СОВРЕМЕННЫЕ ТЕХНОЛОГИИ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

УДК 004.415.2.052.03, 004.042, 004.046

ПОДХОД К ВЫДЕЛЕНИЮ МНОГОПОТОЧНОЙ СТРУКТУРЫ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

И. С. Свирин, П. А. Силин

ЗАО «Нордавинд»

Поступила в редакцию 20.04.2015 г.

Аннотация. В статье рассматривается архитектура средства извлечения многопоточной структуры программного обеспечения. На основе анализа задачи извлечения многопоточной структуры и ее места в рамках подхода к решению проблемы взаимных блокировок формируются общие требования к архитектуре средства. Подробно рассматривается двухуровневая архитектура средства.

Ключевые слова: многопоточность, взаимные блокировки, извлечение многопоточной структуры программного обеспечения, динамический анализ.

Annotation. In the article the architecture of multi-threaded structure extractor software tool is discussed. General requirements to extractor software tool based on the analysis of the multi-threaded structure extraction problem and its place in the approach to solving problem of deadlocks are introduced. Detailed analysis of the two-level architecture of extractor software tool is performed.

Keywords: multi-threading, deadlocks, multi-threaded structure extraction, dynamic analysis.

ВВЕДЕНИЕ

Одной из основных проблем разработки многопоточного программного обеспечения (ПО) является обеспечение доступа различных потоков к разделяемым ресурсам. Для решения данной проблемы современные системы и средства программирования предоставляют средства синхронизации. Однако использование средств синхронизации приводит к проблеме возникновения взаимных блокировок – ситуаций, когда группа потоков не может продолжать выполнение независимо от действий остальных потоков системы. Ошибки, связанные с взаимной блокировкой потоков, чрезвычайно трудно выявить, поскольку их проявление напрямую связано с относительной динамикой выполнения потоков в ПО, зависящей от множества факторов, которые могут проявиться, например, при переходе на новую платформу или добавлении новой подсистемы. Эта особенность делает принципиально невозможным создание алгоритма выявления взаимных блокировок на этапе тестирования ПО.

Существует несколько подходов к решению данной проблемы.

Динамический анализ, который основан на мониторинге выполняющейся программы на предмет обращения к ресурсам и осуществления различных вызовов [1]. Этот подход характеризуется низкими затратами вычислительных ресурсов, однако, обладает большим количеством ложных срабатываний и не гарантирует нахождения всех потенциальных ситуаций блокировки.

Статический анализ использует исходные коды или объектные файлы ПО для построения моделей, которые проверяются на наличие блокировок [2]. Этот подход является

достаточно эффективным, хотя порождает большое количество ложных срабатываний и плохо применим к ПО со сложной объектной структурой.

Верификация моделей по методу Model Checking основана на построении формальной модели ПО [3] с последующей верификацией данной модели с помощью специализированных средств [4]. Этот подход принципиально не дает ложных срабатываний и исключает возможность пропуска блокировок, но чрезвычайно требователен к вычислительным ресурсам.

Независимо от выбранного подхода, решение проблемы можно разделить на два логических шага: извлечение многопоточной структуры анализируемого ПО и анализ извлеченной структуры.

Данная статья описывает структуру программного средства, автоматизирующего первый шаг. Результаты извлечения могут быть проанализированы в рамках разработанной авторами математической модели взаимных блокировок [5].

В разделе 1 производится постановка задачи и на ее основе формируются требования к структуре средства, затем в разделе 2 приводится подробное описание структуры средства.

1. АНАЛИЗ ПРОБЛЕМЫ

Первым шагом формирования подхода к извлечению многопоточной структуры ПО является общий анализ проблемы: уточнение входных и выходных данных, выделение основных ограничения на решение задачи, формирование на основе полученной информации принципиального способа решения задачи. Затем, принципиальное решение уточняется в степени, достаточной для реализации на его основе средства извлечения многопоточной структуры (СИМС) ПО.

