**Отладка систем реального времени**

К.А. Костюхин, НИИСИ РАН

**1. Введение**

**1.1. Основные определения**

Предметом настоящего обзора является отладка систем реального времени.

Под системой реального времени (СРВ) мы понимаем систему, в которой корректность функционирования зависит от соблюдения временных ограничений.

Существующие СРВ являются многозадачными. Многозадачность реализуется через многопроцессность\*) и многопоточность.

Под процессом понимается держатель ресурсов (например, память, данные, дескрипторы открытых файлов), которые не разделяются с другими процессами. В рамках одного процесса выполняются один или несколько потоков. Они совместно используют ресурсы процесса.

Многопроцессность в СРВ имеет существенные недостатки, поскольку требует поддержки времени выполнения для доступа к памяти, и, следовательно, при переключении контекстов системе нужно выполнить дополнительные действия.

Многопоточность - это наиболее распространенный подход при проектировании систем реального времени, при котором СРВ представляет собой один процесс, в рамках которого запущено несколько потоков.

Недостатком многопоточности является возможность модификации чужих данных какой-либо задачей (из-за отсутствия защиты). В связи с этим в СРВ представлены средства синхронизации, то есть средства, обеспечивающие задачам доступ к разделяемым ресурсам. К таким средствам относятся семафоры (бинарные и счетчики), мьютексы, очереди сообщений (см. [1],[3],[25]).

Структура СРВ приведена на рис.1, где прикладной код - это совокупность пользовательских потоков управления, ОСРВ - операционная система реального времени, обеспечивающая планирование, синхронизацию и взаимодействие пользовательских потоков управления.

 Рис. 1. Структура системы реального времени

Будем называть распределенную систему распределенной системой реального времени (РСРВ), если корректность ее функционирования зависит также и от ограничений, накладываемых на время обмена между компонентами системы.

1.2. Особенности отладки в системах реального времени

Отладка в СРВ направлена на обнаружение и исправление ошибок в прикладном коде. Она является одним из этапов кросс-разработки, схему которой можно представить следующим образом. Разработка приложения ведется как минимум на двух машинах: инструментальной и целевой. На инструментальной платформе происходит написание исходного текста, компиляция и сборка. На целевой - загрузка приложения, его тестирование и отладка.

Ввиду того, что целевая платформа, как правило, обладает более ограниченными ресурсами, чем инструментальная, отладка распределенных систем реального времени может быть двух видов.

Первый из них - имитация архитектуры целевой платформы, то есть возможность отладки целевых программных средств без использования самой платформы. Подобная имитация, как правило, не дает возможности провести подробное и полное тестирование ПО. Поэтому, такой тип отладки применяется только в случае отсутствия целевой платформы.

Второй способ - удаленная отладка (кросс-отладка). Кросс-отладка позволяет использовать ресурсы инструментальной системы при изучении поведения некоторого процесса в целевой системе.

Эффективность удаленной отладки зависит от типа связи инструментальной и целевой машин, а также от поддержки средств отладки со стороны целевой архитектуры.

Ключевым требованием к средствам отладки является возможность наблюдать и анализировать весь процесс выполнения отлаживаемых задач, а также системы в целом. В данной работе рассматриваются два метода отладки: активная отладка и мониторинг.

Суть активной отладки состоит в том, что отладчик имеет право останавливать выполнение задачи или всей системы, начинать или продолжать выполнение с некоторого адреса, отличного от точки останова, изменять значения переменных и регистров, и.т.д. Недостаток этого метода заключается в том, что отладчик может вносить серьезные сбои в нормальную работу системы в связи с устанавливаемыми временными ограничениями. Этого можно избежать, остановив некоторую группу задач или всю систему целиком, о чем будет подробнее сказано ниже. Преимущество метода состоит в возможности корректировать поведение задачи в процессе ее выполнения.

Под мониторингом понимается сбор данных о задаче (значения регистров, переменных, и.т.д) или о системе в целом (стадии выполнения задач, происходящие события, и.т.д).

Осуществлять сбор данных помогает псевдо-агент (набор инструкций, встроенных в код задачи). Обычно его добавляют на этапе проектирования. Простой пример псевдо-агента - вызов assert, позволяющий вести диагностику работы задачи.

В процессе мониторинга отладчик практически не вмешивается в работу системы, обеспечивая нормальное ее функцирование, но вместе с тем не имеет возможности влиять на ход выполнения отлаживаемого приложения.

1.3. Ошибки в системах реального времени

Отмеченные выше методы отладки позволяют выявлять и устранять ошибки следующего характера:

Ошибки в программном обеспечении, влекущие неправильное выполнение задачи (безотносительно времени). Обычные ошибки, обнаруживаемые средствами активной отладки. Эти средства будут рассмотрены в разделе 2.

Ошибки в ОСРВ: ошибки планирования, синхронизации и связи.Для отладки, в этом случае, надо использовать один из способов мониторинга. Подробно средства мониторинга описаны в разделе 3.

Логические ошибки, связанные с асинхронностью. Пример такого рода ошибок и способы их устранения будут приведены в разделе 4.

Ошибки, связанные с тем, что данные задачи были изменены другой задачей. Локализацию ошибок лучше проводить, используя мониторинг, а именно: осуществлять периодический контроль целостности данных, временно запрещать другим задачам доступ к некоторым участкам кода или данных.

