ИНФОРМАТИКА И ИНФОРМАЦИОННЫЕ ПРОЦЕССЫ/INFORMATICS AND INFORMATION PROCESSES

DOI: https://doi.org/10.60797/itech.2025.6.2

СРАВНЕНИЕ АЛГОРИТМОВ ДИНАМИЧЕСКОГО УПРАВЛЕНИЯ ПАМЯТЬЮ ДЛЯ ОПЕРАЦИОННЫХ СИСТЕМ РЕАЛЬНОГО ВРЕМЕНИ

Научная статья

Исса Я.[1, *], Волкова О.Р.[2]

[1, 2] Московский государственный технологический университет «Станкин», Москва, Российская Федерация

* Корреспондирующий автор (yasseressa1990[at]gmail.com)

Аннотация

Динамическое управление памятью является важнейшим аспектом современных вычислительных систем, влияющим на производительность, результативность и использование ресурсов. В этом исследовании представлено всестороннее сравнение известных алгоритмов управления динамической памятью, включая First Fit, Best Fit, Worst Fit, Buddy System, распределение памяти в слотах и управление пулом памяти. Каждый алгоритм анализируется с точки зрения его механизмов выделения и разгрузки, фрагментации (как внутренней, так и внешней), эффективности использования памяти и производительности в различных сценариях. В исследовании рассматриваются компромиссы между скоростью, гибкостью и фрагментацией, дающие представление о пригодности каждого алгоритма для конкретных случаев использования, таких как системы реального времени, встраиваемые системы и вычисления общего назначения. Цель данного исследования — оценить эти алгоритмы и помочь разработчикам систем выбрать наиболее подходящую стратегию управления памятью для своих приложений, сбалансировав производительность и эффективность использования ресурсов.

Ключевые слова: управление памятью, распределение памяти, динамическое распределение памяти, алгоритмы распределения памяти, операционная система реального времени.

COMPARISON OF DYNAMIC MEMORY MANAGEMENT ALGORITHMS FOR REAL-TIME OPERATING SYSTEMS

Research article

Essa Y.[1, *], Volkova O.R.[2]

[1, 2] Moscow State Technological University "Stankin", Moscow, Russian Federation

* Corresponding author (yasseressa1990[at]gmail.com)

Abstract

Dynamic memory management is a critical aspect of modern computing systems that affects performance, efficiency, and resource utilisation. This study presents a comprehensive comparison of well-known dynamic memory management algorithms, including First Fit, Best Fit, Worst Fit, Buddy System, memory slot allocation, and memory pool management. Each algorithm is analysed in terms of its allocation and offloading mechanisms, fragmentation (both internal and external), memory efficiency and performance in different scenarios. The research examines the trade-offs between speed, flexibility and fragmentation, providing insight into the suitability of each algorithm for specific use cases such as real-time systems, embedded systems and general purpose computing. The aim of this study is to evaluate these algorithms and help system designers select the most appropriate memory management strategy for their applications, balancing performance and resource efficiency.

Keywords: memory management, memory allocation, dynamic memory allocation, memory allocation algorithms, real-time operating system.

Введение

Динамическое управление памятью является важнейшим аспектом современных вычислительных систем, влияющим на производительность, экономичность и использование ресурсов. В этом исследовании представлено всестороннее сравнение известных алгоритмов управления динамической памятью. Каждый алгоритм анализируется с точки зрения его механизмов выделения и разгрузки, фрагментации, эффективности использования памяти и производительности в различных сценариях. В исследовании рассматриваются компромиссы между скоростью, гибкостью и фрагментацией, дающие представление о пригодности каждого алгоритма для конкретных случаев использования, таких как системы реального времени, встраиваемые системы и вычисления общего назначения. Цель данного исследования — оценить эти алгоритмы и помочь разработчикам систем выбрать наиболее подходящую стратегию управления памятью для своих приложений, сбалансировав производительность и эффективность использования ресурсов.

