Федеральное агентство связи
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Сибирский государственный университет телекоммуникаций и информатики»
(СибГУТИ)

Кафедра вычислительных систем
Допустить к защите
зав. кафедрой д.т.н. доцент
_____ Мамойленко С.Н.

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА МАГИСТРА

Исследование методов синхронизации в параллельных программах

Магистерская диссертация
по направлению 09.04.01 «Информатика и вычислительная техника»

Студент _____ /Егоров Б.В./

Руководитель к.т.н. доцент _____ /Курносов М.Г./

Рецензент к.т.н. доцент _____ /Нечта И.В./

Новосибирск 2016 г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Широкий спектр современного аппаратного обеспечения: гаджеты, мобильные устройства, планшеты, персональные компьютеры, суперкомпьютеры, кластеры, — обладает несколькими вычислительными ядрами или процессорами. Для эффективной работы на таком оборудовании программное обеспечение должно использовать доступные ресурсы как можно более рационально. Во многих случаях использование нескольких одновременно работающих (параллельных) процессов или потоков исполнения позволяет достичь наиболее высокой производительности.

Переход от последовательного исполнения к параллельному зачастую связан с усложнением программного обеспечения, в силу необходимости разбиения задач и/или данных, равномерной загрузки имеющихся вычислителей, и выбора способа синхронизации ветвей исполнения.

Возрастает востребованность методик и инструментов, упрощающих создание эффективного конкурентного программного обеспечения [1] [2].

Важной задачей при реализации параллельных программ является выбор подходящего метода синхронизации потоков исполнения. Выбранный метод сказывается на производительности, масштабируемости и компонуемости создаваемого программного обеспечения.

В данной работе предложена методика анализа эффективности синхронизации в существующем программном обеспечении и подбора более производительных методов синхронизации. Создан инструмент, который с помощью методики выявляет неэффективное использование примитивов синхронизации и выдаёт рекомендации по применению альтернативных методов.

1 Методы синхронизации потоков в параллельных программах

Существуют следующие основные методы синхронизации параллельных программ: взаимная блокировка, неблокирующие структуры данных, транзакционная память. Традиционно более простые методы недостаточно масштабируемы и требуют большого внимания при использовании, в то время как более сложные методы хорошо масштабируются, но нетривиально реализуются.

Выбор того или иного механизма синхронизации значительно сказывается на производительности программы, поэтому важно делать выбор обоснованно. Рассмотрим основные методы синхронизации более подробно.

1.1 Взаимное исключение

Взаимное исключение — гарантия, что два или более конкурентных потока исполнения не будут находиться в участке кода, использующего общий ресурс. Такая часть кода называется критической секцией. Если один из потоков находится в критической секции, любой другой поток, пытающийся в неё войти, не должен иметь возможности это сделать. Вместо этого поток должен активно ожидать покидания секции первым потоком или выполнять другую работу. Исторически взаимное исключение является первым способом организации конкурентных алгоритмов[3].

Существуют как аппаратные, так и программные решения для обеспечения взаимного исключения.

Среди аппаратных решений можно выделить следующие:

- Отключение прерываний в системах с одним вычислительным ядром. Такое решение запрещает операционной системе останавливать поток, находящийся в критической секции. Недостатками подхода является невозможность работы системных часов во выполнения критической секции и полное зависание системы при зависании программы внутри критической секции.
- Использование атомарных операций наподобие test-and-set для реализации активного ожидания. Позволяет убедиться, что только один из потоков выставил необходимый флаг для доступа к общему ресурсу.

Программные решения для работы со взаимным исключением обычно представляют интерфейс, состоящий из следующего набора функций:

- lock() — Взятие мьютекса. Вызывается перед входом в критическую секцию. Если на данный момент он не заблокирован, вызывающий поток начинает выполнение критической секции. В противном случае, поток «засыпает», ожидая освобождения мьютекса, и после пытается захватить его снова.

- unlock() — Освобождение мьютекса. Вызывается при выходе из критической секции.
- trylock() — Вызов, аналогичный lock(), но завершающийся мгновенно в случае, когда мьютекс уже заблокирован. Позволяет продолжать работу вне критической секции, чтобы избежать излишнего ожидания.

Синхронизация возможна без использования вызова trylock(), поэтому он не является обязательным, но предоставляется многими реализациями[4][5][6].

Далее рассмотрены некоторые программные алгоритмы для обеспечения взаимного исключения.

1.1.1 Программные реализации

Алгоритм Деккера

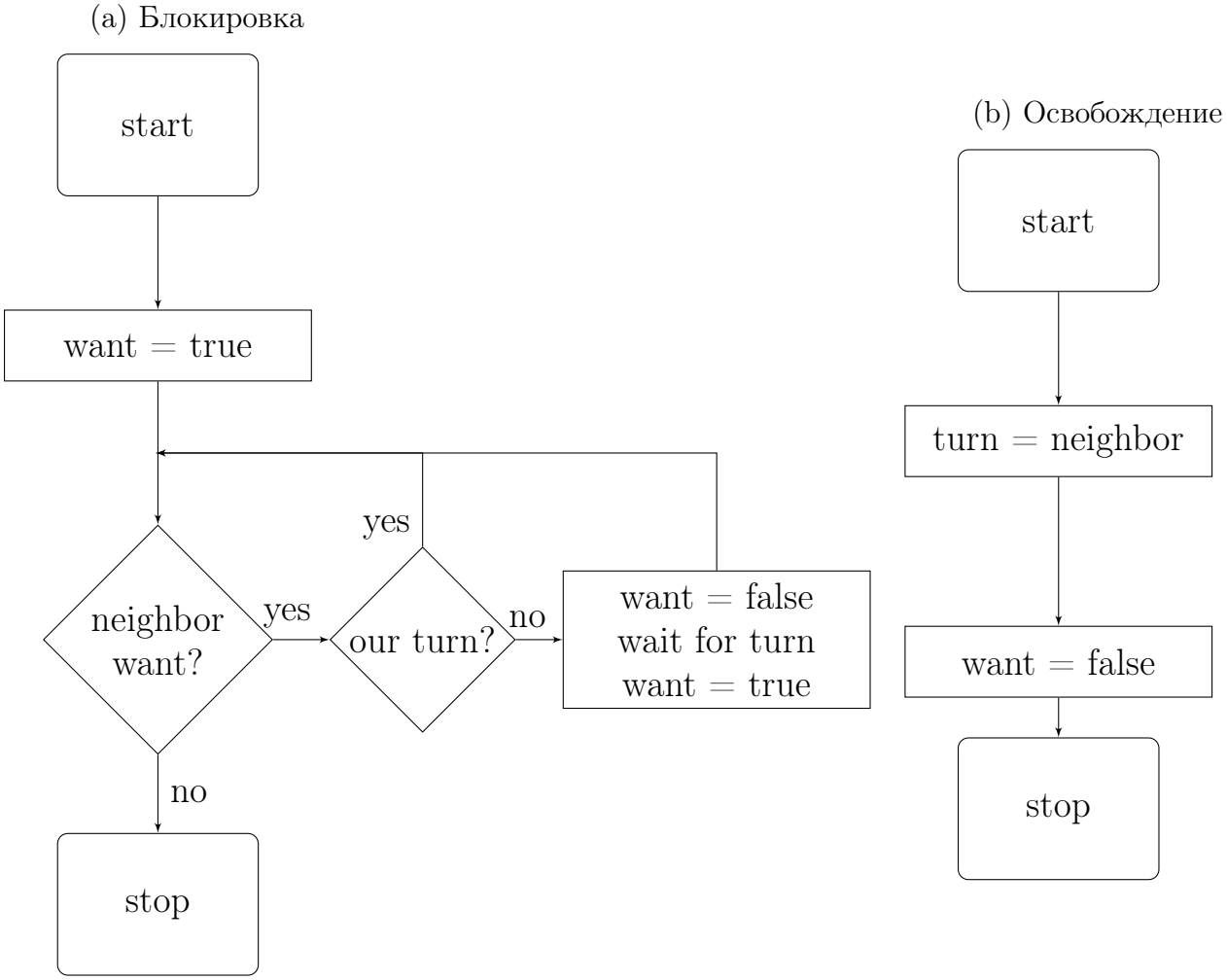
Алгоритм Деккера является исторически первым корректным решением проблемы взаимного исключения в конкурентном программировании[7]. Автором алгоритма считается голландский математик Теодор Деккер.

Алгоритм определяет очерёдность выполнения критической секции двумя потоками с помощью двух флагов, сигнализирующих о желании войти в критическую секцию, и переменной, определяющей приоритет между процессами.

Для входа в критическую секцию поток выставляет ассоциированный с ним флаг wants_to_enter и проверяет, выставлен ли флаг у другого потока. В случае, если второй поток находится в критической секции (переменная приоритета равна идентификатору второго потока и выставлен соответствующий ему флаг), первый поток уступает (сбрасывает значение своего флага) и вступает в фазу активного ожидания. Логическая схема работы приведена на рисунке 1.1a.

После выхода из критической секции поток сбрасывает значение своего флага и выставляет переменную очереди на соседний поток (рисунок 1.1b).

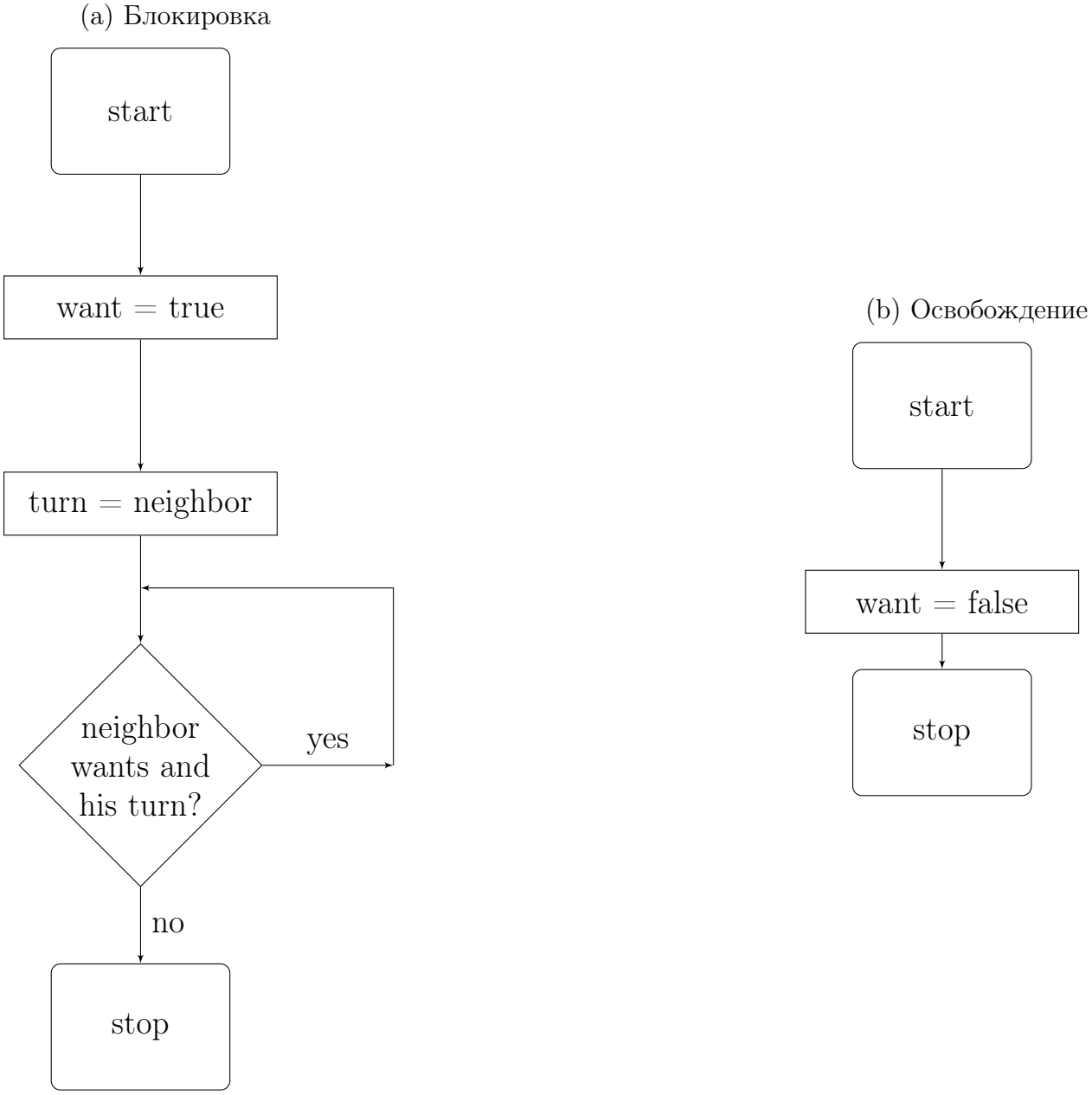
Рисунок 1.1 — Алгоритм Деккера



Алгоритм Петерсона

Алгоритм Петерсона, подобно алгоритму Деккера, использует два флага и одну переменную для определения очередности выполнения критической секции двумя потоками. Однако, его реализация более проста и элегантна[8]. Принцип блокировки изображён на рисунке 1.2a. При входе в критическую секцию поток выставляет свой флаг wants_to_enter и выставляет очередность выполнения на соседний поток, уступая другому потоку. Затем поток входит в режим активного ожидания своей очереди. Для разблокировки снимается свой флаг (рисунок 1.2b).

Рисунок 1.2 — Алгоритм Петерсона



Алгоритм пекарни Лэмпорта

Алгоритм пекарни Лэмпорта обеспечивает выполнение принципа «первый пришёл — первый обслужен» с помощью распределённой версии выдающих номера машин, которые можно увидеть в пекарне: каждый поток выбирает номер и затем ожидает, пока все потоки с меньшими адресами пытаются в него войти.[8].

В алгоритме пекарни каждый поток A обладает флагом wants_to_enter[A], сигнализирующим, что A хочет войти в критическую секцию, и порядком number[A] на вход в критическую секцию. При входе в критическую секцию поток в два этапа генерирует свой порядковый номер. Сначала он считывает все порядковые номера других потоков. Затем он создаёт

порядковый номер на единицу больше максимального порядка на данный момент. Код от поднятия флага до записи порядка можно сравнить с порогом пекарни. Порог устанавливает порядок взятия мьютекса потоками. Для избежания коллизий в порядке используется лексикографическое сравнение пар (number[], thread_id). В ожидающей части алгоритма поток периодически перечитывает порядки других потоков, ожидая момента, когда ни один поток с поднятым флагом не будет иметь лексикографически меньшую пару порядок/идентификатор.

