СБОРНИК **НАУЧНЫХ ТРУДОВ НГТУ. – 2014. – № 3(77).** – 119–124

УДК 62-50:519.216

ЭВРИСТИЧЕСКИЕ АЛГОРИТМЫ ПОИСКА ПРОГРАММНЫХ ОШИБОК*

Д.О. РОМАННИКОВ

630073, РФ, г. Новосибирск, пр. Карла Маркса, 20, Новосибирский государственный технический университет, кандидат технических наук, старший преподаватель кафедры автоматики. E-mail: rom2006@gmail.com

Цель данной работы – показать основные этапы поиска ошибок в программном обеспечении (ПО). Для этого в начале работы приведено описание того, что понимается под задачей поиска ошибок в ПО и из каких этапов состоит сам поиск. В следующей части статьи приводятся различные методы, которыми разработчики ПО пользуются на практике при поиске ошибок. Приводятся описания поиска ошибок в однопоточном и многопоточном ПО, ошибок, связанных с потреблением памяти, и др. Основной задачей для разработчика при поиске ошибки является ее локализация, т. е. определение участка кода, где она находится. Основным методом для этого является сопоставление логов программы и ее исходного кода таким образом, чтобы по данным из логов было понятно, по какому участку кода выполняется программа и с какими данными. При анализе многопоточного ПО основной задачей для поиска является определение участков кода, где данные считываются из общих ресурсов и выполняется их запись. В заключительной части работы приводится обобщение рассматриваемых подходов нахождения ошибок в ПО и факторов, которые влияют на этот процесс.

Ключевые слова: программное обеспечение, тестирование, входные интервалы, формальная верификация, верификация, модели программного обеспечения, графы, тотальная корректность программ

ВВЕДЕНИЕ

Поиск ошибок в программном обеспечении (ПО) является актуальной задачей, которой посвящены многие работы [1–10]. Среди наиболее известных подходов выделяются проверка моделей [1–4], статический анализ и абстрактная интерпретация [3–6] и другие формальные техники и подходы [7–10].

В данной работе выполнено обобщение эмпирических способов поиска программных ошибок, которые разработчики используют в каждодневной основе в своей работе. Приведено детальное описание того, как выполняется

* Статья получена 16 июля 2014 г.

Работа выполнена при финансовой поддержке Минобрнауки России по государственному заданию № 2014/138. Тема проекта «Новые структуры, модели и алгоритмы для прорывных методов управления техническими системами на основе наукоемких результатов интеллектуальной деятельности».

поиск ошибок на практике и из каких основных этапов такой поиск состоит. Анализируется поиск ошибок в однопоточном и многопоточном программном обеспечении, а также ошибки, связанные с выделением памяти.

Далее работа разделена на следующие разделы: в ОПИСАНИИ ЗАДАЧИ приводится описание процесса поиска ошибки и ее исправление; в разделе ЛОКАЛИЗАЦИЯ ОШИБОК приводятся эмпирические способы поиска программных ошибок в различных видах ПО. Работа завершается разделом ОБОБЩЕНИЕ ТЕХНИК ЛОКАЛИЗАЦИИ ОШИБОК, в котором приводится анализ вышеприведенных техник и выделение основных факторов, которые влияют на поиск.

1. ОПИСАНИЕ ЗАДАЧИ

Задачу исправления разработчиком ПО ошибки можно разделить на несколько этапов. 1. Понять, в чем заключается ошибка и какое поведение программы является правильным (или ожидаемым). Данное действие не всегда бывает очевидным, так как при постоянной работе с программным продуктом происходит «замыливание взгляда», что приводит к игнорированию некоторых ошибок по причине привыкания к ним, и т. д. Кроме того, из практики написания ПО известно, что для разных пользователей разрабатываемого продукта одни и те же действия должны приводить к различным результатам. Это в первую очередь связано с тем, что различные пользователи используют продукт в разных целях. В некоторых случаях первый этап исследования может быть пропущен, например, когда условия возникновения ошибки неизвестны (сервис в процессе работы по неизвестной причине останавливается). 2. Следующим шагом является процесс нахождения ошибки в коде ПО и определение, какие конкретно данные, последовательности действий, условия окружения приводят к ошибке. В отличие от первого шага, на данном этапе разработчик полностью посвящен работе с кодом программы. Результатом работы является определение строки или ограниченного участка кода, в котором допущена ошибка. 3. После локализации ошибки необходимо понять, как ее исправить, и чтобы это не привело к новым ошибкам.

