index

Symbols

#include <thread>, example of 13
<atomic> header, reference 390
<chrono> header 88
  reference 360
<condition_variable> header 69
  reference 375
<future> header 76
  reference 424
<mutex> header 446
  reference 446
  std::lock_guard 38
  std::mutex 38
<ratio> header 471
  reference 471
<thread> header
  introduced 13
  reference 477
  std::this_thread::get_id 31
  std::thread class 16

A

ABA problem 222
abstraction penalty 11
ACE, and multithreaded code 10
actor model 100
address space
  limit 9
  shared 5
Amdahl's law 251
  serial fraction 251
atomic integral types
  available operations 116
  return value of operations 116
atomic operation
  categories 110
  compare_exchange_strong
    bitwise comparison of a whole structure 215
    checking for unchanged value 220
    updating a whole structure atomically 217
    using to avoid looping 220
    using to extract a node from a queue 215
    using to set a value conditionally 214
    See also std::atomic class template, compare_exchange_strong() member function
  compare_exchange_weak() member function
    compared to compare_exchange_strong 195
    memory ordering in stack::push 205
    use in a loop 191
    See also std::atomic class template, compare_exchange_weak() member function
  compare-exchange functions 185
    avoiding race conditions with 185–186
    double-word-compare-and-swap (DWCAS) 117
  defined 107
  enforcing ordering with 146
    example 145
    simple example 120–121
  fences 143
    and memory_order_relaxed 143
  fetch_add() member function, and release sequences 207
  happens-before relationship 120
  implicit load 218
  inter-thread happens-before relationship, and sequenced-before 123
  lock-free, enabling by using 202

atomic operation (*continued*)
  memory ordering options 140
    default 123
    tags for memory ordering models 123
  memory ordering, default 218
  memory ordering, memory_order_seq_cst 218
  on std::shared_ptr 119, 200
    example 119
  sequentially-consistent ordering 124
  store operations 110
  synchronizes-with relationship 120
    defined 121
atomic operations 11
  enforcing ordering with 119
  memory ordering options 123
atomic types
  compare_exchange_weak() member function 185
  compare-exchange functions 113–114
    memory ordering parameters 114
    spurious failure 113
  internal locks 184
  introduction 107
  is_lock_free() member function 107
  lack of assignment 111
  lack of copy-construction 111
  list of alternative names 108
  list of typedefs 109, 118
  lock-free operations 184
  naming convention for alternative names 108
  nonmember functions
    naming convention 117
    std::atomic_compare_exchange_weak 118
    std::atomic_compare_exchange_weak_explicit 118
    std::atomic_flag_clear 119
    std::atomic_flag_clear_explicit 119
    std::atomic_flag_test_and_set 119
    std::atomic_flag_test_and_set_explicit 119
    std::atomic_is_lock_free 118
    std::atomic_load 118
    std::atomic_store 118
    std::atomic_store_explicit 118
  operation return values 110
  operations with nonmember functions 117–119
  overview of operations 109
  restrictions 109
  std::atomic class template
    and relationship with alternative names 108
    use with user-defined type 214
    using compare_exchange_strong on a structure 217
    using with a user-defined type 202
  std::atomic_address, use with hazard pointers 198
atomic types, assignment operator, return type 112
atomic variable, use as a done flag 258
ATOMIC_ADDRESS_LOCK_FREE macro 392
ATOMIC_BOOL_LOCK_FREE macros 392
ATOMIC_CHAR_LOCK_FREE macro 392
ATOMIC_CHAR16_T_LOCK_FREE macro 392
ATOMIC_CHAR32_T_LOCK_FREE macro 392
ATOMIC_FLAG_INIT macro 110
ATOMIC_INT_LOCK_FREE macro 392
ATOMIC_LLONG_LOCK_FREE macro 392
ATOMIC_LONG_LOCK_FREE macro 392
ATOMIC_SHORT_LOCK_FREE macro 392
ATOMIC_VAR_INIT macro 393
ATOMIC_WCHAR_T_LOCK_FREE macro 392
auto keyword 337
automatic type deduction
  and function template parameters 337
  deduction rules 337
  syntax 337

B

background threads 21
bit-field, using to keep structure within machine word 214
blocked thread 301
blocking 301
Boost, and multithreaded code 10–11
boost::shared_mutex 171, 173–174
bounded queue 169

C

C++ Standard
  overhaul 1
  support for multithreaded programs 1
cache ping-pong, and performance 236
callable type 16
code reviews 303
  guidelines 304
  rubber chicken 304
Communicating Sequential Processes 97
  actor model 100
  and lack of shared data 97
compare-exchange functions. *See* atomic operation, compare-exchange functions
concurrency
  and constructors 150
  and destructors 150
  and multithreading 9–12

