2010 Математические основы информатики и программирования №2(8)

DOI 10.17223/20710410/8/10

УДК 681.3

ОБ ОБНАРУЖЕНИИ ОШИБОЧНОЙ РАБОТЫ С РЕСУРСАМИ В ПРОГРАММНОМ ОБЕСПЕЧЕНИИ[1]

В. В. Горелов

Томский государственный университет, г. Томск, Россия

E-mail: skylark@mail.tsu.ru

Предлагается метод обнаружения и при необходимости предотвращения ошибочной работы программы с ресурсами. Количество ресурсов, с которыми может работать система, теоретически не ограничено ввиду применения разработанной схемы по описанию широкого спектра ресурсов и функций по работе с ними. Анализируются операции, влияющие на надёжность, безопасность и оптимальность использования ресурсов в программном обеспечении.

Ключевые слова: *программное обеспечение, инструментирование, ресурсы, обнаружение ошибок.*

Введение

Как известно, подавляющее большинство программного обеспечения состоит из алгоритмов преобразования информации и механизмов использования разнообразных ресурсов для получения, обработки и передачи информации. Со стороны программного обеспечения в процессе использования ресурсов (и, как показывает практика, довольно часто) возникают ситуации опасных действий с ними. Было замечено, что для ряда ресурсов встречаются одни и те же шаблонные ситуации ошибочного использования. Предлагается метод автоматизированного обнаружения таких ситуаций; с его помощью создана экспериментальная система по выявлению опасных участков в программе. Результатом работы системы является информация о конкретных местах и типах найденных дефектов в программе. Принципиальной особенностью обнаружения является отделение самой системы от типов ресурсов, с которыми она работает. Такой подход необходим для распознавания шаблонных ошибок для теоретически неограниченного количества ресурсов без внесения изменений в текст программы либо с незначительными изменениями для некоторых нестандартных классов ресурсов. Таким образом, предложенная система состоит из трёх частей:

1) языка для описания ресурсов и функций по работе с ними;
2) программы-преобразователя, которая на вход принимает описание ресурсов и создаёт необходимый код по перехвату функций, работающих с интересующими ресурсами;
3) программы-анализатора, которая считывает соответствующие события, возникающие в процессе работы отлаживаемой программы, с дальнейшим выводом обнаруженных опасных участков для анализа человеком.

Типы обнаруживаемых ошибок

К основным опасным ситуациям при использовании ресурсов отнесены следующие состояния: утечка ресурсов; использование ресурсов после их освобождения; повторные освобождения ресурсов; использование (неинициализорованных) ресурсов без их

[1]Работа выполнена в рамках реализации ФЦП «Научные и научно-педагогические кадры инновационной России» на 2009–2013 годы. (Гос. контракт № П1010.)

предварительного захвата; использование ресурсов за их границами (относится к динамической памяти и адресному пространству) и другие. Некоторые из перечисленных ситуаций особо опасны, поскольку могут привести к исполнению произвольного кода, предварительно изготовленного злоумышленником, в адресном пространстве отлаживаемой программы.

Область применения и особенности реализации

Рассматриваемый метод реализован для языка программирования Си, однако при необходимости расширения системы возможна поддержка и других языков программирования. В настоящее время реализация [1] системы работает с прикладным программным обеспечением в сетевом и отсоединённом режимах. Под сетевым режимом подразумевается ситуация, когда отлаживаемая программа и программа-анализатор поступающих событий обмениваются сообщениями через TCP/IP-сеть. В отсоединённом режиме отлаживаемая программа автоматически протоколирует возникающие события в файл, который впоследствии передаётся на вход к программе-анализатору для создания отчёта, пригодного для чтения человеком. Система поддерживает 16-, 32- и 64-битные режимы адресации отлаживаемых POSIX- и Windows-программ. Программа-преобразователь и программа-анализатор работают в средах POSIX и Windows в 32- и 64-битных режимах. Сборка системы проверена с компиляторами GCC [2] и Microsoft Visual Studio [3]. Поскольку система обнаружения кроссплатформенная, вероятно, и другие компиляторы могут быть использованы. Единственной используемой внешней библиотекой является `libxml++` [4].

