

Index

A

AAs. *See* **Architectural Artifacts**

abort(), 128

abstract classes, 335

abstraction (C++), 86

Accelerated Graphics Port (AGP), 23, 32

acceptance testing, 380

«access», 350

access policy implementation, with read-write locks, 259–261

access specifiers (visibility property), 343

access synchronization, 244

ACM CCS (Association of Computing Machinery-Computing Classification System), 284, 285, 286, 290

actions

entry/exit, 367, 368

objects and, 357

call, 357

create, 357

destroy, 357

return, 357

send, 357

transition and, 369

active class, 359

active objects, diagramming, 359–360

activity diagrams, 62, 64, 407

adaptors, 353

[addOnly]

association property, 352

attribute property, 340

ADDR, 116

address space

processes and, 152

threads and, 152

virtual, 110, 111

adornments, 350, 351, 358

after-the-fact exception handling, 381

agent model of pipeline, 278–282. *See also* **text files filtering project**

agents, 290–291

autonomous/semiautonomous, 291

definitions of, 290–291

controversy in, 290

FIFA, 291

five-part, 291

interactional, 291

knowbot, 290

multiagent architectures, 291–293, 328

rational, 299–300

situated, 291

smart object, 290

society of

structured, 291

whole more than sum in, 291

softbot, 290

software broker, 290

agile model, 39

AGP (Accelerated Graphics Port), 23, 32

AMD multicore Opteron. *See* **Opteron**

anonymous instances of classes, 346

Answer (global variable), 231, 237, 244, 257

APIs (Application Programming Interfaces), 68, 75. *See also* **IPC**

operating system, 69, 70, 74, 75, 76, 86, 93, 94, 210, 282

POSIX, 70, 75, 76

interface classes for, 86–93

layer 2 PADL, 288, 289, 290

application architecture selection (layer 5-PADL), 288, 290–300

concurrency flexibility of, 300

application design, 283–329

differing notions of, 284–286

for massive multicore processors, 284–287

PADL model, 287–326, 328–329. *See also* PADL model

PBS approach, 287, 326–328, 329, 375

‘story’ of, 323, 326, 329

Application Programming Interfaces. *See* **APIs Approach, of Grady Booch**, 332

architectural approach

in modeling concurrent systems, 331, 371–372

to parallelism, 287. *See also* **application design**

Architectural Artifacts (AAs), 41, 44, 45

declarative models and, 45

defined, 45

WBS v., 45

architectures

- multicore, 2–3
 - CBE, 3, 17, 28–31
 - challenges in, 64
 - comparison of, 21
 - Core 2 Duo, 21, 31–33
 - designs for, 19–34
 - hybrid, 3–4
 - Opteron dual core, 21–25
 - software redesigning for, 1
 - trend in, 3, 14, 36, 45, 65, 86, 283
 - UltraSparc T1, 25–28
- single core processor, 5–15
 - application development, 1
 - multicore programming and, 15–17
- of system, 371
 - definition (Booch, et. al.), 371
 - deployment view, 371
 - design view, 371
 - implementation view, 371
 - process view, 371
 - use cases, 371
- arguments, passed to threads, 163–165**
- asio class library, 305**
- assertion class, 280**
- assignment operator, 343, 344**
- association (stereotype), 358**
- Association of Computing Machinery-Computing Classification System (ACM CCS), 284, 285, 286, 290**
- associations (between objects), 333, 349**
 - bi-directional, 349
 - binary, 349
 - constraints, 352
 - link, 357
 - multiplicity in, 352
 - names for, 352
 - n-ary, 349
 - navigation, 351
 - properties, 352
 - [addOnly], 352
 - [changeable], 352
 - [frozen], 352
 - role in, 352
 - type, 352
 - unidirectional, 349, 352
- asynchronous cancellation, 173**
- asynchronous processes, 133–135**
- atomic functions, 192**
- attention layer, 324**
- attribute object**
 - condition variable, 264–265
 - mutex, 254–256
 - pthread, 167–171
 - pthread_cond_t, 264–265

- pthread_mutex_t, 254–256
- pthread_rwlock_t, 258–259
- thread, 154, 155

attributes (of classes), 337. See also specific attributes

- ordering, 343–345
 - by access, 343
 - by category, 343, 344
 - by category names, 344, 345
- properties, 339, 340, 342
 - [addOnly], 340
 - [changeable], 340
 - [frozen], 340
- visibility of, 343
- visualizing, 336–339

attrp parameter, 100–102

Austin Common Standards Revision Group, 63.

See also Single Unix Specification Version 3

auto parallelization option, 6, 33

automated painters. See painters

autonomous/semiautonomous agents, 291

B

Back Side Bus (BSB), 14

base class, 349

«become», 350

behavior part, of collaboration, 356

behavioral perspective (modeling concurrent systems), 331, 357–370

bi-directional association, 349

binary associations, 349

binary semaphores, 248

«bind», 350

binding, 348

- explicit, 348
- implicit, 348

Blackboard(s), 293–294

- architecture, logical layout, 298–299
- as critical section, 317–318
- defined, 293–294
- formal usage, 290
- as iterative shared solution space, 298–299
- memory configurations for, 295
- ‘question and answer browser’ program. See ‘question and answer browser’ program
- solution spaces, 295, 297, 298–299, 324
- structuring approaches for, 294–299

Blackboard model, 293–300, 328

- control strategies in, 324–326
- problem solvers in, 293, 294, 295
- with threads, 420–421

Blackboard Systems (Englemore & Morgan), 299

block event, 366

blocked state. *See* **waiting state**

Booch, Grady, 371, 372

Approach of, 332

Boost C++ Libraries, 316. *See also* **C++0x standard**

mutex class, implementation of (listing 8–2),
312–316

thread class, implementation of (listing 8–1),
305–312

boss thread, 161, 269, 271, 412, 413, 414

boss/worker approaches. *See* **delegation model**

bottom-up multicore programming, 5, 17, 21, 300

declarative, 300

imperative, 300. *See also* imperative programming

boundary related errors, 390

broadcast operation, condition variable and, 263

BSB. *See* **Back Side Bus**

build and fix model, 39

bus connection, 14–15

DCA and, 22

buses, 14–15

Back Side, 14

EIB, 28, 30, 31

Front Side, 14

processor-to-, configuration, 14–15

C

C++

abstraction, 86

compilers. *See* **compilers**

encapsulation, 86. *See also* encapsulation

interface classes. *See* **interface classes**

ISO standard and, 17, 62, 63

library approach to parallelism and, 62–64

multiparadigm programming, 5, 46, 64, 86

multiprocessing implementation in, 62–64

predicates. *See* **predicates**

Stroustrup and, 62–63, 322

C++0x (C++09) standard, 304–305

class libraries, 305

asio, 305

early implementations, 316

Interprocess, 305

MPI, 305

mutex interface class, 312

implementation of (listing 8–2), 312–316

cache, 12

compiler options, 8–9, 13

L1, 12

L2, 12–13

calculation errors, 390

calculus, situational, 286

«call», 350

call action, 357

cancelability state, 172, 173

cancelability type, 172, 173

cancelable thread, 172

cancellation, 172–180

asynchronous, 173

deferred, 173

disabled, 173

of peer thread, 174

cancellation points, 175–176

list of, 177–178

cancellation-safe

library functions, 177

system calls, 177

catch [] block, 392

CBE (IBM Cell Broadband Engine), 3, 17, 28–31

comparison, 21

heterogeneous architecture, 28, 29, 30, 33

Linux and, 29

memory models, 29–30

PPE, 28, 29, 30, 31

processor overview, 28–29

SDK, 29

SPEs, 28, 29, 30, 31

SPUs, 30, 31

CCS (Computing Classification System), 284, 285, 286, 290

Cell Broadband Engine. *See* **CBE**

challenges (concurrency), 35–65

concurrency designs, 37, 61–62, 401–409

concurrent access to data/resources by multiple
tasks/agents, 51–56

debugging, 60–61

decomposition. *See* **decomposition**

list of, 36–37

multiprocessing implementation in C++, 62–64

processes/threads, necessary number of,
59–60

resource contention. *See* **resource contention**

task-to-task communication, 47–51

testing and, 60–61, 377–379. *See also* **testing**

[changeable]

association property, 352

attribute property, 340

child processes, 79, 80, 82, 141, 152

chip designs

CBE, 3, 17, 28–31

Core 2 Duo, 21, 31–33

Opteron dual core, 21–25

UltraSparc T1, 25–28

chip manufacturers, 1

CMPs and, 2

chip multiprocessors. *See* **CMPs**

chip-level multithreading (CMT), 25, 29, 149.

