Для цитирования: Русяев Р. М., Нейман-заде М. И., Ермолицкий А. В., Волконский В. Ю. Программно-аппаратные средства выявления ошибок обращения к памяти для архитектуры «Эльбрус» // Вопросы радиоэлектроники. 2017. № 3. С. 33–38.
УДК 519.688

Р. М. Русяев[1,2], М. И. Нейман-заде[1,2], А. В. Ермолицкий[1], В. Ю. Волконский[2]

[1] АО «МЦСТ», [2] ПАО «ИНЭУМ им. И. С. Брука»

ПРОГРАММНО-АППАРАТНЫЕ СРЕДСТВА ВЫЯВЛЕНИЯ ОШИБОК ОБРАЩЕНИЯ К ПАМЯТИ ДЛЯ АРХИТЕКТУРЫ «ЭЛЬБРУС»

Сопоставляются возможности контроля выхода за границу памяти, основанные на аппаратной поддержке и программных методах. Особое внимание уделяется аппаратной поддержке защищенного режима исполнения пользовательских приложений в архитектуре «Эльбрус». В качестве программного решения рассматривается AddressSanitizer – один из распространенных инструментов обнаружения ошибок обращения к памяти. Выполнен сравнительный анализ достоинств и недостатков этих средств. Приведено сравнение производительности задач, скомпилированных для защищенного режима, и задач, инструментированных AddressSanitizer.

Ключевые слова: *архитектура Эльбрус, AddressSanitizer, защищенный режим, ошибки обращения к памяти.*

Введение

При использовании таких языков, как C, C++, в крупных проектах неизбежно приходится сталкиваться с ошибками обращения в некоторую область памяти. Эта область может быть глобальной, выделяемой при запуске программы и доступной на протяжении ее исполнения, может располагаться в «куче» (heap), т.е. динамически выделяться и освобождаться через обращение к операционной системе, может выделяться в пользовательском стеке специальной инструкцией процессора. В архитектуре «Эльбрус» реализованы разнообразные средства обнаружения ошибок обращения к памяти, которые выполняются как аппаратурой, поддерживающей защищенный режим исполнения программ, так и программным путем при использовании AddressSanitizer.

Поддержка защищенного режима в микропроцессорах с архитектурой «Эльбрус»

В числе аппаратных средств, поддерживающих защищенный режим исполнения программ, в архитектуре «Эльбрус» реализована аппаратная поддержка типизации данных в применении к указателям.

Ключевым понятием в концепции защищенного режима является дескриптор. Это объект, описывающий определенные данные в памяти, тип которых задается его тегом. Рассмотрим некоторые типы дескрипторов:

- Дескриптор метки процедуры. В значении дескриптора содержится виртуальный адрес участка кода, на который будет совершен переход.
- Дескриптор указателя на массив. Значение дескриптора описывает область памяти определенного размера и представляется структурой с тремя полями – базового виртуального адреса, размера и текущего указателя. Первые два поля не могут быть изменены произвольным образом, поле текущего указателя доступно для модификации.
- Дескриптор указателя на массив в стеке. Эквивалентен дескриптору указателя на массив. Разница заключается в наличии поля, называемого уровнем процедуры в стеке, которое будет рассмотрено ниже.

Помимо этого, существует ряд тегов, обозначающих другие типы.

Применительно к операциям с адресными данными справедливы следующие условия:

- При операциях целочисленной и вещественной арифметики возникает прерывание, если в качестве аргументов используются значения адресного типа, – адресная арифметика реализуется специальными инструкциями.
- При операциях обращения в память по дескрипторам возникает исключительная ситуация, если

в качестве аргументов используются значения несоответствующего типа.

На работу с тегами накладываются следующие ограничения:

- При обмене между памятью и регистрами процессора данные перемещаются вместе со своими тегами.
- Теги не могут быть изменены произвольным образом. Явное изменение возможно только привилегированными командами: в непривилегированном режиме доступно только ограниченное подмножество команд, формирующих и модифицирующих теги.