2. Средства активной отладки

2.1. Архитектура средств активной отладки

В общем случае кросс-отладчик состоит из 2 основных модулей: менеджера на инструментальной платформе и агента отладки на целевой стороне. Менеджер служит для обеспечения пользовательского интерфейса, то есть для приема команд, их обработки и посылки на целевую сторону, а также для приема, обработки и выдачи информации от агента, который осуществляет непосредственную работу с отлаживаемой системой. Возможности агента отладки зависят от особенностей архитектуры системы, а именно:

имеет ли система встроенные средства отладки (в этом случае агенту достаточно осуществлять вызов соответствующих функций и пересылать результаты менеджеру);

какие она предоставляет возможности по созданию обработчиков (для контроля за происходящими событиями агенту может потребоваться собственный обработчик);

вызовы каких функций дозволено делать агенту.

Кроме того, агент отладки должен поддерживать возможность приема и передачи информации от псевдо-агентов, встроенных в код отлаживаемых программ. Агент отладки может состоять из нескольких модулей, например, один осуществляет сбор данных, другой производит фильтрацию, третий осуществляет пересылку данных менеджеру.

Общая структура активного кросс-отладчика приведена на рис.2.

 Рис. 2. Активный кросс-отладчик

Рассмотрим протокол "менеджер-агент" на примере отладчика VxGDB (Wind River Systems, целевая система - VxWorks). Этот протокол базируется на RPC (Remote Procedure Call). Запросы менеджера можно классифицировать следующим образом:

Запросы работы с модулями Сюда относятся запрос на загрузку модуля, запрос на получении информации о загрузочном файле и запрос на получение информации о символе.

Запросы работы с задачами Это запросы на запуск, остановку и удаление задачи, на присоединение и отсоединение от запущенной задачи, на установку и удаление точки прерывания, на продолжение выполнения остановленной задачи.

ptrace-запросы Агент отладки эмулирует функцию ptrace и передает ей соответствующие запросы на чтение и запись.

2.2. Отладочные действия

Существует набор базовых действий, позволяющих пользователю осуществлять контроль за выполнением отлаживаемой задачи. Эти действия можно классифицировать следующим образом:

предварительные действия отладчика;

прерывание выполнения задачи;

получение информации;

продолжение/изменение выполнения задачи.

1) Предварительные действия отладчика

загрузка модуля на целевой стороне;

запуск нужной задачи;

если задача была запущена средствами системы, то присоединение к ней (в этом случае выполнение задачи останавливается и становится возможным производить отладочные действия);

переключение между задачами.

Отладчик также выполняет и обратные действия, то есть отсоединение от задачи, прекращение ее выполнения и выгрузка соответствующего модуля.

2) Прерывание выполнения задачи

Как правило, выполнение задачи может быть прервано в результате того, что произошло одно из следующих событий:

Достигнута точка прерывания.

Точки прерывания бывают двух видов: программные и аппаратные. Установка программной точки прерывания состоит в том, что запоминается инструкция, расположенная по тому адресу, где будет находиться точка прерывания, затем по этому адресу прописывается так называемая "break"-инструкция, после обработки которой процессором генерируется некоторое исключение (trap exception). Обработчик этого исключения сохраняет регистры задачи, меняет инструкцию точки прерывания на настоящий код и передает управление менеджеру. Недостаток программных точек прерывания в том, что соответствующий адрес должен находиться в той области памяти, куда разрешена запись.

Аппаратная точка прерывания не требует модификации памяти, так как ее адрес хранится в специальном регистре (debug register, DR), который просматривается процессором при выполнении очередной инструкции. Если происходят действия, заложенные в контрольный регистр (например, выполнение по заданному адресу или чтение/запись в область, адресуемую значением DR), то процессор генерирует соответствующее прерывание, обработав которое, отладчик получает необходимую информацию.

Отладчик предоставляет пользователю возможность установки точки прерывания на начало некоторой строки исходного текста, на точку входа определенной функции или на конкретный адрес. Помимо этого, отладчик отличает точки прерывания, поставленные им, а также (в многозадачных системах) имеет возможность устанавливать и обрабатывать точки прерывания, специфичные для данного множества задач. Это означает, что если задача не входит в такое отлаживаемое множество, то точка прерывания игнорируется.

Помимо работы с точками прерывания, устанавливаемыми на конкретный адрес, отладчик работает с так называемыми прерываниями доступа (access breakpoints), используемыми для определения момента доступа к некоторой области памяти, и прерываниями наступления событий (event detection), которые срабатывают, когда происходит соответствующее событие.

Используется режим пошаговой отладки.

Существует много вариантов степени детализации выполнения. В частности, остановка после выполнения строки исходного текста или остановка после выполнения каждой ассемблерной инструкции (с заходом в вызываемые функции или с их пропуском).

Отладчик может использовать этот режим в служебных целях, например, при установленном прерывании выполнения при изменении данных (watchpoint). В этом случае после исполнения очередной инструкции надо проверять значение того выражения, на котором стоит точка прерывания.

Перехвачена исключительная ситуация.

Если в процессе выполнения отлаживаемой задачи возникла исключительная ситуация (например, деление на 0), то отладчик должен корректно ее обработать и сообщить пользователю.

Получен сигнал прерывания от пользователя.

Отладчик предоставляет пользователю возможность в любой момент остановить выполнение программы (например, введя соответствующий символ).