See also **multithreading**

chipset, 32

Northbridge, 31, 32
Southbridge, 31, 32

class(es). See also interface classes; specific classes

abstract, 335
active, 359
associated, multiplicity between, 347
attributes. *See* attributes
base, 349
collection, 335
concrete, 335
container, 335
datatypes, 335
defined, 334
domain, 335
exception. *See* exception classes
icon, 336, 347, 348
instances of a, 345
 visualizing, 345–347
instantiation of a, 345
interface. *See* interface classes
logic_error, 394, 395
modeling, 334–336
 concepts, 334
 real-world physical entities, 334
 software constructs, 334
multiplicity of, 347
name of, 336
nice, 343, 344
node, 335
objects and
 relationship between, 349–353
parent, 349
pure abstract, 347
realization and, 354
responsibilities of, 336
 visualizing, 336–339
root, 349
runtime_error, 394
services of. *See* services
simple name of, 336
singleton, 347
subclass, 349
superclass, 349
support/utility, 335
template, 335, 348
 visualizing, 348–349
types, 335
visualizing, 336

class diagrams, 401–405
class libraries (C++0x standard), 305
asio, 305
Interprocess, 305
MPI, 305

cleanup stack, 178, 179

close(), 177, 229. See also mq_close()
definition for, 229

CMD, 116**CMPs (chip multiprocessors), 2, 17**

chip manufacturers and, 2
cost, 4, 16, 96
gatekeeper of, operating system as, 69–71,
93, 312
heterogeneous designs, 28, 29, 30, 33
homogeneous designs, 29, 33
massively parallel, 86, 94, 282, 284–287, 290,
300, 304, 326, 328, 398
as multicore, 2, 96
octa-core, 2, 17, 283
parallel programming and, 4
quad core, 2, 3, 31, 73, 283
trend in, 3, 14, 36, 45, 65, 86, 283

CMT (chip-level multithreading), 25, 29, 149. See also multithreading**code generation (compiler option), 7, 33****code listings. See listings****code, reentrant, 130, 190****collaborating objects, 357–359****collaboration, 356**

parts of, 356
behavioral, 356
structural, 356

collaboration diagrams, 62, 357–358, 405–406**collection classes, 335****COMMAND, 116****command-line arguments, 48**

environment variables and, 212–213
exec family of functions and, 238

communication. See also IPC; ITC

dependencies, 205–207
flows of, multiple, 360–361
painters and, 51, 204
between processes, 47–51, 150–151
task-to-task, 47–51
between threads, 51, 150–151

comparing

multicore architectures, 21
CBE, 21
Core 2 Duo, 21
Opteron, 21
thread ids, 166–167

compilers, 7–8, 17

compilation process, 7–8
GNU C++, 8, 28, 29, 64
instruction set and, 7–9
Intel C/C++, 64
options, 6–7, 8, 9, 17, 33
 cache, 8–9, 13
role of, 7–8

- SIMD and, 6, 7, 9
- Sun C/C++, 28, 64
- compiling threaded programs, 162**
- [complete], 351**
- component diagrams, 61**
- composite state, 369**
- Computing Classification System (CCS), 284, 285, 286, 290**
- conceptual design layers (layers 4, 5), 289, 300, 303**
- concrete classes, 335**
- concurrency. *See also* multicore programming; parallel programming**
 - in C++ program, 141
 - challenges. *See* challenges
 - defined, 37
 - designs, 61–62, 401–409
 - synchronizing, 238–268
- concurrency models, 411–425**
 - Blackboard approach
 - with threads, 420–421
 - CRCW, 242–243, 423, 424
 - CREW, 242–243, 282, 425
 - ERCW, 242–243, 424, 425
 - EREW, 54, 242–243, 282, 424, 425
 - IPC. *See* IPC
 - ITC. *See* ITC
 - MIMD, 294, 301, 303, 324, 421, 422
 - monitor approach, 420
 - in PADL (layer 4), 288, 300–303
 - peer-to-peer
 - processes, 415–416
 - thread strategies, 271–272, 289, 415–416
 - pipeline model, 233, 269, 273–274, 277, 418
 - PRAM model, 423, 424
 - producer-consumer thread strategies, 272–273, 289, 418–419
 - SIMD, 6, 7, 9, 29, 30, 32–33, 84, 421, 422
 - workpile approaches, 417
- concurrent (service property), 340, 342**
- concurrent access to data/resources by multiple tasks/agents, 51–56**
- Concurrent Read and Concurrent Write. *See* CRCW**
- Concurrent Read and Exclusive Write. *See* CREW**
- concurrent state diagrams, 62, 64, 408**
- concurrent systems, model checking and, 326, 398**
- concurrent_hash_map, 317**
- concurrent_queue, 317**
- concurrent_vector, 317**
 - in listing 8–3, 318–321
- condition variables, 263–268**
 - attribute object, 264–265
 - multiple, testing and, 378
 - mutex semaphores and, 264
 - SF synchronization relationship (listing 7–13), 266–267
 - operations, 263, 264
 - testing and, 378
- constraints**
 - for associations, 352
 - [complete], 351
 - declarative models and, 46
 - [disjoint], 351
 - for generalization relationships, 351
 - [implicit], 352
 - [incomplete], 351
 - [ordered], 352
 - [overlapping], 351
- constructor**
 - copy, 343, 344
 - default, 343, 344
- consumer-producer model. *See* producer-consumer model**
- container adaptors, 353**
- container classes, 335**
- containers**
 - deque, 353, 354
 - list, 353, 354
 - vector, 353, 354
- contention scopes, 147, 154, 157–159**
 - process, 154, 157, 158
 - pthread attr_set functions, 168
 - setting, 187–188
 - system, 157, 158
- context**
 - of process, 147
 - of thread, 147–149
- context failure resilience, 381**
- context switching, 37, 60, 121, 147, 150**
- control flow errors, 390**
- control strategies (layer 3, PADL), 323–326**
- cooperation dependencies, 207**
- “copy”, 350**
- copy constructor, 343, 344**
- Core 2 Duo (Intel), 31–33**
 - comparison, 21
 - instruction set, 32–33
 - motherboard, 31–32
 - block diagram, 31
 - chipset, 32
 - Northbridge, 31, 32
 - PCI-E, 32
 - Southbridge, 31, 32
- cores, 2. *See also* multicore**
 - logical, 149
 - octa, 2, 17, 283
 - quad, 2, 3, 31, 73, 283
 - SMT, 149
 - V9, 27

cost

- CMPs, 4, 16, 96
- parallel programming, 16, 35
- Playstation 3, 29
- single processor computers, 35

counting semaphore, 248

counting task dependencies, 208–210

CP, 116

%CPU, 116, 118

CPU-intensive processes, 114

CRCW (Concurrent Read and Concurrent Write), 242–243, 423, 424

create action, 357

creation functions (pthread_mutex_t attribute object), 254

CREW (Concurrent Read and Exclusive Write), 242–243, 282, 425

critical sections, 241, 340

- Blackboard as, 317–318
- managing, with mutex semaphores, 257

crossbar

- Opteron, 23
- UltraSparc T1, 27

CTEST Laboratories, 287, 328

D

data level parallelism

- MIMD approach, 421, 422
- SIMD approach, 421, 422

data races (race conditions), 52–54, 151, 190, 191, 192, 204, 390

data resources, 130

data rollback, 392

data segment, 109

data structures

- ITC and, 231–233
- thread-safe, 268

data synchronization, 239, 240

datatypes classes, 335

DCA. See Direct Connect Architecture

deadlocks, 54–55, 64, 174, 219, 239, 241, 246, 255, 282, 387, 395

debugging, 60–61, 376, 377, 399. See also testing

decision symbol, 364

declarative models, 45–46, 64, 329

- AAs and, 45
- concurrent constraint, 46
- constraint, 46
- functional, 46
- languages in, 46
- object-oriented, 38, 39, 40, 46, 141
- procedural models v., 45, 64

decomposition, 41–47, 83, 291

- AAs, 41, 44, 45
- challenges, 43
- defined, 41, 43
- four-task application, 73–75
- guess_it program, 83–85
- models and, 44–46
- multicore programming and, 83
- and operating system's role, 83–85
- painters and, 42–44, 46–47
- in text files filtering project. See text files filtering project
- WBS, 41, 45, 268

default constructor, 343, 344

defect removal

- approach for, 381–386
- defect survival v., 380–381
- during testing stages, 380

defect survival

- defect removal v., 380–381
- during exception handling, 380

defects

- defined, 378

deferred cancellation, 173

deferred events, 367, 368

delegation (boss/worker) model

- with processes, 412–413
- thread strategy approaches, 269–271, 289, 412–414

dependency relationships, 205–210, 282, 333, 349

- communication, 205–207
- cooperation, 207
- stereotypes for, 349–351
- task, 208–210

dependency stereotypes, 349–351

- «access», 350
- «become», 350
- «bind», 350
- «call», 350
- «copy», 350
- «extend», 350
- «friend», 350
- «include», 350
- «instanceof», 350
- «instantiate», 350
- «refine», 350
- «use», 350

deployment diagrams, 61, 62, 85

deployment view, 371

deque container, 353, 354

descendant class, 91

design layers (layers 4, 5), 289, 300, 303

design view, 371

Designing Concurrent, Distributed, and Real-Time Applications with UML (Gomaa), 373

designs. *See also* application design; multicore architectures

- concurrency, 61–62, 401–409
- for multicore architectures, 19–34
- SDLC activity, 38, 85

destroy action, 357

destruction functions (pthread_mutex_t attribute object), 254

destruction operation

- condition variables, 264
- mutex semaphores, 253
- POSIX semaphores, 249
- read-write locks, 258

destructor, 343, 344

detached state, 154

detached threads, 155, 169–171

development tool, operating system as. *See* operating system

diagrams (UML language primitive), 333. *See also* UML

DictionaryLexicon, 336, 337, 338, 353, 356, 372

- attributes, categorization of, 344–345
- multiplicity between associated classes, 347
- multiplicity of classes, 347

die, 2, 21

Direct Connect Architecture (DCA), 22

disabled cancellation, 173

[disjoint], 351

distributed programming, 63, 64

distributed shared memory (DSM) architecture, 25

documentation techniques

- concurrency design. *See* concurrency
- dependency graph, 209
- errors, 391
- modeling. *See* modeling
- Specifications (SDLC activity), 38
- UML. *See* UML

domain classes, 335

do_something(), 89, 91, 92, 194, 197, 200

DSM. *See* distributed shared memory architecture

dual core Opteron. *See* Opteron

dynamic priority, 114

E

EIB (interconnect bus), 28, 30, 31

encapsulation

- C++, 86
- of message queue functions, 226–230
- of operating system's role, 30, 33, 34, 68, 86–93
- predicates and. *See* predicates
- in text files filtering project. *See* text files filtering project
- thread_object interface class, 87–90, 93

