УДК 681.3

Э. М. Рябиков, М. М. Заславский, К. В. Кринкин

МЕТОДИКА НЕИНВАЗИВНОГО ПРОФИЛИРОВАНИЯ РАЗДЕЛЯЕМЫХ БИБЛИОТЕК В ОС LINUX

Рассмотрена методика профилирования вызовов функций в динамических библиотеках Linux в условиях недоступности исходных текстов профилируемых программ или полномочий для отладки. Рассматриваются методы внедрения функций-редиректоров вызовов и бинарная инструментализация программ процедурами измерения времени выполнения и подсчета количества вызовов. Приводятся результаты тестирования и обсуждаются ограничения данной методики.

Оптимизация производительности ПО, Linux, ELF, разделяемые библиотеки, динамическая компоновка, динамическая загрузка, неинвазивное профилирование

Оптимизация производительности современных программных систем занимает одну из ключевых ролей в процессе разработки. При этом оптимизации программ предшествует профилирование [1], [2]. От результатов и качества профилирования зависит возможность улучшения характеристик целевой программы. Одним из специальных случаев профилирования является выявление временных и частотных характеристик вызовов подпрограмм из динамически компонуемых библиотек (shared libraries) [3], [4]. Стандартные методы, основанные на инструментализации исходного кода изучаемых программ, имеют широкий диапазон применения, однако не позволяют работать при отсутствии возможностей для компиляции библиотек, что особенно *актуально* при отладке встраиваемых приложений и систем.

Рассматриваемая в данной статье методика позволяет выполнять профилирование при отсутствии исходного кода и привилегий суперпользователя в операционной системе Linux.

Ограничения стандартных методов профилирования разделяемых библиотек в Linux. На сегодняшний день для ОС Linux существует большое количество разнообразных инструментов профилирования. Среди них наиболее популярными являются:

– GNU Profiler (gprof);

– GNU Coverage testing tool (gcov);

– Google Performance Tools (GPT);

– Valgrind.

При использовании перечисленных программ пользователи сталкиваются со следующими проблемами, препятствующими проведению оптимизации в описанной ранее ситуации:

1. Необходимость повторной компиляции или перекомпоновки анализируемой программы (в случае gprof, gcov и GPT).

2. Для запуска профилируемой программы требуется создание специального окружения (в случае Valgrind), что, по сути, не позволяет увидеть реальное поведение программы.

3. Проблемы с профилированием динамически загружаемых функций [4], [5].

4. Необходимость в запуске инструмента с правами суперпользователя.

Наличие метода и инструментов профилирования, позволяющих избежать перечисленных трудностей, является актуальной проблемой для большого числа разработчиков ПО.

Принципы неинвазивного профилирования разделяемых библиотек. Как уже отмечалось, разработанная методика неинвазивного (noninvasive) профилирования позволяет решить описанные проблемы.

Основные ее принципы состоят в следующем:

1. Инструмент профилирования не должен внедряться в программный код и оснащать его дополнительными средствами.

2. Профилирование проводится над выбранным множеством функций, а не над всеми сразу.

3. Профилирование должно проходить в конкретные моменты вызова интересующих функций.

4. Действия по профилированию не должны затрагивать алгоритм работы анализируемого приложения.

5. Профилировщик должен потреблять минимальное количество ресурсов процессора.

6. Профилирование не должно зависеть от прав на использование системных средств.

7. Полученная в результате профилирования информация о времени выполнения участков программы должна быть максимально точной.

Реализация данных идей основана на внедрении в процесс работы с разделяемыми библиотеками на стадии компоновки и загрузки.

На рис. 1 показано, на каком этапе ld-linux.so и libdl.so участвуют в сборке и загрузке программ в Linux.

