



ОГЛАВЛЕНИЕ

Предисловие	13
Благодарности	15
Об этой книге	17
Об иллюстрации на обложке	21
ГЛАВА 1. Здравствуй, параллельный мир!	22
1.1. Что такое параллелизм?	23
1.1.1. Параллелизм в вычислительных системах	23
1.1.2. Подходы к организации параллелизма	26
1.2. Зачем нужен параллелизм?	29
1.2.1. Применение параллелизма для разделения обязанностей	29
1.2.2. Применение параллелизма для повышения производительности	30
1.2.3. Когда параллелизм вреден?	32
1.3. Параллелизм и многопоточность в C++	33
1.3.1. История многопоточности в C++	34
1.3.2. Поддержка параллелизма в новом стандарте	35
1.3.3. Эффективность библиотеки многопоточности для C++	36
1.3.4. Платформенно-зависимые средства	37
1.4. В начале пути	38
1.4.1. Здравствуй, параллельный мир	38
1.5. Резюме	40
ГЛАВА 2. Управление потоками	41
2.1. Базовые операции управления потоками	41
2.1.1. Запуск потока	42
2.1.2. Ожидание завершения потока	45
2.1.3. Ожидание в случае исключения	46
2.1.4. Запуск потоков в фоновом режиме	48
2.2. Передача аргументов функции потока	51
2.3. Передача владения потоком	54
2.4. Задание количества потоков во время выполнения	58
2.5. Идентификация потоков	61
2.6. Резюме	64

ГЛАВА 3. Разделение данных между потоками 65

3.1. Проблемы разделения данных между потоками.....	66
3.1.1. Гонки.....	68
3.1.2. Устранение проблематичных состояний гонки.....	69
3.2. Защита разделяемых данных с помощью мьютексов.....	70
3.2.1. Использование мьютексов в C++.....	71
3.2.2. Структурирование кода для защиты разделяемых данных.....	73
3.2.3. Выявление состояний гонки, внутренне присущих интерфейсам.....	74
3.2.4. Взаимоблокировка: проблема и решение.....	83
3.2.5. Дополнительные рекомендации, как избежать взаимоблокировок.....	86
3.2.6. Гибкая блокировка с помощью <code>std::unique_lock</code>	94
3.2.7. Передача владения мьютексом между контекстами.....	95
3.2.8. Выбор правильной гранулярности блокировки.....	97
3.3. Другие средства защиты разделяемых данных.....	100
3.3.1. Защита разделяемых данных во время инициализации.....	100
3.3.2. Защита редко обновляемых структур данных.....	105
3.3.3. Рекурсивная блокировка.....	107
3.4. Резюме.....	108

ГЛАВА 4. Синхронизация параллельных операций 110

4.1. Ожидание события или иного условия.....	111
4.1.1. Ожидание условия с помощью условных переменных.....	112
4.1.2. Потокбезопасная очередь на базе условных переменных.....	115
4.2. Ожидание одноразовых событий с помощью механизма будущих результатов.....	121
4.2.1. Возврат значения из фоновой задачи.....	122
4.2.2. Ассоциирование задачи с будущим результатом.....	125
Передача задач между потоками.....	127
4.2.3. Использование <code>std::promise</code>	129
4.2.4. Сохранение исключения в будущем результате.....	131
4.2.5. Ожидание в нескольких потоках.....	133
4.3. Ожидание с ограничением по времени.....	136
4.3.1. Часы.....	137
4.3.2. Временные интервалы.....	138
4.3.3. Моменты времени.....	140
4.3.4. Функции, принимающие таймаут.....	142
4.4. Применение синхронизации операций для упрощения кода.....	144

4.4.1. Функциональное программирование с применением будущих результатов	145
4.5. Резюме	156

ГЛАВА 5. Модель памяти C++ и атомарные операции 158

5.1. Основы модели памяти	159
5.1.1. Объекты и ячейки памяти	159
5.1.2. Объекты, ячейки памяти и параллелизм	161
5.1.3. Порядок модификации	162
5.2. Атомарные операции и типы в C++	163
5.2.1. Стандартные атомарные типы	163
5.2.2. Операции над <code>std::atomic_flag</code>	167
5.2.3. Операции над <code>std::atomic<bool></code>	170
5.2.4. Операции над <code>std::atomic<T*></code> : арифметика указателей....	173
5.2.5. Операции над стандартными атомарными целочисленными типами	175
5.2.6. Основной шаблон класса <code>std::atomic<></code>	175
5.2.7. Свободные функции для атомарных операций	177
5.3. Синхронизация операций и принудительное упорядочение	180
5.3.1. Отношение синхронизируется-с	182
5.3.2. Отношение происходит-раньше	183
5.3.3. Упорядочение доступа к памяти для атомарных операций... 185	
5.3.4. Последовательности освобождений и отношение синхронизируется-с	208
5.3.5. Барьеры	212
5.3.6. Упорядочение неатомарных операций с помощью атомарных	214
5.4. Резюме	216

