САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ОТЧЕТ О НАУЧНО-ИССЛЕДОВАТЕЛЬСКОЙ РАБОТЕ

Шаго Павел Евгеньевич

22.Б07-ПУ, 01.03.02 Прикладная математика и информатика

Планировщики 2

Научный руководитель

к.ф.-м.н., доцент Корхов В. В.

Санкт-Петербург

10 мая 2025 г.

Содержание

Введение

Данная работа продолжает исследования, связанные с планировщиками, и фокусируется на конкретных реализациях, архитектурных решениях и сравнительном анализе. Помимо этого, она углублённо рассматривает более сложные теоретические аспекты, касающиеся алгоритмов планирования. Ключевые из них можно аннотировать следующим образом: алгоритмы честного планирования, NP-трудность задачи планирования, а также сведения задач планирования к известным NP-трудным задачам FINISHME.

Несмотря на то что в предыдущей работе Планировщики [1] было показано, что программные планировщики позволяют решать широкий спектр задач самого разного характера, в настоящем исследовании основное внимание сосредоточено на планировщике операционной системы Linux — Completely Fair Scheduler (CFS)/Earliest Eligible Virtual Deadline First (EEVDF).

Выбор CFS/EEVDF обусловлен его высокой степенью развития по сравнению с альтернативами, а также эффективностью в решении задач планирования как отдельных процессов, так и их групп в рамках операционных систем (ОС) на базе ядра Linux. В контексте CFS/EEVDF подробно проанализированы его архитектура и принцип работы. Особое внимание уделено причинам, по которым в Linux произошла замена алгоритма выбора следующей задачи: вместо ранее использовавшегося, хорошо зарекомендовавшего себя подхода в CFS, описанного в [1], была внедрена модель виртуальных дедлайнов, реализованная и описанная в EEVDF.

Дополнительно была рассмотрена новейшая подсистема sched_ext (scheduler extensions), которая предоставляет интерфейсы для более точной адаптации ОС к специализированным сценариям использования – например, планирование с учётом перспективы интерфейса на экране пользователя. Такой механизм позволяет повысить эффективность решения задачи планирования за счёт использования дополнительной информации о контексте исполнения.

Исследование алгоритмов планирования остаётся критически важным направлением в области системного программирования, поскольку именно планировщик определяет эффективность использования ресурсов, отзывчивость и предсказуемость поведения операционной системы. Совершенствование планировщиков напрямую влияет на производительность вычислительных систем и на качество взаимодействия пользователя с программным обеспечением.

1 Продвинутые алгоритмы планирования

1.1 Введение

NP трудная задача

2 Earliest eligible virtual deadline first (EEVDF)

2.1 Введение

EEVDF — планировщик потоков в ядре Linux, который в 2023 году (с версии ядра Linux 6.6) вытеснил CFS в качестве планировщика по умолчанию.

CFS на тот момент использовался в ядре 16 лет, то есть ровно половину всего времени существования ядра Linux. С результатом работы CFS косвенно сталкивался практически любой человек из-за широкого использования Linux в различных программно-аппаратных комплексах. Это решение поражает, казалось бы CFS за такой срок уже доведен до идеала, поэтому причина замены достаточно интересна.

На самом деле это не замена — это улучшение CFS, EEVDF теперь является алгоритмом выбора следующей задачи

Теоритическая основа планировщика EEVDF описана в статье 1996 года Earliest Eligible Virtual Deadline First : A Flexibleand Accurate Mechanism for Proportional Share Resource Allocation написанной Ионом Стоикой и Хусейном Абдель-Вахабом. Сама же реализация EEVDF в Linux выполнена Инго Молнаром (автор CFS) и его коллегой по компании Red Hat Питером Зейлстрой. В основном реализация совпадает со статьей, хотя и несколько отличается, по теме отличий я также предлагаю подглаву FIXME.

2.2 Рассмотр причин отказа от CFS

2.3 Погружение в алгоритм работы

2.4 Отличие реализации от статьи

3 sched_ext

3.1 Введение

sched_ext (scheduler extensions) — это инфраструктура ядра Linux, позволяющая описывать планировщик с помощью набора BPF-программ (поэтому sched_ext также называют BPF scheduler). Также можно встретить альтернативное название — ECX (Extensible Scheduler Class).

sched_ext предоставляет возможность реализовывать собственные алгоритмы планирования, которые могут опираться на дополнительную информацию о планируемых сущностях. С помощью BPF-событий такие планировщики передают ядру необходимые состояния для управления задачами.

3.2 Основа sched_ext — BPF

BPF (Berkeley Packet Filter) — это специальный байт-код, исполняемый встроенной в ядро виртуальной машиной, который позволяет добавлять функциональность в ядро из пользовательского пространства. Изначально он был разработан для динамического добавления правил низкоуровневой фильтрации пакетов сетевыми картами внутри ядра Linux.

