На правах рукописи

Тетерев Михаил Александрович

МЕТОД ОБНАРУЖЕНИЯ ОШИБОК ПРИ РАБОТЕ С ПАМЯТЬЮ НА СТАТИЧЕСКОМ ЭТАПЕ ОТЛАДКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Специальность: 05.13.11
«Математическое и программное обеспечение
вычислительных машин, комплексов и компьютерных сетей»

А В Т О Р Е Ф Е Р А Т
диссертации на соискание ученой степени
кандидата технических наук

Москва - 2013

Работа выполнена в ОАО «Концерн «Моринсис-Агат», г. Москва.

Научные руководители:	доктор технических наук, профессор, профессор НИУ «МЭИ» Губонин Н.С.
	доктор технических наук, профессор, зав. кафедрой 304 МАИ (НИУ) Брехов О.М.
Официальные оппоненты:	доктор технических наук, профессор, генеральный конструктор ОАО «Государственный научно-исследовательский институт приборостроения» Гаврилин Б.Н.
	кандидат технических наук, доцент, зав. кафедрой «Автоматизированные системы обработки информации и управления» МГУЭСиИ Микрюков А.А.
Ведущая организация:	научно-исследовательский институт им. М.А. Карцева (НИИВК), 117437, г. Москва, ул. Профсоюзная, д.108.

Защита диссертации состоится « » декабря 2013 г. в _____ на заседании Диссертационного Совета Д212.125.01 при Московском авиационном институте (национальном исследовательском университете) «МАИ» по адресу: 125993, г. Москва, А-80, ГСП-3, ул. Волоколамское шоссе, д. 4.

С диссертацией можно ознакомиться в библиотеке МАИ.

Автореферат разослан «____» ноября 2013 г.

Ученый секретарь
диссертационного совета Д212.125.01
кандидат технических наук, доцент А. В. Корнеенкова

Актуальность работы

Надёжность и эффективность функционирования многих сложных систем народно-хозяйственного и оборонного назначения в значительной степени определяется надёжностью ПО. А на надёжность ПО таких систем, работающих в режиме реального времени, заметное влияние оказывают ошибки работы с памятью (ОРП) - программные ошибки при динамическом выделении и освобождении оперативной памяти. Более того, в ряде случаев такие ошибки могут приводить к катастрофическим последствиям. Поэтому своевременное выявление и устранение ОРП крайне важно. Стремительный рост сложности и объёма ПО, связанный с возрастанием сложности и ростом номенклатуры функций, выполняемых такими системами, приводит к необходимости постоянного совершенствования инструментария для выявления и устранения ОРП в ПО реального времени.

В настоящее время динамические системы решают сложные и ответственные задачи, поэтому очень важно, чтобы программное обеспечение таких систем содержало минимум ошибок.

В процессе программирования возникают ситуации, когда заранее не известно, сколько объектов – чисел, строк текста и прочих данных будет хранить программа. В таком случае при разработке программного обеспечения (ПО) используется динамическое выделение памяти. Многие современные системы характеризуются большим объемом данных, что также требует динамического выделения памяти.

ПО современных систем реального времени (СРВ) написано преимущественно на языках программирования C и C++. Данные языки программирования позволяют разработчику динамически распределять и освобождать память. Удобство динамического распределения памяти в C и C++ состоит в том, что язык программирования предоставляет доступ к памяти, позволяет изменять размер выделенной программе памяти, перемещать, копировать и освобождать ее во время исполнения приложения. Однако в результате написания исходного кода программ у программистов-разработчиков часто возникают ошибки работы с памятью, которые могут приводить к потере информации, некорректному поведению системы, системным сбоям и отказам, что критично для систем реального времени.

Программная ошибка при работе с памятью - это ошибка, которую совершает разработчик на стадии кодирования ПО, связанная с некорректным динамическим выделением или освобождением сегментов оперативной памяти (ресурса).

Критичными для систем реального времени являются ошибки, пагубно сказывающиеся на времени реакции системы на внешние воздействия или на работоспособность системы в целом.

В настоящее время можно выделить 2 основных направления поиска ошибок в программном обеспечении: на статическом и на динамическом этапе отладки ПО.

На динамическом этапе отладке известны методы: анализ трасс выполнения, мониторинг, различные виды тестирования. В основе динамического этапа отладки программного обеспечения лежит применение средств анализа непосредственно во время работы исследуемой программы (например, средство анализа Valgrind).

Применение таких средств может быть эффективным для обнаружения ошибок в процессе выполнения. Однако, помимо существенного снижения производительности тестируемой программы, данные средства проверяют корректность выполнения программы лишь на конечном множестве детерминированных наборов входных данных (тестов) и не могут гарантировать корректность работы исследуемой программы в процессе дальнейшей эксплуатации при произвольных наборах.

На статическом этапе отладке известны методы: дедуктивная верификации, верификация на основе проверки моделей, статический анализ. Статический этап отладки основан на анализе программного обеспечения, производимом без реального выполнения исследуемых программ. Анализ производится над исходным кодом проекта. Наиболее перспективным методом выявление программных ошибок при работе с памятью представляется статический анализ. Статический анализ позволяет охватывать весь исходный код проверяемых проектов, а процедура проверки не может повредить или изменить сам исходный код. К существующим алгоритмам статического анализа относятся: анализ потока управления, абстрактная интерпретация, анализ значений, включающий в себя интервальный анализ и анализ указателей; анализ зависимостей программы.

На статическом этапе отладки в настоящее время не существует эффективных автоматизированных средств, специализирующихся на поиске ошибок при работе с памятью на языках C и C++, пригодных для применения в рассматриваемой предметной области. Основной задачей современных средств статического анализа (например, Coverity, Cppcheck) является поиск логических и стилистических ошибок в исходном коде программного обеспечения, а также выдача советов по оптимизации анализируемой программы. Указанные средства не позволяют охватить весь спектр программных ошибок при работе с памятью, возникает большое количество ложных срабатываний, кроме того эти средства тратят излишнее время для поиска указанных ошибок.

