УДК 004.94

МЕТОДЫ СИНХРОНИЗАЦИИ ПОТОКОВ В МНОГОПОТОЧНОМ ПРИЛОЖЕНИИ НА ЯЗЫКЕ C++

Раздобудов С.А., Мартышкин А.И.
Пензенский государственный технологический университет, Пенза, e-mail: Alexey314@yandex.ru

В статье приводятся методы синхронизации потоков в многопоточном приложении на языке C++. Основная возникающая трудность при написании программ для параллельных систем – синхронизация одновременно работающих потоков. Приводится довольно подробный обзор публикаций по теме исследования. Выделяют 4 общих типа синхронизации потоков в одном процессе или процессов в одном приложении: старт-старт, финиш-старт, старт-финиш и финиш-финиш. С помощью этих основных типов отношений можно описать координацию задач между потоками и процессами. Известен ряд программных инструментов, которые могут помочь программисту защитить разделяемые данные и сделать код потокобезопасным. Их называют примитивами синхронизации, среди которых наиболее распространенные – мьютексы, семафоры, условные переменные и спин-блокировки. Приводится подробное описание указанных примитивов синхронизации. Большую роль в многопоточном программировании играет, собственно, синхронизация потоков. В любой программе, функционирующей с несколькими потоками, есть ряд ресурсов, которые могут в конкретный момент времени работать только с единственным потоком. Невозможно пытаться избежать использования таких ресурсов в программе, а потому приходится задуматься над тем, как наиболее эффективно организовать совместную работу потоков для того, чтобы они не мешали друг другу. Основные методы синхронизации потоков работают так: пока один поток работает с ресурсом, другим потокам доступ к этому ресурсу закрывается. В конце работы приводятся основные результаты.

Ключевые слова: операционная система, многопоточность, язык программирования, многозадачность, процессор, поток, параллелизм, компилятор

METHODS FOR SYNCHRONIZING THREADS IN MULTITHREADED APPLICATION IN C++

Razdobudov S.A., Martyshkin A.I.
Penza State Technological University, Penza, e-mail: Alexey314@yandex.ru

This article provides methods for synchronizing threads in a multithreaded C++application. The main difficulty that arises when writing programs for parallel systems is synchronization of concurrently running threads. A detailed review of the publications on the subject of the study is provided. There are four common types of thread synchronization in one process or processes in one application: start-start, finish-start, start-finish and finish-finish. These basic types of relationships can be used to describe the coordination of tasks between threads and processes. There are a number of software tools that can help a programmer protect shared data and make code thread-safe. They are called synchronization primitives, among which the most common are mutexes, semaphores, conditional variables, and spin locks. A detailed description of these synchronization primitives is provided. Synchronization of threads plays an important role in multithreaded programming. In any program that operates with multiple threads, there are a number of resources that can only work with a single thread at a given time. It is impossible to avoid the use of such resources in the program, and therefore we have to think about how to most effectively organize the joint work of flows in order that they do not interfere with each other. The basic methods for synchronizing threads work like this: while one thread is working on a resource, other threads are denied access to that resource.

Keywords: operating system, multithreading, programming language, multitasking, processor, thread, parallelism, compiler

С появлением вычислительной техники человек постоянно стремился повысить ее производительность. Это может быть достигнуто через непрерывное совершенствование аппаратной части вычислительной техники, путем повышения скорости ее работы. Или же через усовершенствование программного обеспечения.

Одной из главных задач современных операционных систем является эффективное использование ресурсов компьютера. При помощи многопоточности мы можем выделить в приложении несколько потоков, которые будут выполнять различные задачи одновременно.

Многопоточный режим работы предоставляет новые возможности программистам, за которые приходится расплачиваться усложнением процесса проектирования приложения и отладки. Основная возникающая трудность – синхронизация одновременно работающих потоков.