Englemore, Robert, 299

entry/exit actions, 367, 368

environment variables, 126–127, 212–213

- command-line arguments and, 212–213
- list of, 126

equality operator, 343, 344

ERCW (Exclusive Read and Concurrent Write), 242–243, 424, 425

EREW (Exclusive Read Exclusive Write), 54, 242–243, 282, 424, 425

error handling, 391

errors

- boundary related, 390
- calculation, 390
- control flow, 390
- deadlocks, 54–55, 64, 174, 219, 239, 241, 246, 255, 282, 387, 395
- defined, 378
- documentation, 391
- in handling/interpreting data, 391
- hardware, 391
- indefinite postponement, 56, 230, 239, 282, 301, 387
- initial state, 390
- later state, 390
- load conditions, 391
- parallel programming, 387
- performance degradation, 387
- priority inversion, 387
- race conditions, 52–54, 151, 190, 191, 192, 204, 390
- during testing, 391
- thread exhaustion, 387
- user interface, 390

ethical/moral responsibility, in software development, 377, 399

evaluation. *See* testing

event

- block, 366
- exit, 366
- wakeup, 366

event trigger, 369

evolutionary programming techniques, 300

exception classes, 394

- deriving new, 395–396
- extension of, 393
- protecting, from exceptions, 396

exception handler, 392–396

exception handling, 380–381

- after-the-fact, 381
- defect survival during, 380
- defined, 378
- strategies, 392–393

exception object, throwing, 394–395

exclusive access, to resources, 130

Exclusive Read and Concurrent Write. *See* ERCW

Exclusive Read Exclusive Write. *See* EREW

exec() family of functions, **63, 122–125, 135**

command-line arguments and, 238

failure conditions for, 122–123

restrictions, 125

execl() functions, 123–124

execution units, 75, 83, 96, 97. *See also* kernel threads; processes

tasks associated with (four-stage process), 96

execv() functions, 124–125

exit(), 128

exit event, 366

explicit binding, 348

«extend», **350**

extreme programming model, 39

F

failover, 393

failures, 379–381

Fast option, 6

fault handling

defined, 378

fault tolerance

defined, 378, 379

exception handling, 380–381

after-the-fact, 381

defect survival during, 380

defined, 378

strategies, 392–393

logical, 391–397, 399

strategy for, 396–397

FF relationship. *See* finish-to-finish relationship

FIFA (Foundation for Intelligent Physical Agents) specification, 291

FIFO (First-In, First-Out)

interface class, 221–222

named pipes. *See* named pipes

scheduling policies, 115–116, 152, 160

file action attribute functions, 100

file descriptors, 48, 98, 99, 100, 128, 131, 132, 148, 149, 213–214

file handles, for ITC, 231, 238

file_actions parameter, 99–100

files

filtering. *See* text files filtering project

format of, 213

with locking facilities, 48

names of, 213

open, 213

strings in, 213

transferring process, 213

filesystem management, 71

filter_thread class, 196, 197

filter_thread objects, multiple, 198–200

final state, 367

find_code program, 78, 79

multithreaded version, 80–83

object-oriented, 87–88

posix_spawn in, 104–105

finish-to-finish (FF) relationship, 57–58, 246, 265

finish-to-start (FS) relationship, 57–58, 245–246, 265

finite-state concurrent systems, model checking and, 326, 398

First-In, First-Out. *See* FIFO

First-Order Logic analysis, 326

floating-point option, 6, 8

flows of control/communication, multiple, 360–361

focus layer, 324

fork-exec(), 98, 135, 141, 143

fork() function, 98, 104, 122, 123, 135, 141

Found variable, 82, 92

Foundation for Intelligent Physical Agents (FIFA) specification, 291

four-stage process, execution units and, 96

four-task application, decomposition of, 73–75

«friend», **350**

Front Side Bus (FSB), 14

[frozen]

association property, 352

attribute property, 340

FS relationship. *See* finish-to-start relationship

FSB. *See* Front Side Bus

functions. *See* specific functions

functions (POSIX), 98

atomic, 192

multithreaded versions of, 191–192

for setting resource limits, 131–133

spawn/manage processes, 97–98

adding file actions, 98

creating processes, 98

destroying attributes, 98

initializing attributes, 98

setting/retrieving attribute values, 98

G

game scenario. *See* guess_it program

gatekeeper of CMP, operating system as, 69–71, 93, 312

generalization relationships, 333, 349

constraints, 351

stereotype, 351

getenv(), 126

getpid(), 107

getppid(), 107

getpriority(), 120, 121
getrlimit(), 132
getting process priorities, 119–121
global (stereotype), 350
global data, ITC and, 231–233
global variable, Answer, 231, 237, 244, 257
GNU C++ compiler, 8, 28, 29, 64
Gomaa, Hassan, 373
Grady Booch's Approach, 332
group effort, software development as, 304
group id, 98
guard condition, 369
guarded (service property), 340, 341
guess_it program, 78–85

- concurrency infrastructure, block diagram of, 302
- listing 4–1, 78–80
- listing 4–2, 80–83
- listing 5–3, 106
- listing 5–5, 137–139
- multiagent architectures and, 292–293
- operating system components
 - decomposition, 83–85
- PADL analysis in, 291–293, 301
- PBS breakdown of, 327
- physical pieces of, 85
- posix_spawn and, 104–107
- testing/pretest (flow of processes), 382–383
 - concurrency model for agents, 382
 - declarative implementation of PBS, 383
 - listing 10-1 (declarative implementation of guess_it), 383
 - listing 10-2 (valid_code predicate, declaration of), 384
 - listing 10-3 (valid_code predicate, definition of), 384–385
- PBS of agent solution model, 383
- problem statement, 382
- processes before, 382–383
- revised agent model, 382
- rough-cut solution model, 382
- solution model-with layer 5 PADL, 382
- strategy, 383

Guide for Developing Software Life Cycle Processes (IEEE std 1074), 287

H

handling data, errors in, 391
hardware errors, 391
hardware resources, 130
hardware synchronization, 239
hardware-independent interface, 20, 33, 34
hardware-specific issues, 20, 21, 33, 34
heterogeneous CMP designs, 28, 29, 30, 33

hiding. See encapsulation
\$HOME, 126
homogeneous CMP designs, 29, 33
house painters. See painters
HT technology. See HyperTransport technology
hybrid multicore architectures, 3–4
hybrid threads, 144, 145
hyperthreaded processor, 3, 25
hyperthreading, 3, 31, 149
HyperTransport (HT) technology, 22, 23, 25

I

IBM Cell Broadband Engine. See CBE
icon, class, 336, 347, 348
id

- PID, 107, 116
- PPID, 109, 116
- thread, 166–167
- user, 98

IEEE (Institute of Electrical and Electronics Engineers), 386

- POSIX standards, 63
 - for Process Management, 567–592
 - Single Unix Specification Version 3 and, 63–64, 248
 - for Thread Management, 427–526
- standard software engineering tests, 386–399
- std 1003.1–2001, 63
- std 1008, 386
- std 1012, 386, 387, 389
- std 1074, 287

ILP (inductive logic programming) techniques, 300
imperative programming, 46, 287, 300, 326, 327, 328, 329, 381
«implementation», 351
implementation (SDLC activity), 38
‘implementation layer’ mapping, 323
implementation model of PADL (layer 3), 289, 304–326

- control strategies, 323–326

implementation view, 371
[implicit], 352
implicit (constraint), 352
implicit binding, 348
«include», 350
[incomplete], 351
incremental model, 39
indefinite postponement, 56, 230, 239, 282, 301, 387
inductive logic programming (ILP) techniques, 300
infinite number (multiplicity notation), 347

- one to, 347
- zero to, 347

inheritance, 89, 90, 98, 119, 121, 126, 130, 141

exception class extension through, 393
multiple/single, UML diagram, 404

initial state, 367

errors, 390

initialization operation

condition variables, 264
mutex semaphores, 253
POSIX semaphores, 248
read-write locks, 258

«instanceof», 350**instances of a class, 345–347**

anonymous, 346
multiobjects and, 347, 348
multiple, 347
named, 345
orphan, 346
visualizing, 345–347

«instantiate», 350**instantiation of a class (object), 345****Institute of Electrical and Electronics Engineers.**

See IEEE

instruction set, 7–9

compilers and, 7–9
Core 2 Duo, 32–33

integration testing, 379**Intel C/C++ compiler, 64****Intel Core 2 Duo processor. *See* Core 2 Duo****Intel Threading Building Blocks library. *See* TBB library****interaction diagrams, 405–407****interactional agents, 291****interactive objects, organization of, 356****interconnect bus (EIB), 28, 30, 31****interface, 68, 93. *See also* APIs; POSIX; SPIs****«interface», 354****interface classes (C++), 86–93, 335, 353. *See also* posix_process interface class; thread_object interface class**

FIFO, 221–222
message queue, 224–230
mutex (C++0x standard), 312–316
predicates, processes and, 137–141
stack class as, 354, 355
thread (C++0x standard), 305–312
visualizing, 353–355
as wrappers, 86, 192, 193
thread_object, 193, 201

internal transitions, 367, 368**International Organization for Standardization. *See* ISO standard****interpreting data, errors in, 391****Interprocess class library, 305****Interprocess Communications. *See* IPC****Interthread Communications. *See* ITC****I/O controller, 32****I/O management, 71****IPC (Interprocess Communications), 47–51, 152, 210–230, 282, 411–412**

as bridges, 49–50
challenges, 49
defined, 210, 282
diagram, 412
ITC v., 51, 231
list, 48–49
Manager, 71, 94
operating system APIs and, 210, 282
persistence of, 211–212

