another very demanding and time-consuming endeavor, but I enjoyed the challenges. While preparing the book manuscripts for publication, I have been blessed with the encouragements and helps from numerous people, including many of my former TaiDa EE60 classmates. I would like to take this opportunity to thank all of them. I am also grateful to Springer International Publishing AG for allowing me to disclose the source code of this book to the public for free, which are available at <http://www.eecs.wsu.edu/~cs460/ARMhome> for download.

Special thanks go to Cindy for her continuing support and inspirations, which have made this book possible. Last but not least, I would like to thank my family again for bearing with me with endless excuses of being busy all the time.

Pullman, WA
October, 2016

K.C. Wang

Contents

1	Introduction	1
1.1	About This Book	1
1.2	Motivations of This Book	1
1.3	Objective and Intended Audience	1
1.4	Unique Features of This Book	2
1.5	Book Contents	3
1.6	Use This Book as a Textbook for Embedded Systems	4
1.7	Use This Book as a Textbook for Operating Systems	5
1.8	Use This Book for Self-study	5
	References	5
2	ARM Architecture and Programming	7
2.1	ARM Processor Modes	8
2.2	ARM CPU Registers	8
2.2.1	General Registers	8
2.2.2	Status Registers	9
2.2.3	Change ARM Processor Mode	9
2.3	Instruction Pipeline	10
2.4	ARM Instructions	10
2.4.1	Condition Flags and Conditions	10
2.4.2	Branch Instructions	11
2.4.3	Arithmetic Operations	12
2.4.4	Comparison Operations	12
2.4.5	Logical Operations	12
2.4.6	Data Movement Operations	13
2.4.7	Immediate Value and Barrel Shifter	13
2.4.8	Multiply Instructions	14
2.4.9	LOAD and Store Instructions	14
2.4.10	Base Register	14
2.4.11	Block Data Transfer	14
2.4.12	Stack Operations	14
2.4.13	Stack and Subroutines	15
2.4.14	Software Interrupt (SWI)	15
2.4.15	PSR Transfer Instructions	15
2.4.16	Coprocessor Instructions	15
2.5	ARM Toolchain	16
2.6	ARM System Emulators	16
2.7	ARM Programming	17
2.7.1	ARM Assembly Programming Example 1	17
2.7.2	ARM Assembly Programming Example 2	19
2.7.3	Combine Assembly with C Programming	20

2.8	Device Drivers	27
2.8.1	System Memory Map	27
2.8.2	GPIO Programming.	27
2.8.3	UART Driver for Serial I/O	29
2.8.4	Color LCD Display Driver.	33
2.9	Summary	45
	References.	46
3	Interrupts and Exceptions Processing	47
3.1	ARM Exceptions	47
3.1.1	ARM Processor Modes	47
3.1.2	ARM Exceptions	47
3.1.3	Exceptions Vector Table	49
3.1.4	Exception Handlers	49
3.1.5	Return from Exception Handlers.	50
3.2	Interrupts and Interrupts Processing.	51
3.2.1	Interrupt Types	51
3.2.2	Interrupt Controllers	51
3.2.3	Primary and Secondary Interrupt Controllers.	53
3.3	Interrupt Processing.	54
3.3.1	Vector Table Contents.	54
3.3.2	Hardware Interrupt Sequence	54
3.3.3	Interrupts Control in Software.	54
3.3.4	Interrupt Handlers	55
3.3.5	Non-nested Interrupt Handler	55
3.4	Timer Driver	56
3.4.1	ARM Versatile 926EJS Timers.	56
3.4.2	Timer Driver Program	57
3.5	Keyboard Driver.	61
3.5.1	ARM PL050 Mouse-Keyboard Interface	61
3.5.2	Keyboard Driver.	62
3.5.3	Interrupt-Driven Driver Design	62
3.5.4	Keyboard Driver Program	63
3.6	UART Driver.	67
3.6.1	ARM PL011 UART Interface.	67
3.6.2	UART Registers	68
3.6.3	Interrupt-Driven UART Driver Program.	69
3.7	Secure Digital (SD) Cards	73
3.7.1	SD Card Protocols	74
3.7.2	SD Card Driver	74
3.7.3	Improved SDC Driver	80
3.7.4	Multi-sector Data Transfer	82
3.8	Vectored Interrupts	84
3.8.1	ARM PL190 Vectored Interrupts Controller (VIC)	84
3.8.2	Configure VIC for Vectored Interrupts.	85
3.8.3	Vectored Interrupts Handlers	86
3.8.4	Demonstration of Vectored Interrupts	86
3.9	Nested Interrupts.	87
3.9.1	Why Nested Interrupts.	87
3.9.2	Nested Interrupts in ARM	88
3.9.3	Handle Nested Interrupts in SYS Mode	88
3.9.4	Demonstration of Nested Interrupts	89

