

---

# Contents

---

<i>List of Example Programs</i> . . . . .	xii
<i>Preface</i> . . . . .	xv
Intended audience . . . . .	xvi
About the author . . . . .	xvi
Acknowledgments . . . . .	xvii
<b>1 Introduction</b> . . . . .	<b>1</b>
1.1 The “bailing programmers” . . . . .	3
1.2 Definitions and terminology . . . . .	4
1.2.1 Asynchronous . . . . .	4
1.2.2 Concurrency . . . . .	4
1.2.3 Uniprocessor and multiprocessor . . . . .	5
1.2.4 Parallelism . . . . .	5
1.2.5 Thread safety and reentrancy . . . . .	6
1.2.6 Concurrency control functions . . . . .	7
1.3 Asynchronous programming is intuitive... . . . .	8
1.3.1 . . . because UNIX is asynchronous . . . . .	9
1.3.2 . . . because the world is asynchronous . . . . .	11
1.4 About the examples in this book . . . . .	12
1.5 Asynchronous programming, by example . . . . .	13
1.5.1 The baseline, synchronous version . . . . .	14
1.5.2 A version using multiple processes . . . . .	15
1.5.3 A version using multiple threads . . . . .	17
1.5.4 Summary . . . . .	19
1.6 Benefits of threading . . . . .	20
1.6.1 Parallelism . . . . .	20
1.6.2 Concurrency . . . . .	22
1.6.3 Programming model . . . . .	24
1.7 Costs of threading . . . . .	25
1.7.1 Computing overhead . . . . .	26
1.7.2 Programming discipline . . . . .	26
1.7.3 Harder to debug . . . . .	27
1.8 To thread or not to thread? . . . . .	28
1.9 POSIX thread concepts . . . . .	29
1.9.1 Architectural overview . . . . .	30
1.9.2 Types and interfaces . . . . .	30

1.9.3	Checking for errors . . . . .	31
<b>2</b>	<b>Threads . . . . .</b>	<b>35</b>
2.1	Creating and using threads . . . . .	35
2.2	The life of a thread . . . . .	39
2.2.1	Creation . . . . .	40
2.2.2	Startup . . . . .	41
2.2.3	Running and blocking . . . . .	42
2.2.4	Termination . . . . .	43
2.2.5	Recycling . . . . .	44
<b>3</b>	<b>Synchronization . . . . .</b>	<b>45</b>
3.1	Invariants, critical sections, and predicates . . . . .	45
3.2	Mutexes . . . . .	47
3.2.1	Creating and destroying a mutex . . . . .	49
3.2.2	Locking and unlocking a mutex . . . . .	52
3.2.2.1	Nonblocking mutex locks . . . . .	58
3.2.3	Using mutexes for atomicity . . . . .	61
3.2.4	Sizing a mutex to fit the job . . . . .	62
3.2.5	Using more than one mutex . . . . .	63
3.2.5.1	Lock hierarchy . . . . .	63
3.2.5.2	Lock chaining . . . . .	70
3.3	Condition variables . . . . .	70
3.3.1	Creating and destroying a condition variable . . . . .	74
3.3.2	Waiting on a condition variable . . . . .	77
3.3.3	Waking condition variable waiters . . . . .	81
3.3.4	One final alarm program . . . . .	82
3.4	Memory visibility between threads . . . . .	88
<b>4</b>	<b>A few ways to use threads . . . . .</b>	<b>97</b>
4.1	Pipeline . . . . .	98
4.2	Work Crew . . . . .	106
4.3	Client/Server . . . . .	120
<b>5</b>	<b>Advanced threaded programming . . . . .</b>	<b>131</b>
5.1	One-time initialization . . . . .	131
5.2	Attributes objects . . . . .	134
5.2.1	Mutex attributes . . . . .	135
5.2.2	Condition variable attributes . . . . .	137
5.2.3	Thread attributes . . . . .	138
5.3	Cancellation . . . . .	142
5.3.1	Deferred cancelability . . . . .	147
5.3.2	Asynchronous cancelability . . . . .	150