При получении сигнала останова отладчик может остановить текущую отлаживаемую задачу, остановить выполнение некоторой группы задач и остановить всю систему. Такой механизм дает возможность применять средства активной отладки без опасения вызвать новые ошибки, связанные со спецификой СРВ, поскольку система или набор задач, которые могут повлиять на отлаживаемую, будут остановлены, и временные соотношения их выполнения нарушены не будут. Подобный подход реализован в отладчике X-ray (Microtec Division, целевая система VRTX).

3) Получение информации

Когда задача остановилась, становится возможным осуществлять сбор различных данных, которые могут помочь при локализации логических ошибок в программе.

Просмотр содержимого стека.

Эта команда дает возможность увидеть, как задача попала в текущее положение. Каждый кадр стека содержит данные, связанные с вызовом некоторой функции, включая ее аргументы, локальные переменные и адрес точки вызова.

Просмотр таблицы символов.

Используя отладчик, можно получать доступ к информации о символах (имена переменных, функций и типов), определенных в задаче. Эта информация состоит из имени, типа и адреса соответствующей переменной или функции.

Просмотр исходных файлов.

Пользователь может выборочно смотреть исходный текст программы, задавая номера строк, имена функций или некоторый адрес.

Просмотр данных.

Отладчик способен получать и пересылать пользователю значение любой переменной или функции, доступной отлаживаемой задаче, а также содержимое регистров, памяти и дизассемблированный код.

Кроме того, у пользователя может возникнуть необходимость осуществить в процессе сеанса отладки вызов некоторой функции. Для поддержки этого существуют различные способы, например, можно передать соответствующий запрос агенту отладки, и тот запустит требуемую функцию либо в контексте отлаживаемой в данный момент задачи, либо в некотором специальном контексте. В GDB (GNU debugger, Free Software Foundation) реализован другой механизм, суть которого заключается в том, что все предварительные действия (установка точки выполнения, и.т.д.) производятся на инструментальной стороне, а агенту передается запрос на выполнение с текущего адреса.

4) Продолжение/изменение выполнения

Особенностью активной отладки является возможность изменения выполнения задачи.

Изменение данных.

Отладчик имеет возможность изменять значения переменных, содержимое регистров и памяти. Это позволяет корректировать выполнение задачи, проверяя предположения об ошибках в ее коде.

Продолжение выполнения задачи с любого адреса.

Пользователь может потребовать продолжить выполнение задачи с другого адреса (например, обходя критический участок кода).

Продолжение выполнения задачи до некоторого адреса.

В этом случае ставится временная точка прерывания на нужный адрес, и задача выполняется, пока этот адрес не будет достигнут. Как и в случае с обычной точкой прерывания, отладчик обеспечивает установку временной точки прерывания на начало строки исходного текста, на конкретный адрес или на точку входа некоторой функции. Однако, помимо этого отладчик может ставить временную точку прерывания на адрес возврата текущей функции, реализуя выполнение задачи вплоть до завершения этой функции.

Механизм установки временной точки прерывания используется и в режиме пошаговой отладки. Тогда точка прерывания ставится на следующую исполняемую инструкцию или (в случае отладки без захода в вызываемые функции) на инструкцию, следующую за вызовом функции.

Возврат из функции.

Может возникнуть ситуация, когда пользователю понадобится завершить выполнение функции с определенным возвращаемым значением. В этом случае отладчик выполняет следующие действия:

Приведение заданного пользователем значения к типу возвращаемого значения этой функции.

Восстановление сохраненных регистров.

Установка возвращаемого значения в требуемую область памяти/регистр.

Установка указателя стека на предыдущий кадр.

Установка точки выполнения программы на адрес возврата заданной функции.

Уничтожение текущего кадра стека.

Описывая отладочные действия, стоит упомянуть об инструментальной системе ЭСКОРТ ([27]), созданной в Научно-исследовательском институте системных исследований РАН как средство повышения производительности труда профессиональных программистов. Основу ЭСКОРТа составляет интегрированная система редактирования - компиляции - выполнения.

Программы в ЭСКОРТе имеют нетекстовое представление. Более точно, все программные объекты представлены как объекты базы данных. Средствами БД ЭСКОРТа реализовано хранение предыдущих состояний объектов (и, в частности, значений переменных), откатка, "откатка откатки" и т.п., что позволяет дать в руки программисту мощные средства контролируемого выполнения (пошаговое, с контролем значений переменных, обратное и т.д.). Правда, на сегодняшний день, для современных систем реального времени, средства, предлагаемые в рамках ЭСКОРТа, представляются слишком тяжеловесными (хотя и весьма перспективными).

2.3. Пользовательский интерфейс

Помимо получения необходимой информации отладчик должен предоставить ее в удобном для пользователя виде. Для этого служат интерфейсные команды и функции.

Интерфейс отладчика состоит из:

графического интерфейса;

режима комадной строки;

команд представления данных.

1) Графический интерфейс

Основное требование, предъявляемое к графическому интерфейсу активного отладчика - одновременная визуализация информации об отлаживаемой задаче (например, окно исходного текста, диалоговое окно, окно отображения данных и сообщений). При отладке группы задач, необходимо отображать полную информацию о каждой из них, как это реализовано в X-ray.

2) Режим командной строки

В режиме командной строки отладчик должен предоставлять пользователю возможность редактировать вводимые команды, а также поддерживать хранение ранее введенных команд в некотором буфере с целью их последующего вызова и модификации.

3) Команды представления данных

Приведем некоторые способы представления и хранения данных, реализация которых в значительной степени упрощает работу пользователя:

История значений.

Хранение ранее отображенных значений позволяет следить за изменением инспектируемого выражения. Значения могут храниться в некотором буфере или файле.