ISO (International Standardization) standard
C++ and, 17, 62, 63**isQuery (service property), 340****ITC (Interthread Communications), 50, 51, 230–238, 282, 411–412**

data structures and, 231–233
defined, 282
diagram, 412
file handles for, 231, 238
global data and, 231–233
IPC v., 51, 231
parameters for, 234–238
variables and, 231–233

iteration marker, 362**iterative shared solution space, Blackboard as, 298–299**

J

Jacobson, Ivar, 371, 372**joining threads, 165–166**

K

kernel mode, 108, 146**kernel processes, 108****kernel stack, 109****kernel threads (lightweight processes), 51, 73, 75, 77, 144, 200****kill() function, 128–129****killing processes, 127–129. *See also* terminating****knowbot, 290. *See also* agents****knowledge sources (KS), 294, 299–300**

anatomy of, 299–300
for 'question and answer browser'
program, 297
rational agents as, 299–300

Kripke Structures, 398**KS. *See* knowledge sources**

L**L1 cache, 12****L2 cache, 12–13**

UltraSparc T1, 28

lag time, 58**language primitives, UML, 332–333****languages**

in declarative models, 46

in software development methodologies, 39

Universal Modeling Language. *See* UML**later state, errors in, 390****layers, PADL model**

1 (operating system), 288, 289

2 (POSIX APIs), 288, 289, 290

3 (implementation model of PADL), 288, 289, 304–326

4 (concurrency models in PADL), 288, 300–303

5 (application architecture selection), 288, 290–300

levels, software layer

1, 69–71

2, 69–71. *See also* APIs3, 69–71. *See also* TBB library4, 69–71. *See also* STAPL**Lexicon system**

activity diagram with swimlanes in, 364–365

packages in, 372

sequence diagram of objects in, 362

structural part of, 356

libraries

approach, to parallelism, 62–64

Boost C++, 316

mutex class, implementation of (listing 8–2), 312–316

thread class, implementation of (listing 8–1), 305–312

C++0x standard, 305

asio, 305

Interprocess, 305

MPI, 305

multicore-aware, 71

multithreaded versions of, 191–192

pthread, 154, 162

shared, 130

STAPL, 69, 70, 76, 84, 86, 93, 94, 321–323, 567

TBB, 70, 71, 86, 317–321

thread safe, 71, 190–192

library functions, cancellation-safe, 177**lightweight processes (kernel threads), 51, 73, 75, 77, 144, 200****linear-sequential model, 40****link (between objects), 357****linking threaded programs, 162****Linux, CBE and, 29****list container, 353, 354****listings (code)**

2-1 (UltraSparc T1), 25, 29

4-1 (guess_it program), 78–80

4-2 (find_code, multithreaded version), 80–83

4-3 (ofind_code), 87–88

4-4 (thread_object, declaration), 88–90

4-5 (thread_object, implementation), 90–91

4-6 (user_thread class, definitions), 91–92

5-1 (ofind_code, posix_spawn and), 104–105

5-2 (posix_process interface class), 105

5-3 (guess_it, posix_process and), 106

5-4 (posix_process, method definitions for), 106–107

5-5 (guess_it, declarative form), 137–139

5-6 (valid_code), 139

5-7 (valid_code, definitions for), 139–140

6-1 (passing arguments to thread), 163–165

6-2 (thread_object/user_thread, declaration of), 193–194

6-3 (thread_object class, definition of), 194–196

6-4 (filter_thread class, definition of), 196–197

6-5 (filter_thread objects, multiple), 198–200

7-1 (named pipe, creation), 218–219

7-2 (named pipe, reading), 219–220

7-3 (posix_queue class, declaration of), 224–226

7-4, 7-5, 7-6 (ITC and global data/ variables/data structures), 231–234

7-7, 7-8 (parameters for ITC), 234–238

7-9 (semaphores, on output file), 249–250

7-10 (semaphores, on input file), 250–251

7-11 (object-oriented mutex class, declaration), 261

7-12 (permit class, definition of), 262

7-13 (synchronization relationship, condition variables/mutexes), 266–267

8-1 (C++0x thread class, implementation of), 305–312

8-2 (C++0x mutex class, implementation of), 312–316

8-3 (concurrent_vector, parallel_for, from TBB), 318–321

10-1 (declarative implementation of guess_it), 383

10-2 (valid_code predicate, declaration of), 384

10-3 (valid_code predicate, definition of), 384–385

10-4 (user-defined posix_process interface class, definitions for), 388–389

load conditions, 391**local (stereotype), 358****locality**

spatial, 12

temporal, 12

locking facilities, 48

logic

- First-Order Logic analysis, 326
- ILP techniques, 300
- modal, 286
- possible worlds in, 398
- logic programming, 39, 40, 186**
- logical cores, 149**
- logical fault tolerance, 391–397, 399. See also fault tolerance**
- logic_error classes, 394, 395**
- loop unrolling, 7, 33**
 - compiler option, 7
- low-level operating system primitives, 93, 287, 290, 300, 312, 323, 329**
- LWP, 116**

M

- MagicCode, 82, 85, 89**
- \$MAIL, 126**
- maintenance (SDLC activity), 38**
- managing processes. See process management**
- massively parallel CMPs, 7, 86, 94, 282, 284–287, 290, 300, 304, 326, 328, 398**
- %MEM, 116, 118**
- memory, 9–14. See also cache; PRAM models; RAM; registers**
 - hierarchy, 10–14
 - models, CBE, 29–30
 - shared, 49, 152, 214
 - POSIX, 214–216
 - sizes, 10
 - speeds, 10–11
- memory bandwidth, 7**
 - compiler option, 7
- memory controller hub, 32**
- memory controllers**
 - UltraSparc T1, 28
- memory flow controller (MFC), 30, 31**
- memory management, 71**
- memory management unit (MMU), 30, 31**
- Message Passing Interface (MPI) class library, 305**
- message queues (POSIX), 49, 50, 222–230**
 - functions
 - encapsulation of, 226–230
 - list, 74
 - interface class, 224–230
 - posix_queue class, 224–230
 - declaration of, 224–226
 - using, 222–224
- message sequences, between objects, 361–363**
- MFC (memory flow controller), 30, 31**
- MIMD (Multiple Instruction Multiple Data), 294, 301, 303, 324, 421, 422**

- minimal standard interface, 344**
- mkfifo(), 218–219**
- MMU (memory management unit), 30, 31**
- MMX registers, 6, 33**
- modal logic, 286**
- mode**
 - kernel, 108, 146
 - user, 108, 146
- model checking, 326, 398**
- model implementation, 304**
- model, of system, 331, 372**
- modeling, 331**
- modeling concurrent systems, 331–373**
 - architectural perspective (whole system), 331, 371–372
 - behavioral perspective (concurrent behavior), 331, 357–370
 - Designing Concurrent, Distributed, and Real-Time Applications with UML*, 373
 - diagrams, 373
 - for active objects, 359–360
 - activity, 62, 64, 407
 - class, 89, 401–405
 - collaboration, 62, 357–358, 405–406
 - component, 61
 - of concurrent substates, 369–370
 - deployment, 61, 62, 85
 - interaction, 405–407
 - object, 401–405
 - package, 409
 - sequence, 62, 361–362, 406
 - state/concurrent state, 62, 64, 366–367, 408
 - as UML language primitive, 333
 - structural perspective (system structure), 331, 334–356
 - classes, 334–336
 - UML and, 332–333. *See also* UML
 - visualizing
 - attributes, 336–339
 - classes, 336
 - instances of a class, 345–347
 - interface classes, 353–355
 - responsibilities, 336–339
 - services, 336–339
 - template classes, 348–349
- models, 44–46**
 - CBE memory models, 29–30
 - concurrency. *See* concurrency models
 - declarative, 45–46, 64
 - AAs and, 45
 - concurrent constraint, 46
 - constraint, 46
 - functional, 46
 - languages in, 46

- object-oriented, 46
- procedural models v., 45, 64
- decomposition and, 44–46
- defined, 44, 331
- PADL. *See* PADL model
- procedural, 45, 64
 - declarative models v., 45, 64
 - sequential model of programming and, 45
 - WBS and, 45
- sequential. *See* sequential model of programming
- software development. *See* software development methodologies
- termination, 397
- monitor approach, 420**
- monitoring processes, with ps utility, 105, 116–119**
- moral/ethical responsibility, in software development, 377, 399**
- Morgan, Tony, 299**
- MorphologyAgent, 336, 356, 364, 365, 372**
- motherboard, Intel Core 2 duo, 31–32**
- MPI (Message Passing Interface) class library, 305**
- MPMD (Multiple Program Multiple Data), 274–275**
- mq_close(), 74, 223, 229**
- mq_getattr(), 74**
- mq_notify(), 74**
- mq_open(), 74**
- mq_receive(), 74, 178, 224, 228**
- mq_send(), 74**
- mq_setattr(), 74**
- MTSD (Multiple Threads Single Data), 275, 281**
- multiagent architectures, 291–293, 328**
 - guess_it program and, 292–293
- multicore. *See also* CMPs**
 - as CMPs, 2, 96
 - defined, 2
 - as multiprocessor, 96
- multicore architectures, 2–3**
 - CBE, 3, 17, 28–31
 - challenges in, 64
 - comparison of, 21
 - Core 2 Duo, 21, 31–33
 - designs for, 19–34
 - hybrid, 3–4
 - Opteron dual core, 21–25
 - software redesigning for, 1
 - trend in, 3, 14, 36, 45, 65, 86, 283
 - UltraSparc T1, 25–28
- multicore programming**
 - bottom-up, 5, 17, 21
 - challenges, 35–65
 - decomposition and, 83. *See also* decomposition
 - operating system's role in. *See* operating system's role
 - SDLC and, 37–41, 59, 64, 65
 - sequential programming v., 2, 4, 15, 36, 65
 - single core to, 15–17
 - testing for. *See* testing
 - top-down, 17
- multilevel priority queue, 114**
- multiobjects, 347, 348**
- multiparadigm programming, 5, 46, 64, 86**
- multiple condition variables, 378**
- multiple filter_thread objects, 198–200**
- multiple flows of control/communication, 360–361**
- multiple instances of classes, 347**
- Multiple Instruction Multiple Data. *See* MIMD**
- Multiple Program Multiple Data. *See* MPMD**
- Multiple Threads Single Data. *See* MTSD**
- multiplicity**
 - between associated classes, 347
 - in associations, 352
 - of classes, 347
 - notation, 347
 - 0 to 1, 347
 - 0 to infinite number, 347
 - infinite number, 347
 - one instance, 347
 - one to infinite number, 347
 - one to specified number n, 347
- multiprocessing, 15**
 - implementing, in C++, 62–64
- multiprocessors, 96. *See also* CMPs**
 - classic, 3
 - defined, 15, 96
 - massive, application design for, 284–287
 - multicore as, 96
- multiprogramming, 15**
- multitasking, 76, 359–361**
- multithreaded process, 143–144, 200**
- multithreaded version(s)**
 - find_code program, 80–83
 - of libraries/functions, 191–192
- multithreading, 143–201, 359–361. *See also* threads**
 - chip-level, 25, 29, 149
 - key points, 200–201
- mutex attribute object, 254–256. *See also* pthread_mutex_t attribute object**
- mutex class, object-oriented, 261**
- mutex exhaustion, 387**
- mutex interface class (C++0x standard), 312**
 - implementation of (listing 8–2), 312–316

