

Index



- `abort()`, 74, 75
- Access synchronization, 185
- Action/activity state, 386
- Active class, 380
- Active object, 380
- Activity, 386
 - diagram, 386
- Adapters, 373
- Adding parallel programming capabilities to C++ through PVM, 215–253
 - accessing standard input (`stdin`) and standard output (`stdout`) within PVM tasks, 251–252
 - retrieving standard output (`cout`) from child task, 252
- basic mechanics of PVM, 240–251
 - message packing and sending, 243–251
 - process management and control routines, 242–243
- classic parallelism models supported by PVM, 216–217
- PVM library for C++, 217–240
- approaches to using PVM tasks, 228–240
- combining runtime library and PVM library, 228
- compiling and linking a C++/PVM program, 221–222
- executing a PVM program as stand-alone, 222–224
- PVM preliminary requirements checklist, 224–228
- `addOnly`, 363
- Address spaces, 279
- Adornments, 371, 378
- Agent-oriented, 35
- Agent-Oriented Program. *See* AOP
- Agents, 10–11, 280, 326, 484, 486–487
 - agent class, 484–486
 - agent loop, 486–487
 - inference methods, 487–496
 - major categories of, table, 462
- Aggregation, 371, 408
- `allocbuf()`, 515

- Anonymous instance, 361
- AOP, 467
- API, 405
- Architecture, 32, 399
- Association, 369, 371
- Asynchronous, 32, 82–87, 104
- Atomic functions, 170
- Attribute, 359
 - object, 110
- Autonomy, 496–498

- basic_filebuf, 433, 451
- Basic Object Adapter. *See* BOA
- basic_streambuf, 433
- basic_stringbuf, 433
- Behavioral part, 376
- Beliefs, 471
- Beliefs Desires and Intentions (BDI)
 - model, 461
- BFS, 472
- Binary associations, 370
- Binary semaphore, 195
- blackboard, 382, 394
- Blackboard architectures using PVM,
 - threads, and C++ components, 500–537
 - activating knowledge sources using pthreads, 534–536
 - approaches to structuring the blackboard, 504–506
 - blackboard model, 501–504
 - control strategies for blackboards, 509–512
 - implementing blackboard using CORBA objects, 512–530
 - black_board interface class, 516–518
 - CORBA blackboard, example, 513–516
 - spawning knowledge sources in blackboard's constructor, 518–530
 - implementing blackboard using global objects, 530–534
 - knowledge source, anatomy of, 507–508
- black_board interface class, 515, 516–518
- Blackboards, 6, 7, 280, 282, 283, 326, 404, 415, 500–537
- Block, 390
- BOA, 318
- boa, 306
- Booch, Grady, 357, 399
- Boss-worker, 189; *See also* Producer–consumer
- Bound, 368
- Breadth First Search. *See* BFS
- Buffer classes, types of, table, 433

- Cancellability, 140
 - state, 140
 - type, 140
- Cancellation, 140, 147
 - points, 142–146
- catch block, 269
- CDS, 471, 477, 484, 485
- CGI, 306, 321, 322
- Child/subclass, 369
- Class diagram, 358
- Client-server, 9, 10, 31, 282, 291
 - paradigm, 325–326
- Clipboard, 14
- Clusters, 330, 398
- Cognitive Data Structures. *See* CDS
- Collaboration, 376
 - diagram, 377
- Common Object Request Broker Architecture. *See* CORBA
- Communication, 12
 - group, 334
- Component, 398
- Concurrency, 2–6, 13–14, 35, 88, 254, 377–397, 403–458, 533
- Concurrency-safe, 404, 420
- Concurrent, 185, 191, 255, 381

- Concurrent programming, description of, 1–20
 - basic layers of software concurrency, 13–14
 - applications, 14
 - introduction level, 13
 - object level, 14
 - routine level, 14
 - benefits of distributed programming, 8–11
 - multiagent (peer-to-peer) distributed model, 10–11
 - simplest models, 9–10
 - benefits of parallel programming, 5–8
 - PRAM model, 6–7
 - simplest parallel classification, 7–8
 - definition, 2–5
 - two basic approaches, 3–5
 - minimal effort required, 11–13
 - communication, 12
 - decomposition, 11–12
 - synchronization, 12–13
 - parallelism in C++, 15–19
 - CORBA standard, 18
 - library implementation based on standards, 18–19
 - MPI standard, 17
 - options for implementing, 15–16
 - PVM: standard for cluster programming, 17–18
 - programming environments for parallel and distributed programming, 19–20
- Concurrent Read Concurrent Write algorithm. *See* CRCW
- Concurrent Read Exclusive Write algorithm. *See* CREW
- Condition variables, 207–213
 - to manage synchronization relationships, 209–213
- contentionscope, 112, 158
- Context switching, 54–55, 101
- copy(), 443, 448
- CORBA, 14, 16, 18, 284, 293–307, 477, 503, 512–530; *See also* PVM
 - application, basic blueprint of, 302–307
 - consumer, basic anatomy of, 298–300
 - implementing blackboard using CORBA objects, 512–530
 - producer, basic anatomy of, 300–302
 - simple distributed web services using, 321–323
- CORBA: :BOA_var, 301
- CORBA: :Object_var, 312, 315
- CORBA: :ORB_init(), 299
- CORBA: :ORB_var, 299
- CORBA: :string_dup(), 316
- CosNaming: :Name, 312, 315
- CosNaming.h, 312
- Counting semaphore, 195
- Course adviser system, 376
- CRCW, 168, 191, 192, 364, 469, 509, 515, 536
- CREW, 167, 191, 192, 204, 415, 469, 502, 509, 515
- crontab, 496–498
- Data members, 463
- Data race, 25–26
- Data resources, 79
- Data segment, 40
- Data structure, 470
- Deadlock, 26–27, 34
- Decomposition, 11–12, 281–286
- decrementThreadAvailability(), 174
- Deduction, induction, and abduction, 464–465
- Delegation, 119, 120–122, 171–176
 - model, 120–122
 - for multithreading, 171–176
- Dependency, 369, 371
- Deployment, 396
 - diagram, 396

