Методы и средства повышения эффективности отладки программного обеспечения

Р.А. Зеленов ИПИ РАН

Аннотация: В статье описаны методы и средства повышения эффективности отладки программного обеспечения, которые позволяют в автоматическом режиме определять местонахождение и причину возникновения ошибок.

Введение

В разработке программного обеспечения огромный процент времени занимает процесс отладки. И от времени его выполнения, а так же от его эффективности будет зависеть дальнейшая судьба создаваемого продукта. При этом большинство программистов применяет стандартные средства: пошаговая отладка, логирование/трассировка и пр. Во многом это не автоматизированный процесс, на результат которого влияет человеческий фактор [1].

Так как главной задачей является нахождение и определение причины появления ошибки, очевидно, что необходимо использовать средства, позволяющие программисту быстрее найти ошибку и определить, что ее вызвало. На данном этапе развития средств поддержки программирования в большинстве случаев применяются отладчики времени выполнения, которые позволяют видеть ход выполнения программы, в том числе и по шагам. Однако это не исключает того факта, что решение о месте и причине возникновения ошибки, которая не привела к аварийному завершению программы, приходится принимать программисту [2].

Статический и динамический анализ кода

К средствам повышения эффективности отладки будем относить не те инструменты, которые помогают показать состояние памяти, процессов и переменных в программе, а те, которые могут указать места и причины возникновения ошибок.

Статический анализ кода - анализ программного обеспечения без выполнения программы. Анализу может подвергаться как исходный текст программы, так и ее объектный код [3]. Большинство современных компиляторов имеют возможность определения как явно ошибочных мест в программе, так и кода, который будучи формально верным, может привести к ошибке. Примерами результата работы такого анализатора могут служить следующие сообщения: переменная не инициализирована, переменная будет потеряна, несовпадение типов при операции присвоения значения и пр. Как видим - все эти ошибки достаточно тривиальны и, если программист будет более сконцентрирован, то он не допустит их. Таким образом, компиляторы, как правило, помогают избежать появления легко обнаруживаемых и так же легко исправляемых ошибок.

Для определения более сложных ошибок создаются сторонние статические анализаторы, которые привязаны к конкретным языкам программирования. Для

оценки эффективности их работы анализу подвергаются не тестовые программы, а довольно распространенные продукты с огромным объемом исходного кода [4].

Принцип работы статического анализатора кода заключается в том, что строится синтаксическое дерево разбора и по нему ведется анализ всех возможных ветвлений. Один из плюсов данного подхода в том, что может бытъ проанализирован код, который выполняется крайне редко и обычными тестами ошибку в нем найти маловероятно. Так же могут быть найдены парные проблемы, находящиеся в разных частях кода, например несоответствия **в** конструкторе и деструкторе [5].

Подобные средства позволяют создать целую среду для автоматизации анализа: система интеграции с контролем версий, позволяющая отдельно проверять каждую внесенную часть кода и моментально локализовать время, место и автора ошибки, система рецензирования, позволяющая создать общую базу данных по ошибкам, которая исключает повторный анализ уже исправленных ошибок, так как положение ошибки характеризуется не просто именем файла и номером строки, а контекстом, и поэтому даже когда ошибка переместилась в другое место, то она не будет заявлена как новая.

Анализаторы очень эффективно могут использовать многоядерные системы для существенного ускорения своей работы.

Динамический анализ кода - это анализ программного обеспечения, осуществляемый в ходе выполнения программы. Как правило, требует подгрузки специальных библиотек и перекомпиляцию исходного кода [6].

Вследствие того, что при данном виде анализа покрывается только тот код, который задействован при подаче конкретных входных данных, может потребоваться проведение большого количества запусков с разными входными данными для более широкого покрытия исходного кода.

Определяются только те ошибки, которые возникли по факту выполнения и, соответственно, количество предупреждений будет гораздо меньше, чем при статическом анализе. В случае, когда приложение содержит малое количество ошибок - это будет плюсом, так как внимание разработчика не будет отвлекаться зачастую неверными прогнозами анализатора.

Ранее применение динамических анализаторов было затруднено тем фактом, что для их работы требовались большие вычислительные мощности. Однако с ростом сложности разрабатываемых приложений и с ростом производительности вычислительных систем - применение данного вида анализа является весьма востребованным.

Интеллектуальные автоматические отладчики

Интеллектуальные автоматические отладчики - отладочные программы, которые могут автоматически, на основе опыта предыдущих запусков, определять ошибки в программе и предлагать пути их исправления. В настоящее время на рынке не представлено широко используемых коммерческих интеллектуальных отладчиков, однако научные работы в этой области ведутся [7].

Какого-либо проверенного принципа их создания не существует. Однако можно в общих чертах описать основные компоненты подобных отладчиков.