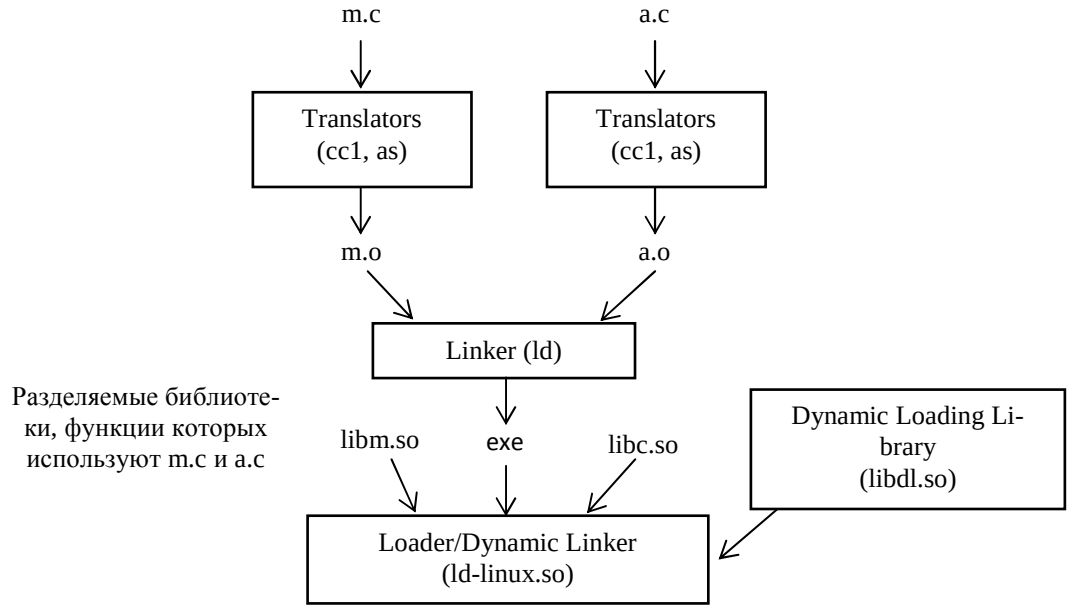

Рис. 1

Динамическая компоновка и загрузка программ в Linux. В ОС Linux существует 2 метода работы с динамическими библиотеками:

– динамическая компоновка;

– динамическая загрузка.

При использовании динамической компоновки ядро Linux сначала загружает образ программы в виртуальное адресное пространство создаваемого процесса, а затем передает управление динамическому компоновщику (ld-linux.so) [3], [5]. Далее динамический компоновщик выполняет необходимые перемещения (relocations), включая совместно используемые объекты, на которые ссылаются требуемые разделяемые библиотеки [4].

В случае динамической загрузки приложение само “решает”, какие библиотеки необходимо загрузить, после чего вызывает библиотечные функции. В этом процессе используется библиотека динамической загрузки libdl.so, которая тоже загружается при помощи ld-linux.so [3], [6].

Как видно, данные компоненты системы работают с уже готовыми исполняемыми файлами и не требуют вмешательства в исходный код.

Релокация символов для профилирования. Работа механизмов динамической компоновки и загрузки в общем случае состоит из загрузки динамической библиотеки и выполнения процесса перемещения символов (symbol relocation), или релокации. Релокация символов выполняется при работе программы. Во время этого процесса ссылки на символы (переменные или функции) из разделяемой библиотеки заменяются фактическими адресами расположения символов в виртуальной памяти [4], [5].

Реализации релокации в ld-linux.so и libdl.so в своей сути похожи. Как libdl.so, так и ld-linux.so в своем составе имеют функцию, которая на вход принимает имя релоцируемой функции, а на выходе выдает адрес функции в виртуальной памяти. Для ld-linux.so такой функцией является _dl_fixup, а для libdl.so – dlsym.

Процесс релокации может быть напрямую модифицирован для выполнения неинвазивного профилирования. Сама модификация состоит в разработке некоторого механизма, который позволит заменить адрес, передаваемый указанными функциями, адресом служебного кода, необходимого для профилирования.

Реализация механизма перехвата вызовов. Перехват вызовов профилируемых функций является основой реализации методики неинвазивного профилирования. В общем случае данный механизм состоит из двух частей:

1. Код перенаправления вызова (редиректор, redirector).