mutex semaphores, 252–257

- condition variables and, 264
 - SF synchronization relationship (listing 7–13), 266–267
- critical sections and, 257
- private v. shared, 256
- testing and, 378

mutual exclusion, 252. See also mutex semaphores
MXCSR registers, 33

N

name(s)

- for associations, 352
- of class, 336
- of files, 213
- of state, 367

named instances of classes, 345

named pipes, 216, 218–221. See also FIFO

- creation of, 218–219
- reader for, 219–220

n-ary associations, 349

natural language processing (NLP) system, 276, 296. See also text files filtering project

NI, 116

nice classes, 343, 344

nice value, 119, 120, 121

nice() function, 119, 120, 121

NLP (natural language processing) system, 276, 296. See also text files filtering project

NLWP, 116

node classes, 335

Non-Uniform Memory Access (NUMA) architecture, 24–25

Northbridge, 31, 32

notation, multiplicity, 347

notification, for outside assistance, 393

NUMA architecture. See Non-Uniform Memory Access architecture

numbers

- multiplicity notation, 347
 - 0 to 1, 347
 - 0 to infinite number, 347
 - infinite number, 347
 - one instance, 347
 - one to infinite number, 347
 - one to specified number n, 347

O

object(s), 345

- actions and, 357
- call, 357
- create, 357

destroy, 357

return, 357

send, 357

active, 359–360

associations, 333, 349

bi-directional, 349

binary, 349

constraints, 352

link, 357

multiplicity in, 352

names for, 352

n-ary, 349

navigation, 351

properties, 352

role in, 352

type, 352

unidirectional, 349, 352

attribute. *See* attribute object

classes, relationships between, 349–353

collaborating, 357–359

defined, 345

interactive, organization of, 356

link between, 357

message sequences between, 361–363

template, 348

visibility of, 357–358

object diagrams, 401–405

Object Management Group (OMG), 332

Object Modeling Technique (OMT), 332

object-oriented find_code program. See find_code program

object-oriented model, 38, 39, 40, 46, 141

object-oriented mutex class, 261

object-oriented programming (OOP), 38, 45, 86

Object-Oriented Software Engineering (OOSE), 38, 332

object-oriented (OO) software systems, 332

octa-core CMPs, 2, 17, 283

ofind_code, 87–88, 104–105

OMG (Object Management Group), 332

OMT (Object Modeling Technique), 332

one instance (multiplicity notation), 347

one to infinite number (multiplicity notation), 347

one to specified number n (multiplicity notation), 347

‘one-to-one’ thread mapping, 146

OO software systems, 332

OOP (object-oriented programming), 38, 45, 86

OOSE (Object-Oriented Software Engineering), 38, 332

open(), 214, 218

definition for, 226

Open Group Base Specifications Issue 6, 63. See also IEEE

OpenMP, 322

- directives, 6
- parallelization with, 6

operating system

- APIs. *See* APIs
- core services, 71–75, 94
- as gatekeeper of CMP, 69–71, 93, 312
- hardware-independent interface, 20, 33, 34
- layer 1 (PADL), 288, 289
- POSIX-compliant, 17, 26
- primitives, 93, 287, 290, 300, 312, 323, 329
- SPIs. *See* SPIs
- transparency, 72, 86, 94, 105

operating system's role (in multicore programming), 67–94, 283, 316

- decomposition and, 83–85
- developer's interaction and, 69–71
- encapsulation of, 30, 33, 34, 68, 86–93
- overview, 68–69
- resource management, 68–69, 94
- software interface, 68, 93
- software layers and, 69–71

operation retry, 393**operational testing, 380****operations. *See* services****operator(), 139, 140, 279, 280, 384****operator**

- assignment, 343, 344
- equality, 343, 344

Opteron (dual core, AMD), 21–25

- block diagram, 22
- comparison, 21
- crossbar, 23
- DCA, 22
- HT technology, 22, 23, 25
- NUMA architecture, 24–25
- SRI, 23

order of execution models, 244**[ordered], 352****ordering attributes/services, 343–345**

- by access, 343
- by category, 343, 344
- by category names, 344, 345

orphan instances of classes, 346**[overlapping], 351****P****package diagrams, 409****packages, 333, 371–372, 373****PADL (Parallel Application Design Layers) model, 287–326, 328–329**

- goal of, 375
- guess_it program and, 291–293, 301

layer 1 (operating system), 288, 289

layer 2 (POSIX APIs), 288, 289, 290

layer 3 (implementation model of PADL), 288, 289, 304–326

layer 4 (concurrency models in PADL), 288, 300–303

layer 5 (application architecture selection), 288, 290–300

overview, 287–290

page frame number, 110**page tables, 109****pages, 110****painters (example)**

- communication in, 51, 204
- data race in, 52–54
- deadlock in, 54–55
- decomposition in, 42–44, 46–47
- indefinite postponement in, 56
- synchronization in, 204
- timing considerations in, 58–59

pAlgorithms, 322**Parallel Application Design Layers. *See* PADL model****parallel design, documentation of, 61–62****parallel programming, 4, 15–16. *See also* multicore programming**

- CMPs and, 4
- cost, 16, 35
- defined, 15, 17
- errors, 387
- testing for. *See* testing
- tradition and, 35

Parallel Random-Access Machine model. *See* PRAM models**parallel STL library. *See* STAPL****parallel_for, 317**

- in listing 8–3, 318–321

parallelism

- architectural approach to, 287. *See also* application design
- data level
 - MIMD approach, 421, 422
 - SIMD approach, 421, 422
- implementation of
 - C++ ISO standard, 62, 63
 - library approach to, 62–64
 - massive, 7, 86, 94, 282, 284–287, 290, 300, 304, 326, 328, 398
 - Stroustrup on, 62–63

parallelization with OpenMP option, 6**parallel_reduce, 317****parallel_scan, 317****parallel_sort, 317****parallel_while, 317****parameter (stereotype), 358**

parameters

- attrp, 100–102
- file_actions, 99–100
- for ITC, 234–238
- path, 99, 123
- resource, 131
- parent class, 349**
- parent process id. See PPID**
- parent processes, 80, 141, 152**
- path parameter, 99, 123**
- \$PATH, 126**
- patterns of work, 288, 290, 291, 323, 326, 329**
- PBS (Predicate Breakdown Structure), 287, 326–328, 329, 375**
 - coding, 328
 - goal of, 375
 - for guess_it program, 327
 - PADL, SDLC and, 328
- PCB (process control block), 108–109, 141**
- PCI (Peripheral Component Interconnect), 14, 32**
- PCI-Express (PCI-E), 32**
- pContainers, 322**
- peer thread**
 - cancellation of, 174
- peer threads, 150, 153**
 - terminating, 172
- peer-to-peer model**
 - with processes, 415–416
 - thread strategy approach, 271–272, 289, 415–416
- performance degradation, 387**
- Peripheral Component Interconnect. See PCI**
- permit class, 261–262**
 - definition of, 262
- persistence, 211**
 - of IPC, 211–212
- PFN (physical page frame number), 110**
- physical page frame number (PFN), 110**
- PID (process id), 107, 116**
- pipeline (TBB generic parallel algorithm), 317**
- pipelines**
 - agent model, 278–282. *See also* text files filtering project
 - six-stage, 27
 - thread model, 233, 269, 273–274, 277, 289, 418
- pipes, 49, 216–222**
- platform-specific optimizations, 20, 21, 33, 34**
- Playstation 3, 29**
- Portable Operating System Interface. See POSIX**
- POSIX (Portable Operating System Interface)**
 - APIs. *See* APIs
 - defined, 75–76
 - exec() functions. *See* exec() family of functions
 - functions. *See* functions
 - IPC. *See* IPC
 - message queues. *See* message queues
 - scheduling policies. *See* scheduling policies
 - semaphores, 248–252
 - shared memory, 214–215
 - standards, 17, 63, 64
 - for operating systems, 17, 26
 - for Process Management, 567–592
 - Single Unix Specification Version 3 and, 63–64, 248
 - for Thread Management, 427–526
 - thread library, 154
 - threads. *See* pthreads
- posix_process interface class, 105, 106, 138, 141, 389**
 - definitions for (listing 10–4), 388–389
 - listing 5–2, 105
 - listing 5–3, 106
 - listing 5–4, 106–107
- PosixQueue, 82, 85, 89**
- posix_queue class, 224–230**
 - declaration of, 224–226
- posix_spawnattr_t structure, 100–103**
 - spawn process attributes functions in, 101–102
- posix_spawn_file_actions object, 98, 99, 100**
- posix_spawn() function, 63, 74, 78, 79, 80, 98–107, 135**
 - example, 103–104
 - guess_it program and, 104–107
- posix_spawnp(), 98, 99**
- possible worlds, 398**
- PowerPC Processing Element. See PPE**
- PPE (PowerPC Processing Element), 28, 29, 30, 31**
- PPID (parent process id), 109, 116**
- PRAM (Parallel Random-Access Machine) models, 242–243, 282, 423, 424**
 - CRCW, 242–243, 423, 424
 - CREW, 242–243, 282, 425
 - ERCW, 242–243, 424, 425
 - EREW, 54, 242–243, 282, 424, 425
- pRange, 322**
- Predicate Breakdown Structure. See PBS**
- predicate class**
 - valid_code
 - declaration of, 384
 - definition of, 384–385
- predicate exceptions, 397–398**
- predicates, 141, 384**
 - defined, 137
 - interface classes, processes and, 137–141
- PRI, 116, 118**
- primary thread, 143, 159, 200**
- primitives**
 - operating system, 93, 287, 290, 300, 312, 323, 329
 - UML language, 332–333