Внутренний язык отладчика.

Внутренний язык - средство, дающее возможность пользователю определять собственные функции. Кроме этого отладчик может обладать встроенным интерпретатором какого-нибудь известного языка сценариев. Например, VxGDB обладает встроенным TCL-интерпретатором, позволяющим не только определять новые функции обработки данных, но и (при поддержке с целевой стороны) эмулировать ряд функций VxWorks.

Поддержка разных форматов представления данных.

Уже упоминавшиеся средства отладки (VxGDB, X-Ray) предоставляют возможность вывода интересующего значения в любом числовом или символьном формате. Кроме того, в них имеется поддержка сложных элементов языка, таких как массивы или структуры.

Поддержка нескольких языков.

Если программа собрана из модулей, написанных на разных языках программирования, то для ее полноценной отладки необходима поддержка всех этих языков. Кроме того, VxGDB поддерживает синтаксис основных языков программирования (C, fortran) для своего внутреннего языка, обеспечивая автоматическую его смену при выполнении функции, реализованной на языке, отличном от текущего.

Регулярное отображение данных.

При отладке может потребоваться просмотр некоторых данных (или выполнение команд) при каждой остановке задачи.

GDB в данном случае предоставляет следующие возможности:

Команда display. После остановки выполнения задачи (но не в случае, когда она завершилась) производится отображение данных, заданных в качестве аргументов этой команды.

Команда commands привязывает к точке прерывания, номер которой задан в качестве аргумента, набор действий, которые нужно реализовать отладчику при достижении этой точки прерывания.

2.4. Интеграция со средствами разработки ПО

Обычно, программный продукт проходит стадии разработки, представленные на рис.3.

 Рис. 3. Стадии разработки ПО

В [24] описан способ, позволяющий уменьшить общее время разработки программного продукта за счет объединения средств тестирования и отладки. Такую возможность предоставляет отладчик Pilot (Kvatro Telecom). Подобное совмещение обладает следующими преимуществами:

сразу выявляются ошибки в тесте;

имеется возможность генерировать тесты в процессе отладки;

результаты теста сразу обрабатываются и в случае ошибки передаются отладчику, который интерактивно воспроизводит этот тест.

Средства поддержки отладки могут закладываться на стадии написания исходного текста и компиляции. Во время написания исходного текста в программный продукт может закладываться псевдо-агент - набор функций, осуществляющих некоторые отладочные действия.

Для отладки с использованием исходных текстов приложения, необходимо при компиляции генерировать дополнительную информацию, состоящую из описаний символов программы (переменные, функции, типы) и псевдо-символов, позволяющих отладчику определять адреса строк исходного текста, адреса секций, и.т.д.

3. Средства мониторинга

3.1. Архитектура средств мониторинга

Архитектура средств мониторинга в общем случае совпадает с архитектурой средств активной отладки, с той лишь разницей, что агенту отладки запрещено останавливать отлаживаемую задачу и модифицировать ее данные. Вообще, основное требование к средствам мониторинга - сбор данных с минимальным вмешательством в работу целевой системы. Для этого агенту отладки нужно как можно реже обращаться к менеджеру, то есть хранить полученные данные в некотором буфере и пересылать их по мере заполнения буфера. Кроме того, при пересылке данных можно производить их фильтрацию или сортировку в соответствие с некоторым критерием значимости.

При мониторинге агенту отладки не нужно поддерживать прямой связи с псевдо-агентами, так как псевдо-агенты могут посылать собранные данные в буфер, где они будут накапливаться вместе с остальными данными.

 Рис. 4. Архитектура отладчика, осуществляющего мониторинг

3.2. Отладочные действия

Отладочные действия при мониторинге можно разделить на следующие категории:

сбор данных;

анализ данных;

профилирование системы;

"посмертный" анализ.

1) Сбор данных

Существует несколько способов сбора данных на целевой машине и передачи их менеджеру:

Передавать данные на инструментальную сторону по мере их поступления.

Этот способ применяется при оперативной отладке.

Передавать данные в случае заполнения буфера.

Обычный способ сбора данных при мониторинге.

Сохранять данные на диске.

Таким способом можно получать данные для последующего анализа (конечно, лучше осуществлять сохранение данных с инструментальной машины, уже получив их).

Сбор данных может осуществляться однократно, циклически или непрерывно. При этом отладчик может совершать следующие действия:

начать сбор данных (указывается способ сбора, буфер или имя файла, период сбора, время между циклами или время завершения);

прервать сбор данных;

запланировать начало и/или конец сбора данных по времени, получению сигнала или произошедшему событию.

Данные могут представлять собой как информацию о задаче (содержимое регистров, стека, и.т.д), так и информацию о системе в целом (протокол произошедших событий, протокол выделения памяти).

Один из способов протоколирования событий заключается в том, что на функции, исполнение которых приводит к некоторому событию, ставится специального вида точка прерывания (eventpoint). Такой подход позволяет пользователю определять собственные события (как это сделано в WindView - Wind River Systems, целевая система VxWorks).

Теперь рассмотрим технологию сбора данных на примере StethoScope (Wind River Systems, целевая система VxWorks). При сборе данных о функции используется механизм вставки исполняемого кода перед и после вызова отлаживаемой функции. Его суть в том, что пользователь может задать функции, вызовы которых будут предварять и завершать исполнение требуемой процедуры. Этот механизм используется и в служебных целях, например при трассировке задачи. Реализовать его можно так: на первую инструкцию отлаживаемой функции ставится точка прерывания, обрабатываемая особым образом, а именно:

ставится точка прерывания на точку возврата из отлаживаемой функции;

передается управление функции, которая должна быть вызвана перед отлаживаемой (если такая определена);

запускается выполнение отлаживаемой функции.