2. Функция-обертка (wrapper).

Редиректор представляет собой набор машинных кодов для следующих ассемблерных инструкций (пример для платформы x86):

push $fcnPtr

jmp $wrapper_addr

Функции редиректора:

1. Сохранение адреса профилируемой функции в стеке программы.

2. Безусловный переход по адресу функции-обертки.

Работа редиректоров показана на рис. 2.

Цифрами на данной схеме обозначены следующие действия:

1. Передача адреса редиректора с помощью функции релокации (dlsym либо _dl_fixup).

2. Передача управления коду редиректора вместо функции.

3. Переход по адресу функции-обертки с передачей адреса профилируемой функции в стеке программы.

4. Работа функции-обертки по профилированию и вызову интересующей функции.

5. Возврат в вызывающую программу.

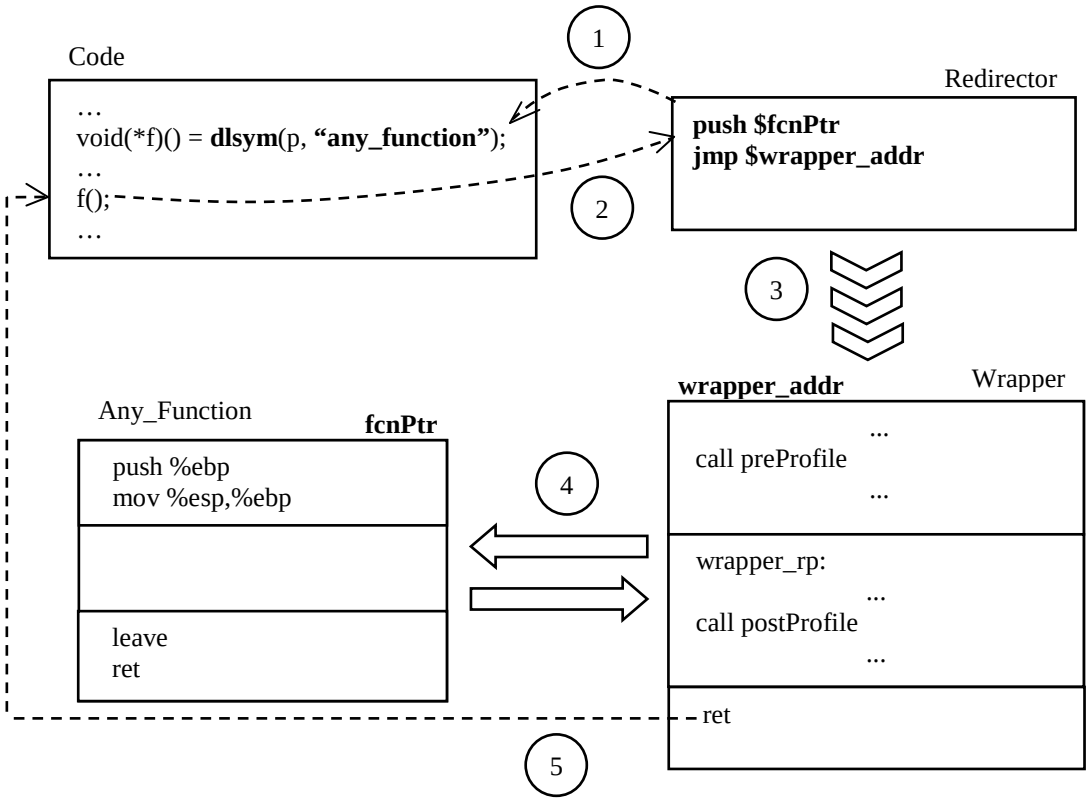

Рис. 2

Особенность кода перенаправления вызовов состоит в том, что он создается индивидуально для каждой профилируемой функции и размещается непосредственно в сегменте данных виртуальной памяти процесса. При этом средства ОС дают возможность сделать данный набор инструкций исполняемым, что позволяет ему выполняться при вызове вместо интересующей функции.