Уточнение входных данных для формирования подхода фактически сводится к выбору класса ПО, для которого можно будет применить СИМС. В качестве целевого класса выбрано многопоточное ПО, работающее под управлением ОС Linux, написанное на алгоритмическом языке C++, с использованием библиотеки управления потоками libpthread. К данному классу относится большое количество ПО, разработанного в промышленных или научных целях, с доступными исходными кодами и характеризуемого высокой вероятностью наличия ошибок синхронизации.

Выходные данные СИМС должны выражаться в терминах математической модели взаимных блокировок [5], представляющей многопоточную структуру ПО в виде некоторого конечного набора субъектов доступа – потоков, взаимодействующих с различными средствами синхронизации посредством выполнения их операторов. Субъект доступа моделируется на основе системы переходов, т. е. субъект отождествляется с совокупностью своих линейных цепочек выполнения с отождествленными начальным и конечным состояниями (состояние покоя). Субъект может включать в себя точки ветвления и точки зацикливания.

На рис. 1 изображено графическое представление субъекта доступа, «0» – обозначает состояние покоя, отождествленное с завершающим состоянием, «О1»–«О6» – операторы взаимодействия со средствами синхронизации, «Т1» – точка ветвления, «Т2» – точка зацикливания.

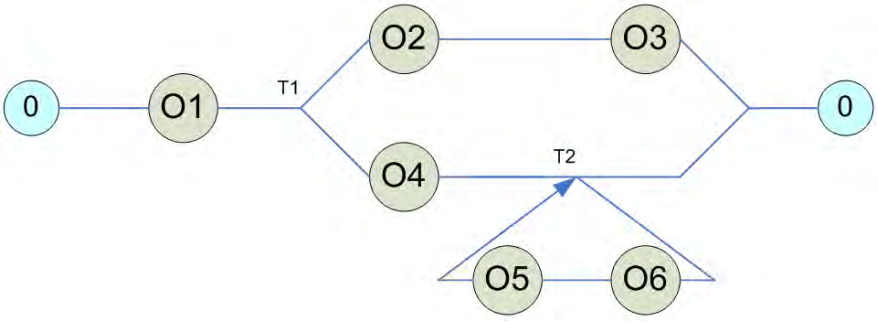

Рис. 1. Графическое изображение субъекта доступа

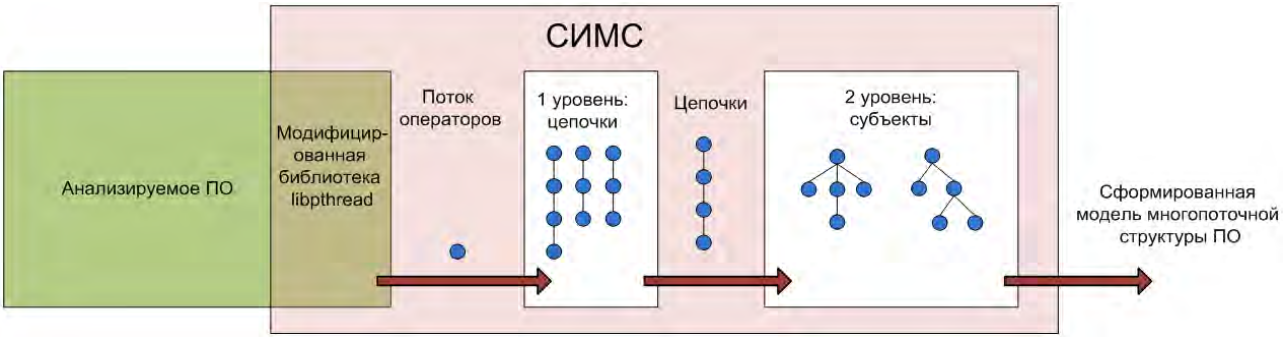

Рис. 2. Схема работы СИМС

После рассмотрения входных и выходных данных СИМС необходимо проанализировать ограничения, накладываемые на него в процессе работы. СИМС необходимо выделять многопоточную структуру из ПО промышленных размеров (~100 потоков, ~100 используемых средств синхронизации и не менее 10000 строк исходных кодов). Время выделения многопоточной структуры должно быть соизмеримо со временем выполнения группы тестовых сценариев анализируемого ПО для использования СИМС непосредственно в процессе разработки.