Пример реализации алгоритма приведён в листинге 1.1.

Листинг 1.1 — lamport_bakery_lock.h

```
1  #pragma once
2
3  #include <algorithm>
4  #include <atomic>
5  #include <cstdlib>
6
7  constexpr int kNumberOfThreads = 2;
8
9  std::pair<size_t, size_t> rank(std::atomic<size_t>& i, size_t j)
10 {
11     return std::make_pair(i.load(), j);
12 }
13
14 struct LamportBakeryLock {
15     size_t opposite_id(size_t id) const
16     {
17         return kNumberOfThreads - id - 1;
18     }
19     void lock(size_t id)
20     {
21         wants_to_enter[id] = true;
22         auto max = std::max_element(number, number + kNumberOfThreads);
23         number[id] = 1 + max->load();
24         wants_to_enter[id] = false;
25         for (auto j = size_t{0}; j < kNumberOfThreads; ++j) {
26             while (wants_to_enter[j]) {
27                 ;
28             }
29             while ((number[j] != 0) && (rank(number[j], j) < rank(number[id],
                   id))) {
30                 ;
31             }
32         }
33     }
34     void unlock(size_t id)
35     {
```

```
36            number[id] = 0;
37        }
38        std::atomic<bool> wants_to_enter[kNumberOfThreads]{};
39        std::atomic<size_t> number[kNumberOfThreads]{};
40    };
```

1.1.2 Пример использования

Как уже упоминалось, основным интерфейсом взаимного исключения обычно является пара функций lock()/unlock(), которые соответственно захватывают и освобождают примитив синхронизации. В листинге 1.2 показан простейший пример использования взаимного исключения.

Листинг 1.2 — mutex.cxx

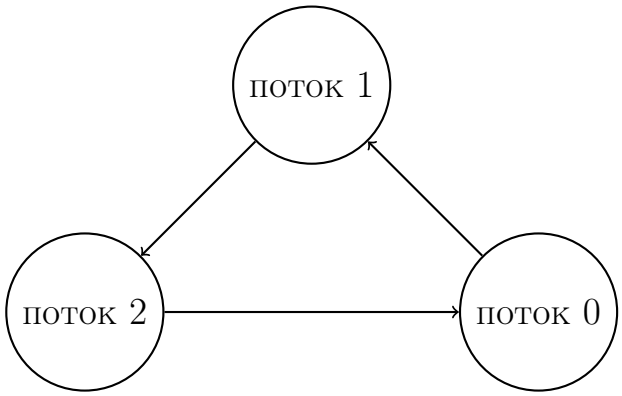
```
1  #include <iostream>
2  #include <unordered_map>
3  #include <string>
4  #include <thread>
5  #include <mutex>
6
7  std::unordered_map<std::string, std::string> pages;
8  std::mutex pages_mutex;
9
10 void download_page(const std::string& url)
11 {
12     std::string content = "<empty>";
13     // ... download page content from network ...
14
15     pages_mutex.lock();
16     pages[url] = content;
17     pages_mutex.unlock();
18 }
19
20 int main()
21 {
22     std::thread t1(download_page, "http://example.com/index1.html");
23     std::thread t2(download_page, "http://example.com/index2.html");
24     t1.join();
25     t2.join();
26
27     for (const auto& pair : pages) {
28         std::cout << pair.first << " => " << pair.second << '\n';
29     }
30 }
```

1.1.3 Проблемы при использовании

Нетривиальной задачей при использовании механизмов взаимного исключения является определение размера кода, который должен войти в критическую секцию (так называемая гранулярность кода). Если пользователь выбирает слишком большие участки кода, то большая часть программы начинает исполняться последовательно, нивелируя использование параллелизма и, возможно, неоправданно прерывая работу ожидающего потока. Если, напротив, выбирается слишком малый участок кода, вырастает конкуренция на использование блокировки.

Другая проблема, возникающая при использовании двух или более мьютексов — возможность получения ситуации взаимной блокировки (deadlock). При этом два или более потока исполнения ожидают освобождения мьютекса от другого потока, образуя циклическую структуру зависимостей. Ни один из ожидающих потоков не может выполнить никакой работы, а следовательно, прогресс выполнения программы значительно замедляется или останавливается совсем. Иллюстрация взаимной блокировки приведена на рисунке 1.3. В программе имеется три мьютекса и три потока, каждый из которых заблокировал один из мьютексов и ожидает освобождения другого. Прогресс выполнения программы остановлен.

Рисунок 1.3 — Пример взаимной блокировки



Наконец, при использовании механизма взаимного исключения возможна ситуация голодания (starvation), когда одному или нескольким потокам не удаётся на протяжении долгого времени заблокировать мьютекс. Таким образом, поток остаётся в постоянном ожидании и не выполняет свою работу.

1.1.4 Итог

Взаимное исключение — механизм синхронизации, который сравнительно просто реализуется, но имеет множество недостатков: возможность взаимной блокировки, возможность голодания одного из потоков исполнения, возможность инверсии приоритетов потоков, сложность масштабирования по имеющемуся количеству процессоров.

1.2 Неблокирующие структуры данных

Потребность в неблокирующих структурах данных возникла из-за ограничений блокирующих методов синхронизации: последовательного выполнения критических секций, снижающего производительность параллельной программы; накладных расходов, связанных с работой примитивов синхронизации; слабой масштабируемости, особенно в условиях высокой нагрузки. Неблокирующие структуры данных призваны решить упомянутые проблемы.

Выделяют три основных типа неблокирующих структур данных, которые отличаются гарантиями по прогрессу выполнения программы[9].

1. Свободные от препятствий (obstruction-free) — поток выполняет работу, если нет препятствий со стороны других потоков. Примером может служить простая реализация мьютекса на основе циклической блокировки (spinlock).
2. Свободные от блокировок (lock-free) — несколько потоков должны быть способны получить доступ к общему ресурсу одновременно. Кроме этого, при остановке одного потока планировщиком все другие потоки должны иметь возможность работать дальше без ожидания приостановленного потока.
3. Свободные от ожидания (wait-free) — свободные от блокировок алгоритмы, которые обладают следующим свойством. Каждый поток, обращающийся к общему ресурсу, может выполнить свою работу за ограниченное число шагов, независимо от поведения других потоков. Такие алгоритмы не могут бесконечно перезапускать свои действия из-за взаимодействия с другими потоками.

Большинство реализаций неблокирующих структур данных полагаются на возможности современных микропроцессоров. Так, зачастую требуется наличие атомарных операций. Особенной популярностью пользуются операции атомарного изменения (read-modify-write, RMW), позволяющие прочесть выравненную ячейку памяти и произвести в неё запись. Базовой операцией атомарного изменения считается сравнение с обменом (compare and swap, CAS). На основе неё можно построить более сложные атомарные операции и структуры данных. Псевдокод CAS-операции приведён в листинге 1.3. Если значение по указателю равно старому значению old_value, значение по указателю заменяется на новое new_value и функция возвращает true. В противном случае, возвращается false. Используя операцию CAS в цикле, можно получить переменные и структуры, которые могут обновляться из нескольких потоков.

Листинг 1.3 — cas.h

```
1 bool cas(int* p, int old_value, int new_value) {
2     atomically {
3         if *p != old_value {
```

```
4                 return false;
5             }
6             *p = new_value;
7             return true;
8         }
9     }
```

1.2.1 Пример реализации

Пример реализации потокобезопасного стека без блокировок приведён в листинге 1.4. Реализация функции push создаёт новый узел с заданным значением, устанавливает указатель next нового узла на текущую голову (head) стека. Затем с помощью CAS-операции, вызываемой в цикле, функция убеждается, что указатель head всё ещё имеет полученное ранее значение, в случае необходимости получает его заново, и устанавливает его на новый элемент. Функция pop, в свою очередь, пытается в цикле установить голову стека на следующий элемент, и после этого вернуть первое значение списка[9].

Стоит упомянуть, что продемонстрированная реализация не освобождает память от извлечённых элементов стека. Можно так же сказать, что такая реализация обладает безопасностью при работе с памятью (memory safety), но подвержена утечкам памяти (memory leaks), которые являются ортогональными понятиями[10]. Реализация без утечек памяти требует существенно больших усилий по её воплощению.

Листинг 1.4 — nonblocking_stack.h

```
1  #include <atomic>
2  #include <memory>
3
4  template<typename T>
5  class lock_free_stack
6  {
7  private:
8      struct node {
9          std::shared_ptr<T> data;
10         node* next;
11
12         node(T const& data_)
13             : data{sta::make_shared<T>(data_)}
14         {}
15     };
16     std::atomic<node*> head;
17 public:
18     void push(T const& data)
19     {
20         const node* new_node = new node(data);
21         new_node->next = head.load();
```

```
22            while(!head.compare_exchange_weak(new_node->next, new_node));
23        }
24        std::shared_ptr<T> pop()
25        {
26            node* old_head = head.load();
27            while(old_head && !head.compare_exchange_weak(old_head, old_head->next)
                  );
28            return old_head ? old_head->data : std::shared_ptr<T>();
29    };
```

1.2.2 Проблемы при использовании

Существенным недостатком неблокирующих структур данных является сложность их реализации. Практичная реализация требует тщательного проектирования и обширного тестирования, требуется отличное знание архитектур процессоров, которые должны поддерживаться. Дополнительные сложности возникают в языках программирования с ручным управлением памятью. В условиях многопоточности проблемы работы с памятью: двойные освобождения, использование после освобождения, утечки памяти, — решаются значительно сложнее, чем в однопоточных программах.

К примеру, при удаления узла структуры данных, на который имеется ссылка в другом потоке, дальнейшие обращения по этой ссылке приведут к неопределённому поведению в работе программы. Одним из решений проблемы является использование указателей опасности (hazard pointers), сигнализирующих об использовании каких-либо объектов и опасности их удаления. Другим подходом является подсчёт ссылок (reference counting): для каждого объекта хранится число потоков, его использующих. Объект не может быть удалён, пока счётчик не достигнет нулевого значения.

Другую сложную проблему называют ABA-проблемой: один из потоков P_1 читает значение A из общей памяти, после чего замещается потоком P_2. P_2 меняет значение A на B и обратно на A до своего замещения. Когда P_1 продолжает выполняться, он не видит изменений прочтённого ранее значения и работает с ним далее. Зачастую, это некорректное поведение, которого нужно избегать. Обнаружение и устранение ABA-проблемы в существующих кодовых базах требует больших усилий.

1.2.3 Итог

Неблокирующие структуры данных позволяют достичь высокой эффективности и масштабируемости параллельных программ, однако чрезвычайно сложно реализуются и подвержены проблеме зацикливания алгоритма (livelock).

1.3 Транзакционная память

Транзакционная память (transactional memory, TM) — алгоритм синхронизации, аналогичный механизму транзакций баз данных. Транзакцией является последовательность шагов, выполняемая одним потоком. Если при завершении транзакции обнаруживается конфликт данных, транзакция отменяется. Отмена транзакции приводит к отмене действий, которые совершал поток за время транзакции. После этого транзакция обычно перезапускается, либо вызывается функция, заранее указанная как «запасной выход».

Транзакции должны быть сериализуемы, то есть, выполняться последовательно, одна за другой. Сериализуемость определяет атомарность на уровне транзакции — внутри транзакции можно делать вызовы к произвольным объектам. При корректной реализации транзакционная память не подвержена взаимным блокировкам и зацикливаниям[8]. Важной особенностью транзакционной памяти является её компонуемость: транзакции могут вкладываться друг в друга и использовать несколько разных объектов без каких-либо сложностей.

Подход к управлению конкурентностью с использованием транзакционной памяти называют оптимистичным: считается, что потоки работают независимо друг от друга и в редких случаях изменяют одни и те же данные. В таком случае большинство транзакций заканчивается успешно. В противоположность этому, подходы с использованием блокировок пессимистичны: мы полагаем, что потоки будут конфликтовать всегда и всегда запрещаем им находиться в критической секции одновременно[11].

Выделяют аппаратные (hardware transactional memory, HTM) и программные (software transactional memory, STM) подходы к реализации транзакционной памяти. Далее описаны подробности нескольких реализаций TM.

1.3.1 Аппаратные реализации

Sun Rock

Первым микропроцессором с аппаратной поддержкой транзакционной памяти был 64-разрядный Rock архитектуры SPARC v9 с 16 вычислительными ядрами. Компания Sun Microsystems заявила о поддержке TM в 2007[12]. Были введены две процессорные инструкции и статусный регистр:

- chkpt <fail_pc> — инструкция начала транзакции;
- commit — инструкция завершения транзакции;
- cps — регистр, благодаря которому определяется причина отмены транзакции.

При отмене транзакции происходил переход на адрес <fail_pc>, а причинами отмены могли быть конфликты данных, промахи TLB, прерывания, сложные инструкции[11].

Проект Sun Rock существовал до приобретения компании Sun Microsystems

корпорацией Oracle. После покупки проект был объявлен неперспективным и был свёрнут.

IBM BlueGene/Q

BlueGene/Q — первая коммерческая система, поддерживающая транзакционную память. Реализация основана на метке «версия» в кэше второго уровня процессора PowerPC A2, используемого в системе. Кэш может хранить несколько версий одних и тех же данных. Процессор сравнивает версии данных при подтверждении транзакции. Если версия совпадает с версией на начало транзакции, значит другие потоки не модифицировали те же данные, и транзакция успешно завершается. В противном случае, транзакция отменяется[13].

Стоит отметить, что первой инсталляцией суперкомпьютера на архитектуре BlueGene/Q стал Sequoia, установленный в Ливерморской национальной лаборатории. Sequoia уже более 4 лет входит в список пяти самых высокопроизводительных суперкомпьютерных систем в мире[14].

Intel TSX

Компания Intel объявила о поддержке расширения транзакционной синхронизации (Transactional Synchronization Extensions, TSX) в 2012, и начиная с 2013 аппаратная поддержка транзакционной памяти впервые появилась на потребительском рынке. Позднее в реализации был найден баг, за которым последовало отключение технологии в процессорах Haswell и ранних моделях Broadwell[15] (Broadwell-Y). Последующие модели поддерживают транзакционную память без известных проблем.

Реализация TSX в процессорах Intel состоит из двух частей: аппаратного исключения блокировок (Hardware Lock Elision, HLE) и ограниченной транзакционной памяти (Restricted Transactional Memory, RTM).