2. ЛОКАЛИЗАЦИЯ ОШИБОК

Классическим способом, с академической точки зрения, является отладка ПО с помощью отладчика (дебаггера), который позволяет выполнить останов в нужном месте программы по нужным условиям, а затем по шагам отладить куски исходного кода, изменить состояние программы и даже вернуться на несколько шагов назад. К сожалению, такой способ часто невозможно применить по различным причинам: продуктовое ПО «собирается» с отключенными опциями отладки (дебага), заказчик против установки дополнительных модулей и т. д.

Для локализации ошибок на практике одним из наиболее распространенных способов является *сопоставление сообщений в логах программы с местом их вызова*. Такой способ наиболее распространен в силу того, что он гарантирован-

но позволяет локализовать ошибку, но время, потраченное на такой поиск, заранее оценить приблизительно можно только из опыта работы с кодом и интуитивной оценкой. В больших программных проектах на это может тратиться до нескольких дней из-за большого объема кода.

В условиях, когда логи отсутствуют из-за невозможности выполнить код программы или по другим причинам, исходный код программы исследуют вручную, т. е. *выполняется интерпретация программного кода в мозгах разработчика*. При этом выбирают участки кода, в которых вероятность возникновения исследуемой ошибки наиболее высока, а затем анализируют программный код, пытаясь понять, где и как произошла ошибка. В силу того, что по «симптомам» ошибки можно предположить ее причины, такие исследования часто завершаются положительно.

Если вышеприведенные способы не принесли желаемых результатов, но при этом удается локализовать ошибку в участке кода (размеры таких участков могут сильно различаться и достигать сотен тысяч строк кода с учетом сторонних библиотек), имеет смысл отлаживать программу по частям, для которых могут быть применены вышеизложенные приемы.

Отдельно следует сказать про поиск ошибок в многопоточном ПО. Ошибки в таком виде ПО являются наиболее сложными и легкодопустимыми, так как кроме того, что необходимо следить за логикой программы, также нужно следить за доступом к общим ресурсам (переменным, файлам, данным из базы данных и т. д.), что является совсем нетривиальным. Для поиска ошибок данного вида нет общепризнанных практик, и каждый случай разработчику приходится рассматривать индивидуально. Например, в программной системе содержится две утилиты, в которых происходит изменение одного ресурса. Тогда при одновременном запуске этих утилит возможна ситуация, когда утилиты считывают данные и выполняют их изменение в памяти, но конечные изменения ресурса зависят от порядка записи измененных данных в ресурсе. Обнаружение таких ошибок возможно или при предварительном тщательном планировании дизайна программной системы, или при использовании специализированных инструментов, позволяющих выполнить анализ ПО на ошибки в многопоточных системах. Чаще всего оба вышеприведенных способа невозможны на практике, и в результате разработчику приходится выявлять ошибки в ПО вручную. Основным средством для выявления ошибок в многопоточном ПО являются логи выполнения программы и ее код. На основании логов программист просматривает код на возможные сценарии выполнения программы. При этом основное внимание уделяется данным из общих для потоков ресурсов – файлам, записям их базы данных и т. д., которые изменяются в процессе выполнения программы. При нахождении ошибочных участков кода и исправлении их отсутствие других ошибок *не гарантируется* и проверяется только лишь практическим использованием программы.

Для ошибок, связанных с утечкой памяти, вышеприведенные методы подходят с одним отличием – в логи добавляется количество потребляемой памяти и ищется место, где память выделяется, но не освобождается.

В крайних случаях прибегают к практике повторения трудновыявляемых ошибок на специально подготовленном тестовом программном окружении, где можно выполнять различные эксперименты.