На основании описанных выше требований целесообразно реализовывать СИМС в рамках динамического подхода, поскольку затруднительно использовать статический анализ и Model Checking для промышленного ПО.

В рамках динамического анализа авторами была предложена схема функционирования СИМС на основе модернизации библиотеки libpthread путем добавления функции уведомления некоторого системного процесса о вызовах ее API. Уведомляемый системный процесс обрабатывает полученную информацию, формируя субъекты доступа.

2. УТОЧНЕНИЕ СТРУКТУРЫ ПРОГРАММНОГО СРЕДСТВА

СИМС в процессе работы получает на вход неструктурированный поток выполненных операторов, который необходимо организовать в субъекты доступа, таким образом, чтобы не потерять общность, поскольку потеря общности может повлечь потерю потенциальных ситуаций взаимной блокировки. Воссозданные субъекты доступа должны состоять из как можно меньшего числа операторов, поскольку входной поток операторов, генерируемый многопоточной структурой промышленных масштабов, может быть весьма интенсивным.

Процесс воссоздания субъектов доступа может быть разделен на два уровня:

1. Создание из потока операторов линейной цепочки выполнения субъекта;

2. Добавление созданной цепочки к субъекту доступа.

Общая схема работы СИМС, отражающая данную идею деления, изображена на рис. 2.

Рассмотрим подробнее первый уровень алгоритма. В его основе лежит механизм выделения цепочки выполнения субъекта. Будем считать, что если некоторая последовательность операторов переводит субъект в состояние покоя, то она является его цепочкой выполнения. Такой механизм не приводит к потере общности. В данном механизме следует обратить внимание на оператор захвата исключающего семафора (обозначается L), поскольку после его выполнения субъекту для перехода в состояние покоя необходимо, как минимум, выполнить оператор освобождения соответствующего исключающего семафора (обозначается U). Если захвачен один или несколько исключающих семафоров, формируемая цепочка выполнения может сколь угодно долго расти, пока не будут переданы соответствующие операторы освобождения.

Для минимизации роста формируемой цепочки в процессе ее формирования выделяются повторяющиеся последовательности операторов и заменяются циклами. Для этого с каждой формируемой цепочкой выполне-