HLE позволяет добавить транзакции в существующие программы, поддерживая обратную совместимость. Для этого модифицируются инструкции для изменения участка памяти, используя префиксы XACQUIRE (взятие блокировки) и XRELEASE (освобождение блокировки). Между взятием и освобождением блокировки процессор отслеживает участки памяти, которые читают и записывают потоки. Конфликт данных обнаруживается с помощью протокола когерентности кэша. При обнаружения конфликта транзакция отменяется[16].

RTM является полноценной реализацией транзакционной памяти, вводящей три новые инструкции:

- XBEGIN — начало транзакции с указанием «запасного выхода»;
- XEND — окончание транзакции;
- XABORT — явное прерывание транзакции.

RTM уходит от обратной совместимости, но позволяет контролировать

транзакции более полно, чем HLE, упрощая создание системного программного обеспечения на основе транзакционной памяти.

1.3.2 Программные реализации

Haskell STM

Примечательной программной реализацией транзакционной памяти является модуль Control.Concurrent.STM платформы Haskell. Благодаря мощной системе типов, компилятор гарантирует, то внутри транзакции используются только разрешённые транзакционные типы данных[17](в частности, транзакционные переменные TVar).

Пример использования транзакционной памяти приведён в листинге 1.5.

Листинг 1.5 — transactional_memory.hs

```
1  import System.IO
2  import Control.Concurrent.STM
3
4  type Account = TVar Int
5
6  withdraw :: Account -> Int -> STM ()
7  withdraw acc amount = do
8      bal <- readTVar acc
9      writeTVar acc (bal - amount)
10
11 deposit :: Account -> Int -> STM ()
12 deposit acc amount = withdraw acc (- amount)
13
14 transfer :: Account -> Account -> Int -> IO ()
15 transfer from to amount
16     = atomically (do deposit to amount
17                      withdraw from amount)
18
19 showAccount :: Account -> IO Int
20 showAccount acc = atomically (readTVar acc)
21
22 main = do
23     from <- atomically (newTVar 200)
24     to   <- atomically (newTVar 100)
25     transfer from to 50
26     v1 <- showAccount from
27     v2 <- showAccount to
28     putStrLn \$ (show v1) ++ ", " ++ (show v2)
```

GCC TM

Набор компиляторов GCC поддерживает транзакционную память с версии 4.7[18]. Реализация представлена библиотекой libitm, созданной с оглядкой на черновик транзакционных конструкций, предложенных на включение в стандарт C++[19]. Транзакции предоставляют гарантии синхронизации, схожие с теми, которые дал бы единая глобальная блокировка, охраняющая все транзакции. Для корректной работы в программе не должно быть гонок данных. К примеру, нетранзакционная запись, выполняющаяся конкурентно с транзакционным чтением, приведёт к программу в состояние гонки. Как и большинство практичных реализации транзакционной памяти, реализацию GCC можно охарактеризовать как гибридную (STM + HTM)[20].

Реализация вводит следующие конструкции:

- __transaction_atomic {...} — блок синхронизации;
- __transaction_relaxed {...} — указание, что небезопасный код внутри блока не приводит к побочным эффектам;
- __transaction_cancel — явная отмена транзакции;
- attribute((transaction_safe)) — указание транзакционно-безопасной функции;
- attribute((transaction_pure)) — указание функции без побочных эффектов.

Пример использования реализации приведён в листинге 1.6.

Листинг 1.6 — tm.cxx

```
1 __transaction_atomic {
2     for (i = 0; i < d->sz; ++i) {
3         struct pixel p = d->pixels[i];
4
5         unsigned int luminance = rY * p.red + gY * p.green + bY * p.blue;
6
7         ++histogram[luminance/BORDER];
8     }
9 }
```

1.3.3 Проблемы при использовании

В настоящее время использование транзакционной памяти связано со следующими проблемами.

Во-первых, существуют ограничения на типы операций, которые могут быть использованы внутри транзакции. В частности, невозможно выполнения ввода-вывода информации и установка точек остановки отладчика. Из этого вытекает сложность отладки и тестирования транзакционного кода.

Во-вторых, стоимость коллизии данных может быть достаточно высокой. Внутри транзакции может выполняться множество действий, и её откат

отменяет всю выполненную работу.

Наконец, чисто программные реализации имеют не слишком высокую производительность. Так, большие накладные расходы привносит мониторинг общей памяти, используемой в транзакции. Стоит отметить, что внедрение аппаратных реализаций позволит повысить производительность и снизить актуальность этой проблемы.

1.3.4 Итог

Транзакционная память — перспективный метод синхронизации, лишённый множества проблем традиционных методов синхронизации. Несмотря на имеющиеся недостатки, на сегодняшний день это один из самых перспективных подходов к решению проблемы синхронизации. Метод просто используется, хорошо масштабируется, компонуется, и позволяет достичь приемлемого уровня производительности. Распространение решений на основе транзакционной памяти позволит создавать более надёжные и поддерживаемые параллельные программы.

1.4 Постановка задачи

Как было показано ранее, существует несколько методов синхронизации параллельных программ. Чтобы программа работала эффективно, важно выбрать наиболее подходящий алгоритм синхронизации и иметь возможность оценить производительность его использования.

В рамках данной работы требуется разработать методику анализа производительности методов синхронизации параллельного программного обеспечения и автоматической генерации рекомендаций по применению алгоритмов синхронизации. Для демонстрации работы методики необходимо спроектировать программный инструмент, который на основании методики будет способен облегчить работу по распараллеливанию существующих программ и нахождению «узких горлышек» синхронизации. Предполагается использование статического и динамического анализа для получения информации из программного кода и профилирования анализируемого приложения соответственно. Для краткости далее инструмент будет именоваться ph (parallelize helper) или анализатором.

2 Адаптивный метод синхронизации

На основе собранной информации выработан подход выбора метода синхронизации на основе синтаксического анализа исходного кода анализируемой программы и динамического анализа её выполнения. Далее приведены детали предлагаемого подхода и необходимые проверки статического и динамического анализа. Детали реализации приведены в разделе 3.

2.1 Статический анализ

Статический анализ кода позволяет оценить использование программой параллелизма без непосредственного её запуска. Таким образом, подобные проверки значительно проще автоматизируются. В силу достаточной сложности проверок, для целей ph необоходим доступ к абстрактному синтаксическому дереву (abstract syntax tree, AST) анализируемой программы. Поскольку выявление всех возможных ошибок в программе не представляется возможным, статический анализ фокусируется исключительно на проблемах синхронизации и производительности параллельных программ. Далее перечислены реализуемые статические проверки исходного кода.

2.1.1 Поиск не потокобезопасных функций

Одной из задач, возникающей при параллелизации существующих программ, является поиск функций API операционной системы, использование которых не потокобезопасно. Зачастую использование подобных функций в параллельных программах приводит к неопределённому поведению или снижению производительности. Для данной проверки был составлен список функций стандарта POSIX, которые не должны быть потокобезопасными[21]. Анализатор должен найти в исходном коде анализируемой программы такие функции и предупредить пользователя о небезопасности или неэффективности их использования. Реализация может просто перебирать все вызовы функций и искать среди них не потокобезопасные.

2.1.2 Анализ кода внутри критических секций

Анализ кода внутри критических секций позволяет найти неэффективное использование примитивов синхронизации и в конечном счёте сформировать рекомендации по применению более подходящих методов синхронизации.

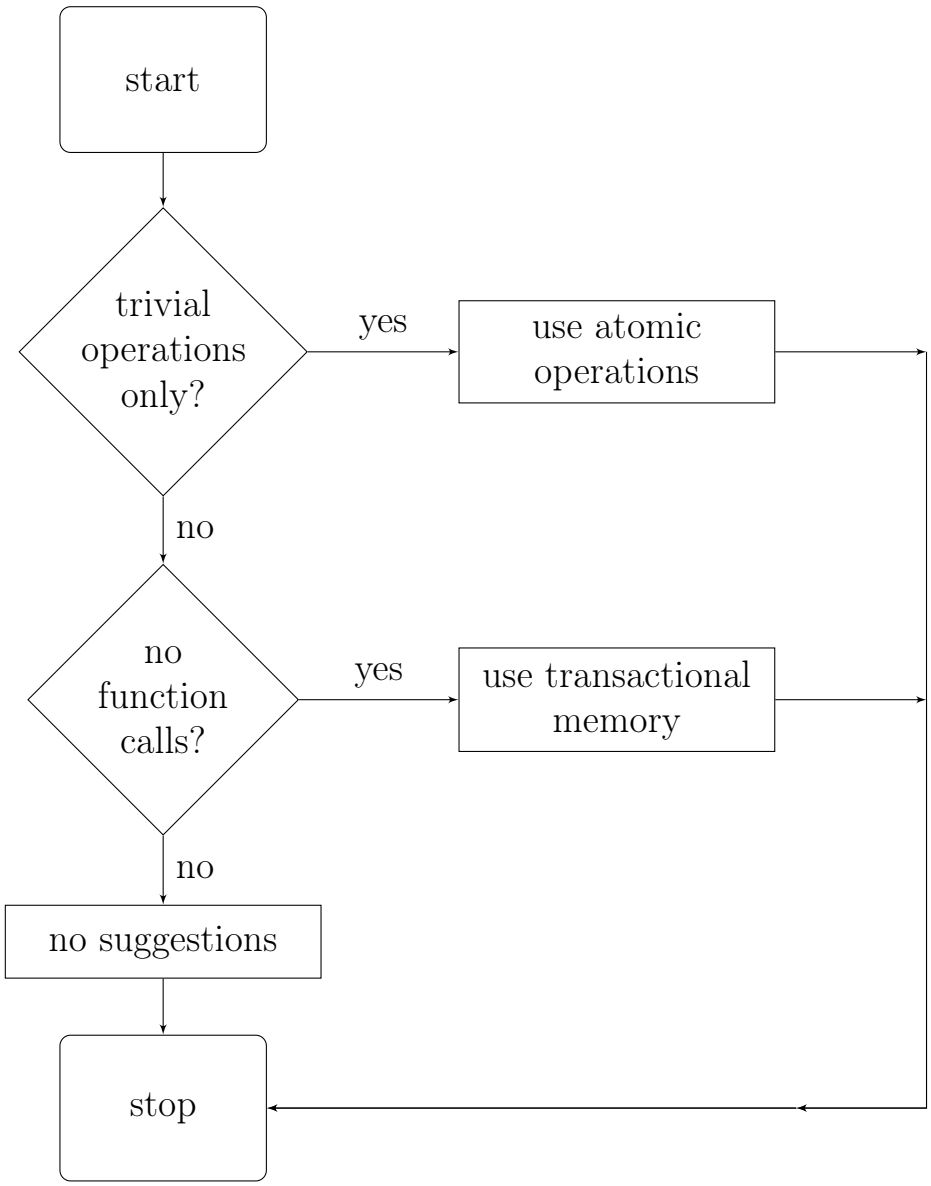
Анализатор должен собрать статистику о типах операций внутри критических секций и на основе неё сгенерировать диагностики. Предполагается разделять операции на следующие типы:

- определения переменных;

- обращения к указателям;
- обращения к массивам;
- вызовы функций;
- тривиальные вычисления — операции, не являющиеся обращениям к указателям или массивам, или вызовам функций; обычно представляют собой работу с несколькими переменными.

Логическая схема генерации рекомендаций на основе созданной статистики приведена на рисунке 2.1. При отсутствии работы с массивами, указателями, и функциями рекомендуется применять атомарные операции. В подобных случаях вся работа внутри критической секции сводится к тривиальным модификациям переменных, которые можно объявить атомарными для большей эффективности. К примеру, реализация счётчика на основе атомарной переменной будет значительно эффективнее использования взаимной блокировки. В случае, когда к тривиальным операциям добавляется работа с указателями и/или массивами, выдаётся рекомендация использовать транзакционную память. Очевидно, что в отсутствие вызовов функций транзакционная память может выполняться эффективно и, вероятно, с небольшим количеством отмен транзакций. Наконец, при наличии вызовов функций анализатор отказывается выдавать рекомендации. Анализ произвольных функций значительно сложнее анализа последовательных операций, и в проектируемом инструменте не осуществляется.

Рисунок 2.1 — Генерация рекомендаций по использованию методов синхронизации



2.2 Динамический анализ

Динамический анализ программы позволяет выявить детали работы, которые не возможно установить на этапе синтаксического анализа хотя бы потому, что порядок выполнения потоков параллельной программы не является детерминированным. Недостатком динамического анализа является необходимость собираемости и работы анализируемой программы и анализатора на одной платформе, а также потребность типового запуска программы или её тестов.

Аналогично этапу статического анализа, рассматриваются лишь особенности работы программы, связанные с производительностью и синхронизацией параллельной программы.

2.2.1 Анализ количества одновременно работающих потоков

Простейшая статистика, которую можно получить с помощью динамического анализа — максимальное количество одновременно работающих потоков в программе. При слишком большом количестве потоков будет возрастать конкуренция между ними, и система будет тратить слишком много времени на переключения между потоками. С другой стороны, при недостаточном количестве потоков аппаратные ресурсы будут простаивать, вместо выполнения полезной работы.

Реализация может мониторить количество запущенных потоков на протяжении работы команды и вести простой счётчик их максимального количества. В случае, если значение счётчика превышает количество вычислительных ядер, пользователю выдаётся предупреждение. Для учёта возможности простаивания потоков, можно выводить предупреждение при превышении числа ядер числом потоков более чем в два раза.

2.2.2 Анализ использования блокировок

Полезной информацией, собираемой при выполнении параллельной программы, может оказаться статистика использования мьютексов. Определим понятие «конкурентность» (contention) как количество случаев, в которых поток попытался заблокировать заблокированный ранее мьютекс. Такой показатель будет полезным для поиска наиболее используемых, а следовательно и наиболее замедляющих работу программы мьютексов.

В отрыве от времени выполнения программы конкурентность имеет мало смысла: более высокое значение параметра при долговременном исполнении может говорить лишь о периодическом использовании мьютекса, но не оценивает степень его использования. Для анализа производительности будем использовать показатель cps, cps = contention/running_time_s, где running_time_s — время выполнения программы в секундах. Параметр cps является численной мерой «горячести» примитива синхронизации, и высокое его значение говорит о неэффективном использовании параллелизма.

