УДК 004.4'233
DOI: 10.15827/0236-235X.119.378-383
Дата подачи статьи: 04.05.17
2017. Т. 30. № 3. С. 378–383

ПРОБЛЕМЫ ОТЛАДКИ МНОГОПРОЦЕССНЫХ СИСТЕМ

В.А. Галатенко, д.ф.-м.н., зав. сектором, galat@niisi.ras.ru;
К.А. Костюхин, к.ф.-м.н., старший научный сотрудник, kost@niisi.ras.ru
(Федеральный научный центр Научно-исследовательский институт системных исследований РАН (ФНЦ НИИСИ РАН), Нахимовский просп., 36, корп. 1, г. Москва, 117218, Россия)

Статья посвящена проблеме отладки сложных многопроцессных систем. Разработка качественных сложных аппаратно-программных систем – длительный, трудоемкий процесс. Считается, что около половины времени уходит на отладку. Переход на многоядерные процессорные архитектуры сделал параллелизм нормой, равно как и специфичные для параллельных систем ошибки. Это делает отладку еще более важной и одновременно более сложной.

Основная особенность многопроцессных систем – использование сложных асинхронных взаимодействий между компонентами системы, эта особенность влияет на подходы к отладке систем, определяет выбор инструментов и методов отладки. Сложность отладки также обусловлена количеством и разнородностью компонентов многопроцессных систем, часть из которых могут быть аппаратными. Традиционный набор инструментов отладки в широком смысле (интерактивный отладчик, трассировщик, библиотеки самоконтроля, воспроизведения и отладки производительности) не теряют своей актуальности, но варианты их применения изменяются.

В работе сначала рассматриваются существующие средства и методы отладки, выделяются их достоинства и недостатки. Затем описываются существующие проблемы отладки многопроцессных систем и предлагается архитектура отладчика многопроцессных систем. В заключении представлены выводы авторов.

Ключевые слова: *контролируемое выполнение, отладка, мониторинг, трассировка, многопроцессные системы.*

Разработка качественных сложных аппаратно-программных систем – длительный и трудоемкий процесс. Считается, что около половины времени уходит на отладку (см., например, [1]). Переход на многоядерные процессорные архитектуры сделал параллелизм нормой, равно как и специфичные для параллельных систем ошибки. Это делает отладку еще более важной и одновременно более сложной.

Основная особенность многопроцессных систем – использование сложных, асинхронных взаимодействий между компонентами системы, эта особенность влияет на подходы к отладке систем, определяет выбор инструментов и методов отладки. Сложность отладки также обусловлена количеством и разнородностью компонентов многопроцессных систем, часть из которых могут быть аппаратными. Традиционный набор инструментов отладки (интерактивный отладчик, трассировщик, библиотеки самоконтроля, воспроизведения и отладки производительности) не теряет своей актуальности, но варианты применения инструментов отладки изменяются.

Основные способы отладки, их возможные сочетания и особенности

Интерактивная отладка. Традиционно под отладкой понимается прежде всего интерактивная отладка. В то же время, чем сложнее отлаживаемая система, тем менее эффективна подобная отладка. Слишком много факторов приходится принимать во внимание и много данных необходимо проанализировать, чтобы ответить, например, на простые, но важные вопросы: случались ли уже в системе ошибки и каковы их первопричины.

Наиболее популярным инструментом интерактивной отладки является GDB [2]. На момент написания статьи он оставался однопроцессным; многопроцессные возможности были на начальной стадии реализации, и, как показал практический опыт, их нельзя было считать работоспособными. В то же время развитие GDB в интересующем авторов направлении активно обсуждается в литературе.

В [3] предложена реализация так называемой безостановочной многопроцессной отладки, ориентированной на слабосвязанные системы. Безостановочность заключается в возможности отладки отдельного потока в операционной среде Linux без остановки процесса целиком.

