Денис Силаков

Cooperative Bug Isolation — привлекаем пользователей к поиску ошибок в программах

Локализация ошибок и поиск их первопричин является одной из основных задач при разработке и отладке программ. За десятилетия развития индустрии ИТ появилось немало инструментов, частично эту задачу автоматизирующих. Однако большинство этих инструментов начинает работать только непосредственно при возникновении ошибки - например, сохраняя образ памяти процесса, историю вызовов функций, приведших к проблеме, состояние окружения и тому подобное.

Безусловно, подобная информация для отладки необходима, но не всегда ее достаточно. Например, при анализе последовательности вызова функций, приведших к краху приложения, нередко встает вопрос - а какие действия пользователя привели к тому, что функции вызывались в такой последовательности и с такими аргументами? Ведь часто ошибки проявляются в каких-то специфических ситуациях, которые не были предусмотрены разработчиками, и в которых программа ведет себя не совсем ожидаемым образом.

Одной из попыток расширить спектр данных, автоматически собираемых инструментами формирования отчетов об ошибках, является проект Cooperative Bug Isolation - CBI (http://research.cs.wisc.edu/cbi/). CBI зародился как исследовательский проект в Висконсинском университете в Мадисоне (University of Wisconsin–Madison) и послужил основой для диссертации одного из основных авторов - Бенжамина Либлита (Benjamin Liblit). Диссертация была защищена еще в 2004 году, а в 2005 году была признана лучшей в области компьютерных наук организацией ACM (Association for Computing Machinery, Ассоциация вычислительной техники — авторитетная научная организация, присуждающая, помимо прочего, премию Тьюринга). После защиты работа не была заброшена (как это нередко случается не только у нас, но и за рубежом), и проект продолжает развиваться и сейчас.

Принципы работы

Основной идеей CBI является сравнение хода работы программы, приведшего к аварийной ситуации, с работой в штатном режиме. Для сравнения используются трассы потока управления программы. В трассы помещается информация о последовательности вызовов функций, сработавших условиях ветвления и о прочих аспектах, влияющих на процесс выполнения приложения. В принципе, схожие сведения можно добыть и из образа памяти процесса с помощью отладчика (при условии, что анализируемая программа собрана с отладочной информацией), однако CBI использует свой собственный формат трасс, с которым авторам удобнее оперировать.

Для сбора необходимых данных, необходимо внести соответствующие изменения непосредственно в код приложения. Модифицированная программа самостоятельно составляет трассу потока управления при каждом запуске. Когда программа завершается (неважно, аварийно или нет), отчет с трассой отправляется в централизованное хранилище на сайте CBI. В этом хранилище содержатся трассы, соответствующие корректной работе различных приложений; при получении трассы, приведшей к ошибке, эта трасса сравнивается с корректными. По задумке авторов, подобный анализ должен дать ключ к нахождению источника ошибки.

Процесс сравнения трассы, приведшей к ошибке, и обычных трасс — отнюдь не тривиальный и включает в себя как статистический анализ, так и элементы машинного обучения (нацеленные на обнаружение отклонений от типичного поведения). Собственно, этот анализ и представляет собой существенную часть диссертации по CBI; за подробностями отошлю к тексту диссертации господина Либлита, который можно скачать на сайте проекта. Здесь же отмечу, что анализ трасс производится авторами CBI, пользователям об этом заботиться не надо.

Установка

Если вы решили внести свой вклад в тестирование программ с помощью CBI, то вам необходимо установить в свою систему инструментированные версии программ и активно ими пользоваться.

Если у вас 32-битный дистрибутив Fedora, как и у авторов проекта, то вам повезло - вы можете скачать и установить подготовленный авторами CBI пакет cbi-package-config-16-11.i686.rpm, после чего в вашей системе будет подключен репозиторий проекта. В репозитории находятся как утилиты CBI, так и инструментированные приложения, которых на данный момент насчитывается девять. Имена у инструментированных приложений те же, что и у обычных - gimp, liferea и так далее. При установке приложений и при наличии альтернатив, установщик Yum будет отдавать предпочтение репозиторию CBI. Так что если вы попросите Yum установить gimp, то он выберет сборку gimp от CBI, а не ту, что находится в основном репозитории Fedora. В добавку к самому приложению, необходимо установить пакет с информацией для CBI, имеющий суффикс 'samplerinfo' (например, 'gimp-samplerinfo').