- Depth First Search. *See* DFS
- Designing components that support concurrency, 403–458
 - framework classes components, 452–457
 - interface classes, using, 405–411
 - object-oriented mutual exclusion and interface classes, 411–420
 - maintaining the stream metaphor, 420–427
 - semi-fat interfaces, 412–420
 - object-oriented pipes and FIFOs as low-level building blocks, 429–452
 - accessing anonymous pipes using `iostream_iterator`, 439–446
 - connecting pipes to `iostream` objects, 434–439
 - FIFOs (named pipes), `iostreams`, and `iostream_iterator`, 446–452
 - user-oriented classes, 427–429
- DFS, 472
- Dispatched state, 114
- Distributed, 3, 28, 29, 282, 357–402
- Distributed object-oriented programming in C++, 279–328
 - accessing objects on other address spaces, 286–298
 - Interface Definition Language (IDL), 293–298
 - IOR access to remote objects, 288–289
 - ORBs (Object Request Brokers), 289–293
 - anatomy of basic CORBA consumer, 298–300
 - anatomy of basic CORBA producer, 300–302
 - blueprint of basic CORBA application, 302–307
 - IDL compiler, 305–306
 - obtaining IOL for remote objects, 306–307
 - client/server paradigm, 325–326
 - decomposition and encapsulation, 281–286
 - communication between distributed objects, 284–285
 - error and exception handling in the distributed environment, 285–286
 - synchronization between local and remote objects, 285
 - implementation and interface repositories, 320–321
 - naming service, 307–317
 - creating naming contexts, 312–314
 - name service consumer/client, 314–317
 - object adapters, 318–320
 - simple distributed web services using CORBA, 321–323
 - trading service, 323–325
- Distributed objects, communication between, 284–285
- Distributed programming
 - benefits of, 8–11
 - techniques, 3
- Distributed shared-memory, 503
- Dividing C++ programs into multiple tasks, 37–98
 - asynchronous and synchronous processes, 82–87
 - created with `fork()`, `exec()`, `system()`, `posix_spawn()` functions, 85
 - `wait()` function call, 85–87
 - context switching, 54–55
 - creating a process, 55–73
 - identifying parent and child with process management functions, 73
 - parent-child relationship, 56–59
 - using `exec` family of system calls, 60–66
 - using `fork()` function call, 60
 - using POSIX functions to spawn processes, 66–73
 - using `system()` to spawn processes, 66

- multitasking and multithreading, 88–97
 - processes along function and object lines, 95–97
- process, anatomy of, 40–43
- process, definition of, 37–39
 - kinds of, 38
 - process control block, 38
- process resources, 76–82
 - POSIX functions to set limits, 79–82
 - types of, 78–79
- process scheduling, 46–54
 - policy, 47–49
 - setting and returning process priority, 52–54
 - using `ps` utility, 49–52
- process states, 43–46
- terminating a process, 73–76
 - `exit()`, `kill()`, and `abort()` calls, 74–76
- Dividing C++ programs into multiple threads, 99–183
 - anatomy of a thread, 108–111
 - attributes, 110–111
 - creating threads, 129–138
 - creating detached threads, 134–135
 - joining threads, 133–134
 - thread id, 133
 - using pthread attribute object, 135–138
 - dividing program into multiple threads, 171–182
 - using delegation model, 171–176
 - using peer-to-peer model, 176–177
 - using pipeline model, 177–178
 - using producer-consumer model, 178–182
 - managing threads, 138–168
 - managing a critical section, 161–168
 - managing thread's stack, 149–152
 - setting scheduling and priorities, 152–158
 - terminating, 138–149
 - using `sysconf()`, 159–161
 - pthread library, 125–126
 - simple threaded program, anatomy of, 126–129
 - compiling and linking multithreaded programs, 128–129
 - thread models, 119–125
 - delegation, 120–122
 - peer-to-peer, 122–123
 - pipeline, 123
 - product-consumer, 123–124
 - SPMD and MPMD for threads, 124–125
 - thread resources, 118–119
 - threads, definition of, 100–108
 - advantages of, 103–106
 - context requirements, 100–102
 - disadvantages of, 106–108
 - and processes, comparison, 102–103
 - thread safety and libraries, 168–171
 - thread scheduling, 111–118
 - policy and priority, 116–118
 - states, 113–114
 - and thread contention scope, 114–116