- PRIO_PGRP, 120**
- PRIO_PROCESS, 120**
- priorities**
 - dynamic, 114
 - process, setting/getting, 119–121
 - static, 114
 - of threads, 155
- priority ceiling functions (pthread_mutex_t attribute object), 254**
- priority inversion, 387**
- priority queue, multilevel, 114**
- PRIO_USER, 120**
- private v. shared mutex semaphores, 256**
- problem decomposition. See decomposition**
- problem solvers (Blackboard), 293, 294, 295**
- procedural models, 45, 64. See also imperative programming**
 - declarative models v., 45, 64
 - sequential model of programming and, 45
 - WBS and, 45
- process (stereotype), 359**
- process contention scope, 154, 157, 158**
- process context, 147**
- process control, 109**
- process control block (PCB), 108–109, 141**
- process creation, 121–125**
- process environment variables, 126–127**
- process group id, 98, 99, 129**
- process id (PID), 107, 116**
- process image, 109**
- process lifecycle, 76–77**
- process management, 71, 76–83**
 - game scenario, 77–83
 - POSIX functions for, 97–98
- process resources. See resources**
- process states, 111–114**
- process tables, 111**
- process view, 371**
- processes, 47–48, 108–137**
 - address space, virtual, 110, 111
 - anatomy, 47, 50, 109–111
 - asynchronous, 133–135
 - child, 79, 80, 82, 141, 152
 - communication between, 47–51
 - concurrency models with. *See* concurrency models
 - context of, 147
 - CPU-intensive, 114
 - creation of. *See* process creation
 - defined, 97
 - delegation model with, 412–413
 - differences, with threads, 152–153, 203
 - four-stage, execution units and, 96
 - IPC mechanisms, 47–51
 - kernel, 108
 - killing, 127–129
 - lightweight, 51, 73, 75, 77, 144, 200
 - monitoring, with ps utility, 105, 116–119
 - multitasking with, 76, 359–361
 - multithreaded, 143–144, 200
 - necessary number of, 59–60
 - parent, 80, 141, 152
 - peer-to-peer model with, 415–416
 - POSIX Standard for Process Management, 567–592
 - predicates, and interface classes, 137–141
 - priorities, setting/getting, 119–121
 - programs v., 97
 - reuse and, 151
 - scheduling of, 114–116
 - similarities, with threads, 152–153, 201
 - spawn/manage, POSIX functions for, 97–98
 - state of, 111–114
 - synchronous, 133–135
 - system, 108
 - tasks to, 74
 - threads v., 50, 97, 150–153
 - user, 108
 - variables for, 188–189
- processor architecture. See multicore architectures; single core processor architecture**
- processor state, 109**
- processor-to-bus configuration, 14–15**
- producer-consumer model (thread strategies), 272–273, 289, 418–419**
- profilers, 61**
- program code, 109**
- program profiles, 26. See also listings**
- programmer ethics, 377, 399**
- programming. See multicore programming; parallel programming**
- programming languages. See languages**
- programs, 95, 97. See also processes**
 - processes v., 97
- properties. See specific properties**
- protocol functions (pthread_mutex_t attribute object), 254–255**
- ps utility, 105, 116–119**
- pthread attribute object, 167–171**
 - default values, 168–169
 - Linux/Solaris environment, 169
 - detached threads from, 169–171
- pthread attr_set functions, 167–168**
 - contention scope, 168
 - detach state, 167
 - inheritance scheduling, 168
 - initialization, 167
 - parameter scheduling, 169
 - policy scheduling, 168
 - stack management, 167

pthread_attr_t, 167–168
pthread_cond_t, 263–265
pthread_create(), 80, 82, 85, 89, 91, 138, 161, 162, 163, 166, 168, 196, 201, 312
pthread_mutex_t attribute object, 254–256
 functions, 254–256
 creation/destruction, 254
 priority ceiling, 254
 protocol, 254–255
 shared, 255
 type, 255
pthread_rwlock_t attribute object, 258–259
 pthreads (POSIX threads), 29, 63, 82
 library, 154, 162
pure abstract classes, 347

Q

quad core CMPs, 2, 3, 31, 73, 283
quantum, 76, 77, 108
‘question and answer browser’ program (Blackboard), 296–298
 components of solution, 296–297
 concurrency infrastructure, block diagram of, 302
 KS for, 297
 listing 8–3, 318–321
 PADL analysis of, 301

R

race conditions (data races), 52–54, 151, 190, 191, 192, 204, 390
RAM (system main memory), 13–14
rapid prototyping model, 40
rational agents, 299–300
reader, 219–220
read-write locks, 257–263
 access policy implementation with, 259–261
 testing and, 378
ready queues, 112, 113, 114, 115, 116, 159
ready state. *See* runnable state
realization, 333, 354
 of template class, 354, 355
receive(), 228. *See also* mq_receive()
redundancy, 393
reentrant code, 130, 190
«refine», 350
registers, 10, 11–12
 MMX, 6, 33
 MXCSR, 33
 XMM, 33
regression testing, 379

relationships
 (UML language primitive), 333
 associations, 333, 349
 constraints, 352
 navigation, 351
 properties, 352
 type, 352
 between classes and objects, 349–353
 dependency, 333, 349. *See also* dependency stereotypes
 stereotypes for, 349–351
 generalization, 333, 349
 constraints, 351
 stereotype, 351
 synchronization. *See* synchronization
 whole-part, 352
release ownership operation
 mutex semaphores, 253
 POSIX semaphores, 249
 read-write locks, 258
reliability
 defined, 378
 programmer ethics and, 377, 399
remove(), 229
request ownership operation
 mutex semaphores, 253
 POSIX semaphores, 248
 read-write locks, 258
resource contention, 59
resource limits, POSIX functions for, 131–133
resource management, 68–69, 94
resource parameter, 131
resource reallocation/deallocation, 392
resources
 data, 130
 defined, 129–133
 exclusive access to, 130
 hardware, 130
 sharable, 130, 238–239
 software, 130
 thread, 149–150
 types of, 131
 unshared, 130
responsibilities (of class), 336
 visualizing, 336–339
return action, 357
reuse
 processes and, 151
 threads and, 151
RLIM_INFINITY, 131
RLIMIT_AS, 132
RLIMIT_CORE, 132
RLIMIT_CPU, 132
RLIMIT_DATA, 132
RLIMIT_FSIZE, 132

- RLIMIT_NOFILE, 132**
 - RLIMIT_STACK, 132**
 - RLIM_SAVED_CUR, 131**
 - RLIM_SAVED_MAX, 131**
 - role (in associations), 352**
 - root class, 349**
 - round robin (RR) scheduling policies, 115–116, 159**
 - RR (round robin) scheduling policies, 115–116, 159**
 - RSS, 116**
 - Rumbaugh, James, 371, 372**
 - OMT of, 332
 - run() method, 87, 91, 92, 107**
 - runnable (ready) state, 111, 112, 113, 114**
 - running state, 111, 112, 113, 114**
 - Runtime system (STAPL), 322**
 - runtime_error classes, 394**
 - RUSAGE_CHILDREN, 132, 133**
 - RUSAGE_SELF, 132, 133**
- S**
- S (header), 116**
 - safe**
 - cancellation-
 - library functions, 177
 - system calls, 177
 - thread-
 - data structures, 268
 - SCHED_FIFO, 77**
 - SCHED_OTHER, 77**
 - SCHED_RR, 77**
 - SCHED_SPORADIC, 77**
 - scheduling allocation domains, 160**
 - scheduling, of processes, 114–116**
 - scheduling policies (POSIX), 76–77, 99, 100, 101, 115, 149, 152**
 - FIFO, 115–116, 152, 160
 - ‘other,’ 160
 - round robin, 115–116, 159
 - thread, 155
 - setting, 183–187
 - SDLC. See Software Development Life Cycle**
 - self (stereotype), 358**
 - self-delegation symbol, 363**
 - self-transitions, 367, 368**
 - semaphores, 49, 247–257**
 - binary, 248
 - counting, 248
 - on input file (listing 7–10), 250–251
 - mutex, 252–257
 - testing and, 378
 - operations, 247–248
 - on output file (listing 7–9), 249–250
 - POSIX, 248–252
 - operations, 248–249
 - variable for, 188–189
 - semiautonomous/autonomous agents, 291**
 - send(), 177. See also mq_send()**
 - definition for, 227
 - send action, 357**
 - sequence diagrams, 62, 361–363, 406**
 - sequence synchronization, 244**
 - sequential (service property), 340, 341**
 - sequential model of programming, 1, 35–36**
 - defined, 36
 - multicore programming v., 2, 4, 15, 35–36, 65
 - procedural model and, 45
 - testing in, 60
 - tradition, 35
 - Serial Peripheral Interface (SPI), 32**
 - services (of classes), 336–339**
 - ordering, 343–345
 - by access, 343
 - by category, 343, 344
 - by category names, 344, 345
 - properties
 - concurrent, 340, 342
 - guarded, 340, 341
 - isQuery, 340
 - sequential, 340, 341
 - visibility of, 343
 - setenv(), 127**
 - setpriority(), 120, 121**
 - setrlimit() function, 131, 132**
 - setting process priorities, 119–121**
 - SF relationship. See start-to-finish relationship**
 - sharable resources, 130, 238–239**
 - shared functions (pthread_mutex_t attribute object), 255**
 - shared libraries, 130**
 - shared memory, 49, 152, 214**
 - POSIX, 214–216
 - shared v. private mutex semaphores, 256**
 - \$SHELL, 126**
 - SID, 116**
 - signal masks, 98, 101, 149**
 - signaling operation, conditioning variable and, 264**
 - SIMD (Single Instruction Multiple Data), 84, 274–275, 421, 422**
 - compiler and, 6, 7, 9
 - Core 2 Duo instruction set and, 32–33
 - PPE and, 30
 - SPEs and, 29, 30
 - SSE and, 6, 32–33
 - simple name, of class, 336**
 - simultaneous multithreaded (SMT) cores, 149**

