Advanced Thread Pool

Manual

Терминология

Известно, что создание, уничтожение и переключение между потоками – это дорогостоящие операции. Для того чтобы избежать накладных расходов, связанных с этим, используется паттерн проектирования «Пул потоков».

Реализация *AdvThreadPool* данного паттерна определяет следующие сущности и их множества:

* Ядро процессора (Core) – вычислительный модуль процессора, производящий все вычисления параллельно другим ядрам; *AdvThreadPool* программным образом определяет количество ядер, а затем распределяет по ним создаваемые потоки. Это достигается двумя способами – автоматическим (при помощи самого планировщика ОС), или же вручную (с указанием AffinityMask для каждого потока через графический интерфейс пула).
* Поток (Thread) – логическая сущность, выполняемая ядром процессора. Операционная система выделяет для нее процессорное время, а также гарантирует сохранение контекста при его переключении. Контекст потока содержит все необходимые данные для возобновления выполнения (включая набор регистров процессора и стек) в адресном пространстве приложения. Наилучший вариант – один поток на одно ядро для минимизации времени, которое затрачивается на переключение контекста.

*AdvThreadPool* определяет для потоков два логических типа – приватный (unshared thread, «неразделяемый поток») и общий (shared thread):

* Приватный поток предназначен для запуска в своем контексте длинных задач и не имеет поддержки очереди задач, в силу очевидной бессмысленности этой идеи.
* Общий поток предназначен для запуска коротких и перезапускаемых задач, то есть тех, которые имеют короткий жизненный цикл. В силу этого, согласно классической идее паттерна, общий поток поддерживает очередь задач, которую разгребает по мере необходимости, то есть после завершения предыдущей задачи.
* Задача (Task) – часть кода, оформленная в виде функции и переданная на пул посредством паттерна «команда», для выполнения в контексте одного из освободившихся потоков;

*AdvThreadPool* имеет три типа задач – длинные, короткие и перезапускаемые.

* Длинная задача (long task) – подразумевает занятие приватного потока (unshared thread) для неопределенно долгого выполнения на нем; удаляется из пула при насильственном прекращении задачи (например, останов задачи оператором).
* Короткая задача (short task) – используется для выполнения какой-либо единовременной процедуры в контексте отдельного общего потока (shared thread); удаляется из пула по своему завершению.
* Перезапускаемая задача (repeat task) – подобна короткой, с одним исключением – она не удаляется из очереди задач по своему завершению, а напротив отслеживается специальным системным потоком пула, для перезапуска на одном из освободившихся потоков по истечении заданного интервала времени. Этот интервал указывается при создании задачи, но также может изменяться в момент ее завершения. Для этого задача должна возвратить новое значение интервала.
* Запускаемый объект (runnable object) – объект-обертка для задачи, хранящий ее основные параметры, и передаваемый на пул потоков для последующего исполнения задачи в контексте одного из потоков.

Использование

* Запуск пула

Подключить к проекту библиотеку AdvThreadPool.a (pathToSrc – путь к исходникам проекта, pathToLib – путь к папке lib):

INCLUDEPATH += pathToSrc /AdvThreadPool \

pathToSrc/AdvThreadPool/ui

LIBS += -L./pathToLib

LIBS += -lAdvThreadPool

Подключить в исходниках заголовочный файл:

#include "AdvThreadPool.h"

Запустить пул потоков при инициализации программы. Указать при этом количество общих (shared thread) и приватных (unshared thread) потоков, режим работы пула (pool mode) и путь к файлу с настройками. Режим работы пула может быть USE\_REPEATED\_TASK\_MODE или ORDINARY\_MODE. В первом случае будут использоваться повторяемые задачи и запущен системный поток, во втором случае – пул работает только с длинными и короткими задачами, и системный поток, соответственно, не запускается.

#define \_QUANTITY\_SHARED\_THREADS 10

#define \_QUANTITY\_UNSHARED\_THREADS 4

CAdvThreadPool::startThreadPool(

\_QUANTITY\_SHARED\_THREADS,

\_QUANTITY\_UNSHARED\_THREADS,

eThreadPoolMode::USE\_REPEATED\_TASK\_MODE,

"ThreadPoolSettings.ini");

Если для модуля требуется графический интерфейс, его можно создать следующим образом:

CAdvPoolGUI\* m\_pDlgThreadPool = new CAdvPoolGUI(this);

m\_pDlgThreadPool->createThreadPoolShell(); //create link between thread pool and it's dialog

m\_pDlgThreadPool->show();

Внешний вид GUI представлен в пункте 7.

* Создание задачи типа «Long Task»

Для всех типов задач требуется подключить заголовочный файл:

#include "Runnable.h"

Объявляем указатель на запускаемый объект (runnable), инкапсулирующий выполняемый код задачи:

CRunnable<CTaskClass,int> \*m\_pTask;

Здесь CTaskClass и int – шаблонные параметры, причем CTaskClass – класс, содержащий выполняемую задачу (функцию), а int – тип возвращаемого этой функцией значения.

