Embedded Linux System Design and Development

P. Raghavan • Amol Lad • Sriram Neelakandan

Auerbach Publications
Taylor & Francis Group
Boca Raton New York

Contents

1 Intr oduction 1

1.1 History of Embedded Linux 2
1.1.1 Year 1999 3
1.1.2 Year 2000 4
1.1.3 Year 2001 4
1.1.4 Year 2002 5
1.1.5 Year 2003 6
1.1.6 Year 2004 6
1.2 Why Embedded Linux? 7
1.2.1 Vendor Independence 7
1.2.2 Time to Market 8
1.2.3 Varied Hardware Support 8
1.2.4 Low Cost 8
1.2.5 Open Source 9
1.2.6 Standards (POSIX®) Compliance 10
1.3 Embedded Linux Versus Desktop Linux 10
1.4 Frequently Asked Questions 11
1.4.1 Is Linux Too Large? 11
1.4.2 Is Linux Real-Time Enough? 11
1.4.3 How Can I Protect My Proprietary Software? 12
1.4.4 Should I Buy a Commercial Embedded Linux Distribution? 12
1.4.5 Which Embedded Linux Distribution Do I Choose? 12
1.5 Embedded Linux Distributions 13
1.5.1 BlueCat Linux 14
1.5.2 Cadenux 15
1.5.3 Denx 17

1.5.4 Embedded Debian (Emdebian) 18
1.5.5 ELinOS (SYSGO) 19
1.5.6 Metrowerks 20
1.5.7 MontaVista Linux 22
1.5.8 RTLinuxPro™ 23
1.5.9 TimeSys Linux 24
1.6 Porting Roadmap 26
Notes 28

2 Getting Started 29
2.1 Architecture of Embedded Linux 29
2.1.1 Real-Time Executive 29
2.1.2 Monolithic Kernels 30
2.1.3 Microkernel 31
2.2 Linux Kernel Architecture 32
2.2.1 Hardware Abstraction Layer (HAL) 33
2.2.2 Memory Manager 33
2.2.3 Scheduler 34
2.2.4 File System 35
2.2.5 IO Subsystem 36
2.2.6 Networking Subsystems 36
2.2.7 IPC 36
2.3 User Space 36
2.4 Linux Start-Up Sequence 41
2.4.1 Boot Loader Phase 42
2.4.2 Kernel Start-Up 43
2.4.3 User Space Initialization 47
2.5 GNU Cross-Platform Toolchain 48
2.5.1 Building Toolchain 50
2.5.2 Building Toolchain for MIPS 55

3 Boar d Support Package 59
3.1 Inserting BSP in Kernel Build Procedure 60
3.2 The Boot Loader Interface 62
3.3 Memory Map 66
3.3.1 The Processor Memory Map — MIPS Memory Model 67
3.3.2 Board Memory Map 68
3.3.3 Software Memory Map 68
3.4 Interrupt Management 72
3.5 The PCI Subsystem 77
3.5.1 Uniqueness of PCI Architecture 77
3.5.2 PCI Software Architecture 79

3.6 Timers 81
3.7 UART 81
3.7.1 Implementing the Console 81
3.7.2 The KGDB Interface 82
3.8 Power Management 83
3.8.1 Hardware and Power Management 83
3.8.2 Power Management Standards 85
3.8.3 Supporting Processor's Power-Saving Modes 86
3.8.4 Unified Driver Framework for Power Management 87
3.8.5 Power Management Applications 88

4 Embedded Storage 89
4.1 Flash Map 89
4.2 MTD—Memory Technology Device 91
4.2.1 The MTD Model 91
4.2.2 Flash Chips 92
4.2.3 Flash Disks 92
4.3 MTD Architecture 94
4.3.1 `mtd_info` Data Structure 96
4.3.2 Interface Between MTD Core and Low-Level Flash Drivers 96
4.4 Sample MTD Driver for NOR Flash 97
4.5 The Flash-Mapping Drivers 106
4.5.1 Filling up `mtd_info` for NOR Flash Chip 106
4.5.2 Filling up `mtd_info` for NAND Flash Chip 108
4.5.3 Registering `mtd_info` 109
4.5.4 Sample Mapping Driver for NOR Flash 111
4.6 MTD Block and Character Devices 114
4.7 Mtdutils Package 116
4.8 Embedded File Systems 116
4.8.1 Ramdisk 117
4.8.2 RAMFS 117
4.8.3 CRAMFS (Compressed RAM File System) 117
4.8.4 Journaling Flash File Systems — JFFS and JFFS2 117
4.8.5 NFS — Network File System 119
4.8.6 PROC File System 119
4.9 Optimizing Storage Space 120
4.9.1 Kernel Space Optimization 120
4.9.2 Application Space Optimization 121
4.9.3 Applications for Embedded Linux 122
4.10 Tuning Kernel Memory 124

5 Embedded Drivers 127
5.1 Linux Serial Driver 128
5.1.1 Driver Initialization and Start-Up 130
5.1.2 Data Transmission 134
5.1.3 Data Reception 134
5.1.4 Interrupt Handler 134
5.1.5 Terminos Settings 138
5.2 Ethernet Driver 138
5.2.1 Device Initialization and Clean-Up 140
5.2.2 Data Transmission and Reception 142
5.3 I2C Subsystem on Linux 144
5.3.1 I2C Bus 145
5.3.2 I2C Software Architecture 147
5.4 USB Gadgets 152
5.4.1 USB Basics 153
5.4.2 Ethernet Gadget Driver 158
5.5 Watchdog Timer 161
5.6 Kernel Modules 162
5.6.1 Module APIs 162
5.6.2 Module Loading and Unloading 164
Notes 164