Исходя из сказанного, создание методик и программно-инструментальных средств автоматизированного выявления программных ошибок при работе с оперативной памятью в сложных динамических системах, реализованных на языках программирования C и C++, является актуальной задачей.

Цель работы

Целью диссертационной работы является разработка и исследование методики обнаружения и локализации основных программных ошибок при работе с памятью на статическом этапе отладке программного обеспечения.

Объект и предмет исследования

Объектом исследования является программное обеспечение сложных динамических систем, включая системы реального времени.

Предметом исследования является процесс отладки программного обеспечения.

Научная задача исследования

Научную задачу исследования составляет разработка методики выявления программных ошибок при работе с памятью, позволяющей автоматизировано находить наиболее опасные для систем реального времени ошибки.

Декомпозиция научной задачи позволяет выделить следующие **частные задачи** исследования.

1) Критический анализ области, существующих подходов и инструментов обнаружения программных ошибок.

2) Классификация программных ошибок при работе с памятью в ПО.

3) Создание методики автоматизированного обнаружения программных ошибок при работе с памятью на основе анализа исходного кода ПО.

4) Создание на основе предложенной методики программного средства, позволяющего проводить анализ ПО на статическом этапе отладки за существенно меньшее время, чем существующие средства.

5) Сопоставительный анализ предложенного инструментария с известными.

Научная новизна

Научная новизна работы состоит в следующем:

1) Разработана методика обнаружения и локализации основных программных ошибок при работе с памятью, использующая оригинальный метод анализа исходного кода и выявления наиболее опасных ОРП на статическом этапе отладки программного обеспечения.

2) Предложен метод анализа исходного кода ПО, реализованного на языках программирования C/C++, для выявления ОРП. Метод *отличается тем*, что вводится механизм контроля состояний сегментов памяти после построения промежуточного представления данных в режиме статической отладки. Метод позволяет выявлять наиболее опасные ошибки работы с памятью в исходном коде ПО на статическом этапе отладки за короткий промежуток времени.

3) Разработана модель состояний объектов памяти и основанные на этой модели правила выявления ОРП, в совокупности направленные на минимизацию затрат времени.

4) Предложена классификация ошибок, *ориентированная на* программные ошибки при работе с памятью, являющиеся критичными для систем реального времени.

5) Предложена структура программного средства отладки ПО, *позволяющая обнаруживать ошибки работы с памятью* в программном обеспечении на статическом этапе отладки *за короткий промежуток времени*.

6) Предложена система из четырёх показателей качества и методика сравнения по этим показателям программно-инструментальных средств выявления ошибок работы с памятью на статическом этапе отладки, использующая результаты работы этих средств на проектах ПО разного объёма и сложности.

Достоверность результатов

Достоверность представленных в настоящей работе положений и выводов подтверждена полученными результатами экспериментальных исследований разработанного прототипа программного средства MEDIS для отладки ПО.

Практическая значимость

Разработанная методика предназначена для автоматизации обнаружения программных ошибок при работе с памятью в программном обеспечении. Применение данной методики позволяет повысить качество как вновь разрабатываемых, так и уже созданных программных систем, а также снизить объем трудозатрат при отладке путем переноса основной нагрузки на статический этап. Предложенная методика может быть использована для выявления ОРП и в других приложениях, которые критичны к ошибкам такого рода.

Реализованный в рамках данной работы прототип средства обнаружения и локализации ошибок позволяет автоматизировано находить наиболее опасные ОРП за короткий промежуток времени.

Теоретические и практические результаты диссертационной работы могут являться основой для разработки промышленного средства выявления программных ошибок.

Реализация результатов работы

Реализацией результатов работы является прототип программного средства отладки MEDIS, в котором использован разработанный в диссертации метод. В работе продемонстрированы функциональные возможности созданного прототипа, а также его эффективность на реальных проектах ПО СРВ.

Результаты диссертационной работы используются в текущих разработках ПО, что подтверждается соответствующим актом.

Апробация работы

Основные научные положения и результаты работы докладывались на научно-технических конференциях: «Взгляд в будущее 2011», «Состояние, проблемы и перспективы создания корабельных информационно-управляющих комплексов 2011», «Корабельные системы управления и обработки информации. Проектирование и изготовление 2011», «Корабельные системы управления и обработки информации. Проектирование и изготовление 2011», «Взгляд в будущее 2012», «Состояние, проблемы и перспективы создания корабельных информационно-управляющих комплексов 2012», «Корабельные системы управления и обработки информации. Проектирование и изготовление 2012», «Состояние, проблемы и перспективы создания корабельных информационно-управляющих комплексов 2013».

Публикации

По результатам диссертационной работы опубликовано 12 статей, в том числе 3 в изданиях, рекомендованных ВАК.

Получено авторское Свидетельство о Государственной регистрации программы для ЭВМ №2013616813.

Структура и объем работы

Диссертационная работа состоит из введения, трех разделов, заключения, списка использованных источников. Общий объем работы составляет 136 печатных страниц. Работа содержит 23 рисунка, 20 таблиц, список источников из 107 наименований, 2 приложения.

На защиту выносятся

1) Методика обнаружения и локализации основных программных ошибок при работе с памятью на статическом этапе отладке программного обеспечения, позволяющая повысить качество ПО и снизить трудозатраты на динамическом этапе отладки путем переноса основной нагрузки на статический этап.