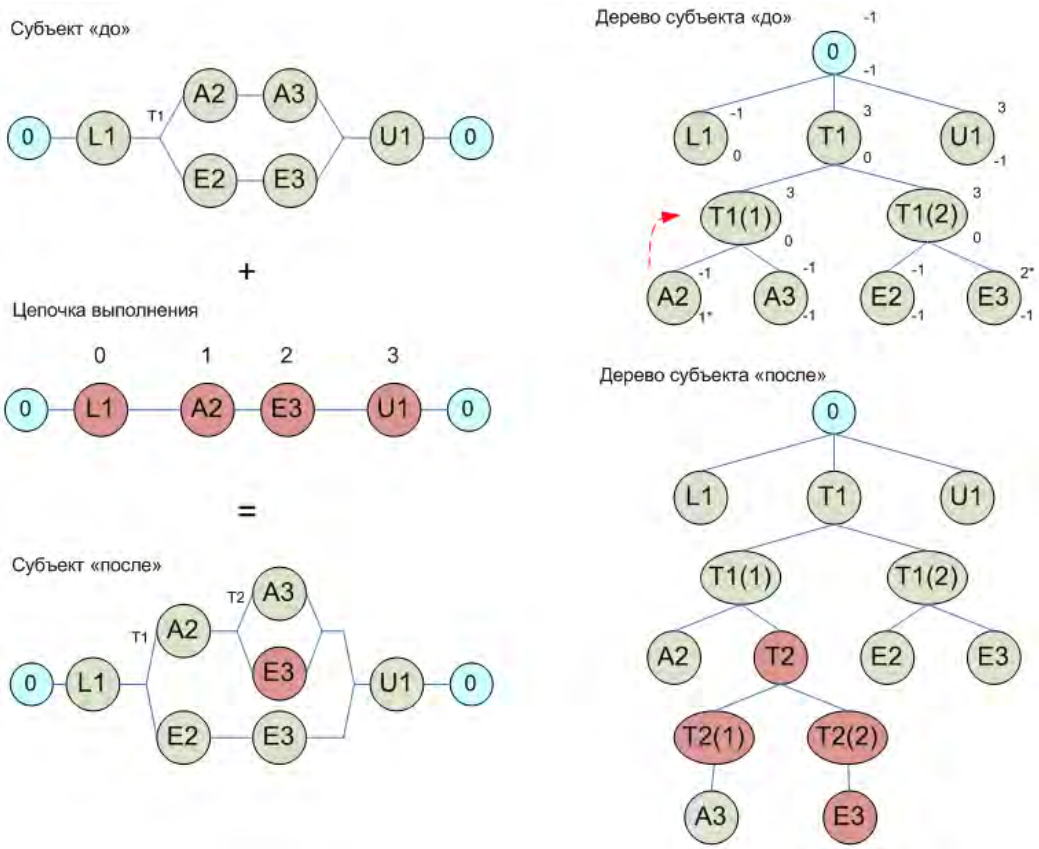

Рис. 3. Пример добавления цепочки выполнения к субъекту доступа

ния отождествляется конечный автомат, который может находиться в трех состояниях:

- начальное состояние (отсутствуют повторяющиеся последовательностей операторов);
- проверка последовательности (обнаружено некоторое число потенциальных повторяющихся последовательностей);
- активный цикл (обнаружена повторяющаяся последовательность).

Переход между состояниями осуществляется в зависимости от обнаружения и обработки повторяющейся последовательности.

После рассмотрения основных принципов работы первого уровня алгоритма перейдем к анализу второго уровня. На втором уровне алгоритма производится добавление созданной цепочки выполнения к субъекту доступа. Субъекты доступа удобно представлять в виде деревьев, с вершиной в начальном состоянии и листовыми вершинами, соответствующими операторам взаимодействия со средствами синхронизации. Дерево субъекта доступа включает в себя вершины следующих типов: листовая вершина, точка зацикливания, точка ветвления, ветвь точки ветвления и пустая.

Ключевой проблемой второго уровня алгоритма является сопоставление линейной структуры (цепочка выполнения) и древовидной структуры (субъект доступа). Эти две структуры необходимо сопоставить таким образом, чтобы выявить как можно больше общих операторов взаимодействия со средствами синхронизации.

Операция сопоставления субъекта и цепочки выполнения производится в несколько шагов. На первом шаге осуществляется прямая и обратная маркировка. Прямая маркировка – это рекурсивная операция одновременного обхода дерева субъекта и цепочки выполнения в прямом порядке с целью определения их максимальной общей части, если считать от начала цепочки выполнения. Пример добавления цепочки доступа к субъекту изображен на рис. 3. Результат прямой маркировки изображен в виде индексов, расположенных в правом нижнем углу каждой из

вершин дерева субъекта, неотрицательное число обозначает номер соответствующего данной вершине оператора цепочки выполнения. На рис. 3 индекс опорной вершины снабжен звездочкой (A2).

Обратная маркировка аналогична прямой маркировке, с той разницей, что обход цепочки выполнения и дерева субъекта доступа производится в обратном порядке. Результат обратной маркировки изображен на рис. 3 в виде индексов, расположенных в правом верхнем углу каждой из вершин дерева субъекта. Опорной вершиной обратной маркировки называется листовая вершина с минимальным неотрицательным индексом или корневая вершина, если листовых вершин с неотрицательным индексом нет. На рис. 3 индекс опорной вершины снабжен звездочкой (E3).