6 Porting Applications 165
6.1 Architectural Comparison 165
6.2 Application Porting Roadmap 166
6.2.1 Decide Porting Strategy 167
6.2.2 Write an Operating System Porting Layer (OSPL) 169
6.2.3 Write a Kernel API Driver 170
6.3 Programming with Pthreads 171
6.3.1 Thread Creation and Exit 172
6.3.2 Thread Synchronization 174
6.3.3 Thread Cancellation 180
6.3.4 Detached Threads 181
6.4 Operating System Porting Layer (OSPL) 182
6.4.1 RTOS Mutex APIs Emulation 182
6.4.2 RTOS Task APIs Emulation 185
6.4.3 IPC and Timer APIs Emulation 191
6.5 Kernel API Driver 191
6.5.1 Writing User-Space Stubs 194
6.5.2 Kapi Driver Implementation 195
6.5.3 Using the Kapi Driver 199
Note 200

7 Real-Time Linux 201
7.1 Real-Time Operating System 202
7.2 Linux and Real-Time 202
7.2.1 Interrupt Latency 203
7.2.2 ISR Duration 204
7.2.3 Scheduler Latency 205
7.2.4 Scheduler Duration 207
7.2.5 User-Space Real-Time 209
7.3 Real-Time Programming in Linux 209
7.3.1 Process Scheduling 210
7.3.2 Memory Locking 213
7.3.3 POSIX Shared Memory 223
7.3.4 POSIX Message Queues 225
7.3.5 POSIX Semaphores 232
7.3.6 Real-Time Signals 233
7.3.7 POSIX.1b Clock and Timers 241
7.3.8 Asynchronous I/O 246
7.4 Hard Real-Time Linux 252
7.4.1 Real-Time Application Interface (RTAI) 253
7.4.2 ADEOS 258

8 Building and Debugging 261
8.1 Building the Kernel 263
8.1.1 Understanding Build Procedure 265
8.1.2 The Configuration Process 266
8.1.3 Kernel Makefile Framework 268
8.2 Building Applications 270
8.2.1 Cross-Compiling Using Configure 273
8.2.2 Troubleshooting Configure Script 274
8.3 Building the Root File System 275
8.4 Integrated Development Environment 278
8.4.1 Eclipse 279
8.4.2 KDevelop 279
8.4.3 TimeStorm 279
8.4.4 CodeWarrior 280
8.5 Debugging Virtual Memory Problems 280
8.5.1 Debugging Memory Leaks 282
8.5.2 Debugging Memory Overflows 286
8.5.3 Debugging Memory Corruption 287
8.6 Kernel Debuggers 291
8.7 Profiling 293
8.7.1 eProf—An Embedded Profiler 294
8.7.2 OProfile 300

8.7.3 Kernel Function Instrumentation 302
Notes 308

9 Embedded Graphics 309
9.1 Graphics System 309
9.2 Linux Desktop Graphics—The X Graphics System 311
9.2.1 Embedded Systems and X 312
9.3 Introduction to Display Hardware 313
9.3.1 Display System 313
9.3.2 Input Interface 316
9.4 Embedded Linux Graphics 316
9.5 Embedded Linux Graphics Driver 316
9.5.1 Linux Frame Buffer Interface 317
9.5.2 Frame Buffer Internals 326
9.6 Windowing Environments, Toolkits, and Applications 328
9.6.1 Nano-X 335
9.7 Conclusion 340
Notes 340

10 uClinux 341
10.1 Linux on MMU-Less Systems 341
10.1.1 Linux Versus uClinux 342
10.2 Program Load and Execution 343
10.2.1 Fully Relocatable Binaries (FRB) 345
10.2.2 Position Independent Code (PIC) 345
10.2.3 bFLT File Format 346
10.2.4 Loading a bFLT File 347
10.3 Memory Management 358
10.3.1 Heap 358
10.3.2 Stack 363
10.4 File / Memory Mapping—The Intricacies of mmap( ) in uClinux 364
10.5 Process Creation 365
10.6 Shared Libraries 367
10.6.1 uClinux Shared Library Implementation (libN.so) 367
10.7 Porting Applications to uClinux 370
10.7.1 Creating uClinux Programs 370
10.7.2 Creating Shared Libraries in uClinux 371
10.7.3 Using Shared Library in an Application 373
10.7.4 Memory Limitations 375
10.7.5 mmap Limitations 375
10.7.6 Process-Level Limitations 375

10.8 XIP—eXecute In Place 375
10.8.1 Hardware Requirements 377
10.8.2 Software Requirements 378
10.9 Building uClinux Distribution 378
Notes 380

Appendices

A Booting Faster 383
Techniques for Cutting Down Bootloader Initialization 384
Tuning Kernel for Decreased Boot-Up Time 385
Tuning User Space for Decreased Boot-Up Time 385
Measuring Boot-Up Time 386

B GPL and Embedded Linux 387
User-Space Applications 387
Kernel 388
Points to Remember 389
Notes 390

Index 391