2) Новый метод автоматизированного выявления программных ошибок при работе с памятью, позволяющий существенно снизить временные затраты при анализе ПО. Метод имеет малую чувствительность к потоку данных (кроме переменных, являющихся указателями на выделенную память), не учитывает межпоточное взаимодействие, не обрабатывает рекурсивные вызовы.

3) Модель состояний объектов памяти и правила выявления ошибок работы с памятью.

4) Классификация ошибок при работе с памятью, позволяющая систематизировать исследуемую предметную область в соответствии с особенностями ПО.

5) Структура программного средства автоматизированного обнаружения и локализации ошибок, связанных с некорректной работой с памятью в ПО, эффективность которого продемонстрирована как на ряде специально подобранных модельных примеров, так и на реальных проектах.

6) Система из четырёх показателей качества и методика сравнения по этим показателям программно-инструментальных средств выявления ошибок работы с памятью на статическом этапе отладки.

СОДЕРЖАНИЕ РАБОТЫ

Во **введении** дана общая характеристика работы, раскрыта ее актуальность, сформулированы основные задачи и цель исследования, определены научная новизна и практическая значимость полученных результатов, а также основные положения, выносимые на защиту.

В **первом** разделе описаны источники и причины возникновения ошибок при работе с оперативной памятью и их классификация. По данным компании Coverity, в программах на языке C, в среднем, содержится 0.25 нефункциональных ошибок на 1000 строк исходного кода. Ошибки работы с оперативной памятью составляют существенную долю среди всего спектра ошибок программного обеспечения и являются особо критичными для систем реального времени, в то же время считается, что данный тип ошибок является одним из самых трудно обнаруживаемых.

Перечень функций и операторов, осуществляющих работу с оперативной памятью ЭВМ (распределение и освобождение), рассматриваемый в данной работе, представлен в таблице 1.

Таблица 1. Функции и операторы работы с памятью в языках C и C++

Наименование функции/оператора	Тип работы с памятью	Язык программирования
malloc	выделение	C, C++
free	освобождение	C, C++
calloc	выделение	C, C++
realloc	выделение	C, C++
new	выделение	C++
delete	освобождение	C++
new[]	выделение	C++
delete[]	освобождение	C++
fopen	выделение (ресурс)	C, C++
fclose	освобождение (ресурс)	C, C++
popen	выделение (ресурс)	C, C++
pclose	освобождение (ресурс)	C, C++
asprintf	выделение	C, C++
vasprintf	выделение	C, C++

Большие проекты ПО насчитывают сотни и тысячи вызовов функций распределения и освобождения памяти.

В настоящее время известны следующие способы предотвращения ошибок при работе с памятью:

1) Отказ от динамической памяти. Например, FORTRAN-77 полностью отказывается от применения механизмов динамического распределения памяти, что исключает подобные ошибки, но существенно ограничивает функциональность программ.

2) Перезапуск программы. В случаях, когда устранение утечки памяти не представляется возможным, например, при использовании кода, поставляемого в виде программных модулей и изготовленного сторонними разработчиками, применяется способ игнорирования утечек памяти. Код, подверженный утечкам, размещается в отдельной программе, а эта программа с нужной периодичностью перезапускается. Завершение и новый запуск программы выполняются внешней программой, которая также подаёт входные данные и забирает результаты. Поскольку при завершении программы вся память, затребованная ей у операционной системы, возвращается операционной системе, такой метод не позволяет утечкам приобрести катастрофический характер. Однако данный подход неприменим к системам, предназначенным для длительной работы без перезагрузок.

3) Владеющие указатели позволяют согласовать время жизни объекта и указателя на этот объект. Тем не менее, использование владеющих указателей не помогает в случае циклических ссылок между объектами.

4) Специальные инструментальные средства, предназначенные для отладки использования памяти, обнаружения утечек памяти, а также профилирования.

5) Сборка мусора. Некоторые языки программирования (например, Java, Оберон, языки платформы .NET) предоставляют средства, позволяющие автоматически освобождать неиспользуемую память. Сборщики мусора решают также и проблему циклических ссылок, но сборка мусора является ресурсоёмкой операцией. В результате использование подобных средств снижается быстродействие системы.

Анализ современных средств контроля памяти показывает, что они находятся на достаточно высоком уровне и позволяют находить основные ошибки при работе с памятью в приложении. Однако указанные средства не позволяют охватить весь спектр ошибок при работе с памятью, возникает большое количество ложных срабатываний, кроме того данные средства имеют излишне длительное для поиска указанного рода ошибок время работы.

Наиболее перспективным методом обнаружения ОРП представляется статический анализ.

Статический анализ основан на исходном коде программного обеспечения. По этой причине, первым этапом выполнения статического анализа является построение модели по исходному коду программы. На следующем этапе происходит применение собственно алгоритмов статического анализа с целью аппроксимировать состояния программы во всех точках. Алгоритмы, используя семантику конструкций языка (а точнее – модели программы), вычисляют возможные значения всех объектов программы. На последнем этапе рассчитанные состояния используются алгоритмами обнаружения ошибок для выявления и локализации имеющихся в программе ошибок.

К существующим алгоритмам статического анализа относятся: анализ потока управления, абстрактная интерпретация, анализ значений, включающий в себя интервальный анализ и анализ указателей; анализ зависимостей программы;

обнаружение дефектов; анализ сложных программ, включающий в себя: анализ фи-функций, анализ циклов и межпроцедурный анализ.

В настоящее время широко распространено программно-инструментальное средство (ПИС) Cppcheck – инструмент для статического анализа кода, написанного на языках программирования C и C++. Он служит для обнаружения ошибок и потенциальных уязвимостей, которые не обнаруживаются компилятором.

Основные проблемы при анализе исходного кода данным средством возникают с проектами большого объема, время анализа которых может быть неоправданно большим.

Второй раздел посвящен разработке методики обнаружения программных ошибок при работе с памятью.