3.10	Nested Interrupts and Process Switch	92
3.11	Summary	92
	References.	94
4	Models of Embedded Systems	95
4.1	Program Structures of Embedded Systems	95
4.2	Super-Loop Model	95
4.2.1	Super-Loop Program Examples.	96
4.3	Event-Driven Model	97
4.3.1	Shortcomings of Super-Loop Programs	97
4.3.2	Events	97
4.3.3	Periodic Event-Driven Program.	98
4.3.4	Asynchronous Event-Driven Program	101
4.4	Event Priorities.	103
4.5	Process Models.	103
4.5.1	Uniprocessor Process Model.	103
4.5.2	Multiprocessor Process Model	103
4.5.3	Real Address Space Process Model.	103
4.5.4	Virtual Address Space Process Model	103
4.5.5	Static Process Model	104
4.5.6	Dynamic Process Model	104
4.5.7	Non-preemptive Process Model.	104
4.5.8	Preemptive Process Model	104
4.6	Uniprocessor (UP) Kernel Model	104
4.7	Uniprocessor (UP) Operating System Model.	104
4.8	Multiprocessor (MP) System Model	105
4.9	Real-Time (RT) System Model.	105
4.10	Design Methodology of Embedded System Software.	105
4.10.1	High-Level Language Support for Event-Driven Programming	105
4.10.2	State Machine Model.	105
4.10.3	StateChart Model	110
4.11	Summary	110
	References.	111
5	Process Management in Embedded Systems.	113
5.1	Multitasking.	113
5.2	The Process Concept.	113
5.3	Multitasking and Context Switch	114
5.3.1	A Simple Multitasking Program	114
5.3.2	Context Switching.	116
5.3.3	Demonstration of Multitasking	120
5.4	Dynamic Processes	121
5.4.1	Dynamic Process Creation	121
5.4.2	Demonstration of Dynamic Processes	124
5.5	Process Scheduling	124
5.5.1	Process Scheduling Terminology.	124
5.5.2	Goals, Policy and Algorithms of Process Scheduling	125
5.5.3	Process Scheduling in Embedded Systems	125
5.6	Process Synchronization	126
5.6.1	Sleep and Wakeup	126
5.6.2	Device Drivers Using Sleep/Wakeup	127

5.7	Event-Driven Embedded Systems Using Sleep/Wakeup	129
5.7.1	Demonstration of Event-Driven Embedded System Using Sleep/Wakeup	131
5.8	Resource Management Using Sleep/Wakeup	132
5.8.1	Shortcomings of Sleep/Wakeup.	133
5.9	Semaphores	133
5.10	Applications of Semaphores	134
5.10.1	Semaphore Lock.	134
5.10.2	Mutex lock.	135
5.10.3	Resource Management using Semaphore	135
5.10.4	Wait for Interrupts and Messages	135
5.10.5	Process Cooperation	135
5.10.6	Advantages of Semaphores.	136
5.10.7	Cautions of Using Semaphores	136
5.10.8	Use Semaphores in Embedded Systems	137
5.11	Other Synchronization Mechanisms.	139
5.11.1	Event Flags in OpenVMS	140
5.11.2	Event Variables in MVS	140
5.11.3	ENQ/DEQ in MVS	141
5.12	High-Level Synchronization Constructs	141
5.12.1	Condition Variables.	141
5.12.2	Monitors	142
5.13	Process Communication.	142
5.13.1	Shared Memory	142
5.13.2	Pipes	142
5.13.3	Signals	147
5.13.4	Message Passing	147
5.14	Uniprocessor (UP) Embedded System Kernel	152
5.14.1	Non-preemptive UP Kernel	152
5.14.2	Demonstration of Non-preemptive UP Kernel.	158
5.14.3	Preemptive UP Kernel	158
5.14.4	Demonstration of Preemptive UP Kernel	164
5.15	Summary	165
	References.	167
6	Memory Management in ARM	169
6.1	Process Address Spaces	169
6.2	Memory Management Unit (MMU) in ARM	169
6.3	MMU Registers	170
6.4	Accessing MMU Registers.	171
6.4.1	Enabling and Disabling the MMU.	171
6.4.2	Domain Access Control	172
6.4.3	Translation Table Base Register	173
6.4.4	Domain Access Control Register.	173
6.4.5	Fault Status Registers	173
6.4.6	Fault Address Register.	174
6.5	Virtual Address Translations.	174
6.5.1	Translation Table Base	175
6.5.2	Translation Table	175
6.5.3	Level-One Descriptor	175
6.6	Translation of Section References	176

