

Contents

Foreword	xviii
Disclaimer	xix
Preface	xx
About the Authors	xxiv
Acknowledgments	xxv

1. A Memory-centric system model

1.1 Overview	2
1.2 Modeling the system	2
1.2.1 The simplest possible model	2
1.2.2 What is this “system state”?	3
1.2.3 Refining non-processor actions	4
1.2.4 Interrupt requests	4
1.2.5 An important peripheral: the timer	5
1.3 Bare-bones processor model	5
1.3.1 What does the processor do?	5
1.3.2 Processor internal state: registers	6
1.3.3 Processor instructions	7
1.3.4 Assembly language	8
1.3.5 Arithmetic logic unit	8
1.3.6 Instruction cycle	8
1.3.7 Bare bones processor model	10
1.4 Advanced processor model	11
1.4.1 Stack support	11
1.4.2 Subroutine calls	12
1.4.3 Interrupt handling	13
1.4.4 Direct memory access	13
1.4.5 Complete cycle-based processor model	14
1.4.6 Caching	15
1.4.7 Running a program on the processor	18
1.4.8 High-level instructions	19
1.5 Basic operating system concepts	20
1.5.1 Tasks and concurrency	20
1.5.2 The register file	21
1.5.3 Time slicing and scheduling	21
1.5.4 Privileges	23

1.5.5	Memory management	23
1.5.6	Translation look-aside buffer (TLB)	24
1.6	Exercises and questions	
1.6.1	Task scheduling	25
1.6.2	TLB model	25
1.6.3	Modeling the system	25
1.6.4	Bare-bones processor model	25
1.6.5	Advanced processor model	25
1.6.6	Basic operating system concepts	25
<hr/>		
2	A Practical view of the Linux System	
2.1	Overview	30
2.2	Basic concepts	30
2.2.1	Operating system hierarchy	31
2.2.2	Processes	31
2.2.3	User space and kernel space	32
2.2.4	Device tree and ATAGs	32
2.2.5	Files and persistent storage	32
	Partition	32
	File system	33
2.2.6	'Everything is a file'	33
2.2.7	Users	34
2.2.8	Credentials	34
2.2.9	Privileges and user administration	35
2.3	Booting Linux on the Arm (Raspberry Pi 3)	36
2.3.1	Boot process stage 1: Find the bootloader	36
2.3.2	Boot process stage 2: Enable the SDRAM	36
2.3.3	Boot process stage 3: Load the Linux kernel into memory	37
2.3.4	Boot process stage 4: Start the Linux kernel	37
2.3.4	Boot process stage 5: Run the processor-independent kernel code	37
2.3.5	Initialization	37
2.3.6	Login	38
2.4	Kernel administration and programming	38
2.4.1	Loadable kernel modules and device drivers	38
2.4.2	Anatomy of a Linux kernel module	39
2.4.3	Building a custom kernel module	41
2.4.4	Building a custom kernel	42

2.5 Kernel administration and programming	42
2.5.1 Process management	42
2.5.2 Process scheduling	43
2.5.3 Memory management	43
2.5.4 Concurrency and parallelism	43
2.5.5 Input/output	43
2.5.6 Persistent storage	43
2.5.7 Networking	44
2.6 Summary	44
2.7 Exercises and questions	44
2.7.1 Installing Raspbian on the Raspberry Pi 3	44
2.7.2 Setting up SSH under Raspbian	44
2.7.3 Writing a kernel module	44
2.7.4 Booting Linux on the Raspberry Pi	45
2.7.5 Initialization	45
2.7.6 Login	45
2.7.7 Administration	45
<hr/>	
3. Hardware architecture	
3.1 Overview	49
3.2 Arm hardware architecture	49
3.3 Arm Cortex M0+	50
3.3.1 Interrupt control	51
3.3.2 Instruction set	51
3.3.3 System timer	52
3.3.4 Processor mode and privileges	52
3.3.5 Memory protection	52
3.4 Arm Cortex A53	53
3.4.1 Interrupt control	53
3.4.2 Instruction set	54
Floating-point and SIMD support	55
3.4.3 System timer	56
3.4.4 Processor mode and privileges	56
3.4.5 Memory management unit	57
Translation look-aside buffer	58
Additional caches	59