Выделяем память на запускаемый объект, используя полиморфный шаблонный класс CLongTask. При этом используем те же параметры шаблона, что и при объявлении указателя на запускаемый объект. 1-ым аргументом конструктора является указатель на экземпляр класса, содержащего функцию-задачу. 2-ой аргумент – указатель на функцию-задачу. 3-ий аргумент – указатель на функцию, устанавливающую флаг выполнения задачи в значение false (соответственно, вызов данной функции ведет к останову длинной задачи); 4-ый аргумент – строковое описание длинной задачи.

m\_pTask = new CLongTask<CTaskClass, int>(

objectPointer,

&CTaskClass::StartRunnable,

&CTaskClass::StopRunnable,

QString("Task description"));

Завершение задачи должно произойти после вызова функции:

objectPointer->StopRunnable();

Также пул может прибегнуть к этой функции в случае экстренного завершения приложения, без предварительного останова задач на уровне пользовательского кода. В этом случае, он дернет указанную функцию для каждой из выполняемых задач и, дождавшись их завершения, позволит приложению завершиться.

* Создание задачи типа «Short Task»

Короткие и перезапускаемые задачи могут принимать на вход аргументы, запакованные в кортеж (если требуется). Рассмотрим пример, когда задача имеет следующие три параметра – указатель на char (l\_pchMessageBuff), и два целочисленных значения (l\_iMessageSize и l\_iType). Тогда формируем кортеж из указанных аргументов.

auto \_tuple = std::make\_tuple<>(

(char\*)l\_pchMessageBuff,

(int)l\_iMessageSize,

(int)l\_iType);

Выделяем память на запускаемый объект, используя полиморфный шаблонный класс CShortTask.

typedef TaskReturnType bool;

auto m\_pTask = new CShortTask<CTaskClass, TaskReturnType, char\*, int, int> (

objectPointer,

&CTaskClass::FunctionTask,

\_tuple);

Здесь используются следующие параметры шаблона: 1-ый параметр – класс, содержащий функцию-задачу; 2-ой параметр – тип возвращаемой переменной; последние параметры – типы данных, упакованных в кортеже, для дальнейшей передачи в задачу.

Параметры конструктора: 1-ый - указатель на экземпляр класса, содержащего задачу; 2-ой – указатель на функцию-задачу; 3-ий – кортеж с упакованными данными.

Замечание: если сигнатура функции-задачи не имеет входных параметров, то надобность в кортеже отпадает.

* Создание задачи типа «Repeat Task»

Для примера создадим перезапускаемую задачу без входных параметров:

int l\_iTimeInterval = 1000;//msc

auto m\_pTask = new CRepeatTask<CTaskClass, int>(

objectPointer,

&CTaskClass::processingAssignments,

l\_iTimeInterval,

"Task description");

Здесь параметры шаблона такие же, как и в предыдущем случае. К аргументам конструктора добавились: интервал перезапуска задачи и строковый дескриптор задачи (для отображения в GUI). Переданный временной интервал в дальнейшем может корректироваться самой задачей – для этого по завершении она должна возвратить пулу новое значение интервала. Если новое значение интервала равно -1 (минус один), пул прекращает выполнение указанной перезапускаемой задачи, если значение интервала равно 0 – пул оставляет предыдущее значение интервала перезапуска для задачи.

Ниже представлен механизм передачи перезапускаемой задачи на пул, а также способ ее останова при необходимости:

auto future = CAdvThreadPool::launchRunnableObject<int,

CRepeatTask<CTaskClass, int>>(m\_pTask);

CAdvThreadPool::getInstance().stopRunnable\_RepeatTask(future->res\_task\_ID);

Выше приведенная функция демонстрирует останов перезапускаемой задачи по ее идентификатору, полученному из объекта future.

* Передача задачи на пул потоков

Вызываем статическую шаблонную функцию пула потоков launchRunnableObject(), и передаем ей на вход указатель на запускаемый объект.

auto future = CAdvThreadPool::launchRunnableObject<int,

CRunnable<CTaskClass, int>>(m\_pTask);

Замечание 1: здесь мы видим, что параметром шаблона может быть, как базовый класс CRunnable, так и его наследник (CLongTask, CShortTask, CRepeatTask)

Замечание 2: если задача принимает параметры (кортеж), то передача запускаемого объекта будет выглядеть, например, так:

CAdvThreadPool::launchRunnableObject<int,

CTaskClass <CTaskClass, int, char\*, int, int>>(m\_pTask);

Где последние параметры шаблона, исключая первые два, как раз и есть типы, описывающие параметры кортежа.

Функция launchRunnableObject возвращает объект «будущего», в котором содержится идентификатор задачи. По данному идентификатору можно, в частности, прекратить выполнение перезапускаемой задачи.

* Завершение работы пула потоков

При завершении работы приложения необходимо остановить пул потоков:

CAdvThreadPool::stopThreadPool();

Все потоки пула будут остановлены. При этом вызовется функция StopRunnable() для каждой длинной задачи и, дождавшись завершения всех задач, пул передаст управление приложению для последующего закрытия.

* GUI AdvThreadPool

Ниже представлен графический интерфейс пула потоков, который позволяет оператору гибко настроить модуль: указать количество общих и приватный потоков, указать привязку потоков по ядрам процессора (если требуется). Все настройки пула сериализуются в конфигурационный xml-файл, который также может правиться оператором.