6.7	Translation of Page References	176
6.7.1	Level-1 Page Table	176
6.7.2	Level-2 Page Descriptors	176
6.7.3	Translation of Small Page References	177
6.7.4	Translation of Large Page References	178
6.8	Memory Management Example Programs	178
6.8.1	One-Level Paging Using 1 MB Sections	178
6.8.2	Two-Level Paging Using 4 KB Pages	183
6.8.3	One-Level Paging with High VA Space	184
6.8.4	Two-Level Paging with High VA Space	189
6.9	Summary	190
	Reference	191
7	User Mode Process and System Calls	193
7.1	User Mode Processes	193
7.2	Virtual Address Space Mapping	193
7.3	User Mode Process	194
7.3.1	User Mode Image	194
7.4	System Kernel Supporting User Mode Processes	197
7.4.1	PROC Structure	197
7.4.2	Reset Handler	201
7.4.3	Kernel Code	203
7.4.4	Kernel Compile-Link Script	209
7.4.5	Demonstration of Kernel with User Mode Process	209
7.5	Embedded System with User Mode Processes	209
7.5.1	Processes in the Same Domain	209
7.5.2	Demonstration of Processes in the Same Domain	210
7.5.3	Processes with Individual Domains	212
7.5.4	Demonstration of Processes with Individual Domains	212
7.6	RAM Disk	213
7.6.1	Creating RAM Disk Image	215
7.6.2	Process Image File Loader	216
7.7	Process Management	218
7.7.1	Process Creation	218
7.7.2	Process Termination	219
7.7.3	Process Family Tree	220
7.7.4	Wait for Child Process Termination	220
7.7.5	Fork-Exec in Unix/Linux	221
7.7.6	Implementation of Fork	222
7.7.7	Implementation of Exec	223
7.7.8	Demonstration of Fork-Exec	225
7.7.9	Simple sh for Command Execution	226
7.7.10	vfork	228
7.7.11	Demonstration of vfork	230
7.8	Threads	231
7.8.1	Thread Creation	231
7.8.2	Demonstration of Threads	233
7.8.3	Threads Synchronization	234
7.9	Embedded System with Two-Level Paging	234
7.9.1	Static 2-Level Paging	236
7.9.2	Demonstration of Static 2-Level Paging	237
7.9.3	Dynamic 2-Level Paging	238
7.9.4	Demonstration of 2-Level Dynamic Paging	240

7.10	KMH Memory Mapping	241
7.10.1	KMH Using One-Level Static Paging	241
7.10.2	KMH Using Two-Level Static Paging	244
7.10.3	KMH Using Two-Level Dynamic Paging.	245
7.11	Embedded System Supporting File Systems	245
7.11.1	Create SD Card Image.	246
7.11.2	Format SD Card Partitions as File Systems	247
7.11.3	Create Loop Devices for SD Card Partitions.	247
7.12	Embedded System with SDC File System	248
7.12.1	SD Card Driver Using Semaphore.	248
7.12.2	System Kernel Using SD File System	252
7.12.3	Demonstration of SDC File System.	253
7.13	Boot Kernel Image from SDC	253
7.13.1	SDC Booter Program.	254
7.13.2	Demonstration of Booting Kernel from SDC	258
7.13.3	Booting Kernel from SDC with Dynamic Paging	259
7.13.4	Two-Stage Booting	260
7.13.5	Demonstration of Two-Stage Booting	261
7.14	Summary	262
	References.	264
8	General Purpose Embedded Operating Systems	265
8.1	General Purpose Operating Systems	265
8.2	Embedded General Purpose Operating Systems.	265
8.3	Porting Existing GPOS to Embedded Systems	265
8.4	Develop an Embedded GPOS for ARM.	266
8.5	Organization of EOS.	266
8.5.1	Hardware Platform	266
8.5.2	EOS Source File Tree	267
8.5.3	EOS Kernel Files	267
8.5.4	Capabilities of EOS.	268
8.5.5	Startup Sequence of EOS.	268
8.5.6	Process Management in EOS	269
8.5.7	Assembly Code of EOS.	271
8.5.8	Kernel Files of EOS	279
8.5.9	Process Management Functions.	282
8.5.10	Pipes	282
8.5.11	Message Passing	283
8.5.12	Demonstration of Message Passing	285
8.6	Memory Management in EOS	285
8.6.1	Memory Map of EOS	285
8.6.2	Virtual Address Spaces	285
8.6.3	Kernel Mode Pgdir and Page Tables	285
8.6.4	Process User Mode Page Tables	286
8.6.5	Switch Pgdir During Process Switch	286
8.6.6	Dynamic Paging	286
8.7	Exception and Signal Processing.	288
8.7.1	Signal Processing in Unix/Linux.	288
8.8	Signal Processing in EOS	289
8.8.1	Signals in PROC Resource.	289
8.8.2	Signal Origins in EOS.	289
8.8.3	Deliver Signal to Process.	289
8.8.4	Change Signal Handler in Kernel	290