ЗАКЛЮЧЕНИЕ: ОБОБЩЕНИЕ ТЕХНИК ЛОКАЛИЗАЦИИ ОШИБОК

В данном разделе приводится попытка обобщить приемы локализации программных ошибок для обобщенной задачи, представленной в разделе ОПИСАНИЕ ЗАДАЧИ.

Все техники и способы, приведенные в разделе ЛОКАЛИЗАЦИЯ ОШИБОК, сводятся к определению участка кода, в котором присутствует ошибка, или определению значений переменных в заданном участке кода. Для локализации ошибок в ПО применяют различные вариации метода последовательного приближения к участку кода (отсечение частей кода на основании информации из логов, «выполнения участков кода в уме» или просмотра результатов в отладчике), где находится ошибка. Причем скорость такого приближения зависит от выбора начального участка кода, что в основном определяется опытом и знанием кода.

Важным фактором, от которого зависит скорость обнаружения ошибки, является начальное место поиска. Его определение зависит в основном от опыта разработчика и его знания кода программы. Также часто при отладке программы программистам помогают название функций, классов, переменных, которые дают понимание того, какого формата данные должны в них содержаться или что выполняет функция только исходя из названия, что в значительной мере ускоряет поиск.

СПИСОК ЛИТЕРАТУРЫ

1. *Кларк Э., Грамберг О., Пелед Д.* Верификация моделей программ. Model checking. – М.: МЦНМО, 2002. – 416 с.

2. *Карпов Ю.Г.* Model checking. Верификация параллельных и распределенных программных систем. – СПб.: БХВ-Петербург, 2010. – 560 с.

3. *Шелехов В.И.* Методы доказательства корректности программ с хорошей логикой [Электронный ресурс] // Международная конференция «Современные проблемы математики, информатики и биоинформатики», посвященная 100-летию со дня рождения члена-корреспондента АН СССР А.А. Ляпунова, 11–14 октября 2011 г.: доклады. – Новосибирск, 2011. – С. 1–21. – URL: http://conf.nsc.ru/files/conferences/Lyap-100/fulltext/74974/75473/Shelekhov_prlogic.pdf (дата обращения: 31.10.2014).

4. *Глухих М.И., Ицыксон В.М., Цесько В.А.* Использование зависимостей для повышения точности статического анализа программ // Моделирование и анализ информационных систем. – 2011. – Т. 18, № 4. – С. 68–79.

5. *Cousot P., Cousot R.* Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints // Conference

record of the fourth ACM symposium on principles of programming languages, Los Angeles, California, Jan. 17–19, 1977. – New York: ACM Press, 1977. – P. 238–252.

6. *Bush W.R., Pincus J.D., Sielaff D.J.* A static analyzer for finding dynamic programming errors // Software: practice and experience. – 2000. – Vol. 30, iss. 7. – P. 775–802. – doi: 10.1002/(SICI)1097-024X(200006)30:7<775::AID-SPE309>3.0.CO;2-H.

7. Coherent clusters in source code / S. Islam, J. Krinke, D. Binkley, M. Harman // Journal of systems and software. – 2014. – Vol. 88. – P. 1–24.

8. *Воевода А.А., Марков А.В., Романников Д.О.* Разработка программного обеспечения: проектирование с использованием UML диаграмм и сетей Петри на примере АСУ ТП водонапорной станции // Труды СПИИРАН. – 2014. – Вып. 3 (34). – С. 218–231.

9. *Falk H., Marwedel P.* Source code optimization techniques for data flow dominated embedded software. – New York: Springer Science: Business Media, 2004. – 226 p.

10. *Марков А.В., Романников Д.О.* Алгоритм трансляции диаграммы активности в сеть Петри // Доклады Академии наук высшей школы Российской Федерации. – 2014. – № 1 (22). – С. 104–112.