1. Структурное представление ресурсов

Для обеспечения раздельного описания ресурсов от системы обнаружения предусмотрен специальный язык, с помощью которого независимый разработчик может в довольно простом текстовом виде описать ресурсы, которые необходимо отследить. Формат описания ресурсов представляется в специально-оформленном XML-файле [5] с заполнением соответствующих полей и атрибутов.

1.1. Настройки режима работы

Данные настройки указываются один раз в начале файла-шаблона, который описывает ресурсы и функции по работе с ними. Далее перечислены атрибуты с описанием их значения.

— `format`: порядковый номер формата шаблона. Необходим для того, чтобы программа-преобразователь могла проверить, является ли переданный ей для обработки файл именно такого формата, который она понимает.
— `name`: название шаблона, отражающее его содержимое и предназначение; данная информация впоследствии помещается в создаваемые файлы с целью определения (при необходимости), по какому шаблону они были созданы.
— `prefix`: задаёт префикс для замещаемых функций. Требуется для исключения возможных столкновений между замещаемыми функциями с префиксом и существующими функциями с такими же именами. Например, для префикса «`art_`» и функции захвата памяти `malloc()` препроцессором будет добавлено определение функции `art_malloc()`, однако в пространстве имён отлаживаемого проекта может присутствовать функция с таким же именем. Для разрешения данной ситуации и предусмотрена возможность задать уникальный префикс.
— `errlogmode`: указывает, как реагировать служебному коду проекта в случае обнаружения невозможности протоколировать возникающие события. Имеет два за-

резервированных значения: «» — полностью игнорировать, «`console`» — выводить информацию через устройство вывода. Любое другое значение будет использовано как имя файла, в который будет произведена попытка записи информации о сбое протоколирования.

— `trap_on_io_err`: принимает значение истина или ложь. Влияет на поведение созданного кода при выявлении невозможности протоколирования. В первом случае приводит к записи информации об этом тем способом, который задан пунктом `errlogmode`. В противном случае ошибка игнорируется и осуществляется попытка продолжить работу.
— `remote_mode`: принимает значение истина или ложь. В первом случае даёт указание создавать код для работы в сетевом режиме. В противном случае создаётся код, записывающий события в файл протокола.
— `force_flush`: принимает значение истина или ложь. В первом случае даёт указание сбрасывать буфера при каждой операции записи или пересылки событий. Вне процесса отладки должно быть всегда установлено в значение истина. В противном случае будет происходить буферизирование передаваемой информации и в случае ошибки при её передаче часть информации может быть потеряна. Из-за этого программа-анализатор будет лишена важной информации, которая, вероятно, и привела к невозможности передать очередную порцию данных (например, код нарушил работу ядра операционной системы).
— `multithreaded`: принимает значение истина или ложь. В первом случае даёт указание создавать обёрточный код для блокирования одновременного вызова функций, за которыми ведётся наблюдение. Данное значение необходимо для многопоточных программ. Для программ, работающих в синхронном режиме, это значение должно быть задано как ложь.
— `threading`: указывает, какой создавать код для случая, когда отлаживается многонитевая программа. Поддерживаемые значение: `posix` и `win32` — для соответствующих сред.
— `compiler_type`: указывает, примитивы какого компилятора использовать. Принимает значения `gcc` и `win32` для соответствующих сред.
— `trace_target`: строка, которая для сетевого режима содержит адрес и порт, на который будут отсылаться события. Для отсоединённого режима данное поле означает имя файла, в который события будут записаны.

На этом настройки режима работы заканчиваются. Далее рассматривается непосредственно описание ресурсов и функций, которые с ними работают.