Такая функциональность достигается за счет расширения возможностей агента отладки, а также существенной переработки внутреннего устройства отладчика, создания механизма асинхронного обслуживания событий и контроля применимости отладочных действий. Этот шаг очень важен и для развития многопроцессной отладки.

Интерактивной отладке присуща также такая проблема, как внесение недопустимо серьезных возмущений в процесс функционирования отлаживаемых систем. Этот процесс и без отладчика, вообще говоря, является недетерминированным, поэтому заранее нельзя сказать, проявятся ли ошибки в данном конкретном запуске; отладочные действия усугубляют недетерминизм, что способно привести к исчезновению предмета отладки.

Проблеме минимизации возмущений, вносимых отладочными инструментами категории GDB, посвящена статья [4]. В ней описываются различные механизмы, необходимые для многопроцессной отладки: централизованный ввод-вывод отладочных сообщений, параллельный сбор данных во время выполнения, использование встроенных в приложение отладочных средств и перехват передачи сообщений между процессами.

Командный язык отладки. При отладке многопроцессных систем возрастает роль командного языка отладчика. Условные точки прерывания и реакция на их достижение могут и должны программироваться заранее. Весьма вероятно, что эти программы должны затрагивать не только непосредственно отлаживаемый процесс, но и процессы, возможно, функционирующие на других процессорах. Необязательно видеть состояние сразу всех процессов (для больших систем это не имеет смысла), но необходимо иметь возможность воздействовать на все процессы – поодиночке или группами. Подобное групповое воздействие – обязательная черта многопроцессного отладчика.

В настоящее время в качестве командного языка отладчика выбирается интерпретируемый полноценный язык программирования, такой как Python. Поддержка Python появилась в GDB версии 7.0. Наличие полноценного языка с большим количеством библиотек позволяет расширить возможности отображения данных, в том числе в графическом формате, организовать взаимодействие с базами данных и реализовать сложные механизмы управления приложением.

Трассировка. Вероятно, основным инструментом отладки многопроцессных систем можно считать трассировку. Она может иметь различную степень детализации – от машинной инструкции до программных транзакций. При аппаратной поддержке возмущения, вызываемые трассировкой, могут быть приемлемыми, а центр тяжести переносится на последующий анализ собранной информации. Еще одно применение этой информации – детерминированное воспроизведение запуска параллельной системы, при котором проявились ошибки. Детерминированное воспроизведение может сопровождаться интерактивной отладкой; вероятно, это можно считать наиболее уместным применением последней.

При трассировке с высокой степенью детализации объем собранной информации может исчисляться гигабайтами. Для его уменьшения существуют средства сжатия трасс и, что более существенно, условные точки трассировки, благодаря которым можно накапливать только информацию, вероятно, имеющую отношение к ошибкам и их проявлениям. В таких точках может также активизироваться интерактивный отладчик, который в данном случае остается средством ближнего боя.

Помещать события в трассу можно не только внешними средствами, но и из программы, используя механизмы самоконтроля и генерации событий. Подобное оснащение программ представляется наиболее целесообразным, так как оно учитывает семантику программ и предполагаемую первопричину разыскиваемых ошибок, минимизирует возмущения, вносимые в процесс функционирования систем, и объем собираемой информации. И здесь возможна активизация интерактивного отладчика.

Практичный инструментарий трассировки – популярный предмет публикаций и реализаций [5, 6].

Самоотладка. Весьма удачным представляется подход, предложенный в статье [7] для отладки беспроводных сетей датчиков. Окна интерактивной отладки ассоциируются с некими представительными элементами аппаратно-программной конфигурации, но за счет использования групповых операций (глобальных команд) gstop, gcontinue, gdetach, greset, gbreak отладка сильносвязанных систем оказывается реальной.

Многопроцессорные конфигурации способны не только создавать новые проблемы, но и использоваться при отладке: на одних процессорах может функционировать отлаживаемая система, а на других – средства отладки. Если память подобной конфигурации является разделяемой, то возмущения, вносимые отладочными действиями, могут быть минимальными.