Другой концепцией в построении защищенного режима является модульный подход. Модуль представляет собой готовый к загрузке файл – это может быть либо исполняемая программа, полученная в результате статической линковки, либо разделяемая библиотека, полученная в режиме сборки позиционно-независимого кода. Контекстом модуля называется совокупность кода и данных этого модуля. Под данными понимаются объекты в памяти, время жизни которых равно времени жизни исполняемой программы. При загрузке исполняемой программы все разделяемые библиотеки, подгружаемые динамическим линковщиком, являются отдельными модулями со своим собственным контекстом. В защищенном режиме каждый модуль описывается парой контекстных регистров, являющихся указателями на код и данные этого модуля. С их помощью производится аппаратный контроль выхода за границу адресов, доступных текущему модулю. Регистр, описывающий область контекста данных, называется дескриптором глобалов (GD). Регистр, описывающий код модуля, называется дескриптором единицы компиляции (CUD). При исполнении операции вызова производится анализ адреса назначения перехода. Если он выходит за границы области, описываемой регистром CUD, то происходит аппаратное переключение контекстных регистров на соответствующие контекстные регистры того модуля, к которому относится адрес перехода [1].

Стоит отметить, что в архитектуре «Эльбрус» все адреса возврата в вызывающую процедуру находятся в специальной области памяти, называемой стеком связующей информации. Стек связующей информации недоступен для записи непривилегированными операциями, что делает невозможной подмену адреса возврата. При выполнении операции вызова функции происходит автоматическое сохранение адреса возврата в стек связующей информации, а при исполнении соответствующей операции возврата из процедуры – автоматическое считывание из стека связующей информации.

Аппаратная поддержка контроля обращения к памяти в защищенном режиме

Важнейшим отличием адресации данных в защищенном режиме является расширение функциональности указателя до концепции дескриптора, в котором не только задается расположение данных в памяти, но также хранится дополнительная информация, необходимая для полного контроля за выделенной памятью на аппаратном уровне. В защищенном режиме все обращения к любым объектам в памяти осуществляются только через дескрипторы. В свою очередь, операции над дескрипторами и действия с ними строго регламентированы. Дескриптор может быть сформирован только специальной командой при создании объекта в памяти или при системном заказе области памяти. Во втором случае операционная система вместе с заказываемой областью конструирует и описывающий ее дескриптор, т.е. создание дескриптора и выделение описываемой им области памяти является атомарной операцией.

Предотвращение некорректного обращения к глобальной памяти пользовательского приложения осуществляется на аппаратном уровне. Все обращения к глобальной памяти могут выполняться только через дескриптор глобалов данного модуля, и обращение за пределы заданных в нем адресов приводит к аппаратному прерыванию.

Контроль за переполнением динамической памяти выполняет операционная система. Любой системный заказ динамической памяти осуществляет также создание дескриптора этой области, в котором содержится информация о ее начале и размере. Всякое обращение за эти границы через сформированный дескриптор приведет к генерации исключительной ситуации.

Выход за границу стековой памяти на основе аппаратной поддержки обнаруживается с помощью дескрипторов специального вида, в которых, помимо прочего, содержится информация о глубине текущего стекового фрейма – это целочисленное значение, которое называется уровнем процедуры в стеке (procedure stack level – *psl*). Дескриптор, описывающий объект в стеке, нельзя сохранять в области памяти, время жизни которой превышает время жизни данного стекового объекта. Другими словами, дескриптор стекового объекта, находящегося в текущем стековом фрейме процедуры, можно сохранять только в этом фрейме либо передавать в качестве параметра в вызываемую процедуру (передавать вверх по стеку). Для соблюдения данного ограничения и предназначены значения *psl*.

Учитывая сказанное, контроль за стековыми объектами осуществляется следующим образом. Специальный системный регистр содержит информацию о выделенной в стеке памяти, указатель на текущий стековый фрейм и значение его глубины (*psl*). При каждом выполнении операции вызова значение *psl* инкрементируется, при выполнении соответствующей операции возврата из процедуры совершается обратное действие. Далее, при выделении памяти в стеке текущий *psl* записывается во вновь сформированный дескриптор и в дальнейшем используется для контроля границ памяти. При занесении в стековую память дескриптор объекта в стеке нельзя записывать в глубину стека, т.е. *psl* адреса записи не должен быть меньше *psl* записываемого дескриптора.