Предлагается **классификация ошибок при работе с памятью,** которая подразумевает разделение основных ОРП на 5 категорий:

1. Повторное выделение ресурса без его предварительного освобождения (ошибка 1 вида).

Данная ошибка возникает в случае, когда программист динамически выделяет сегмент памяти, сохраняя адрес выделенного участка памяти в переменную, уже являющуюся указателем на динамически выделенный участок памяти.

2. Потеря последней ссылки на ресурс (ошибка 2 вида).

Данная ошибка возникает при динамическом выделении памяти, когда не производится сохранение адреса выделяемого участка памяти или не производится освобождения более не используемого локального ресурса перед выходом из функции.

3. Освобождение ресурса до его выделения (ошибка 3 вида).

Данная ошибка происходит при освобождении ресурса (памяти), который не был предварительно выделен с помощью соответствующего вызова функции динамического выделения ресурсов.

4. Повторное освобождение ресурса (ошибка 4 вида).

Данная ошибка связана с вызовом функции освобождения для ресурса (сегмента памяти), который уже прошел процедуру освобождения, но не был после этого снова выделен.

5. Освобождение ресурса функцией, не предназначенной для освобождения этого типа ресурсов (ошибка 5 вида).

К данной категории относятся ошибки, связанные с несоответствием использования функций распределения и освобождения памяти. Например, распределение с помощью оператора *new*, с последующим освобождением с помощью функции *free*().

Приведённая классификация ошибок при работе с памятью охватывает спектр ошибок, наиболее часто возникающих у программистов при работе с оперативной памятью, способных критически сказаться на работоспособности систем реального времени.

Разработанная **методика обнаружения ОРП** предполагает разделение множества исследуемых проектов ПО на классы сложности: A – простой; B –

средний; С - сложный проекты (см. табл. 2). Отнесение проекта к одному из классов определяется числом $N_{стр}$ строк исходного кода.

Таблица 2. Классы ПО

Класс	*A*	*B*	*C*
$N_{стр}$	$N_{стр} \in [1,10^3]$	$N_{стр} \in (10^3,10^6]$	$N_{стр} \in (10^6,\infty]$

При **построении модели программы** для проектов, принадлежащих к классам А и В, применяется синтаксический анализ с использованием средств компиляции, которые позволяют получить доступ к промежуточному представлению данных.

Для построения модели программы для проектов, принадлежащих к классу С, предлагается использовать упрощенный синтаксический анализатор.

После построения модели необходимо вычислить возможные **состояния программы** в различных точках ее исполнения.

Для проектов, принадлежащих к классу сложности А и В, используется метод абстрактной интерпретации.

Метод абстрактной интерпретации по принципу работы напоминает традиционную интерпретацию программы. Традиционный интерпретатор оперирует конкретными значениями переменных, хранящихся в памяти; состояние программы в конкретный момент ее выполнения с точки зрения такого интерпретатора представляет собой множество пар (obj, value), где obj- программный объект, а value – его значение. При абстрактной интерпретации состояние программы представляется множеством пар (obj, valueset), причем valueset – множество значений программного объекта.

Для проектов, принадлежащих к классу сложности С, применяется метод упрощенного анализа потока управления.

При использовании **алгоритмов анализа значений** состояние программы представляется в виде множества пар (ячейка памяти, множество допустимых значений). В ячейках памяти может располагаться как переменная примитивного типа, так и элемент массива или поле структуры.

Существует два основных вида значений ячеек памяти:

- числовые значения, для которых используется интервальный анализ для проектов класса сложности А и В.

Алгоритмы интервального анализа предназначены для определения возможных значений переменных числового типа. Операции над точными значениями выполняются в соответствии с правилами того языка, на котором написана программа. При анализе ветвлений интервальные значения могут разъединяться с последующим объединении при анализе фи-функций.

- указатели, для которых используется анализ указателей.

Алгоритмы анализа указателей предназначены для определения возможных значений переменных указательного типа и используются для проектов сложности всех трех классов.

Алгоритмы анализа зависимостей применяются при анализе проектов класса сложности А и В.

Зависимость – определенная связь между значениями двух или нескольких ячеек памяти. При анализе фи-функций происходит пересечение имеющихся в разных ветвях зависимостей.

При **анализе циклов** если количество итераций циклов неизвестно, то используются следующие способы решения подобной проблемы:

1. Для проектов класса А несколько итераций цикла анализируются обычным образом, после чего делается попытка выделить «инвариант цикла» и выполнить последнюю итерацию анализа в обобщенной форме. Инвариантом в данном случае называется утверждение или набор утверждений, справедливых на любой итерации цикла.
2. Для проектов класса В и С выполняется прерывание анализа цикла на итерации с заданным номером (для проекта класса С - 1). Состояние программы на выходе данной итерации переносится на выход цикла.

При **анализе рекурсивных вызовов** для проектов класса А и В при небольшом уровне рекурсии значения формальных параметров необходимо хранить отдельно для каждого вызова функции. При большом уровне рекурсии используется подход контекстно-нечувствительной аппроксимации: функции анализируются один раз при обобщенном значении формальных параметров и в дальнейшем используется полученный результат.

Анализ многопоточных программ производится для проектов класса А. Для представления функций над потоками и выбранными объектами синхронизации используется базис конструкций управления параллельным выполнением программы. Для привязки конструкций программы к потокам, в которых они выполняются, конструкции объединяются в блоки программы. Блок программы - множество последовательных конструкций, среди которых отсутствуют межблоковые конструкции, которыми являются *if* , фи-функции, *init* , *create* , *join* , *lock* , *unlock* , *wait* , *post* и *state.* Конструкции синхронизации взаимодействуют друг с другом с помощью блокировки и освобождения объекта синхронизации с помощью отношений синхронизации.