Детерминированное воспроизведение выполнения. Запуски, в которых проявляются ошибки, могут не только детерминированно воспроизводиться (примером каркаса для подобного воспроизведения может служить PinPlay [8]), но и синтезироваться как результат анализа программ (см. Execution Synthesis [9]). Вообще предварительный и «посмертный» анализ программ и данных, собранных в процессе функционирования, позволяет минимизировать вносимые отладкой возмущения (см. Analyzing Multicore Dumps [10], ConMem [11]).

Вариантом детерминированного воспроизведения может считаться механизм создания контрольных точек останова. Такие контрольные точки могут использоваться для более быстрого воспроизведения ошибки при ее обнаружении. При этом выполнение может быть неполным, но способствовать более быстрому воспроизведению аварийной ситуации.

Отладка аппаратных конфигураций. Современная постановка задачи отладки затрагивает такие аспекты, как отладка специфических аппаратных конфигураций, включающих в себя процессоры цифровой обработки сигналов или графические процессоры (см. Total Recall [12], NUDA [13]). Для отладки сложных сетей на кристалле предназначены инструменты, описанные в работе [14].

Отладка производительности. Еще один важный аспект – отладка производительности. Возможные решения в этой области представлены в работах [15, 16].

Проблемы многопроцессной отладки

Все рассмотренные средства отладки полезны и для отладки последовательных приложений. Сложности при отладке многопроцессных приложений

связаны со значительным увеличением числа отслеживаемых сущностей, с усложнением их взаимодействий.

Рассмотрим, как увеличение количества сущностей отражается на отладочных действиях в системе.

Взаимодействие с вычислительной средой. В многопроцессных взаимодействиях часто проявляется и особая сущность – подсистема взаимодействий. Она обеспечивает передачу данных между процессами и их синхронизацию. Это может быть система доставки сообщений, встроенная в ядро операционной системы, например, подсистема разделяемой памяти или программная библиотека, такая как MPICH [17]. Подобно тому, как реализация отладки требует поддержки операционной системы, в многопроцессных приложениях важно взаимодействие средств отладки и системы межпроцессного взаимодействия.

Описание конфигурации отлаживаемой системы. Первая проблема многопроцессной отладки – описание и отображение отлаживаемой конфигурации. При наличии большого количества вычислительных ресурсов возникает необходимость их распределения. Это распределение может быть статическим, когда конфигурация системы задается в файле или в специальной системе планирования. Примером такого распределения является конфигурация ARINC процессов в ОС3000.

Следует отметить, что проблема большого числа сущностей возникает не только при отладке многопроцессного приложения. Разработчику приложения приходится решать задачу разделения вычислительных ресурсов. Эта задача часто решается разделением множества модулей на функциональные компоненты, каждый из которых решает специфическую подзадачу. Например, некоторое множество процессов может отвечать за планирование, другое – за сбор информации, третье – за обработку, четвертое – за хранение. В такой системе естественным образом возникает необходимость работы с группами процессов в отладчике. Описание системы, созданное при ее проектировании, должно быть объединено с описанием системы в отладчике.

Распределение задачи по процессам может быть и динамическим. Например, сети автономных датчиков могут самоорганизовываться, определять имеющиеся ресурсы и планировать выполнение. Более того, часто в больших системах число компонент может со временем как расти, так и уменьшаться. Чем больше число процессов, тем больше вероятность отказа некоторых из них. Если приложение состоит из трех процессов, аварийное завершение одного из них равносильно завершению приложения. В системе из 1 000 процессов отказ – нормальное явление.

В любом случае отладочные средства должны иметь доступ к распределению процессов, возможность идентифицировать процессы, находить для каждого из них дополнительную информацию, такую как исходный код, расположение в памяти, аппаратную архитектуру.