Эксперименты с синтетическими тестами и реальными программными продуктами (см. раздел 4) позволили классифицировать мьютексы по величине параметра cps. В таблице 2.1 демонстрируется ранг мьютекса, выводимый на основе диапазона значений cps и информация, предоставляемая ph для каждого из рангов. Наиболее полная информация отображается для наиболее конкурентных мьютексов, наглядно представляя наиболее проблемы синхронизации. Детальная информация о мьютексах с низким рангом скрывается, чтобы не замусоривать вывод анализатора.

Таблица 2.1 — Ранг мьютекса в зависимости от параметра cps

Диапазон значений cps	Ранг	Отображаемая информация
$[0.0 : 5.0)$	Низкий	Значение cps
$[5.0 : 50.0)$	Средний	Значение cps, статистика использования мьютекса
$[50.0 : 100.0)$	Высокий	Значение cps, статистика использования мьютекса, трассировка вызовов
$[100.0 : \infty)$	Экстремальный	Значение cps, статистика использования мьютекса, трассировка вызовов

3 Функциональная структура пакета

Отдельные части анализатора ph следуют идеологии инструментов, на которых они основаны. Далее рассматриваются подробности реализации каждой из частей анализатора.

3.1 Статический анализ

Статический анализ кода параллельной программы осуществляется с помощью дополнения для инфраструктуры LLVM/Clang[22], предоставляющего несколько программных интерфейсов для добавления новых возможностей[23]:

- Библиотека LibClang — стабильный высокоуровневый интерфейс к Clang.
- Дополнения Clang — позволяют выполнять дополнительные действия на AST программы во время компиляции. Дополнения являются динамическими библиотеками, загружаемые компилятором во время выполнения.
- LibTooling — интерфейс на языке C++, предназначенный для создания автономных инструментов и интеграции в сервисы, запускающие инструменты Clang.

Из предлагаемых интерфейсов был выбран интерфейс дополнений Clang[24], как наиболее гибкий и соответствующий целям анализатора. В качестве языка разработки выбран C++, так как именно для этого языка инфраструктура предоставляет API. Каркас проекта позаимствован у статического анализатора Clazy[25], фокусирующегося на проектах, созданных с использованием платформы Qt.

Использование данных технологий существенно упростило создание анализатора, позволив сфокусироваться на непосредственной логике анализа, избегая сложной работы по разбору исходного кода и построению синтаксического дерева программы. Так, для добавления новой проверки достаточно добавить один заголовочный и один исходный файл на C++ и добавить их в систему сборки. Проверки получают доступ к абстрактному синтаксическому дереву программы с помощью программного паттерна Посетитель[26]: каждый оператор программы посещается проверкой единожды. В приложении Б приведён исходный код разработанных проверок.

3.1.1 Поиск не потокобезопасных функций

Проверка выбирает вызовы функций среди всех операторов программы и пытается отыскать название функции в списке не потокобезопасных. В случае нахождения такой функции в списке, генерируется предупреждение компилятора. Пользователю предоставляется возможность расширить список не потокобезопасных функций, добавив имена таких функций в текстовый

файл. Пример диагностики при обнаружении не потокобезопасной функции приведён в листинге 3.7.

Листинг 3.7 — pwnam.txt

```
1 $ ph --binary-path ./test/c_pwnam/pwnam --project-path test/c_pwnam/ -l C
2 [*] statical analysis of source code
3 [*] lock usage analysis
4         pwnam.c:13:26: warning: Non-reentrant function used [-Wph-non-reentrant
              -function]
5             struct passwd *pwd = getpwnam(username);
6                                  ^
```

3.1.2 Анализ кода внутри критических секций

Критические секции в исходном коде программы обнаруживаются с помощью преднастроенного списка имён функций, ограничивающих критическую секцию. Доступны как функции блокировки в стиле языка программирования C (pthread_mutex_lock, pthread_mutex_unlock), так и основанные на RAII блокировщики, доступные в C++ (например, std::lock_guard). После обнаружения начала критической секции начинается сбор статистики об используемых типах операций. Структура операции clang::Stmt поочерёдно приводится к различным типам операций и в случае успешного приведения инкрементируется один из счётчиков, соответствующий типу операции. Сбор статистики прекращается при окончании области видимости (для RAII-блокировщиков) или обнаружении функции разблокировки. Собранная статистика далее анализируется главной программой.

Пример диагностики приведён в листинге 3.8. Анализатор выявил, что в критической секции программа не использует вызовов функций, но использует доступ по указателям и/или обращается к массиву. В таком случае выдаётся рекомендация использовать для синхронизации транзакционную память.

Листинг 3.8 — c_pwnam.txt

```
1  [*] statical analysis of source code
2  [*] lock usage analysis
3          counter.c:10:34: warning: unused parameter 'arg' [-Wunused-parameter]
4          static void * thread_start(void *arg)
5                                      ^
6          total locks: 1
7          1 warning generated.
8
9          [Lock information]
10                 Location:              counter.c:12
11                 array subscripts:      0
12                 pointer manipulations: 3
13                 trivial calculations:  0
```

```
14                  declarations:           0
15                  calls:                  0
16
17          [Suggestion]
18          Lock uses various parts of memory. Consider using transactional memory.
```

3.2 Динамический анализ

Динамический анализ параллельной программы осуществляется с помощью адаптированной утилиты mutrace[27]. Данная утилита использует механизм LD_PRELOAD для подгрузки библиотеки libmutrace.so при запуске исследуемой программы. Библиотека содержит переопределения различных системных функций (таких, как pthread_create, pthread_mutex_lock, pthread_mutex_unlock) и ведёт сбор статистики по использованию этих функций. Благодаря описанному устройству, утилита позволяет проанализировать работу параллельной программы с небольшими накладными расходами при её исполнении.

Язык реализации — C. В оригинальную программу был добавлен сбор дополнительной статистики о работе подконтрольной программы и реализована возможность вывода информации в файл для дальнейшего анализа главной программой ph.

3.2.1 Анализ количества одновременно работающих потоков

Поток считается исполняющимся в промежутке между вызовами функций pthread_create и pthread_join (или pthread_detach). Полученное значение nthreads сравнивается с количеством доступных ядер процессора nkernels, и если nkernels $<$ 2 * nthreads, выводится предупреждение о возможности снижения производительности программы при наличии такого количества потоков.

Пример диагностики приведён в листинге 3.9.

Листинг 3.9 — c_pwnam.txt

```
1 [*] thread usage analysis
2           threads launched: 101
3           threads supported by hardware: 8
4           NOTE: number of threads can be too high
```

3.2.2 Анализ использования блокировок

Второй набор статистических значений, собираемых mutrace, является более полезным. Утилита собирает статистику об использовании блокирующих механизмов синхронизации: количество раз, которое мьютекс был захвачен, проведённое в блокировке время, конкурентность contention. Собранная

информация передаётся главной программе ph, которая вычисляет значение cps, классифицирует мьютексы согласно таблице 2.1 и при необходимости выводит детали использования.

Пример диагностики приведён в листинге 3.10. Анализатор обнаружил мьютекс с высоким значением параметра cps и вывел всю доступную ему информацию об этом мьютексе: значение cps, статистику использования, трассировку вызовов.

Листинг 3.10 — lock_usage.txt

```
1  [*] lock contention analysis
2          Running time: PT0.012813815S
3          Contention for mutex 0 is too high: 6867.587833912071 per second
4          Mutex info: MutexStat { id: 0, locked: 100, changed: 99, contention:
               88, contention_time: 142.418, total_time: 3.965, avg_time: 0.04,
               flags: "M-.--." }
5          Stacktrace:
6                  /usr/local/lib/libmutrace.so(pthread_mutex_lock+0x119) [0
                       x7fe114e5243c]
7                  test/cxx_pwnam/build/pwnam() [0x406683]
8                  test/cxx_pwnam/build/pwnam(std::mutex::lock()+0x15) [0x406bb5]
9                  test/cxx_pwnam/build/pwnam(std::lock_guard<std::mutex>::
                       lock_guard(std::mutex&)+0x23) [0x4067c3]
10                 test/cxx_pwnam/build/pwnam(pnam_thread(char const*)+0x1f) [0
                       x4062cf]
11                 test/cxx_pwnam/build/pwnam(void std::_Bind_simple<void (*(char
                       const*))(char const*)>::_M_invoke<0ul>(std::_Index_tuple<0ul
                       >)+0x42) [0x407f62]
12                 test/cxx_pwnam/build/pwnam(std::_Bind_simple<void (*(char const
                       *))(char const*)>::operator()()+0x15) [0x407f15]
13                 test/cxx_pwnam/build/pwnam(std::thread::_Impl<std::_Bind_simple
                       <void (*(char const*))(char const*)> >::_M_run()+0x19) [0
                       x407bf9]
14                 /lib64/libstdc++.so.6(+0xb8f20) [0x7fe114965f20]
```

3.3 Главная программа анализатора

Управление отдельными частями анализатора осуществляет главная программа анализатора. Она выполняет следующие задачи:

- подбор подходящей системы сборки (Make, CMake) и её конфигурация;
- сборка проверяемой программы с нужными опциями;
- запуск статического анализа и динамического анализа;
- выполняет свою часть анализа (генерация рекомендаций на основе статистики использования критических секций, вычисление cps, ранжирование мьютексов);
- форматирование вывода для удобочитаемости отчёта о проверке.

В качестве языка реализации был выбран Rust[28], позволяющий достичь высокой безопасности программы и приемлемого уровня производительности. Как и остальные части анализатора, главная программа следует идеологии платформы, на которой она реализована. Для сборки проекта используется Cargo, для вспомогательных нужд подключаются зависимости от модулей clap, num_cpus, time, clippy. Исходный код главной программы приведён в приложении А.

4 Эксперименты

После введения параметра cps в 2.2.2, было проведено несколько экспериментов на реальных программах и на синтетических тестах, заведомо неэффективно использующих примитивы синхронизации. На основе собранных данных, значения параметра cps были разделены на несколько диапазонов. При низких значениях параметра пользователю лишь выдаётся общая информация о мьютексе и параметр cps. При средних и высоких значениях параметра пользователю выводится полная трассировка мьютекса. Трассировка позволяет легко найти последовательность вызовов, приводящую к неэффективному использованию параллелизма.

4.1 Синтетические тесты

Для отладки и тестирования работы анализатора был создан набор синтаксических тестов, которые используют параллелизм не самым эффективным способом. Далее приведены примеры диагностик ph при анализе тестов.

4.1.1 Пример диагностики для теста c_pwnam

c_pwnam — программа, запускающая большое число потоков, синхронизирующихся с помощью мьютекса и использующих не потокобезопасную функцию getpwnam. Данный тест демонстрирует большинство диагностик, выдаваемых анализатором ph. Пример вывода приведён в листинге 4.11.

Листинг 4.11 — c_pwnam.txt

```
1  [*] statical analysis of source code
2  [*] lock usage analysis
3          pwnam.c:13:26: warning: Non-reentrant function used [-Wph-non-reentrant
               -function]
4              struct passwd *pwd = getpwnam(username);
5                                   ^
6          1 warning generated.
7
8          [Lock information]
9                  Location:               pwnam.c:12
10                 array subscripts:       0
11                 pointer manipulations:  0
12                 trivial calculations:   0
13                 declarations:           1
14                 calls:                  7
15
16         No suggestions
17 [*] thread usage analysis
```

```
18          threads launched: 101
19          threads supported by hardware: 8
20          NOTE: number of threads can be too high
21  [*] lock contention analysis
22          Running time: PT0.012981157S
23          Contention for mutex 0 is too high: 6008.70939316118 per second
24          Mutex info: MutexStat { id: 0, locked: 100, changed: 99, contention:
                78, contention_time: 57.746, total_time: 2.901, avg_time: 0.029,
                flags: "M-.--." }
25          Stacktrace:
26                  /usr/local/lib/libmutrace.so(pthread_mutex_lock+0x11c) [0
                        x7f32aae793c7]
27          ./test/c_pwnam/pwnam() [0x400946]
28          /lib64/libpthread.so.0(+0x75ba) [0x7f32aac5e5ba]
```

4.1.2 Пример диагностики для теста trivial_counter

trivial_counter — тест, запускающий несколько потоков, синхронизирующихся с помощью мьютекса и инкрементирующих переменную целочисленного типа. В данном случае анализатор определяет, что внутри критической секции используются только тривиальные вычисления с использованием нескольких переменных. Пользователю выдаётся рекомендация использовать атомарные операции. Пример диагностик приведён в листинге 4.12.

Листинг 4.12 — trivial_counter.txt

```
1  [*] statical analysis of source code
2  [*] lock usage analysis
3           [Lock information]
4                   Location:               counter.c:15
5                   array subscripts:       0
6                   pointer manipulations:  0
7                   trivial calculations:   3
8                   declarations:           0
9                   calls:                  0
10
11          [Suggestion]
12          Lock used only in trivial calculations. Consider using atomic
                operations.
13 [*] thread usage analysis
14          threads launched: 9
15          threads supported by hardware: 8
16 [*] lock contention analysis
17          Running time: PT0.007883739S
18          Contention for mutex 0 is too high: 253.6867341752435 per second
19          Mutex info: MutexStat { id: 0, locked: 8, changed: 7, contention: 2,
                contention_time: 0.1, total_time: 0.003, avg_time: 0, flags: "M
                -.--." }
```

```
20          Stacktrace:
21                  /usr/local/lib/libmutrace.so(pthread_mutex_lock+0x11c) [0
                        x7f3a0054c3c7]
22          ./test/c_counter_fundamental/counter() [0x4008de]
23          /lib64/libpthread.so.0(+0x75ba) [0x7f3a003315ba]
```

4.2 Анализ программы Kate

Первой программой, исследованной с использованием ph, является текстовый редактор Kate[29]. Динамический анализ редактора выявил средний показатель параметра cps в одном из мьютексов, используемых программой. Вызов функции относится к библиотеке libxcb и не относится к исходному коду самого Kate. Из данного эксперимента можно сделать вывод, что узким горлышком синхронизации может оказаться не самое очевидное место параллельной программы. Пример диагностик при исследовании Kate приведён в листинге 4.13.