Затем при обработке второй точки прерывания вызывается функция, реализующая некоторый набор завершающих действий.

При сборе информации о динамическом выделении памяти можно использовать такой подход (RTILib - Real-Time Innovations, целевая система VxWorks). Заменить функции выделения и освобождения памяти (malloc, calloc, realloc, free) на соответствующие функции, выполняющие, помимо работы с памятью, некоторые отладочные действия, а именно: маркировку границ выделенного блока и последующий контроль за ее сохранностью (так можно фиксировать выход за границы), установку флага доступа к блоку (для запрещения/разрешения обращения к этому блоку), сбор статистики по использованию памяти, протоколирование информации по выделенным блокам.

2) Анализ данных

Нас интересует следующая классификация видов анализа:

Анализ на инструментальной стороне.

Рассматриваемые средства отладки обеспечивают анализ не только данных, полученных во время текущего сеанса отладки, но и данных, полученных ранее и сохраненных на диске. Кроме того, при обработке данных фиксируется время их получения, а для событий - время, в которое они произошли.

Анализ на целевой стороне.

Некоторые данные требуют анализа на целевой стороне. Например, контроль работы задач с динамически выделенной памятью. В этом случае при обращению к адресу в памяти происходит проверка на принадлежность его к какому-либо блоку и возможность доступа. Чтобы избежать проблем с динамическим выделением памяти и ускорить доступ к ней, StethoScope предоставляет возможность создания пула из некоторого числа буферов одинакового размера. Тогда, используя соответствующие функции доступа к этому пулу, можно осуществлять быстрый захват и освобождение его буферов.

Распределенный анализ.

При передаче данных на инструментальную сторону отладчик может производить их предварительный анализ - фильтрацию. Фильтрация представляет собой отбор данных в соответствии с некоторым заданным шаблоном (фильтром). Например, можно рассматривать события только некоторых определенных типов (переключение контекста, запуск задачи, и.т.д) или сравнивать полученные данные с некоторой маской значений. Фильтрация данных нужна, чтобы уменьшить вмешательство в работу целевой системы, то есть можно разбить отладку на несколько уровней: от минимального (исследование событий одного вида, например, переключение контекстов) до максимального (получение подробной информации о каждом событии в системе и данных о выполняемых задачах). В WindView представлены 3 уровня отладки:

протоколирование переключения контекстов

протоколирование состояний задач

протоколирование статусов системных объектов.

3) Профилирование системы

Под профилированием понимается один из способов мониторинга, позволяющий следить за выполнением некоторого множества задач (или всей системы) и предоставляющий пользователю информацию о том, как конкретная задача использует процессор (включая распределение времени в задаче).

Профилирование заключается в том, что с некоторой частотой производятся выборки данных об активных в этот промежуток времени задачах. При детальном профилировании собираются данные о каждой функции (количество ее вызовов, время выполнения). Приведем описание модуля ScopeProfile, входящего в StethoScope и представляющего собой типичный образец агента профилирования.

Сбор данных запускается либо специальной высокоприоритетной задачей, либо посредством ISR (Interrupt Service Routine). Существует два уровня профилирования: профилирование задачи способом "процедура за процедурой" (procedure-by-procedure) и профилирование системы способом "задача за задачей" (task-by-task). Основные параметры профилирования: частота выборки (sample rate) - частота, с которой производится сбор данных; частота анализа (analysis rate) - частота, с которой производится анализ имеющихся в буфере выборок. Анализирующая процедура представляет собой низкоприоритетную задачу, которая подсчитывает среднее из значений для задач и функций из всех выборок буфера и значений, полученных в ходе предыдущего анализа.

Цель профилирования задачи - выявить наиболее часто исполняемые блоки для последующей их оптимизации. Для этого служит способ "процедура за процедурой", позволяющий получать информацию о каждой вызываемой в процессе профилирования функции. Эта информация состоит из среднего времени, которое функция выполнялась, с учетом и без учета времени вызова из нее других функций. Также подсчитывается число ее вызовов. В результате анализа стека строится так называемое дерево выполнения (execution tree), показывающее маршрут вызова каждой функции.

Способ "задача за задачей" служит для сбора информации об активности системы в целом, а именно, какое процессорное время использует каждая задача.

4) "Посмертный" (post-mortem) анализ

Подобный анализ производится в том случае, если в работе системы произошел сбой. Тогда пользователя интересуют не все события, а только те, которые произошли за некоторое время до "обвала" системы. В этом случае, чтобы влияние отладчика на систему было сведено к минимуму, все данные сохраняются в некотором буфере и обновляются по мере его заполнения, но не пересылаются на инструментальную сторону. Адрес для буфера надо выбирать так, чтобы при перезагрузке системы его содержимое не уничтожалось. В VxWorks это можно реализовать следующим образом: надо изменить функцию sysMemTop(), определяющую верхний адрес локальной памяти, так, чтобы она возвращала значение, меньшее действительного адреса. Тогда в образовавшемся адресном пространстве, недоступном системе, можно расположить буфер данных. В результате после перезагрузки в распоряжение отладчика поступают данные о последних событиях, произошедших в системе и, возможно, повлекших ее сбой.