concurrency (*continued*)
  and responsiveness 254
  and user interfaces 253
  approaches to 5
  data structure 149
  definition 2–6
  design guidelines 149–150
  efficiency 11
  enabling 150
    by separating data 160
    with fine-grained locking 164
  hardware 3, 14
  Hello World 13
  illusion of 2
  in computer systems 2–4
  limiting 157
  memory model 11, 37
  multiple processes 5
    advantage 5
    downside 5
  multiple threads 5
    advantage 6
    downside 6
  opportunity for 149
  optimization 9
  overhead 6, 8
  parallel execution 3
  performance 7–8
  performance problem
    lock granularity 46
  potential for 149, 164
  preventing 149
  problem
    contention 47
    deadlock 47
    race condition 35
  reasons for using 6
    performance 9
    separation of concerns 9
  reasons to avoid 8
  safe 5
  support in C++11 standard 10
concurrency bugs
  and shared memory 303
  broken invariants 302
  code review questions 304
  deadlock 301
  difficulty of reproducing 305
  distinguishing from other bugs 306
  lifetime issues 302
  livelock 301
  locating 303–314
  stale data 305
  symptoms 303
  testing 311–313
  unwanted blocking 301
  with code reviews 303
  wrong mutex scope 305
constant expressions 324
  and constexpr 324
  and literal types 326
  array bounds 324
  in-class initializers 324
  nontype template parameters 324
  places that require them 324
  static initialization for aggregates 324
constexpr functions 324
  and constant expressions 324
  and mutex initialization 327
  and static initialization 327
    avoiding race conditions 327
  and templates 328
  constructors 327
    and static initialization 327
    trivial constructors 328
  member functions 326
    and virtual 328
    implied const 327
  no side effects 328
  not always constant expressions 325
  pure functions 328
  requirements 328
  syntax 325
constexpr objects 327
  implicit const 327
context switching 9
  defined 3
conveniently concurrent 8

D

daemon threads 21
data parallelism 8
data race 36, 302
  defined 106
  dereferencing a loaded pointer 212
  due to unsupported usage 211
  undefined behavior 106
  with double-checked locking 61
deadlock
  avoiding 47
    breaching guidelines 175
    fixed-order locking 47, 49, 174
    no callbacks 49
    no nested locks 49
    with a lock hierarchy 51
    with std::lock 48
  defined 47
  guidelines for avoiding 49–54
  impossibility with hierarchical mutex 52

deadlock (*continued*)
  with children's toys 47
  with std::thread objects 49
defaulted functions 321
  and accessibility 321
  and aggregates 322
  and copy constructors 321
  and default constructors 322
  and default initialization 323
  and trivial functions 322
  and virtual destructors 321
  examples 321
  reasons for defaulting 321
  syntax 321
  things that can be defaulted 321
deleted functions 319
  and overload resolution 321
  example 21
  preventing copying 320
  preventing narrowing 321
  syntax 320
dividing work 225
  by task 231–233
  comparing complexity of alternatives 264
  data access patterns 240–241
  dividing data recursively 262
  early completion 258
  generalist threads vs. specialist threads 226
  guidelines 242
  matrix multiplication 240
  performance characteristics 233
  pipeline 232
  recursively 227–231
  separating concerns 231–232
  specialist threads 231
  splitting data 226–227
  std::async 228
  std::thread::hardware_concurrency 228
  threads vs. tasks 228
  to reduce data accessed in each thread 241
double-checked locking
  defined 60
  race condition 60
  undefined behavior 61
doubly-linked list
  deleting from 34
  invariant 34
dual-core 3
dummy nodes, in a lock-free queue 211
DWCAS (double-word-compare-and-swap) 117

E

embarrassingly parallel 8
Erlang 5, 97
event loops, and responsiveness 253
event-driven architecture 253
example
  basic barrier 267
  exception_safe_parallel_accumulate 248
    with std::async 249
  function to run a pending task from a thread pool 281
  GUI event loop 253
  interruptible_thread interface 289
  join_threads class 248
  naïve parallel_accumulate 29
  parallel_accumulate
    with a thread pool 279
    with std::packaged_task 245
  parallel_find 258
    using std::async 261
  parallel_for_each 255, 257
  parallel_partial_sum
    with even division 264
    with pairwise updates 269
  parallel_quick_sort using thread pool 281
  queue for work-stealing 285
  scoped_thread 27
  thread pool with work stealing 286
  thread_guard 20
  with per-thread work queues 283
examples, test for concurrent queue 312
exception safety 243–250
  achieving with std::async 249
  and destroying futures 249
  and std::packaged_task 245
  in std::stack 43
  single vs. multithreaded 243
exceptions, and std::async 83

F

false sharing
  avoiding with padding 242
  between mutex and protected data 242
  performance implications 238
fine-grained locking 158
  difficulties 159
  placing locks 161
functional programming 93
  and (lack of) race conditions 93
future, defined 76
futures
  and stored exceptions 83
  getting from a std::packaged_task 79
  getting from a std::promise 82
  getting from std::async 77
  storing exceptions 83–84
  using for thread return values 77

G

global variables 5
GUI frameworks 80
  and threads 80

H

hardware concurrency 4
  defined 3
hardware threads 3
hazard pointer
  checking for outstanding pointers 197
  defined 193
  marking a pointer as hazardous 193
  overhead from simple implementation 199
  patent 200
  reclaiming memory 198
  simple allocation scheme 195
  simple example of reclaiming nodes 197
  trading memory for speed 199
  updating 194
  usage overview 193
hierarchical mutex
  example 51
  sample implementation 52

I

impure functions, defined 93
initial function 13
interrupting threads 289–299
  and exception safety 297
  background threads 298
  basic interface 289
  basic interruptible_thread implementation 290
  catching interruptions 297
  detecting interruption 291
  example implementation 293
  handling interruptions 297
  interrupt function 290
  interruptible_wait on a
    condition_variable_any 294
  interrupting future waits 296
  interrupting waiting on a
    condition_variable 291
  interruption point 289
  interruption_point function 291
  per-thread data structure 289
  race condition in naïve implementation 292
  sample implementation 294
  using a timeout 292
invariant
  and exception safety 243
  and race conditions 36, 302
  and thread-safety 150
  broken 34, 302
  defined 34
  for a queue 162
  in lock-free data structures 183
  preserving 37, 152
iterators, and concurrency 170

J

join_threads class, use with thread pool 276
join, std::threads, example of 14
joining threads, when exceptions thrown 276