1.2. Шаблон ресурса

В файле-шаблоне для описания каждого отдельного ресурса применяется элемент `domain` с иерархическим включением других элементов. Элементы могут быть простыми — строки и двоичные значения — и составными, содержащими, в свою очередь, другие элементы. Таким образом, в шаблоне представлено дерево, описывающее все интересующие ресурсы, функции и прочую необходимую для создания кода информацию. Далее перечислены атрибуты элемента `domain`:

— `name`: человекопригодное имя для ресурса; например, для памяти, захватываемой из кучи, можно присвоить значение `heap memory`.
— `float_handle`: принимает значение истина или ложь. На этом атрибуте стоит остановиться подробнее. Все ресурсы с точки зрения системы по этому атрибуту делятся на два класса: так называемые «плавающие ресурсы» и «цельные». Под пер-

выми понимается класс ресурсов, к которым можно обращаться как к некой размерности. Это память и адресное пространство. Под вторым — ресурсы, которые не обладают данным свойством. В качестве примера можно привести дескриптор типа `int`, файловый дескриптор `FILE*` и т. д. Такое разделение введено для поддержки отслеживания границ адресных пространств и оценки эффективности их использования. Под последним имеется в виду, насколько захваченный ресурс был использован, как именно и был ли использован вообще.

— `handle`: тип ресурса для целевого языка программирования. Зависит от ресурса и может быть произвольным, например `void*`, `int`, `FILE*`, `some_type_t` и т. д.

— `includesGlobal`: принимает значение истина или ложь. Указывает, как именно препроцессор должен включать заголовочные файлы с определением ресурсов и функций. В первом случае используется для ресурсов, доступных в стандартных каталогах поиска заголовочных файлов (например, в среде `UNIX` это `/usr/include`). В противном случае включаются локальные заголовочные файлы. Последний вариант необходим при слежении за ресурсами, которые описаны и используются в самой отлаживаемой программе и не входят в стандартные библиотеки и API-системы.

— `includes`: представляет векторный тип с перечислением самих заголовочных файлов. Для ресурса типа «память из кучи» это будет вектор с одном элементом `stdlib.h`. Для своего собственного ресурса это может быть `myResource.h`.

— `bad_handle`: элемент, описывающий выражение, представляющее неверный ресурс. Необходим при создании кода, распознающего правильность ресурса. Например, для указателя это NULL, для дескриптора — -1. Однако самого значения недостаточно для проверки, является ли указатель «правильным». Далее отдельно рассмотрим атрибуты элемента `expr`.

1.3. Представление выражений

Элемент выражения `expr` служит для определения истинности некоторого значения с применением заданного критерия сравнения и состоит из следующих простых атрибутов:

— `oper`: строка, отражающая саму операцию. Например, `==`, `!=`, `<`.

— `type`: описывает имя типа, используемого в сравнении. Например, `int`, `ssize_t`, `gint`.

— `value`: произвольное значение пользователя для операции сравнения: 0, `NULL`, -1, `ERR_OK` и т. п.

Таким образом, для произвольного ресурса возможно следующее представление:
`type` t = «некоторое значение»;
`if`(t `oper value`) {код обработки истинности} `else` {код обработки неистинности}.

1.4. Представление операторов

Рассматриваются операторы по захвату, перераспределению, использованию и освобождению ресурса. Для их представления задаются элементы `tor`, каждый из которых имеет следующие атрибуты:

— `type`: принимает значение в зависимости от типа оператора (`allocator`, `reallocator`, `operator`, `deallocator`).

— `name`: имя оператора в рамках языка программирования. Например, `malloc`, `fopen`, `myAllocation`.