3.3. Пользовательский интерфейс

При анализе данных или профилировании важную роль играет представление полученной информации. Как и в случае активной отладки пользовательский интерфейс делится на три составляющих: графический интерфейс, режим командной строки и команды представления данных.

Графическое представление данных - наиболее важная часть пользовательского интерфейса при мониторинге. Поскольку, как правило, анализируется взаимодействие задач в системе, то нужна визуализация всех событий и задач системы. В WindView для этого служит специальное окно View Graph. По горизонтали откладывается время (единичный интервал времени может меняться), по вертикали приведен список всех задач в системе и уровни прерываний. В такой системе координат легко увидеть, какая задача в какой момент времени в каком состоянии находилась. Особыми значками отмечаются происходящие события, подробности о которых (также как и о задачах) можно увидеть в другом окне.

При мониторинге текущего состояния системы удобно пользоваться графическим интерфейсом, но при анализе сохраненных ранее данных, а также при "посмертной" отладке, можно использовать режим командной строки. Требования к нему предъявляются аналогичные тем, что были у средств активной отладки.

Помимо описанных в предыдущей главе команд представления данных у активных отладчиков средства мониторинга могут располагать также такими командами:

исполнение некоторой последовательности действий при произошедшем событии, поступившем сигнале или наступлении определенного момента времени;

определение пользователем собственных событий (eventpoints в WindView) и сигналов (комбинации известных сигналов "derived signals" в StethoScope).

3.4. Интеграция со средствами разработки ПО

При мониторинге особое внимание уделяется работе псевдоагентов, которые закладываются в код программы на этапе компиляции. Для их успешной работы по сбору необходимой информации требуется наличие на целевой машине ряда функций, вызываемых псевдо-агентами. Эти функции могут быть собраны в одну библиотеку, так называемую библиотеку доступа (access library).

В [20] описывается средство работы с псевдо-агентами - "Alamo monitor". На Рис.5 приведена его архитектура.

 Рис. 5. Alamo monitor

Координатор мониторинга посылает запросы псевдо-агенту и производит фильтрацию полученной информации.

 Рис. 6. Получение информации от псевдо-агента

В зависимости от возможностей, предоставляемых библиотекой доступа, изменяется и роль псевдо-агентов. Возможна ситуация, когда на целевой стороне присутствует только агент доставки данных из буфера, а все данные поставляются псевдо-агентами. Такой подход уменьшает воздействие отладчика на систему, так как все отладочные действия заложены при компиляции, и агент отладки только передает данные менеджеру по мере заполнения буфера, то есть не влияет на ход выполнения отлаживаемых задач.

Имеется и другое применение псевдо-агентов. При помощи встраивания в код программы некоторого некорректно работающего блока, можно моделировать критические ситуации и анализировать поведение задачи или всей системы в таких ситуациях.

4. Особенности отладки многоплатформных распределенных систем

4.1. Особенности архитектуры

Если раньше система реального времени рассматривалась нами как один процесс (с точки зрения ресурсов), то распределенные СРВ представляют уже набор взаимодействующих процессов.

Специфика заключается в том, что отлаживаемое приложение может быть распределено на нескольких платформах с разными процессорами, поэтому эффективность отладчика зависит от:

наличия на каждом компоненте системы агента отладки;

способности менеджера работать со всеми процессорами системы;

возможности агентов отладки осуществлять связь между собой и с менеджером.

Кроме того, если система состоит из большого числа компонент, использование двухуровневой архитектуры "менеджер-агент" становится неэффективным, поскольку менеджер не сможет обработать в надлежащее время информацию от агентов. В этом случае целесообразно использовать многоуровневую структуру, в которой компоненты, объединенные по своим функциональным возможностям, представляют некоторые подсистемы, которыми управляют соответствующие менеджеры, являющиеся агентами на следующем уровне предложенной иерархии.

При включении в систему новой платформы необходимо, чтобы менеджер имел возможность осуществлять связь с агентом. Для этого к набору исходных текстов менеджера добавляется файл, содержащий функции взаимодействия с агентом. Здесь возможны следующие варианты:

Файл описания платформы требует компиляции и сборки вместе с остальными исходными текстами.

Существенный недостаток такого подхода в том, что менеджер или какую-либо его часть придется пересобирать.

Файл описания интерпретируется внутренними средствами менеджера.

В [17] описан отладчик Panorama, платформо-зависимые черты, которого вынесены в отдельные файлы: файл описания платформы (агента) и tcl-скрипт, в котором описаны необходимые функции. Таким образом, имея встроенный tcl-интерпретатор, Panorama способен работать с новой платформой без пересборки менеджера. Архитектура этого отладчика приведена на рис.7.

 Рис. 7. Отладчик Panorama

В случае, если один агент обслуживает несколько менеджеров, целесообразно организовать промежуточное звено, в которое вынести все платформо-зависимые черты менеджеров. Такой подход реализован в среде разработки ПО реального времени TORNADO (система VxWorks). Он заключается в том, что на целевой стороне имеется универсальный агент (target agent), осуществляющий связь со средствами разработки ПО посредством целевого сервера (target server). В этом случае, во-первых, все клиенты работают с одним сервером (и, соответственно, с одним агентом) и, во-вторых, они имеют возможность обмениваться данными между собой, используя целевой сервер.

4.2. Некоторые подходы к отладке распределенных приложений

При отладке распределенного приложения в целом нужно представлять общее его состояние, которое включает структуры данных, распределенные по нескольким платформам. Кроме того, необходимо иметь протокол взаимодействия задач в системе.