3.4.6 Memory system	59
5.5.5 L1 Cache	59
5.5.5 L2 Cache	60
Data cache coherency	61
3.5 Address map	61
3.6 Direct memory access	63
3.7 Summary	64
3.8 Exercises and questions	64
3.8.1 Bare-bones programming	64
3.8.2 Arm hardware architecture	65
3.8.3 Arm Cortex M0+	65
3.8.4 Arm Cortex A53	65
3.8.5 Address map	65
3.8.6 Direct memory access	65
<hr/>	
4. Process management	
4.1 Overview	70
4.2 The process abstraction	70
4.2.1 Discovering processes	71
4.2.2 Launching a new process	71
4.2.3 Doing something different	73
4.2.4 Ending a process	73
4.3 Process metadata	74
4.3.1 The /proc file system	75
4.3.2 Linux kernel data structures	76
4.3.3 Process hierarchies	77
4.4 Process state transitions	79
4.5 Context switch	81
4.6 Signal communications	83
4.6.1 Sending signals	83
4.6.2 Handling signals	84
4.7 Summary	85
4.8 Further reading	85
4.9 Exercises and questions	85
4.9.1 Multiple choice quiz	85
4.9.2 Metadata mix	86

4.9.3 Russian doll project	86
4.9.4 Process overload	86
4.9.5 Signal frequency	86
4.9.6 Illegal instructions	86
<hr/>	
5. Process scheduling	
5.1 Overview	90
5.2 Scheduling overview: what, why, how?	90
5.2.1 Definition	90
5.2.2 Scheduling for responsiveness	90
5.2.3 Scheduling for performance	91
5.2.4 Scheduling policies	91
5.3 Recap: the process lifecycle	91
5.4 System calls	93
5.4.1 The Linux syscall(2) function	94
5.4.2 The implications of the system call mechanism	95
5.5 Scheduling principles	95
5.5.1 Preemptive versus non-preemptive scheduling	96
5.5.2 Scheduling policies	96
5.5.3 Task attributes	96
5.6 Scheduling criteria	96
5.7 Scheduling policies	97
5.7.1 First-come, first-served (FCFS)	97
5.7.2 Round-robin (RR)	98
5.7.3 Priority-driven scheduling	98
5.7.4 Shortest job first (SJF) and shortest remaining time first (SRTF)	99
5.7.5 Shortest elapsed time first (SETF)	100
5.7.6 Priority scheduling	100
5.7.7 Real-time scheduling	100
5.7.8 Earliest deadline first (EDF)	101
5.8 Scheduling in the Linux kernel	101
5.8.1 User priorities: niceness	102
5.8.2 Scheduling information in the task control block (TCB)	102
5.8.3 Process priorities in the Linux kernel	104
Priority info in task_struct	105
Priority and load weight	106

5.8.4	Normal scheduling policies: the completely fair scheduler (CFS)	107
5.8.5	Soft real-time scheduling policies	110
5.8.6	Hard real-time scheduling policy	112
	Time budget allocation	113
5.8.7	Kernel preemption models	115
5.8.8	The red-black tree in the Linux kernel	116
	Creating a new rbtree	116
	Searching for a value in a rbtree	117
	Inserting data into a rbtree	117
	Removing or replacing existing data in a rbtree	118
	Iterating through the elements stored in a rbtree (in sort order)	118
	Cached rbtrees	119
5.8.9	Linux scheduling commands and API	119
	Normal processes	119
	Real-time processes	120
5.9	Summary	120
5.10	Exercises and questions	120
	5.10.1 Writing a scheduler	120
	5.10.2 Scheduling	120
	5.10.3 System calls	121
	5.10.4 Scheduling policies	121
	5.10.5 The Linux scheduler	121
<hr/>		
6.	Memory management	
6.1	Overview	126
6.2	Physical memory	126
6.3	Virtual memory	127
	6.3.1 Conceptual view of memory	127
	6.3.2 Virtual addressing	128
	6.3.3 Paging	129
6.4	Page tables	130
	6.4.1 Page table structure	130
	6.4.2 Linux page tables on Arm	132
	6.4.3 Page metadata	134
	6.4.4 Faster translation	136
	6.4.5 Architectural details	137