Все дело в том, что, как правило, в любой многопоточной программе есть ряд ресурсов, которые могут в данный момент времени работать только с одним-единственным потоком. Эти ресурсы могут быть разными. Это могут быть файлы, коллекции объектов, последовательные порты и многое другое. Пытаться избежать использования ресурсов такого рода в своей программе невозможно, а потому приходится озаботиться тем, как наиболее эффективно организовать совместную работу потоков для того, чтобы они не мешали друг другу. Как правило, это решается очень просто: пока один поток ра-

ботает с каким-либо ресурсом, другим потокам доступ к этому ресурсу закрывается. Вот именно на этой простой идее и основана синхронизация потоков.

На сегодняшний день, объем литературы, написанной по языку программирования C++, невероятно огромен. Каждый найдет для себя подходящий учебник по языку, вне зависимости от его уровня владения им. Стоит отметить книгу [1], написанную непосредственно автором языка C++ и являющуюся наиболее авторитетным и каноничным изложением возможностей, предоставляемых языком программирования. На страницах этой книги найдутся доказавшие свою эффективность подходы к решению разнообразных задач проектирования и программирования, а также примеры, демонстрирующие как высокий стиль программирования на ядре C++, так и современный объектно-ориентированный подход к созданию программных продуктов.

Учебник [2] представляет необходимые сведения для работы на многопроцессорной системе. В нем уделяется большое внимание практическим вопросам создания параллельных программ.

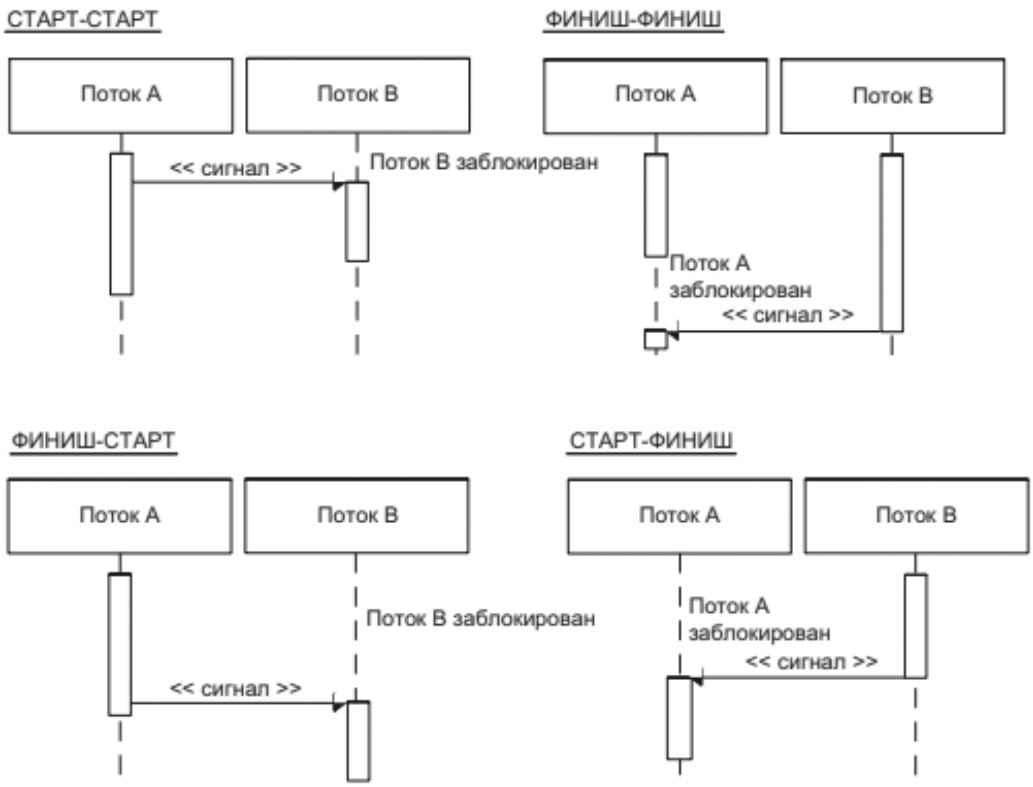
Книга [3] рассказывает о поддержке многопоточности в C++. Она включает в себя описания библиотеки потоков, atomics-библиотеки, модели памяти C++, блокировок и мьютексов (взаимных исключений) вместе с распространенными проблемами дизайна и отладки многопоточных приложений.