Программный контроль обнаружения ошибок обращения к памяти

На текущий момент в оптимизирующем компиляторе для архитектуры «Эльбрус» реализована утилита AddressSanitizer [2, 3], которая изначально разрабатывалась как часть проекта LLVM, а затем была реализована и в компиляторе GCC [4]. AddressSanitizer состоит из двух частей, первая из которых входит в состав компилятора, а вторая – в библиотеку поддержки *compiler-rt* [5]. Принцип работы состоит в инструментировании кода пользовательского приложения на этапе компиляции, в результате чего во время его исполнения возможен вызов функций из библиотеки *compiler-rt*, обеспечивающей поддержку работы с динамической памятью, а также предоставляющей ряд интерфейсов для подробного отчета о произошедшей ошибке в пользовательской программе.

Определим несколько понятий, используемых в дальнейшем тексте. «Красной зоной» называется область памяти, необходимая для контроля границ выделенной пользователю памяти. Память, которая не была затребована пользовательским приложением, называется «отравленной».

Основная идея сводится к разбиению виртуального адресного пространства приложения на три зоны, первая из которых содержит адреса, доступные пользователю, вторая (так называемая «теневая память») хранит метаданные для первой зоны, третья недоступна для пользовательского приложения и представляет собой чисто техническое решение для предотвращения обращения пользователя к теневой памяти. Между элементами первых двух зон установлено взаимно-однозначное соответствие – каждые 8 байт первой зоны отображаются в 1 байт второй, т.е. значение каждого байта теневой памяти описывает состояние адресов пользовательской памяти. Возможны следующие варианты:

- Нулевое значение байта теневой памяти означает, что все байты, расположенные по адресу в пользовательской памяти, являются неотравленными.
- Отрицательное значение в байте теневой памяти означает, что все 8 байт пользовательского приложения являются отравленными.
- Байт в теневой памяти содержит положительное число k. Это значит, что первые k байт в пользовательской памяти являются неотравленными, а остальные 8-k байт – отравленными.

Принцип работы AddressSanitizer состоит в следующем. На этапе компиляции пользовательской программы для каждой операции обращения к памяти, будь то чтение или запись, происходит построение кода, осуществляющего вычисление соответствующего адреса теневой памяти с последующим считыванием байта по этому адресу, а затем выполняется проверка значения данного байта. Суть проверки состоит в том, что если в данном байте содержится ненулевое положительное значение k, то производится переход на дополнительный код, который осуществляет проверку того, что адрес последнего байта, к которому происходит доступ, меньше, чем 8-k отравленных байт пользовательской памяти. Если данное условие не выполнено или значение байта в теневой памяти является отрицательным, то производится вызов функции, сообщающей об ошибке в пользовательском приложении, из библиотеки *compiler-rt*. Методы обнаружения ошибок различных видов памяти состоят в следующем.

Контроль границ динамической памяти производится средствами библиотечной части AddressSanitizer. Все функции работы с динамической памятью, такие как *malloc* и *free*, заменяются собственными версиями, предоставляемыми библиотекой. Любой запрос на выделение динамической памяти приводит к формированию вокруг этой памяти красных зон и последующее отравление этих зон. Таким образом, при всяком обращении к динамической памяти за пределами выделенных зон, как было описано выше, происходит вызов функции, сообщающей о переполнении «кучи». Вызовы функций по освобождению динамической памяти регистрируют, что данный участок памяти является освобожденным. Как следствие, это помогает при детектировании ошибок использования памяти после ее освобождения.

При работе с глобальной памятью поддержка контроля ее границ осуществляется как инструментальной, так и библиотечной частью. Каждый глобальный по времени жизни объект в процессе инструментирования приложения дополняется красной зоной. Затем строятся вызовы функций

из библиотеки *compiler-rt*, которые регистрируют данные глобалы, а также отравляют их красные зоны. Так как каждое обращение к памяти подвергается проверке на ее корректность, то обращение к красной зоне любого глобала приведет к завершению работы приложения.