Со временем BPF значительно эволюционировал. Современная версия BPF является синонимом к **eBPF** (extended BPF), изначальный же вариант BPF сейчас называется cBPF (classic BPF) и практически не используется. В наименованиях существует некоторая асимметрия: в сообществе программистов чаще используется обозначение BPF [16, 17, 18], тогда как в научной литературе предпочитается термин eBPF [3, 5, 7, 8, 11, 12, 13, 14, 15].

BPF предоставляет ограниченный набор инструкций и спроектирован так, чтобы его можно было легко верфицировать, то есть чтобы до выполнения можно было быстро узнать нет ли в данном коде небезопасных последовательностей инструкций. BPF зародился в сообществе Linux, но сейчас также был портирован на Windows [19].

BPF может быть использован (помимо планировщика) для доработки различных систем ядра, некоторые из которых: tc, XDP, kprobes, uprobes, tracepoints, LSM, perf events, seccomp-bpf.

BPF считается в сообществе революционной технологией [3, 5, 8, 12], но есть также и критика [7].

3.3 Оптимизация

Использование BPF позволяет вынести алгоритм

3.4 За гранью оптимизации

Данный раздел представляет собой обзор статьи Concurrency Testing in the Linux Kernel via eBPF [5].

Планировщик, построенный с использованием sched_ext, может быть полезен не только для повышения производительности систем с заранее известной спецификой. В [5] рассматривается применение специализированных алгоритмов планирования типа Controlled Concurrency Testing (CCT) для проведения fuzzing-тестирования параллельного кода в ядре Linux.

Fuzzing-тестирование — это распространённый подход к тестированию программного решения путем отправления в него случайно сгенерированного входа. Такой подход позволяет значительно увеличить одну из главных метрик в тестировании программного обеспечения – покрытие (англ. coverage). Повышенное покрытие, в свою очередь, снижает вероятность ошибок в сложных и редко возникающих сценариях исполнения.

Подобные сценарии действительно существуют и нередко становятся причиной трудноуловимых сбоев. Их природа заключается в том, что некоторая последовательность команд исполняется в параллельной системе и лишь при определённом (достаточно маловероятном) порядке межпоточного взаимодействия возникает состояние, не предусмотренное разработчиком.

Авторы в [5] предлагают программное решение Lightweight Automated Concurrency-tester via EBPF (LACE) для проведения такого рода тестирования. Они выделяют четыре основные проблемы, характерные для существующих решений, которые призван устранить LACE: ограниченный контроль над процессом планирования, высокие накладные расходы, отсутствие совместимости с будущими версиями, слабая расширяемость.

Ограниченность контроля проявляется в том, что во многих тестовых пакетах точки планирования могут быть реализованы лишь в определённых местах, а управление исполнением часто сводится к вставке искусственных задержек. Такой подход не всегда обеспечивает достаточную точность для выявления всех потенциальных ошибок в параллельном исполнении.

Высокие накладные расходы связаны с тем, что некоторые существующие пакеты тестирования требуют запуска в режиме гипервизора, что значительно усложняет и утяжеляет процесс тестирования.

Отсутствие совместимости с будущими версиями ядра у существующих решений обусловлено их изначальной архитектурой, плохо адаптированной к быстрому темпу развития Linux. Эта проблема выражается в необходимости внесения масштабных изменений в инструменты тестирования при обновлении ядра. Например, для адаптации инструмента krace к современным версиям требуется применение патчей объёмом более 10000 строк, затрагивающих 105 файлов. Подобные объёмы изменений значительно усложняют сопровождение, усложняют автоматизацию и снижают практическую применимость таких решений в долгосрочной перспективе.

Ограниченная расширяемость объясняется тем, что алгоритмы планирования в существующих системах часто являются уникальными и жёстко встроены в ключевые компоненты своей реализации. Попытки расширить или привнести новые алгоритмы в кодовую базу оказываются крайне сложными и требуют глубокой экспертной подготовки в области низкоуровневого планирования. В результате на практике до сих пор широко применяются традиционные фреймворки, полагающиеся на поведение планировщика по умолчанию, несмотря на их ограниченную эффективность в выявлении ошибок исполнения.

LACE значительно смягчает или полностью решает эти проблемы, он предлагает собственный CCT планировщик написанный на eBPF в инфраструктуре sched_ext, а также точечную стратегию внедрения прерываний в код ядра. Кроме того, в LACE реализован двухфазный подход к мутациям, FAKEME при котором сначала изменяется структура сценария исполнения, а затем производится точечная модификация параметров для более глубокого исследования пространства состояний.

Если оценить решение на актуальных версиях ядра Linux, то можно получить следующие результаты: LACE обеспечивает на 38% больше охвата ветвлений по сравнению с SegFuzz, демонстрирует высокую эффективность в исследовании вариантов планирования, обеспечивая ускорение в 11.4 раза. Кроме того, с его помощью было обнаружено восемь новых ошибок в ядре, четыре из которых были исправлены, а ещё две подтверждены разработчиками.