— `float_arg_n1`: неотрицательная величина, указывающая номер первого аргумента, отвечающего за задание размерности или смещения. Актуально только для «плавающих ресурсов».
— `float_arg_n2`: неотрицательная величина, указывающая номер второго (если есть) аргумента, отвечающего за задание размерности или смещения. Актуально только для «плавающих ресурсов».
— `validateDom_before`: векторная величина, указывающая, какие аргументы проверять по критерию правильности соответствующего ресурса. В качестве значений используется заданное в начале описания `domain` имя ресурса, например `heap memory`.
— `is_handle_arg_out`: принимает значение истина, если у функции существует возвращаемый идентификатор ресурса (для функций захвата и перераспределения ресурсов), или ложь — в этом случае следующий атрибут игнорируется.
— `handle_arg_out`: неотрицательное значение, указывающее номер выходного идентификатора ресурса: 0 — возвращаемый аргумент, 1 — первый аргумент функции и т. д.
— `args`: набор элементов, который служит для описания возвращаемого аргумента функции и аргументов самой функции. Каждый элемент `args` содержит тип переменной и её имя. Для возвращаемого значения имя не указывается.
— `badRetCode`: описывает выражение для распознавания возвращаемого значения функции, если оно есть. Распознаются значения «удачно» и «отказ» при проведении всех операций. Введение данного выражения необходимо в силу того, что формат возвращаемого статуса операции может быть представлен как через специально зарезервированное значение идентификатора ресурса, так и через раздельное использование ресурса и возвращаемого значения.

2. Преобразование описания ресурсов в код для инструментирования

Целью программы является преобразование описания ресурсов из файла-шаблона в исходный код на языке Си. На вход программе подаётся имя файла-шаблона и имена двух файлов, в которые должен быть помещён создаваемый код на Си. Поскольку файл-шаблон выражен в форме XML, то для его разбора используется библиотека `libxml++` (которая, в свою очередь, — удобная обвязка на языке Си++ для библиотеки `libxml2` [6], реализованной на языке программирования Си). Поскольку программе не требуется изменять содержимое шаблона, использован событийный разборщик (SAX parser). Для связи библиотечного движка разбора с самой программой преобразования реализованы классы, получающие события от библиотеки. Принятые события распознаются как элементы, атрибуты проверяются на логическое соответствие внутренней схеме представления. В процессе обхода файла-шаблона происходит проверка и формирование узлов шаблона, но уже в рамках языка программирования Си++, с применением его родных типов. Второй аргумент — создаваемый заголовочный файл, в который помещаются директивы препроцессора и определения замещаемых функций. В последнем аргументе — имя файла, куда помещаются тела реализации этих функций. Вызов преобразователя может быть осуществлён, например, так:

```
artgen posix-gcc-mt-file-lint.xml art.h art.c
```

В случае удачного обхода дерева описания из файла-шаблона управление передаётся части непосредственного создания кода, который будет использован для инструментирования отлаживаемой программы.

Последующее применение созданых файлов для отлаживаемой программы `main.c` на примере компилятора `GCC`:
`gcc -c art.c` (создаёт объектный файл `art.o`)
`gcc -include art.h main.c art.o -o main` (создаёт двоичный файл `main`).

2.1. Фаза создания заголовочного кода

Первое, что добавляется в файл — стандартная защита от повторного включения, а также указание препроцессору компилятора Си++, что все дальнейшие строки текста являются определениями на языке Си. Это позволяет использовать систему не только в чистых проектах на Си, но и в чистых Си++ или смешанных проектах на Си/Си++. Далее поведение кодогенератора практически полностью управляется значениями, считанными из шаблона, и, в зависимости от значений последнего, происходит обход дерева и комбинирование заготовок кода и логических конструкций из библиотеки кодогенератора.

Далее происходит подключение заголовочных файлов для поддержки многонитевости, если так указано в шаблоне. После этого подключаются заголовочные файлы отлаживаемых ресурсов. За ними следуют служебные определения функций системы. Основные из них следующие:

— `void art_trace(int n, ...)`: при вызове протоколирует n-е количество аргументов соответствующим образом (в файл или через сеть);
— `void art_start(char *appname)`: в случае необходимости (например, если задана отладка многонитевой программы) производит инициализацию служебных объектов синхронизации (`mutex`) и других переменных. Эта функция должна быть добавлена первой функцией в отлаживаемую программу;
— `void art_stop()`: функция автоматически вызывается последней при неаварийном завершении работы программы. Вызов этой функции регистрируется в функции `art_start()` через механизм `atexit()`.