Листинг 4.13 — kate.txt

```
1  [*] statical analysis of source code
2  [*] lock usage analysis
3  [*] thread usage analysis
4          threads launched: 3
5          threads supported by hardware: 8
6  [*] lock contention analysis
7          Running time: PT167.455753568S
8          Contention for mutex 35 is low: 0.9077024632556219 per second
9          Contention for mutex 10696 is moderate: 8.599286494000628 per second
10         Mutex info: MutexStat { id: 10696, locked: 30761, changed: 3982,
               contention: 1440, contention_time: 2021.529, total_time: 2335.671,
               avg_time: 0.076, flags: "Mx.--." }
11         Stacktrace:
12                 /usr/local/lib/libmutrace.so(pthread_mutex_init+0x1b0) [0
                       x7f80efbadfd5]
13                 /lib64/libxcb.so.1(xcb_connect_to_fd+0x8e) [0x7f80e6ff3c5e]
14
15         Contention for mutex 11227 is low: 0.017915180195834643 per second
```

4.3 Анализ программы BIRD

Следующим проектом, над которым был выполнен анализ, является демон маршрутизации BIRD[30]. Демон запускался, устанавливал связь с одним соседом по протоколу BGP и BFD, и завершался. Пример диагностик приведён в листинге 4.14. Из отчёта можно заключить, что демон использует

многопоточность аккуратно, но содержит множество не потокобезопасных функций, что может сказываться на работе данной программы.

Листинг 4.14 — bird.txt

```
[*] statical analysis of source code
[*] lock usage analysis
    io.c:464:8: warning: Non-reentrant function used [-Wph-non-reentrant-
        function]
      tm = localtime(&t);
           ^
    main.c:434:43: warning: Non-reentrant function used [-Wph-non-reentrant-
        function]
           log(L_INFO "CLI connection dropped: %s", strerror(err));
                                                    ^
    main.c:664:8: warning: Non-reentrant function used [-Wph-non-reentrant-
        function]
      pw = getpwnam(s);
           ^
    main.c:687:8: warning: Non-reentrant function used [-Wph-non-reentrant-
        function]
      gr = getgrnam(s);
           ^
    main.c:712:15: warning: Non-reentrant function used [-Wph-non-reentrant-
        function]
      while ((c = getopt(argc, argv, opt_list)) >= 0)
                ^
    4 warnings generated.
    2 warnings generated.
    netlink.c:331:24: warning: unused function 'rta_get_ip4' [-Wunused-function
        ]
    static inline ip4_addr rta_get_ip4(struct rtattr *a)
                          ^
    netlink.c:334:24: warning: unused function 'rta_get_ip6' [-Wunused-function
        ]
    static inline ip6_addr rta_get_ip6(struct rtattr *a)
                          ^
    netlink.c:802:1: warning: unused function 'nh_bufsize' [-Wunused-function]
    nh_bufsize(struct mpnh *nh)
    ^
    printf.c:236:8: warning: Non-reentrant function used [-Wph-non-reentrant-
        function]
                          s = strerror(errno);
                              ^
    printf.c:239:8: warning: Non-reentrant function used [-Wph-non-reentrant-
        function]
                          s = strerror(va_arg(args, int));
                              ^
    2 warnings generated.
```

```
36      netlink.c:181:51: warning: Non-reentrant function used [-Wph-non-reentrant-
            function]
37          log_rl(&rl_netlink_err, L_WARN "Netlink: %s", strerror(ec));
38                                                        ^
39      4 warnings generated.
40      cf-lex.c:1977:13: warning: unused function 'yy_fatal_error' [-Wunused-
            function]
41      static void yy_fatal_error (yyconst char* msg )
42                  ^
43      cf-lex.l:360:30: warning: Non-reentrant function used [-Wph-non-reentrant-
            function]
44            sprintf(patt, "%s/%s", dirname(dir), arg);
45                                   ^
46      2 warnings generated.
47  [*] thread usage analysis
48          threads launched: 2
49          threads supported by hardware: 4
50  [*] lock contention analysis
51          Running time: PT87.206834821S
52          Contention for mutex 1 is low: 0.011466991114315653 per second
```

4.4 Анализ программы spider-cpp

spider-cpp[31] — поисковый робот (краулер), рекурсивно отыскивающий ссылки на веб-странице и сохраняющий файлы заданного пользователем формата. Проект активно использует многопоточность. В частности, используется пул потоков, разработанный внутри проекта.

Статический анализ spider-cpp обнаружил неэффективную реализацию атомарного счётчика с помощью мьютексов. Диагностика ph приведена в листинге 4.15.

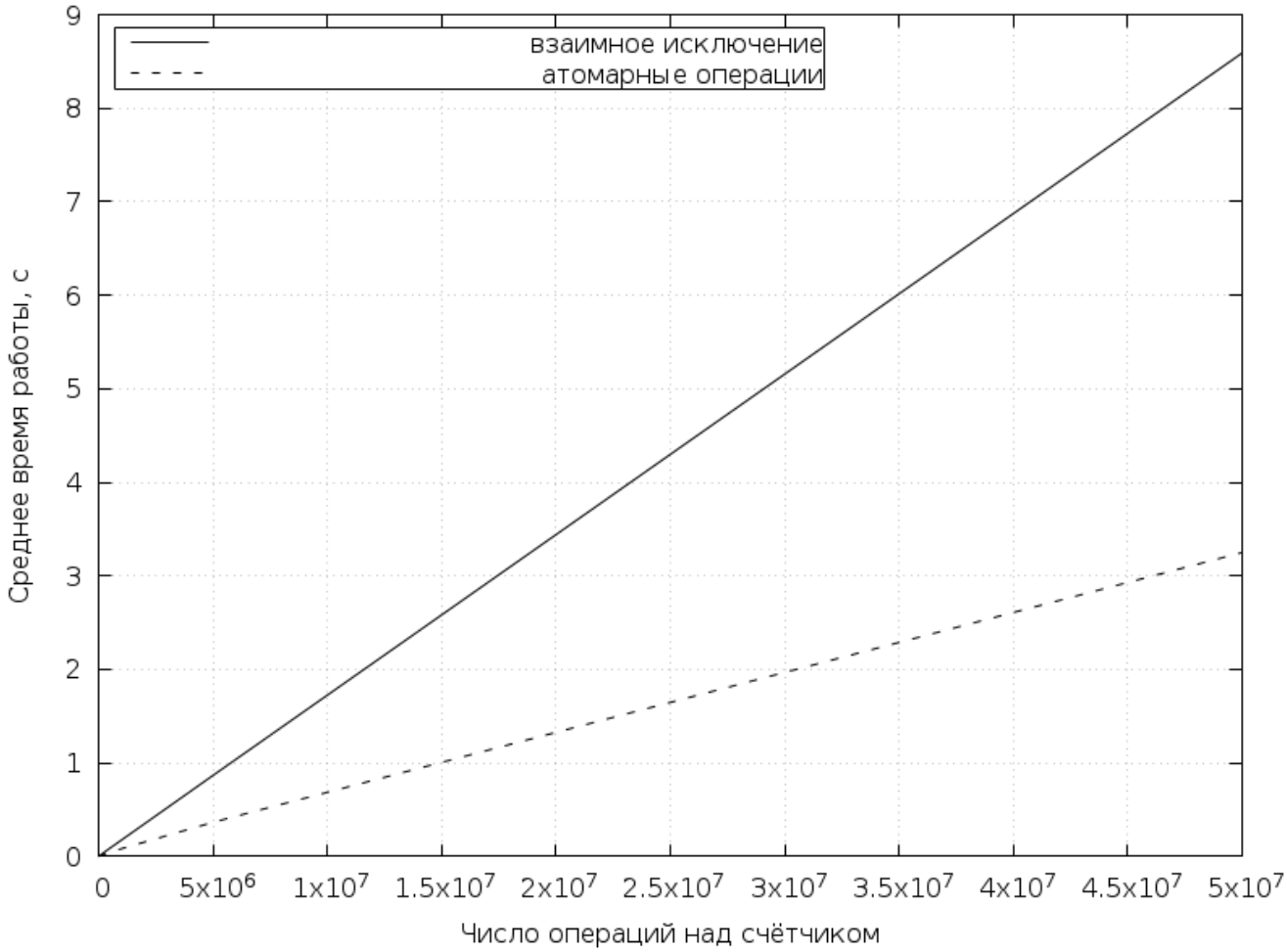
Анализируемый проект разделён на хорошо инкапсулированные классы, что позволило использовать для анализа эффективности лишь интересуемый класс счётчика. Была создана тестовая программа, которая использовала счётчик из нескольких потоков. Половина запускаемых потоков увеличивала счётчик заданное количество раз, половина — уменьшала. Производился замер времени выполнения всех потоков в зависимости от количества операций над счётчиком. После проведения измерений на оригинальном исходном коде, согласно диагностике ph примитив синхронизации был заменён на атомарный счётчик. На основании замеров времени до и после модификаций кода был построен график 4.1. Легко увидеть, что модифицированный код выполняется быстрее оригинального, а следовательно, с помощью ph получен прирост производительности.

Листинг 4.15 — spider-cpp.txt

```
[*] statical analysis of source code
[*] lock usage analysis
        In file included from /home/boris/projects/mine/ph/test/cxx_spider/src/
            counter.cpp:2:
        /home/boris/projects/mine/ph/test/cxx_spider/src/counter.hpp:8:11:
            warning: padding size of 'spider::Counter' with 4 bytes to alignment
             boundary [-Wpadded]
            class Counter {
                  ^
        1 warning generated.
        /home/boris/projects/mine/ph/test/cxx_spider/src/test_counter.cpp
            :16:24: warning: declaration requires a global constructor [-Wglobal
            -constructors]
        static spider::Counter counter;
                              ^~~~~~~
        In file included from /home/boris/projects/mine/ph/test/cxx_spider/src/
            test_counter.cpp:1:
        /home/boris/projects/mine/ph/test/cxx_spider/src/counter.hpp:8:11:
            warning: padding size of 'spider::Counter' with 4 bytes to alignment
             boundary [-Wpadded]
            class Counter {
                  ^
        2 warnings generated.

        [Lock information]
                Location:               /home/boris/projects/mine/ph/test/
                    cxx_spider/src/counter.cpp:13
                array subscripts:       0
                pointer manipulations: 0
                trivial calculations:  1
                declarations:           0
                calls:                  0

        [Suggestion]
        Lock used only in trivial calculations. Consider using atomic
            operations.
        [Lock information]
                Location:               /home/boris/projects/mine/ph/test/
                    cxx_spider/src/counter.cpp:21
                array subscripts:       0
                pointer manipulations: 0
                trivial calculations:  0
                declarations:           0
                calls:                  1

        No suggestions
[*] thread usage analysis
        threads launched: 3
```

```
38          threads supported by hardware: 4
39  [*] lock contention analysis
40          Running time: PT17.019211800S
41          Contention for mutex 1 is too high: 87440.06582020443 per second
42          Mutex info: MutexStat { id: 1, locked: 21000000, changed: 1488482,
                contention: 1488161, contention_time: 11794.427, total_time:
                3297.258, avg_time: 0, flags: "M-.--." }
43          Stacktrace:
44                  /usr/local/lib/libmutrace.so(pthread_mutex_lock+0x119) [0
                        x7f8acabf843c]
45                  test/cxx_spider/build/run_spider(spider::Counter::decrement()+0
                        x1d) [0x401c9d]
```

Рисунок 4.1 — Скорость выполнения тестовой программы при разных методах синхронизации
Число потоков = 2
Intel (R) Core (TM) i3-2100 CPU @ 3.10GHz, 8GB RAM



5 Ограничения и возможные расширения инструмента

Использование существующих технологий (LLVM/Clang, mutrace) вместо создания инструмента с чистого листа упростило создание и расширение анализатора, позволив сфокусироваться на непосредственной логике анализа. Вместе с тем, применение этих инструментов наложило некоторые ограничения на использование ph.

5.1 Ограничения

Во-первых, в сравнении с набором компиляторов GCC, инфраструктура LLVM/Clang поддерживает существенно меньшее количество целевых архитектур: порядка 20 против 11[32], что сужает область применения инструмента.

Во-вторых, отдельные части анализатора тесно привязаны к конкретной реализации системных библиотек и используемого компилятора. В частности, утилита mutrace может применяться только в Unix-системах с системной библиотекой glibc и аллокатором памяти из этой библиотеки. Динамический анализ кода становится невозможным при использовании других библиотек. К примеру, некоторые крупные проекты (Mozilla Firefox, Chromium) используют библиотеку jemalloc для меньшей фрагментированности памяти[33][34]

В-третьих, наиболее полезные рекомендации вырабатываются инструментом при одновременном применении всех стадий анализа. Такое поведение затрудняет использование инструмента при кросс-компиляции параллельной программы.

Наконец, разработанный инструмент не пытается отыскать все возможные ошибки в параллельных программах. Концентрация на анализе использования примитивов синхронизации упростила создание ph, оставив поиск неопределённого поведения и иных ошибок другим инструментам. Тем не менее, расширение ph легко осуществить, добавив необходимые проверки.

5.2 Расширения

Благодаря модульной архитектуре, расширение ph не составляет большого труда. Для добавления новой проверки на этапе синтаксического анализа требуется лишь создание двух файлов и включение их в систему сборки. Добавление других возможностей может быть не таким тривиальным. Наиболее полезными расширениями анализатора могут являться:

- возможность запускать исследуемую программу с пользовательским набором аргументов;
- возможность настраивать список функций/объявлений, которые считаются началом и окончанием критической секции;

- поддержка большего количества системных библиотек (musl, jemalloc);
- возможность сохранять результаты отдельных этапов анализа и выводить рекомендации на основе нескольких запусков инструмента;
- более тщательный анализ целевой платформы: сбор информации о доступных аппаратных средствах и реализациях примитивов синхронизации;
- интеграция с интегрированными средами разработки (IDE).

ЗАКЛЮЧЕНИЕ

В рамках работы проведено исследование существующих методов синхронизации параллельных программ, разобраны преимущества и недостатки каждого из подходов. Основываясь на особенностях различных алгоритмов синхронизации, выработана методика их подбора для существующего программного обеспечения.

Спроектирован и разработан инструмент, выдающий рекомендации по использованию подходящих методов синхронизации и позволяющий оценить эффективность от их применения. Проведены эксперименты на синтетических тестах и существующих проектах. Продемонстрирован результат применения рекомендаций инструмента к существующему программному обеспечению. Показаны ограничения по использованию инструмента, предложены дальнейшие способы расширения возможностей инструмента.

Разработанная методология позволяет упростить процесс выбора способа синхронизации существующего программного обеспечения. Продемонстрирована польза применения статического и динамического анализа для профилирования и автоматической генерации диагностик.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] [llvm-dev] RFC: EfficiencySanitizer. — 2016. — URL: http://lists.llvm.org/pipermail/llvm-dev/2016-April/098355.html (online; accessed: 17.04.2016).