L

lambda function 329
  and local variables 330
  as predicates 333
  captures 330
    and class members 333
    by copy 331
    by reference 331
    mixing copy and reference captures 332
    this 333
  example with std::condition_variable::wait 69
  lambda introducer 329
  return values 329
    automatic deduction 329–330
    explicit declaration syntax 330
  syntax 329
  use with std::condition_variable 70
  with parameters 329
lazy initialization
  defined 59
  example of 59
  example using mutex 60
  example using std::call_once 61
lifetime issues 17
lightweight processes 5
literal type
  and constant expressions 326
  and constexpr functions 326
  and static initialization 327
  defined 325
  example 325
live lock, defined 183
local objects, destruction of 20
lock-based data structures
  bounded queue 169
    vs. unbounded queue 169
  list 175–179
    find_first_if() 178
    for_each() 178
    interface design 175

lock-based data structures (*continued*)
    iteration 175
    operations 175
    opportunities for concurrency 178
    push_front() 177
    remove_if() 178
    sample implementation 176
  lookup table 169–174
    consequences of hash table implementation 171
    consequences of sorted array implementation 171
    exception safety 173
    extracting contents 174
    fine grained locking 171
    hash function 171
    implementation choices 171
    interface design 170
    operations 170
    protecting buckets 173
  queue 154–169
    analysis of required locks 163
    and exception safety 164
    avoiding data races 163
    exception safety and waiting 155
    fine-grained locks 158–159
    implementation using std::shared_ptr 156
    implementation with fine-grained locking and waiting 166–168
    interface differences from std::queue 154
    invariants 162
    potential for concurrency 164
    sample implementation with fine-grained locking 161
    simple implementation 154
    waiting 165
    waiting for an entry 155
  stack 151–154
    and deadlock 153
    and race conditions 152
    basic thread safety 152
    exception safety 152
    potential for concurrency 153
    sample code 151
    serialization 153
  unbounded queue 169
    vs. bounded queue 169
lock-free data structures
  ABA problem 222
  and compare-exchange 182
  and invariants 183
  and memory ordering constraints 183
  atomic operations 214
  busy-wait 217
  defined 182
  guidelines for writing 221–223
  helping stalled threads 217
  impossibility of deadlocks 183
  introduced 181
  lock-free property, ensuring by helping other thread 222
  managing memory 188–204, 221
    alternatives to new and delete 220
    freeing memory at quiescent points 191
    hazard pointers 193–200
    memory reclamation schemes 221
    recycling nodes 222
    split reference count, primary reason 204
    split reference counts 201
  maximizing concurrency 183
  optimizing memory allocation 220
  performance 183, 217
  prototyping 221
  queue 209–221
    avoiding race conditions 212
    handling multiple producers 211–221
    implementation of pop() with a ref-counted tail 214
    obtaining a new reference to a node 216
    pop() implementation for lock-free push() 218
    push() implementation using helping for lock-free semantics 219
    releasing a reference on a node 215
    releasing an external counter to a node 216
    sample implementation of push() with ref-counted tail 212
    single producer, single consumer 211
    single-producer, single-consumer implementation 209
    synchronizing push() and pop() 210
  split-count reference counting, deleting nodes 215
  stack 184
    adding a node 184
    basic push function 185
    basic structure 184
    example counting threads in pop 189
    example implementation with memory leak 187
    exception safety 187
    implementation of pop using hazard pointers 194
    implementation with reference counting and relaxed operations 207
    popping a node with split reference counts 203
    potential race conditions 185
    pushing a node with split reference counts 201

lock-free data structures (*continued*)
removing nodes 186
sample implementation using shared_ptr 200
starvation 182
std::atomic_is_lock_free function, std 200
unintended locking 217
lock-free programming, defined 37
locking
granularity 46
choosing 57–59
coarse-grained 57
defined 57
fine-grained 47, 57, 175
too large 46
too small 46
hand over hand 50, 175, 178
multiple mutexes, std::lock 48–49
lookup table, interface design
choices for add and change 170
choices for querying values 170
lvalue references
and rvalue references 316
and temporaries 316

M

massively parallel 235
memory allocation, moving outside a lock 157
memory barriers 143
memory location
and bit-fields 104
zero-length 104
and object 104
and race conditions 105
defined 104
example struct 105
memory model 10
acquire-release ordering 132
and happens-before 136
and inter-thread happens-before 136
and mutexes 138
and synchronizes-with 132
and synchronizing data between threads 136
carries-a-dependency-to relation defined 139
choice of semantics for read-modify-write operations 138
data dependency 138–140
dependency-ordered-before
and inter-thread-happens-before 139
compared to synchronizes-with 139
relation defined 139
example of data dependent ordering 139
example of transitive synchronization 136
example sequence 134
example with imposed ordering 134
example with no total order 133
lack of synchronization with relaxed operations 135
man in cubicle analogy 135
memory_order_acq_rel 132, 137–138
memory_order_acquire 132, 135, 138
memory_order_consume 138–139
memory_order_release 132, 135, 138
mixing with sequentially-consistent ordering 138
synchronization cost 138
synchronization using acquire and release pairings 135
transitive synchronization
across threads 136
using read-modify-write operations 137
applying to lock-free stack 205–209
applying, identifying required relationships 205
fences
example of ordering relaxed operations 143
memory_order_acquire 144
memory_order_release 144
synchronizes-with 144
happens-before relationship 120
and relaxed ordering 127
and sequenced-before 146
between push() and pop() on a queue 211
between threads 123
defined 122
in a single-thread 122
nonatomic operations 146
inter-thread happens-before relationship
and synchronizes-with 123
defined 123
lock() example memory ordering 146
memory ordering 204
memory ordering options
and ordering models 123
and varying costs with CPU architecture 123
default 123
default ordering 124
modification order
and relaxed ordering 127
defined 106
non-sequentially-consistent orderings
and lack of agreement between threads 127
and lack of global order 126
optimizing memory ordering 221
ordering constraints 184
relaxed ordering 127–132
example 127
example sequence 128
man-in-cubicle analogy 131
memory_order_relaxed 127, 132

