

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
УНИВЕРСИТЕТ**

ОТЧЕТ О НАУЧНО-ИССЛЕДОВАТЕЛЬСКОЙ РАБОТЕ

Шаго Павел Евгеньевич

22.Б07-ПУ, 01.03.02 Прикладная математика и информатика

Планировщики

Научный руководитель

к.ф.-м.н., доцент Корхов В. В.

Санкт-Петербург

24 декабря 2024 г.

Содержание

Введение	3
1 Планировщики ОС	4
1.1 Переключение контекста и роль планировщика	4
2 Алгоритмы планирования	5
2.1 Алгоритм первый пришел первый вышел (FIFO)	5
2.2 Алгоритм кратчайшей работы следующей (SJN)	5
2.3 Алгоритм очереди с приоритетами	5
2.4 Алгоритм Раунд-Робин (RR)	6
2.5 Алгоритм многоуровневых очередей	6
2.6 Модификация алгоритма 2.5	6
3 Классификация планировщиков	7
3.1 Пакетные планировщики	7
3.2 Интерактивные планировщики	7
3.3 Планировщики реального времени	7
4 Уровни планирования	8
4.1 Долгосрочное планирование	8
4.2 Среднесрочное планирование	8
4.3 Краткосрочное планирование	8
5 Метрики планирования	9
5.1 Время поступления T_A	9
5.2 Время ожидания T_W	9
5.3 Время выполнения T_S	9
5.4 Время оборота T_R	9
5.5 Пример оценки метрик	9
6 Реализации планировщиков потоков в Linux	10
6.1 BFS (Brain Fuck Scheduler)	10
6.2 CFS (Completely Fair Scheduler)	10
7 Планирование в пользовательском пространстве	11
7.1 Планировщик в Go	11

7.2 Планировщик в userver	11
Заключение	12
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	13

Введение

С усложнением систем построенных человеком, появилась необходимость этими системами управлять. Значительные запросы на построение таких управляющих механизмов приходят из автоматизации и компьютеризации. Так, почти любое современное производство немыслимо без компьютера или микроконтроллера, который, например, управляет станком.

Крайне распространено Unix-подобное ядро Linux для операционных систем (ОС). Этот компонент содержит огромное количество планировщиков, среди которых 3 различных реализации планировщика потоков (процессов) ОС. Однако кроме них существуют также и специализированные решения, например, планировщики сетевого трафика или ввода-вывода, всего в ядре Linux содержится более 20 различных планировщиков и более 6 из них работают постоянно практически на всех системах.

Улучшения планировщиков имеют особую важность, поскольку они сразу же затрагивают большое число систем которые их используют. Такие улучшения несомненно приводят к значительному повышению общей производительности всех систем.

К планировщикам необходимо предъявлять различные требования в зависимости от задач, которые они выполняют. Например, при работе с потоками важно учитывать ограничения необходимые для реализации напрямую взаимодействующих с планировщиком клиентов. Так, для реализации сильных семафоров необходимо накладывать на планировщик дополнительное условие (ограничение) описывающее его работу. В системах реального времени планировщики должны гарантировать выполнение задач в строго определенных временные интервалы, что требует высокой предсказуемости и низкой задержки при переключении контекста. Кроме того, в целом мы стремимся к тому, чтобы планировщики обладали минимальными затратами на выполнение (англ. runtime cost) и высокой эффективностью, что позволяет оптимально использовать доступные ресурсы системы и обеспечивать высокую производительность при минимальных затратах.

Настоящая работа посвящена изучению современных подходов к разработке и оптимизации планировщиков, включая анализ их преимуществ, недостатков и возможных сфер применения. Основное внимание уделено вопросам моделирования задач планирования, выбора алгоритмов и оценки их производительности.

1 Планировщики ОС

1.1 Переключение контекста и роль планировщика

Переключение контекста (англ. context switch) — это процесс сохранения состояния текущего выполняющегося процесса и восстановления состояния следующего процесса, который будет запущен на выполнение.

Это фундаментальная операция в многозадачных системах, позволяющая эффективно использовать процессорное время. Однако ключевой вопрос заключается в том, какой процесс выбрать следующим для выполнения. Ответ на этот вопрос дает алгоритм планирования, реализуемый в компоненте ядра операционной системы, известном как планировщик процессов.

