
Блокирующая очередь - 5 баллов

Реализуйте блокирующую очередь фиксированной емкости с помощью условных переменных.

Очередь должна быть шаблонной и передавать данные произвольного типа: BlockingQueue<T>

Емкость очереди (capacity) должна задаваться в конструкторе.

Операции:

- void put(T item) - метод, который принимает объект, который необходимо поместить в очередь, если очередь не выключена. Если очередь выключена, то необходимо бросить исключение, унаследованное от std::exception. Если превышена capacity контейнера необходимо заблокироваться и ждать до тех пор пока не появится свободное место или пока очередь не выключат.

- bool get(T& item**)** - должен переместить по ссылке в аргументе первый объект, который лежит в очереди, достать его из контейнера и вернуть true. Если очередь выключена, то необходимо вернуть false (по ссылке тогда можно ничего не писать).

- void shutdown() - выключает очередь, разблокируя все ждущие потоки и запрещает запись в очередь.

Пул потоков - 5 баллов

Очередь задач (task queue) или пул потоков (thread pool) - это распространенный паттерн многопоточного программирования для асинхронного исполнения задач.

Под асинхронной задачей здесь понимается любая функция, замыкание или функтор, которые запускаются в отдельном потоке и не блокируют на время своего исполнения поток, который ставит эту задачу.

Замечание: понятие асинхронности в общем случае чуть сложнее и не привязано к многопоточности. Но в контексте пула потоков нам будет достаточно и такого определения.

Как выглядит работа с очередью задач?

Для начала передаем задачу на выполнение в пул потоков:

```
auto future = thread_pool.submit(async_task);
```

Здесь future - это placeholder для результата асинхронного вычисления, которое, возможно, пока еще не завершилось.

Дальше пул потоков запускает полученный async_task в одном из своих выделенных потоков. При этом выполнение потока-клиента не блокируется на время выполнения задачи, он может продолжать работу.

Когда потоку понадобится результат асинхронного вычисления, то он может попросить у своего асинхронного результата вернуть ему вычисленное значение:

```
auto result_value = future.get();
```

Вызов этого метода - точка синхронизации потока-клиента и асинхронного вычисления. Если задача уже завершилась к моменту вызова, то get() сразу вернет результат. Если же задача еще выполняется, то метод get() заблокирует клиентский поток до окончания выполнения задачи.

Что представляет из себя пул потоков изнутри? Это фиксированное число выделенных потоков-воркеров (worker threads), исполняющих пользовательские задачи, и очередь, по которой задачи отправляются к воркерам.

Потоки-воркеры не создаются на каждую задачу: воркер, завершивший выполнение одной задачи, тут же принимается за следующую, если она есть, либо блокируется на ожидании.

Задачи поступают воркерам через потокобезопасную очередь, что обеспечивает порядок их выполнения: чем раньше задача была отправлена в пул, тем раньше она начнет выполняться.

Общая очередь балансирует нагрузку между воркерами: если в пуле появился свободный поток-воркер, то он сразу пытается взять из очереди очередную задачу и выполнить ее.

Зачем вообще нужен пул из фиксированного числа потоков, если для каждой асинхронной задачи можно просто запустить отдельный поток и выполнить задачу в нем (тем более, такой механизм в стандартной библиотеке уже есть и называется std::async)?

Во-первых, запуск потока - это системный вызов и последующая работа в ядре операционной системы, временные расходы на которую могут превышать время выполнения самой асинхронной задачи. В

таком случае никакого выигрыша от асинхронного исполнения не будет.

Пул потоков не запускает новый поток на каждую задачу, а создает потоки в начале работы и потом переиспользует их для запуска новых задач.

Во-вторых, если запускать новую задачу каждый раз в отдельном потоке, то, вероятно, число одновременно запущенных потоков скоро станет большим и превысит число аппаратных ресурсов (ядер процессора). Такая ситуация называется переподпиской (oversubscription) и плоха она тем, что планировщик операционной системы, стараясь быть честным (fair), будет постоянно прерывать одни потоки и переключать их на другие (preemption), т.к. на всех не хватает ядер.