Dynamic binding polymorphism, 346

Dynamic priority, 46

Encapsulation, 281–286

`envp[]`, 62

ERCW, 168, 191, 192, 469, 509

EREW, 167, 191, 192, 364, 469, 502, 509

Error handling, exceptions, and software reliability, 254–278
 - defect definitions, 259–260
 - event diagrams, logic expressions, and logic diagrams, 275–278
 - exception handling mechanisms in C++, 269–275
 - exception classes, 270–275
 - failures in software layers and hardware components, 257–259

- Error handling, exceptions, and software reliability (*cont.*)
 - handling defects *versus* handling exceptions, 260–263
 - plans for reliability, 263–265
 - software reliability, definition of, 256–257
 - using map objects in error handling, 265–268
- Events, 390
- Exception handling mechanisms in C++, 269–275
 - exception classes, 270–275
 - deriving new classes, 273–274
 - logic_error classes, 272–273
 - protecting exception classes from exceptions, 274–275
 - runtime_error classes, 271–272
- Exceptions, 257
- Exclusive access, 77
- Exclusive Read Concurrent Write algorithm. *See* ERCW
- Exclusive Read Exclusive Write algorithm. *See* EREW
- exec(), 60, 64, 66, 73, 85, 429
- exec1(), 61, 62, 65
- exec family of system calls, 60–66
 - exec1() functions, 60–63
 - execv() functions, 63–64
 - reading and setting environment variables, 64–66
 - restrictions of execv() functions, 64
- execle(), 61, 62, 65
- execlp(), 61, 62, 65
- execv(), 63, 65
- execve(), 63, 65
- execvp(), 63, 65
- exit(), 74
- exit(), kill(), and abort() calls, 74–76
- Exit event, 390
- Explicit binding, 368
- Failure-is-not-an-option feature, 275
- Fault tolerant system, 256
- fclose(), 75
- FF, 186, 189, 209
- FIFO
 - policy, 47
 - scheduling, 49
- FIFOs, 47, 48, 99, 116, 429–452, 468
- file_action, 68
- filebuf, 352
- Files, 190
- Filtering, 396
- Final state, 391
- Finish-to-finish relationship. *See* FF
- Finish-to-start relationship. *See* FS
- First-In First-Out. *See* FIFO
- for_each(), 519, 523, 527
- fork(), 60, 66, 73, 85, 203, 429, 436
- fork-exec, 66, 429, 513, 518, 526
- Foundation for Intelligent Physical Agents (FIPA), 460
- Framework, 404
- freebuf(), 515
- frozen, 363
- FS, 186, 188, 209
- fstream, 433, 434–439
- Function object, 523
- General flow of control, 452
- Generalization, 371
- get_buffer(), 515
- getenv(), 443
- getgrgrid_r(), 169
- getgrnam_r(), 169
- getpid(), 58
- getppid(), 58
- getpriority(), 53, 54
- getpwuid_r(), 169
- getrlimit(), 81, 82
- GIOP, 284
- GNU, 437

- Hardware resources, 78–79
- has-a*, 408
- IDL, 293–298, 302, 305–306
 - compiler, 305–306
- ifstream*, 352, 433, 434–439, 447
- IIOP, 18, 284, 289, 302
- Implementation and interface repositories, 320–321
- Implementing agent-oriented architectures, 459–499
 - agent-oriented programming, definition of, 466–470
 - agents and parallel programming, 468–470
 - why agents work for distributed programming, 466–468
 - agents, definition of, 460–466
 - difference between agents and objects, 462–466
 - types of, 462
 - basic agent components, 470–478
 - cognitive data structures, 470–478
 - implementing agents in C++, 478–498
 - agent class, 484–496
 - proposition datatypes and belief structures, 478–484
 - simple autonomy, 496–498
 - multiagent systems, 498
- Implicit binding, 368
- importantOperation()*, 269
- incrementThreadAvailability()*, 174
- Inference, 472
- Inference implementation techniques, basic, table, 473
- Initial naming context, 312
- Initial state, 391
- Instance, 359
- Intelligent agents, 508
- Interface, 373, 412
 - classes, 373–374
- Interface Definition Language. *See* IDL
- Internal transition, 393
- Internet Inter-ORB Protocol. *See* IIOP
- Interoperable Object Reference. *See* IOR
- Interprocess, 99
- Inter-Process Communication. *See* IPC
- Inter-Thread Communication. *See* ITC
- IOR, 288–289, 300
- ios*, 423
- ios: :app*, 448
- iostream*, 348, 352, 405, 431–432
 - objects, 434–439
- iostream_iterator*, using, 439–452
 - operations available, table, 439
- IPC, 28, 39, 190, 420
- isQuery*, 363, 364
- istream*, 351, 352, 421, 423
- istream_iterator*, 439–446
- istringstream*, 352
- ITC, 420
- Iteration marker (*), 384
- Jacobson, Ivar, 357, 399
- Kernel mode, 38
- kill()*, 75
- Knowledge sources, 507–508
- Landscape, 308
- Last-In First-Out. *See* LIFO
- length()*, 515
- LIFO, 471
- Lightweight processes, 531
- Links, 311
- Linux, 38, 44, 46, 47, 49, 92, 100, 129, 241, 431, 446, 526
- Location tag, 397
- lock()*, 195, 407, 411, 414, 419
- Logical operators, mapping operators to, table, 480