2 Алгоритмы планирования

2.1 Алгоритм первый пришел первый вышел (FIFO)

FIFO (First in first out) — классический метод планирования, следует принципу «первым пришел — первым обслужен». Процессы обрабатываются в порядке их поступления. Этот алгоритм отвечает за непосредственное распределение процессорного времени между потоками. Планировщик определяет, какому потоку и на какой период времени будет выделен процессор. Этот уровень также включает координацию выполнения потоков на многоядерных или многопроцессорных системах, обеспечивая эффективное использование аппаратных ресурсов. Алгоритм прост в реализации, но редко применяется в чистом виде, поскольку не учитывает особенности задач и может приводить к неэффективному использованию ресурсов. Он лучше всего подходит для длительных, требовательных к вычислительным ресурсам процессов, которые требуют операций ввода-вывода.

2.2 Алгоритм кратчайшей работы следующей (SJN)

Алгоритм кратчайшей работы следующей (англ. shortest job next) выбирает для выполнения тот процесс, который требует наименьшего процессорного времени (англ. burst time). Одной из основных трудностей является оценка времени выполнения процесса, которая не всегда может быть точной. Вытесняющая модификация этого алгоритма сортирует процессы по оставшемуся времени выполнения, позволяя быстро завершать короткие процессы и эффективно снижать общее время оборота.

2.3 Алгоритм очереди с приоритетами

Приоритетное планирование назначает каждому процессу числовой приоритет, определяющий его порядок выполнения. Высокий приоритет (низкое числовое значение) означает более раннее выполнение. Однако этот метод сталкивается с проблемой старва (голодания), когда процессы с низким приоритетом не получают процессорного времени из-за непрерывного выполнения процессов с высоким приоритетом. Для решения этой проблемы часто используется механизм старения, который постепенно повышает приоритет процессов, долго находящихся в очереди на исполнение.

2.4 Алгоритм Раунд-Робин (RR)

Алгоритм Round Robin (RR) основан на вытесняющей многозадачности и предоставляет каждому процессу фиксированный квант времени (q) для выполнения. После истечения этого времени процесс вытесняется и помещается в конец очереди. Этот метод обеспечивает равномерное распределение процессорного времени и высокую отзывчивость системы.

Однако выбор размера кванта времени критичен: слишком большой квант приводит к поведению, схожему с FIFO, а слишком маленький увеличивает накладные расходы на переключение контекста. Также Раунд-Робин может быть неэффективен для процессов с интенсивным вводом-выводом.

2.5 Алгоритм многоуровневых очередей

Метод многоуровневых очередей классифицирует процессы по различным критериям и распределяет их по нескольким очередям с разными уровнями приоритетов. Внутри каждой очереди может применяться свой алгоритм планирования, например, Раунд-Робин или FIFO. Этот подход обеспечивает гибкость и позволяет оптимизировать время отклика для критически важных процессов, одновременно эффективно используя ресурсы для фоновых задач.

2.6 Модификация алгоритма 2.5

Модификация этого метода с обратной связью позволяет динамически изменять приоритеты процессов на основе их поведения, предотвращая старвацию и улучшая общую производительность системы. Например, процессы, долго ожидающие выполнения, могут получать повышение приоритета, а активно работающие процессы — понижаться.

3 Классификация планировщиков

3.1 Пакетные планировщики

Пакетные планировщики предназначены для выполнения длительных задач, требующих значительных вычислительных ресурсов. Частые прерывания процессов в таких планировщиках не предусмотрены, что позволяет эффективно использовать процессорное время для интенсивных вычислений.

3.2 Интерактивные планировщики

Интерактивные планировщики стремятся минимизировать время отклика, обеспечивая отзывчивость пользовательского интерфейса на действия пользователя, такие как перемещение курсора мыши или нажатие клавиш. Это особенно важно для персональных компьютеров, где скорость реакции системы напрямую влияет на удобство и эффективность работы.

3.3 Планировщики реального времени

Планировщики реального времени разработаны для систем, требующих соблюдения временных рамок выполнения задач (дедлайнов). Они часто используются в критически важных приложениях, где нарушение временных ограничений может привести к серьезным последствиям. В таких системах первостепенным является соблюдение дедлайнов, а другие показатели, такие как время отклика, могут быть менее значимыми.

4 Уровни планирования

4.1 Долгосрочное планирование

На долгосрочном уровне принимается решение о том, какие новые задачи будут добавлены в систему для исполнения. Это включает управление загрузкой задач в систему и контроль за количеством активных процессов.