ГЛАВА 6. Проектирование параллельных структур данных с блокировками 218

6.1. Что понимается под проектированием структур данных, рассчитанных на параллельный доступ?	219
6.1.1. Рекомендации по проектированию структур данных для параллельного доступа	220
6.2. Параллельные структуры данных с блокировками	222
6.2.1. Потокобезопасный стек с блокировками	222
6.2.2. Потокобезопасная очередь с блокировками и условными переменными	226
6.2.3. Потокобезопасная очередь с мелкогранулярными блокировками и условными переменными	231
6.3. Проектирование более сложных структур данных с	

блокировками.....	245
6.3.1. Разработка потокобезопасной справочной таблицы с блокировками	246
6.3.2. Потокобезопасный список с блокировками	253
6.4. Резюме	258

ГЛАВА 7. Проектирование параллельных структур данных без блокировок 260

7.1. Определения и следствия из них.....	261
7.1.1. Типы неблокирующих структур данных.....	262
7.1.2. Структуры данных, свободные от блокировок	262
7.1.3. Структуры данных, свободные от ожидания	263
7.1.4. Плюсы и минусы структур данных, свободных от блокировок	264
7.2. Примеры структур данных, свободных от блокировок	266
7.2.1. Потокобезопасный стек без блокировок	266
7.2.2. Устранение утечек: управление памятью в структурах данных без блокировок	271
7.2.3. Обнаружение узлов, не подлежащих освобождению, с помощью указателей опасности	277
7.2.4. Нахождение используемых узлов с помощью подсчета ссылок	287
7.2.5. Применение модели памяти к свободному от блокировок стеку.....	293
7.2.6. Потокобезопасная очередь без блокировок.....	299
7.3. Рекомендации по написанию структур данных без блокировок.....	313
7.3.1. Используйте <code>std::memory_order_seq_cst</code> для создания прототипа	314
7.3.2. Используйте подходящую схему освобождения памяти..	314
7.3.3. Помните о проблеме ABA.....	315
7.3.4. Выявляйте циклы активного ожидания и помогайте другим потокам.....	316
7.4. Резюме	317

ГЛАВА 8. Проектирование параллельных программ 318

8.1. Методы распределения работы между потоками	319
8.1.1. Распределение данных между потоками до начала обработки	320
8.1.2. Рекурсивное распределение данных	322
8.1.3. Распределение работы по типам задач.....	327
8.2. Факторы, влияющие на производительность параллельного кода.....	330

8.2.1. Сколько процессоров?.....	331
8.2.2. Конкуренция за данные и перебрасывание кэша	333
8.2.3. Ложное разделение	335
8.2.4. Насколько близки ваши данные?.....	336
8.2.5. Превышение лимита и чрезмерное контекстное переключение	337
8.3. Проектирование структур данных для повышения производительности многопоточной программы	338
8.3.1. Распределение элементов массива для сложных операций	339
8.3.2. Порядок доступа к другим структурам данных	342
8.4. Дополнительные соображения при проектировании параллельных программ.....	344
8.4.1. Безопасность относительно исключений в параллельных алгоритмах.....	344
8.4.2. Масштабируемость и закон Амдала	353
8.4.3. Соккрытие латентности с помощью нескольких потоков...	355
8.4.4. Повышение скорости реакции за счет распараллеливания...	356
8.5. Проектирование параллельного кода на практике.....	359
8.5.1. Параллельная реализация <code>std::for_each</code>	359
8.5.2. Параллельная реализация <code>std::find</code>	362
8.5.3. Параллельная реализация <code>std::partial_sum</code>	369
8.6. Резюме	380

ГЛАВА 9. Продвинутое управление потоками ... 382

9.1. Пулы потоков	383
9.1.1. Простейший пул потоков	383
9.1.2. Ожидание задачи, переданной пулу потоков.....	386
9.1.3. Задачи, ожидающие других задач.....	391
9.1.4. Предотвращение конкуренции за очередь работ	394
9.1.5. Занимание работ	396
9.2. Прерывание потоков	401
9.2.1. Запуск и прерывание другого потока	402
9.2.2. Обнаружение факта прерывания потока	404
9.2.3. Прерывание ожидания условной переменной.....	405
9.2.4. Прерывание ожидания <code>std::condition_variable_any</code>	409
9.2.5. Прерывание других блокирующих вызовов	411
9.2.6. Обработка прерываний.....	412
9.2.7. Прерывание фоновых потоков при выходе из приложения.....	413
9.3. Резюме	415