Сбор, отображение и анализ данных. Ключевая проблема многопроцессной отладки – сбор и отображение большого количества данных. При этом нужно отметить, что в многопроцессных системах часто отсутствуют понятия времени и состояния или эти понятия являются размытыми. Соответственно, часто нельзя говорить о состоянии системы, смене состояний. В лучшем случае в системе присутствуют контрольные точки, в которых состояние системы определено. Многопроцессные системы порождают большое количество данных, слабо связанных между собой. Часто эти данные не позволяют говорить о том, корректно ли работает приложение.

Многопроцессный отладчик должен уметь работать со значительными потоками информации, а именно:

– собирать и группировать данные от различных процессов и групп процессов;

– анализировать данные, выявлять закономерности и аномалии;

– иметь возможность детализации представленной информации.

Наиболее популярное средство многопроцессной отладки – использование функций ввода-вывода для печати отладочной информации. Средства обработки вывода большого числа процессов описаны в статье [18]. Например, используется представление стеков вызова группы процессов, в котором функции объединены.

Кроме состояния системы, важно уметь отобразить взаимодействия системы. Для сложных систем это может быть нетривиальной задачей.

Управление выполнением. Гибкое управление выполнением – традиционная возможность многопроцессных отладчиков. В стандарте HPDS описываются возможности управления группами процессов – группы процессов, групповые точки останова, барьеры и т.п. Группы процессов могут быть полезны и для других задач, например, для воспроизведения состояния. Воспроизведение работы подсистемы – более простая задача, если такое воспроизведение возможно.

Существует важная задача, решение которой в стандарте HPDS возложено на пользователя, – поддержание целостности системы в процессе отладки. Например, останов некоторой группы процессов для начала интерактивной отладки может не сказаться на работе системы. Связи компонентов необязательно должны быть жесткими, часто слабая связанность компонентов должна обеспечивать поддержание работы системы. Использование информации о таких связях в отладчике позволяет упростить отладку многопроцессных приложений. Например, система самоконтроля может останав-

ливать только группу процессов при наступлении аварийного состояния. Информация о связанности компонентов, их взаимодействиях может быть извлечена из конфигурации системы.

Оптимизация отладочных действий. Отладочные действия в многопроцессной конфигурации могут требовать значительных ресурсов. Многие задачи отладки, такие как воспроизведение, анализ стека вызовов, сжатие трассы, используют и процессор, и сетевую среду. При этом число данных растет даже быстрее, чем число процессов. Например, при трассировке взаимодействий многопроцессных систем объем информации может быть пропорционален квадрату числа процессов.

Экономия ресурсов при многопроцессной отладке достигается за счет следующих решений:

– обработка отладочных действий на каждом из вычислительных узлов;

– многоуровневая система агентов;

– использование широковещательных сообщений;

– интроспекция отладочных интерфейсов;

– возросшее использование эффективных отладочных средств, например трассировки;

– сжатие отладочной информации.

Масштабируемость отладочной системы – одна из главных проблем многопроцессной отладки. Сложность систем определяет необходимость использования специальных протоколов, а также архитектурных решений. При этом большинство протоколов, используемых в отладке, недостаточно расширяемые (например, простой протокол удаленной отладки, используемый в GDB).

В работе [19] описана реализация отладочных агентов с использованием протокола WSRF (Web Service Resource Framework). Это решение позволило создать отладчик для масштабной гетерогенной системы.

Проблемы многопроцессной отладки на различных этапах поиска и устранения ошибок

Функциональные свойства средств отладки – не единственное решение многопроцессных проблем. Другой класс вопросов, связанных с процессом отладки, показывает, что не только инструменты отладки должны быть другими, но и отлаживаемая система должна вести себя по-другому [20]. Рассмотрим, как процесс отладки определяет взаимодействия отладчика и многопроцессного приложения.