Предложенная методика позволяет обнаруживать ошибки работы с памятью в программном обеспечении на статическом этапе отладки за короткое время для проектов всех классов сложности. Наибольшую сложность при анализе представляют проекты класса С – крупные проекты.

Для выявления ОРП в сложных проектах ПО разработан **метод обнаружения ОРП**, который отличается от существующих введением механизма контроля состояний сегментов памяти после построения промежуточного представления данных в режиме статической отладки.

Структурная схема разработанного метода проиллюстрирована на рисунке 2.

Метод выявления ошибок при работе с памятью состоит из трех стадий: построения промежуточного представления данных, контроля состояний сегментов динамически выделенной памяти и генерации сообщений о выявленных ошибках.

На рисунке 3 изображен алгоритм обхода графа вызовов функций.

Правила работы семантического анализатора (СА) – это правила выявления ОРП по модели состояний объектов памяти.

Будем обозначать объект динамически выделенной памяти M_i^j, где i - порядковый номер вызова функции работы с динамической памятью в функции в соответствии с таблицей сведений по функциям работы с памятью, j - порядковый номер функции в таблице описаний функций.

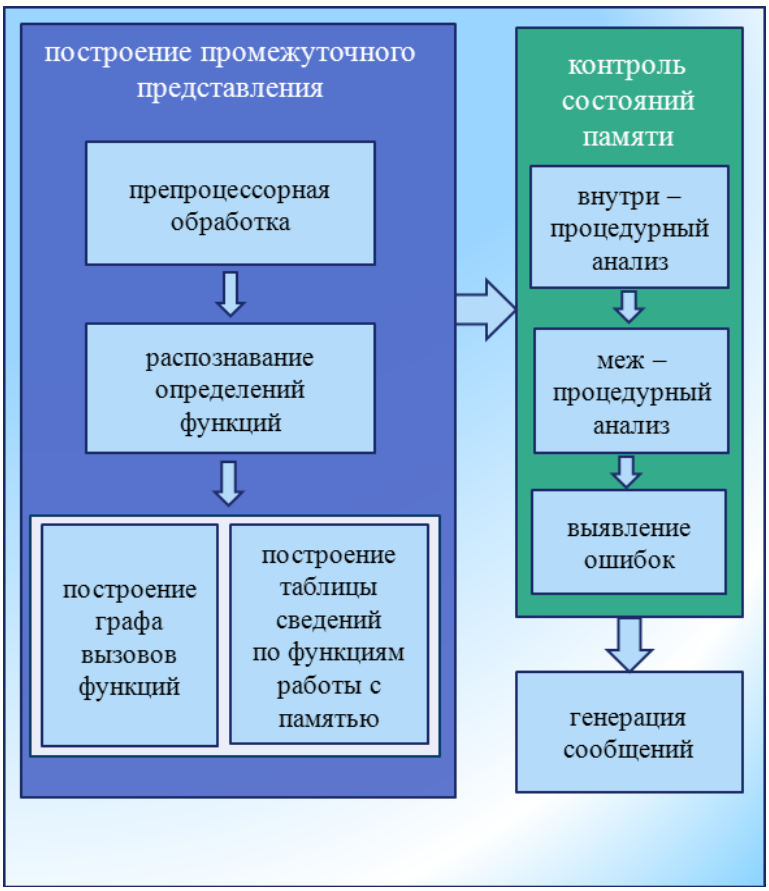

Рисунок 2. Структурная схема метода выявления ошибок

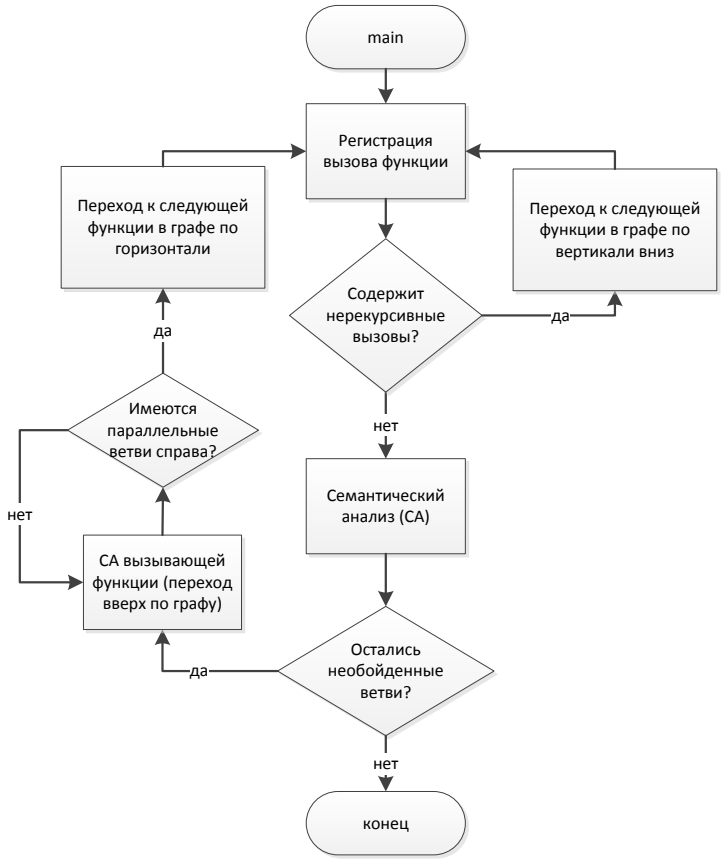

Рисунок 3. Блок-схема алгоритма обхода графа вызовов функций

Определим R_S - множество допустимых состояний памяти:

- «память, выделенная функцией ***malloc***()»;
- «память, выделенная оператором ***new***()»;
- «ресурс, выделенный функцией ***fopen***()»;
- «ресурс, выделенный функцией ***popen***()»;
- «память, освобожденная функцией ***free***()»;
- «память, освобожденная оператором ***delete***»;
- «ресурс, освобожденный функцией ***fclose***()»;
- «ресурс, освобожденный функцией ***pclose***()»;
- «пустое состояние (не соответствует ни одному из названных выше)».

