УДК 004.272

Адаптивный алгоритм операции захвата мьютекса

А. В. Гайдай, И. И. Кулагин

В данной исследовательской работе затрагивается проблема производительности операции захвата мьютекса при возникновении конфликтной ситуации (contention): когда уже захваченным мьютексом пытаются завладеть другие потоки. Такие ситуации негативно сказываются на времени выполнения операции захвата. Реализованный программный пакет «mutex-optimizer» и его функциональные возможности, включают в себя средства профилирования и оптимизации многопоточных программ, благодаря которым достигается увеличение производительности при использовании мьютексов. Эффективность разработанного пакета исследовалось на синтетических тестах (microbenchmark). Результаты представлены в презентации.

Ключевые слова: мьютекс, конфликт, профилирование, оптимизация, многопоточность.

1. Введение

Многопоточность (multi-threading) – одна из наиболее интересных и актуальных тем в области информационных технологий. Актуальность этой темы особенно велика, в связи с широким распространением многоядерных процессоров и их стремительном развитии. Трудно себе представить современные отрасли производства, науки, экономики и др. без многопоточных приложений.

Параллельный процесс имеет несколько управляющих потоков (thread) – особого рода параллельные процессы, выполняемые в том же адресном пространстве, что и процесс-родитель. Каждый поток имеет свой набор значений регистров и собственный стек. В многопоточных приложениях появляется ряд сложностей, связанных с синхронизацией одновременного доступа потоков к одним и тем же участкам памяти. Для решения этих проблем существуют примитивы синхронизации такие как семафоры и мьютекса. В данной работе рассматриваются мьютексы.

Мьютекс (mutex, mutual exclusion – взаимное исключение) – примитив синхронизации, позволяющий создавать в коде программы критические секции, выполнение которых возможно только одним потоком в каждый момент времени. Мьютексы могут находиться в одном из двух состояний – открытом или закрытом. При выполнении потоком операции захвата (mutex lock), мьютекс переводится в закрытое состояние – становится недоступен для захвата другими потоками. Когда поток освобождает мьютекс (mutex unlock), его состояние становится открытым – доступным для захвата.

Однако, при попытке захвата мьютекса может возникать ситуация, называемая конфликтом (contention) [1], при которой несколько потоков пытаются одновременно захватить мьютекс. Операция захвата мьютекса в случае возникновения конфликта требует больше времени, чем операция захвата без возникновения конфликтной ситуации за счёт негативных явлений процессорного кэша. Мьютекс является обычной переменной, которая кэшируется, используясь на разных ядрах. Таким образом, согласно протоколу согласования кэшей MESI, при одновременном обращении нескольких потоков к одному мьютексу, фактически происходит одновременный доступ к одной и той же области памяти и, как следствие, используемые данные становятся невалидными для других ядер. Повышение производительности операции

захвата мьютекса достигается за счёт уменьшения накладных расходов при работе с критическими секциями.

Промышленным стандартом реализации мьютексов при разработке параллельных программ является динамическая библиотека pthread в GNU libc, однако, реализация в текущей версии (glibc 2.23) не учитывает динамически изменяющееся состояние конфликта при захвате мьютекса. Поэтому в данной работе реализован адаптивный алгоритм захвата мьютекса, который учитывает динамически изменяющиеся характеристики критических секций.

2. Адаптивный алгоритм захвата мьютекса

При конкуренции потоков в конечном итоге только один из них сможет захватить мьютекс, а остальные будут вынуждены либо продолжать пытаться захватить мьютекс в цикле (spinlock блокировка), либо, не тратить процессорное время, а освободить вычислительные ресурсы для выполнения другого потока и перейти в состояние ожидания. Обычно каждый конфликтующий поток сначала пытается захватить мьютекс с помощью spinlock'а и только в случае неудачи перестаёт расходовать временные ресурсы процессора (или ядра) погружаясь в «сон» на определённое время. Очевидно, что чем больше количество конфликтующих потоков, тем менее эффективно будет расходоваться процессорное время. Однако, если сократить количество попыток захвата мьютекса, перед тем как отправить тот или иной поток «спать», общее время простоя также сократиться. Таким образом для оптимизации процесса использования мьютексов необходима некоторая статистика возникновения возможных конфликтных ситуаций при работе с ним. Данная информация – это результат предварительного профилирования.

Реализованный алгоритм захвата мьютекса можно разделить на два этапа:

1) Профилирование.

2) Оптимизация.

2.1. Этап профилирования

На этапе профилирования производится подсчёт конфликтных ситуаций, возникающих при захвате мьютекса. Вся информация записывается в хэш-таблицу, где ключом является адрес конкретного мьютекса, а значением – частота возникновения искомой ситуации для него.