Получение информации об ошибке. Многопроцессная система состоит из большого набора компонент, возможно, слабосвязанных. Даже обнаружение ошибки является проблемой. Часто проявления ошибки настолько скрыты, что в потоке порождаемой информации их просто невозможно увидеть. Например, некоторые компоненты системы могут перезапускаться по аварийному сигналу, но это никак не отразится на внешнем результате. О наличии ошибки можно судить только по падению производительности, или ошибка остается незамеченной.

Важно, чтобы некорректное поведение проявлялось как можно раньше. Возможные средства обнаружения ошибки включают средства самоконтроля, анализа поведения, обнаружения аномалий поведения [21].

Изоляция ошибки. Изоляция ошибки – важная задача даже при наличии инструментов воспроизведения. Последние не всегда позволяют работать с системой целиком, их применимость может ограничиваться некоторой подсистемой.

Этот аспект многопроцессной отладки более прост, чем отладка приложения в единой памяти. Средства изоляции компонент, слабая связанность, спецификации интерфейсов, средства контроля безопасности помогают найти проблемный компонент. Зачастую он более прост, чем сложная последовательная программа.

Проверка гипотез. После того как ошибка изолирована, необходимо найти ее причину. Поиск причины состоит из формулировки гипотез о ее нахождении и проверки этих гипотез. Эффективный способ проверки гипотез – использование встраиваемых в код агентов. Современные системы поддерживают способы модификации приложения, не требующие компиляции. Например, обработчики системных событий могут компилироваться на лету и эффективно выполняться. Примером такой системы может служить система [16], посвященная не только отладке производительности, но и традиционной многопроцессной отладке.

Изучение последствий. Заключительный этап исправления ошибки – изучение последствий ошибки. Как наличие ошибки отразилось на поведении системы в прошлом, к чему это привело? Возникали ли в прошлом ситуации, в которых ошибка могла привести к негативным последствиям? Привело ли к негативным последствиям исправление ошибки?

Система протоколирования должна быть разработана таким образом, чтобы протокол событий мог ответить на подобные вопросы. К сожалению, авторам неизвестны специальные инструменты, предназначенные для нахождения последствий ошибки.

Архитектура многопроцессной отладки

Увеличение числа сущностей отладки определяет возросшую важность архитектуры отладочных средств.

Архитектура позволяет решать проблемы минимизации последствий отладки, интеграции отладочных средств, адаптации отладочных средств к аппаратно-программной конфигурации. Большое число работ по многопроцессной отладке уделяет внимание архитектуре.

В качестве примера приведем архитектуру отладчика из [18]. Конечно, важным фактором, определяющим архитектуру отладчика, является программно-аппаратная платформа, но все же можно перечислить основные принципы построения многопроцессных отладчиков:

- древовидная структура;
- инкапсуляция;
- фильтрация данных на всех уровнях;
- агрегация данных;
- групповые операции;
- распределение полномочий отладки;
- использование хранилища данных для обеспечения доступа средств анализа;
- слабая связанность;
- асинхронные взаимодействия;
- интеграция с вычислительной платформой;
- интеграция средств отладки и приложений.

Перечисленные решения традиционны для архитектуры сложной многокомпонентной системы, например, системы управления. В большинстве случаев их применение оправдано, так как многопроцессная отладочная система действительно сложна. Подобная архитектура многопроцессных отладчиков обеспечивает их расширяемость, настраиваемость и эффективность.

Заключение

Набор отладочных инструментов, применяемых в многопроцессной отладке, значительно шире набора инструментов традиционных последовательных отладчиков. Это усложнение – необходимость, определяемая сложностью отлаживаемой системы. Без средств визуализации трассировки, воспроизведения, самоконтроля найти ошибку зачастую невозможно. Эффективная отладка требует применения всего набора инструментов, причем каждый из них должен быть разработан с учетом сложности системы, с использованием вышеперечисленных подходов.