- logic_error, 271, 272–273
- Lollipop, 374

- MAF, 498
- main(), 95
- Maintenance, 396
- Main thread, 100
- Massive Parallel Processor. *See* MPP
- maximum(), 515
- Member functions, 463
- Message passing, 30, 190, 504
- Message Passing Interface. *See* MPI
- Methods, 463
- MICO, 18, 19, 285, 305, 312, 313, 319, 321, 498, 524
- MIMD, 7, 8, 18, 216, 237–240, 502, 510, 519, 528, 534
- Minimal effort required, 11–13
 - communication, 12
 - decomposition, 11–12
 - synchronization, 12–13
- Minimal standard interface, 165
- Mini self-contained applications, 452
- MIOR, 307
- MISD, 216
- MIWCO, 307
- Modeling, 357
- MPI, 5, 16, 17, 18, 329–330, 344, 405, 412, 418, 457, 468, 477, 498
 - standard, 17
- MPI-2, 331
- MPI_ANY_SOURCE, 333
- MPI_ANY_TAG, 333
- MPICH, 17, 18, 19, 330, 344
- MPI_Comm_create(), 334
- MPI_Comm_rank(), 331
- MPI_COMM_WORLD, 332, 333, 334
- MPI_ERROR, 333
- MPI_Finalize(), 334–338
- MPI_Init(), 334–338
- MPI_Recv(), 332, 348, 352

- MPI_Send(), 332, 352
- MPI_Status, 333
- MPL, 216
- MPMD, 7, 8, 18, 124–125, 229, 237–240, 268, 280, 329–356, 510, 528
- MPP, 6, 20, 215, 239, 330, 531
- MTMD, 124
- Multiagent, 6, 32, 280, 283, 326, 498
- Multiagent Facility. *See* MAF
- Multiagent (peer-to-peer) distributed model, 10–11
- Multiagent systems, 455, 468
- Multilevel priority queue, 46
- Multiple Instruction Multiple Data. *See* MIMD
- Multiple Program Multiple Data. *See* MPMD
- Multiple Threads Multiple Data. *See* MTMD
- Multiplicity, 361
- Multiprocessing, 2
- Multiprocessor, 2, 17
- Multitasking, 88–97, 101
- Multithreaded, 35, 100, 101, 169, 180–182, 213, 531
- Multithreaded objects, creating, 180–182
- Multithreading, 88, 99–183
- Mutexes, 76, 118, 162, 169, 195–203, 213, 285, 405, 420, 509, 534
- Mutex semaphores, 195–203
 - for managing critical sections, 200–203
 - pthread_mutex_t functions, table, 196
 - using mutex attribute object, 197–200

- Name binding, 308
- Named instance, 359
- Naming context, 308
- NamingContext_var, 312
- Naming graph, 308
- Naming service, 307–317
- n-ary associations, 370

- `nice()`, 53, 58
- Nice value, 53, 54
- NLP, 229
- Nodes, 311, 396
- Nondistributed, 513

- Object adapters, 318–320
- Object-oriented classes, types of, table, 404
- Object-oriented mutual exclusion and interface classes, 411–420
- Object Request Brokers. *See* ORBs
- Object-to-Object Communication. *See* OTOC
- `ofstream`, 352, 433, 434–439, 447, 449
- OMG, 357
- `open()`, 38, 449
- `operator!()`, 479, 480
- `operator &&()`, 479, 480
- `operator()`, 347, 351–352, 523
- `operator=()`, 271
- `operator[]`, 515
- `operator||()`, 479, 480
- ORBs, 289–293, 524
- `ostream`, 351, 352, 421, 423
- `ostream_iterator`, 439–452
- `ostreamstringstream`, 352
- OTOC, 420
- Output only, 448
- `overflow_error`, 270

