А. А. Кондратович

АНАЛИЗ ПРОБЛЕМ «LOCK-FREE» АЛГОРИТМОВ В СОВРЕМЕННЫХ ИНФОРМАЦИОННЫХ СИСТЕМАХ

Программирование без блокировок — это подход к разработке алгоритмов и структур данных, которые не нуждаются в блокировке или мьютексах. Когда разным потокам необходимо получить доступ к одним и тем же данным, необходимо убедиться, что эти данные всегда находятся в целостном состоянии.

Введение

Мьютексы —- отличный инструмент, но при ипользовании блокировок возникают проблемы. Основная из них —- неэффективное использование вычислительный ресурсов. Альтернативный подход предлагает использовать атомарные операций при написании параллельных алгоритмов и структур данных. По определению, так называемый «lock-free» алгоритм гарантирует, что время между двумя подряд идущими завершенными операциями ограничено сверху некоторой величиной, которая не зависит от того, как операционная система эти потоки распределит во времени. Вместо использования блокировок, «lock-free» алгоритмы используют команду, известную как «compare and swap» (CAS).

Рассмотрим основные трудности, которые возникают при разработке алгоритмов и структур данных без блокировок.

I. Потеря модификаций

Иначе эта проблема называется ABA. Она возникает, когда ячейка памяти читается дважды, оба раза прочитано одинаковое значение, и признак «значение одинаковое» трактуется как «ничего не менялось». Другой поток может выполниться между этими двумя чтениями, поменять значение, сделать что-нибудь ещё, и восстановить старое значение. Первый поток может продолжать работу, но его поведение будет неправильным из-за скрытых изменений памяти, которые он не отслеживал. Обычное обходное решение -— использовать дополнительные биты для проверки, сколько раз указатель был изменён.

II. Не «lock-free»

Очень часто в разработке используются сторонние библиотеки, которые не всегда являются lock-free. Более того, на большинстве платформ работа по выделению и освобождению памяти также не является lock-free. Чтобы решить проблему с заведением памяти, можно просто не заводить память во время работы алгоритма. Другим решением является использовать нестандарные механизмы управления памятью, лишенные этой проблемы.

III. Одновременный доступ к памяти

При изменении участка памяти не гарантируется, что другие потоки увидят модификацию сразу. Причина в том, что синхронизация таких изменения между кешем процессора и общей памятью, иначе называемая «безопасной публикацией», является весьма тяжелой операцией. При параллельной работе с памятью, чтобы гарантировать корректность данных, в различных платформах используются различные подходы. В их основе лежит использование барьеров памяти.

IV. Порядок обращений к памяти

Хорошо известно, что как компилятор, так и процессор могут поменять порядок выполнения команд. Хотя внутри одного потока об этом можно не беспокоиться, в многопоточной среде результаты операций, произведённых другими потоками, могут наблюдаться не в том порядке. Подобные ситуации очень трудно обнаружить. В настоящее время большинство платформ имеют определенные спецификации, касающиеся модели памяти, которые призваны добиться предсказуемости поведения программного кода.

Выводы

«Lock-free» алгоритм гарантирует, что система всегда будет выполнять полезную работу. Но такой алгоритм не гарантирует, что это будет достигнуто эффективно. При необходимости использовать lock-free алгоритмы нужно убедиться, что они того стоят с точки зрения производительности и сложности реализации.

1. Alexandrescu A. – Lock-Free Data Structures [Электронный ресурс]. Минск, 2004. – Режим доступа: http://bit.ly/14TFyoB. – Дата доступа: 21.03.2013.
2. Kristoffersen J. – Atomic operations with multithreaded environments. [Электронный ресурс]. Минск, 2010. – Режим доступа: http://bit.ly/14TFAwv. – Дата доступа: 21.03.2013.

Кондратович Андрей Александрович, магистрант кафедры интеллектуальных информационных технологий БГУИР, andrew.kondratovich@gmail.com.

Научный руководитель: Голенков Владимир Васильевич, заведующий кафедрой интеллектуальных информационных технологий БГУИР, доктор технических наук, профессор, golen@bsuir.by.