И, наконец, обнаружение ошибок обращения к стеку возлагается на инструментальную часть AddressSanitizer, реализуемую в компиляторе. Все объекты, создаваемые в стеке, дополняются красными зонами. Затем происходит построение кода, производящего отравление соответствующих красных зон стековых объектов. При любом обращении за границу стекового объекта происходит вызов функции, сообщающей об ошибке переполнения стека. При возврате управления из функции красные зоны всех стековых объектов подвергаются очистке.

Сравнительный анализ методов аппаратной поддержки защищенного режима и программных методов AddressSanitizer

Главным ограничением, которое накладывает защищенный режим, является предъявление жестких требований к работе с языковыми указателями, касающихся приведения типов между указателями и целыми числами. Так как адресация пользовательских данных в защищенном режиме основана на понятии дескриптора, а не просто целого числа, представляемого неким адресом в виртуальном адресном пространстве пользователя, то тип языкового указателя не может быть отождествлен с числовым типом. Иначе говоря, для преобразования указателя к целому типу в архитектуре существует специальная операция, но обратное преобразование с последующим разыменованием указателя приведет к генерации исключительной ситуации. Причина заключается в том, что нельзя сформировать дескриптор отдельно от описываемого им участка памяти, т.е. отсутствует возможность восстановить дескриптор по адресу, не имея данных о реальном типе объекта, расположенного по данному адресу. Если бы данное требование не было обязательным, могла бы возникнуть ситуация, при которой дескриптор и описываемый этим дескриптором объект в памяти не совпадали бы по типу, что, в свою очередь, привело бы к потенциальному нарушению границ памяти.

Что касается программного обнаружения ошибок границ памяти с использованием AddressSanitizer, то здесь можно указать на некоторые недостатки.

Прежде всего отметим, что при инструментировании пользовательского приложения на каждую операцию обращения к памяти выполняется конструирование кода, проверяющего корректность границ данной памяти. В результате значительно увеличивается результирующий объем исполняемого файла и существенно снижается производительность исполняемой программы. Помимо этого, значительно увеличивается память, необходимая пользовательскому приложению, за счет создания красных зон для всех объектов.

Еще одна слабая сторона использования красных зон заключается в невозможности покрыть все адресное пространство пользователя. Например, если красная зона для некоего объекта составляет 32 байт, а обращение к памяти относительно его расположения превышает данное значение, то средствами AddressSanitizer обнаружить данное нарушение нельзя, в то время как при исполнении программы в защищенном режиме произойдет выработка исключительной ситуации по нарушению границ памяти.

Существенным недостатком AddressSanitizer является также отсутствие возможности обнаружить ошибки обращения к памяти на границе адресов, которые соответствуют двум смежным байтам теневой памяти. Первый теневой байт содержит информацию о том, что адрес, который описывается значением теневого байта, определен как неотравленный, тогда как второй байт указывает на то, что соответствующий адрес является отравленным. Подобная ситуация иллюстрируется следующим примером на языке С:

```
...
char a[10]; /* предполагаем, что адрес начала массива
выровнен на 8 байт */
int *p = (int *)&a[7];
...
/* здесь массив должен быть инициализирован */
...
return *p; /* чтение четырех байтов, последний из ко-
торых является некорректным */
```

Ошибочной ситуацией здесь является чтение по адресу, указываемому переменной-указателем *p*. Первые три прочитанных байта будут находиться в выделенной стековой памяти пользователя, а последний – выходит за границы заказанной памяти в стеке. При проверке границ памяти в теневую память отобразится адрес, содержащийся в переменной *p*, который будет соответствовать тому байту в теневой памяти, который содержит нулевое значение. Таким образом, данная ошибка останется незамеченной. При исполнении программы, содержащей данный код в защищенном режиме, упомянутая проблема будет обнаружена благодаря использованию дескрипторов, которые имеют точные данные о размерах доступной пользователю памяти.