- Packages, 372, 400
- Page frame number, 41
- Pages, 41
- Page table, 42
- Parallel, 3, 29, 35
- Parallel access, types of, table, 509
- Parallel and distributed programming
 - challenges of, 21–36
 - architecture selection, 32–33
 - big paradigm shift, 22–23
 - communication of design, 34–35
 - coordination issues, 23–30
 - hardware and software failures, 30–31
 - negative consequences of too much parallelization or distribution, 31–32
 - required testing and debugging techniques, 33–34
 - environments for, 19–20
 - typical architecture, figure, 4
- Parallelism, 4, 5, 22, 23, 30, 88, 119, 120, 232, 330, 403, 469, 500, 510, 536
- Parallelism in C++, 15–19
 - CORBA standard, 18
 - library implementations based on standards, 18–19
 - MPI standard, 17
 - options for implementing, 15–16
 - PVM standard for cluster programming, 17–18
- Parallelization, 29, 31, 232
- Parallel programming
 - and agents, 468–470
 - benefits of, 5–8
 - techniques, 3
- Parallel Random Access Machine. *See* PRAM
- Parallel Virtual Machine. *See* PVM
- Parametric polymorphism, 346
- Parent-child, 39, 56–59, 60, 73, 103
 - relationship, 56–59
- Parent/superclass, 369
- Pattern of work, 453
- PCB, 39, 55
- Peer-to-peer, 10, 32, 119, 120, 122–123, 171, 176–177, 189, 282, 283, 291, 446
 - model, 122–123
 - for multithreading, 176–177
- PID (process id), 39
- `pipe()`, 432–433, 446
- Pipeline, 119, 120, 123, 171, 177–178
 - model, 123
 - for multithreading, 177–178

- Pipes, 28, 190, 429–452
- POA, 318
- poa, 306
- POA_black_board, 517
- Polymorphism, 268, 341–346
- Portable Object Adapter. *See* POA
- Portable Operating System Interface. *See* POSIX
- POSIX, 16, 18, 19, 28, 38, 79–82, 125–126, 147, 159, 168, 200, 412, 418
 - functions, using to spawn processes, 66–73
 - threads, 15–16, 18–19, 110, 405, 519, 531; *See also* pthreads
- posix_spawn(), 66–73, 85, 504, 513, 518, 526–530
 - activating knowledge sources using, 526–530
- posix_spawnattr_t, 68, 70, 72
- posix_spawn_file_actions, 69
- posix_spawn_file_actions_t(), 68, 72
- POSIX_SPAWN_RESETEIDS, 69
- POSIX_SPAWN_SETGROUP, 69, 70
- POSIX_SPAWN_SETSCHEDPARAM, 69, 70
- POSIX_SPAWN_SETSCHEDULER, 69, 71
- POSIX_SPAWN_SETSIGDEF, 69, 71
- POSIX_SPAWN_SETSIGMASK, 69, 71
- Postponement, 26, 34
- PPID (parent id), 39
- PRAM, 6–7, 8, 191–193, 364
- Preempted, 113–114
 - state, 114
- Preemptive, 184
- Primary thread, 100
- Problem solver, 501
- Process, 37–54, 101, 107, 108
 - creating a, 55–73
 - identification, 39
 - image, 40
 - priority, setting and returning, 52–54
 - scope, 111
 - tables, 42
 - terminating a, 73–76
 - transitions, table, 45
- Process control, 39
- Process control block. *See* PCB
- Processes, 4, 28, 34, 380
- Processor, 7
- Process resources, 76–82
 - types of, 78–79
 - using POSIX functions to set limits, 79–82
- Producer-consumer, 10, 119, 123–124, 166, 171, 178–180, 193, 264, 282, 283, 291
 - model, 123–124
 - for multithreading, 178–182
- pstree utility, 58–59
- ps utility, 49–52
 - common headers in Solaris/Linux environment, table, 50
- pthread_attr_destroy(), 135–136
- pthread_attr_detachstate(), 138
- pthread_attr_getdetachstate(), 137–138
- pthread_attr_getinheritsched(), 153
- pthread_attr_getschedparam(), 153
- pthread_attr_getschedpolicy(), 153
- pthread_attr_getscope(), 158
- pthread_attr_getstack(), 151–152
- pthread_attr_getstacksize(), 149–152
- pthread_attr_init(), 135–136, 138
- pthread_attr_set(), 136
- pthread_attr_setdetachstate(), 136–138
- pthread_attr_setinheritsched(), 153