Алгоритмы используемые в динамическом управлении памятью

2.1. First-Fit Алгоритм

First-Fit — это метод выделения памяти, используемый в операционных системах для выделения памяти процессу. Этот метод выполняет поиск по списку свободных блоков памяти, начиная с начала списка, до тех пор, пока не будет найден блок, который является достаточно большим, чтобы удовлетворить запрос памяти от процесса. Как только

подходящий блок найден, операционная система разбивает его на две части: ту часть, которая будет выделена процессу, и оставшийся свободным блок [1].

	размер	Выделено для	Потеря памяти после того, как процесс займет
Блок 1	100	P1	100 - 20 = 80
Блок 2	50	P4	50 - 40 = 10
Блок 3	30	-	-
Блок 4	120	P2	120 - 60 = 60
Блок 5	35	-	-

Рисунок 1 - First-Fit Алгоритм
DOI: https://doi.org/10.60797/itech.2025.6.2.1

Преимущества:

1. Простота. С точки зрения принципа реализации этого алгоритма, это простая и понятная идея и подходящий вариант для систем с ограниченными вычислительными ресурсами.

2. Скорость выделения памяти. Поиск начинается с начала списка, и при первом подходящем блоке выделяется память.

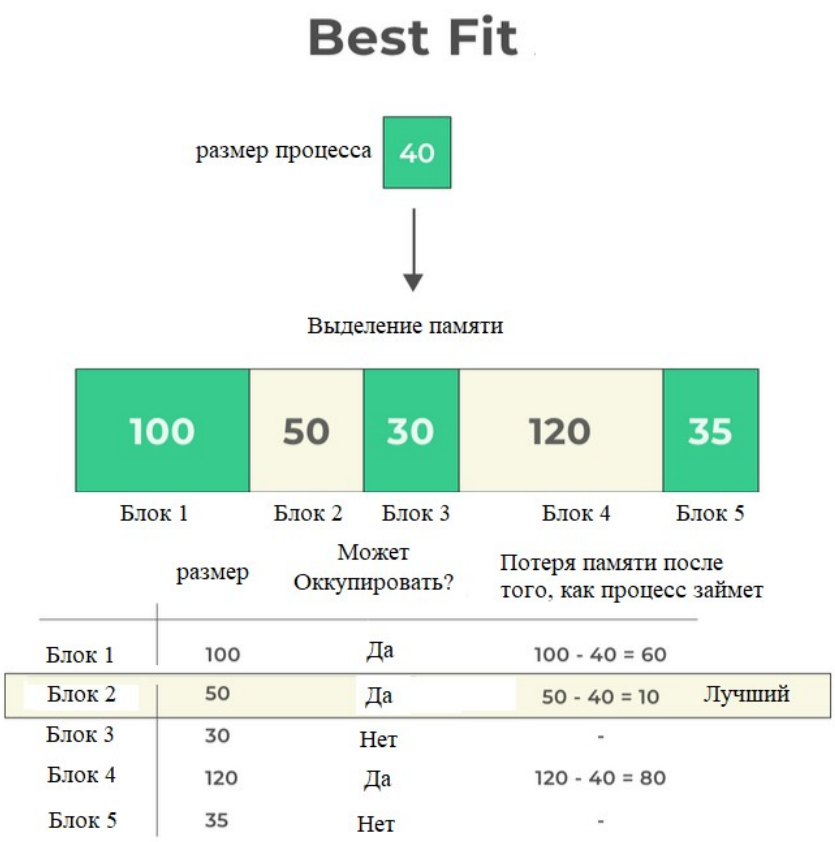
Недостатки:

1. Фрагментация. Очевидно, что этот алгоритм приводит к значительной фрагментации памяти, как внутренней, так и внешней.

2. Не подходит для больших процессов. Этот алгоритм выполняет поиск в блоках памяти по порядку, и поиск подходящей памяти для больших процессов может занять много времени, что приводит к медленному выделению памяти.