Подводя итог, можно констатировать, что за счет использования дескрипторов в защищенном

Таблица. Отношение времен компиляции/исполнения задач при использовании AddressSanitizer и в защищенном режиме

Название задачи	Отношение времен компиляции	Отношение времен исполнения
164.gzip	1,846	2,258
256.bzip2	2,492	5,796
179.art	2,141	3,078
183.equake	2,761	2,613
188.ammp	1,908	1,690
Среднее геометрическое	2,113	2,096
Среднее арифметическое	1,852	2,570

режиме покрывается большинство случаев обнаружения ошибок обращения к памяти, независимо от размера смещения, причем все проверки на корректность диапазонов памяти происходят с аппаратной поддержкой, что значительно повышает производительность программ. В отличие от AddressSanitizer, здесь не требуется выделения красных зон вокруг объектов в памяти. В то же время:

- защищенный режим предусматривает серьезные требования в части приведения целочисленных типов к указателям. В случаях, когда перенос крупных проектов для защищенного режима доставляет большие трудности, для обнаружения ошибок обращения к памяти предпочтительно использовать AddressSanitizer;
- в защищенном режиме отсутствуют механизмы описания произошедшей в программе ошибки – если границы памяти были нарушены, то генерируется исключительная ситуация. AddressSanitizer предназначен именно для отладки пользовательского приложения, поэтому при обнаружении ошибки обращения к памяти будет предоставлен подробный отчет.

Сравнение показателей производительности

Как отмечалось, при инструментировании пользовательского приложения с помощью AddressSanitizer происходит существенное увеличение размера кода, а также замедление времени исполнения инструментированного приложения. Создание операций, проверяющих границы пользовательской памяти, также негативно влияет на скорость компиляции. Ниже приводятся результаты замеров производительности по времени компиляции и исполнению некоторых задач из пакета spec cpu2000 [6]. Замеры производились на машине «Эльбрус 401-PC» с микропроцессором «Эльбрус-4С» (тактовая частота 750 MHz). Задачи были скомпилированы с уровнем оптимизации – O2. Компиляция задач выполнена для двух режимов: защищенного режима исполнения и с помощью инструмента AddressSanitizer. В таблице показано отношение времен компиляции и исполнения для этих режимов.

Как видно из таблицы, увеличение времени компиляции при инструментировании, выполненном AddressSanitizer, по сравнению с компиляцией задачи для защищенного режима в среднем составляет 1,85 раза, а время исполнения инструментированной задачи в среднем возрастает в 2,57 раза.

Заключение

В работе было рассмотрено подмножество средств обнаружения ошибок выхода за границу памяти, имеющихся в архитектуре «Эльбрус» (помимо описанных механизмов средствами защищенного режима детектируются, например, такие виды ошибок, как использование неинициализированных данных или «зависших указателей»). В то же время использование инструмента AddressSanitizer позволяет обнаруживать и другие виды ошибок, например, обращение к памяти после ее освобождения или обнаружения утечек памяти, которое достигается интеграцией LeakSanitizer в AddressSanitizer. Дальнейшие работы будут направлены на исследование более широкой номенклатуры ошибок этого класса; кроме того, в проекте оптимизирующего компилятора для архитектуры «Эльбрус» планируется разработка ряда средств, позволяющих выявить различные виды ошибок в пользовательских приложениях.

СПИСОК ЛИТЕРАТУРЫ

1. Волконский В.Ю. Безопасная реализация языков программирования на базе аппаратной и системной поддержки // Вопросы радиоэлектроники. 2008. Т. 4. № 2. С. 98–141.
2. AddressSanitizer [Электронный ресурс]. URL: http://clang.llvm.org/docs/AddressSanitizer.html
3. AddressSanitizerAlgorithm [Электронный ресурс]. URL: https://github.com/google/sanitizers/wiki/AddressSanitizerAlgorithm

4. GCC4.8 Release Series Changes, New Features, and Fixes [Электронный ресурс]. URL: http://gcc.gnu.org/gcc-4.8/changes.html
5. «Compiler-rt» runtime libraries [Электронный ресурс]. URL: http://compiler-rt.llvm.org
6. SPEC CPU2000 V1.3 (RETIRED: February 2007) [Электронный ресурс]. URL: https://www.spec.org/cpu2000