- `pthread_attr_setschedparam()`, 153, 154
- `pthread_attr_setschedpolicy()`, 153
- `pthread_attr_setscope()`, 158
- `pthread_attr_setstacksize()`, 149–152
- `pthread_attr_t`, 135–136
- `pthread_cancel()`, 139, 144, 531
- `PTHREAD_CANCEL_ASYNCHRONOUS`, 141, 144
- `PTHREAD_CANCEL_DEFFERED`, 141, 144
- `PTHREAD_CANCEL_DISABLE`, 141
- `PTHREAD_CANCELLED`, 145
- `PTHREAD_CANCEL_ENABLE`, 141
- `pthread_cleanup_pop()`, 148–149
- `pthread_cleanup_push()`, 147–149
- `pthread_cond_broadcast()`, 531
- `pthread_cond_signal()`, 531
- `pthread_cond_t`, 207–208, 210
- `pthread_cond_wait()`, 146
- `pthread_create()`, 127–128, 129–132, 138
- `PTHREAD_CREATE_DETACHED`, 137, 138
- `PTHREAD_CREATE_JOINABLE`, 137
- `pthread_detach()`, 134–135
- `pthread_exit()`, 130, 134, 138–139, 148
- `PTHREAD_EXPLICIT_SCHED`, 153–154
- `pthread.h`, 195, 198–199, 204, 205, 208, 210
- `PTHREAD_INHERIT_SCHED`, 153
- `pthread_join()`, 127–128, 133–134, 138, 145, 146, 189, 531
- `pthread` library, 125–126
- `PTHREAD_MIN_STACK`, 150
- `Pthread` mutex, 413
- `pthread_mutexattr_t`, 407
- `pthread_mutex_destroy()`, 162
- `pthread_mutex_init()`, 162, 197, 410
- `pthread_mutex_lock()`, 162, 174, 197, 410
- `pthread_mutex_t`, 162, 195–197, 405–411
- `pthread_mutex_trylock()`, 410, 414
- `pthread_mutex_unlock()`, 162, 174, 410
- `pthread` read-write lock variable, 413
- `pthread_rwlock()`, 416
- `pthread_rwlockattr_t`, 416
- `pthread_rwlock_rdlock()`, 203
- `pthread_rwlock_t`, 203, 205, 416
- `pthread_rwlock_wrlock()`, 203
- `pthreads`, 16, 125–126, 411, 418, 531
 - activating knowledge sources using, 534–536
- `PTHREAD_SCOPE_PROCESS`, 158
- `PTHREAD_SCOPE_SYSTEM`, 158
- `pthread_self()`, 133
- `pthread_setcancelstate()`, 140–141, 142–144
- `pthread_setcanceltype()`, 140–141, 142–146
- `pthread_setschedparam()`, 156
- `pthread_setschedprio()`, 156–158
- `pthreads_join()`, 535
- `PTHREAD_STACK_MIN`, 150–151, 159
- `pthread_t()`, 133
- `pthread_testcancel()`, 141, 142–146
- `pthread_timedwait()`, 146
- Pure abstract class, 361
- PVM, 4, 16, 17–18, 19, 30, 215, 217–252, 405, 412, 418, 421, 457, 467, 498, 504, 507
 - basic mechanics of, 240–252
 - accessing stdin and stdout within tasks, 251–252
 - message packing and sending, 243–251
 - process management and control routines, 242–243
 - classic models supported by, 216–217

- PVM (*cont.*)
- group routines, table, 521
 - PVM library for C++, 217–240
 - pvm_addhosts(), 243
 - PVM_ARCH, 222, 224–226
 - pvm_barrier(), 520–521
 - pvm_bufinfo(), 246
 - PVM_CPLX, 249
 - pvmcmd, 240, 252
 - PvmDataDefault, 245, 251
 - PvmDataInPlace, 245, 251
 - PvmDataRaw, 245, 251
 - pvm_delhosts(), 243
 - pvm_exit(), 219, 220, 223, 242, 526
 - PVM_FLOAT, 249
 - pvm_freebuf(), 251
 - pvm_getrbuf(), 250
 - pvm_getsbuf(), 250
 - pvm_halt(), 243
 - pvm_hostfile, 227
 - pvm_hosts, 227
 - pvm_initsend(), 219, 245, 246
 - pvm_joingroup(), 520–521
 - pvm_kill(), 242
 - PVM_LONG, 249
 - pvm_mcast(), 246
 - pvm_mkbuf(), 251
 - pvm_mytid(), 219, 223
 - pvm_nrecv(), 248
 - pvm_parent(), 223
 - pvm_pk, 244
 - pvm_pk, 423, 425
 - pvm_pkbyte(), 235
 - pvm_pkdouble(), 235
 - pvm_pkfloat(), 422
 - pvm_pkint(), 422
 - pvm_pkstr(), 219, 220, 245
 - pvm_precv(), 248
 - pvm_probe(), 250
 - pvm_psend(), 247
 - pvm_recv, 424, 425
 - pvm_recv(), 219, 220, 247, 248
 - PVM_ROOT, 224
 - pvm_send, 231, 240, 423, 425
 - pvm_send(), 219, 220, 235, 247
 - pvm_setopt(), 252
 - pvm_setrbuf(), 250
 - pvm_setsbuf(), 250
 - PVM_SHORT, 249
 - pvm_spawn(), 219, 231, 238, 239, 242, 429, 523
 - PVM_STR, 249
 - pvm_stream, 407
 - pvm_trecv(), 248, 249
 - PVM_UINT, 249
 - pvm_upk, 424, 425
 - pvm_upkfloat(), 219, 220
- Race conditions, 190
- rand_r(), 169
 - range_error, 270
 - Raw, 245
 - read(), 38
 - readdir_r(), 169
 - Read lock, 203, 415
 - Read/write, 415
 - Read-write algorithms, four basic, table, 7
 - Read-write locks, 195, 203–207, 213
 - to implement access policy, 204–207
 - Realization, 374
 - Refinement, 368
 - release(), 515
 - Remote Method Invocation. *See* RMI
 - replace(), 515
 - Resource allocation graph, 78
 - resources, values for, table, 81
 - Responsibility of the class, 359
 - Resumption, 265
 - model, 264–265
 - rlim_cur, 80
 - RLIM_INFINITY, 81
 - rlimit, 80

