DOI: 10.15514/ISPRAS-2019-31(4)-1

Средства трассировки ОС РВ семейства «Багет»

А.Н. Годунов, ORCID: 0000-0001-5952-9185 <nkag@niisi.ras.ru>
Ф.Н. Чемерев, ORCID: 0000-0002-6494-716X <nkfedor@niisi.ras.ru>
ФГУ ФНЦ Научно-исследовательский институт системных исследований РАН,
117218, Россия, г. Москва, Нахимовский пр., д. 36, к. 1

Аннотация. Статья посвящена проблемам построения средств трассировки систем жесткого реального времени. В настоящее время практически в каждой операционной системе реального времени (ОС РВ) имеются программные средства трассировки событий. Их задача состоит в поиске «обычных» программных ошибок (с которыми не справляются традиционные отладчики) и ошибок реального времени. При этом приходится анализировать не только последовательности событий, но и «утечки» памяти, динамику состояний процессора и потоков управления (профилирование), состояний семафоров, мьютексов и других средств синхронизации, а также очереди потоков управления, ожидающих освобождения необходимых им ресурсов. Рассматривается методология проектирования программ просмотра и анализа журналов событий (трасс), сформированных приложениями реального времени. Рассмотрены особенности отображения (в том числе, в виде деревьев) журналов событий и временных диаграмм состояний объектов анализируемой системы, представленных наборами данных, содержащими большое количеством записей. Предложено формальное описание моделей данных трассировки, механизмов их визуализации и механизмов управления запросами к записям трассы и состояниям объектов. Эффективность указанных моделей и механизмов подтверждена опытом эксплуатации программы просмотра и анализа протоколов событий ОС РВ семейства «Багет», реализованной с помощью библиотеки графических элементов GTK+.

Ключевые слова: ОСРВ; операционная система реального времени; средства трассировки; журнал событий; профилирование; Model/View/Controller; индексация записей.

Для цитирования: Годунов А.Н., Чемерев Ф.Н. Средства трассировки ОС РВ семейства «Багет». Труды ИСП РАН, том 31, вып. 4, 2019 г., стр. 7-28. DOI: 10.15514/ISPRAS-2019-31(4)-1

Благодарности: Публикация выполнена в рамках государственного задания по проведению фундаментальных научных исследований по теме «Исследование и реализация программной платформы для перспективных многоядерных процессоров» (№ 0065-2019-0002).

Tracing Tools for «Baget» Family RTOS

A.N. Godunov, ORCID: 0000-0001-5952-9185 <nkag@niisi.ras.ru>
F.N. Chemerev, ORCID: 0000-0002-6494-716X <nkfedor@niisi.ras.ru>
Scientific Research Institute for System Analysis of the Russian Academy of Sciences,
GSP-1, 36/1, Nakhimovsky pr., Moscow, 117218, Russia

Abstract. The paper deals with the problems of developing tracing software for hard real-time systems. Currently, almost every real-time operating system (RV OS) has event tracking software. The goal of this software is to search for «ordinary» software errors (which traditional debuggers cannot handle) and real-time errors. In this case, it is necessary to analyze not only the sequence of events, but also the «memory leak», the dynamics of the processor states and control flows (profiling), the states of semaphores, mutexes, and other synchronization tools, as well as the queue of control flows waiting to release the resources they need. The methodology for designing programs for viewing and analyzing event logs (traces) generated by RTOS-based software systems is regarded. Specifics of visualizing RTOS events and time diagrams of states of objects in the analyzed systems, represented by data sets containing a large number of records are discussed. A formal specification is proposed to the tracing data models, the methods for their visualization and for filter management of trace records and object states. The effectiveness of these models and methods is confirmed by the operating experience of the Tool for Viewing and Analyzing the Event Logs for RTOS for «Baget» family developed with the toolkit GTK+ for creating graphical user interfaces.

Keywords: RTOS; events; logging; run-time behavior; tracing tool; Model/View/Controller; MVC

For citation: Godunov A.N., Chemerev F.N. Tracing Tools for «Baget» Family RTOS. Trudy ISP RAN/Proc. ISP RAS, vol. 31, issue 4, 2019, pp. 7-28 (in Russian). DOI: 10.15514/ISPRAS-2019-31(4)-1

Acknowledgements. This work was supported by the State Research Program of the Russian Federation (Project No. 0065-2019-0002).

1. Введение

В настоящее время практически в каждой операционной системе реального времени (ОС РВ) имеются программные средства трассировки событий. Под трассировкой принято понимать порождение, накопление и анализ данных о событиях, имевших место при выполнении пользовательского приложения [1]. В 2001 году в стандарт POSIX 1003.1 внесены дополнения, специфицирующие интерфейс прикладной программы со средствами трассировки [1].

Задача трассировки — поиск «обычных» программных ошибок (с которыми не справляются традиционные отладчики) и ошибок реального времени. При этом приходится анализировать не только последовательности событий, но и «утечки» памяти, динамику изменения состояний процессора и объектов ОС – потоков управления, семафоров, мьютексов и других средств синхронизации, а также очереди потоков, ожидающих освобождения необходимых им ресурсов.

Средства трассировки должны предусматривать возможность представления информации о зарегистрированных событиях в виде, приемлемом для непосредственного восприятия человеком – в виде текста, графиков, диаграмм и т.п. В зависимости от задач, возложенных на средства трассировки, результаты визуализации событий могут выводиться в обычный текстовый файл, но могут быть представлены содержимым окон в приложениях, реализующих графический пользовательский интерфейс (GUI).

1.1 Средства трассировки в существующих ОС РВ

В настоящее время большинство операционных систем оснащено собственными инструментами трассировки. Имеется также положительный опыт создания единого инструментария для визуализации трасс, сформированных под управлением различных ОС (*Tracealyzer* фирмы *Percepio* [3]).

События, связанные с вызовами ядра, работой планировщика, обработкой прерываний, созданием, уничтожением и изменением состояний потоков и процессов, а также события прикладной программы может генерировать большинство используемых в настоящее время ОС РВ, например, *QNX Neutrino* [4], *INTEGRITY-178* [5], *ThreadX* [6] , *RTOS-32* [7], *VxWorks* [8] и др. ОС РВ *ThreadX* допускает генерацию и последующую обработку событий промежуточного программного обеспечения из стеков *NetX*, *FileX* и *UsbX*.

В ходе регистрации содержимое буферов, в которых накапливаются результаты генерации событий, направляется на устройство вывода (например, в файл инструментальной ЭВМ). При этом на данные трассировки могут быть наложены фильтры – как при генерации, так и в процессе регистрации. Управление регистрацией данных трассировки может обеспечиваться как непосредственно средствами соответствующей ОС РВ (такими, например, как *tracelogger* в *QNX*, средства ведения журнала аудита в ОС РВ *INTEGRITY-178*, средства трассировки, включенные в разделы (*partition*), в ОС РВ *PikeOS* [9]), так и с помощью функций, доступных прикладной программе (таких, например, как *TraceEvent*() в

QNX). В первом случае разработчики ОС РВ предоставляют средства конфигурирования (обычно на основе *GUI*). В *QNX*, например, при конфигурировании инструментированного ядра обычно задаются режимы регистрации, размер буфера, а также фильтры, указывающие, какие события и классы событий должны регистрироваться. При конфигурировании ОС РВ *PikeOS* записи трассы могут быть, кроме того, расширены и использованы в качестве триггеров.

Наличие в различных ОС РВ функций управления регистрацией данных трассировки является предпосылкой создания единого инструментария для визуализации трасс, сформированных под управлением этих ОС. Таким инструментом является *Tracealyzer* фирмы *Percepio* [3]. Он либо использует данные, полученные регистратором событий трассировки, входящим в соответствующую библиотеку регистратора трасс ОС РВ (*VxWorks*, *µC/OS-III*, *FreeRTOS*, *ThreadX*), либо экспортирует данные трассировки, как в случае *On Time RTOS-32*, – с помощью инструмента *Trace Exporter*, который поставляется в исходном коде на *C* и должен быть включен в приложение реального времени.

Собранные данные могут быть обработаны как в режиме реального времени (*online*-анализ), так и в автономном режиме (*offline*-анализ). В качестве инструмента обработки чаще всего предлагаются инструменты, использующие графический пользовательский интерфейс (*GUI*), как автономные, например, система мониторинга *CODEO* данных трассировки ОС РВ *PikeOS* [9] или *Tracealyzer* фирмы *Percepio* [3], так и включенные в интегрированную среду разработки (*IDE*) приложений реального времени, как, например, в *QNX*. Графический инструмент для анализа, проверки и профилирования приложений реального времени *OpenTracer* фирмы *Altreonic* [10] интегрирован со средой разработки *OpenComRTOS Designer*. Фирмой *Percepio* поддерживаются несколько ведущих *IDE*, в которые интегрирован *Tracealyzer*. Среди них – *IAR Embedded Workbench*, *Keil µVision (MDK)*, *Atmel Studio*, *Microchip MPLAB X IDE*, *Wind River Workbench* и большинство *IDE* на основе *Eclipse* (например, *Atollic TrueStudio*, *SW4STM32*, *Code Composer Studio (TI CCS)*, *NXP LPCxpresso/MCUxpresso* и т.д. [3]).