memory model (*continued*)
 release sequence
  and memory_order_relaxed 141
  defined 141
  example 141
  example sequence 143
  in lock-free stack 206
  memory_order_acq_rel 141
  memory_order_acquire 141
  memory_order_consume 141
  memory_order_release 141
  memory_order_seq_cst 141
 sequenced-before relationship, and happens-before 146
 sequential consistency, defined 124
 sequentially-consistent ordering 204
  additional implied ordering relationships 126
  and global order 124
  and happens-before 126
  and mutexes 138
  and synchronizes-with 124
  cost 204
  example 125
  memory_order_seq_cst 125–126
  mixing with acquire-release ordering 138
  ordering diagram 126
  performance penalty 124
  synchronization cost 126
  using for prototyping 221
 splitting operations to vary memory ordering 207
 synchronizes-with relationship 120
  and relaxed ordering 127
  defined 121
 unlock() example memory ordering 146
message passing
 actor model 100
 and state machines 97
 ATM example 98
  actor model 100
  division into threads 98
  example state function 101
  handling different types of message 101
  handling messages 100
  messages as structs 98
  sample code 99
  state machine model 99
  waiting for matching messages 100
  waiting for messages 100
 implicit synchronization 100
 no shared state 100
move constructors
 and copy constructors 317
 example 317
move semantics 316
 and deleted copy operations 320
 and optimization 316
 and rvalue references 317
 and rvalues 316
 and std::thread 318
 and std::unique_ptr 317
 move constructors, example 317
 moved-from state 318
  of std::thread 318
 move-only types 320
  example 320
  in the thread library 318
 moving from an lvalue 320
multicore 3, 7
multiprocessor 3
multithreaded 5
multithreading
 history 10
 memory model 10
 support in the C++11 standard 14
mutex
 defined 37
 lock 37
 lock ownership, transferring 55
 recursive 65
 unlock 37
 user-defined
  example 52
  example implementation 111, 181
  requirements 52
 using 38–59
mutual exclusion 149

N

native_handle 12
naturally parallel 8
nonblocking operations, defined 182

O

object
 and variables 105
 defined 104
 division into subobjects 104–105
 example struct 105
 subobject 104
oversubscription, avoiding 30

P

parallel_find, and out-of-order processing 262
performance 233–239
 Amdahl's law 251
 and cache line pressure 238

performance (*continued*)
  and data structures 239–243
  and multiple processors 236
  and mutexes 236
  and reader-writer mutexes 237
  and task switching 238
  cache miss 238, 241
  cache ping-pong 236–237
  data contention 235
  data proximity 238
  false sharing 237
  hiding latency 252
  high contention 236
  I/O latency 252
  idle threads 239
  low contention 236
  massively parallel 235
  multiple applications 234
  number of processors 234
  oversubscription 234, 239
  reducing time taken vs. processing more data 251
  scalability 250
  scaling 234
  std::async 234
  std::thread::hardware_concurrency 234
  thread pools 234
  too many threads 239
platform-specific facilities 6, 10, 12
pure function, defined 93

Q

queue
  bounded queue 169
  single-threaded implementation 158
    with dummy node 160
  unbounded 169
Quicksort
  example parallel implementation 228
  FP-style sequential implementation 94
  simple parallel implementation 95

R

race condition 35–37
  and exception safety 43
  avoiding
    by atomic update of a whole structure 217
    with atomic compare-exchange functions 185
    with enforced ordering 105
  benign 36
  data race 36
    defined 106
  defined 36
  eliminating 43
  example 59
  in interface 40, 46
    stack example 42
  initialization of local static 62
  problematic 36
  window of opportunity 36
  with double-checked locking 60
race conditions 302
  and testing 305
RAII. *See* Resource Acquisition Is Initialization
recursive mutex 65
reduction 226
reference to a reference 319
Resource Acquisition Is Initialization 10
  and threads 20, 27
  for mutexes, std::lock_guard 38
  scoped_thread example 27
  thread_guard example 20
responsiveness 7
rvalue references
  and lvalue references 316
  and move semantics 316
  and static_cast 318
  and std::move 318
  and template argument type deduction 319
  and template parameters 318
  and temporaries 316
  described 315
  parameters treated as lvalues 318

S

scalability 250
  Amdahl's law 251
  parallel vs. serial sections 251
scoped_thread example 27
separation of concerns, and concurrency 6
serialization 60, 149
  minimizing 150
shared data
  encapsulating 38
  failure to protect 39–40
    example 39
  problems 34–36
  protecting 37–65
    guideline 40
    incomplete protection 40
    initialization 59
    single global mutex 46
    structuring code 39
    with correct mutex 40
    with mutex 43