Функция-обертка (wrapper) служит для вызова профилируемой функции и измерения времени ее работы при передаче управления от редиректора.

Работа данной функции состоит из следующих стадий:

1. Сохранение состояния всех регистров в стеке программы и вызов функции предварительного профилирования.

2. Функция предварительного профилирования создает специальную структуру – контекст вызова, в которой сохраняется адрес профилируемой функции, адрес возврата и время начала работы. Контекст вызова на время выполнения профилируемой функции помещается в локальное хранилище потока выполнения (Thread-Local Storage, TLS) [4], [6]. Это позволяет избежать одновременного обращения к данному участку памяти при профилировании многопоточных приложений.

3. Wrapper заменяет адрес возврата профилируемой функции, находящийся в стеке процесса, на адрес некоторой метки внутри себя и восстанавливает сохраненное состояние регистров.

4. Производится безусловный переход по адресу профилируемой функции.

5. Профилируемая функция выполняет все необходимые действия и возвращает управление в функцию-обертку, производя переход по замененному адресу возврата.

6. Wrapper снова сохраняет все регистры в стеке и вызывает функцию завершающего профилирования.

7. Функция завершающего профилирования восстанавливает контекст вызова из TLS, измеряет время окончания работы и по полученным данным вычисляет определенные характеристики профилирования (длительность выполнения функции и количество вызовов). При этом в функцию-обертку передается настоящий адрес возврата профилируемой функции.

8. Wrapper помещает полученный адрес возврата в стек и восстанавливает сохраненные регистры.

9. Производится возврат в вызывающую программу.

Схематично принцип работы механизма “оборачивания” функций показан на рис. 3.

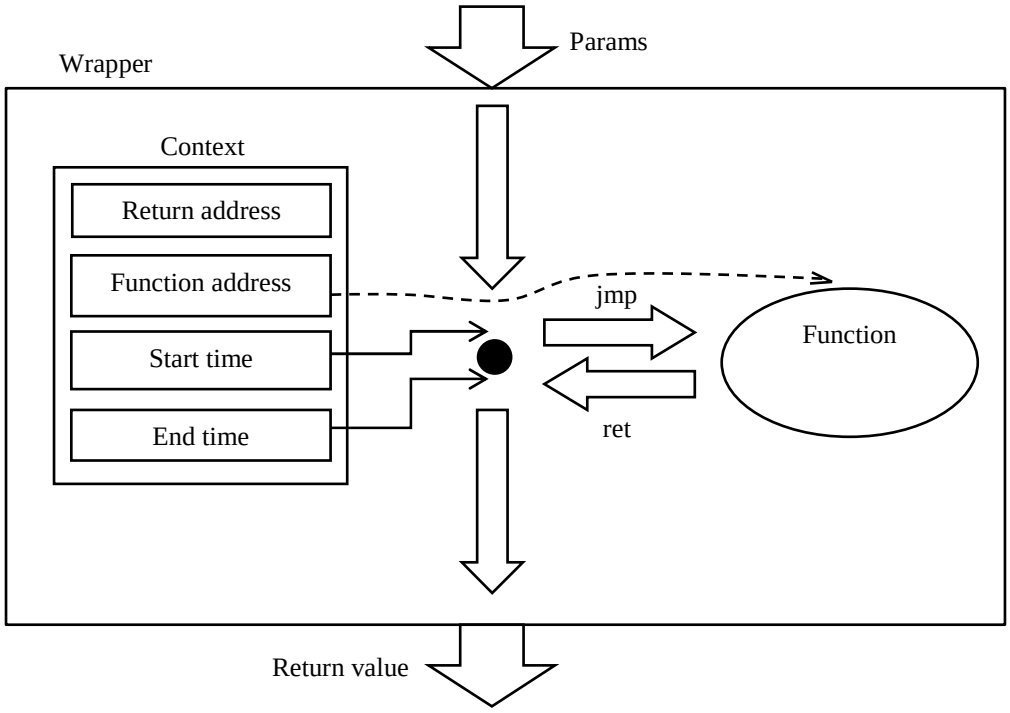

Рис. 3

Главная особенность механизма “оборачивания” функций – это то, что при профилировании не нарушается состояние регистров процессора и стека программы. Это позволяет работать с большим множеством функций, написанных на языках C/C++, независимо от архитектуры, соглашения о вызовах либо способа оптимизации программ [4], [6].