Наряду с интерактивными средствами, использующими *GUI*, некоторые разработчики ОС РВ предлагают инструменты на основе программного интерфейса приложения (*API*). В *QNX* [4], например, имеется набор функций, в том числе, *traceparcer*() и *traceprinter*(), с помощью которых пользователь может реализовать собственные процедуры обработки данных трассировки.

1.2 Обзор средств трассировки в ОС РВ семейства «Багет»

Положения стандарта POSIX 1003.1 учтены при разработке операционных систем семейства «Багет», ориентированных на использование в целевых (встраиваемых) ЭВМ, работающих в системах жёсткого реального времени. Предназначенные для функционирования на целевых ЭВМ прикладные программы (приложения реального времени) разрабатываются на инструментальной ЭВМ. Трассы, сформированные средствами трассировки ОС РВ семейства «Багет», просматриваются и анализируются на инструментальной ЭВМ после завершения работы приложения на целевой ЭВМ (*offline*-анализ).

Интерактивным средством просмотра и анализа протоколов событий, сформированных системой протоколирования ОС РВ семейства «Багет», является Трассировщик – программа, реализующая графический пользовательский интерфейс. Существенной особенностью программы является то, что основным средством отображения данных трассировки, в том числе объединенных в иерархическую структуру, являются таблицы, которые могут содержать большое количество строк.

Важным показателем эффективности Трассировщика, является время реакции на действия пользователя при работе с программой:

- время открытия (первичной обработки) трассы – от момента выбора файла протокола событий до появления на экране содержимого трассы, которую пользователь может просматривать и анализировать;
- время вторичной обработки трассы (при этом формируются трассы состояний – агрегаты данных, отражающие динамику изменения состояния процессора и объектов ОС);
- время, необходимое для перерисовки содержимого отображаемых больших таблиц при их прокрутке (скроллинге);
- время, затрачиваемое на формулирование условий отбора записей трассы, состояний процессора и объектов трассировки;
- время выполнения процедур отбора записей протокола событий и трасс состояний.

Снижение затрат времени предполагает отказ от тотальной обработки трассы. Высокую скорость отбора записей трассы и доступа к ним обеспечивает специальная модель трассы. При открытии трассы обрабатываются лишь те данные, которые необходимы для построения этой модели – при этом идентифицируются объекты ОС, а записи трассы индексируются. Остальные данные Трассировщик обрабатывает по мере необходимости – например, когда требуется отобразить их на экране. Минимизация затрат времени на формулирование условий отбора обеспечивается предельно лаконичным механизмом задания этих условий в наиболее важных типичных случаях.

В данной статье рассматриваются подходы и технические решения, лежащие в основе моделей, используемых средствами трассировки ОС РВ семейства «Багет».

2. Основные понятия

2.1 События трассировки и режимы протоколирования

Сведения о действиях, производимых при выполнении приложения, фиксируются в виде агрегатов данных, называемых событиями трассировки. События записываются в потоки трассировки, которые содержат также служебные данные, необходимые для интерпретации событий. Для получения копии потока для последующего анализа, содержимое буфера сбрасывается в журнал трассировки (протокол событий, трасса) – файл, располагающийся в долговременной памяти.

Одним из главных требований к системе трассировки является минимизация накладных расходов. Чтобы удовлетворить этому требованию, потоки трассировки размещают в оперативной памяти – при этом, как правило, используется кольцевой буфер. Перенос данных из буфера в файл может производиться автоматически по мере заполнения буфера, когда процессор не занят выполнением «штатных» функций приложения реального времени, или по явному требованию.

Действия при переполнении буфера определяются настройками системы протоколирования. В частности, возможна остановка при переполнении буфера. Имеется также возможность циклической записи, когда при достижении конца буфера запись продолжается в начало буфера – новые записи замещают старые. В последнем случае, копия содержимого буфера (самые «свежие» записи) создается при закрытии потока трассировки.

В процессе выполнения прикладной программы могут иметь место фатальные ошибки, в результате которых система окажется неработоспособной. В подобных случаях полезным источником информации является содержимое буфера на момент краха системы. В ОС РВ семейства «Багет» имеется возможность создать копию его содержимого при последующем запуске приложения.

2.2 Системные и пользовательские события

В соответствии со стандартом POSIX 1003.1 события трассировки подразделяются на две категории:

- системные – генерируются в ответ на действия ОС, в том числе, при обращении прикладной программы к функциям ОС;
- пользовательские (они же события прикладной программы) – генерируются при вызове функции $posix\_trace\_event()$.

Пользовательские события могут использоваться как маркеры наиболее важных участков прикладного кода – их можно ассоциировать с точками входа в функции и отдельными ветвями реализующих эти функции алгоритмов. События прикладной программы могут генерироваться средствами ее самоконтроля [11]. Наконец, с пользовательскими событиями в трассе можно связать реальные события, произошедшие на объекте управления: срабатывания реле, закрытие и открытие клапанов, факт выхода значений критически важных параметров (температура, давление) за пределы допуска.

2.3 Служебные записи трассы

Наряду с событиями трасса может содержать служебные записи. Если трасса имеет блочную структуру (как в ОС РВ семейства «Багет»), то служебной может быть, например, запись, содержащая номер блока, или «пустая» запись, заполняющая его «хвост». Некоторые служебные записи (назовем их существенными служебными) содержат важную информацию, необходимую для интерпретации событий трассировки.

Заголовок трассы, например, содержит сведения о версии ОС, под управлением которой трасса была сгенерирована. Другие служебные записи содержат характеристики объектов ОС (семафоров, мьютексов, потоков управления, очередей сообщений и т.п.).

Источником этих данных являются события, регистрируемые в момент создания этих объектов. Система протоколирования ОС РВ семейства «Багет» дублирует эти характеристики, помещая их в служебные блоки системы протоколирования. В отличие от содержимого циклического буфера, служебные блоки не затираются. В момент завершения протоколирования они вместе с записью представления (заголовком трассы) отписываются в файл трассы.

2.4 Типы записей трассы

Каждая запись трассы относится к некоторому типу t, принадлежащему множеству T_r типов записей ($t \in T_r$). Их номенклатура, т.е. множество T_r определяется версией ОС РВ. Типы событий и типы существенных служебных записей составляют множество T_e типов существенных записей ($T_e \subset T_r$). Номенклатура типов событий и служебных записей определяется системой протоколирования конкретной версии ОС.

Каждый тип записи характеризуется уникальными именем и идентификатором. Имена используются при визуализации записей, тогда как в потоке трассировки хранятся числовые идентификаторы их типов. Соответствие имен и идентификаторов типов обеспечиваются средствами просмотра и анализа трассы

2.5 Объекты трассировки

В записях трассы, соответствующих системным событиям, могут присутствовать поля, значения которых идентифицируют объекты ОС (потоки управления, семафоры, мьютексы и т.п.). Поля записей, соответствующих пользовательским событиям, могут содержать тексты или целые числа, используемые в качестве идентификаторов пользовательских объектов.

В модели данных Трассировщика номенклатура типов объектов трассировки наряду с типами объектов ОС включает в себя прерывания и сигналы, которые в совокупности составляют

множество типов объектов трассировки $T_{(obj)}$. В целом эта номенклатура отражает наиболее общие представления об операционных системах семейства «Багет». Целочисленные идентификаторы i типов объектов трассировки могут быть представлены набором констант, задаваемых перечисляемым типом (в языке C – $enum$).

Два типа объектов «прерывание» и «поток управления» составляют множество T_c типов контекстов ($T_c \subset T_{(obj)}$).

Множество объектов трассировки типа $t \in T_{(obj)}$ обозначим через O_t, $|O_t|$ – зарегистрированное в трассе количество объектов типа t. Множество всех зарегистрированных объектов трассировки обозначим через $O = \bigcup_{t \in T_{(ob\ )}} O_t$.

2.6 Состояния процессора и объектов ОС

Наряду с событиями, представленными в трассе, приходится анализировать состояния процессора (*CPU*) и объектов ОС.