Также значимый вклад в понимание многопоточности и многопоточного программирования привносят online курсы, которые без труда можно найти в интернете.

Так, в курсах «Принцип многопоточного программирования» и «Основы разработки на C++» рассматривается в контексте разработки сетевых и высоконагруженных систем. Главной целью данных курсов является обучение межпроцессному взаимодействию (IPC) и синхронизации потоков.

Методы синхронизации необходимы в случае, когда имеет место параллелизм – ситуация, при которой одновременно выполняется два или более потока, имеющие возможность потенциального взаимодействия друг с другом.

Выделяют 4 общих типа (рисунок) синхронизации любых двух потоков в одном процессе или любых двух процессов в одном приложении: старт-старт, финиш-старт, старт-финиш и финиш-финиш. С помощью этих базовых типов отношений можно описать координацию задач между потоками и процессами.



Возможные варианты синхронизации потоков/процессов

Синхронизация типа старт-старт. Одна задача может начаться раньше другой, но не позже.

Синхронизация типа финиш-старт задача А не может завершиться до тех пор, пока не начнется задача В. Этот тип отношений типичен для процессов типа родитель-потомок.

Синхронизация типа старт-финиш может считаться обратным вариантом синхронизации типа финиш-старт.

Синхронизация типа финиш-финиш. Одна задача не может завершиться до тех пор, пока не завершится другая, т.е. задача А не может финишировать до задачи В.

Существует ряд программных инструментов, которые могут помочь разработчику защитить разделяемые данные и сделать код потокобезопасным, их называют примитивами синхронизации, среди которых наиболее распространенные – мьютексы, семафоры, условные переменные и спин-блокировки [4–7]. Все они защищают часть кода, давая только определенному потоку право получать доступ к данным и блокируя остальные.

Семафор – это переменная особого типа, которая может изменяться с положительным или отрицательным приращением, но обращение к переменной в ответственный момент всегда атомарно даже в многопоточных программах. Это означает, что, если два или несколько потоков в программе пытаются изменить значение семафора, система гарантирует, что все операции будут на самом деле выполняться одна за другой.

Семафоры лучше всего представлять себе как счетчики, управляющие доступом к общим ресурсам. Чаще всего они используются как блокирующий механизм, не позволяющий одному процессу захватить ресурс, пока этим ресурсом пользуется другой.

Мьютекс (взаимоисключение, mutex) – примитив синхронизации, устанавливающийся в особое сигнальное состояние, когда не занят каким-либо потоком. Только один поток владеет этим объектом в любой момент времени, отсюда и название таких объектов – одновременный доступ к общему ресурсу исключается.

Задача мьютекса – обеспечить такой механизм, чтобы доступ к объекту в определенное время был только у одного потока. Если поток 1 захватил мьютекс объекта А, остальные потоки не получат к нему доступ, чтобы что-то в нем менять. До тех пор, пока мьютекс объекта А не освободится, остальные потоки будут вынуждены ждать.

C++ предоставляет нам 3 типа операций над базовыми мьютексами [8]:

1. lock – если мьютекс не принадлежит никакому потоку, тогда поток, вызвавший lock, становится его обладателем. Если же некий поток уже владеет мьютексом, то текущий поток (который пытается овладеть им) блокируется до тех пор, пока мьютекс не будет освобожден и у него не появится шанса овладеть им.

2. try_lock – если мьютекс не принадлежит никакому потоку, тогда поток, вызвавший try_lock, становится его обладателем и метод возвращает true. В противном случае возвращает false. try_lock не блокирует текущий поток.