Измерение времени. Как уже отмечалось, задача измерения времени ложится на функцию-обертку. При этом для наиболее точного измерения времени целесообразно применять непосредственное чтение состояния счетчика тактов процессора (Time Stamp Counter, TSC). Данный подход производит измерения с максимальной разрешающей способностью.

Однако использование такого метода измерений в многопроцессорной системе имеет побочный эффект, связанный с немонотонностью работы отдельно взятого процессорного таймера и асинхронностью нескольких таймеров вместе.

В связи с этим общие результаты, получаемые при профилировании с таким измерением времени, следует считать относительными. Иными словами, более точные результа-

ты профилирования можно получить, измеряя время работы функций для двух состояний системы: нагруженного и ненагруженного.

Таким образом, в результате исследования и разработки методики неинвазивного профилирования были достигнуты следующие результаты:

• Разработан инструмент профилирования вызовов функций из разделяемых библиотек, основанный на использовании динамического компоновщика и библиотеки динамической загрузки в ОС Linux.

• Протестирована работа разработанного средства при профилировании системных утилит, а также сторонних приложений в операционной системе Debian GNU/Linux 6.0.

Дальнейшее развитие разработанного инструмента и совершенствование механизмов профилирования будет включать:

– решение проблем измерения времени в многопроцессорных системах;

– разработку решения для других дистрибутивов операционной системы Linux.

СПИСОК ЛИТЕРАТУРЫ

1. Оптимизация ПО. Сборник рецептов / Р. Гербер, А. Бик, К. Смит, К. Тиан. СПб.: Питер, 2010.
2. Liu H. H. Software Performance and Scalability. A Quantitative Approach. Hoboken, New Jersey: John Wiley & Sons, Inc., 2009.
3. Matthew N., Stones R. Beginning Linux Programming. N. Matthew, Indianapolis, Indiana: Wiley Publishing, 2008.
4. Kerrisk M. The Linux Programming Interface. San-Francisco: No Starch Press, 2010.
5. Levine J. R. Linkers and Loaders. Waltham, Massachusetts: Morgan Kauffman, 2000.
6. Love R. Linux System Programming. Sebastopol, California: O'Reilly Media, 2007.

E. M. Ryabikov, M. M. Zaslavskiy, K. V. Krinkin

METHOD OF NONINVASIVE PROFILING OF LINUX SHARED LIBRARIES

This paper discusses the technique of software profiling in Linux that solves problems of the impossibility of performance analysis without of source code or root privileges in operating system. Lists basic principles of the technique. Describes the implementation of mechanisms for profiling on the basis of the developed method.

Software optimization, Linux, ELF, shared libraries, dynamic linking, dynamic loading, noninvasive profiling

УДК 519.7+681.51

Т. Л. Качанова, К. А. Туральчук

МЕТОД ОТБРАКОВКИ НЕГОДНЫХ ПОЛУПРОВОДНИКОВЫХ ИЗДЕЛИЙ НА БАЗЕ ТЕХНОЛОГИЙ ФИЗИКИ СИСТЕМ

Разработан набор правил, позволяющих отбраковывать негодные изделия по значениям всех параметров производственного процесса. Базой для построения правил служит системное знание, автоматически извлекаемое технологиями физики открытых систем из данных измерений процесса.

Физика систем, технологии генерации системного знания, правила классификации, модель идентификации

В области производства полупроводниковых изделий важной задачей является идентификация годных и негодных изделий. Современные полупроводниковые производства оснащены оборудованием, позволяющим осуществлять мониторинг на разных этапах технологического процесса. Число измеряемых показателей может достигать сотен и тысяч.