При анализе загрузки процессора и поведения объектов ОС РВ удобно оперировать состояниями с разной степенью детальности. Состояния, для которых не предусмотрена детализация, будем называть атомарными или просто состояниями. Состояния, допускающие детализацию будем называть агрегированными (укрупненными). Примером агрегированного состояния процессора является состояние «*выполняются потоки (не важно какие)*», детализируемое состоянием «*выполняется конкретный поток*»

Процессор может находиться либо в состоянии «*простой*», либо в одном из состояний, когда выполняются «*обработчики прерываний (не важно какие)*», «*потоки (не важно какие)*», «*конкретный обработчик прерывания*», «*конкретный процесс*», «*конкретный поток*». Эти состояния (как атомарные, так и агрегированные) могут быть представлены множеством:

$$S_{(CPU)} = \{cpuIdle, cpuInterrupts, cpuThreads, cpuIntr, cpuProc, cpuThr\}$$

Множество состояний объектов типа t обозначим через S_t. Например, поток управления может пребывать в следующих состояниях: «*не исполняется*», «*исполняется*», «*не готов*», «*готов*», «*прерван*». Соответственно множество состояний потока управления (атомарных и агрегированных) может быть представлено следующим образом:

$$S_{(Thread)} = \{thrNotRun, thrRun, thrNotReady, thrReady, thrInterrupted\}.$$

2.7 Отображение данных, имеющих иерархическую структуру

В основе наиболее популярных современных средств отображения данных, имеющих иерархическую структуру, лежит концепция *MVC* (*Model/View/Controller*) [12]. Реализации этой концепции в разных фреймворках – кроссплатформенных библиотеках графических элементов (таких, например, как *GTK+* [13] и *Qt* [14]), предназначенных для создании пользовательского интерфейса (*GUI*), – могут иметь существенные различия. Однако, обсуждаемые в рамках данной статьи подходы и технические решения не касаются аспектов концепции, связанных с редактированием данных. Важным является лишь то обстоятельство, что в рамках концепции *MVC* модель данных (*Model*) отделена от представления (*View*), а собственно данные (более точно, источник данных – *Data*) отделены от модели (*Model*), которая по отношению к элементам данных выступает в роли навигатора.

В дальнейшем будет показано, что временны́е характеристики используемых Трассировщиком средств отображения данных могут быть существенно улучшены именно благодаря тому, что в основе их реализации лежит концепция *MVC*. Ниже приводится формальное описание наиболее важных с этой точки зрения ее механизмов. (Приведенные в данном разделе функции являются условными обозначениями методов классов, реализующих модели *Model*, представления *View*, или комбинаций этих методов.)

Интересующее нас представление (*View*) отображает данные в виде таблицы. Каждой ее строке соответствуют позиция в дереве, представляющая собой последовательность $P =$

$(p_i)_{i=1}^{|P|}$, ($p_i \in \mathbb{N}$). Элементы p_i этой последовательности задают логические координаты узла дерева – путь от p_1-го элемента верхнего уровня дерева к p_n-му элементу ($n = |P|$), соответствующему уровню $|P|$ дерева. В случае одноуровневого дерева (списка) позиция $P = \{p_1\}$ любой строки определяется ее номером, равным p_1. Логические координаты используются представлением *View*, с одной стороны, для адресации к отображаемым данным, с другой – для позиционирования в пространственных координатах виджета, реализующего представление *View*. В *GTK+*, например, эти координаты представлены структурами типа *GtkTreePath*.

Модель *Model* обеспечивает связь представления *View* с источником данных *Data*. Взаимодействие модели с этими данными обеспечиваются классом, реализующим методы интерфейса, определяемого конкретным фреймворком. В случае *GTK+*, например, этой цели служит абстрактный интерфейс *GtkTreeModel* [13], в случае *Qt* – интерфейс базового класса *QAbstractItemModel* [14]. Эти методы, с одной стороны, должны обеспечивать навигацию внутри источника данных (привязку узла модели *Model* к элементу данных источника *Data*), а с другой – соотнесение узла модели с позицией P в дереве представления *View*.

Узел модели в *GtkTreeModel* описывается структурой типа *GtkTreeIter*, в *QAbstractItemModel* – объектом класса *QModelIndex*. Введем условное обозначение $node$ для описателя узла любой модели *Model* (в том числе, для типа *GtkTreeIter* или *QModelIndex)*. Несмотря на то, что номенклатура методов модели и их сигнатура могут быть разными у различных фреймворков, эти методы обеспечивают:

- определение типа (с помощью функции $type(node)$) и значения ($value(node)$) данного из хранилища *Data*;
- установление соответствия – с помощью функция $node(P)$ – между узлом $node$ модели *Model* и позицией P в дереве представления *View* (тем самым реализуется доступ методов представления к данным из хранилища *Data*), а также обратного соответствия – с помощью функция $P(node)$;
- навигацию по узлам модели *Model* (с помощью функций $next(node)$, $prev(node)$, $parent(node)$, $nth\_child(node,\ n)$);
- доступ к характеристикам узла, таким, например, как количество дочерних узлов.

Столбцы таблицы, реализуемой представлением *View*, могут быть представлены в виде последовательности c_i описателей столбцов $C = (c_i)_{i=1}^{|C|}$, где $|C|$ – их количество. С каждым столбцом c_i связан объект $render_i$, отображающий содержимое ячейки из этого столбца с помощью низкоуровневых графических средств, и функция $paint(render_i, c_i, node)$, переопределяющая значения внутренних параметров объекта $render_i$, используемых им при «отрисовке» ячейки (таких как текст, цвет шрифта и т.п.) – исходя из интерпретации. Реализация функции $paint()$ возлагается на разработчика *GUI*-приложения. Ему же предоставляется выбор класса, которому принадлежит объект $render_i$, из некоторого фиксированного набора. Связь между логическими координатами P строки, и узлом $node$ модели *Model*, необходимые для позиционирования области рисования, предоставляются соответствующими методам *Model*.

3. Модель трассы событий

Современные ЭВМ, используемые в качестве инструментальных при просмотре и анализе протоколов событий, обладают достаточным объемом оперативной памяти и механизмом виртуальной памяти, что позволяет помещать протоколы событий в виртуальную память инструментальной ЭВМ целиком. В дальнейшем, если специально не оговаривается обратное, считается, что все описываемые крупные агрегаты данных (трассы, индексные массивы) размещены в виртуальной памяти. В частности, адреса записей трассы и их полей – это адреса в виртуальной памяти инструментальной ЭВМ.

Трасса представляет собой агрегат данных, моделью которого является последовательность записей $L = (r_i)_{i=1}^{|L|}$, где $|L|$ – количество записей в трассе. В свою очередь, запись трассы r_i – это агрегат данных, которому можно поставить в соответствие его адрес $a_i = Adr(i)$ в оперативной памяти, размер $l_i = Size(i)$ и идентификатор типа записи $t = Eid(i)$, где $t \in T_r$.

3.1 Индексация записей трассы

Эффективная реализация функций $Adr()$, $Size()$ и $Eid()$ предполагает наличие того или иного механизма индексации записей трассы. Решение этой задачи существенно упрощаются, если в начале каждой записи r_i присутствует заголовок (дескриптор), содержащий длину записи и идентификатор ее типа, а сами записи удовлетворяют условию связности трассы:

$$a_{i+1} = a_i + l_i \tag{1}$$

где a_i – адрес i-й записи, l_i – ее размер.

В ОС РВ семейства «Багет» дескриптор записи представлен двумя полями *type_record* и *size_record*, что позволяет – в процессе создания новых версий ОС – в широких пределах варьировать как номенклатуру типов записи, так и их размер.

Первичным индексом записи r_i трассы будем называть ее порядковый номер i. Для быстрого поиска записи по ее первичному индексу используется индексный массив I_L. С учетом условия связности (1) индексация (назовем ее первичной) сводится к построению индексного массива беззнаковых целых чисел, реализующего последовательность $I_L = (\Delta_i)_{i=1}^{|L|}$, где $\Delta_i = a_i - a_1$ – смещение записи r_i относительно начала трассы L.

Концепция Трассировщика предполагает, что размер трассы не превышает 4 ГБ. Поэтому использование 32-разрядных смещений Δ_i для индексации записей позволяет в условиях 64-разрядной инструментальной ОС в два раза уменьшить размер индексного массива по сравнению с непосредственным использованием адресов.

Рассмотрим последовательность E, полученную из последовательности L путем исключения из нее несущественных служебных записей. Каждому номеру j элемента последовательности E соответствует первичный индекс i, такой что $\overline{r}_j = r_i$, $r_i \in L$.

3.2 Описание структуры записи трассы на языке *C*

В ОС РВ семейства «Багет» записи трассы формируются с помощью функций, реализованных на языке *C*. Поэтому структурные типы этого языка программирования являются наиболее органичным средством описания записей, поля которых могут быть как простыми (описываемыми типами *short*, *unsigned int* и т.п.), так и составными: структурами (*struct*) и объединениями (*union*). В записях трассы ОС РВ семейства «Багет» агрегату данных типа *union* предшествует простое поле, значение которого определяет поле объединения *union*, в соответствии с которым следует интерпретировать содержимое этого агрегата данных.

Любое поле f (простое или составное) любой существенной записи имеет тип, заданный в соответствии со стандартом языка *C*. Все типы полей, встречающиеся в описаниях существенных записей всех типов образуют множество T_d, определяемое версией ОС РВ.

В концепции Трассировщика выделяется множество базовых типов T_b – единое для всех трасс, вне зависимости от того, какой именно версией ОС РВ семейства «Багет» трасса была получена. Это множество включает в себя стандартные типы данных языка *C*, а также некоторые структурные типы данных (например, *struct timespec* стандарта POSIX 1003.1). В последних версиях ОС РВ семейства «Багет» полям записи, содержащим идентификаторы объектов ОС, назначаются типы, имена которых однозначно определяют тип объекта ОС (например, «*thread_ptr*», «*chan_id*»). Эти типы (и им подобные) также являются элементами

множества базовых типов T_b. Множество базовых типов, применимых к отдельно взятой версии ОС РВ, является подмножеством множества T_b.