Взаимодействие задач, исполняемых на разных процессорах, можно протоколировать, используя вместо стандартных функции связи, передающие необходимую информацию менеджеру. Чем более полной является эта информация, тем проще менеджеру с ней работать, но тем большее влияние на работу системы оказывает сеанс отладки, в результате чего могут возникать новые динамические ошибки. В [9] описана система DARTS (Debug Assistant for Real-Time Systems). С ее помощью можно проводить полноценный сеанс отладки без наличия какой-либо отладочной информации в приложении. Для этого необходимо правильно сопоставить полученный от системы поток событий с исходными текстами приложения. Этот процесс происходит в 2 этапа: разбор исходных текстов и непосредственно само сопоставление.

При разборе исходного текста для каждой задачи генерируется последовательность следующих программных элементов:

системные вызовы;

условные конструкции;

циклы;

вызовы функций, описанных в программе;

библиотечные вызовы.

После трассировки приложения необходима полная информация для полученного потока событий с целью его дальнейшей отладки. Для этого происходит такое сопоставление:

системные вызовы сравниваются по именам и параметрам;

условная конструкция считается обнаруженной в протоколе, если присутствует один из вариантов;

цикл считается найденным, если он присутствует в протоколе 0 и более раз (каждый раз ищется максимальное число его вхождений);

программные вызовы идентифицируются по вхождению в протокол тела подпрограммы;

для каждой библиотечной функции строится набор возможных последовательностей системных вызовов. Функция считается присутствующей в протоколе, если обнаружена некоторая последовательность из ее характеристического набора.

В результате получается набор гипотез о ходе выполнения приложения (включая вызовы функций, время и процессор). Информация может уточняться, если задавать некоторые интервалы выполнения (например, протоколирование выполнения конкретной задачи). После таких уточнений получаем символьную и строковую информацию о произошедших событиях.

Еще один подход к отладке распределенных приложений предложен в [16]. Описанный там отладчик Ariadne позволяет проверять, произошла ли некоторая заданная для конкретной задачи последовательность событий. Механизм проверки осуществлен следующим образом. Сперва создается граф хода выполнения приложения, построенный на протоколе работы приложения. Затем пользователь задает цепи - последовательности событий, которые будут искаться в графе хода выполнения приложения. Следующим шагом является создание p-цепей - это цепи, для которых указаны конкретные задачи, где ожидается возникновение данной последовательности событий. В итоге формируется шаблон поиска - pt-цепи, которые представляют собой логические композиции p-цепей. Если в графе хода выполнения встречается pt-цепь, то считается, что запрос удовлетворен, и возвращается так называемое абстрактное событие - подграф, содержащий встретившиеся экземпляры событий. Эти экземпляры удаляются из графа хода выполнения, и анализ событий продолжается. Если все pt-цепи присутствуют в графе, то тест считается успешно завершенным. Ввиду асинхронности выполнения ошибка может состоять в том, что нарушен порядок возникновения абстрактных событий. Для локализации таких ошибок в Ariadne реализованы следующие соотношения между абстрактными событиями:

A предшествует B, если существует зависимость некоторого события из B от некоторого события из A;

A параллельно B, если события в A и в B независимы;

A перекрывает B, если существует как зависимость события из A от события из B, так и обратная зависимость.

Проверка полученных абстрактных событий на соответствие этим соотношениям позволяет выявлять ошибки, связанные с асинхронностью.

В [26] излагается способ отладки РСРВ посредством моделирования системы сетями Петри с временными ограничениями (timing constraint Petri nets, TCPN). TCPN - это граф ; где P - множество позиций; T - множество переходов; F - множество дуг, соединяющих позиции и переходы; C - множество целочисленных пар (TCmin(pt),TCmax(pt)), где pt может быть и позицией, и переходом; D - множество чисел FIRE(pt), обозначающих время срабатывания pt; и М - множество маркеров.

Говорят, что переход t разрешен, если каждая из входных позиций содержит по крайней мере один маркер. Если к моменту Т0 переход t разрешен, то он может сработать в течении времени от Т0 + ТCmin(t) до T0 + TCmax(t). Переход t сработал успешно, если он продолжался не более FIRE(t) временных единиц, иначе происходит срабатывание других переходов. В случае, когда не срабатывает ни один из переходов, все маркеры остаются на своих местах. Таким образом локализуются ошибки в РСРВ.

Одной из серьезных ошибок, связанных с работой распределенного приложения в системе реального времени, является недетерминированность. Ее суть заключается в том, что при разных запусках приложения при одних и тех же входных данных получаются разные результаты.

В [8] описан подход к обнаружению недетерминированности в системах, использующих в качестве связи между задачами сообщения. В таких системах недетерминированность может быть вызвана либо задержками сообщений, либо сменой алгоритма планирования. Следует отметить, что в приложении может быть специально заложена некая недетерминированность, поэтому нужно такой случай выделять. Предлагается такая стратегия обнаружения ошибочной недетерминированности:

для каждого сообщения определяется некоторый идентификатор;

при получении сообщения идентификатор обрабатывается, и создается некоторая, специфическая для получившей задачи, интерпретация сообщения;

совершается проверка, удовлетворяет ли эта интерпретация некоторому порядку получения сообщений данной задачей. Такой подход позволяет обходить случаи встроенной недетерминированности путем определения одинаковой интерпретации для соответствующих сообщений.