single core processor architecture, 5–15

- application development, 1
- multicore programming and, 15–17

Single Instruction Multiple Data. See SIMD

Single Program Multiple Data. See SPMD

Single Unix Specification Version 3, 63–64, 248.

See also IEEE

singleton class, 347

situated agents, 291

situational calculus, 286

six-character code guessing. See guess_it program

six-stage pipeline, 27

sleeping state, 113

smart object, 290. See also agents

SMP. See symmetric multiprocessor

SMT (simultaneous multithreaded) cores, 149

society, of agents

- structured, 291
- whole more than sum in, 291

sockets, 49

softbot, 290. See also agents

software

- decomposition. *See* decomposition
- interface, 68, 93. *See also* APIs; POSIX; SPLs
- operating system. *See* operating system
- programs. *See* programs
- redesigning, for multicore processors, 1
- reliability, programmer ethics and, 377, 399

software broker, 290. See also agents

software development, 37–38, 44

- concurrency challenges. *See* challenges
- defined, 44
- ethical/moral responsibility in, 377, 399
- as group effort, 304

software development group, 304

Software Development Life Cycle (SDLC), 37–41

- activities, 38, 41
- design, 38, 85
- Guide for Developing Software Life Cycle Processes*, 287
- implementation, 38
- maintenance, 38
- multicore programming and, 37–41, 59, 64, 65
- PADL and, 287. *See also* PADL model
- PBS, PADL and, 328
- specifications, 38
- testing and evaluation, 38, 375
 - during all activities, 381

software development methodologies, 38–40

- agile, 39
- build and fix, 39
- extreme programming, 39
- incremental, 39
- languages in, 39
- linear-sequential model, 40

object-oriented, 38, 39, 40, 46, 141

rapid prototyping, 40

similarities in, 41

spiral, 40

structured, 40

waterfall, 40

software layers, 69–71. See also operating system

level 1, 69–71

level 2, 69–71

level 3, 69–71

level 4, 69–71

software resources, 130

sharing, 238–239

software system. See system

software-automated painters. See painters

solution decomposition. See decomposition

solution spaces, 295, 297, 298, 324

iterative shared, Blackboard as, 298–299

some_assertion method, 281

source state, 369

Southbridge, 21, 32

spatial locality, 12

spawn_() function. See posix_spawn() function

spawn process attributes functions, 101–102

spawn/manage processes, POSIX functions for, 97–98

adding file actions, 98

creating processes, 98

destroying attributes, 98

initializing attributes, 98

setting/retrieving attribute values, 98

specification testing, 379

specifications (SDLC activity), 38

SPEs. See Synergistic Processor Elements

spiral model, 40

SPIs (Serial Peripheral Interfaces), 32

SPLs (System Programming Interfaces), 68, 69, 70, 72, 94

SPMD (Single Program Multiple Data), 274–275

SPUs. See synergistic processor units

SRI. See System Request Interface

SS relationship. See start-to-start relationship

SSE (Streaming SIMD Extensions), 6, 32–33

stack class, 353

as interface class, 354, 355

stacks

address, 154

cleanup, 178, 179

location of, setting, 182–183

managing, 180–183

segment, 109

size, 154

setting, 181–182

- standard software engineering testing, 386–399**
- Standard Template Adaptive Parallel Library.**
See STAPL
- STAPL (Standard Template Adaptive Parallel Library), 69, 70, 76, 84, 86, 93, 94, 321–323, 567**
 - components of, 322
 - pAlgorithms, 322
 - pContainers, 322
 - pRange, 322
 - Runtime, 322
 - Views, 322
 - goal of, 70, 94
 - online information for, 323
 - Stroustrup and, 322
 - structure of (block diagram), 322–323
 - TBB and, 322
- START, 116**
- start-to-finish (SF) relationship, 57–58, 246, 265**
 - condition variables/mutexes in (listing 7–13), 266–267
- start-to-start (SS) relationship, 57–58, 245**
- STAT, 116, 118, 119**
- state(s)**
 - cancelability, 172, 173
 - composite, 369
 - defined, 365
 - detached, 154
 - diagrams, 62, 64, 366–367, 408
 - final, 367
 - initial, 367
 - parts of, 367
 - deferred events, 367, 368
 - entry/exit actions, 367, 368
 - internal transitions, 367, 368
 - name, 367
 - self-transitions, 367, 368
 - of processes, 111–114
 - runnable (ready), 111, 112, 113, 114
 - running, 111, 112, 113, 114
 - sleeping, 113
 - stopped, 112, 113, 114
 - waiting (blocked), 112, 113, 114
 - zombied, 111, 112, 113, 114
 - of processor, 109
 - source, 369
 - substates, 369–370
 - superstate, 367, 369
 - target, 369
 - thread, 156–157
- state machines, 365–370**
- state transitions, 112–113**
 - defined, 112
 - list of, 112–113
- static priority, 114**
- stereotypes, 350**
 - association, 358
 - dependency, 349–351. *See also* dependency stereotypes
 - generalization, 351
 - global, 350
 - local, 358
 - parameter, 358
 - process, 359
 - self, 358
 - thread, 359
- STIME, 116**
- stopped state, 112, 113, 114**
- 'story' of application, 323, 326, 329**
- Streaming SIMD Extensions (SSE), 6, 32–33**
- stress testing, 379**
- strings, in files, 213**
- Stroustrup, Bjarne**
 - C++ and, 62–63, 322
 - on parallelism, 62–63
 - STAPL and, 322
- struct usage attributes, 133**
- structural part, of collaboration, 356**
- structural perspective (modeling concurrent systems), 331, 334–356**
- structured model, 40**
- subclass, 349**
- substates, 367, 369–370**
- Sun C/C++ compiler, 28, 64**
- Sun UltraSparc T1 multiprocessor. *See* UltraSparc T1**
- superclass, 349**
- superstate, 367, 369**
- support/utility classes, 335**
- swimlanes, 364, 365**
- symbols**
 - decision, 364
 - self-delegation, 363
 - visibility, 343
- symmetric multiprocessor (SMP), 24**
- synchronization**
 - access, 244
 - of concurrency, 238–268
 - data, 239, 240
 - hardware, 239
 - mechanisms, 246–268, 282
 - in painters example, 204
 - problems, 56. *See also* data races; deadlocks; indefinite postponement
 - relationships, 56–58, 244–246, 282
 - condition variables and, 265–267
 - FF, 57–58, 246, 265
 - FS, 57–58, 245–246, 265
 - SF, 57–58, 246, 265

synchronization (Continued)

- SS, 57–58, 245
 - sequence, 244
 - task, 239, 282
 - types of, 240
- synchronization bar, 364**
- synchronous processes, 133–135**
- synergistic memory flow controller, 30, 31**
- Synergistic Processor Elements (SPEs), 28, 29, 30, 31**
- synergistic processor units (SPUs), 30, 31**
- sysconf() function, 26, 125, 188–190**
- system**

- architecture of, 371
 - deployment view, 371
 - design view, 371
 - implementation view, 371
 - process view, 371
 - use cases, 371
 - concurrent. *See* modeling concurrent systems
 - defined, 371
 - Booch, et al, 371
 - model of, 331, 372
 - OO, 332
 - structure of, modeling, 334–356
 - whole, visualizing, 371–372
- system calls, cancellation-safe, 177. *See also* cancellation points**
- system contention scope, 154, 157, 158**
- system libraries, 63–64**
- system main memory. *See* RAM**
- system processes, 108**
- system programming, 13, 70**
- System Programming Interfaces. *See* SPIs**
- System Request Interface (SRI), 23**
- system() function, 127, 135**
- SZ, 116**