Определим R_A - множество допустимых действий над объектом памяти:

- выделение памяти функцией ***malloc***() и производными;
- выделение памяти оператором ***new***;
- выделение ресурса функцией ***fopen***() и производными;
- освобождение памяти функцией ***free***();
- освобождение памяти оператором ***delete***;
- освобождение ресурса функцией ***fclose***() и производными;
- выделение памяти функцией ***asprintf***() и производными;
- бездействие.

Введем условные обозначения для элементов таблицы состояний памяти:

$\Psi(M_i^j)$ - имя переменной, содержащей указатель на сегмент памяти;

$\Omega(M_i^j)$ - текущее состояние объекта памяти; $\Omega \in R_S$;

$\Omega_к(M_i^j)$ - конечное состояние памяти; $\Omega_к \in R_S$;

$\Omega_0(M_i^j)$ - первоначальное состояние памяти; $\Omega_0 \in R_S$.

Обозначим η - номер параметра в функции, которым является данный объект памяти. Данный параметр используется при межпроцедурном анализе и задается значением «-1» для идентификации возвращаемого значения оператором ***return*** или порядковым номером параметра в вызывающей функции.

В таком случае множество имен указателей на динамически выделенную память, которые хранятся в таблице состояний памяти, будем обозначать R_Ψ.

Введем условные обозначения для элементов таблицы сведений по функциям работы с памятью:

$\Psi_0(M_i^j)$ - первоначальное имя переменной-указателя на динамически выделенную память;

$\Psi_k(M_i^j)$ - новое имя переменной-указателя на динамически выделенную память.

Будем обозначать текущее действие φ над объектом памяти M_i^j - $\varphi(M_i^j)$. Данное действие соответствует некоторому типу работы с памятью: $\varphi(M_i^j) \in R_T$.

Модель состояний объектов памяти, строящаяся при ***внутрипроцедурном*** анализе каждой функции для объекта памяти, который является локальным для данной функции, выглядит следующим образом:

$$\{\Psi(M_i^j) = \varnothing : [\Psi(M_i^j) = \Psi_0(M_i^j); \Omega_0(M_i^j) = \varphi(M_i^j); \Omega(M_i^j) = \Omega_0(M_i^j);\ \Omega_k(M_i^j) = \Omega(M_i^j)]\},$$

$$\{\Psi(M_i^j) \neq \varnothing : [Defect_0(\Omega(M_i^j), \varphi(M_i^j)); \Omega(M_i^j) = \varphi(M_i^j); \Omega_k(M_i^j) = \Omega(M_i^j)]\}. \quad (1)$$

$Defect_0$ - правила выявления ОРП.

Запись $\Psi(M_i^j) = \varnothing$: означает, что далее рассматривается математическое описание состояния для нового объекта памяти.

Модель состояний объектов памяти, строящаяся при ***межпроцедурном*** анализе, выглядит следующим образом:

$$\{\Psi_0 \in R_\Psi, \eta \neq \varnothing : [\Psi(M_i^j) = \Psi_k(M_i^j); \Omega(M_i^j) = \varphi(M_i^j)]\} \quad (2)$$

Для обозначения соответствия состояния памяти Ω одному из значений множества состояний памяти будет использоваться следующая запись:

$\Omega \in R_S[a-b]$, которая обозначает, что данное состояние совпадает с одним из состояний: $R_a, R_{a+1}, ..., R_{b-1}, R_b$, где $a < b$.

Аналогично для имен переменной, являющихся указателем на выделенную память.

Правила выявления ОРП $Defect_0$, используемые при построении и корректировки таблицы состояний памяти, выглядит следующим образом:

Обнаружение ошибки 1 вида (Err_1) «Повторное выделение ресурса без его предварительного освобождения» выполняется с помощью следующего правила:

$$Err_1: \{\Omega(M_i^j) \in R_S[1-4],\ \Psi(M_i^j) \in R_A[1,2,3,7]\}. \quad (3)$$

Обнаружение ошибки 2 вида (Err_2) «Потеря последней ссылки на ресурс» выполняется с помощью следующего правила:

$$Err_2: \{\Psi(M_i^j) \in R_A[1,2,3,7],\ \Psi(M_i^j) = \varnothing,\ N = \varnothing\}. \quad (4)$$

Обнаружение ошибки 3 вида (Err_3) «Освобождение ресурса до его выделения» выполняется с помощью следующего правила:

$$Err_3: \{\Omega_0(M_i^j) \in R_S[5-8]\}. \quad (5)$$

Обнаружение ошибки 4 вида (Err_4) «Повторное освобождение ресурса» выполняется с помощью следующего правила:

$$Err_4: \{\Omega(M_i^j) \in R_S[5-8],\ \Psi(M_i^j) \in R_A[4-6]\}. \quad (6)$$

Обнаружение ошибки 5 (Err_5) вида «Освобождение ресурса функцией, не предназначенной для освобождения этого типа ресурсов» выполняется с помощью следующих правил:

$$Err_5: \{\Omega(M_i^j) \in R_S[1],\ \Psi(M_i^j) \in R_A[5,7]\};$$

$$Err_5: \{\Omega(M_i^j) \in R_S[2],\ \Psi(M_i^j) \in R_A[4,6]\};$$

$$Err_5: \{\Omega(M_i^j) \in R_S[3,4],\ \Psi(M_i^j) \in R_A[4,5]\}. \quad (7)$$

Для окончательного выявления ОРП на этапе выявления ошибок помимо алгоритма $Defect_0$ используется следующее правило:

$$Err_2: \{\Omega_к(M_i^j) \in R_S[1-4]\}, \quad (8)$$

Разработанный метод статического выявления ошибок при работе с памятью позволяет обнаруживать ОРП за короткое время в крупных проектах.