[2] Introduction to Parallel Computing - Performance Analysis and Tuning. — 2016. — URL: https://computing.llnl.gov/tutorials/parallel_comp/#DesignPerformance (online; accessed: 18.04.2016).

[3] Dijkstra E. W. Solution of a Problem in Concurrent Programming Control // Commun. ACM. — 1965. — Sep. — Vol. 8, no. 9. — P. 569–. — URL: http://doi.acm.org/10.1145/365559.365617.

[4] pthread_mutex_lock. — 2016. — URL: http://pubs.opengroup.org/onlinepubs/009695399/functions/pthread_mutex_lock.html (online; accessed: 21.04.2016).

[5] Lock (Java Platform SE 7). — 2016. — URL: https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/locks/Lock.html#tryLock() (online; accessed: 21.04.2016).

[6] std::sync::Mutex - Rust. — 2016. — URL: https://doc.rust-lang.org/std/sync/struct.Mutex.html#method.try_lock (online; accessed: 21.04.2016).

[7] Dijkstra Edsger W. Cooperating sequential processes. — 1968. — published as [35]. URL: http://www.cs.utexas.edu/users/EWD/ewd01xx/EWD123.PDF.

[8] Herlihy M., Shavit N. The Art of Multiprocessor Programming. — Elsevier Science, 2011. — ISBN: 9780080569581. — URL: https://books.google.ru/books?id=pFSwuqtJgxYC.

[9] Williams A. C++ Concurrency in Action: Practical Multithreading. Manning Pubs Co Series. — Manning, 2012. — ISBN: 9781933988771. — URL: https://books.google.ru/books?id=EttPPgAACAAJ.

[10] Memory leaks are Memory Safe | Huon on the internet. — 2016. — URL: https://huonw.github.io/blog/2016/04/memory-leaks-are-memory-safe/ (online; accessed: 22.04.2016).

[11] Транзакционная память: история и развитие / Хабрахабр. — 2016. — URL: https://habrahabr.ru/post/221667/ (online; accessed: 22.04.2016).

[12] Rock (processor) - Wikipedia, the free encyclopedia. — 2016. — URL: https://en.wikipedia.org/wiki/Rock_(processor) (online; accessed: 22.04.2016).

[13] IBM's new transactional memory: make-or-break time for multithreaded revolution | Ars Technica. — 2016. — URL: http://arstechnica.com/gadgets/2011/08/ibms-new-transactional-memory-make-or-break-time-for-multithreaded-revolution/ (online; accessed: 22.04.2016).

[14] Sequoia - BlueGene/Q, Power BQC 16C 1.60 GHz, Custom | TOP500 Supercomputer Sites. — 2016. — URL: http://top500.org/system/177556 (online; accessed: 22.04.2016).

[15] Errata prompts Intel to disable TSX in Haswell, early Broadwell CPUs - The Tech Report. — 2016. — URL: https://techreport.com/news/26911/errata-prompts-intel-to-disable-tsx-in-haswell-early-broadwell-cpus (online; accessed: 22.04.2016).

[16] Transactional memory going mainstream with Intel Haswell | Ars Technica. — 2016. — URL: http://arstechnica.com/business/2012/02/transactional-memory-going-mainstream-with-intel-haswell/ (online; accessed: 22.04.2016).

[17] 3. Software Transactional Memory - School of Haskell. — 2016. — URL: https://www.schoolofhaskell.com/school/advanced-haskell/beautiful-concurrency/3-software-transactional-memory (online; accessed: 22.04.2016).

[18] Software Transactional Memory in GCC. — 2016. — URL: http://www-users.cs.umn.edu/~boutcher/stm/ (online; accessed: 22.04.2016).

[19] Technical Specification for C++ Extensions for Transactional Memory. — 2016. — URL: http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2015/n4514.pdf (online; accessed: 22.04.2016).

[20] TransactionalMemory - GCC Wiki. — 2016. — URL: https://gcc.gnu.org/wiki/TransactionalMemory (online; accessed: 22.04.2016).

[21] General Information - Thread-Safety. — 2016. — URL: http://pubs.opengroup.org/onlinepubs/9699919799/functions/V2_chap02.html#tag_15_09_01 (online; accessed: 18.04.2016).

[22] The LLVM Compiler Infrastructure Project. — 2016. — URL: http://llvm.org/ (online; accessed: 18.04.2016).

[23] Choosing the Right Interface for Your Application – Clang 3.9 documentation. — 2016. — URL: http://clang.llvm.org/docs/Tooling.html (online; accessed: 18.04.2016).

[24] Clang Plugins. — 2016. — URL: http://clang.llvm.org/docs/ClangPlugins.html (online; accessed: 21.04.2016).

[25] KDE/clazy: Qt oriented code checker based on clang framework. Krazy's little brother. — 2016. — URL: https://github.com/KDE/clazy (online; accessed: 21.04.2016).

[26] Design Patterns: Elements of Reusable Object-oriented Software / Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides. — Boston, MA, USA : Addison-Wesley Longman Publishing Co., Inc., 1995. — ISBN: 0-201-63361-2.

[27] mutrace.git - mutrace Mutual Exclusion Tracer. — 2016. — URL: http://git.0pointer.net/mutrace.git/ (online; accessed: 18.04.2016).

[28] The Rust Programming Language. — 2016. — URL: https://www.rust-lang.org/ (online; accessed: 22.04.2016).

[29] Kate | Get an Edge in Editing | The Kate Editor Homepage. — 2016. — URL: https://kate-editor.org/ (online; accessed: 22.04.2016).

[30] The BIRD Internet Routing Daemon Project. — 2016. — URL: http://bird.network.cz/ (online; accessed: 22.04.2016).

[31] jehugaleahsa/spider-cpp: Recursively scans HTML pages for URLs and downloads desired content. — 2016. — URL: https://github.com/jehugaleahsa/spider-cpp (online; accessed: 22.04.2016).

[32] GCC vs LLVM - WikiVS. — 2016. — URL: https://www.wikivs.com/wiki/GCC_vs_LLVM#Hardware_Architectures (online; accessed: 21.04.2016).

[33] Adoption - jemalloc/jemalloc Wiki. — 2016. — URL: https://github.com/jemalloc/jemalloc/wiki/Adoption (online; accessed: 21.04.2016).

[34] chromium/base/allocator at baster - ChromiumWebApps/chromium. — 2016. — URL: https://github.com/ChromiumWebApps/chromium/tree/master/base/allocator (online; accessed: 21.04.2016).

[35] Dijkstra Edsger W. Cooperating sequential processes // Programming Languages: NATO Advanced Study Institute / Ed. by F. Genuys. — Academic Press, 1968. — P. 43–112.

ПРИЛОЖЕНИЕ А. Исходный код главной программы пакета

В данном приложении представлена главная программа (main) разработанного приложения.

Листинг А.5.16 — Cargo.toml

```
1  [package]
2  name = "ph"
3  version = "0.1.0"
4  authors = ["Boris Egorov <egorov@linux.com>"]
5
6  [dependencies]
7  clap = "2"
8  num_cpus = "0.2"
9  time = "0.1"
10 clippy = { version = "*", optional = true }
11
12 [features]
13 default = []
```

Листинг А.5.17 — main.rs

```
1  #![feature(question_mark)]
2  #![cfg_attr(feature="clippy", feature(plugin))]
3  #![cfg_attr(feature="clippy", plugin(clippy))]
4
5  #[macro_use]
6  extern crate clap;
7  extern crate num_cpus;
8  extern crate time;
9
10 use std::io;
11
12 mod common;
13 mod statical;
14 mod dynamic;
15
16 arg_enum!{
17     #[derive(Debug, Clone, Copy)]
18     pub enum ProjectLanguage {
19         C,
20         CXX
21     }
22 }
23
24 #[derive(Debug)]
25 pub enum PhError {
```

```
26      IoError,
27      ProcessError,
28      InvalidData,
29      MutexStatParseError,
30      BuildError,
31      TimeError,
32  }
33
34  pub type Result<T> = std::result::Result<T, PhError>;
35
36  impl From<io::Error> for PhError {
37      fn from(_: io::Error) -> Self {
38          PhError::IoError
39      }
40  }
41
42  impl From<std::num::ParseIntError> for PhError {
43      fn from(_: std::num::ParseIntError) -> Self {
44          PhError::InvalidData
45      }
46  }
47
48  impl From<std::num::ParseFloatError> for PhError {
49      fn from(_: std::num::ParseFloatError) -> Self {
50          PhError::InvalidData
51      }
52  }
53
54  impl From<time::OutOfRangeError> for PhError {
55      fn from(_: time::OutOfRangeError) -> Self {
56          PhError::TimeError
57      }
58  }
59
60  fn _run_lock_usage_analyzer() {
61      println!("\tMost popular actions during lock:");
62      println!("\t\tincrementing value - consider using atomic");
63      println!("\t\tmanipulating distinct addreses - consider using TM");
64      println!("\t\tmad lock contention - consider different arhitecture");
65  }
66
67  fn main() {
68      use clap::{Arg, App};
69
70      let app = App::new("ph")
71          .version("0.1")
72          .author("Boris Egorov <egorov@linux.com>")
73          .arg(Arg::with_name("lang")
```

```
74                .possible_values(&ProjectLanguage::variants())
75                .short("l")
76                .long("project-language")
77                .required(true)
78                .help("Project language (C or CXX)")
79                .takes_value(true))
80            .arg(Arg::with_name("project_path")
81                .short("p")
82                .long("project-path")
83                .required(true)
84                .help("Path to project root (sources)")
85                .takes_value(true))
86            .arg(Arg::with_name("binary_path")
87                .short("b")
88                .long("binary-path")
89                .required(true)
90                .help("Path to program executable")
91                .takes_value(true));
92
93        let matches = app.get_matches();
94
95        let project_path = matches.value_of("project_path").unwrap();
96        let binary_path = matches.value_of("binary_path").unwrap();
97        let language = value_t!(matches.value_of("lang"), ProjectLanguage).
              unwrap_or_else(|e| e.exit());
98
99        match statical::run_static_analyzer(project_path, language) {
100           Ok(()) => (),
101           Err(PhError::BuildError) => println!("[ERROR] Build error. Static
                  analysis skipped"),
102           Err(e) => println!("[ERROR] {:?}", e),
103       }
104       match dynamic::run_mutrace(binary_path, language) {
105           Ok(()) => (),
106           Err(e) => println!("[ERROR] {:?}", e),
107       }
108   }
```

Листинг A.5.18 — common.rs

```
1  use std::process::Output;
2
3  pub fn print_output(output: Output) {
4      println!("{} {}",
5               String::from_utf8_lossy(&output.stdout),
6               String::from_utf8_lossy(&output.stderr));
7  }
```

Листинг A.5.19 — static.rs

```
1  use std::default::Default;
2  use std::fmt;
3  use std::fs::{self, File};
4  use std::io::BufReader;
5  use std::io::prelude::*;
6  use std::path::PathBuf;
7  use std::process::Command;
8  use ::ProjectLanguage;
9  use ::PhError;
10 use ::Result;
11 use common::print_output;
12
13 const PH_CLANG_OUTPUT_FILE : &'static str = "/tmp/criticalsection_usage.txt";
14 const SUGGESTION_ATOMIC : &'static str = "\tLock used only in trivial
       calculations. \
15 Consider using atomic operations.";
16 const SUGGESTION_TM : &'static str = "\tLock uses various parts of memory. \
17 Consider using transactional memory.";
18
19 #[derive(Debug)]
20 struct Lock {
21     array_subscripts: u32,
22     pointer_manipulations: u32,
23     trivial_calculations: u32,
24     declarations: u32,
25     calls: u32,
26     filename: String,
27     lineno: u32,
28 }
29
30 impl Default for Lock {
31     fn default() -> Self {
32         Lock {
33             array_subscripts: 0,
34             pointer_manipulations: 0,
35             trivial_calculations: 0,
36             declarations: 0,
37             calls: 0,
38             filename: String::new(),
39             lineno: 0,
40         }
41     }
42 }
43
44 impl fmt::Display for Lock {
45     fn fmt(&self, f: &mut fmt::Formatter) -> fmt::Result {
46         try!(writeln!(f, "\t[Lock information]"));
47         try!(writeln!(f, "\t\tLocation:              {}:{}", self.filename,
```

```
            self.lineno));
48          try!(writeln!(f, "\t\tarray subscripts:      {}", self.array_subscripts
                ));
49          try!(writeln!(f, "\t\tpointer manipulations: {}", self.
                pointer_manipulations));
50          try!(writeln!(f, "\t\ttrivial calculations:  {}", self.
                trivial_calculations));
51          try!(writeln!(f, "\t\tdeclarations:          {}", self.declarations));
52          writeln!(f, "\t\tcalls:                {}", self.calls)
53      }
54  }
55
56  fn cmake_config(project_root: PathBuf) -> Result<PathBuf> {
57      let mut build_directory = project_root;
58      build_directory.push("build");
59
60      let output = try!(Command::new("mkdir")
61          .arg("-p")
62          .arg(&build_directory)
63          .output());
64      if !output.status.success() {
65          print_output(output);
66          return Err(PhError::ProcessError);
67      }
68
69      let output = try!(Command::new("/usr/bin/cmake")
70          .current_dir(&build_directory)
71          .arg("..")
72          .arg("-DCMAKE_C_COMPILER=ph")
73          .arg("-DCMAKE_CXX_COMPILER=ph_cxx")
74          .arg("-DCMAKE_CXX_FLAGS='-rdynamic -ggdb3'")
75          .output());
76      if !output.status.success() {
77          print_output(output);
78          return Err(PhError::ProcessError);
79      }
80      Ok(build_directory)
81  }
82
83  fn print_lock_usage_suggestions(lock_usage: Lock) {
84      println!("{}", lock_usage);
85      if lock_usage.array_subscripts == 0 &&
86          lock_usage.pointer_manipulations == 0 &&
87          lock_usage.declarations == 0 &&
88          lock_usage.calls == 0 &&
89          lock_usage.trivial_calculations != 0 {
90          println!("\t[Suggestion]");
91          println!("{}", SUGGESTION_ATOMIC);
```

```
92      } else if (lock_usage.array_subscripts != 0 ||
93          lock_usage.pointer_manipulations != 0) &&
94          lock_usage.calls == 0 {
95          println!("\t[Suggestion]");
96          println!("{}", SUGGESTION_TM);
97      } else {
98          println!("\tNo suggestions");
99      }
100 }
101
102 fn analyse_output(lock_stat_file: &str) -> Result<()> {
103     let file = try!(File::open(lock_stat_file));
104     let reader = BufReader::new(file);
105
106     for line in reader.lines() {
107         let line = try!(line);
108         let words = line.split_whitespace().map(|s| s.to_string()).collect::<
                Vec<String>>();
109         let mut lock_usage = Lock::default();
110         lock_usage.array_subscripts = try!(words[0].parse());
111         lock_usage.pointer_manipulations = try!(words[1].parse());
112         lock_usage.trivial_calculations = try!(words[2].parse());
113         lock_usage.declarations = try!(words[3].parse());
114         lock_usage.calls = try!(words[4].parse());
115         lock_usage.lineno = try!(words[5].parse());
116         lock_usage.filename = words[6].to_string();
117         print_lock_usage_suggestions(lock_usage);
118     }
119     Ok(())
120 }
121
122 pub fn run_static_analyzer(path: &str, _: ProjectLanguage) -> Result<()> {
123     let path = PathBuf::from(path);
124
125     let mut cmake_path = path.clone();
126     cmake_path.push("CMakeLists.txt");
127
128     let mut make_path = path.clone();
129     make_path.push("Makefile");
130
131     // TODO: use cmake information to get binary
132     let build_root = if cmake_path.exists() {
133         let root = try!(cmake_config(path));
134         root
135     } else if make_path.exists() {
136         path
137     } else {
138         return Err(PhError::BuildError);
```

```
139     };
140
141     // Clean to get warnings anew
142     let output = try!(Command::new("make")
143         .current_dir(&build_root)
144         .arg("clean")
145         .output());
146     if !output.status.success() {
147         print_output(output);
148         return Err(PhError::ProcessError);
149     }
150
151     fs::remove_file(PH_CLANG_OUTPUT_FILE);
152
153     println!("[*] statical analysis of source code");
154     println!("[*] lock usage analysis");
155     let output = try!(Command::new("make")
156         .current_dir(&build_root)
157         .env("CXX", "ph_cxx")
158         .env("CC", "ph")
159         .arg("-j")
160         .arg(format!("{}", ::num_cpus::get()))
161         .output());
162
163     if !PathBuf::from(PH_CLANG_OUTPUT_FILE).exists() {
164         println!("[WARNING] No locks detected in source code");
165         Ok(())
166     } else {
167         let output = String::from_utf8_lossy(&output.stderr);
168         for line in output.split('\n') {
169             println!("\t{}", line);
170         }
171         analyse_output(PH_CLANG_OUTPUT_FILE)
172     }
173 }
```