3. unlock – освобождает ранее захваченный мьютекс.

Условные переменные, позволяют блокировать один или более потоков, пока либо не будет получено уведомление от другого потока, либо не произойдет «ложное/случайное пробуждение».

Есть две реализации условных переменных, доступных в заголовке:

1. condition_variable: требует от любого потока перед ожиданием сначала выполнить std::unique_lock;

2. condition_variable_any: более общая реализация, которая работает с любым типом, который можно заблокировать. Эта реализация может быть более дорогим (с точки зрения ресурсов и производительности) для использования, поэтому ее следует использовать только если необходима те дополнительные возможности, которые она обеспечивает.

Спин-блокировки представляют собой чрезвычайно низкоуровневое средство синхронизации, предназначенное в первую очередь для применения в многопроцессорной конфигурации с разделяемой памятью. Они обычно реализуются как атомарно устанавливаемое булево значение. Аппаратура поддерживает подобные блокировки командами вида «проверить и установить».

При попытке установить спин-блокировку, если она захвачена кем-то другим, как правило, применяется активное ожидание освобождения, с постоянным опросом в цикле состояния блокировки. Естественно, при этом занимается процессор, так что спин-блокировки следует устанавливать только на очень короткое время и их владелец не должен приостанавливать свое выполнение.

Основным достоинством и одновременно недостатком спин-блокировки является отсутствие переключения контекста выполнения, однако при этом происходит активное использование процессора что не всегда оправдано.

В заключение можно сказать, что многопоточность играет огромную роль в современном программировании. Эта тема настолько объемна и многогранна, что охватить ее всю не представляется возможным.

Немалую роль в многопоточном программировании играет синхронизация потоков. В любой многопоточной программе есть ряд ресурсов, которые могут в данный момент времени работать только с одним-единственным потоком. Невозможно пытаться избежать использования таких ресурсов в своей программе, а потому приходится озаботиться тем, как наиболее эффективно организовать совместную работу потоков для того, чтобы они не мешали друг другу.

Большинство методов синхронизации потоков сводится к одной простой схеме: пока один поток работает с каким-либо ресурсом, другим потокам доступ к этому ресурсу закрывается. Именно благодаря данной концепции и появились такие программные примитивы как мьютексы, семафоры, условные переменные и спин-блокировки.

Список литературы

1. Старченко А.В., Данилкин Е.А., Лаева В.И., Проханов С.А. Практикум по методам параллельных вычислений. – М.: Изд-во Московского университета, 2010. – 200 с.

2. Страуструп Б. Язык программирования C++ / Бином, 2015. – 1136 с.

3. Хьюз К., Хьюз Т. Параллельное и распределенное программирование с использованием C++, 2004. – 672 с.

4. Бикташев Р.А., Мартышкин А.И. Комплекс программ для измерения производительности функций операционных систем // XXI век: итоги прошлого и проблемы настоящего плюс. – 2013. – № 10 (14). – С. 190–197.

5. Мартышкин А.И. Математическое моделирование и оценка производительности средств синхронизации процессов в многопроцессорных системах // Наукові записки Міжнародного гуманітарного університету: статті учасників другої міжнародної мультидисциплінарної конференції, 2016. – С. 151-157.

6. Мартышкин А.И. Планирование и диспетчеризация задач в высокопроизводительных системах: монография. – Пенза: Издательство Пензенского государственного технологического университета, 2016. – 135 с.

7. Мартышкин А.И., Мартенс-Атюшев Д.С., Полетаев Д.А. Методы планирования и модели для оценки производительности средств синхронизации взаимодействующих процессов в параллельных вычислительных системах // Современные социально-экономические процессы: проблемы, закономерности, перспективы: сборник статей III Международной научно-практической конференции: в 2 ч., 2017. – С. 37-40.

8. Уильямс Э. Параллельное программирование на C++ в действии. Практика разработки многопоточных программ / ДМК Пресс, 2014. – 672 с.