Каждое такое переключение (context switch) - это существенные накладные расходы (переключение в режим ядра, работа планировщика, сохранение/восстановление содержимого регистров), в результате та же самая работа по выполнению задач происходит медленнее, чем при меньшем числе потоков.

Таким образом, пул потоков позволяет переиспользовать потоки и ограничивать число одновременно бегущих задач.

Указания по реализации:

1) Для передачи задач в пул потоков (они могут быть обычными функциями, замыканиями или врапперами, которые вернул std::bind) вам пригодится std::function - универсальная обертка, которая может оборачивать произвольные callable-объекты.

http://en.cppreference.com/w/cpp/utility/functional/function

2) Как запускать в пуле функции, которые принимают на вход аргументы?

На самом деле это не нужно, пулу достаточно принимать только функции без аргументов. Если клиент хочет добавить в пул функцию f с аргументами x и y, то он просто заворачивает этот вызов с помощью std::bind(f, x, y). В результате получается объект-враппер с оператором () без аргументов, который внутри вызывает f(x, y). А дальше этот объект уже можно передавать в пул потоков:

```
thread_pool.submit( std::bind(f, x, y) );
```

При желании можно сделать дополнительный метод, который будет принимать функцию и список ее аргументов с помощью: http://en.cppreference.com/w/cpp/language/parameter_pack

3) Пулу достаточно запускать задачи без аргументов, для остальных случаев есть std::bind. Так что единственная степень свободы - тип результата, который возвращает задача. Этим типом можно параметризовать очередь задач: thread_pool<ResultType>

4) Для возврата асинхронного результата клиенту из потока-воркера нужно использовать механизм future/promise - простой асинхронный канал для публикации/чтения значения.

Скринкаст: http://www.youtube.com/watch?v=o0pCft99K74&index=4
http://bartoszmilewski.com/2009/03/03/broken-promises-c0x-futures/

Внутри пула задача будет представлять собой пару из std::function (сама функция, которую нужно запустить потоку-воркеру) и std::promise, с помощью которого воркер опубликует результат вызова этой функции (или сообщит, что было брошено исключение).

Поток-клиент при добавлении задачи получит std::future, с помощью которого сможет получить асинхронный результат, который опубликует поток-воркер.

```
thread_pool<R> tasks(...);
std::future<R> async_result = tasks.submit( [](){ return 42; } );
R result_value = async_result.get();
```

Альтернативный (и более универсальный) вариант - использовать std::packaged_task, который инкапсулирует в себе задачу и канал для возврата асинхронного результат + имеет специализацию для случая, когда пользовательская задача возвращает void.

5) При создании пула пользователь должен иметь возможность указать число потоков-воркеров. Также должна быть возможность создать пул без указания числа воркеров, тогда пул потоков сам должен подбирать разумное число воркеров с помощью std::thread::hardware_concurrency

A) thread_pool tasks{4};

B) thread_pool tasks; // число потоков-воркеров подбирается автоматически

Учтите, что std::thread::hardware_concurrency может вернуть 0, в таком случае нужно использовать разумное значение по умолчанию.

Чтобы не писать всю эту логику по подбору числа воркеров прямо в конструкторе, заведите отдельный метод default_num_workers()

Шаблонный класс, параметризуемый типом возвращаемого значения: thread_pool<R> tasks

Запуск задачи:

std::future<R> thread_pool<R>::submit(std::function<R> func)

У пула должен быть удаленный конструктор копирования и оператор присваивания, в многопоточном мире операция копирования не имеет смысла.

У пула потоков должен быть публичный метод shutdown. Вызов должен завершаться только тогда, когда все потоки-воркеры завершили свою работу.

При разрушении пул должен корректно останавливать все потоки-воркеры (вызывая shutdown).

Быстрая сортировка - 5 баллов

Реализуйте quicksort в пуле потоков.