shared data (*continued*)
with std::call_once 61
with std::mutex 38
read-only 34
shared memory 5
SI ratios 88
single-core 3
Single-Instruction/Multiple-Data (SIMD) 267
single-threaded 4
sizeof..., and variadic templates 336
Software Transactional Memory, defined 37
spurious wake, defined 71
std::adopt_lock
and std::lock_guard 48
example 48
std::any_of 257
std::async
and asynchronous tasks 77
and dividing work recursively 227
and exceptions 83, 249
and oversubscription 96
and parallel_for_each 257
and std::future 77, 425, 445
compare to std::thread 77
comparison with std::packaged_task 79
example of passing arguments 77
exception propagation 262
exception safety 262
introduction 77
passing additional arguments 77
passing reference arguments with std::ref 262
reference 445
simple example 77
unsuitability for interdependent tasks 266
use for exception safety 249
vs. custom spawn_task function 96
vs. simple thread pools 276
vs. thread pools 280
std::atomic class template 116
and relationship with alternative names 108
and user-defined types 110, 116–117
available operations 117
bitwise comparison and compare/exchange functions 116
class definition 398
compare_exchange_strong() member function 404
compare_exchange_weak() member function 406
conversion assignment operator 401
conversion constructor 401
conversion to nonatomic type 403
default constructor 400
exchange() member function 404
fetch_add() member function 412
for pointer specializations 421
fetch_and() member function 414
fetch_or() member function 415
fetch_sub() member function 413
for pointer specializations 422
fetch_xor() member function 416
is_lock_free() member function 401
load() member function 402
operator-- postdecrement 417
for pointer specializations 423
operator-- predecrement 417
for pointer specializations 423
operator^= 418
operator&= 418
operator++ postincrement 417
for pointer specializations 423
operator++ preincrement 417
for pointer specializations 423
operator+= 417
for pointer specializations 423
operator-= 418
for pointer specializations 424
operator|= 418
reference 397
restricted operations 110
specializations 408
for built-in types 107
for integral types 408
for pointer types 418
See also std::atomic partial specialization for pointer types
std::atomic_compare_exchange_strong_explicit() nonmember function 406
std::atomic_compare_exchange_strong() nonmember function 405
std::atomic_compare_exchange_weak_explicit() nonmember function 407
std::atomic_compare_exchange_weak() nonmember function 407
std::atomic_exchange nonmember() function 404
std::atomic_exchange_explicit() nonmember function 404
std::atomic_fetch_add_explicit() nonmember function 413
for pointer specializations 422
std::atomic_fetch_add() nonmember function 413
for pointer specializations 421
std::atomic_fetch_and_explicit() nonmember function 415
std::atomic_fetch_and() nonmember function 414

std::atomic class template (*continued*)
 std::atomic_fetch_or_explicit() nonmember function 416
 std::atomic_fetch_or() nonmember function 415
 std::atomic_fetch_sub_explicit() nonmember function 414
  for pointer specializations 422
 std::atomic_fetch_sub() nonmember function 413
 std::atomic_fetch_sub() nonmember function for pointer specializations 422
 std::atomic_fetch_xor_explicit() nonmember function 416
 std::atomic_fetch_xor() nonmember function 416
 std::atomic_init() nonmember function 400
 std::atomic_is_lock_free() nonmember function 401
 std::atomic_load_explicit() nonmember function 402
 std::atomic_load() nonmember function 402
 std::atomic_store_explicit() nonmember function 403
 std::atomic_store() nonmember function 403
 store() member function 403
std::atomic partial specialization for pointer types 114–115
 arithmetic operators 115
  memory ordering 115
  return value 115
 fetch_add() member function 115
  example 115
  memory ordering choices 115
  return value 115
 fetch_sub() member function 115
  memory ordering choices 115
  return value 115
std::atomic_flag 110–112, 184
 and ATOMIC_FLAG_INIT 110
 and std::atomic_flag_test_and_set nonmember function 396
 class definition 395
 clear() member function 107, 111, 396
 default constructor 395
 initialization 110
 initialization with ATOMIC_FLAG_INIT 395
 introduction 107
 nonmember functions, std 119
 states 110
 std::atomic_flag_clear_explicit() nonmember function 397
 std::atomic_flag_clear() nonmember function 397
 std::atomic_flag_test_and_set_explicit() nonmember function 396
 test_and_set() member function 107, 111, 396
std::atomic_signal_fence() nonmember function 394
std::atomic_thread_fence() nonmember function 394
std::atomic_xxx typedefs 391
std::atomic<bool> 112–114
 assignment from bool 112
  return type 112
 compare_exchange_strong() member function 113
 compare_exchange_weak() member function 113
  loop to handle spurious failures 113
 comparison with std::atomic_flag 112
 exchange() member function 112
 is_lock_free() member function 114
 load() member function 112
 store() member function 112
std::atomic<double>, and compare_exchange_strong 117
std::atomic<float>
 and compare_exchange_strong 117
 lack of arithmetic operations 117
std::call_once
 and std::once_flag 471
 example 61
 introduced 61
 using local static as alternative 62
std::chrono::duration 88
 and duration-base timeouts 89
 count() member function 89
std::chrono::duration class template
 and std::recursive_timed_mutex::try_lock_for 457
 and std::shared_future::wait_for 433
 and std::this_thread::sleep_for 486
 and std::timed_mutex::try_lock_for 453
 and std::unique_lock 463
 and std::unique_lock::try_lock_for 467
 class definition 361
 converting constructor from a count 362
 converting constructor from another duration 363
 count() member function 363
 default constructor 362, 369
 duration_cast() nonmember function 368
 max() static member function 366, 371
 min() static member function 366, 371
 operator 367–368
 operator- 364
 operator– postdecrement 364
 operator– predecrement 364