Также в данном разделе приводится оценка сложности разработанного алгоритма, которая выражается формулой:

$$O(R+L)*N^2, \quad (9)$$

где

R - глубина графа вызовов.

L – размер функции в строках.

N - количество определенных в проекте функций.

В **третьем** разделе рассмотрены практические аспекты реализации результатов исследования.

Memory-related Errors Detection and Isolation System (MEDIS) – программно-инструментальное средство обнаружения и локализации ошибок, связанных с распределением сегментов оперативной памяти принцип работы которого основан на разработанном методе выявления ошибок в ПО.

Для оценки эффективности работы программно-инструментальных средств (ПИС) выявления ОРП предлагается следующий набор показателей качества (ПК), характеризующих работу:

- состав ошибок (виды ОРП), выявляемых данным ПИС,
- число выявленных (или, наоборот, число не выявленных) ошибок,
- число выявленных ошибочно (ложных) ошибок,

- время работы (необходимое время для выполнения всех предусмотренных операций).

Для сравнения выбрано ПИС Cppcheck в связи с тем, что данное средство широко распространено и имеет открытый исходный код.

Оценка ПК ПИС MEDIS и Cppcheck проводилась по результатам их работы на проектах ПО всех трёх классов. Прежде чем переходить к более общим результатам в работе приводится результат анализа одного из проектов класса В. При проведении данного испытания, как и всех последующих испытаний, описанных в работе, перед началом испытания вручную в Cppcheck были заданы все макроподстановки (в MEDIS это делается автоматически). Это требует длительного и кропотливого анализа, но позволяет существенно сократить последующее время определения ОРП, которое и приводится в таблицах. Если сначала не вводить все макроопределения, то Cppcheck затрачивает на обнаружение того же количества ОРП время порядка 10^4с.

Исследование показали, что для данного проекта по ПК k_1 оба ПИС *эквивалентны*, а по каждому из остальных ПК k_i $i=\overline{2,4}$ ПИС MEDIS *лучше,* чем ПИС Cppcheck. Следовательно, по совокупности ПК $\left\{k_i \mid i=\overline{1,4}\right\}$ ПИС MEDIS *предпочтительнее* ПИС Cppcheck по критерию Парето.

Выявленная тенденция оказывается справедливой и для других проектов ПО этого и других классов. Более того, в большинстве случаев оказывается, что и по ПК k_1 ПИС MEDIS *лучше,* чем ПИС Cppcheck. А это означает, что по совокупности ПК $\left\{k_i \mid i=\overline{1,4}\right\}$ ПИС MEDIS *предпочтительнее* ПИС Cppcheck по критерию Слейтера (каждый из 4-х ПК ПИС MEDIS *лучше,* чем соответствующий ПК ПИС Cppcheck). Однако сами значения ПК отличаются при переходе от проектов ПО одного класса к проектам другого класса. Менее сильно это сказывается на показателях k_1, k_3, которые характеризуют состав выявляемых ОРП и ложные ошибки. К тому же при практическом использовании ПИС основное внимание уделяется не этим показателям, а показателям k_2, k_4, определяющих число выявленных ошибок и затраченное на это время. Поэтому далее при анализе работы MEDIS и Cppcheck на ПО разных классов используется лишь два ПК: k_2, который характеризует «положительный эффект» работы ПИС, и ПК k_4, который характеризует затраченное время, т.е. «затраты». При этом для большей наглядности, используются показатели «число выявленных ошибок $n_{ош}$» и «время t». Для общей характеристики класса используются усреднённые значения этих показателей качества:

$$\left.\begin{array}{l} n_{ош}(S)=\dfrac{1}{n_S}\sum\limits_{i=1}^{n_S} n_{ош}(S,i),\ \ t(S)=\dfrac{1}{n_S}\sum\limits_{i=1}^{n_S} t(S,i) \\ \left\{\Pi(S,i)\mid i=\overline{1,n_S}\right\},\ \ S\in\{A,B,C\} \end{array}\right\}, \qquad (10)$$

где $n_{ош}(S,i)$, $t(S,i)$ – это число выявленных ОРП и затраченное на это время при испытаниях $i-$ го проекта ПО $П(S,i)$ из класса S.

В каждом из классов ПО *A, B, C* было выбрано по три проекта ПО и проведены испытания по выявлению ОРП с помощью ПИС MEDIS и Cppcheck. Полученные значения двух ПК для каждого проекта приведены работе. По этим данным для проектов всех классов были вычислены усреднённые показатели качества по формулам (10). Эти показатели приведены в табл. 3.

Также в данной таблице для каждого класса приведён усреднённый *обобщённый показатель эффективности* $Э(S)$ соответствующего ПИС:

$$Э(S) = \frac{1}{n_S}\sum_{i=1}^{n_S}\frac{n_{ош}(S,i)}{t(S,i)}, \qquad (11)$$

который имеет смысл среднего (по множеству проектов ПО данного класса) значения числа выявленных ОРП в единицу времени.