Для каждого базового типа $b \in T_b$ создается набор функций, используемых при интерпретации и визуализации значений полей записей трассы, принадлежащих этому типу. Указатели на эти функции – вместе с данными о размере поля типа b и способе его выравнивания в памяти ЭВМ находятся в описателе базового типа b. В Трассировщике поддерживается механизм, устанавливающий соответствие между именами базовых типов и их описателями.

Структурный тип данных – элемент множества T_d – характеризуется именем типа. В языке C с этим именем связана либо структура (*struct*), либо объединение (*union*).

Описания записей и описания их полей содержатся в заголовочных файлах (*h*-файлах) операционной системы. Каждое *C*-описание структуры записи типа e содержит исчерпывающую информацию, необходимую для получения значения любого поля f простого типа любой записи трассы.

Если программа просмотра и анализа трасс в своей существенной части написана на языке *C*, в принципе возможно непосредственное использование заголовочных файлов в ходе анализа. В этом случае необходимые *h*-файлы ОС РВ могут быть использованы в процессе компиляции программы просмотра как ее собственные заголовочные файлы. Но, во-первых, такая программа просмотра не будет адекватно работать с трассами, созданными другими версиями ОС РВ, *h*-файлы которых отличаются от тех, что были использованы в проекте программы просмотра. А во-вторых, даже в рамках одной и той же версии ОС РВ в заголовочных файлах присутствуют зависимости от архитектуры процессора, используемого на целевой машине. Это означает, что для каждой версии ОС и для каждой архитектуры придется собирать отдельную программу просмотра (или, по крайней мере, существенную ее часть, ответственную за интерпретацию записей трассы).

В Трассировщике ОС РВ семейства «Багет» реализован иной подход, в рамках которого описание записей и их полей представлено в виде файла на языке *XML*, формируемого на основании *C*-описаний. При этом *XML*-описание трассы дополняется сведениями, отсутствующими в *h*-файлах ОС РВ, например, спецификациями форматов, используемых при отображении записей трассы.

3.3 Описание структуры записи трассы на языке *XML*

В *XML*-описании трассы множество T_r типов записей представлено последовательностью *XML*-элементов $\langle event \rangle$. Атрибут $Name$ элемента $\langle event \rangle$ определяет имя записи, атрибут id – значение ее идентификатора. Каждый элемент $\langle event \rangle$ состоит из элементов $\langle field \rangle$ с атрибутами $Name$, dim и $type\_name$. Порядок элементов $\langle field \rangle$ совпадает с порядком элементов *C*-описания структуры. Имя элемента в *C*-структуре соответствует значению атрибута $Name$ элемента $\langle field \rangle$, размерность – значению атрибута dim, тип данных – значению атрибута $type\_name$.

Множество структурных (составных) типов данных $(T_d \setminus T_b)$ представлено в описании трассы *XML*-элементом $\langle Common \rangle$, содержанием которого является *XSD*-последовательность (*xs:sequence*) *XML*-элементов $\langle dt \rangle$, различающихся значениями атрибута $Name$.

Совпадение значения атрибута $type\_name$ с именем базового типа $b \in T_b$ свидетельствует о том, что поле, описываемое элементом $\langle field \rangle$ относится к базовому типу b. В противном случае этому полю соответствует *XML*-элемент $\langle dt \rangle$ с атрибутом $Name$, значение которого равно значению атрибута $type\_name$ элемента $\langle field \rangle$. Структура элемента $\langle dt \rangle$, также как и структура элемента $\langle event \rangle$, состоит из *XML*-элементов $\langle field \rangle$ с атрибутами $Name$, dim и $type\_name$.

XML-описания типов записей трассы и составных типов данных является частью общего описания трассы, которое для Трассировщика является основным источником исходных данных, необходимых для интерпретации трассы. Каждой версии ОС РВ соответствует определенный *XML*-файл.

В настоящее время используется простой и надежный способ генерации *XML*-описаний трасс, не предполагающий лексического разбора заголовочных файлов ОС РВ. Программа, использующая кросс-отладчик *rdp-gdb*, автоматически генерирует *XML*-описание трассы – свое для каждой из поддерживаемых ОС РВ аппаратных платформ целевой ЭВМ. При запуске отладчика, ему в качестве аргумента передается целевой образ – файл, сконфигурированный под определенную процессорную архитектуру целевой ЭВМ.

Основное достоинство этой программы, наряду с ее очевидной простотой и надежностью, – это точность, с которой сгенерированное *XML*-описание отражает особенности процессорной архитектуры.

3.4 Дерево описания записи трассы

Непосредственное использование *XML*-описания трассы сопряжено со значительными затратами времени как в процессе первичной обработки трассы, так и при визуализации ее записей.

В Трассировщике реализована модель записи трассы, источником данных для которой является *XML*-описание трассы. Эта модель применима к трассам, сформированным различными версиями ОС РВ. Она, во-первых, обеспечивает приемлемую скорость обработки трассы, а, во-вторых, непосредственно используется инструментарием *GTK+* при визуализации ее записей. Эта модель формируется в самом начале обработки трассы – после того, как из заголовка трассы (записи представления) извлечена информация о версии ОС РВ и архитектуре процессора.

Любая существенная запись трассы имеет, вообще говоря, иерархическую структуру, описание которой может быть представлено в виде последовательности описателей непосредственно содержащихся в ней полей:

$$R_e = (d_i)_{i=1}^{|R_e|},$$

где e – тип записи, $|R_e|$ – количество полей записи (элементов последовательности R_e), d_i – описатель i-го поля.

Среди описателей полей последовательности R_e может присутствовать один или более описатель d составного (структурного) поля, который, аналогично описанию структуры самой записи R_e, может быть представлен последовательностью описателей полей:

$$d = (\tilde{d}_i)_{i=1}^{|d|},$$

где $|d|$ – количество элементов в последовательности d.

В Трассировщике описатель d является структурой, содержащей, в частности, имя поля и указатели на «соседние» описатели: «*родительский*», «*следующий*», «*предыдущий*», «*первый подчиненный*». Система этих указателей реализует иерархию дерева описания записи типа e, узлами которого являются описатели полей. Описатели d_i – узлы верхнего уровня, объединенные посредством указателей «*следующий*» в список, началом которого является элемент d_1, – определяют последовательность R_e. Таким образом, R_e описывает древовидную структуру записи типа $e \in T_e$.

Дерево описания записи R_e формируется в процессе последовательного обхода *XML*-описания записи e. При обнаружении в *XML*-описании очередного элемента $\langle field \rangle$ создается соответствующий ему описатель d. Этот описатель встраивается в существующую часть дерева R_e (обновляются указатели на «соседей» у самого описателя d и у смежных узлов).

В ходе формирования дерева R_e используются элементы структуры описателя d: dt («*указатель на тип поля*») и $offset$ («*смещение относительно начала записи*»). Если

значение атрибута $type\_name$ элемента $\langle field \rangle$ совпадает с именем базового типа $b \in T_b$, элементу dt структуры присваивается значения указателя на описатель базового типа b. Если значение атрибута $type\_name$ элемента $\langle field \rangle$ совпадает с именем атрибута $Name$ элемента $\langle dt \rangle$, описывающего составное поле, узел d «достраивается» в соответствии с «содержимым» *XML*-элемента $\langle dt \rangle$. Эта процедура рекурсивна, поскольку «содержимое» $\langle dt \rangle$ может включать поля структурных типов. При этом всякий раз для описателя d рассчитывается значение смещения ($offset$) описываемого им поля относительно начала записи – с учетом размеров и способов выравнивания полей базовых типов.

Размер простого поля f в записи $\overline{r}_j \in E$ трассы и способ его выравнивания в оперативной памяти инструментальной ЭВМ определяется его типом b ($b \in T_b$), тогда как адрес – индексом i записи r_i и позицией P (полным путем от корня) в структуре R_e, описывающей запись r_i типа e трассы L. Каждому базовому типу b соответствует функция, возвращающая значение поля f по его адресу: значение v_f задаваемое путем P простого поля f записи r_i, может быть получено с помощью функции $v_f = Value(i, P)$. Таким «значением» может быть, например, строка, число, значение указателя на структуру в адресном пространстве целевой ЭВМ, содержащую данные об объекте ОС. С типом b можно связать также функцию отображения $s = Show(v_f, F)$, преобразующую значение v_f в строку s в соответствии с форматом F.

3.5 Идентификаторы объектов трассировки в записи трассы

Идентификатор oid объекта трассировки, обнаруженный в поле f_P, записи $\overline{r}_i$ типа e, определяется значением поля f_P, позиция которого в записи задана путем P: $oid = Value(i, P)$. Однако, для идентификации объекта этого значения недостаточно: необходимо знание идентификатора процесса, в котором, участвует объект. Как правило, идентификатор pid процесса является значением одного из полей записи $\overline{r}_i$. В общем случае для идентификации объекта o в записи, описываемой деревом R_e, используется пара описателей полей записи $\overline{r}_i$: d_o (описатель поля идентификатора объекта) и d_p (описатель поля идентификатора процесса). В случае «однопроцессных» ОС РВ, а также в случае объектов трассировки, не привязанных к конкретному процессу, используется нулевое значение идентификатора процесса.

