Оглавление

Введение

Работая с операционной системой Linux [1], пользователю может потребоваться отслеживать её загруженность. Для обнаружения и предотвращения сбоев необходимо иметь хорошую систему мониторинга, которая будет анализировать работу операционной системы. Данный курсовой проект посвящен исследованию структур ядра, хранящим информацию о процессах в системе и памяти, и способам перехвата системных вызовов ядра с их последующим логированием.

Целью данной курсовой работы является разработка загружаемого модуля ядра, предоставляющего информацию о загруженности системы: количество системных вызовов за выбранный промежуток времени, количество выделенной памяти в текущий момент, статистика по процессам и в каких состояниях они находятся.

Для достижения поставленной цели необходимо выполнить следующие задачи:

- изучить структуры и функции ядра, которые предоставляют информацию о процессах и памяти;
- проанализировать существующие подходы к перехвату системных вызовов и выбрать наиболее подходящий;
- реализовать загружаемый модуль ядра.

1 Аналитическая часть

В данном разделе будут рассмотрены и проанализированы:

- различные подходы к трассировке ядра и перехвату функций;
- структуры и функции ядра, предоставляющие информацию о процессах и памяти;
- основные принципы загружаемых модулей ядра;
- способы получения пользователем информации из ядра.

1.1 Трассировка ядра

Трассировка ядра – получение информации о том, что происходит внутри работающей системы. Для этого используются специальные программные инструменты, регистрирующие все происходящие события в системе.

Такие программы могут одновременно отслеживать события как на уровне отдельных приложений, так и на уровне операционной системы. Полученная в ходе трассировки информации может оказаться полезной для диагностики и решения системных проблем.

Во время трассировки записывается информация о событиях, происходящих на низком уровне. Их количество исчисляется сотнями и даже тысячами.

Далее будут рассмотрены существующие различные подходы к трассировке ядра и перехвату вызываемых функций, и выбран наиболее подходящий для реализации в курсовой работе.

1.1.1 Linux Security Module

`Linux Security Module` (LSM) [2] – это специальный интерфейс, созданный для перехвата функций. В критических местах кода ядра расположены вызовы security-функций, которые вызывают коллбеки (англ.

callback [3]), установленные security-модулем. Данный модуль может изучать контекст операции и принимать решение о её разрешении или запрете [2].

Особенности рассматриваемого интерфейса:

- security-модули являются частью ядра и не могу быть загружены динамически;
- в стандартной конфигурации сборки ядра флаг наличия LSM неактивен - большинство уже готовых сборок ядра не содержут внутри себя интерфейс LSM;
- в системе может быть только один security-модуль [2].

Таким образом, для использования Linux Security Module необходимо поставлять собственную сборку ядра Linux, что является трудоёмким вариантом – как минимум, придётся тратить время на сборку ядра. Кроме того, данный интерфейс обладает излишним функционалом (например решение о блокировке какой-либо операции), который не потребуется в написании разрабатываемого модуля ядра.

1.1.2 Модификация таблицы системных вызовов

1.1.3 kprobes

1.1.4 Kernel tracepoints

`Kernel tracepoints` [4] – это фреймворк для трассировки ядра, реализованный через статическое инструментирование кода. Большинство важных функций ядра статически инструментировано – в теле функций добавлены вызовы функций фреймворка рассматриваемого фреймворка.

Особенности рассматриваемого фреймворка:

- минимальные накладные расходы – необходимо только вызвать функцию трассировки в необходимом месте;

- отсутствие задокументированного API;
- не все функции ядра статически инструментированны;
- не работает, если ядро не сконфигурировано должным образом [5].

1.1.5 ftrace

`ftrace` [6] – это фреймворк для трассировки ядра на уровне функций, реализованный на основе ключей компилятора `-pg` [7] и `mfentry` [7]. Данные функции вставляют в начало каждой функции вызов специальной трассировочной функции `mcount()` или `__fentry()__`. В пользовательских программах данная возможность компилятора используется профилировщиками, с целью отслеживания всех вызываемых функций. В ядре эти функции используются исключительно для реализации рассматриваемого фреймворка.