8.8.5	Signal Handling in EOS Kernel	290
8.8.6	Dispatch Signal Catcher for Execution in User Mode	291
8.9	Device Drivers	291
8.10	Process Scheduling in EOS	292
8.11	Timer Service in EOS	292
8.12	File System	294
8.12.1	File Operation Levels	294
8.12.2	File I/O Operations	296
8.12.3	EXT2 File System in EOS	302
8.12.4	Implementation of Level-1 FS	304
8.12.5	Implementation of Level-2 FS	310
8.12.6	Implementation of Level-3 FS	316
8.13	Block Device I/O Buffering	318
8.14	I/O Buffer Management Algorithm	318
8.14.1	Performance of the I/O Buffer Cache.	321
8.15	User Interface.	321
8.15.1	The INIT Program	321
8.15.2	The Login Program.	322
8.15.3	The sh Program	322
8.16	Demonstration of EOS.	323
8.16.1	EOS Startup.	323
8.16.2	Command Processing in EOS.	323
8.16.3	Signal and Exception Handling in EOS	323
8.17	Summary	326
	References.	327
9	Multiprocessing in Embedded Systems.	329
9.1	Multiprocessing	329
9.2	SMP System Requirements	329
9.3	ARM MPcore Processors.	331
9.4	ARM Cortex-A9 MPcore Processor.	331
9.4.1	Processor Cores	331
9.4.2	Snoop Control Unit (SCU).	332
9.4.3	Generic Interrupt Controller (GIC)	332
9.5	GIC Programming Example	332
9.5.1	Configure the GIC to Route Interrupts.	332
9.5.2	Explanations of the GIC Configuration Code	339
9.5.3	Interrupt Priority and Interrupt Mask	340
9.5.4	Demonstration of GIC Programming	340
9.6	Startup Sequence of ARM MPcores	340
9.6.1	Raw Startup Sequence.	341
9.6.2	Booter Assisted Startup Sequence	341
9.6.3	SMP Booting on Virtual Machines	341
9.7	ARM SMP Startup Examples	341
9.7.1	ARM SMP Startup Example 1	342
9.7.2	Demonstration of SMP Startup Example 1	346
9.7.3	ARM SMP Startup Example 2	347
9.7.4	Demonstration of ARM SMP Startup Example 2	348
9.8	Critical Regions in SMP	349
9.8.1	Implementation of Critical Regions in SMP	349
9.8.2	Shortcomings of XCHG/SWAP Operations	350
9.8.3	ARM Synchronization Instructions for SMP.	350