Важно, что подход к многопроцессной отладке требует не только применения различных инструментов, но и их интеграции в отлаживаемую систему, взаимодействия средств отладки с вычислительной средой. Эту новую по сравнению с традиционной отладкой особенность необходимо принимать во внимание не только при разработке отладчика, но и при проектировании приложений.

Литература

1. Timmerman M., Gielen F., Lambrix P. High level tools for the debugging of real-time multiprocessor systems. ACM SIGPLAN Notices, 1993, vol. 28, iss. 12, pp. 151–157.
2. GNU Debugger. URL: http://www.gnu.org/software/gdb/ (дата обращения: 03.05.2017).
3. Non-stop Multi-Threaded Debugging in GDB. URL: http://www.codesourcery.com/publications/non_stop_multi_threaded_debugging_in_gdb.pdf (дата обращения: 03.05.2017).
4. Beynon M.D., Andrade H., Saltz J. Low-Cost Non-Intrusive Debugging Strategies for Distributed Parallel Programs, IEEE Intern. Conf. on Cluster Comp., 2002, pp. 439–442.
5. Yusen Li, Feng Wang, Gang Wang, Xiaoguang Liu, Jing Liu. MKtrace: an innovative debugging tool for multi-threaded programs on multiprocessor systems. Soft. Eng. Conf., APSEC 2007, pp. 510–517.
6. Moore L.J., Moya A.R. Non-intrusive debug technique for embedded programming, Soft. Reliability Eng., ISSRE 2003, pp. 375–380.
7. Yang Jing, Soffa Mary Lou, Selavo Leo, Whitehouse Kamin. Clairvoyant: a comprehensive source-level debugger for wireless sensor networks. Proc. of the 5th Intern. Conf. on Embedded Networked Sensor Systems, SenSys '07, 2007, pp. 189–203.
8. Patil Harish, Pereira Cristiano, Stallcup Mack, Lueck Gregory, Cownie James. PinPlay: a framework for deterministic replay and reproducible analysis of parallel programs. Proc. 8th Annual IEEE/ACM Intern. Symp. Code Generation and Optimization, CGO'10, 2010, pp. 2–11.
9. Zamfir C., Candea G. Execution synthesis: a technique for automated software debugging. Proc. 5th Europ. Conf. on Comp. Systems, EuroSys '10, 2010, pp. 321–334.
10. Weeratunge Dasarath, Zhang Xiangyu, Jagannathan Suresh. Analyzing multicore dumps to facilitate concurrency bug reproduction, SIGPLAN Not., 2010, vol. 45, no. 3, pp. 155–166.
11. Zhang Wei, Sun Chong, Lu Shan. ConMem: detecting severe concurrency bugs through an effect-oriented approach. SIGARCH Comput. Archit. News, 2010, vol. 38, no. 1, pp. 179–192.
12. Sharif Ahmad, Lee Hsien-Hsin S. Total recall: a debugging framework for GPUs. Proc. 23rd ACM SIGGRAPH/Eurographics Sympos. on Graphics Hardware, GH '08, 2008, pp. 13–20.
13. Wen Chi-Neng, Chou Shu-Hsuan, Chen Tien-Fu, Su Alan Peisheng. NUDA: a non-uniform debugging architecture and non-intrusive race detection for many-core. Proc. 46th Annual Design Automation Conf., DAC '09, 2009, pp. 148–153.
14. Albrecht Mayer, Harry Siebert, Klaus D. McDonald-Maier. Boosting debugging support for complex systems on chip. Computer, 2007, vol. 40, no. 4, pp. 76–81.
15. West Paul E., Peress Yuval, Tyson Gary S., McKee, Sally A. Core monitors: monitoring performance in multicore processors. Proc. 6th ACM Conf. on Comp. Frontiers, CF '09, 2009, pp. 31–40.
16. Wisniewski R.W., Rosenburg B. Efficient, unified, and scalable performance monitoring for multiprocessor operating systems. Proc. 2003 ACM/IEEE Conf. on Supercomputing, SC '03, 2003, pp. 3–14.
17. MPICH. URL: www.mcs.anl.gov/mpi/mpich/ (дата обращения: 03.05.2017).
18. Susanne M. Balle, Bevin R. Brett, C.-P.Chih-Ping Chen, David LaFrance-Linden. Extending a traditional debugger to debug massively parallel applications. Jour. of Parallel and Distributed Computing, 2004, vol. 64, iss. 5, pp. 617–628.
19. Kurniawan D., Abramson D. A WSRFCompliant debugger for grid applications. Parallel and Distributed Proc. Sympos., 2007. IPDPS 2007. IEEE Intern., 2007, pp. 26–30.
20. Галатенко В.А., Костюхин К.А., Малиновский А.С., Шмырев Н.В. Моделирование и верификация программ как элемент контролируемого выполнения // Программные продукты и системы. 2008. № 4. С. 13–16.
21. Галатенко В.А., Костюхин К.А. Отладка и мониторинг распределенных разнородных систем // Программирование. 2002. № 1. С. 27–37.