6.5 Managing memory over-commitment	138
6.5.1 Swapping	138
6.5.2 Handling page faults	138
6.5.3 Working set size	141
6.5.4 In-memory caches	142
6.5.5 Page replacement policies	142
Random	143
Not recently used (NRU)	143
Clock	143
Least recently used	144
Tuning the system	144
6.5.6 Demand paging	145
6.5.7 Copy on Write (CoW)	146
6.5.8 Out of memory killer	147
6.6 Process view of memory	148
6.7 Advanced topics	149
6.8 Further reading	151
6.9 Exercises and questions	151
6.9.1 How much memory?	151
6.9.2 Hypothetical address space	152
6.9.3 Custom memory protection	152
6.9.4 Inverted page tables	152
6.9.5 How much memory?	153
6.9.6 Tiny virtual address space	153
6.9.7 Definitions quiz	153

7. Concurrency and parallelism

7.1 Overview	158
7.2 Concurrency and parallelism: definitions	158
7.2.1 What is concurrency?	158
7.2.2 What is parallelism?	158
7.2.3 Programming model view	158
7.3 Concurrency	159
7.3.1 What are the issues with concurrency?	159
Shared resources	159
Exchange of information	160

186	7.3.2 Concurrency terminology	161
186	Critical section	161
187	Synchronization	161
190	Deadlock	161
191	7.3.3 Synchronization primitives	163
194	7.3.4 Arm hardware support for synchronization primitives	163
196	Exclusive operations and monitors	163
197	Shareability domains	164
197	7.3.5 Linux kernel synchronization primitives	165
198	Atomic primitives	165
199	Memory operation ordering	169
200	Memory barriers	170
201	Spin locks	173
202	Futexes	174
202	Kernel mutexes	174
203	Semaphores	175
203	7.3.6 POSIX synchronization primitives	177
204	Mutexes	178
206	Semaphores	178
206	Spin locks	179
207	Condition variables	179
207	7.4 Parallelism	181
207	7.4.1 What are the challenges with parallelism?	181
208	7.4.2 Arm hardware support for parallelism	182
209	7.4.3 Linux kernel support for parallelism	183
209	SMP boot process	183
210	Load balancing	183
210	Processor affinity control	184
211	7.5 Data-parallel and task-parallel programming models	184
212	7.5.1 Data parallel programming	184
213	Full data parallelism: <i>map</i>	184
213	Reduction	185
213	Associativity	185
213	Binary tree-based parallel reduction	185
214	7.5.2 Task parallel programming	186

7.6	Practical parallel programming frameworks	186
7.6.1	POSIX Threads (pthreads)	186
7.6.2	OpenMP	189
7.6.3	Message passing interface (MPI)	190
7.6.4	OpenCL	191
7.6.5	Intel threading building blocks (TBB)	194
7.6.6	MapReduce	195
7.7	Summary	195
7.8	Exercises and questions	195
7.8.1	Concurrency: synchronization of tasks	196
7.8.1	Parallelism	196
<hr/>		
8.	Input/output	
8.1	Overview	202
8.2	The device zoo	202
8.2.1	Inspect your devices	203
8.2.2	Device classes	203
8.2.3	Trivial device driver	204
8.3	Connecting devices	206
8.3.1	Bus architecture	206
8.4	Communicating with devices	207
8.4.1	Device abstractions	207
8.4.2	Blocking versus non-blocking IO	207
8.4.3	Managing IO interactions	208
	Polling	209
	Interrupts	209
	Direct memory access	210
8.5	Interrupt handlers	210
8.5.1	Specific interrupt handling details	211
8.5.2	Install an interrupt handler	212
8.6	Efficient IO	213
8.7	Further reading	213
8.8	Exercises and questions	213
8.8.1	How many interrupts?	213
8.8.2	Comparative complexity	214
8.8.3	Roll your own Interrupt Handler	214
8.8.4	Morse Code LED Device	214