Каждому объекту o соответствует свое описание – структура, содержащая, в частности, имя объекта, его тип, и ссылку на служебную запись трассы, содержащую его характеристики. В целом, эту структуру можно рассматривать как программную реализацию объекта o. Тип объекта $t \in T_o$ может быть получен с помощью функции $t = Type(o)$.

В Трассировщике для идентификации объектов трассировки используется хеш-таблица H, из которой объект o типа t может быть извлечен с помощью функции $o = GetObject(H, t, pid, oid)$. В случае отсутствия в таблице объекта с такими характеристиками, описание объекта создается, указатель на него заносится в таблицу H.

Описание R_e записи типа e, может содержать описатели (образующие множество O_e) нескольких полей, значения которых интерпретируются как идентификаторы объектов. Описатель одного из полей может быть объявлен описанием поля идентификатора главного объекта. Выбор этот, в принципе, произволен и определяется лишь тем, насколько актуален отбор записей трассы, в которых фигурирует объект, выбранный в качестве главного. Для события «захват семафора», например, главным объектом естественно считать семафор. Программной реализацией множества O_e в Трассировщике является список *GList* (из библиотеки *GLib*). Описатель поля, содержащего идентификатор объекта, объявленного главным, помещается в начало списка.

3.6 Описание записи трассы как модель данных для отображения в таблице

Используемое в ходе анализа дерево описания записи R_e ориентировано на обработку записей как в рутинном (автоматическом) режиме, так и в интерактивном, когда записи просматриваются в скроллируемых таблицах.

В *GTK+* данные, имеющие иерархическую структуру, отображаются с помощью виджета *GtkTreeView*, реализующего представление *View* концепции *MVC* (см. 2.7) Для взаимодействия с этими данными всегда используется класс, реализующий методы абстрактного интерфейса *GtkTreeModel*.

При заполнении ячеек таблицы, отображаемой виджетом *GtkTreeView*, используются функции обратного вызова с сигнатурой *GtkTreeCellDataFunc* (см. 2.7, функция $paint()$). В зависимости от того, какие именно данные модели требуется отобразить, эти функции переопределяют значения внутренних параметров рендерера – объекта класса *GtkCellRenderer*, аналога объекта $render_i$ (см. 2.7), связанного с i-м столбцом таблицы.

В Трассировщике для отображения детальной информации, содержащейся в записи трассы, реализован класс *DiTree*, дочерний по отношению к классу *GObject* и реализующий методы интерфейса *GtkTreeModel*. Виджет *GtkTreeView* отображает существенную запись $\overline{r}_j$ типа e, воспроизводя структуру дерева описания записи R_e, узлы которого снабжены именами, совпадающими с именами полей в *C*-описании (*h*-файлах ОС РВ). Тип записи определяется первичным индексом i ($e = Eid(i)$), соответствующим номеру j элемента последовательности E существенных записей. В каждом случае, когда требуется отобразить содержимое записи типа e, виджету *GtkTreeView* в качестве модели данных D_e предъявляется объект класса *DiTree*. Единственным атрибутом (свойством) этого объекта является R_e – дерево описания записи типа e.

Привязка модели данных D_e записи типа e к виджету *GtkTreeView* происходит в момент выбора существенной записи $\overline{r}_j$. Функции, реализующие интерфейс *GtkTreeModel*, используют указатели на «смежные» узлы дерева, содержащиеся в структуре описателей полей. Значением в узле дерева R_e, полученным с помощью функции $get\_value()$, является указатель на описатель d этого узла. Значения полей, определяемые адресом записи $a_i = Adr(i)$ и смещениями $offset$ из описателей полей записи, вычисляются в теле функций, задающих параметры рендерера.

Наличие у модели D_e интерфейса *GtkTreeModel*, позволяет инкапсулировать детали ее реализации. Функции интерфейса можно использовать для перемещения по структуре записи – независимо от того, как именно этот интерфейс реализован.

4. Статистика событий и вторичная индексация трассы

Анализ работы приложения реального времени, предполагает наличие хотя бы минимальной статистики, позволяющей оценить, какие именно события происходили, в каких контекстах, какие объекты ОС в них участвовали.

4.1 Тройки $\langle e, c, o \rangle$

В фазе первичной обработки записей трассы определяется тип e очередной существенной записи $\overline{r}_j$, идентифицируются объекты трассировки и пополняется хеш-таблица H, используемая для поиска объектов по их идентификаторам (см. 3.5). Среди объектов трассировки, обнаруженных в трассе, выделяется контекст. Для любой существенной записи $\overline{r}_j$ можно указать тройку $\langle e, c, o \rangle$ (иначе – *eco*-тройку), где $e \in T_e$, $c \in O$, $Type(c) \in T_c$, главный объект $o \in O$, $Type(o) \in T_{(obj)}$.

В ходе последовательной обработки существенных записей трассы формируется хеш-таблица – для поиска *eco*-троек по значениям типа события и указателей на описатели

контекста и главного объекта. При обнаружения в трассе новой (отсутствующей в хеш-таблице) тройки, создается ее описатель – структура, содержащая тип события и ссылки (указатели) на описатели контекста и главного объекта, а также счетчик, в котором ведется учет «появлений» тройки в записях трассы. Каждой новой тройке назначается (в порядке обнаружения) идентификатор – очередной номер $ecoId$ тройки, после чего указатель на ее описатель заносится в хеш-таблицу $H_{(eoc)}$. Таким образом, каждая запись $\overline{r}_j$ оказывается снабжена идентификатором $ecoId$ соответствующей eco-тройки.

На заключительном этапе первичной обработки трассы, когда все eco-тройки занесены в хеш-таблицу $H_{(eoc)}$, формируется массив $ecoD$ троек, упорядоченный по идентификаторам $ecoId$, после чего доступ к тройке может быть осуществлен как по ее идентификатору $ecoId$, так и по совокупности значений ее компонент e, c и o (с помощью таблицы $H_{(eoc)}$).

4.2 Вторичная индексация трассы

Средства анализа трассы должны обеспечивать эффективный доступ в первую очередь к существенным записям трассы (несущественные записи, как правило, должны быть скрыты от пользователя). Другим требованием, предъявляемым к средствам анализа трассы, является наличие эффективных механизмов задания условий отбора и фильтрации записей, необходимых для решения конкретных задач анализа.

Как показывает опыт реализации средств трассировки семейства «Багет», подавляющее большинство задач анализа может успешно решаться в рамках процедур отбора записей, условия которых накладываются исключительно на компоненты eco–троек. Это означает, что для проверки условий отбора нет необходимости применять их к каждой записи $\overline{r}_j \in E$. Для начала (в первой фазе отбора) достаточно проверить на соответствие этим условиям все eco–тройки трассы, «помечая» в битовом массиве те из них, которые этим условиям удовлетворяют. (В дальнейшем – при обходе существенных записей трассы – можно воспользоваться тем, что с каждой записью связана *eco*-тройка, уже прошедшая проверку.)

В основе доступа к существенным записям трассы – наличие соответствия между номером j существенной записи $\overline{r}_j$ (элемента последовательности E) и определяющим адрес записи $\overline{r}_j$ первичным индексом i – номером элемента индексного массива I_L (см. 3.1). С другой стороны, каждому номеру j существенной записи $\overline{r}_j \in E$ соответствует eco–тройка с идентификатором $ecoId$. Это позволяет завершить отбор записей $\overline{r}_j$ процедурой их обхода (вторая фаза отбора), помечая те из них, «чьи» eco–тройки были «помечены» в первой фазе отбора.

Первичный индекс i существенной записи $\overline{r}_j$ вместе с идентификатором $ecoId$ соответствующей этой записи eco–тройки образует пару $\langle i, ecoId\rangle$. Компонента i этой пары обеспечивает быстрый доступ к адресу существенной записи, компонента $ecoId$ – к описателю eco–тройки, используемому при отборе записей.

Вторичным индексом существенной записи $\overline{r}_j$ трассы будем называть ее порядковый номер j. Для быстрого поиска записи $\overline{r}_j$ по ее индексу и проверки ее соответствия условиям отбора используется индексный массив, реализующий последовательность пар $\langle i, ecoId\rangle$:

$$I_E = (\langle i, ecoId\rangle)_{j=1}^{|E|},$$

такую что $\overline{r}_j = r_i, r_i \in L, e = Eid(i), e \in T_e$, $|E|$ – количество существенных записей.

4.3 Дерево событий – результат переупорядочивания массивов троек $\langle e, c, o\rangle$

Располагая информацией о количестве «появлений» в трассе каждой из троек $\langle e, c, o\rangle$, можно, например, получить статистику реализации события типа e в различных сочетаниях контекстов и главных объектов. Для этого достаточно упорядочить соответствующим

образом тройки и просуммировать счетчики тех из них, которые этому сочетанию удовлетворяют. То же самое можно сказать о любой другой компоненте тройки. В конечном счете, достаточно создать три массива Eco, Coe, Oec, являющиеся переупорядоченными копиями массива $ecoD$.