- RLIMIT_FSIZE, 81
- rlim_max, 80
- RLIM_SAVED_CUR, 81
- RLIM_SAVED_MAX, 81
- rlim_t, 80
- RMI, 284
- Root/base class, 369
- Round-robin. *See* RR
- RR, 47, 48, 116
 - policy, 47
 - scheduling, 49
- rsh, 226, 227
- rtymame_r(), 169
- Rumbaugh, James, 357, 399
- Runnable, 44
- Running state, 391
- runtime_error, 271–272
- rusage, 83
- r_usage, 82
- RUSAGE_CHILDREN, 82
- RUSAGE_SELF, 82

- sched_get_priority_max(), 154
- sched_get_priority_min(), 154
- sched_param, 154
- Schedule product, 396
- Scheduling, 22, 116–118
- Self-delegation, 384
 - symbol, 384
- Self-transition, 393
- Semantic networks, 311
- Semaphores, 76, 162, 193–213, 285, 412, 413, 418, 419, 420, 509
 - condition variables, 207–213
 - and mutexes, differences between, table, 420
 - mutex semaphores, 195–203
 - operations, 194–195
 - read-write locks, 203–207
- Sequence, 383, 515, 525
 - diagram, 383
 - synchronization, 185
- Servant objects, 319
- Service, 359
- setenv(), 66, 442
- setpriority(), 53, 54
- setrlimit(), 80, 81, 82
- SF, 186, 188–189, 209
- Sharable, 77
- Shared libraries, 79
- SIMD, 7, 18, 124, 216, 229–237, 528
- Simple name, 359
- Simultaneously, 25, 162
- Single Instruction Multiple Data. *See* SIMD
- Single Program Multiple Data. *See* SPMD
- Single Thread Multiple Data. *See* STMD
- SISD, 216
- Sleeping state, 44, 390, 391
- SMP, 20, 330, 531
- Socket, 330
- Software concurrency, layers of, 13–14
 - applications level, 14
 - instruction level, 13
 - object level, 14
 - routine level, 14
- Software resources, 79
- Spawned, 96
- Spawning, 56
- Spawn N, 510
- Spawns, 56, 57, 93
- SPMD, 7, 8, 18, 124–125, 216, 229–237, 268, 280, 329–356, 510, 528
- SPMD and MPMD using templates and MPI, 329–356
 - simplifying MPI communications, 348–356
 - using template functions, 338–347
 - adding MPMD with function objects, 346–347
 - instantiating templates and SPMD (datatypes), 339–340
 - using polymorphism to implement MPMD, 340–346

- SPMD and MPMD using templates and MPI (*cont.*)
 - work breakdown structure for MPI, 330–338
 - anatomy of MPI task, 334, 337–338
 - differentiating tasks by rank, 331–333
 - grouping tasks by communicators, 334, 335–336
- SS, 186, 187–188, 209
- SSH, 323
- ssh, 226
- SSL, 323
- Stack segment, 40
- Start-to-finish relationship. *See* SF
- Start-to-start relationship. *See* SS
- Starvation, 117
- State, 392
 - access policies, table, 469
 - diagram, 390
 - parts of, table, 392
 - transition, 43
- State of the processor, 39
- States, 372
- Static priority, 46
- Stereotypes, 371–373, 378
- `sterror_r()`, 169
- STMD, 124
- Stopped state, 45
- `streambuf`, 352
- `stringbuf`, 352
- `strtok_r()`, 169
- Structural part, 376
- Subclass, 369
- Substate, 394–396
- Subtask, 23
- Superclass, 369
- Superstate, 394
- Superstate/composite state, 394
- Swimlanes, 388, 389
- SWI-Prolog, 518
- Synchronization, 12–13, 25, 31, 34, 107, 185, 186, 209–213, 387
 - bar, 387
 - Synchronizing concurrency between tasks, 184–214
 - coordinating order of execution, 185–189
 - finish-to-finish (FF) relationship, 189
 - finish-to-start (FS) relationship, 188
 - relationships between synchronized tasks, 186–187
 - start-to-finish (SF) relationship, 188–189
 - start-to-start (SS) relationship, 187–188
 - semaphores, 193–213
 - condition variables, 207–213
 - mutex semaphones, 195–203
 - operations, 194–195
 - read-write locks, 203–207
 - synchronization as object-oriented, 213
 - synchronizing access to data, 189–193
 - PRAM model, 191–193
- Synchronous, 82–87
- `sysconf()`, 159
 - using, 159–161
- `system()`, 66, 85
 - using, 66
- System architecture, visualizing, 399–401
- System deployment, visualizing, 398–399
- System process, 38
- `takeCorrectiveAction()`, 273
- Task synchronization, 185
- TCP/IP, 284, 290, 302
- Template, 330, 345, 366–369, 408
 - classes, 366–369
- `templateclass C`, 479
- `templateT`, 338–340
- Termination model, 264–265
- Testing, types of, used during software development, table, 261