9. Persistent storage

9.1 Overview	218
9.2 User perspective on the file system	218
9.2.1 What is a file?	218
9.2.2 How are multiple files organized?	219
9.3 Operations on files	221
9.4 Operations on directories	222
9.5 Keeping track of open files	222
9.6 Concurrent access to files	223
9.7 File metadata	225
9.8 Block-structured storage	226
9.9 Constructing a logical file system	228
9.9.1 Virtual file system	228
9.10 Inodes	230
9.10.1 Multiple links, single inode	231
9.10.2 Directories	231
9.11 ext4	233
9.11.1 Layout on disk	233
9.11.2 Indexing data blocks	224
9.11.3 Multiple links, single inode	237
9.11.4 Checksumming	237
9.11.5 Encryption	237
9.12 FAT	238
9.12.1 Advantages of FAT	239
9.12.2 Construct a mini file system using FAT	240
9.13 Latency reduction techniques	242
9.14 Fixing up broken file systems	243
9.15 Advanced topics	243
9.16 Further reading	245
9.17 Exercises and questions	245
9.17.1 Hybrid contiguous and linked file system	245
9.17.2 Extra FAT file pointers	245
9.17.3 Expected file size	245
9.17.4 Ext4 extents	245
9.17.5 Access times	245
9.17.6 Database decisions	246

10. Networking

10.1	Overview	250
10.2	What is networking	250
10.3	Why is networking part of the kernel?	250
10.4	The OSI layer model	251
10.5	The Linux networking stack	252
10.5.1	Device drivers	253
10.5.2	Device-agnostic interface	253
10.5.3	Network protocols	253
10.5.4	Protocol-agnostic interface	253
10.5.5	System call interface	254
10.5.6	Socket buffers	254
10.6	The POSIX standard socket interface library	255
10.6.1	Stream socket (TCP) communications flow	255
10.6.2	Common internet data types	256
	Socket address data type: <i>struct sockaddr</i>	256
	Internet socket address data type: <i>struct sockaddr_in</i>	257
10.6.3	Common POSIX socket API functions	258
	Create a socket descriptor: <i>socket()</i>	258
	Bind a server socket address to a socket descriptor: <i>bind()</i>	258
	Enable server socket connection requests: <i>listen()</i>	259
	Accept a server socket connection request: <i>accept()</i>	259
	Client connection request: <i>'connect()'</i>	260
	Write data to a stream socket: <i>send()</i>	261
	Read data from a stream socket: <i>recv()</i>	261
	Setting server socket options: <i>setsockopt()</i>	262
10.6.4	Common utility functions	263
	Internet address manipulation functions	263
	Internet network/host byte order manipulation functions	263
	Host table access functions	264
10.6.5	Building applications with TCP	264
	Request/response communication using TCP	264
	TCP server	265
	TCP client	266
10.6.6	Building applications with UDP	268
	UDP server	269

UDP client	270
UDP client using connect()	271
10.6.7 Handling multiple clients	271
The select() system call	271
Multiple server processes: fork() and exec()	275
Multithreaded servers using pthreads	276
10.7 Summary	278
10.8 Exercises and questions	278
10.8.1 Simple social networking	278
10.8.2 The Linux networking stack	278
10.8.3 The POSIX socket API	278
<hr/>	
11. Advanced topics	
11.1 Overview	282
11.2 Scaling down	282
11.3 Scaling up	283
11.4 Virtualization and containerization	285
11.5 Security	287
11.5.1 Rowhammer, Rampage, Throwhammer, and Nethammer	287
11.5.2 Spectre, Meltdown, Foreshadow	288
11.6 Verification and certification	289
11.7 Reconfigurability	291
11.8 Linux development roadmap	292
11.9 Further reading	293
11.10 Exercises and questions	293
11.10.1 Make a minimal kernel	293
11.10.2 Verify important properties	293
11.10.3 Commercial comparison	293
11.10.4 For or against certification	293
11.10.5 Devolved decisions	293
11.10.6 Underclock, overclock	293
<hr/>	
Glossary of terms	296
Index	304