В случае массива Eco, имеется в виду лексикографическая упорядоченность $\langle e_1, c_1, o_1\rangle < \langle e_2, c_2, o_2\rangle$ означающая, что выполнено условие

$$(e_1 < e_2) \vee (e_1 = e_2) \wedge ((c_1 < c_2) \vee (c_1 = c_2) \wedge (o_1 < o_2)) \qquad (2)$$

Над массивом Eco можно надстроить массив Ec упорядоченных пар $\langle e, c\rangle$, для которых упорядоченность $\langle e_1, c_1\rangle < \langle e_2, c_2\rangle$, означающая, что $(e_1 < e_2) \vee (e_1 = e_2) \wedge (c_1 < c_2)$, прямо следует из (2). С любым i-м элементом $\langle e_i, c_i\rangle$ этого массива можно связать диапазон номеров $[j_1, j_2]$ элементов массива Eco, которому соответствует последовательность троек $D_i = \langle e_i, c_i, o_j\rangle_{j=j_1}^{j_2}$.

Из массива Ec можно, в свою очередь, сформировать переупорядоченную копию Ce, в котором порядок пар будет определяться условием $\langle e_1, c_1\rangle < \langle e_2, c_2\rangle$, означающим, что $(c_1 < c_2) \vee (c_1 = c_2) \wedge (e_1 < e_2)$. При этом, однако, i-му элементу $\langle e_i, c_i\rangle$ массива Ec и j-му элементу $\langle e_j, c_j\rangle$ массива Ce соответствует один и тот же диапазон D_k элементов массива Eco – при условии, что $(e_i = e_j) \wedge (c_i = c_j)$. Над массивом Ec аналогичным образом надстраивается массив E событий e_i, а над массивом Ce – массив C контекстов c_j. В процессе формирования массивов Ec и E для каждого события e_i могут быть подсчитаны количества его появлений в трассе. Аналогичные расчёты могут проведены для каждого из контекстов, в котором фиксируется событие e_i.

В целом же Eco-упорядоченность троек $\langle e, c, o\rangle$ порождает два варианта иерархии массивов: $E.Ec.Eco$ (события | контексты | объекты) и $C.Ce.Eco$ (контексты | события | объекты). Аналогичным образом Coe-упорядоченность порождает иерархии массивов: $C.Co.Coe$ (контексты | объекты | события) и $O.Oc.Coe$ (объекты | контексты | события), а Oec-упорядоченность – $O.Oe.Oec$ (объекты | события | контексты) и $E.Eo.Oec$ (события | объекты | контексты). Каждой из этих шести иерархий соответствует свой набор статистических данных, каждый из которых (по-своему) представляет интерес для анализа. Эти иерархии, образуют дерево статистики событий (дерево событий), обладающее способностью менять порядок иерархии в зависимости от режима упорядочивания троек $\langle e, c, o\rangle$, назначаемого пользователем.

4.4 Дерево событий – модель для отображения. Интерфейс *GtkTreeModel*

В подразделе 2.7 подчеркивалось, что комфортный просмотр данных, имеющих иерархическую структуру, обеспечивается тем, что над этим данными строится специальный интерфейс, которому в *GTK+*, например, соответствует *GtkTreeModel*. Дерево статистики событий (в своих шести «ипостасях») является, пожалуй, наиболее сложным из всех иерархических объектов, с которыми имеет дело Трассировщик.

Элементом любого из массивов Eco, Coe, Oec является указатель на структуру, описывающую терминальный узел дерева событий. Эта структура, наряду с данными, определяющими соответствующую тройку $\langle e, c, o\rangle$, содержит информацию о типе узла (зависящем от его положением в иерархии, в данном случае – указывающем на то, что содержимым узла является *eco*-тройка), а также ссылки на описатель трассы и на потенциальных родителей (их два – согласно 4.3). Описатель трассы объединяет всю информацию, необходимую для интерпретации трассы, в частности, содержит общий для трассы параметр настройки – вариант иерархии.

Современная версия Трассировщика допускает работу с несколькими трассами. Деревья статистики событий, относящиеся к различным трассам, представлены в общей таблице

(*GtkTreeView*), допускающей выделение нескольких строк. Эта возможность используется для задания списка трасс, сформированных взаимодействующими между собой целевыми ЭВМ. Такие списки обычно используются для объединения трасс в общей таблице, где их записи упорядочены по времени, что позволяет анализировать поведение многомодульной системы в целом.

Каждой трассе соответствует своя ветвь дерева, отображаемого виджетом *GtkTreeView*, в отношении которой используется свой вариант иерархии. Кроме того, в каждой из ветвей вводятся (для большей наглядности) дополнительные уровни – групп событий $\langle G \rangle$, типов контекстов $\langle T_c \rangle$ и типов главных объектов $\langle T_{(obj)} \rangle$. Как следствие всей совокупности узлов дерева событий конкретной трассы соответствуют иерархии типов:

- $\langle G \rangle|\langle e \rangle|\langle e,c \rangle|\langle e,c,o \rangle$,
- $\langle G \rangle|\langle e \rangle|\langle e,o \rangle|\langle e,c,o \rangle$,
- $\langle T_c \rangle|\langle c \rangle|\langle e,c \rangle|\langle e,c,o \rangle$,
- $\langle T_c \rangle|\langle c \rangle|\langle c,o \rangle|\langle e,c,o \rangle$,
- $\langle T_{(obj)} \rangle|\langle o \rangle|\langle e,o \rangle|\langle e,c,o \rangle$,
- $\langle T_{(obj)} \rangle|\langle o \rangle|\langle c,o \rangle|\langle e,c,o \rangle$.

Каждый из моно-узлов $\langle e \rangle$, $\langle c \rangle$ и $\langle o \rangle$ ссылается на соответствующие ему диапазоны двух вариантов дочерних массивов – $\{Ec, Eo\}$, $\{Ec, Co\}$ и $\{Eo, Co\}$, упорядоченных в соответствии с иерархией. Каждый из узлов-пар $\langle e,c \rangle$, $\langle e,o \rangle$, $\langle c,o \rangle$ ссылается на две родительские вершины $\{\langle e \rangle, \langle c \rangle\}$, $\{\langle e \rangle, \langle o \rangle\}$ и $\{\langle c \rangle, \langle o \rangle\}$, ссылка определяется иерархией. Остальным узлам, отличным от терминальных, всегда соответствует один дочерний и один родительский массив.

В предыдущих версиях Трассировщика дерево событий каждой трассы было представлено шестью независимыми деревьями, реализованными как объекты класса *GtkTreeStore*. Следствием такого решения была дорогостоящая процедура формирования из массивов Eco, Coe, Oec списков *GList*, лежащих в основе модели *GtkTreeStore*. Современная модель, основанная на реализации интерфейса *GtkTreeModel*, лишена этих недостатков. Модель внутреннего представления дерева событий едина, представляющие ее массивы ни во что не конвертируются и не меняются при переходе от одного варианта иерархии к другому. Меняется лишь единственный параметр трассы, соответствующий существенному варианту иерархии. Выбирая соответствующее значение этого параметра, пользователь практически мгновенно меняет иерархию отображаемого на экране дерева.

4.5 Дерево событий – инструмент для задания условий отбора записей трассы

Как и в прежних версиях Трассировщика, узлы дерева событий используются для задания условий отбора записей трассы. Они помечаются маркерами «*отобрать*» и «*исключить*» – в любой иерархии. Отбор записей осуществляется в два этапа. На первом этапе обрабатываются *eco*-тройки – либо помеченные непосредственно, либо подчиненные помеченным узлам. При этом маркер любого узла «сильнее» маркера его родителя: он переопределяет условие отбора. На втором этапе – в процессе обхода вторичного индексного массива I_E (см. 0) обновляются значения элементов битового массива – таким образом, что ненулевые биты соответствуют отобранным записям.

5. Отображение агрегатов данных в скроллируемых таблицах

Задачу отображения файла (набора данных в оперативной памяти), записи которого не имеют фиксированной длины, в виде одноуровневой скроллируемой таблицы – достаточно часто приходится решать при разработке *GUI*-приложений. При наличии большого числа записей особую актуальность приобретает скорость вертикальной прокрутки таблицы. Высокая скорость прокрутки обеспечивается в том случае, если в процессе скроллинга вычисление

пространственных координат любой строки требует минимальных затрат времени. В самом общем случае, средства разработки графического интерфейса, такие, например, как *GTK+*, рассчитывает поправку к значениям координат строк по оси Y всякий раз, когда меняется их высота (например, при изменении ширины колонок, допускающих перенос текста). Перед первым выводом содержимого таблицы на экран рассчитываются высоты всех без исключения строк, что сопровождается явным преобразованием данных отображаемой модели в тексты – в соответствии с выбором шрифтов.