Ниже приведён код простой многопоточной программы:

```
#include <stdio.h>
#include <pthread.h>

pthread_mutex_t mut = PTHREAD_MUTEX_INITIALIZER;

int shared_tmp;

void *foo (void *arg)
{
   for (int i = 0; i < 500; ++i) {
      pthread_mutex_lock (&mut);
      shared_tmp = i;
      pthread_mutex_unlock (&mut);
   }
   pthread_exit (NULL);
}

int main (int argc, char *argv[])
{
   pthread_t thr[2];
```

```
    pthread_create (&thr[0], NULL, foo, NULL);
    pthread_create (&thr[1], NULL, foo, NULL);
    pthread_join (thr[0], NULL);
    pthread_join (thr[1], NULL);

    return 0;
}
```

Для данного кода этап профилирования схематично будет выглядеть так (рис.1):

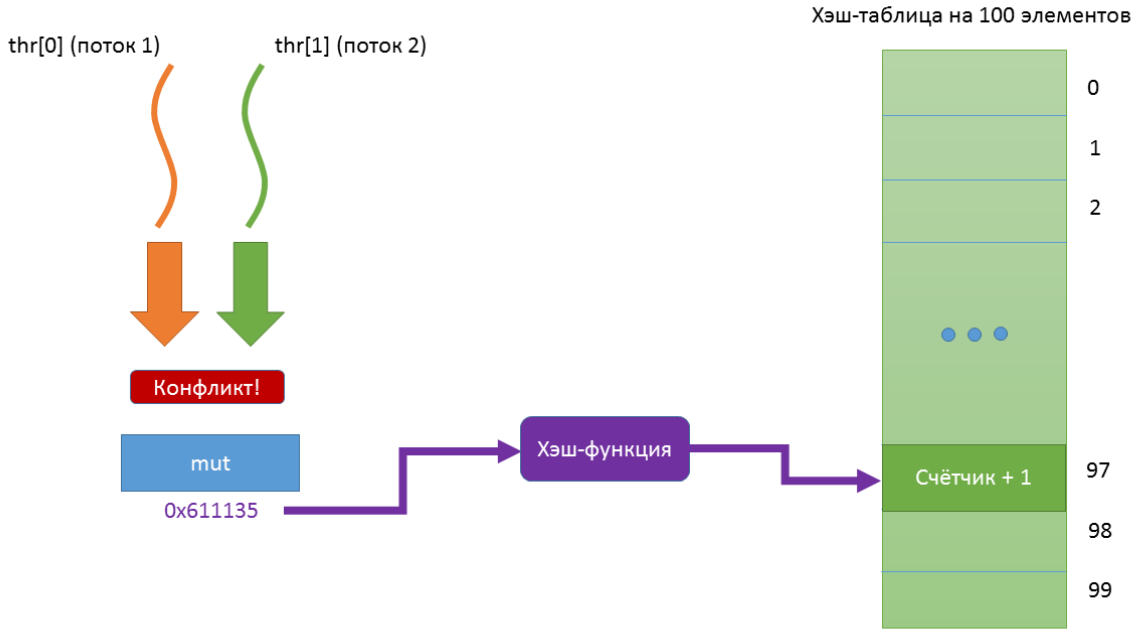

Рис. 1. Схематичное представление этапа профилирования

2.2. Этап оптимизации

На данном этапе производится оптимизация «проблемных мьютексов», то есть таких, значение в хэш-таблице для которых больше среднего. Иначе говоря, для мьютексов, за которые часто одновременно конкурируют множество потоков, будет отрегулирован показатель, отвечающий за количество попыток захвата, перед тем как отправить тот или иной поток «спать». В glibc 2.23 этот показатель является статическим и не регулируется до начала работы многопоточной программы.

3. Функциональная структура программного пакета «mutex-optimizer»

Для профилирования приложения используется методика подмены стандартных функций библиотеки Pthread. Таким образом «mutex-optimizer» включает в себя две динамические библиотеки: profiler.so — позволяет строить статистику возникновения конфликтных ситуаций; optimizer.so — позволяет оптимизировать процесс захвата мьютекса по результатам профилирования. Каждая библиотека используется в определённом режиме функционирования: профилирования или оптимизации.

Выбор режима выполняется при помощи переменной среды окружения LD_PRELOAD:

```
[name@host] $ LD_PRELOAD=profiler.so ./prog.out      — режим профилирования;
[name@host] $ LD_PRELOAD=optimizer.so ./prog.out     — режим оптимизации.
```

Программный пакет «mutex-optimizer» условно можно разделить на две части:

• Модуль профилирования;
• Модуль оптимизации.