На следующем шаге после прямой и обратной маркировок осуществляется поиск общей опорной вершины. Выбирается опорная вершина прямой маркировки, если она маркирована прямым и обратным неотрицательными индексами, то она объявляется общей опорной вершиной. В противном случае осуществляется переход к ее непосредственной родительской вершине, для нее проверяются индексы аналогичным образом. На рис. 3 общей опорной вершиной является T1(1). Процесс поиска общей опорной вершины обозначен пунктирной стрелкой.

На следующем шаге производится наложение цепочки выполнения на дерево субъекта доступа. Непосредственные дочерние вершины общей опорной вершины разбиваются на три возможно пустых подмножества: маркированные вершины (прямая маркировка) (A2 на рис. 3), не маркированные (A3 на рис. 3), маркированные (обратная маркировка) (пустое множество на рис. 3). Наложение производится следующим образом, вместо немаркированного подмножества создается точка ветвления, на одну ее ветвь помещаются немаркированные дочерние вершины, на другую ее ветвь помещаются те операторы цепочки выполнения, которые не были маркированы на данной ветви дерева субъекта доступа.

После наложения производится обработка следующей поступившей с первого уровня цепочки выполнения. Обработка поступающих цепочек продолжается до тех пор, пока не получен сигнал об остановке работы. В этом случае прекращается прием цепочек выполнения, оставшиеся цепочки обрабатываются, затем полученные итоговые субъекты возвращаются клиенту СИМС в виде XML-файла.

ЗАКЛЮЧЕНИЕ

В данной статье описана подробная структура программного средства извлечения многопоточной структуры (СИМС) программного обеспечения. Данное средство является частью подхода, разработанного авторами для решения проблемы взаимных блокировок в многопоточном программном обеспечении.

Средство реализуется на основе динамического подхода с помощью модернизации библиотеки libpthread для средств работающих под управлением ОС Linux, реализованных на алгоритмическом языке C++, что позволяет использовать его в процессе промышленной разработки ПО.

Приводится подробное описание структуры СИМС, состоящей из двух алгоритмических уровней – формирования линейных цепочек выполнения и встраивания их в древовидную структуру субъектов доступа.

СПИСОК ЛИТЕРАТУРЫ

1. *Bensalem S., Havelund K.* Dynamic Deadlock Analysis of Multi-threaded Programs / S. Bensalem, K. Havelund // Haifa Verification Conference, Springer. – 2005. – Vol. 3875. – Pp. – 208–223.

2. *Artho C., Biere A.* Applying Static Analysis to Large-Scale, Multi-threaded Java Programs / C. Artho, A. Biere // 13th Australien Software Engineering Conference, August 2001. – IEEE Computer Society. – 2011. – Pp. 68–75.

3. *Карпов Ю.* Model Checking верификация параллельных и распределенных программ-

ных систем / Ю. Карпов. – СПб. : БХВ-Петербург. – 2010.

4. *Lamport L.* Specifying Systems: The TLA+ Language and Tools for Hardware and Software Engineers / L. Lamport. – Addison-Wesley. – 2002.

5. *Свирин И. С., Силин П. А., Сюзев В. В., Зарецкая Е. А.* Математическая модель взаимных блокировок. Корректное использование исключающих семафоров / И. С. Свирин, П. А. Силин, В. В. Сюзев, Е. А. Зарецкая // Вестник Московского Государственного Технического Университета имени Н. Э. Баумана, ISSN 0236 – 3933, специальный выпуск «Моделирование и идентификация компьютерных систем и сетей». – 2012. – С. 112–121.

Свирин И. С. – к. т. н., генеральный директор ЗАО «Нордавинд».
E-mail: i.svirin@nordavind.ru

Силин П. А. – инженер-математик «Topcon Positioning Systems».
E-mail: silinp@yandex.ru

Svirin Ilya – candidate of Science in Engineering, Nordavind CJSC, CEO.
E-mail: i.svirin@nordavind.ru

Silin Pavel – «Topcon Positioning Systems» software and Algorithm developer.
E-mail: silinp@yandex.ru