std::chrono::duration class template (*continued*)
  operator!= 367
  operator*= 365
  operator/= 365
  operator%= 365–366
  operator+ 363
  operator++ postincrement 364
  operator++ preincrement 364
  operator+= 365, 370
  operator-= 365, 370
  operator== 366
  operator> 367
  operator>= 368
  period member 362
  reference 360
  rep member 362
  time_since_epoch() member function 370
  use in std::chrono::system_clock 372
  use with std::future::wait_for 428
  zero() static member function 366
std::chrono::duration_cast 368
std::chrono::high_resolution_clock 88
  definition 374
  potential relationship with std::chrono::steady_clock 374
  potential relationship with std::chrono::system_clock 374
  reference 374
std::chrono::steady_clock 88
  class definition 373
  duration member typedef 374
  now() static member function 374
  period member typedef 373
  reference 373
  rep member 373
  time_point member typedef 374
std::chrono::system_clock 87
  class definition 371
  reference 371
  time_t
    conversion from time_t to time_point 373
    conversion to time_t from time_point 372
  to_time_point() static member function 90, 373
  to_time_t() member function 372
std::chrono::system_clock::duration member, definition 372
std::chrono::system_clock::now member, definition 372
std::chrono::system_clock::period member, definition 372
std::chrono::system_clock::rep member, definition 372
std::chrono::system_clock::time_point member, definition 372
std::chrono::time_point
  and absolute timeouts 90
  clocks, adjustment during absolute-time-based waits 90
  introduction 89
  time_since_epoch() member function 90
std::chrono::time_point class template
  and std::future::wait_until 428
  and std::recursive_timed_mutex::try_lock_until 458
  and std::shared_future::wait_until 433
  and std::this_thread::sleep_until 486
  and std::timed_mutex::try_lock_until 454
  and std::unique_lock 463
  and std::unique_lock::try_lock_until 467
  class definition 369
  reference 369
  use in std::chrono::system_clock 372
std::condition_variable 69
  and std::notify_all_at_thread_exit 382
  and timeouts when waiting 92
  class definition 375
  default constructor 376
  destructor 376
  example 69
  notify_all() member function 75, 377
    and std::notify_all_at_thread_exit 382
    relationship to wait 75
    vs. notify_one 75
  notify_one() member function 70, 376
    calling outside a lock 165
    relationship to wait 70
    relationship with notify_all() 377
    relationship with wait_for() 377
    relationship with wait_until() 377
    relationship with wait() 377
    use in a thread-safe queue 165
    use with a concurrent queue 156
    vs. notify_all 75
  reference 375
  wait_for() member function 92, 378
    and std::chrono::duration class template 378
    and std::cv_status enum 378
    and std::notify_all_at_thread_exit() 382
    and std::unique_lock class template 378
    with a predicate 379
  wait_until() member function 92, 380
    and std::chrono::time_point class template 380
    and std::notify_all_at_thread_exit() 382
    and std::unique_lock class template 380
    with a predicate 381
  wait() member function 70, 377
    and callable objects 70
    and condition-checking functions 70

std::condition_variable (*continued*)
and functions 70
and spurious wakes 71
and std::notify_all_at_thread_exit 382
and std::unique_lock 70
choosing a predicate 165
relationship to notify_all 75
relationship to notify_one 70
use in a thread-safe queue 165
use with a concurrent queue 155
with a predicate 378
waiting with a timeout
example 90
wait_for vs. wait_until 91
waking all waiting threads 75
waking one waiting thread 75
std::condition_variable_any 69
and timeouts when waiting 92
default constructor 384
destructor 384
notify_all() member function 385
notify_one() member function 384
try_lock_for() member function 92
wait_for() member function 92, 386
with a predicate 387
wait_until() member function 92, 388
with a predicate 389
wait() member function 385
with a predicate 386
std::copy_exception 84
std::count 257
std::current_exception 84
std::cv_status 92
std::equal 257
std::find, parallelizing 257
std::for_each, parallelizing 255
std::function, use with thread pool 276
std::future 76
and 262
and exception safety 246, 249
and propagating exceptions 256
and thread pool tasks 278
and threads from std::async calls 250
and timeouts when waiting 92
and unique ownership 85
conversion to std::shared_future 86
example 77
obtaining from std::packaged_task 81
propagating exceptions 247
propagating exceptions from get() 247
share() member function 86
and automatic deduction of variable type 86
vs. std::shared_future 85
wait_for() member function 92
wait_until() member function 92
std::future class template
and std::async 445
and std::packaged_task 438
and std::promise 442
class definition 425
conversion to std::shared_future 427, 431
default constructor 425
destructor 426
get() member function 429
move assignment operator 426
move constructor 426
reference 425
share() member function 427
valid() member function 427
wait_for() member function 428
and std::chrono::duration 428
wait_until() member function 428
and std::chrono::time_point 428
wait() member function 427
std::future_errc::broken_promise 84
std::future_error, introduction 84
std::future_status 89, 92
std::future<void>, as a done flag 256
std::launch::async, using to guarantee separate threads 313
std::lock
and exceptions 48
and std::unique_lock 55
avoiding deadlock with 48–49
example 48
std::lock function template, reference 469
std::lock_guard
adopting a lock 48
and user-defined types 52
comparison with std::unique_lock 55
example 38
std::lock_guard class template
class definition 459
destructor 460
lock-adopting constructor 459
locking constructor 459
reference 459
std::make_shared, use for allocating a reference-counted object 153
std::memory_order enumeration 393
std::memory_order_acq_rel, reference 394
std::memory_order_acquire 205
reference 393
std::memory_order_consume, reference 394
std::memory_order_relaxed 205–206
reference 393
std::memory_order_release 205
reference 393
std::memory_order_seq_cst, reference 394