2.2. Best-Fit Алгоритм

Этот алгоритм выбирает соответствующий блок, наиболее близкий к размеру процесса, при этом блоки памяти упорядочиваются от наименьшего к наибольшему, и выполняется поиск соответствующего блока, и когда он найден, система разбивает его на два блока, один для процесса, а другой — для свободного блока [1].



	размер	Может Оккупировать?	Потеря памяти после того, как процесс займет	
Блок 1	100	Да	100 - 40 = 60	
Блок 2	50	Да	50 - 40 = 10	Лучший
Блок 3	30	Нет	-	
Блок 4	120	Да	120 - 40 = 80	
Блок 5	35	Нет	-	

Рисунок 2 - Best-Fit Алгоритм
DOI: https://doi.org/10.60797/itech.2025.6.2.2

Преимущества:

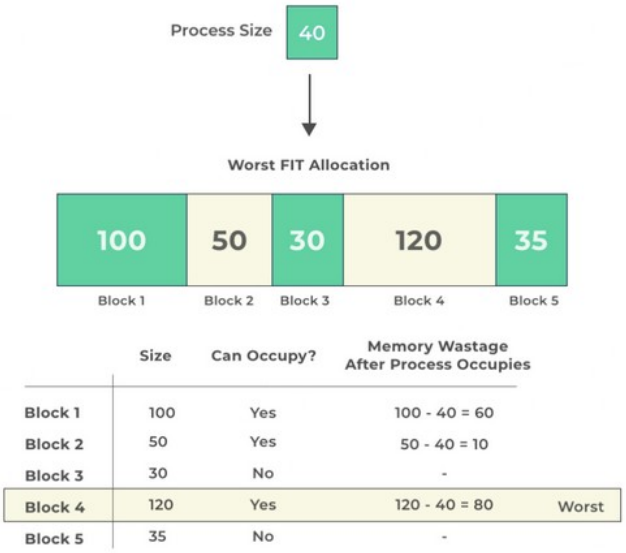
1. Эффективность использования памяти. Операционная система выделяет заданию минимально возможное пространство в памяти, что делает управление памятью очень эффективным.
2. Улучшено использование памяти.
3. Уменьшена фрагментация памяти.
4. Минимизирована внешняя фрагментация.

Недостатки:

1. Это медленный процесс. Проверка всей памяти для каждого задания приводит к очень медленной работе операционной системы. Для завершения работы требуется много времени.
2. Увеличение вычислительных затрат.
3. Может привести к увеличению внутренней фрагментации.
4. Может привести к замедлению выделения памяти.

2.3. Worst-Fit Алгоритм

В этом алгоритме выполняется поиск и резервирование самого большого блока памяти, даже если он намного превышает размер, требуемый процессом [1].



	Size	Can Occupy?	Memory Wastage After Process Occupies	
Block 1	100	Yes	100 - 40 = 60	
Block 2	50	Yes	50 - 40 = 10	
Block 3	30	No	-	
Block 4	120	Yes	120 - 40 = 80	Worst
Block 5	35	No	-	

Рисунок 3 - Worst-Fit Алгоритм
DOI: https://doi.org/10.60797/itech.2025.6.2.3

Преимущества. Поскольку этот процесс выбирает самый большой размер блока, следовательно, будет большая внутренняя фрагментация, которая будет не полезна для небольших процессов.

Недостатки. Требуется много времени, чтобы просмотреть всю память в поисках самого большого блока.

2.4. Система Buddy

Система buddy — это метод выделения памяти и управления ею, который использует размер блоков в степени двойки (например, 2, 4, 8, 16, 32, 64, 128, и т.д.), которые связаны в виде бинарной древовидной структуры, а листья представляют собой наименьшие блоки [2].

Вначале блоки свободны, затем, когда процесс запрашивает память, система ищет наименьший блок, подходящий для процесса, и, если размер блока превышает размер процесса, система разбивает блок на два «параллельных» блока одинакового размера. Система помечает один из вспомогательных блоков как выделенный и добавляет его в таблицу распределения памяти процесса, в то время как другой вспомогательный блок возвращается в пул свободной памяти и связывается обратно с бинарной древовидной структурой. Когда процесс освобождает память, система помечает блок как свободный и возвращает его в свой вспомогательный блок, если он свободен, затем объединенные блоки возвращаются в двоичное дерево.