4.2 Среднесрочное планирование

Среднесрочное планирование занимается управлением памятью, определяя, какие программы временно выгрузить во вторичную память для оптимизации доступного пространства и повышения эффективности работы системы.

4.3 Краткосрочное планирование

Краткосрочное планирование отвечает за непосредственное распределение процессорного времени между потоками. Планировщик определяет, какому потоку и на какой период времени будет выделен процессор. Этот уровень также включает координацию выполнения потоков на многоядерных или многопроцессорных системах, обеспечивая эффективное использование аппаратных ресурсов.

5 Метрики планирования

5.1 Время поступления T_A

Время поступления процесса — момент, когда процесс становится готовым к выполнению.

5.2 Время ожидания T_W

Время ожидания представляет собой период, в течение которого процесс ожидает возможности быть обработанным центральным процессором.

5.3 Время выполнения T_S

Время выполнения — период, в течение которого процесс активно исполняется на процессоре.

5.4 Время оборота T_R

Время оборота включает в себя общее время, затраченное процессом в системе, включая периоды ожидания и активного выполнения.

5.5 Пример оценки метрик

Рассмотрим пример для иллюстрации важности выбора порядка выполнения процессов. Допустим, одновременно поступают три процесса с различным временем выполнения.

□ есть 3 процесса и их burst (время исполнения одного процесса)

$$T_{BURST_1} = 30, T_{BURST_2} = 5, T_{BURST_3} = 5$$

Если мы будем выполнять сначала 1 потом 2 потом 3, то

$$T_{W_1} = 0, T_{W_2} = 30, T_{W_3} = 35 \implies T_{W_{AVG}} = 21\frac{2}{3}$$

А выполнять по порядку 3, 2, 1, то

$$T_{W_3} = 0, T_{W_2} = 5, T_{W_1} = 10 \implies T_{W_{AVG}} = 5$$

Таким образом видим, что изменение порядка выполнения может значительно снизить среднее время ожидания.

6 Реализации планировщиков потоков в Linux

6.1 BFS (Brain Fuck Scheduler)

BFS был разработан Кон Коливасом в 2009 году как альтернативный планировщик для ядра Linux, его задача предоставить планировщик с более простым алгоритмом, который не требует настройки эвристики или параметров настройки (англ. tuning parameters) для адаптации производительности к определенному типу вычислительной нагрузки. Коливас утверждал, что эти настраиваемые параметры сложны для понимания обычным пользователем, особенно в плане взаимодействия нескольких параметров друг с другом, и заявлял, что использование таких параметров настройки часто может привести к улучшению производительности в конкретном целевом типе вычислений ценой ухудшения производительности в общем случае ориентированный на уменьшение задержек и повышение отзывчивости системы, особенно в системах с однопоточными процессорами и низкими задержками.

BFS улучшает отзывчивость на настольных компьютерах Linux с менее чем 16 ядрами. В отличие от других планировщиков, BFS использует единую глобальную очередь для всех процессов, что упрощает управление и устраняет необходимость в сложных механизмах балансировки нагрузки между процессорами.

Каждому процессу присваивается виртуальный дедлайн, рассчитываемый на основе текущего времени, приоритета процесса и фиксированного интервала времени. Это позволяет планировщику определять порядок выполнения задач, отдавая предпочтение процессам с более ранними дедлайнами.

6.2 CFS (Completely Fair Scheduler)

7 Планирование в пользовательском пространстве

7.1 Планировщик в Go

7.2 Планировщик в userver

Заключение

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

- [1] Joseph Y-T. Leung Handbook of Scheduling: Algorithms, Models, and Performance Analysis CRC Press 2004
- [2] Andrew S. Tanenbaum, Herbert Bos MODERN OPERATING SYSTEMS Pearson Education 2023
- [3] Dror G. Feitelson, Larry Rudolph Job Scheduling Strategies for Parallel Processing Springer 2001
- [4] William Stallings Operating Systems: Internals and Design Principles Pearson Education 2020
- [5] Michael J. Morrison Resource Management and Scheduling in Multitasking Operating Systems CRC Press 2017
- [6] Arden Durell Go Scheduler Internals 2018
- [7] Thomas E. Anderson, Brian N. Bershad, Edward D. Lazowska, Henry M. Levy Scheduler activations: effective kernel support for the user-level management of parallelism ACM 1992
- [8] Xiaomin Zhu, Yabing Zha, Ling Liu, Peng Jiao General Framework for Task Scheduling and Resource Provisioning in Cloud Computing Systems IEEE 2016