Романников Дмитрий Олегович – кандидат технических наук, старший преподаватель кафедры автоматики НГТУ. Основное направление научных исследований – формальная верификация, проверка моделей. Имеет 31 публи-кацию. E-mail: rom2006@gmail.com

Heuristic search algorithm of software errors*

D.O. Romannikov

Novosibirsk State Technical University, 20 Karl Marks Avenue, Novosibirsk, 630073, Russian Federation, candidate of Technical Sciences, senior lecturer at the department of automation. E-mail: rom2006@gmail.com

The purpose of this work is to show main stages of search errors in software. In the begging of the paper description of the task of software errors search is given and in what stages this search consists. In the next part of the paper different methods of software errors search that developers use in their practice is given. Description of search of different kind of errors is given: single thread and multithread software, memory management and etc. The main task for developers in errors search is localization – determine part of code with error. The main methods for it is a comparison of program logs and source code in such case that program path and data will be clear by the logs. In the multithread software main task of the errors search is a localization of parts of code where data is read from common resource and where data is written. In the last part of the paper generalization of described methods software errors search and factors that affect this search process.

Keywords: software, testing, input intervals, formal verification, verification, software models, graphs, total correctness of programs

* *Received 16 July 2014.*

REFERENCES

1. Clarke E.M., Grumberg O., Peled D. *Model checking*. Cambridge, London, MIT Press, 2001. (Russ. ed.: Klark E., Gramberg O., Peled D. *Verifikatsiya modelei programm: Model checking*. Moscow, MTsNMO Publ., 2002. 416 p.).

2. Karpov Yu.G. Model Checking. Verifikatsiya parallel'nykh i raspredelennykh programmnykh sistem [Verification of parallel and distributed software systems]. St. Petersburg, BKhV-Peterburg Publ., 2010. 560 p.

3. Shelehov V.I. [Rules of correctness proof for programs with simple logic]. *Mezhdunarodnaya konferentsiya "Sovremennye problemy matematiki, informatiki i bioinformatiki", posvyashchennaya 100-letiyu so dnya rozhdeniya chlena-korrespondenta AN SSSR A.A. Lyapunova, 11–14 oktyabrya 2011 g. Doklady* [International conference "Modern problems of mathematics, informatics and bioinformatics", devoted to the 100th anniversary of professor Alexei A. Lyapunov, Novosibirsk, Russia, 2011, October 11–14. Reports]. Novosibirsk, 2011, pp. 1–21. Available at: http://conf.nsc.ru/files/conferences/Lyap-100/fulltext/74974/75473/Shelekhov_prlogic.pdf (accessed 31.10.2014)

4. Glukhikh M.I., Itsykson V.M., Tses'ko V.A. Ispol'zovanie zavisimostei dlya povysheniya tochnosti staticheskogo analiza programm [The use of dependencies for Improving the precision of program static analysis]. *Modelirovanie i analiz informatsionnykh sistem – Modeling and analisys of information systems*, 2011, vol. 18, no. 4, pp. 68–79.

5. Cousot P., Cousot R. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. Conference record of the fourth ACM symposium on principles of programming languages, Los Angeles, California, Jan. 17–19, 1977. New York, ACM Press, 1977, pp. 238–252.

6. Bush W., Pincus J., Sielaff D. A static analyzer for finding dynamic programming errors. *Software: practice and experience*, 2000, vol. 30, iss. 7, pp. 775–802. doi: 10.1002/(SICI)1097-024X(200006)30:7<775::AID-SPE309>3.0.CO;2-H

7. Islam S., Krinke J., Binkley D., Harman M. Coherent clusters in source code. *Journal of systems and software*, 2014, vol. 88, pp. 1–24.

8. Voevoda A.A., Markov A.V. Romannikov D.O. Razrabotka programmnogo obespecheniia: proektirovanie s ispol'zova-niem UML diagramm i setei Petri na primere ASU TP vodonapornoi stantsii [Software development: software design using UML diagrams and PETRI nets for example automated process control system of pumping station]. *Trudy SPIIRAN – SPIIRAS proceedings*, 2014, iss. 3 (34), pp. 218–231.

9. Falk H., Marwedel P. Source code optimization techniques for data flow dominated embedded software, New York, Springer Science, Business Media, 2004. 226 p.

10. Markov A.V., Romannikov D.O. Algoritm translyatsii diagrammy aktivnosti v set' Petri [Algorithm of automatic conversion of the activity diagram into Petri-net structure formats]. *Doklady Akademii nauk vysshei shkoly Rossiiskoi Federatsii – Proceedings of the Russian higher school Academy of sciences*, 2014, no. 1 (22), pp. 104–112.