Software & Systems Received 04.05.17
DOI: 10.15827/0236-235X.119.378-383 2017, vol. 30, no. 3, pp. 378–383

MULTIPROCESS SYSTEM DEBUGGING PROBLEMS

V.A. Galatenko [1], *Dr.Sc. (Physics and Mathematics), Head of Sector, galat@niisi.ras.ru*
K.A. Kostyukhin [1], *Ph.D. (Physics and Mathematics), Senior Researcher, kost@niisi.ras.ru*

[1] *Federal State Institution "Scientific Research Institute for System Analysis of the Russian Academy of Sciences" (SRISA RAS), Nakhimovsky Ave. 36/1, Moscow, 117218, Russian Federation*

Abstract. The article considers multiprocess complex systems debugging. The development of high-quality complex systems is a long and time-consuming process. It is believed that debugging takes about a half of this time. Switching to multi-core processor architectures has made parallelism a regular task, as well as errors specific for parallel systems. This makes debugging even more important and at the same time more complicated.

The main feature of multiprocessing systems is using complex, asynchronous interactions between system components. This feature influences the approaches to debugging, determines the choice of debugging tools and methods. The complexity of debugging is also determined by the quantity and heterogeneity of multi-process system components, some of them might be hardware. A traditional set of debugging tools is the following: an interactive debugger, a tracer, self-monitoring libraries, reverse execution. Their application changes.

The first chapter describes the existing debugging tools and techniques. The authors highlight their advantages and disadvantages. The next two chapters describe the existing problems of multiprocess system debugging. The fourth chapter provides the architecture of multiprocess system debugger. In conclusion authors present the results of their work.

Keywords: controlled execution, debugging, monitoring, tracing, multiprocess systems.