Компонент сбора информации, может быть основан на отладчике времени выполнения, который ведет сбор и запись информации во время выполнения отлаживаемой программы. Хранение подобных отладочных сведений можно осуществлять в любом удобном для будущей обработки виде: база данных, лог-файл и т.п. Стоит отметить, что необходимо избегать избыточности подобной информации, так как последующая ее обработка может потребовать значительных вычислительных ресурсов. Так как анализируемых программ может быть очень много, то пользователь подобной системы должен иметь возможность настраивать по своему усмотрению уровень подробности ведения протокола работы программы.

Следующим является *компонент анализа собранной отладочной информации.* Он должен на основе состояния отлаживаемой программы определить факт появления ошибки, тип и точное место ее возникновения.

Кроме того, необходимо определять причины возникновения ошибочных ситуаций и указывать конкретно их месторасположение в исходном коде.

Данный компонент должен индивидуально настраиваться на работу с определенными типами программ и ошибок: ошибки работы с памятью, ошибки в параллельных программах и т.п.

Последним компонентом интеллектуального автоматического отладчика является *система классификации*, которая должна указать пользователю причину ошибки и предоставить рекомендации по ее устранению. Ее архитектура может быть представлена следующими подкомпонентами: база знаний, модуль приобретения знаний и объяснения ошибочной ситуации, память и интерфейс взаимодействия с пользователем.

Таким образом, интеллектуальный автоматический отладчик - это многокомпонентная система, позволяющая при должном уровне настройки на конкретный тип программ, автоматически выявлять в них ошибки и предлагать пути их устранения. Окончательное решение всегда должно оставаться за программистом.

Заключение

Для повышения эффективности отладки ПО необходимо автоматизировать наиболее затратные по времени этапы: выявление факта возникновения ошибки и обнаружение места и причины ее возникновения. Для проектов, написанных на общепринятых языках программирования, возможно подобрать инструменты, выполняющие статический или динамический анализ кода. Однако следует очень подробно изучать спецификации на подобные продукты, так как необходимо, чтобы они смогли подстроиться под специфику

отлаживаемой программы. Так же следует иметь ввиду, что абсолютно универсальных средств отладки быть не может, и ни одно из них не сможет найти и исправить 100% ошибок. Более того, если средство может автоматически исправлять ошибки в коде - оно может и внести новые. Всегда требуется участие человека. Но в случае, если отладчик сработал эффективно - это позволяет сократить трудозатраты на выявление большого количества ошибок и сконцентрироваться на других этапах разработки.

Применение интеллектуштьных отладчиков затруднено тем, что это. как правило, узкоспециализированные средства, которые должны быть разработаны и настроены на определенный тип программ. В дальнейшем, если такой отладчик создан - он может помочь разработчикам кардинально сократить время отладки приложения, за счет выдачи программисту полной информации об ошибке и о способе ее исправления.

Автор статьи считает, что будущее отладки - за интеграцией вышеперечисленных средств в широко применяемые компиляторы. Если это будет осуществлено разработчиками компиляторов, то эффективность данных средств отладки будет весьма высока, что позволит автоматизировать процесс отладки на подавляющем большинстве разрабатываемых по всему миру приложений.

Литература

1. *Зеленое Р.А., Стетенков Ю А., Волчек В.Н., Хилько Д.В., Шнейдер А.Ю., Прокофьев А.А,,* Система капсульного программирования и отладки // Системы и средства информатики. - М.: ИПИ РАН, 2010. - С. 24-30.
2. *Тэллес М., Ю. Хеш*, «Наука отладки» // Москва: КУДИЦ-ОБРАЗ, 2003.
3. *Ларус* Д., *Болл Т., Дас* М О безошибочных программах. URL: http://www.osp.m/os/20Q4/07/185Q03/ (дата обращения: 20.10.2011).
4. *Карпов А., Рыжков Е.* PVS-Studio - статический анализ кода на языке Си и Си-н*. URL: http://www.viva64.eom/ru/a/0077/ (дата обращения:20.10.2011).
5. Статический анализ кода. URL: http://easy~coding.blogspot.com/20Q9/Q2/blog-post 23.html (дата обращения: 20.10.2011).
6. *Грэм Б., Леру П., Лендри Т.* Использование статического и динамического анализа для повышения качества продукции и эффективности разработки. URL: http://www.swd.m/mdex.php3?pid=828 (дата обращения: 20.10.2011).
7. *Замятина Е.Б., Козлов А С.* Опыт разработки интеллектуального отладчика программ моделирования на MPI // Вестник пермского университета. - П.: ПГУ, 2007. - С.50-55.

Сведения об авторе:

Зеленов Роман Альбертович, год рождения: 1986. **Место обученмг** аспирантура ИПИ РАН. Место работы: ИПИ РАН, 22 отд., инже»ер-исследователь. Направления научных интересов: нетрадиционные **архитектуры** микропроцессоров; проектирование и построение систем программирования **и** отладки ПО для микропроцессоров, построение универсальных отладочных средств.