9.9	Synchronization Primitives in SMP	351
9.9.1	Spinlocks	351
9.9.2	Spinlock Example	352
9.9.3	Demonstration of SMP Startup with Spinlock	353
9.9.4	Mutex in SMP	353
9.9.5	Implementation of Mutex Using Spinlock	354
9.9.6	Demonstration of SMP Startup with Mutex Lock	355
9.10	Global and Local Timers	357
9.10.1	Demonstration of Local Timers in SMP	359
9.11	Semaphores in SMP	359
9.11.1	Applications of Semaphores in SMP	360
9.12	Conditional Locking	361
9.12.1	Conditional Spinlock	361
9.12.2	Conditional Mutex	361
9.12.3	Conditional Semaphore Operations	362
9.13	Memory Management in SMP	362
9.13.1	Memory Management Models in SMP	363
9.13.2	Uniform VA Spaces	363
9.13.3	Non-uniform VA Spaces	365
9.13.4	Parallel Computing in Non-uniform VA Spaces	368
9.14	Multitasking in SMP	371
9.15	SMP Kernel for Process Management	372
9.15.1	The ts.s file	373
9.15.2	The t.c file	379
9.15.3	Kernel.c file	381
9.15.4	Device Driver and Interrupt Handlers in SMP	384
9.15.5	Demonstration of Process Management in SMP	387
9.16	General Purpose SMP Operating System	389
9.16.1	Organization of SMP_EOS	389
9.16.2	SMP_EOS Source File Tree	390
9.16.3	SMP_EOS Kernel Files	390
9.16.4	Process Management in SMP_EOS	391
9.16.5	Protect Kernel Data Structures in SMP	392
9.16.6	Deadlock Prevention in SMP	394
9.16.7	Adapt UP Algorithms for SMP	395
9.16.8	Device Driver and Interrupt Handlers in SMP	396
9.17	SMP_EOS Demonstration System	397
9.17.1	Startup Sequence of SMP_EOS	397
9.17.2	Capabilities of SMP_EOS	397
9.17.3	Demonstration of SMP_EOS	398
9.18	Summary	399
	References	399
10	Embedded Real-Time Operating Systems	401
10.1	Concepts of RTOS	401
10.2	Task Scheduling in RTOS	401
10.2.1	Rate-Monotonic Scheduling (RMS)	402
10.2.2	Earliest-Deadline-First (EDF) Scheduling	402
10.2.3	Deadline-Monotonic Scheduling (DMS)	403
10.3	Priority Inversion	403
10.4	Priority Inversion Prevention	404
10.4.1	Priority Ceiling	404
10.4.2	Priority Inheritance	404

10.5	Survey of RTOS	404
10.5.1	FreeRTOS	404
10.5.2	MicroC/OS (μ C/OS)	405
10.5.3	NuttX	405
10.5.4	VxWorks	406
10.5.5	QNX	407
10.5.6	Real-time Linux	407
10.5.7	Critique of Existing RTOS	409
10.6	Design Principles of RTOS	412
10.6.1	Interrupts Processing	412
10.6.2	Task Management	412
10.6.3	Task Scheduling	412
10.6.4	Synchronization Tools	412
10.6.5	Task Communication	412
10.6.6	Memory Management	412
10.6.7	File System	412
10.6.8	Tracing and Debugging	413
10.7	Uniprocessor RTOS (UP_RTOS)	413
10.7.1	Task Management in UP_RTOS	413
10.7.2	Task Synchronization in UP_RTOS	414
10.7.3	Task Scheduling in UP_RTOS	414
10.7.4	Task Communication in UP_RTOS	414
10.7.5	Protection of Critical Regions	415
10.7.6	File System and Logging	415
10.7.7	UP_RTOS with Static Periodic Tasks and Round-Robin Scheduling	416
10.7.8	UP_RTOS with Static Periodic Tasks and Preemptive Scheduling	426
10.7.9	UP_RTOS with Dynamic Tasks Sharing Resources	432
10.8	Multiprocessor RTOS (SMP_RTOS)	440
10.8.1	SMP_RTOS Kernel for Task Management	440
10.8.2	Adapt UP_RTOS to SMP	456
10.8.3	Nested Interrupts in SMP_RTOS	458
10.8.4	Preemptive Task Scheduling in SMP_RTOS	461
10.8.5	Task Switch by SGI	461
10.8.6	Timesliced Task Scheduling in SMP_RTOS	463
10.8.7	Preemptive Task Scheduling in SMP_RTOS	465
10.8.8	Priority Inheritance	468
10.8.9	Demonstration of the SMP_RTOS System	470
10.9	Summary	474
	References	474
	Index	477

About the Author



K.C. Wang received the BSEE degree from National Taiwan University, in 1960 and the Ph.D. degree in Electrical Engineering from Northwestern University, Evanston, Ill in 1965. He is currently a Professor in the School of Electrical Engineering and Computer Science at Washington State University. His academic interests are in Operating Systems, Distributed Systems and Parallel Computing.