Определение функций, которые будут вызываться в процессе работы программы, происходит следующим образом:

```
префикс-оригинальное название функции(оригинальные аргументы, char* file,
size_t line);
```

Последние два аргумента преследуют цель запротоколировать информацию о месте вызова в отлаживаемой программе.

Следующим блоком идут определения для препроцессора, который и проделает работу по замене вызовов оригинальных функций на «перехваченные»:

```
#define origName(zzz) префикс-системы_origName(zzz, __FILE__, __LINE__)
```

Последние два аргумента также являются ключевыми словами для препроцессора, в которые он подставит имя файла и номер строки, где произведена замена вызова оригинальной функции на функцию системы отладки.

2.2. Фаза создания тел реализации для инструментируемых функций

Первым создаётся код для служебных функций системы отладки, они относительно немногочисленны и носят характер поддержки сети, файловых дескрипторов, объектов синхронизации, функций инициализации, преобразования чисел в строки и т. д. Стоит остановиться лишь на паре ключевых служебных функций:

— `art_start()`: кроме прочих служебных функций, протоколирует описания ресурсов и функций по работе с ними в заголовке протокола. Эта информация необходима в фазе анализа;

— `art_stop()`: протоколирует информацию о том, что программа завершилась. Без этой информации на стороне сервера весьма затруднительно понять, пора делать выводы об утечке ресурсов или нет.

Работа по созданию инструментируемых функций разделена на четыре этапа.

1) *Пролог.* Заключается в создании начальной части кода с заданием и инициализацией вспомогательных переменных в соответствии с указаниями из шаблона. Создаваемый флаг (`mutex`) необходим для синхронизации протоколирования информации, поступающей от разных нитей. Применение его обязательно, поскольку запись в файл или гнездо должна быть произведена в определённом порядке, с сохранением очерёдности содержимого транзакций. Кроме того, протоколируется информация о типе ресурса и обрабатываемой функции с указанием имени файла и номера строки, откуда произведён вызов.
2) *Проверка.* По информации из шаблона создаётся код, проверяющий правильность переданных аргументов. Результат этой проверки протоколируется.
3) *Вызов оригинальной функции.* В соответствии с информацией из шаблона для произвольной функции происходит запись интересующих выходных данных.
4) *Эпилог.* Создаётся код для проверки правильности значения выходного ресурса и возвращаемых значений об успешности проведения операции. Полученная информация протоколируется, снимается блок с объекта синхронизации, происходит возврат из функции с передачей управления отлаживаемой программе.

На этом работа программы-преобразователя завершается.

3. Устройство программы-анализатора

Обработка протокола производится двумя методами — через чтение файла либо по сети. Протокол устроен таким образом, что для анализа нет принципиальной разницы, откуда поступают данные. Единственное различие заключается том, что в сетевом режиме можно влиять на разрешение выполнения произвольных действий на стороне отлаживаемой программы. Структурно программа анализа разделена на три части.

3.1. Воссоздание информации о ресурсах и функциях

Путём разбора заголовочной части происходит воссоздание необходимых для анализа элементов шаблона с информацией о ресурсах и функциях, с которыми они работают. Поскольку полностью начальный файл-шаблон в формате XML уже не нужен, в простом текстовом виде передаётся только необходимая информация о ресурсах, что сокращает размер заголовочной части и ускоряет начало непосредственной обработки информации. Также отпадает необходимость разбора XML.

3.2. Цикл обработки транзакций

После завершения работы по заполнению легковесной версии шаблона запускается основной цикл, в рамках которого анализируются одна за другой транзакции. Под транзакцией понимается вызов функции с тремя обязательными полями: пролог, проверка, эпилог.