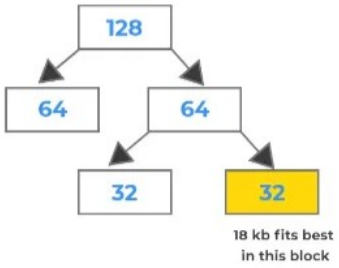


Рисунок 4 - Buddy Алгоритм
DOI: https://doi.org/10.60797/itech.2025.6.2.4

Типы бинарных систем:

1. Бинарная Buddy Система.
2. Фибоначчи Buddy Система.
3. Взвешенная Buddy Система.
4. Третичная Buddy Система.

Преимущества:

1. Эффективное использование памяти.
2. Быстрое выделение и разгружение памяти.
3. Обеспечивает оптимальное распределение памяти, поскольку выделяет правильный размер.

4. Может работать с большим количеством небольших блоков, что предотвращает фрагментацию.
5. Это очень полезно для встраиваемых систем.

Недостатки:

1. Это приводит к внутренней фрагментации.
2. Для этого требуется весь размер в степени, равной 2.
3. Это общее распределение памяти, возможно, бесполезное для некоторых приложений.

2.5. Slab-распределитель

Алгоритм Slab-распределителя — это метод, используемый для динамического управления памятью в операционных системах. Он разработан для обеспечения быстрого и эффективного выделения и освобождения памяти. Он идеально подходит для сценариев, в которых часто возникают запросы на выделение объектов фиксированного размера.

Алгоритм делит память на небольшие области (Slabs), и каждая область выделяется определенному классу объектов одинакового размера.

Каждая область имеет набор небольших «квадратиков» (Chunks), и каждый квадрат — это место, предназначенное для одного объекта [4].

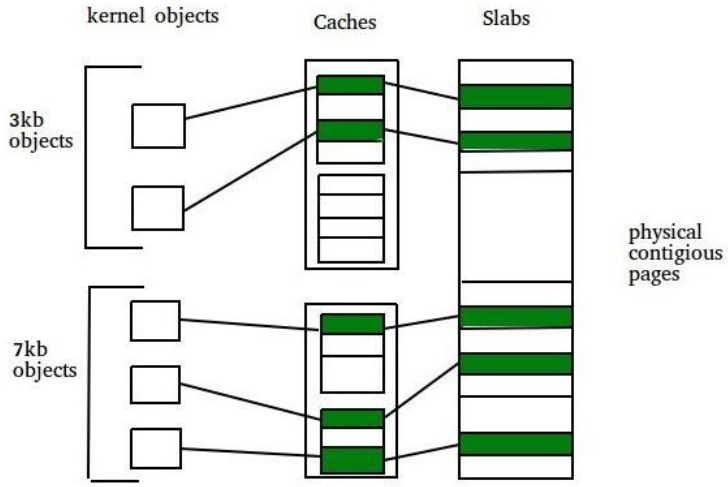


Рисунок 5 - Slot Алгоритм
DOI: https://doi.org/10.60797/itech.2025.6.2.5

Slab — это фактический контейнер данных, связанный с объектами определенного типа, содержащими кэш.

Кэш-память (cache): Кэш-память представляет собой небольшой объем очень быстрой памяти. Кэш состоит из одного или нескольких блоков данных. Для каждой уникальной структуры данных ядра существует отдельный кэш.

Состояния Slab:

– полная (Full) – Все объекты в таблице помечены как использованные;
– пустая (Empty) – Все объекты в таблице помечены как свободные;
– частичная (Partial) – Таблица состоит из двух элементов.

Преимущества:

1. Скорость:

– выделение и отмена ресурсов выполняются быстро, поскольку блоки предварительно разделены и инициализированы;

– нет необходимости в дорогостоящем поиске или разбиении, в отличие от традиционных распределителей (например, malloc).