Скорость формирования таблицы значительно повышается, если фиксировать высоту ее строк – поскольку при этом процедура разметки предельно упрощается, в частности, не требует обращения к функциям, определяющим параметры рендерера (см. 3.6). (Из этого условия, в частности, вытекает требование недопустимости переноса текста внутри ячеек «большой» таблицы.)

5.1 Основная таблица и виджет выделенной записи

Для увеличения скорости скроллинга таблиц с большим числом строк (не только таблицы с трассой событий) может быть применен единый подход, суть которого сводится к следующему:

- записи индексируются, индекс (номер записи) ассоциируется с ее адресом (длина либо задается непосредственно, либо – в случае выполнения условия связности записей – может быть определена как разность адресов соседних записей;
- в ячейках строки таблицы отображается не вся информация, содержащаяся в конкретной записи, а лишь та ее часть, которая присуща всем записям (например, время регистрации записи), что позволяет использовать фиксированную высоту строк таблицы;
- оставшаяся часть записи (либо ее полное содержание) отображается отдельным виджетом, причем лишь тогда, когда она выделена курсором в основной таблице (в мультиселектных таблицах – на последнем шаге выделения).

Реализация этого подхода применительно к трассе событий может быть проиллюстрирована следующей схемой (включающей дерево событий):

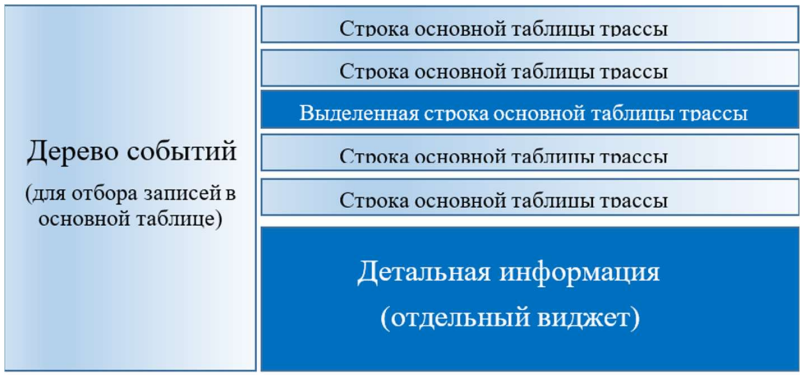

Рис. 1. Отображение детальной информации выделенной записи в отдельном виджете
Fig. 1. Visualization of selected record details by standalone widget

5.2 Линейная модель агрегатов данных

В общем случае наборы данных, отображаемые в скроллируемых одноуровневых таблицах, могут быть представлены последовательностью $D = (a_i)_{i=1}^{|D|}$ агрегатов данных a_i, где $|D|$ – количество агрегатов. Рассмотрим – в рамках концепции *MVC* (раздел 2.7) – одноуровневую

таблицу, являющуюся представлением (*View*) последовательности D. Модель (*Model*), реализующую доступ к элементу a_i этой последовательности по его номеру $i = value(node)$, где $node$ – узел *Model*, назовем линейной.

Позиция P любой строки таблицы определяется ее номером row. В отсутствие фильтрации записей этот номер совпадает с номером i агрегата данных a_i: $row = i$. Такая модель может быть применена к источникам данных любой природы, лишь бы они были представлены последовательностью D. В этом случае *Model* не нуждается в каких либо данных, отличных от количества агрегатов $|D|$.

Под фильтром номер строки не совпадает с номером записи агрегата данных, что побуждает строить массивы, в которых номерам строк, удовлетворяющих фильтру, соответствуют номера записей. Компактной реализацией такого соответствия является битовый массив $BitArray$, доступный для редактирования со стороны функций, внешних по отношению к *Model*. Включение $BitArray$ непосредственно в *Model*, позволяет свести навигацию (с помощью функций $next(node)$, $prev(node)$, $parent(node)$, $nth\_child(node, n)$) по отобранным агрегатам данных, к навигации по массиву $BitArray$.

Реализация функции $paint(render_i, c_i, node)$, определяющей параметры объекта $render_i$, отображающего данные, связанные с узлом $node$, в ячейке столбца c_i, обеспечивается тем, что в реализациях функции $node(P)$ и $P(node)$ также, как и в функциях навигации, используется массив $BitArray$.

В Трассировщике линейная модель представлена классом $Lines$, дочерним по отношению к классу *GObject* и реализующим методы интерфейса *GtkTreeModel*.

Эта модель может быть применена к агрегатам данных различной природы: к трассе, текстовому файлу, записи которого разделяет символ конца строки, к любому бинарному файлу, который отображается как последовательность записей, длина которых не фиксирована и т.п. Для любых типов агрегатов данных могут быть определены условия отбора, присущие только этому типу. От реализующих эти условия алгоритмов требуется одно: результатом отбора должно быть содержимое битового массива линейной модели.

6. Трассы состояний

В каждом из состояний объект ОС (или процессор) непрерывно пребывает в течение некоторого отрезка времени. Последовательность таких отрезков времени есть трасса состояний объекта (иначе – временная диаграмма состояний объекта). Переход из одного состояния в другое всегда есть результат некоторого события. Множество типов событий, обусловливающих переход, в сочетании с условиями перехода (в том числе, учитывающими предшествующее состояние объекта) определяют алгоритм построения последовательности смены состояний объекта – трассу состояний.

Детализация (классификация) состояний (см. 2.6) может иметь несколько уровней агрегирования.

6.1 Агрегированные состояния

Каждому уровню агрегирования соответствует номер l ($l = 1, \dots, N_L$, где N_L – максимальное число уровней агрегирования), а уровню атомарных состояний соответствует номер $l = 1$.

Состояния объекта o операционной системы образуют множество S_t, определяемое типом t объекта o:

$$S_t = \{\sigma_i^t | i = 1, \dots, N_t, t = Type(o), t \in T_{(obj)}\},$$

где N_t – количество возможных состояний объектов типа t.

Множество S_t включает в себя как атомарные, так и агрегированные состояния. Каждому агрегату уровня l соответствует подмножество множества S_t, состоящее из агрегатов уровня $(l-1)$. Для объектов типа t может быть определена функция $Aggregate(\sigma, t, l)$, результатом

выполнения которой является состояние $a \in S_t$, являющееся l-агрегатом по отношению к состоянию σ:

$$a = Aggregate(\sigma, t, l),$$

где $t = Type(o)$, l – номер уровня агрегирования.

6.2 Модель трассы атомарных состояний (временная диаграмма)

Пусть $\tau_1, \dots, \tau_n$ – последовательность моментов времени, в которых происходит смена состояний некоторого объекта o типа $t \in T_{(obj)}$. В течение промежутка времени (τ_i, τ_{i+1}), характеризуемого длительностью $\Delta\tau_i = \tau_{i+1} - \tau_i$, объект пребывает в состоянии $\sigma_i \in S_t$. Обычно (например, при отображении) трасса (временная диаграмма) состояний объекта o представляется в виде последовательности троек $s_i = \langle \tau_i, \sigma_i, \Delta\tau_i \rangle$:

$$\Im_o = (s_i)_{i=1}^{|\Im_o|},$$

где $|\Im_o|$ – количество переходов объекта o из одного состояния в другое, i – номер перехода. Поскольку каждому i-му моменту перехода соответствует определяющее его событие (существенная запись $\overline{r}_{j_i} \in E$, где $j_i \in I_E$ – вторичный индекс записи), то компоненты тройки s_i полностью определяются парой $(j_i$ и $j_{i+1})$ вторичных индексов, соответствующих соседним переходам из одного состояния в другое:

$$\tau_i = Time(j_i),\ \sigma_i = State(j_i, o), \tau_{i+1} = Time(j_{i+1}), \Delta\tau_i = \tau_{i+1} - \tau_i,$$

При этом предполагается, что описатель объекта o содержит необходимые ссылки и к трассе событий, и к ее описанию.

Нетрудно видеть, что хранить временную диаграмму в памяти инструментальной ЭВМ в виде троек $\langle \tau_i, \sigma_i, \Delta\tau_i \rangle$ нет необходимости. Временная диаграмма «атомарных» состояний полностью определяется массивом целых чисел, реализующих последовательность вторичных индексов $I_o = (j_i)_{i=1}^{|\Im_o|}$ $(I_o \subset I_E)$, где i – номер перехода. Получение значений, соответствующих «каноническому» представлению можно отложить до момента, когда они реально требуются – например, при их отображении.

6.3 Агрегированные состояния и многослойная модель агрегатов данных

Трасса агрегированных состояний объекта o уровня (слоя) l может быть представлена в виде последовательности $A_o^l = (\alpha_i)_{i=1}^{|A_o^l|}$, где α_i – i-й переход объекта в одно из возможных агрегированных состояний, $|A_o^l|$ – количество переходов. В то же время агрегированная трасса A_o^l – в соответствии с определением агрегирования – может быть представлена последовательностью целых чисел $I_o^l = (j_i^{l-1})_{i=1}^{|A_o^l|}$, где i – номер перехода внутри слоя l, а j_i^{l-1} – соответствующий ему номер перехода внутри слоя $(l-1)$. Отношения между уровнями агрегирования порождают естественную иерархию реализованных состояний.