T

- target state, 369**
- task-oriented programming. *See* imperative programming**
- tasks. *See also* processes**
- dependencies, counting, 208–210
 - multitasking, 76, 359–361
 - to processes, 74
 - synchronization, 239, 282
- task-to-task communication, 47–51**
- TBB (Intel Threading Building Blocks) library, 70, 71, 86, 317–321**
- concurrent_hash_map, 317
 - concurrent_queue, 317
 - concurrent_vector, 317
 - in listing 8–3, 318–321
 - containers, with concurrency support, 317
 - generic parallel algorithms, 317
 - parallel_for, 317
 - in listing 8–3, 318–321
 - parallel_reduce, 317
 - parallel_scan, 317
 - parallel_sort, 317
 - parallel_while, 317
 - STAPL and, 322
- template classes, 335, 348**
- realization of, 354, 355
 - visualizing, 348–349
- template object, 348**
- temporal locality, 12**
- 10 challenges of concurrency. *See* challenges**
- terminating**
- processes, 127–129
 - threads, 171–180. *See also* cancellation
- termination model, 397**
- \$TERM, 126**
- testing (multicore programming), 60–61, 375–399**
- acceptance, 380
 - concurrency challenges and, 60–61, 377–379
 - condition variables and, 378
 - debugging and, 60–61, 376, 377, 399
 - definitions/terms in, 378, 379
 - errors during, 391
 - fundamental questions and, 383, 386
 - goals of, 376
 - as fundamental questions, 383
 - guess_it program, 382–383
 - concurrency model for agents, 382
 - declarative implementation of PBS, 383
 - listing 10-1 (declarative implementation of guess_it), 383
 - listing 10-2 (valid_code predicate, declaration of), 384
 - listing 10-3 (valid_code predicate, definition of), 384–385
 - PBS of agent solution model, 383
 - problem statement, 382
 - processes before, 382–383
 - revised agent model, 382
 - rough-cut solution model, 382
 - solution model-with layer 5 PADL, 382
 - strategy, 383
- integration, 379
- issues in, 376
 - multiple condition variables and, 378
 - mutex semaphores and, 378
 - operational, 380

- read-write locks and, 378
- regression, 379
- SDLC and, 38, 375, 381
- in sequential programming, 60
- skipping, 376–377
- specification, 379
- standard software engineering, 386–399
 - IEEE, 386–399
 - PADL, PBS, three fundamental questions in, 386
- stress, 379
- terms/definitions in, 378, 379
- types of, 379–380
- unit, 379
 - activities, 390
 - phases, 390
- V&V in, 387
- text files filtering project, 276–282**
 - agent model of pipeline, 278–282
 - assert method, 281
 - assertion class, 280
 - class relationship diagram, 278
 - main line for, 279
 - operator(), 279–280
 - some_assertion method, 281
 - approaches, 277
 - communication/cooperation requirements, 278
 - goals of, 276
 - larger scale problems and, 282
 - observations in, 277
 - problem statement, 276
 - solution, 277–281
 - strategy, 276–277
- text segment, 109**
- things (UML language primitive), 332, 333**
- thread(s), 143–201**
 - anatomy, 50
 - attribute objects, 154, 155. *See also* pthread
 - attribute object
 - attributes
 - pthread attribute object, 167–171
 - setting, 153–155
 - boss, 161, 269, 271, 412, 413, 414
 - cancelable, 172
 - communication between, 51, 150–151
 - concurrency models with. *See* concurrency models
 - contention scopes, 147, 154, 157–159
 - process, 154, 157, 158
 - setting, 187–188
 - system, 157, 158
 - context, 147–149
 - creating, 162–163
 - data corruption from, 151
 - defined, 141, 143–144, 200
 - detached, 155, 169–171
 - differences, with processes, 152–153, 203
 - exhaustion, 387
 - hardware, 149
 - hybrid, 144, 145
 - ids
 - comparing, 166–167
 - getting, 166–167
 - joining, 165–166
 - kernel-level, 144–147
 - managing, 171–192
 - multithreading with, 143–201, 359–361
 - chip-level, 25, 29, 149
 - necessary number of, 59–60
 - passing arguments to, 163–165
 - peer, 150, 153
 - terminating, 172
 - POSIX, 29, 63, 82
 - POSIX Standard for Thread Management, 427–566
 - primary, 143, 159, 200
 - priority of, 155
 - setting, 183–187
 - process termination from, 151
 - processes *v.*, 50, 97, 150–153
 - resources, 149–150
 - reuse and, 151
 - safe data structures, 268
 - safety, libraries and, 71, 190–192
 - scheduling policies, 155
 - setting, 183–187
 - scheduling policies and, 155
 - similarities, with processes, 152–153, 201
 - software, 149
 - stacks of. *See* stacks
 - states, 157
 - strategies. *See* thread strategies
 - terminating, 171–180
 - cancellation process in, 172–180
 - self, 171–172
 - transitions and, 157
 - uncancelable, 172
 - unsafe, 191
 - user, 74–75
 - user-level, 144–147
 - variables for, 188–189
 - thread (stereotype), 359**
 - Thread Checker tool, 7**
 - thread checking option, 7**
 - thread interface class (C++0x standard), implementation of, 305–312**
 - thread library option, 7**
 - thread mapping, one-to-one, 146**

thread strategies, 268–274, 282

- delegation (boss/worker) model, 269–271, 289, 412–414
- peer-to-peer model, 271–272, 289, 415–416
- pipeline model, 233, 269, 273–274, 277, 289, 418
- producer-consumer model, 272–273, 289, 418–419
- WBS and, 268

threaded programs

- compiling/linking, 162
- example, 160–162

Threading Building Blocks library. See

TBB library

thread() method, 91

thread_object interface class, 87, 89, 90, 93

- declaration, 88–90
- extending, 193–200, 201
- implementation, 90–91
- as wrapper, 193, 201

thread-specific data (TSD), 148

throughput, 150

throwing exception object, 394–395

TIME, 116

time slices, 15, 60, 112, 113, 114

timed wait operation, condition variable and, 263

timing considerations, 58–59

top-down approaches

- multicore programming, 17. *See also* application design
- PADL model. *See* PADL model
- PBS. *See* PBS

tradition, 35

transaction rollback, 392

transitions, 367, 368, 369

- internal, 367, 368
- parts of
 - action, 369
 - event trigger, 369
 - guard condition, 369
 - source state, 369
 - target state, 369
- self-, 367, 368
- state, 112–113
- threads and, 157
- triggerless, 363, 369

transparency, operating system, 72, 86, 94, 105

trend, in CMPs, 3, 14, 36, 45, 65, 86, 283

triggerless transitions, 363, 369

try ownership operation

- mutex semaphores, 253
- POSIX semaphores, 249
- read-write locks, 258

TSD (thread-specific data), 148

TT, 116

TTY, 116

type, cancelability, 172, 173

type functions (pthread_mutex_t attribute object), 255

U

UID, 116

UltraSparc T1 (Sun), 25–28

- code listing, 25
- comparison, 21
- compilers, 28
- functional overview, 27
- L2 cache, 28
- listing 2–1, 25, 29
- memory controllers, 28
- processors available to operating system, 25–26
- V9 cores, 27

UMA architecture. See Uniform Memory Access architecture

UML (Universal Modeling Language), 61–62, 372–373, 401–409

- concurrent systems and, 332–333. *See also* modeling concurrent systems
- creators of, 372
- defined, 332–333
- diagrams, 373
 - for active objects, 359–360
 - activity, 62, 64, 407
 - class, 89, 401–405
 - collaboration, 62, 357–358, 405–406
 - component, 61
 - of concurrent substates, 369–370
 - deployment, 61, 62, 85
 - interaction, 405–407
 - object, 401–405
 - package, 409
 - sequence, 62, 361–362, 406
 - state/concurrent state, 62, 64, 366–367, 408
 - as UML language primitive, 333
- grammar, 333
- language primitives, 332–333

uncancelable thread, 172

unidirectional association, 349, 352

Uniform Memory Access (UMA) architecture, 24, 25

unit testing, 379

- activities, 390
- phases, 390

Universal Modeling Language. See UML

UnrecognizedWords, 336, 341, 342, 346, 351, 353, 356, 360, 361, 362

unsafe, thread, 191
unsetenv(), 127
unshared resources, 130
 «use», 350
use cases, 371
USER, 116
user id, 98
user interface errors, 390
user mode, 108, 146
user processes, 108
user stack, 109
user threads, 74–75
user-level threads, 144–147
user_thread class, 87, 89, 92
 definition for (listing 4–6), 91–92
 UML class relationship, 89
\$_USER, 126
utility/support classes, 335

V

V9 cores, 27
validation. See verification and validation
valid_code, 139–140
 predicate class
 declaration of, 384
 definition of, 384–385
variables
 ITC and, 231–233
 for threads/processes/semaphores, 188–189
vector container, 353, 354
vectorization option, 6, 33
vectorizer, 6
verification and validation (V&V), 387, 389–390
Views (STAPL), 322
virtual address space, 110, 111
virtual page frame number (VPFN), 110
visibility, of objects, 357–358
visibility property
 access specifiers, 343
 attributes/services and, 343
 symbols for, 343
visualizing
 attributes, 336–339
 classes, 336
 instances of a class, 345–347
 interface classes, 353–355
 responsibilities, 336–339
 services, 336–339
 template classes, 348–349
 whole system, 371–372

VPFN (virtual page frame number), 110
V&V (verification and validation), 387, 389–390

W

wait operation, condition variable and, 263
wait() function call, 135–137
waiting (blocked) state, 112, 113, 114
waitpid() function, 136
wakeup event, 366
waterfall model, 40
WBS. See Work Breakdown Structure
WCHAN, 116
WEXITSTATUS, 137
whole system, visualizing, 371–372
whole-part relationships, 352
WIFCONTINUED, 137
WIFEXITED, 137
WIFSIGNALED, 137
WIFSTOPPED, 137
William, Anthony, 312
WordExpertAgent, 336, 339, 340
Work Breakdown Structure (WBS), 41, 45, 268
 AAs v., 45
 defined, 45
 procedural models and, 45
 thread strategies and, 268
workpile approaches, 417
worlds, possible, 398
wrappers
 interface classes as, 86, 192, 193
 thread_object, 193, 201
 for system functions, 178
WSTOPSIG, 137
WTERMSIG, 137

X

XMM registers, 33

Z

zero
 multiplicity notation
 0 to 1, 347
 0 to infinite number, 347
zombied state, 111, 112, 113, 114