std::move 25
  and std::thread 26
std::mutex
  class definition 447
  default constructor 447
  example 38
  lock() member function 448
  locking 38
  locking with
    std::lock 48
    std::lock_guard 38
  reference 447
  try_lock() member function 448
  unlock() member function 449
  unlocking 38
  using 38
std::nested_exception, use for accumulating multiple exceptions 247
std::notify_all_at_thread_exit, and std::condition_variable::notify_all() 382
std::once_flag
  default constructor 470
  introduced 61
  reference 470
  std::call_once() nonmember function template 471
std::packaged_task 78
  and exception safety 245
  and std::future 425
  as a callable object 79
  comparison with std::async 79
  compatible callable types 79
  get_future() member function 79
  introduction 79
  obtaining a future 81
  obtaining a std::future 79
  passing tasks between threads 80
  template parameter 79
  wrapping tasks for a thread pool 278
std::packaged_task class template
  and std::future 438
  class definition 434
  construction from a callable object 435
    with an allocator 436
  default constructor 435
  destructor 437
  function call operator() 438
  get_future() member function 438
  make_ready_at_thread_exit() member function 439
  move assignment operator 437
  move constructor 436
  reference 434
  reset() member function 438
  swap() member function 437
  valid() member function 438
std::partial_sum
  alternative approaches 263
  parallelizing 263–271
std::promise 79
  and obtaining a result from multiple threads 258
  and std::future 425
  and stored exceptions 84
  example 82
  get_future() member function 82
  obtaining a future 82
  set_exception() member function 84
  set_value() member function 82, 84
  using for test scheduling 311
std::promise class template
  and std::future 442
  class definition 440
  construction with an allocator 441
  default constructor 440
  destructor 442
  get_future() member function 442
  move constructor 441
  move-assignment operator 441
  reference 440
  set_exception_at_thread_exit() member function 444
  set_exception() member function 444
  set_value_at_thread_exit() member function 443
  set_value() member function 443
  swap() member function 442
std::queue
  and thread safety 71
  interface 71
    potential for race conditions 72
std::ratio class template
  class definition 472
  reference 472
  use in std::chrono::system_clock 372
  use with std::chrono::duration 360
std::ratio_add, reference 473
std::ratio_divide, reference 474
std::ratio_equal, reference 475
std::ratio_greater_equal, reference 477
std::ratio_greater, reference 476
std::ratio_less_equal, reference 476
std::ratio_less, reference 475
std::ratio_multiply, reference 474
std::ratio_not_equal, reference 475
std::ratio_subtract, reference 473
std::recursive_mutex 65
  class definition 449
  default constructor 450

std::recursive_mutex (*continued*)
  destructor 450
  introduced 48
  lock() member function 450
  reference 449
  try_lock() member function 450
  unlock() member function 451
std::recursive_timed_mutex
  and timeouts when locking 92
  class definition 455
  default constructor 456
  destructor 456
  lock() member function 456
  locking operations with timeouts 91
  reference 455
  try_lock_for() member function 92, 457
    and std::chrono::duration 457
  try_lock_until() member function 92
    and std::chrono::time_point 458
  try_lock() member function 456
  unlock() member function 458
std::ref 24
  passing reference arguments to std::async 262
std::result_of, getting task return types 278
std::shared_future 76
  and shared ownership 85
  and timeouts when waiting 92
  conversion from std::future 427
  separate copy per thread 85–86
  spreadsheet example 85
  using for test scheduling 311
  vs. std::future 85
  wait_for() member function 92
  wait_until() member function 92
std::shared_future class template
  class definition 430
  construction from std::future 430–431
  copy constructor 431
  default constructor 430
  destructor 432
  get() member function 434
  move constructor 431
  reference 429
  valid() member function 432
  wait_for() member function 432
  wait_until() member function 433
    and std::chrono::duration 433
    and std::chrono::time_point 433
  wait() member function 432
std::stack, interface 41
std::terminate 17
  and std::thread destructor 17
std::this_thread::get_id 31
std::this_thread::get_id() nonmember function 485
  and std::thread::id 485
std::this_thread::sleep_for 68
  described 91
  example 68
std::this_thread::sleep_for() nonmember function 486
  and std::chrono, duration 486
std::this_thread::sleep_until 91
std::this_thread::sleep_until() nonmember function 486
  and std::chrono::time_point 486
std::this_thread::yield 276
  use in spin-wait 268
std::this_thread::yield() nonmember function 486
std::thread 16
  and standard containers 28
  and std::vector 28
  as function return value 26
  class definition 477
  constructing 16
  constructor 24, 482
    and std::move 25
    moving arguments 25
  constructor accepting multiple arguments 23
  default constructor 481
  destructor 17, 482
    and joinable 483
    and std::terminate 17
  detach() member function 18, 21, 484
    relationship with joinable() 21
  get_id() member function 31, 485
  hardware_concurrency() static member function 28, 30, 485
  join() member function 18–19, 21, 484
    and interaction with joinable() 19
    example of 14
  joinable() member function 19, 21, 484
    and interaction with join() 19
    relationship with detach() 21
  move assignment 26
  move construction 26
  move constructor 482
  move semantics 25
  move-assignment 483
  moving 25
    from temporaries 26
  native_handle_type 481
  native_handle() member function 481
  pass by value 27
  reference 477
  std::swap() nonmember function 484
  std::thread::id 478
    and std::this_thread::get_id() 485