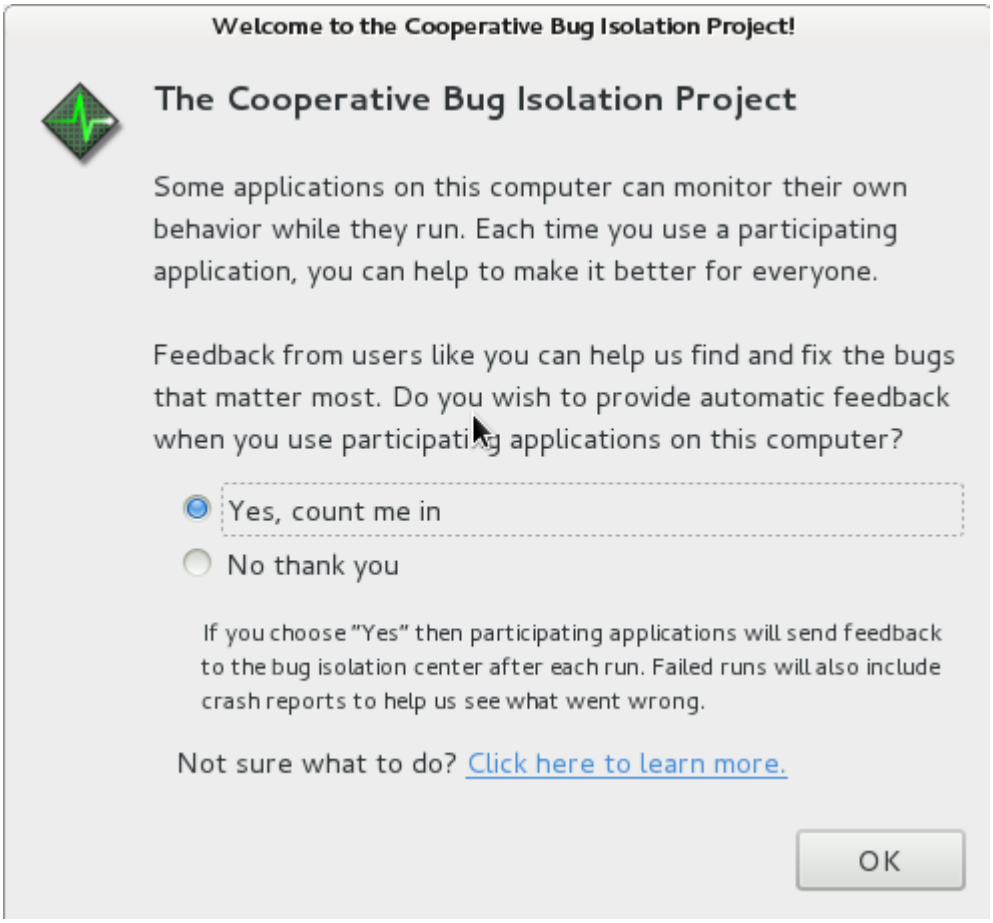
Если вы пользуетесь другим дистрибутивом, использующим RPM, то можно скачать готовые RPM-пакеты непосредственно с сайта - http://research.cs.wisc.edu/cbi/downloads/rpm/. Однако учтите, что зависимости пакетов могут оказаться специфичны для Fedora, так что с некоторой вероятностью придется их ставить с помощью 'rpm --nodeps' и самостоятельно заботится о том, чтобы в системе были все необходимые для их работы компоненты.

Запуск и работа

После установки инструментированного приложения, просто запустите его — никаких дополнительных опций или настроек не требуется. При первом запуске будет показан диалог, спрашивающий, согласны ли вы с тем, что программа будет автоматически отправлять отчеты в хранилище CBI. Если вы согласны, то выберете 'Yes, count me in'. Никакой регистрации не требуется, все данные отсылаются автоматически и никак не привязаны к конкретному пользователю или машине.

Процесс построения трассы не сильно ресурсоемкий. Каких-то ощутимых падений производительности при работе с инструментированными приложениями я не заметил (впрочем, каких-то специальных затратных действий я и не совершал — просто работал, как обычно).

Отчеты на сервер отправляются незаметно для пользователя; штатных способов просмотреть эти отчеты я не заметил. Пришлось воспользоваться wireshark, чтобы убедиться, что отчеты действительно отсылаются. Впрочем, посмотреть их содержимое с помощью wireshark все равно не получилось, поскольку общение с сервером происходит по защищенному протоколу HTTPS. Остается верить авторам, что ничего конфиденциального они не передают (либо самостоятельно изучить исходный код CBI).



Диалог-предупреждение об отсылке данных в хранилище CBI

А есть ли польза?

Итак, для пользователя принять участие в сборе трасс для CBI — задача очень даже несложная. Однако какова реальная польза от их участия? Действительно ли они помогают обнаруживать и исправлять ошибки?