2. Уменьшенная фрагментация:

– устраняется внешняя фрагментация, поскольку блоки всегда делятся на блоки фиксированного размера;
– минимизируется внутренняя фрагментация за счет выделения блоков для определенных типов объектов.

3. Эффективность кэширования: объекты одного типа группируются в одном блоке, что улучшает пространственную локализацию и производительность кэширования процессора.

4. Предсказуемая производительность: поскольку операции требуют фиксированных затрат, это обеспечивает стабильную производительность даже при высоком спросе.

Недостатки:

1. Затраты памяти: предварительное выделение памяти для блоков может привести к образованию неиспользуемых блоков, особенно для редко используемых типов объектов.

2. Сложность: управление кэшами и таблицами распределений усложняет реализацию по сравнению с более простыми распределителями.

2.6. Pooling (Пул памяти)

Пул памяти — это метод управления памятью, используемый для эффективного выделения памяти для объектов одинаковых размеров. Он организует память в предварительно выделенные пулы (или области), сокращая накладные расходы, связанные с частыми выделениями и освобождениями. Он часто используется в системах, критически важных для производительности, таких как операционные системы, приложения реального времени и высокочастотные торговые системы [3].

Типы пулов памяти:

1. Пулы фиксированного размера:

– все блоки в области пула имеют одинаковый размер;

– лучше всего подходит для объектов одинакового размера, таких как сетевые пакеты, строки базы данных или индексные узлы в файловой системе.

2. Пулы с переменным размером:

– пул управляет блоками нескольких размеров или поддерживает отдельные подпулы для блоков разных размеров;

– подходит для приложений, в которых требуются объекты разных размеров.

3. Локальные пулы потоков:

– пулы, предназначенные для определенных потоков в многопоточных системах;

– уменьшает конкуренцию и повышает производительность за счет исключения блокировок.

Рабочий процесс создания пула памяти:

1. Инициализация:

– пул памяти создается путем резервирования большого объема памяти;

– пул делится на блоки определенного размера, которые обычно выравниваются, чтобы избежать аппаратной неэффективности.

2. Распределение:

– когда требуется новый объект, менеджер пула находит свободный блок и помечает его как выделенный;

– если свободного блока не существует, в пул может быть добавлена дополнительная память или запрос может завершиться ошибкой.

3. Освобождение:

– когда объект больше не нужен, его блок помечается как свободный и возвращается в пул;

– блок может быть повторно использован для будущих распределений.

4. Уничтожение: когда пул больше не нужен, вся память освобождается сразу, что сокращает накладные расходы на отдельные операции освобождения.

Преимущества:

1. Быстрое выделение/разгружение памяти:

– предварительно выделенная память позволяет избежать накладных расходов на системные вызовы;

– простая бухгалтерия обеспечивает быструю работу.

2. Уменьшенная фрагментация:

– отсутствие внешней фрагментации, поскольку память расположена в блоках фиксированного размера;

– меньшая внутренняя фрагментация благодаря выделенным пулам для каждого размера.

3. Предсказуемая производительность. Это особенно важно для систем реального времени, где распределение/отмена ресурсов должна быть детерминированной.

4. Возможность повторного использования. Блоки перерабатываются, что снижает необходимость в повторном распределении ресурсов в системе.

5. Отладка пула памяти. Пулы памяти часто включают средства отладки, такие как защитные зоны, отравление и отслеживание выделений.

Недостатки:

1. Накладные расходы на предварительное выделение памяти:

– требуется предварительное выделение памяти, которое может использоваться не полностью;

– может привести к потере памяти, если оценки размера объекта неточны.

2. Сложная реализация. Управление пулами (например, обработка фрагментации или изменение размера) может быть сложным по сравнению с обычными распределителями.

3. Недостаток гибкости:

– пулы фиксированного размера могут неэффективно обрабатывать объекты разного размера;

– неэффективно для больших объектов или неравномерных схем распределения.