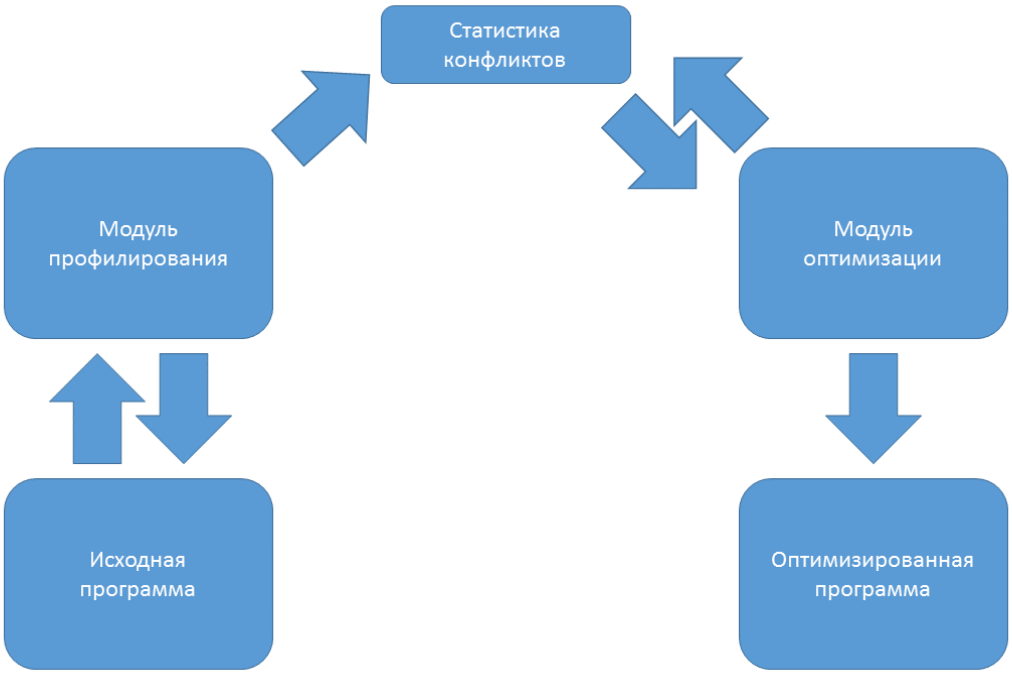

Рис. 2. Структура функционирования программного пакета «mutex-optimizer»

3.1. Структура модуля профилирования

Конструктор, определяющий адреса всех оригинальных функций, которые в дальнейшем будут перехвачены. Также в конструкторе создаётся и инициализируется хэш-таблица, хранящая статистику частоты возникновения конфликтов при работе с мьютексами.

Функция pthread_mutex_lock () подменяющая оригинал: если мьютекс используется впервые — информация о нём добавляется в таблицу; если мьютекс свободен и используется не в первый раз — выполняется оригинал функции; если мьютекс занят и, следовательно, используется не в первый раз — увеличивается счётчик конфликтных ситуаций для данного мьютекса.

Деструктор, записывающий статистику во внешний файл и очищающий память, выделенную под хэш-таблицу.

3.2. Структура модуля оптимизации

Конструктор, определяющий адреса всех оригинальных функций, которые в дальнейшем будут перехвачены.

Функция pthread_mutex_lock () подменяющая оригинал. В этой функции происходит сравнение показателей частоты возникновения конфликтов при захвате мьютекса для частного случая и для общего (средней показатель). Если показатель частного случая больше среднего, то уменьшается количество попыток захвата перед тем как отправить работающий поток «спать».

4. Заключение

В ходе данной исследовательской работы был реализован программный пакет «mutex-optimizer» включающий в себя средства профилирования и оптимизации.

Эффективность разработанного пакета исследовалось на синтетических тестах (microbenchmark). Результаты представленны в призентации.

Литература

1. *Herlihy M., Shavit N. The Art of Multiprocessor Programming, Revised Reprint // Elsevier – 2012.*

Статья поступила в редакцию 19.11.2008;
переработанный вариант — 25.11.2008

Гайдай Анатолий Валерьевич
Студент четвёртого курса факультета информатики и вычислительной техники СибГУТИ (630102, Новосибирск, ул. Кирова, 86), тел. +7 (983) 309-84-33, e-mail: diligent20494@gmail.com.

Кулагин Иван Иванович
Старший преподаватель кафедры вычислительных систем СибГУТИ (630102, Новосибирск, ул. Кирова, 86), тел. (383) 2-698-286, e-mail: ivan.i.kulagin@gmail.com.

Paper Preparation Manual for “Vestnik SibGUTI”

A. Fionov

This manual gives headlines to formatting and preparing papers for “Vestnik SibGUTI”. All papers must be supplied with an abstract in English which is a translation of the Russian language abstract to the paper. The English data will also appear at the web-site. The style for the English part is “VS Abstract”.

Keywords: authors' guidelines, styles, formatting.