ИНФОРМАЦИЯ ОБ АВТОРАХ

Русяев Роман Михайлович, инженер-программист, АО «МЦСТ»; аспирант, ПАО «ИНЭУМ им. И.С. Брука», 119334, Москва, ул. Вавилова, д.24, тел.: 8 (926) 317-77-44, e-mail: roman.m.rusiaev@mcst.ru.
Нейман-заде Мурад Искендер-оглы, к.ф.-м.н., начальник отделения, АО «МЦСТ», ПАО «ИНЭУМ им. И.С. Брука», 119334, Москва, ул. Вавилова, д.24, тел.: 8 (499) 135-88-69, e-mail: muradnz@mcst.ru.
Ермолицкий Александр Викторович, к.т.н., начальник отдела, АО «МЦСТ», 119334, Москва, ул. Вавилова, д.24, тел.: 8 (499) 135-60-94, e-mail: era@mcst.ru.
Волконский Владимир Юрьевич, к.т.н., заместитель генерального директора, ПАО «ИНЭУМ им. И.С. Брука», 119334, Москва, ул. Вавилова, д.24, тел.: 8 (499) 135-33-51, e-mail: vol@mcst.ru.

For citation: Rusyaev R.M., Neiman-Zade M. I., Ermolitskiy A.V., Volkonskiy V. Yu. Hardware and software detection means of memory access errors for «Elbrus» architecture. Voprosy radioelektroniki, 2017, no. 3, pp. 33–38.

R.M. Rusyaev, M.I. Neiman-Zade, A.V. Ermolitskiy, V. Yu. Volkonskiy

HARDWARE AND SOFTWARE DETECTION MEANS OF MEMORY ACCESS ERRORS FOR ELBRUS ARCHITECTURE

A brief review of Protected Execution Mode for user-space applications featured in «Elbrus» architecture is described first. Then, AddressSanitizer, a well-known utility by Google Inc, is considered as an example of a pure software technique of memory control. Comparative analysis of these solutions is given with performance flaws, applicability and boundary violation detection quality.

Keywords: Elbrus architecture, protected mode, AddressSanitizer, memory access errors.

REFERENCES

1. Volkonskiy V. Yu. Secure realization of programming languages based on hardware and system support. *Voprosy radioelektroniki,* 2008, vol. 4. no. 2, pp. 98–141.
2. AddressSanitizer. Available at: http://clang.llvm.org/docs/AddressSanitizer.html
3. AddressSanitizerAlgorithm. Available at: https://github.com/google/sanitizers/wiki/AddressSanitizerAlgorithm
4. GCC4.8 Release Series Changes, New Features, and Fixes. Available at: http://gcc.gnu.org/gcc-4.8/changes.html
5. «Compiler-rt» runtime libraries. Available at: http://compiler-rt.llvm.org
6. SPEC CPU2000 V1.3 (RETIRED: February 2007). Available at: https://www.spec.org/cpu2000

AUTHORS

Rusyaev Roman, graduate student, PJSC «Brook INEUM»; software engineer, JSC «MCST», 24, Vavilova st., Moscow, 119334, Russian Federation, tel.: +7 (926) 317-77-44, e-mail: roman.m.rusiaev@mcst.ru.
Neiman-zade Murad, PhD, head of department, JSC «MCST», PJSC «Brook INEUM», 24, Vavilova st., Moscow, 119334, Russian Federation, tel.: +7 (499) 135-88-69, e-mail: muradnz@mcst.ru.
Ermolitskiy Aleksandr, PhD, head of department, JSC «MCST», 24, Vavilova st., Moscow, 119334, Russian Federation, tel.: +7 (499) 135-60-94, e-mail: era@mcst.ru.
Volkonskiy Vladimir, PhD, deputy general director, PJSC «Brook INEUM», 24, Vavilova st., Moscow, 119334, Russian Federation, tel.: +7 (499) 135-33-51, e-mail: vol@mcst.ru.