ГЛАВА 10. Тестирование и отладка

многопоточных приложений 416

10.1. Типы ошибок, связанных с параллелизмом	417
--	-----

10.1.1. Нежелательное блокирование	417
10.1.2. Состояния гонки	418
10.2. Методы поиска ошибок, связанных с параллелизмом ...	420
10.2.1. Анализ кода на предмет выявления потенциальных ошибок.....	420
10.2.2. Поиск связанных с параллелизмом ошибок путем тестирования	423
10.2.3. Проектирование с учетом тестопригодности	425
10.2.4. Приемы тестирования многопоточного кода	427
10.2.5. Структурирование многопоточного тестового кода.....	431
10.2.6. Тестирование производительности многопоточного кода....	435
10.3. Резюме	436

ПРИЛОЖЕНИЕ А. Краткий справочник по некоторым конструкциям языка C++ 437

A.1. Ссылки на <i>r</i> -значения	437
A.1.1. Семантика перемещения.....	439
A.1.2. Ссылки на <i>r</i> -значения и шаблоны функций.....	442
A.2. Удаленные функции	442
A.3. Умалчиваемые функции	445
A.4. <code>constexpr</code> -функции	449
A.4.1. <code>constexpr</code> и определенные пользователем типы.....	450
A.4.2. <code>constexpr</code> -объекты	454
A.4.3. Требования к <code>constexpr</code> -функциям	454
A.4.4. <code>constexpr</code> и шаблоны	455
A.5. Лямбда-функции	456
A.5.1. Лямбда-функции, ссылающиеся на локальные переменные ...	458
A.6. Шаблоны с переменным числом параметров	461
A.6.1. Расширение пакета параметров	463
A.7. Автоматическое выведение типа переменной.....	466
A.8. Поточно-локальные переменные	467
A.9. Резюме	469

ПРИЛОЖЕНИЕ В. Краткое сравнение библиотек для написания параллельных программ 470

ПРИЛОЖЕНИЕ С. Каркас передачи сообщений и полный пример программы банкомата 472

ПРИЛОЖЕНИЕ D. Справочник по библиотеке C++ Thread Library..... 492

D.1. Заголовок <code><chrono></code>	492
D.1.1. Шаблон класса <code>std::chrono::duration</code>	493

D.1.2. Шаблон класса <code>std::chrono::time_point</code>	503
D.1.3. Класс <code>std::chrono::system_clock</code>	506
D.1.4. Класс <code>std::chrono::steady_clock</code>	508
D.1.5. Псевдоним типа <code>std::chrono::high_resolution_clock</code>	510
D.2. Заголовок <code><condition_variable></code>	511
D.2.1. Класс <code>std::condition_variable</code>	511
D.2.2. Класс <code>std::condition_variable_any</code>	521
D.3. Заголовок <code><atomic></code>	530
D.3.1. <code>std::atomic_xxx</code> , псевдонимы типов	531
D.3.2. <code>ATOMIC_xxx_LOCK_FREE</code> , макросы	532
D.3.3. <code>ATOMIC_VAR_INIT</code> , макрос	533
D.3.4. <code>std::memory_order</code> , перечисление	533
D.3.5. <code>std::atomic_thread_fence</code> , функция	534
D.3.6. <code>std::atomic_signal_fence</code> , функция	535
D.3.7. <code>std::atomic_flag</code> , класс	535
D.3.8. Шаблон класса <code>std::atomic</code>	539
D.3.9. Специализации шаблона <code>std::atomic</code>	552
D.3.10. Специализации <code>std::atomic<integral-type></code>	552
D.4. Заголовок <code><future></code>	571
D.4.1. Шаблон класса <code>std::future</code>	572
D.4.2. Шаблон класса <code>std::shared_future</code>	578
D.4.3. Шаблон класса <code>std::packaged_task</code>	585
D.4.4. Шаблон класса <code>std::promise</code>	592
D.4.5. Шаблон функции <code>std::async</code>	598
D.5. Заголовок <code><mutex></code>	600
D.5.1. Класс <code>std::mutex</code>	601
D.5.2. Класс <code>std::recursive_mutex</code>	603
D.5.3. Класс <code>std::timed_mutex</code>	606
D.5.3. Класс <code>std::recursive_timed_mutex</code>	611
D.5.5. Шаблон класса <code>std::lock_guard</code>	615
D.5.6. Шаблон класса <code>std::unique_lock</code>	617
D.5.7. Шаблон функции <code>std::lock</code>	628
D.5.8. Шаблон функции <code>std::try_lock</code>	629
D.5.9. Класс <code>std::once_flag</code>	630
D.5.10. Шаблон функции <code>std::call_once</code>	630
D.6. Заголовок <code><ratio></code>	631
D.6.1. Шаблон класса <code>std::ratio</code>	632
D.6.2. Псевдоним шаблона <code>std::ratio_add</code>	633
D.6.3. Псевдоним шаблона <code>std::ratio_subtract</code>	634
D.6.4. Псевдоним шаблона <code>std::ratio_multiply</code>	635
D.6.5. Псевдоним шаблона <code>std::ratio_divide</code>	635
D.6.6. Шаблон класса <code>std::ratio_equal</code>	636
D.6.7. Шаблон класса <code>std::ratio_not_equal</code>	636
D.6.8. Шаблон класса <code>std::ratio_less</code>	637