Заключение

В данной работе были рассмотрены продвинутые алгортмы планирования. Были описаны наиболее актуальные решения, используемые для задач планирования в современных операционных системах, такие как CFS/EEVDF и sched_ext. Продолжающееся развитие данной области показывает, что решаемые проблемы актуальны и востребованны.

В работе преднамеренно было опущено обсуждение важности разработки более эффективных планировщиков в тех местах, где это допустимо в рамках формата отчета о научно-исследовательской работе. Это сделано по следующим причинам: во-первых, с момента публикации [1] не произошло качественно новых изменений в оценке значимости рассматриваемой темы – она остаётся исключительно актуальной, и повторение ранее изложенного не представляется целесообразным; во-вторых, данную работу лучше всего рассматривать как логичное продолжение ранее проведённого исследования, поэтому внимание к важности целесообразно уделить именно в заключении.

Разработка более эффективных алгоритмов планирования имеет ключевое значение в условиях постоянно растущих требований к производительности вычислительных систем. Современные задачи, такие как обеспечение предсказуемой задержки, адаптация к разнородным нагрузкам, энергоэффективность, а также гетерогенное устройство наиболее современных вычислительных машин требуют всё более точных и адаптивных решений. Поэтому исследования в этой области способствуют не только улучшению существующих реализаций, но и открывают путь к созданию новых, более универсальных и надёжных планировщиков.

Таким образом, проведённое исследование дополняет и расширяет картину, описанную в [1]. Оно укрепляет базу, необходимую для перехода на прикладной уровень: разработки более эффективных алгоритмов и создания программных решений, лучше соответствующих специфике решаемых задач. В будущем планируется сосредоточиться на подготовке образа тестовой системы с ядром Linux и написании пакета для сбора метрик качества обработки задач планирования. В дальнейшем работа в рамках этой темы может пойти в различных направлениях: от внесения инкрементальных патчей в CFS/EEVDF на основе показаний тест-пакета до создания специализированных sched_ext планировщиков, построенных на продвинутых алгоритмах для разнообразных прикладных задач

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Павел Е. Шаго Отчет о научно-исследовательской работе: Планировщики б. и. 2024

[2] Joseph Y-T. Leung Handbook of Scheduling: Algorithms, Models, and Performance Analysis CRC Press 2004

[3] Tang Qinan, Gao Xing, Li Guilin, Lin Juncong Optimizing Linux scheduling based on global runqueue with SCX IEEE International Conference SMC 2024

[4] Ion Stoica, Hussein Abdel-Wahab Earliest Eligible Virtual Deadline First : A Flexible and Accurate Mechanism for Proportional Share Resource Allocation Old Dominion University 1996

[5] Jiacheng Xu, Dylan Wolff, Han Xing Yi, Jialin Li, Abhik Roychoudhury Concurrency Testing in the Linux Kernel via eBPF arXiv 2025

[6] Andrew S. Tanenbaum, Herbert Bos MODERN OPERATING SYSTEMS Pearson Education 2023

[7] Zhong, Shawn Wanxiang, Jing Liu, Andrea Arpaci-Dusseau, Remzi Arpaci-Dusseau Revealing the Unstable Foundations of eBPF-Based Kernel Extensions Twentieth European Conference on Computer Systems 2025

[8] Liz Rice Learning eBPF O'Reilly Media, Inc 2023

[9] William Stallings Operating Systems: Internals and Design Principles Pearson Education 2020

[10] Michael J. Morrison Resource Management and Scheduling in Multitasking Operating Systems CRC Press 2017

[11] Daniel Hodges Scheduling at Scale : eBPF Schedulers with Sched_ext USENIX Association 2024

[12] Mores Konstantinos User-space guided memory management with eBPF NTUA 2025

[13] Xiaobo Zheng, Zihao Duan, Shiyi Li, Haojun Hu, Wen Xia F4: Fast, Flexible and stable-Fortified User-Space Thread Scheduling Framework International Conference on Networking, Architecture and Storage IEEE 2024

[14] Saito Shogo, Kei Fujimoto Port contention aware task scheduling for SIMD applications IEEE Access 2024

[15] Feitong Qiao, Yiming Fang, Asaf Cidon Energy-Aware Process Scheduling in Linux ACM SIGENERGY Energy Informatics 2024

[16] Torvalds L. Linux Kernel [Электронный ресурс] / L. Torvalds. – Режим доступа: https://github.com/torvalds/linux – Дата обращения 15.04.2025.

[17] sched-ext/scx [Электронный ресурс]. – Режим доступа: https://github.com/sched-ext/scx – Дата обращения 27.04.2025.

[18] lkml [Электронный ресурс]. – Режим доступа: https://lkml.org – Дата обращения 20.04.2025.

[19] eBPF for windows [Электронный ресурс]. – Режим доступа: https://github.com/microsoft/ebpf-for-windows – Дата обращения 01.05.2025.