В диссертационной работе Либлита описано применение CBI к анализу утилит moss, ccrypt, bc и exif, а также к медиапроигрывателю rhythmbox. В случае утилит проверялось, что инструментарий помогает в обнаружении и анализе некоторых уже известных проблем. В принципе, это традиционный подход для научных исследований, но все-таки гораздо больший интерес представляет обнаружение ранее неизвестных ошибок. Пример rhytmbox показывает, что CBI на это способен - автор обнаружил несколько проблем, которые разработчики признали достаточно серьезными.

Вообще в 2006 году база данных CBI содержала отчеты почти о 12.000 различных ошибок (http://technocrat.net/d/2006/6/6/4154/), но вот насколько CBI помог в их разрешении — не очень понятно. За прошедшие с момента защиты диссертации годы Бенжамин Либлит отметился несколькими десятками записей в багзилле Fedora; логично предположить, что без инструментария CBI в них не обошлось, хотя явных упоминаний об этом и не встречается.

CBI и сообщество

Целью проекта CBI декларируется привлечение широких масс пользователей к поиску ошибок в ПО. Однако нельзя сказать, что за прошедшие годы проект приобрел массовую популярность. И проблема даже не в том, что инструментированные программы запускают не очень много людей; самое главное препятствие к популяризации проекта — отсутствие разработчиков, которые бы инструментировали свои приложения, а затем анализировали

собираемые данные. При наличии подобных людей, вопрос увеличения охвата пользователей можно было бы решить достаточно просто. Например, разработчики дистрибутивов могли бы помещать в альфа- или бета-версии своих продуктов пакеты, инструментированные CBI. Ведь предварительные версии дистрибутивов подвергаются тщательному тестированию, работа входящих в них приложений проверяется многими пользователями по различным сценариям, что позволяет выявить существенную долю проблем. При этом никогда не помешает дополнительная автоматизация, которая помогла бы ускорить устранение ошибок — ведь это помогло приблизить срок выпуска финальной версии дистрибутива или хотя бы сократить число известных ошибок.

Однако сейчас единственный источник приложений, инструментированных с помощью CBI — это сайт самого проекта. Здесь предлагаются всего девять приложений, предназначенные для 32-битной версии Fedora. Прямо скажем, охват невелик — современные дистрибутивы содержат тысячи приложений и библиотек. Впрочем, вряд ли стоит ожидать большего охвата от самих авторов CBI. Ведь для полноценной работы с трассами необходимо хорошо ориентироваться в коде исследуемого приложения; да и не под силу небольшой группе людей анализировать трассы тысяч программ.

В общем, проекту CBI явно не помешало бы участие разработчиков приложений и дистрибутивов; в чем же загвоздка? Ведь код CBI открыт, так что можно самостоятельно собрать все нужные программы и даже развернуть собственное хранилище трасс. Процесс инструментирования программ хоть и далек от идеала и поддерживает только программы на языке C, но освоить его можно, и на достаточно большом количестве программ он работает вполне успешно.

Основная проблема связана с анализом трасс. Ведь трассы сами по себе не указывают на ошибку, они всего лишь могут дать ключи к решению проблемы, но ключами этими надо еще уметь воспользоваться. Помимо знакомства с кодом программы, анализ трасс требует хорошего понимания принципов работы CBI — иначе никакой особой пользы, кроме последовательности вызова функций, приведшей к ошибке, извлечь не получится. Но эту информацию можно получить и с помощью стандартных средств отладки. В то же время не очень понятно, насколько полезной окажется дополнительная информация, полученная от CBI. Конечно, некоторое количество ошибок авторы проекта обнаружили, но нельзя сказать, что они осуществили какой-то прорыв в области отладки.

Наконец, детальный анализ трасс — занятие кропотливое и занимает достаточно много времени. Наверное, проанализировав сотни трасс одной программы, можно приноровиться делать все гораздо быстрее; но поможет ли это при анализе трасс других программ — сложный вопрос. Согласитесь, не очень хочется тратить недели на изучение инструментария, потом днями сидеть и разбирать трассы, чтобы в итоге за год найти первопричину двух-трех ошибок. Вполне возможно, гораздо больше ошибок можно было бы выявить традиционными методами.

Так что сегодня CBI стоит рассматривать скорее как еще один вспомогательный инструмент, а не как основное средство отладки и поиска ошибок. Однако инструмент этот достаточно сложен в использовании; конечно, некоторая польза от CBI есть, и энтузиасты могут с ним поэкспериментировать, но к массовому использованию он не готов. И будет ли готов когда-нибудь — большой вопрос, поскольку никаких подвижек в этом направлении не наблюдается.