D.6.9. Шаблон класса <code>std::ratio_greater</code>	637
D.6.10. Шаблон класса <code>std::ratio_less_equal</code>	638
D.6.11. Шаблон класса <code>std::ratio_greater_equal</code>	638
D.7. Заголовок <code><thread></code>	638
D.7.1. Класс <code>std::thread</code>	639
D.7.2. Пространство имен <code>this_thread</code>	649
РЕСУРСЫ	652
Печатные ресурсы	652
Сетевые ресурсы	653
ПРЕДМЕТНЫЙ УКАЗАТЕЛЬ	654



ПРЕДИСЛОВИЕ

С идеей многопоточного программирования я столкнулся на своей первой работе после окончания колледжа. Мы занимались приложением, которое должно было помещать входные записи в базу данных. Данных было много, но все они были независимы и требовали значительной предварительной обработки. Чтобы задействовать всю мощь нашего десятипроцессорного компьютера UltraSPARC, мы организовали несколько потоков, каждый из которых обрабатывал свою порцию входных данных. Код был написан на языке C++, с использованием потоков POSIX. Ошибок мы наделали кучу – многопоточность для всех была внове – но до конца все-таки добрались. Именно во время работы над этим проектом я впервые услышал о комитете по стандартизации C++ и о недавно опубликованном стандарте языка C++.

С тех мой интерес к многопоточному программированию и параллелизму не затухает. Там, где другим видятся трудности и источник разнообразных проблем, я нахожу мощный инструмент, который позволяет программе использовать всё наличное оборудование и в результате работать быстрее. Позднее я научился применять эти идеи и при наличии всего одного процессора или ядра, чтобы улучшить быстроту реакции и повысить производительность, – благодаря тому, что одновременная работа нескольких потоков дает программе возможность не простаивать во время таких длительных операций, как ввод/вывод. Я также узнал, как это устроено на уровне ОС и как в процессорах Intel реализовано контекстное переключение задач.

Тем временем интерес к C++ свел меня с членами Ассоциации пользователей C и C++ (ACCU), а затем с членами комиссии по стандартизации C++ при Институте стандартов Великобритании (BSI) и разработчиками библиотек Boost. Я с интересом наблюдал за началом разработки библиотеки многопоточности Boost, а когда автор забросил проект, я воспользовался шансом перехватить инициативу. С тех пор разработка и сопровождение библиотеки Boost Thread Library лежит в основном на мне.

По мере того как в работе комитета по стандартизации C++ наметился сдвиг от исправления дефектов в существующем стандарте в сторону выработки предложений для нового стандарта (получившего условное название C++0x в надежде, что его удастся завершить до 2009 года, и официально названного C++11, так как он наконец был опубликован в 2011 году), я стал принимать более активное участие в деятельности BSI и даже вносить собственные предложения. Когда стало ясно, что многопоточность стоит на повестке дня, я по-настоящему встрепенулся – многие вошедшие в стандарт предложения по многопоточности и параллелизму написаны как мной самим, так и в соавторстве с коллегами. Я считаю большой удачей, что таким образом удалось совместить две основных сферы моих интересов в области программирования – язык C++ и многопоточность.

В этой книге, опирающейся на весь мой опыт работы с C++ и многопоточностью, я ставил целью научить других программистов, как безопасно и эффективно пользоваться библиотекой C++11 Thread Library. Надеюсь, что мне удастся заразить читателей своим энтузиазмом.