---

5.3.3	Cleaning up . . . . .	154
5.4	<i>Thread-specific data</i> . . . . .	161
5.4.1	Creating thread-specific data . . . . .	163
5.4.2	Using thread-specific data . . . . .	166
5.4.3	Using destructor functions . . . . .	167
5.5	<i>Realtime scheduling</i> . . . . .	172
5.5.1	POSIX realtime options . . . . .	173
5.5.2	Scheduling policies and priorities . . . . .	174
5.5.3	Contention scope and allocation domain . . . . .	181
5.5.4	Problems with realtime scheduling . . . . .	183
5.5.5	Priority-aware mutexes . . . . .	185
5.5.5.1	Priority ceiling mutexes . . . . .	186
5.5.5.2	Priority inheritance mutexes . . . . .	188
5.6	<i>Threads and kernel entities</i> . . . . .	189
5.6.1	Many-to-one (user level). . . . .	190
5.6.2	One-to-one (kernel level) . . . . .	191
5.6.3	Many-to-few (two level) . . . . .	193
<b>6</b>	<b>POSIX adjusts to threads . . . . .</b>	<b>197</b>
6.1	<i>fork</i> . . . . .	197
6.1.1	Fork handlers . . . . .	199
6.2	<i>exec</i> . . . . .	204
6.3	<i>Process exit</i> . . . . .	204
6.4	<i>Stdio</i> . . . . .	204
6.4.1	flockfile and funlockfile . . . . .	205
6.4.2	getchar_unlocked and putchar_unlocked. . . . .	207
6.5	<i>Thread-safe functions</i> . . . . .	209
6.5.1	User and terminal identification . . . . .	210
6.5.2	Directory searching . . . . .	212
6.5.3	String token . . . . .	212
6.5.4	Time representation . . . . .	212
6.5.5	Random number generation . . . . .	213
6.5.6	Group and user database . . . . .	213
6.6	<i>Signals</i> . . . . .	214
6.6.1	Signal actions . . . . .	215
6.6.2	Signal masks. . . . .	216
6.6.3	pthread_kill. . . . .	217
6.6.4	sigwait and sigwaitinfo . . . . .	227
6.6.5	SIGEV_THREAD . . . . .	230
6.6.6	Semaphores: synchronizing with a signal-catching function . . . . .	234
<b>7</b>	<b>“Real code” . . . . .</b>	<b>241</b>

7.1	<i>Extended synchronization</i> . . . . .	241
7.1.1	Barriers . . . . .	242
7.1.2	Read-write locks. . . . .	253
7.2	<i>Work queue manager</i> . . . . .	270
7.3	<i>But what about existing libraries?</i> . . . . .	283
7.3.1	Modifying libraries to be thread-safe. . . . .	284
7.3.2	Living with legacy libraries . . . . .	285
<b>8</b>	<b>Hints to avoid debugging.</b> . . . . .	<b>289</b>
8.1	<i>Avoiding incorrect code</i> . . . . .	290
8.1.1	Avoid relying on “thread inertia” . . . . .	291
8.1.2	Never bet your mortgage on a thread race . . . . .	293
8.1.3	Cooperate to avoid deadlocks . . . . .	297
8.1.4	Beware of priority inversion . . . . .	299
8.1.5	Never share condition variables between predicates . . . . .	300
8.1.6	Sharing stacks and related memory corrupters . . . . .	301
8.2	<i>Avoiding performance problems</i> . . . . .	302
8.2.1	Beware of concurrent serialization . . . . .	302
8.2.2	Use the right number of mutexes . . . . .	303
8.2.2.1	Too many mutexes will not help . . . . .	304
8.2.3	Never fight over cache lines. . . . .	304
<b>9</b>	<b>POSIX threads mini-reference</b> . . . . .	<b>307</b>
9.1	<i>POSIX 1003.1c–1995 options</i> . . . . .	307
9.2	<i>POSIX 1003.1c–1995 limits</i> . . . . .	308
9.3	<i>POSIX 1003.1c–1995 interfaces</i> . . . . .	309
9.3.1	Error detection and reporting . . . . .	310
9.3.2	Use of void* type. . . . .	311
9.3.3	Threads . . . . .	311
9.3.4	Mutexes . . . . .	316
9.3.5	Condition variables. . . . .	319
9.3.6	Cancellation. . . . .	323
9.3.7	Thread-specific data . . . . .	325
9.3.8	Realtime scheduling . . . . .	326
9.3.9	Fork handlers. . . . .	336
9.3.10	<i>Stdio</i> . . . . .	336
9.3.11	Thread-safe functions. . . . .	338
9.3.12	Signals. . . . .	342
9.3.13	Semaphores . . . . .	345
<b>10</b>	<b>Future standardization</b> . . . . .	<b>347</b>
10.1	<i>X/Open XSH5 [UNIX98]</i> . . . . .	347

---

10.1.1	POSIX options for XSH5 .....	348
10.1.2	Mutex type .....	349
10.1.3	Set concurrency level .....	351
10.1.4	Stack guard size .....	353
10.1.5	Parallel I/O .....	354
10.1.6	Cancellation points .....	355
10.2	POSIX 1003.1j .....	356
10.2.1	Barriers .....	358
10.2.2	Read-write locks .....	358
10.2.3	Spinlocks .....	359
10.2.4	Condition variable wait clock .....	359
10.2.5	Thread abort .....	361
10.3	POSIX 1003.14 .....	361
	<i>Bibliography</i> .....	363
	<i>Thread resources on the Internet</i> .....	367
	<i>Index</i> .....	369