Сравнение алгоритмов

Таблица 1 - Сравнение алгоритмов

DOI: https://doi.org/10.60797/itech.2025.6.2.6

Алгоритм	Представление	Фрагментация	Сложность	Пригодность в режиме реального времени
First-Fit	Быстрый	Высокая	Низкая	Умеренно подходящий

Алгоритм	Представление	Фрагментация	Сложность	Пригодность в режиме реального времени
Best-Fit	Медленный	Низкая	Высокая	Умеренно подходящий
Worst-Fit	Медленный	Высокая	Средняя	Менее подходящий
Buddy	Быстрый	Средняя	Средняя	Высокая пригодность
Slab	Очень быстро	Низкая	Низкая	Высокая пригодность
Pooling	Очень быстро	Низкая	Низкая	Высокая пригодност

Объяснение особенностей:

1. Производительность (временная сложность) [5].

Производительность часто измеряется временем выделения и разгрузки ресурсов. Для простоты диапазоны можно разделить следующим образом:

– очень быстро: O(1) операции, обычно продолжающиеся постоянно;
– быстрый: практически равен (1) или O (по логарифму n);
– медленный: Линейный или хуже по времени поиска, O(n) или выше.

2. Фрагментация [6].

Фрагментация относится к использованию памяти и подразделяется на внешнюю и внутреннюю фрагментацию.

– низкая (0–20%);
– средняя (20–50%);
– высокая (50% и более).

3. Сложность (алгоритмическая сложность) [3].

Сюда входит сложность реализации и сопровождения алгоритма:

– низкая: O (1) операций;
– средняя: часто выполняется O (log n) операций;
– высокая: O (n) или более.

4- Пригодность в режиме реального времени [7].

Системы реального времени требуют детерминированного поведения. Пригодность подразделяется на качественные уровни:

– высокая пригодность: детерминированное и предсказуемое время распределения;
– умеренно подходящий: в некоторой степени предсказуемый, но может варьироваться в зависимости от поиска;
– менее подходящий: высокая изменчивость и непредсказуемое поведение.

Заключение

Мы рассмотрели популярные алгоритмы динамического резервирования памяти, используемые в операционных системах, и увидели, что у каждого из них есть свои плюсы и минусы, и идеального алгоритма не существует, потому что каждый из них используется в определенных местах и для определенных нужд и соответствует оборудованию и его назначению.

Благодарности

Московский государственный технологический университет «Станкин».

Конфликт интересов

Не указан.

Рецензия

Все статьи проходят рецензирование. Но рецензент или автор статьи предпочли не публиковать рецензию к этой статье в открытом доступе. Рецензия может быть предоставлена компетентным органам по запросу.

Acknowledgement

Moscow State Technological University "Stankin".

Conflict of Interest

None declared.

Review

All articles are peer-reviewed. But the reviewer or the author of the article chose not to publish a review of this article in the public domain. The review can be provided to the competent authorities upon request.

Список литературы на английском языке / References in English

1. Tanenbaum A.S. Modern Operating Systems / A.S. Tanenbaum. — USA : Pearson, 2014. — 1101 p.
2. Silberschatz A. Operating System Concepts 10th / A. Silberschatz, G. Gagne, P.B. Galvin. — USA : John Wiley & Sons, 2018. — 1040 p.
3. Richard J. Garbage Collection: Algorithms for Automatic Dynamic Memory Management 1st Edition / J. Richard. — USA : Wiley, 1996. — 404 p.

4. Bonwick J. The slab allocator: An object-caching kernel memory allocator / J. Bonwick // USENIX Association. — 1994. — P. 87–98.

5. Knuth D.E. The Art of Computer Programming / D.E. Knuth. — USA : Addison-Wesley Professional, 1997. — Vol. 1. — 672 p.

6. Wilson P.R. Dynamic storage allocation: A survey and critical review / P.R. Wilson, M. Neely, D. Boles // Springer Nature. — 2005.

7. Buttazzo G.C. Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications (Real-Time Systems Series, 24, Band 24) / G.C. Buttazzo. — USA : Springer, 2011. — 540 p.