Листинг А.5.20 — dynamic.rs

```
1  use std::env;
2  use std::f64;
3  use std::fs::{self, File};
4  use std::io::prelude::*;
5  use std::io::BufReader;
6  use std::process::{Command, Stdio};
7  use std::path::Path;
8  use std::str::FromStr;
9  use time::PreciseTime;
10 use ProjectLanguage;
11 use PhError;
```

```
12 use Result;
13 use common::print_output;
14
15 #[derive(Clone, Debug)]
16 struct MutexStat {
17     id: u32,
18     locked: u32,
19     changed: u32,
20     contention: u32,
21     contention_time: f32,
22     total_time: f32,
23     avg_time: f32,
24     flags: String,
25 }
26
27 impl MutexStat {
28     pub fn new(id: u32) -> MutexStat {
29         MutexStat {
30             id: id,
31             locked: 0,
32             changed: 0,
33             contention: 0,
34             contention_time: 0f32,
35             total_time: 0f32,
36             avg_time: 0f32,
37             flags: String::new(),
38         }
39     }
40 }
41
42 impl FromStr for MutexStat {
43     type Err = PhError;
44     fn from_str(s: &str) -> Result<Self> {
45         let words = s.split_whitespace().map(|s| s.to_string()).collect::<Vec<
               String>>();
46         if words.len() != 8 {
47             Err(PhError::MutexStatParseError)
48         } else {
49             let mut stat = MutexStat::new(words[0].parse::<u32>()?);
50             stat.locked = try!(words[1].parse::<u32>());
51             stat.changed = try!(words[2].parse::<u32>());
52             stat.contention = try!(words[3].parse::<u32>());
53             stat.contention_time = try!(words[4].parse::<f32>());
54             stat.total_time = try!(words[5].parse::<f32>());
55             stat.avg_time = try!(words[6].parse::<f32>());
56             stat.flags = words[7].to_string();
57             Ok(stat)
58         }
```

```
59      }
60  }
61
62  #[derive(Debug)]
63  struct MutexInfo<'a> {
64      trace: String,
65      stat: &'a MutexStat,
66  }
67
68  fn get_mutex_data<P: AsRef<Path>>(path: P, stats: &[MutexStat]) -> Result<Vec<
        MutexInfo>> {
69      let file = try!(File::open(path));
70      let reader = BufReader::new(file);
71      let mut mutex_ids = vec![];
72      let mut header = true;
73      let mut trace = String::new();
74
75      for line in reader.lines() {
76          let line = try!(line);
77          if header {
78              let id_width = 6;
79              let (number_part, _) = line.split_at(id_width);
80
81              let id = try!(number_part.trim().parse::<u32>());
82              for stat in stats.iter().filter(|stat| stat.id == id) {
83                  mutex_ids.push(MutexInfo {
84                      trace: String::new(),
85                      stat: stat,
86                  });
87              }
88              header = false;
89          } else if line.is_empty() {
90              let last_pos = mutex_ids.len() - 1;
91              mutex_ids[last_pos].trace = trace;
92              trace = String::new();
93              header = true;
94          } else {
95              trace.push_str(&line);
96              trace.push('\n');
97          }
98      }
99      Ok(mutex_ids)
100 }
101
102 fn get_summary<P: AsRef<Path>>(path: P) -> Result<Vec<MutexStat>> {
103     let file = try!(File::open(path));
104     let reader = BufReader::new(file);
105
```

```
106     let mut stats = vec![];
107
108     for line in reader.lines() {
109         let line = try!(line);
110
111         let stat = try!(line.parse::<MutexStat>());
112         stats.push(stat);
113     }
114     Ok(stats)
115 }
116
117 fn get_max_threads<P: AsRef<Path>>(path: P) -> Result<usize> {
118     let file = try!(File::open(path));
119     let mut s = String::new();
120     let mut reader = BufReader::new(file);
121
122     try!(reader.read_line(&mut s));
123     let nthreads = try!(s.trim().parse::<usize>());
124     Ok(nthreads)
125 }
126
127 enum ContentionRank {
128     No,
129     Low,
130     Moderate,
131     High,
132     Extreme,
133 }
134
135 fn contention_per_second(contention: u32, time: ::time::Duration) -> Result<f64
    > {
136     let time = try!(time.to_std());
137     let time = time.as_secs() * 1_000_000_000 + time.subsec_nanos() as u64;
138     let time = time as f64 / 1_000_000_000f64;
139
140     if contention == 0 || time < 1e-9 {
141         Ok(0f64)
142     } else {
143         Ok(contention as f64 / time)
144     }
145 }
146
147 fn contention_rank(contention_per_second: f64) -> ContentionRank {
148     if contention_per_second <= f64::EPSILON {
149         ContentionRank::No
150     } else if contention_per_second < 5.0 {
151         ContentionRank::Low
152     } else if contention_per_second < 50.0 {
```

```
153             ContentionRank::Moderate
154         } else if contention_per_second < 100.0 {
155             ContentionRank::High
156         } else {
157             ContentionRank::Extreme
158         }
159 }
160
161 fn print_trace(trace: &str, language: ProjectLanguage) -> Result<()> {
162     println!("\tStacktrace: ");
163     match language {
164         ProjectLanguage::C => {
165             println!("\t{}", trace);
166         }
167         ProjectLanguage::CXX => {
168             // demangling
169             let mut cmd = try!(Command::new("c++filt")
170                 .stdin(Stdio::piped())
171                 .stdout(Stdio::piped())
172                 .spawn());
173
174             if let Some(stdin) = cmd.stdin.as_mut() {
175                 try!(stdin.write_all(trace.as_bytes()));
176             }
177
178             let output = try!(cmd.wait_with_output());
179             for line in String::from_utf8_lossy(&output.stdout).split('\n') {
180                 if !line.trim().is_empty() {
181                     println!("\t{}", line);
182                 }
183             }
184         }
185     }
186     println!("");
187     Ok(())
188 }
189
190 pub fn run_mutrace(binary_path: &str, language: ProjectLanguage) -> Result<()>
    {
191     let dir = env::temp_dir();
192
193     // It is most likely exists. Possible errors will be catched later
194     fs::create_dir(&dir);
195
196     let mut summary_file = dir.clone();
197     summary_file.push("summary.txt");
198     let mut mutex_info_file = dir.clone();
199     mutex_info_file.push("mutex_info.txt");
```

```
200     let mut max_threads_file = dir.clone();
201     max_threads_file.push("max_threads.txt");
202
203     let start = PreciseTime::now();
204     let output = try!(Command::new("mutrace")
205         .arg(binary_path)
206         .env("MUTRACE_SUMMARY_FILE", summary_file.as_os_str())
207         .env("MUTRACE_MUTEX_INFO_FILE", mutex_info_file.as_os_str())
208         .env("MUTRACE_MAX_THREADS_FILE", max_threads_file.as_os_str())
209         .output());
210     let end = PreciseTime::now();
211
212     if !output.status.success() {
213         print_output(output);
214         return Err(PhError::ProcessError);
215     }
216
217     let max_threads = try!(get_max_threads(&max_threads_file));
218     println!("[*] thread usage analysis");
219     let number_of_cpus = ::num_cpus::get();
220     println!("\tthreads launched: {}", max_threads);
221     println!("\tthreads supported by hardware: {}", number_of_cpus);
222     if max_threads > number_of_cpus * 2 {
223         println!("\tNOTE: number of threads can be too high");
224     }
225
226     let summary = try!(get_summary(&summary_file));
227     println!("[*] lock contention analysis");
228     let data = try!(get_mutex_data(&mutex_info_file, &summary));
229     let running_time = start.to(end);
230     println!("\tRunning time: {}", running_time);
231     for mutex in data {
232         let cps = try!(contention_per_second(mutex.stat.contention,
                running_time));
233         let rank = contention_rank(cps);
234         match rank {
235             ContentionRank::Extreme | ContentionRank::High => {
236                 println!("\tContention for mutex {} is too high: {} per second
                        ",
237                         mutex.stat.id,
238                         cps);
239                 println!("\tMutex info: {:?}", mutex.stat);
240                 try!(print_trace(&mutex.trace, language));
241             }
242             ContentionRank::Moderate => {
243                 println!("\tContention for mutex {} is moderate: {} per second
                        ",
244                         mutex.stat.id,
```

```
245                             cps);
246                     println!("\tMutex info: {:?}", mutex.stat);
247                     try!(print_trace(&mutex.trace, language));
248                 }
249                 ContentionRank::Low => {
250                     println!("\tContention for mutex {} is low: {} per second",
251                             mutex.stat.id,
252                             cps);
253                 }
254                 _ => (),
255             }
256         }
257 
258         fs::remove_file(&mutex_info_file);
259         fs::remove_file(&max_threads_file);
260         fs::remove_file(&summary_file);
261         Ok(())
262 }
```

ПРИЛОЖЕНИЕ Б. Исходный код анализатора кода

В данном приложении представлена часть программы, отвечающая за статический анализ кода.

Листинг Б.5.21 — criticalsection.h

```
1  /*
2     This file is part of the ph static checker.
3
4    Copyright (C) 2015 äKlarlvdalens Datakonsult AB, a KDAB Group company,
        info@kdab.com
5    Author: éSrgio Martins <sergio.martins@kdab.com>
6
7    Copyright (C) 2015 Sergio Martins <smartins@kde.org>
8
9    Copyright (C) 2015 Boris Egorov <egorov@linux.com>
10
11   This library is free software; you can redistribute it and/or
12   modify it under the terms of the GNU Library General Public
13   License as published by the Free Software Foundation; either
14   version 2 of the License, or (at your option) any later version.
15
16   This library is distributed in the hope that it will be useful,
17   but WITHOUT ANY WARRANTY; without even the implied warranty of
18   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
19   Library General Public License for more details.
20
21   You should have received a copy of the GNU Library General Public License
22   along with this library; see the file COPYING.LIB.  If not, write to
23   the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor,
24   Boston, MA 02110-1301, USA.
25 */
26
27 #ifndef CRITICAL_SECTION_H
28 #define CRITICAL_SECTION_H
29
30 #include "checkbase.h"
31 #include <bitset>
32 #include <vector>
33
34 enum class UnlockType {
35     FunctionCall, // like pthread_mutex_unlock
36     MemberFunctionCall, // like mutex::unlock()
37     EndOfScope, // like std::lock_guard::~lock_guard()
38 };
39
40 using Names = std::vector<std::string>;
41
```

```
42 struct LockUsage {
43     size_t nPointerManipulations;
44     size_t nArraySubscripts;
45     size_t nTrivialCalculations;
46     size_t nDeclarations;
47     size_t nCalls;
48 };
49
50 struct LockInfo {
51     LockInfo(const Names& locks, const Names& unlocks, std::bitset<3>
          unlock_types)
52         : lock_names{locks}
53         , unlock_names{unlocks}
54         , m_unlock_types{unlock_types}
55     {
56     }
57     std::vector<std::string> lock_names{};
58     std::vector<std::string> unlock_names{};
59     std::bitset<3> m_unlock_types{};
60 };
61
62 struct Lock{
63     bool dtor_unlock{false};
64     bool locked{false};
65     std::string filename{};
66     size_t lineno{0};
67     LockUsage usage{};
68 };
69
70 class CriticalSection : public CheckBase
71 {
72 public:
73     CriticalSection(const std::string &name, const clang::CompilerInstance &ci)
          ;
74     ~CriticalSection();
75     void VisitStmt(clang::Stmt *) override;
76 private:
77     std::vector<LockInfo> lock_infos;
78     std::vector<Lock> locks;
79 };
80
81 #endif // CRITICAL_SECTION_H
```