Для большинства современных архитектур процессора доступна оптимизация: динамический `frace` [7]. Ядро знает расположение всех вызовов функций `mcount()` или `__fentry()__` и на ранних этапах загрузки ядра подменяет их машинный код на специальную машинную инструкцию `NOP` [8], которая ничего не делает. При включении трассировки, в нужные функции необходимые вызовы добавляются обратно. Если `ftrace` не используется, его влияние на производительность системы минимально.

Особенности рассматриваемого фреймворка:

- имеется возможность перехватить любую функцию;
- перехват совместим с трассировкой;
- фреймворк зависит от конфигурации ядра, но, в популярных конфигурациях ядра (и, соответственно, в популярных образах ядра) установлены все необходимые флаги для работы;

Вывод

В таблице 1.1 приведено сравнение приведенных выше технологий трассировки ядра.

Название	Дин. загрузка	Перехват любых функций	Любая конфигурация ядра	Простота реализации	Наличие документации
Linux Security Module					
Модификация таблицы системных вызовов					
kprobes					
kernel tracepoints					
ftrace					

Таблица 1.1: Сравнение технологий, позволяющих трассировать ядро

В ходе анализа подходов к перехвату функций, был выбран фреймворк `ftrace`, так как он позволяет перехватить любую функцию зная лишь её имя, может быть загружен в ядро динамически и не требует специальной сборки ядра и имеет хорошо задокументированный API.

1.2 Информация о процессах и памяти

1.3 Загружаемые модули ядра

Одной из особенностей ядра Linux является способность расширения функциональности во время работы, без необходимости компиляции ядра заново. Таким образом, существует возможность добавить (или убрать) функциональность в ядро можно когда система запущена и работает. Часть кода, которая может быть добавлена в ядро во время работы, называется

модулем ядра. Ядро Linux предлагает поддержку большого числа классов модулей. Каждый модуль – это подготовленный объектный код, который может быть динамически подключен в работающее ядро, а позднее может быть выгружен из ядра.

Каждый модуль ядра сам регистрирует себя для того, чтобы обслуживать в будущем запросы, и его функция инициализации немедленно прекращается. Задача инициализации модуля заключается в подготовке функций модуля для последующего вызова. Функция выхода модуля вызывается перед выгрузкой модуля из ядра. Функция выхода должна отменить все изменения, сделанные функций инициализации, освободить захваченные в процессе работы модуля ресурсы.

Возможность выгрузить модуль помогает сократить время разработки – нет необходимости перезагрузки компьютера при последовательном тестировании новых версий разрабатываемого модуля ядра.

Модуль связан только с ядром и может вызывать только те функции, которые экспортированы ядром.

1.4 Получение данных из ядра

Вывод

В данном разделе:

2 Конструкторская часть

> В этом разделе

Вывод

3 Технологическая часть

> В этом разделе

Вывод

Заключение

Литература

[1] Linux - Operating System [Электронный ресурс]. Режим доступа: https://www.linux.org/ (дата обращения: 08.11.2021).

[2] Linux Security Module Usage [Электронный ресурс]. Режим доступа: https://www.kernel.org/doc/html/v4.16/admin-guide/LSM/index.html (дата обращения: 08.11.2021).

[3] Колбэк-функция – Глоссарий – MDN Web Docs [Электронный ресурс]. Режим доступа: https://developer.mozilla.org/ru/docs/Glossary/Callback_function (дата обращения: 08.11.2021).

[4] Using the Linux Kernel Tracepoints [Электронный ресурс]. Режим доступа: https://www.kernel.org/doc/html/latest/trace/tracepoints.html (дата обращения: 08.11.2021).

[5] Механизмы профилирования Linux – Habr [Электронный ресурс]. Режим доступа: https://habr.com/ru/company/metrotek/blog/261003/ (дата обращения: 08.11.2021).

[6] Using ftrace | Android Open Source Project [Электронный ресурс]. Режим доступа: https://source.android.com/devices/tech/debug/ftrace (дата обращения: 08.11.2021).

[7] Трассировка ядра с ftrace – Habr [Электронный ресурс]. Режим доступа: https://habr.com/ru/company/selectel/blog/280322/ (дата обращения: 08.11.2021).

[8] NOP: No Operation (x86 Instruction Set Reference) [Электронный ресурс]. Режим доступа: https://c9x.me/x86/html/file_module_x86_id_217.html (дата обращения: 08.11.2021).