std::thread (*continued*)
    class definition 478
    default constructor 479
    operator 479–481
    operator!= 479
    operator== 479
    operator> 480
    operator>= 480
  swap() member function 483
  variadic constructor 23
std::thread::hardware_concurrency 255
  and choosing the number of threads 228
  and thread management 230
  use with thread pools 274
std::thread::id 31
  comparison operators 31
  specialization of std::hash 31
  stream insertion operator 32
  total order 31
std::timed_mutex
  and timeouts when locking 92
  class definition 452
  default constructor 452
  destructor 452
  lock() member function 452
  reference 451
  support for locking operations with a timeout 91
  try_lock_for() member function 453
    and std::chrono::duration 453
  try_lock_until() member function 92, 454
    and std::chrono::time_point 454
  try_lock() member function 453
  unlock() member function 454
std::try_lock() function template, reference 469
std::unique_lock
  and std::lock 55
  and std::move 55
  and TimedLockable types 92
  comparison with std::lock_guard 55
  constructors that accept timeouts 92
  deferred locking 54
  example of releasing lock early 71
  introduced 54
  moving 55
  owns_lock member() function
    detecting timeout on lock acquisition 92
  returning from a function 56
  try_lock_for() member function 92
  try_lock_until() member function 72, 75, 84, 91–92, 100
  unlocking 56
  using with std::condition_variable 70
std::unique_lock class template
  and std::condition_variable::wait 377
  and std::condition_variable::wait() with a predicate 378
  and std::notify_all_at_thread_exit() 382
  bool conversion operator 468
  class definition 460
  default constructor 461
  deferred-lock constructor 462
  destructor 465
  lock() member function 465
  lock-adopting constructor 462
  locking constructor 461
  move constructor 464
  move-assignment operator 464
  mutex() member function 468
  owns_lock() member function 377–378, 468
    and std::notify_all_at_thread_exit() 382
  reference 460
  release() member function 468
  std::swap() nonmember function 465
  swap() member function 465
  try_lock_for() member function 467
    and std::chrono::duration 467
  try_lock_until() member function 467
    and std::chrono::time_point 467
  try_lock() member function 466
  try-to-lock constructor 462
    with a duration timeout 463
    with a time_point timeout 463
  unlock() member function 466
std::unique_ptr, as example of move semantics 317
std::vector, of std::thread 28
synchronization mechanisms, barrier 267

T

task parallelism 7
task switching 2
  defined 3
  example of 4
  illusion of concurrency 2
templates, variadic 333
testing concurrent code 305
  adjusting the thread count 306
  architecture variations 309
  avoid deadlock in test code 313
  boilerplate test structure 313
  brute force testing 308
  combination simulation testing 309
  controlling thread execution order 310
  controlling thread scheduling 312
  designing for testability 307
  detecting potential deadlocks 310

testing concurrent code (*continued*)
  eliminate concurrency 307
  environmental considerations 307
  false confidence 309
  general guidelines 307
  general outline 311
  general setup 311
  levels of confidence 309
  library functions with internal state 308
  memory ordering issues 309
  on single- vs. multicore systems 306
  performance testing 314
  scalability 314
  techniques 308–310
  test granularity 305
  test structure 306, 311
  testing on single-processor systems 309
  testing with a debug library 310
  thread-specific setup 311
  unpredictable scheduling 305
  verifying a bug is concurrency related 306
  vs. testing single-threaded code 305
thread
  defined 5
  detached, example 22
  fire and forget 21
  function 16
  hardware 3
  launching 16
  running in the background 21
  waiting for 19
    and RAII idiom 20
thread function
  passing arguments 23
    avoiding undefined behavior 23
    by reference 24
    moving 25
    type of arguments 23
  using a member function 24
  using std::ref for arguments 24
thread pool 9, 230
  and exception safety 280
  cache locality 286
  choosing a task size 280
  contention 286
  defined 274
  dynamic sizing 288
  fixed size 274
  potential for deadlock 280
  queue contention 283
  running tasks while waiting 281
  scalability 279
  simple example 275
  stealing tasks from other worker threads 288
  task size 279
  tasks vs. threads 279
  uneven work distribution 284
  vs. std::async 280
  waiting for tasks with std::future 278
  work queue 276
  work stealing 284
  worker threads 274
  wrapping tasks in std::packaged_task 279
thread_guard example 20
thread_local keyword
  example 53
  example usage 196
  reference 338
  use for interrupting threads 289
  use for per-thread work queues 283
threads
  dividing work 29
  hardware, number of 28
  number of, choosing 30
thread-safe queue
  clocks, standard-provided clocks 88
  example interface 72
  initial sample implementation 73
  using condition variables
    push() member function 73
    wait_and_pop() member function 73
thread-safe stack, example 45
thread-safe, defined 149
time facilities 87–92
  <chrono> header 88
  clocks 87
    adjustment during duration-based waits 89
    is_steady member 88
    now() member function 87
    period member 88
    shared epochs 89
    std::chrono::high_resolution_clock 88
    std::chrono::steady_clock 88
    std::chrono::system_clock 87
    steady clocks 88
    tick period 88
    time_point member 87
  durations 88
    duration arithmetic 89
    duration-based waits 89
    explicit conversions 89
    implicit conversions 89
    predefined typedefs 88
    std::chrono::duration 88
  steady clocks 88
  table of functions which accept timeouts 92
  time points
    and absolute timeouts 90
    epoch 89
    representation 89

time_t
  conversion from
    std::chrono::system_clock::time_point 372
  conversion to
    std::chrono::system_clock::time_point 373
timeouts
  absolute 87
  duration-based 87
  function suffixes 87

U

undefined behavior 17, 36
  access after destruction 17
  from dangling pointers 23
  from data race 106
  recursive locking of nonrecursive mutex 64

V

variadic templates 333
  and partial specialization, pattern matching 334
  and std::packaged_task 333
  and std::thread 333
  pack expansion 334
    and function parameters 336
    example 335
    expanding a pattern 335
    syntax 334
  parameter packs 334
    expanding 334
    function parameters 335
  sizeof... 336
  syntax 333

W

wait-free data structures
  and robustness 183
  defined 182
waiting for an event
  by polling 68
  by sleeping and polling 68
  multiple waiting threads 75
  one-off events 76
  with condition variables 69
  with futures 76
worker threads, introduced 274