В процессе работы цикла для каждого из ресурсов создаётся база данных (БД) в памяти со всеми необходимыми полями. На каждом шаге проверяется, насколько тот или иной вызов верен. В результате анализа самого вызова и, при необходимости, анализа БД выявляются следующие типы ошибок: «многократное освобождение ресурса», «ситуация, когда функция захвата возвращает уже захваченный ресурс», «использование незахваченного ресурса», «использование ресурса после освобождения», «освобождение неиспользованного ресурса», «использование ошибочного ресурса» (например,

`NULL` для памяти, -1 для дескрипторов, ...), «освобождение ошибочного ресурса», «использование ресурса за его границами» (например, памяти, отображений, ...). Кроме того, происходит анализ «КПД» использования ресурса, т. е. того, насколько ресурс используется и используется ли вообще. В случае низкого КПД это позволяет сократить требование к ресурсам или полностью исключить использование «паразитного» ресурса в определённом месте программы. Для всех вышеперечисленных ошибок система сообщает полную информацию о местах в программе, в которых они были замечены. Например, для ситуации повторного освобождения ресурса будет выдан примерно такой отчёт: "`deallocation of handle 0x12345678 by heap memory:free at bar.c:16 while the handle allocated at baz.c:10 and freed at foo.c:12`". Список обнаруживаемых ошибок в будущем может быть расширен.

3.3. Обработка оставшихся записей в БД после завершения анализируемой программы

На данном этапе все неосвобождённые ресурсы, информация о которых находится в БД, помечаются как утечки, и полная информация предоставляется разработчику для анализа. Формируется окончательная статистика о степени полноты использования ресурсов. Работа по анализу поступивших данных на этом заканчивается.

4. Планы на будущее

Планируется расширить сферу применения системы анализа на ядра других операционных систем (QNX, Solaris, *BSD, Linux и др.). Добиться этого поможет незначительное изменение какого-либо эмулятора с открытым исходным кодом [7], например QEMU [8] или VirtualBox [9]. Единственной необходимой модификацией должна быть поддержка обработки некоторой фиктивной машинной инструкции, с помощью которой будет возможен вывод протокольной информации из ядра в область среды исполнения эмулятора. Этот вывод легко затем перенаправить в файл или гнездо для передачи по сети. Также небезынтересно обнаружение ситуаций, когда вызов какой-либо функции не завершается. Кроме этого, возможно искусственное создание ситуации «нехватки ресурсов» для отлаживаемой программы. В процессе этого могут быть дополнительно выявлены ошибки, вызванные недостаточной проверкой кода завершения операции со стороны отлаживаемой программы. Дополнительно может быть добавлена функциональность, которая позволит собирать и временну́ю информацию, выявляя «узкие места» в работе функций.

ЛИТЕРАТУРА

1. `http://skylark.tsu.ru/art/` — Домашняя страница проекта A Resource Tracer. 2009.
2. *Von Hagen W.* The Definitive Guide to GCC. Second Ed. N.Y.: APRESS, 2006. 584 p.
3. *Powers L., Snell M.* Microsoft Visual Studio 2008 Unleashed. USA: Sams, 2008. 1248 p.
4. `http://libxmlplusplus.sourceforge.net/` — Домашняя страница библиотеки libxml++. 2010.
5. *Elliotte R. H., Means W. S.* XML in a Nutshell. Third Ed. USA: O'Reilly Media, 2004. 600 c.
6. `http://xmlsoft.org/` — Домашняя страница библиотеки libxml2. 2010.
7. `http://www.opensource.org/` — Сайт «Open Source Initiative». 2010.
8. `http://www.qemu.org/` — Домашняя страница проекта QEMU. 2010.
9. `http://www.virtualbox.org/` — Домашняя страница проекта VirtualBox. 2010.