Массивы, соответствующие слоям агрегирования, никак не связаны с природой данных, которые они объединяют в иерархическую структуру. При использовании *GTK+* они являются основой универсальной многослойной иерархической модели с интерфейсом *GtkTreeModel*. По отношению к этим массивам могут быть применены те или иные процедуры отбора. Для фильтрации отобранных записей (и навигации по ним в режиме отключенного фильтра) в модель встроен механизм, аналогичный тому, что реализован в линейной модели (см. 4.5 и 5.2): с каждым слоем связан битовый массив, ненулевые биты которого соответствуют отобранным элементам слоя.

В предыдущих версиях Трассировщика трассы состояний отображались в трех режимах: «детальная», «промежуточная» и «укрупненная», реализованных с помощью модели

GtkListStore. Отчасти это решение было продиктовано опасениями, что иерархическая модель окажется «медленнее» линейной.

Многослойная модель лежит в основе всех трасс состояний, с которыми работает современный Трассировщик. При этом скорость прокрутки и позиционирования в дереве, отображаемом на экране, скорость раскрытия родительских узлов не вызывает никакого дискомфорта: опасения, что иерархическая модель окажется существенно медленнее линейной, не подтвердились.

В предыдущих версиях Трассировщика трассы состояний отображались – наряду с табличной формой – в виде прямоугольников, размещенных в специальной «области рисования» (виджете *GtkDrawingArea*). В современной версии этой цели служат ячейки виджета *GtkTreeView*, в которых прямоугольники отображаются с помощью объектов класса *GtkCellRendererPixbuf*.

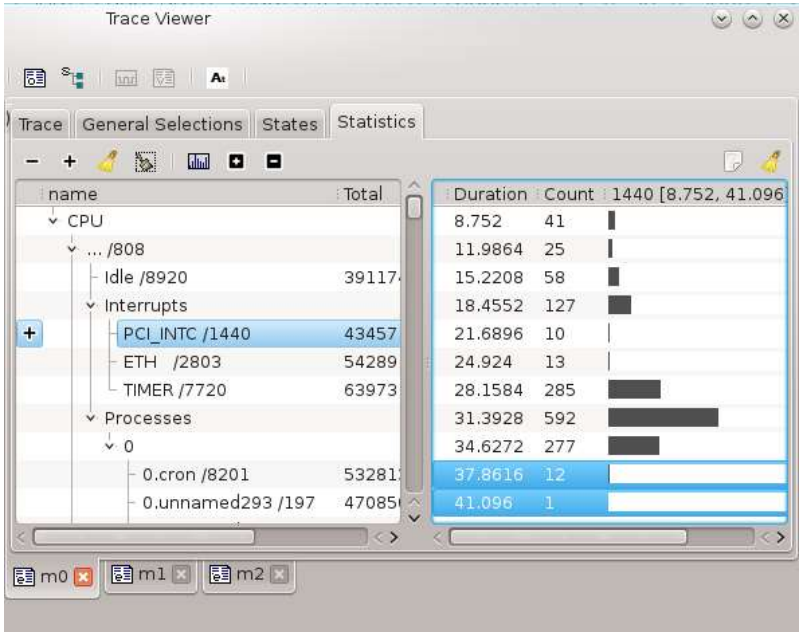

Рис. 2. Гистограмма состояния процессора
Fig. 2. Histogram of CPU state

6.4 Статистика состояний и условия отбора записей трасс состояний

В процессе формирования Трассировщиком трассы состояний *CPU* или объекта ОС рассчитываются его статистические характеристики: общая, средняя и максимальная длительность реализации состояний. Статистические данные по всем объектам, для которых строилась трасса состояний, накапливаются в дереве статистики состояний. Его верхний уровень образуют вершины, соответствующие процессору и тем типам объектов ОС, для которых построены трассы состояний. Каждый тип объединяет вершины второго уровня — соответствующие этому типу объекты ОС.

Содержимое ветви, подчиненной вершине, соответствующей конкретному объекту ОС или процессору (*CPU*) отражает иерархию состояний объекта с точки зрения «степени детальности» трассы, при этом «детали», конкретизирующие «менее подробное» состояние, находятся на более глубоких уровнях.

Условия отбора, накладываемые на записи трасс состояний, можно задавать, помечая узлы дерева статистики состояний маркерами «*отобрать*» и «*исключить*».

Моделью дерева является представление *GtkTreeStore*. С каждым его узлом по запросу пользователя может быть связана таблица (*GtkTreeView)*, отображаемая при выделении узла дерева статистики состояний. Колонки этого виджета служат для представления гистограммы реализаций связанного с узлом состояния. В них представлены количества реализаций состояний в данном диапазоне длительностей – в числах (в ячейках *GtkCellRendererText*), и в виде прямоугольников (в ячейках, поддерживаемых объектами *GtkCellRendererPixbuf*), ширина каждого из которых соответствует этому количеству.

Выделение одной или нескольких строк в таблице, отображающей гистограмму, задает дополнительные условия отбора записей трассы состояний: отобраны будут лишь те из них, у которых длительности реализации будут лежать в диапазонах, соответствующих выделенным «столбцам» гистограммы.

7. Заключение

Рассмотренные в статье модели трасс (протоколов событий), формируемых в процессе работы приложений реального времени, и трасс состояний, формируемых в ходе анализа событий, а также механизмы их интерпретации и визуализации реализованы в современных средствах трассировки ОС РВ семейства «Багет». Опыт их эксплуатации дает основания считать удачным использование предложенных подходов. К числу очевидных положительных результатов следует отнести значительное сокращение объема программного кода, повышение эффективности и надежности реализации основных функций средств трассировки.

Предпринятая в статье попытка представить предлагаемые решения в максимально формализованном виде преследовала единственную цель – показать, что эти решения никак не связаны ни с использованием языка программирования, ни со спецификой ОС РВ семейства «Багет», ни с особенностью реализации функций просмотра и анализа трасс. Несколько более детальное описание моделей и механизмов, реализованных с помощью *GTK+*, не означает, на наш взгляд, что подобные механизмы не могут быть реализованы с помощью иных инструментов, используемых при создании графического пользовательского интерфейса.

Список литературы / References

[1]. А.Н. Годунов, Л.В. Жихарский, П.Е. Назаров, Ф.Н. Чемерев. Средства протоколирования в ос2000. Программные продукты и системы, no. 3, 2007, стр. 22-27 / A.N. Godunov, L.V. Zhikharsky, P.E. Nazarov, F.N. Chemerev. Logging tools in oc2000. Software Products and Systems, no. 3, 2007, pp. 22-27 (in Russian).
[2]. IEEE 1003.1-2001 – IEEE Standard for IEEE Information Technology - Portable Operating System Interface (POSIX(R)). Available at: https://standards.ieee.org/standard/1003_1-2001.html.
[3]. Percepio Tracealyzer. Available at: https://percepio.com/tz/.
[4]. BlackBerry QNX. Available at: http://blackberry.qnx.com/.
[5]. Green Hills INTEGRITY-178. Available at: https://www.ghs.com/products/safety_critical/integrity-do-178b.html.
[6]. Express Logic ThreadX. Available at: https://rtos.com/.
[7]. On Time RTOS-32. Available at: http://www.on-time.com/rtos-32-docs/.
[8]. Wind River VxWorks. Available at: https://www.windriver.com/products/vxworks/.
[9]. SYSGO PikeOS. Available at: https://www.sysgo.com/products/pikeos-hypervisor.
[10]. Altreonic OpenComRTOS. Available at: http://www.altreonic.com/.
[11]. К.А. Костюхин. Средства самоконтроля программ и их применение при отладке систем со сложной архитектурой. В сборнике «Информационная безопасность. Микропроцессоры. Отладка сложных систем». М., НИИСИ РАН, 2004, стр. 151-160 / K.A. Kostyukhin. Tools for program self-control and their application in debugging of systems with complex architecture. In «Information Security. Microprocessors. Debugging complex systems.» M., SRISA/NIISI RAS, 2004, pp. 151-160 (in Russian).

[12]. С. Рогачев. Обобщённый Model-View-Controller. Каркас на основе шаблона проектирования MVC в исполнении Generic Java и C# / S. Rogachev. Generalized Model-View-Controller. Framework based on the MVC design pattern and implemented with Generic Java and C#. Available at: http://rsdn.org/article/patterns/generic-mvc.xml (in Russian).
[13]. GTK+ 3 Reference Manual. Available at: https://developer.gnome.org/gtk3/stable/.
[14]. Qt Documentation. Available at: https://doc.qt.io/.

Информация об авторах / Information about authors

Александр Николаевич ГОДУНОВ – заведующий отделом системного программирования, кандидат физико-математических наук.

Alexander Nikolayevich GODUNOV – head of the system programming department, candidate of physical and mathematical sciences.

Федор Николаевич ЧЕМЕРЕВ, ведущий инженер отдела системного программирования.

Fedor Nikolaevich CHEMEREV, Leading Engineer of the System Programming Department.