Листинг Б.5.22 — criticalsection.cpp

```
1 /*
2    This file is part of the ph static checker.
3
4   Copyright (C) 2015 äKlarlvdalens Datakonsult AB, a KDAB Group company,
```

```
        info@kdab.com
5     Author: éSrgio Martins <sergio.martins@kdab.com>
6
7     Copyright (C) 2015 Sergio Martins <smartins@kde.org>
8
9     Copyright (C) 2015 Boris Egorov <egorov@linux.com>
10
11    This library is free software; you can redistribute it and/or
12    modify it under the terms of the GNU Library General Public
13    License as published by the Free Software Foundation; either
14    version 2 of the License, or (at your option) any later version.
15
16    This library is distributed in the hope that it will be useful,
17    but WITHOUT ANY WARRANTY; without even the implied warranty of
18    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
19    Library General Public License for more details.
20
21    You should have received a copy of the GNU Library General Public License
22    along with this library; see the file COPYING.LIB.  If not, write to
23    the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor,
24    Boston, MA 02110-1301, USA.
25 */
26 #include "criticalsection.h"
27 #include "Utils.h"
28 #include "MacroUtils.h"
29 #include "StringUtils.h"
30 #include "checkmanager.h"
31
32 #include <clang/AST/Expr.h>
33 #include <clang/AST/Stmt.h>
34 #include <algorithm>
35 #include <fstream>
36
37 using namespace clang;
38 using namespace std;
39
40 namespace {
41
42 const char *kOutputFilename = "/tmp/criticalsection_usage.txt";
43
44 } // namespace
45
46 CriticalSection::CriticalSection(const std::string& name, const clang::
      CompilerInstance& ci)
47     : CheckBase(name, ci)
48 {
49     auto functionMask = std::bitset<3>{};
50     functionMask[0] = 1;
```

```
51
52      lock_infos.emplace_back(
53          std::vector<std::string>{
54              "pthread_mutex_lock",
55          },
56          std::vector<std::string>{
57              "pthread_mutex_unlock",
58          },
59          functionMask);
60
61      // GLib
62      lock_infos.emplace_back(
63          std::vector<std::string>{
64              "g_mutex_lock",
65          },
66          std::vector<std::string>{
67              "g_mutex_unlock",
68          },
69          functionMask);
70
71      auto scopeMask = std::bitset<3>{};
72      scopeMask[2] = 1;
73      lock_infos.emplace_back(
74          std::vector<std::string>{
75              "lock_guard",
76          },
77          std::vector<std::string>{},
78          scopeMask);
79
80      lock_infos.emplace_back(
81          std::vector<std::string>{
82              "QMutexLocker",
83          },
84          std::vector<std::string>{},
85          scopeMask);
86  }
87
88  CriticalSection::~CriticalSection()
89  {
90      auto out = std::ofstream{kOutputFilename, std::ios::app};
91      for (const auto& lock : locks) {
92          out << lock.usage.nArraySubscripts << " "
93              << lock.usage.nPointerManipulations << " "
94              << lock.usage.nTrivialCalculations << " "
95              << lock.usage.nDeclarations << " "
96              << lock.usage.nCalls << " "
97              << lock.lineno << " "
98              << lock.filename << "\n";
```

```
99      }
100 }
101
102 using LockInfoIt = std::vector<LockInfo>::iterator;
103 LockInfoIt find_lock(std::vector<LockInfo>& lock_infos, std::string funcName)
104 {
105     return clazy_std::find_if(lock_infos, [funcName](LockInfo& lock) {
106         return clazy_std::contains_if(lock.lock_names, [funcName](const std::
                string& name) {
107             return funcName.find(name) != std::string::npos;
108         });
109     });
110 }
111
112 LockInfoIt find_unlock(std::vector<LockInfo>& lock_infos, std::string funcName)
113 {
114     return clazy_std::find_if(lock_infos, [funcName](LockInfo& lock) {
115         return clazy_std::contains_if(lock.unlock_names, [funcName](const std::
                string& name) {
116             return funcName.find(name) != std::string::npos;
117         });
118     });
119 }
120
121 bool is_fundamental_variable(Expr* e)
122 {
123     if (e && !dyn_cast<ArraySubscriptExpr>(e)) {
124         return e->getType()->isFundamentalType();
125     }
126     return false;
127 }
128
129 bool is_trivial_calculation(UnaryOperator* op)
130 {
131     if (op->isIncrementDecrementOp() || op->isArithmeticOp()) {
132         return is_fundamental_variable(op->getSubExpr());
133     }
134     return false;
135 }
136
137 bool is_trivial_calculation(CompoundAssignOperator* op)
138 {
139     return is_fundamental_variable(op->getLHS());
140 }
141
142 void CriticalSection::VisitStmt(Stmt* stm)
143 {
144     auto compound = dyn_cast<CompoundStmt>(stm);
```

```
145     if (!compound) {
146         return;
147     }
148 
149     for (auto it = compound->body_begin(); it != compound->body_end(); ++it) {
150         auto declStmt = dyn_cast<DeclStmt>(*it);
151         if (declStmt) {
152             if (declStmt->isSingleDecl()) {
153                 auto usingDecl = dyn_cast<UsingDecl>(declStmt->getSingleDecl())
                        ;
154                 if (usingDecl) {
155                     continue;
156                 }
157             }
158             for (auto& lock : locks) {
159                 if (lock.locked) {
160                     ++lock.usage.nDeclarations;
161                 }
162             }
163             for (auto decl : declStmt->decls()) {
164                 auto d = dyn_cast<VarDecl>(decl);
165                 if (d) {
166                     auto name = StringUtils::fullTypeName(d->getType(), lo());
167 
168                     auto lock_it = find_lock(lock_infos, name);
169                     if (lock_it != lock_infos.end()) {
170                         locks.emplace_back();
171                         // std::lock_guard etc. Will end at the end of a scope
172                         auto& manager = this->m_ci.getSourceManager();
173                         auto loc = d->getLocStart();
174                         locks.back().lineno = manager.getPresumedLineNumber(loc
                                );
175                         locks.back().filename = manager.getFilename(loc);
176                         locks.back().locked = true;
177                         locks.back().dtor_unlock = true;
178                     }
179                 }
180             }
181         }
182 
183         auto callStmt = dyn_cast<CallExpr>(*it);
184         if (callStmt) {
185             for (auto& lock : locks) {
186                 if (lock.locked) {
187                     ++lock.usage.nCalls;
188                 }
189             }
190             FunctionDecl* func = callStmt->getDirectCallee();
```

```
191             if (func) {
192                 if (isa<CXXMethodDecl>(func))
193                     continue;
194 
195                 const std::string funcName = func->getNameAsString();
196                 auto lock_it = find_lock(lock_infos, funcName);
197                 if (lock_it != lock_infos.end()) {
198                     locks.emplace_back();
199                     auto& manager = this->m_ci.getSourceManager();
200                     auto loc = callStmt->getLocStart();
201                     locks.back().lineno = manager.getPresumedLineNumber(loc);
202                     locks.back().filename = manager.getFilename(loc);
203                     locks.back().locked = true;
204                 }
205                 auto unlock_it = find_unlock(lock_infos, funcName);
206                 if (unlock_it != lock_infos.end()) {
207                     for (auto rit = locks.rbegin(); rit != locks.rend(); ++rit)
                            {
208                         if (rit->locked && !rit->dtor_unlock) {
209                             if (rit->usage.nCalls) {
210                                 --rit->usage.nCalls; // do not count unlock
                                        call
211                             }
212                             rit->locked = false;
213                             break;
214                         }
215                     }
216                 }
217             }
218         }
219 
220         auto arraySubscript = dyn_cast<ArraySubscriptExpr>(*it);
221         if (arraySubscript) {
222             for (auto& lock : locks) {
223                 if (lock.locked) {
224                     ++lock.usage.nArraySubscripts;
225                 }
226             }
227         }
228 
229         auto unaryOperator = dyn_cast<UnaryOperator>(*it);
230         if (unaryOperator) {
231             if (is_trivial_calculation(unaryOperator)) {
232                 for (auto& lock : locks) {
233                     if (lock.locked) {
234                         ++lock.usage.nTrivialCalculations;
235                     }
236                 }
```

```
237             } else {
238                 // TODO: pessimistic. Can it be something other?
239                 for (auto& lock : locks) {
240                     if (lock.locked) {
241                         ++lock.usage.nPointerManipulations;
242                     }
243                 }
244             }
245             auto opcode = unaryOperator->getOpcode();
246             if (opcode == UO_AddrOf || opcode == UO_Deref) {
247                 for (auto& lock : locks) {
248                     if (lock.locked) {
249                         ++lock.usage.nPointerManipulations;
250                     }
251                 }
252             }
253         }
254 
255         auto compoundAssign = dyn_cast<CompoundAssignOperator>(*it);
256         if (compoundAssign) {
257             if (is_trivial_calculation(compoundAssign)) {
258                 for (auto& lock : locks) {
259                     if (lock.locked) {
260                         ++lock.usage.nTrivialCalculations;
261                     }
262                 }
263             } else {
264                 // TODO: pessimistic. Can it be something other?
265                 for (auto& lock : locks) {
266                     if (lock.locked) {
267                         ++lock.usage.nPointerManipulations;
268                     }
269                 }
270             }
271         }
272 
273         if (dyn_cast<IfStmt>(*it)
274             || dyn_cast<CompoundStmt>(*it)
275             || dyn_cast<ForStmt>(*it)
276             || dyn_cast<WhileStmt>(*it)
277             || dyn_cast<DoStmt>(*it)) {
278             // TODO: pessimistic
279             for (auto& lock : locks) {
280                 if (lock.locked) {
281                     ++lock.usage.nCalls;
282                 }
283             }
284         }
```

```
285     }
286     for (auto& lock : locks) {
287         if (lock.dtor_unlock) {
288             lock.locked = false;
289         }
290     }
291 }
292
293 REGISTER_CHECK_WITH_FLAGS("critical-section", CriticalSection, CheckLevel0)
```

Листинг Б.5.23 — nonreentrantfunctions.h

```
1  /*
2     This file is part of the ph static checker.
3
4    Copyright (C) 2015 äKlarlvdalens Datakonsult AB, a KDAB Group company,
        info@kdab.com
5    Author: éSrgio Martins <sergio.martins@kdab.com>
6
7    Copyright (C) 2015 Sergio Martins <smartins@kde.org>
8
9    Copyright (C) 2015 Boris Egorov <egorov@linux.com>
10
11   This library is free software; you can redistribute it and/or
12   modify it under the terms of the GNU Library General Public
13   License as published by the Free Software Foundation; either
14   version 2 of the License, or (at your option) any later version.
15
16   This library is distributed in the hope that it will be useful,
17   but WITHOUT ANY WARRANTY; without even the implied warranty of
18   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
19   Library General Public License for more details.
20
21   You should have received a copy of the GNU Library General Public License
22   along with this library; see the file COPYING.LIB.  If not, write to
23   the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor,
24   Boston, MA 02110-1301, USA.
25 */
26
27 #ifndef NON_REENTRANT_FUNCTIONS_H
28 #define NON_REENTRANT_FUNCTIONS_H
29
30 #include "checkbase.h"
31
32 class NonReentrantFunction : public CheckBase
33 {
34 public:
35     NonReentrantFunction(const std::string &name, const clang::CompilerInstance
        &ci);
```

```
36      void VisitStmt(clang::Stmt *) override;
37  private:
38      std::vector<std::string> nonReentrantFunctions;
39  };
40
41  #endif // NON_REENTRANT_FUNCTIONS_H
```

Листинг Б.5.24 — nonreentrantfunctions.cpp

```
1   /*
2      This file is part of the ph static checker.
3
4     Copyright (C) 2015 äKlarlvdalens Datakonsult AB, a KDAB Group company,
          info@kdab.com
5     Author: éSrgio Martins <sergio.martins@kdab.com>
6
7     Copyright (C) 2015 Sergio Martins <smartins@kde.org>
8
9     Copyright (C) 2015 Boris Egorov <egorov@linux.com>
10
11    This library is free software; you can redistribute it and/or
12    modify it under the terms of the GNU Library General Public
13    License as published by the Free Software Foundation; either
14    version 2 of the License, or (at your option) any later version.
15
16    This library is distributed in the hope that it will be useful,
17    but WITHOUT ANY WARRANTY; without even the implied warranty of
18    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
19    Library General Public License for more details.
20
21    You should have received a copy of the GNU Library General Public License
22    along with this library; see the file COPYING.LIB.  If not, write to
23    the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor,
24    Boston, MA 02110-1301, USA.
25  */
26  #include "nonreentrantfunctions.h"
27  #include "Utils.h"
28  #include "MacroUtils.h"
29  #include "StringUtils.h"
30  #include "checkmanager.h"
31
32  #include <clang/AST/Expr.h>
33  #include <clang/AST/Stmt.h>
34  #include <fstream>
35
36  using namespace clang;
37  using namespace std;
38
39  namespace {
```

```
40
41 const char *kConfigFilename = "/usr/local/etc/non_reentrant_functions.txt";
42
43 } // namespace
44
45 NonReentrantFunction::NonReentrantFunction(const std::string &name, const clang
       ::CompilerInstance &ci)
46     : CheckBase(name, ci)
47 {
48
49     // Default file was generated from
50     // http://pubs.opengroup.org/onlinepubs/9699919799/functions/V2_chap02.html
          #tag_15_09_01
51     auto in = std::ifstream{kConfigFilename};
52     if (!in) {
53         llvm::errs() << "Cannot read config file " << kConfigFilename
54                      << ", " << __func__ << " check wouldn't work\n";
55     } else {
56         for (std::string s; in >> s;) {
57             nonReentrantFunctions.push_back(s);
58         }
59     }
60 }
61
62 void NonReentrantFunction::VisitStmt(Stmt *stm)
63 {
64     const SourceLocation stmStart = stm->getLocStart();
65
66     bool warn = false;
67
68     if (CallExpr *call = dyn_cast<CallExpr>(stm)) {
69         // Non member function calls not allowed
70
71         FunctionDecl *func = call->getDirectCallee();
72         if (func) {
73             if (isa<CXXMethodDecl>(func))
74                 return;
75
76             const std::string funcName = func->getNameAsString();
77             if (clazy_std::contains(nonReentrantFunctions, funcName)) {
78                 warn = true;
79             } else {
80                 return;
81             }
82         }
83     }
84
85     if (warn) {
```

```
86            emitWarning(stmStart, "Non-reentrant function used");
87        }
88    }
89
90    REGISTER_CHECK_WITH_FLAGS("non-reentrant-function", NonReentrantFunction,
          CheckLevel0)
```