References

1. Timmerman M., Gielen F., Lambrix P. High level tools for the debugging of real-time multiprocessor systems. *ACM SIGPLAN Notices*. 1993, vol. 28, iss. 12, pp. 151–157.
2. *GNU Debugger*. Available at: http://www.gnu.org/software/gdb/ (accessed May 3, 2017).
3. *Non-stop Multi-Threaded Debugging in GDB*. Available at: http://www.codesourcery.com/publications/non_stop_multi_threaded_debugging_in_gdb.pdf (accessed May 3, 2017).
4. Beynon M.D., Andrade H., Saltz J. Low-Cost Non-Intrusive Debugging Strategies for Distributed Parallel Programs. *IEEE Int. Conf. on Cluster Computing*. 2002, pp. 439–442.
5. Li Yu., Wang F., Wang G., Liu X., Liu J. MKtrace: An Innovative Debugging Tool for Multi-Threaded Programs on Multiprocessor Systems. *Software Engineering Conf. (APSEC 2007)*. 2007, pp. 510–517.
6. Moore L.J., Moya A.R. Non-intrusive debug technique for embedded programming. *Software Reliability Engineering (ISSRE 2003)*. 2003, pp. 375–380.
7. Yang Jing, Soffa M.L., Selavo L., Whitehouse Kamin. Clairvoyant: a comprehensive source-level debugger for wireless sensor networks. *Proc. of the 5th int. conf. on Embedded networked sensor systems (SenSys '07)*. 2007, pp. 189–203.
8. Patil H., Pereira C., Stallcup M., Lueck G., Cownie J. PinPlay: a framework for deterministic replay and reproducible analysis of parallel programs. *Proc. 8th Annual IEEE/ACM Int. Symp. on Code Generation and Optimization (CGO '10)*. 2010, pp. 2–11.
9. Zamfir C., Candea G. Execution synthesis: a technique for automated software debugging. *Proc. 5th European conf. on Computer systems (EuroSys '10)*. 2010, pp. 321–334.
10. Weeratunge Dasarath, Zhang Xiangyu, Jagannathan Suresh. Analyzing multicore dumps to facilitate concurrency bug reproduction. *ACM SIGPLAN Not*. 2010, vol. 45, no. 3, pp, 155–166.
11. Zhang Wei, Sun Chong, Lu Shan. ConMem: detecting severe concurrency bugs through an effect-oriented approach. *ACM SIGARCH Comput. Archit. News*. 2010, vol. 38, no. 1, pp. 179–192.
12. Sharif Ahmad, Lee Hsien-Hsin S. Total recall: a debugging framework for GPUs. *Proc. 23rd ACM SIGGRAPH/EUROGRAPHICS Symp. on Graphics Hardware* (*GH '08)*. 2008, pp. 13–20.
13. Wen Chi-Neng, Chou Shu-Hsuan, Chen Tien-Fu, Su Alan Peisheng NUDA: a non-uniform debugging architecture and non-intrusive race detection for many-core. *Proc. 46th Annual Design Automation Conf.* (*DAC '09)*. 2009, pp. 148–153.
14. Mayer A., Siebert H., McDonald-Maier K.D. Boosting Debugging Support for Complex Systems on Chip. *Computer*. 2007, vol. 40, no. 4, pp. 76–81.
15. West P.E., Peress Y., Tyson G.S., McKee S.A. Core monitors: monitoring performance in multicore processors, *Proc. 6th ACM conf. on Computing Frontiers (CF '09)*. 2009, pp. 31–40.
16. Wisniewski R.W., Rosenburg B. Efficient, Unified, and Scalable Performance Monitoring for Multiprocessor Operating Systems. *Proc. 2003 ACM/IEEE Conf. on Supercomputing (SC '03)*. 2003, pp. 3–14.
17. *MPICH*. Available at: www.mcs.anl.gov/mpi/mpich/ (accessed May 3, 2017).
18. Balle S.M., Brett B.R., Chih-Ping C.-P. Chen, LaFrance-Linden D. Extending a traditional debugger to debug massively parallel applications. *Jour. of Parallel and Distributed Computing*. 2004, vol. 64, iss. 5, pp. 617–628.
19. Kurniawan D., Abramson D. A WSRFCompliant Debugger for Grid Applications. *IEEE Int. Parallel and Distributed Processing Symp. IPDPS 2007*. 2007, pp. 26–30.
20. Galatenko V.A., Kostyukhin K.A., Malinovsky A.S., Shmyrev N.V. Program modelling and verification in controlled execution paradigm. *Programmnye produkty i sistemy* [Software & Systems]. 2008, no. 4, pp. 13–16 (in Russ.).
21. Galatenko V.A., Kostiukhin K.A. Debugging and monitoring of the distributed heterogeneous systems. *Programming and Computer Software*. 2002, no. 1, pp. 27–37.