Таблица 3. Результаты анализа ПО средствами MEDIS и Cppcheck

Инструмент	Проект класса А			Проект класса В			Проект класса С		
	$n_{ош}(A)$ [ед.]	$t(A)$ [с]	$Э(A)$ [ед/с	$n_{ош}(B)$ [ед.]	$t(B)$ [с]	$Э(B)$ [ед/с]	$n_{ош}(C)$ [ед.]	$t(C)$ [с]	$Э(C)$ [ед/с]
MEDIS	4	0,14	17,7	17	4,8	3,6	64	10	6,6
Cppcheck	3	0,23	13,0	13	28,7	0,4	52	353	0,15

Сопоставление ПК для ПИС MEDIS и аналогичных ПК для ПИС Cppcheck показывает, что и по двум усреднённым значениям ПК для всех классов ПИС MEDIS предпочтительнее по Слейтеру, чем ПИС Cppcheck.

Помимо общего вывода о предпочтении ПИС MEDIS по Слейтеру для практики представляет интерес и количественная оценка преимуществ, даваемых этим ПИС по частным ПК $n_{ош}(S)$ и $t(S)$. Эти преимущества оцениваются показателями k_n «выигрыша в числе выявленных ошибок» и k_t «выигрыша в затраченном времени»:

$$k_n(S) = \frac{n^M_{ош}(S)}{n^C_{ош}(S)}, \quad k_t(S) = \frac{t^C(S)}{t^M(S)} \qquad (12)$$

Показатели (12), дополненные показателем «сравнительной эффективности $k_Э = Э^M / Э^C$, приведены в табл. 4. Из этих данных можно сделать следующие выводы:

- выигрыш в числе выявленных ошибок лежит в пределах (20-30)%,
- экономия во времени, и сравнительная эффективность резко возрастают с ростом сложности проектов ПО от единиц до (30-40) раз.

Таблица 4 Частные сравнительные показатели ПИС MEDIS и Cppcheck

Сравнительные	Проект класса А	Проект класса В	Проект класса С

показатели			
$k_n = n_{ош}^M / n_{ош}^C$	1,33	1,31	1,23
$k_t = t^C / t^M$	1,64	5,98	35.30
$k_Э = Э^M / Э^C$	1(1,36)	9	44

Таким образом, проведённое сравнение работы инструментов MEDIS и Cppcheck по группам проектов всех трёх классов показало, что во всех классах проектов ПО средство MEDIS по совокупности рассматриваемых показателей качества предпочтительнее средства Cppcheck по критерию Парето, а во многих случаях оно предпочтительнее также по критерию Слейтера.

В **заключении** подведены итоги работы, обобщены ее основные результаты, сделан вывод о целесообразности применения разработанной методики и средства в процессе проектирования и эксплуатации программного обеспечения, а также предложены направления дальнейших исследований.

ПУБЛИКАЦИИ ПО ТЕМЕ ДИССЕРТАЦИИ

1. Тетерев М. А. Анализ функционирования современных поисковых систем // Дни студенческой науки осень - 2009:сборник научных трудов. / Московский Государственный Университет Экономики, Статистики и Информатики, Москва 2009. - С. 76-83.

2. Тетерев М.А. Анализ современных средств контроля памяти в программном обеспечении. // Состояние, проблемы и перспективы создания корабельных информационно-управляющих комплексов: сб. докладов научно-технической конференции, 2011. - С. 158-163.

3. Тетерев М.А. Инструментальное средство “DIME” выявления ошибок работы с памятью специального программного обеспечения // Сборник докладов X молодежно-технической конференции “Взгляд в будущее”. / ОАО “ЦКБМТ “Рубин”, С-Петербург, 2012. - С. 659-665.

4. Тетерев М.А. Метод и средство автоматизированного поиска ошибок работы с памятью в специальном программном обеспечении // Сборник докладов молодежной научно-технической конференции, посвященной 65-летию ОАО «ГРЦ МАКЕЕВА» / ОАО «Государственный ракетный центр имени академика В. П. Макеева», 2012. - С.151-156.

5. Тетерев М.А. Обнаружение и локализация ошибок специального программного обеспечения, связанных с распределением сегментов оперативной памяти // Состояние, проблемы и перспективы создания корабельных информационно-управляющих комплексов: сб. докладов научно-технической конференции, 2012. – С. 43-47.

6. Тетерев М.А. Построение графа вызовов функций для поиска ошибок работы с памятью в СПО. // Корабельные системы управления и обработки информации. Проектирование и изготовление: сб. докладов научно-технической

конференции. / ОАО “Концерн “Научно-производственное объединение “Аврора”, С-Петербург, 2011. - С. 66-68.

7. Тетерев М.А. Применение метода статического анализа для поиска ошибок работы с памятью в СПО. // Сборник докладов IX молодежно-технической конференции “Взгляд в будущее”. / ОАО “ЦКБМТ “Рубин”, С-Петербург, 2011. - С. 567-572.

8. Тетерев М.А. Крылов Д.А., Валяев А.Н. Статическое выявление ошибок работы с памятью в специальном программном обеспечении // Состояние, проблемы и перспективы создания корабельных информационно-управляющих комплексов: сб. докладов научно-технической конференции, 2013. - С. 194-199.

9. Тетерев М.А., Крылов Д.А. Метод диагностики программных дефектов при работе с памятью в системах реального времени // Корабельные системы управления и обработки информации. Проектирование и изготовление: сб. докладов научно-технической конференции. / ОАО “Концерн “Научно-производственное объединение “Аврора”, 2012. – С. 55-58.

10. Тетерев М.А., Губонин Н.С. Анализ способов предотвращения ошибок работы с памятью в программном обеспечении // Вестник МЭИ, 2013, №4. С. 155-160.

11. Тетерев М.А., Губонин Н.С. Выявление программных ошибок при работе с памятью в исходном коде // Вестник МЭИ, 2013, №4. С. 166-172.

12. Тетерев М.А., Губонин Н.С. Метод и инструментальное средство MEDIS обнаружения и локализации ошибок при работе с памятью в программах, написанных на языках С и С++. Приборы и системы. Управление, контроль, диагностика, 2013, №9 С.12-18.