4.3. Способы представления данных

Существуют разные способы представления данных. ([7],[13]). Наиболее распространенный из них - графический. Например, Panorama предоставляет следующие возможности (предполагается, что система использует в качестве механизма связи сообщения):

карта процессоров (процессоры и их соединения, а также текущие сообщения между ними);

окно отладки (состояния задач, данные, и.т.п.);

окно потока сообщений (все сообщения между процессорами во времени (и время приема, и время получения).

В дополнение к графическому способу можно использовать, например, звуковой, как это описано в [14]. Преимущество использования звука при отладке состоит в том, что при большом объеме собранных данных может быть сложно обнаружить ошибку визуально. Например, если в процессе работы затерялось какое-то сообщение от одного процессора к другому, то, анализируя графическое представление взаимодействия процессоров, трудно найти потерянное сообщение, так как оно может быть графически просто не представлено. Однако, если каждое сообщение (от посылки до получения) будет сопровождать некоторый звуковой сигнал, то потерянное сообщение будет сразу обнаруживаться.

Как видно, при отладке распределенных приложений необходимо учитывать связь между процессорами и, в основном, асинхронный ее характер, то есть на первое место выступает обнаружение ошибок, связанных со взаимодействием задач и усиленных тем, что задачи выполняются на разных процессорах.

5. Заключение

Мы рассмотрели основные черты средств активной отладки и мониторинга для распределенных систем реального времени, а также ошибки, характерные для РСРВ, и способы их обнаружения.

Существующие активные отладчики играют важную роль в разработке ПО при поиске логических ошибок, предоставляя широкий набор средств, включающих поддержку исходных текстов, трассировку выполнения приложения, динамическую модификацию памяти, и.т.д. Однако они не позволяют выявлять специфические ошибки СРВ.

Описанные в работе средства мониторинга имеют схожие методы сбора, обработки и представления отладочных данных. Благодаря им становится возможным локализовывать ошибки СРВ, связанные с планированием и синхронизацией.

В случае РСРВ пока не существует общего подхода к отладке, поскольку ошибки, обусловленные связью, имеют разное проявление. В работе приведено описание некоторых способов отладки, позволяющих выявлять такие ошибки. Разработка универсальных методов отладки РСРВ - тема дальнейшего исследования.

**Список литературы**

C.D.Locke, "Fundamentals of Real-Time", Lockhead Martin, 1998

"Realtime CORBA", White Paper -Issue 1.0, 1996/Dec

"It's all a question of time...", Real-time magazine, 1997/4th Quarter

R.O'Farrel, "Choosing a cross-debugging methodology", Embedded systems programming, 1997/Aug

K.Clohessy, "Using object-oriented programming tools to build real-time embedded systems", Real-time engineering, 1996/Fall

V.Encontre, "How to use modeling to implement verifiable, scalable, and efficient real-time application programs", Real-time engineering, 1997/Fall

N.Osawa, H.Morita, T.Yuba, "Animation for perfomance debugging of parallel computing systems", ACM, 1997

S.K.Damodaran-Kamal, J.M.Francioni, "Nondeterminacy: testing and debugging in message passing parallel programs", ACM, 1993

M.Timmerman, F.Gielen, P.Lambrix, "High level tools for the debugging of real-time multiprocessor systems", ACM, 1993

A.von Mayrhauser, A.M.Vans, "Program understanding behavior during debugging of large scale software", ACM, 1997

J.Lang, O.B.Stewart, "A study of the applicability of existing exception-handling techniques to component-based real-time software technology", ACM, 1998

P.Fritzson, T.Gyimothy, M.Kamkar, N.Shahmehri, "Generalized algorithmic debugging and testing", ACM, 1991

D.Zernik, L.Rudolph, "Animating work and time for debugging parallel programs. Foundation and experience", ACM, 1991

J.M.Francioni, L.Albright, J.A.Jackson, "Debugging parallel programs using sound", ACM, 1991

T.Kunz, "Process clustering for distributed debugging", ACM, 1993

J.Cuny, G.Forman, A.Hough, J.Kundu, C.Lin, L.Snyder, D.Stemple, "The Ariadne debugger: scalable application of event-based abstraction", ACM, 1993

J.May, F.Berman, "Panorama: a portable, extensible parallel debugger", ACM, 1993

В.В.Липаев, Е.Н.Филинов, "Мобильность программ и данных в открытых информационных системах.", РФФИ, 1997

A.J.Offutt, J.H.Hayes "A semantic model of program faults", ACM, 1996

C.Jeffery, W.Zhou, K.Templer, M.Brazell "A lightweight architecture for program execution monitoring", ACM SIGPLAN 1998/july

L.Mittag "Multitasking design and implementation", Embedded system programming, 1998/march

E.Ryherd "Software debugging on a single-chip system", Embedded system programming, 1998/march

N.Cravotta "Real-time operating systems", Embedded system programming, 1997/march

J.E.Stroemme "Integrated testing and debugging of concurrent software systems", the sixth IFIP/ICCC conference on information network and data communication, Trondheim, Norway, 1996/june

J.Ready "Distributed applications bend RTOS rules", CMP Media Inc., 1996

J.Tsai, Y.Bi, S.Yang, R.Smith "Distributed real-time systems. Monitoring, visualization, debugging, and analysis", Wiley-Interscience Publication, 1996

В.Б.Бетелин, В.А.Галатенко "ЭСКОРТ - инструментальная среда программирования.", Юбилейный сборник трудов институтов Отделения информатики РАН. Том. II. Москва, 1993.