- `threadAvailability()`, 174
- Thread information block, 110
- Thread models, table, 120
- Threads, 4, 28, 30, 88, 100, 380
 - advantages and disadvantages, 103–108
 - creating, 129–138
 - managing, 138–168
 - and processes, 102–103
 - differences, 102–103
 - table, 104
 - threads controlling other threads, 103
- `throw()`, 274
- Trading service, 323–325
- Transition, parts of, table, 394
- Transitions, 386, 388, 393–394
- Transportation context, 308
- Triggerless, 386
 - transition, 394
- try block, 269
- `trylock()`, 407, 414, 419
- TTS, 229

- UML, 35, 185, 255, 357, 359, 369, 380
 - diagrams, table, 35
- Unbound, 368
- `underflow_error`, 270
- Unified Modeling Language. *See* UML
- UNIX, 16, 19, 129, 431, 446, 526
- `unlock()`, 162, 169, 195, 407, 411, 414, 419
- `unsetenv()`, 66
- Unshared, 77
- User mode, 38
- User process, 38

- Virtual, 41
- Visibility, 165, 365
- Visualizing concurrent and distributed system design, 357–402
 - concurrent behavior, 377–397
 - activities of objects, 386–390
 - collaborating objects, 377–383
 - distributed objects, 396–397
 - message sequences between objects, 383–386
 - state machines, 390–396
 - structures, 358–377
 - classes and objects, 358–369
 - organization of interactive objects, 374–377
 - relationship between classes and objects, 369–374
 - whole system, 397–401
 - architecture, 399–401
 - deployment of systems, 398–399

- `wait()`, 85–87, 127–128, 189
- `waitpid()`, 86
- Wakeup, 390
- WAP WDP, 307
- WBM, 282
- WBS, 11–12, 32, 119, 229, 255, 466, 500
- WCHAN, 50
- WCONTINUED, 86
- wCORBA, 307
- wCorba, 307
- WNOHANG, 87
- Work Breakdown Structure. *See* WBS
- `write()`, 38
- Write lock, 203
- WUNTRACED, 86

- XDR, 245
- XPVM, 223

- Zombied state, 46



Register Your Book

at www.awprofessional.com/register

You may be eligible to receive:

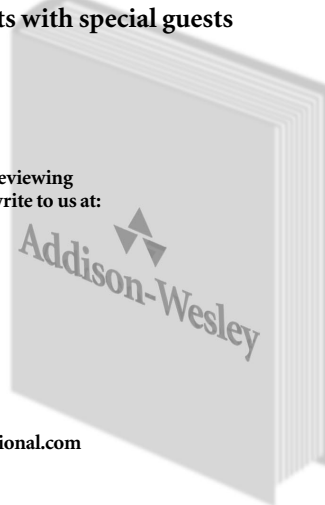
- Advance notice of forthcoming editions of the book
- Related book recommendations
- Chapter excerpts and supplements of forthcoming titles
- Information about special contests and promotions throughout the year
- Notices and reminders about author appearances, tradeshows, and online chats with special guests



Contact us

If you are interested in writing a book or reviewing manuscripts prior to publication, please write to us at:

Editorial Department
Addison-Wesley Professional
75 Arlington Street, Suite 300
Boston, MA 02116 USA
Email: AWPro@aw.com



Visit us on the Web: <http://www.awprofessional.com>



informIT

www.informit.com

YOUR GUIDE TO IT REFERENCE



Articles

Keep your edge with thousands of free articles, in-depth features, interviews, and IT reference recommendations – all written by experts you know and trust.



Online Books

Answers in an instant from **InformIT Online Book's** 600+ fully searchable on line books. Sign up now and get your first 14 days **free**.

POWERED BY
Safari



Catalog

Review online sample chapters, author biographies and customer rankings and choose exactly the right book from a selection of over 5,000 titles.

