УДК 519.6

АЛГОРИТМ ДИНАМІЧНОГО РОЗПОДІЛУ ПАМ'ЯТІ

Довгалець С.М., к.т.н.,доц., Борщова І.П., студ.
Вінницький національний технічний університет
21021, м. Вінниця, вул. Хмельницьке шосе, 95
E-mail: irina_borshchova@ukr.net.

В данной статье описан новый алгоритм динамического распределения памяти, который позволяет достичь большего быстродействия при использовании меньшего объема памяти. В статье описано главные цели разработки, алгоритм и область применения предложенного метода распределения памяти.

Ключевые слова: память, динамическое распределение, алгоритм, стратегия лучшего подходящего, фрагментация, блок.

In the given paper new dynamical memory allocation algorithm which allows reaching better operation speed with lesser memory bill is described. This article presents a description of some of the main design goals, algorithm, and implementation fields for this allocator.

Key words: memory, dynamical allocation, algorithm, best-fit strategy, fragmentation, block.

Вступ. Динамічний розподіл пам'яті – це управління ресурсами обчислювального пристрою під час виконання програми. За допомогою динамічного розподілу пам'яті можна гнучко керувати часом життя об'єктів, що, на відміну від стекового методу розподілу, не призводить до неконтрольованого збільшення необхідної області пам'яті [1].

Динамічний розподіл пам'яті сьогодні є актуальною проблемою розподілу обчислювальних ресурсів. Особливо різко постає ця проблема при проектуванні мікропроцесорних пристроїв на основі мікроконтролерів, коли доводиться обробляти дані невеликого обсягу.

Проте мови програмування високого рівня під ці задачі, такі як С, не мають засобів для їх вирішення. Існуючі універсальні методи динамічного розподілу пам'яті, які могли б бути покладеними в основу вирішення даної проблеми, використовувати недоцільно через неефективність їх функціонування для даної вузької області застосування.

Таким чином, постає необхідність створення алгоритму, який би при невеликому обсязі даних забезпечив економне використання обчислювальних ресурсів, таких як час та пам'ять.

Динамічний розподіл об'єднує в собі три пов'язані процедури:

а) виділення пам'яті, яка по суті є блоком пам'яті необхідного розміру, утвореного з блоків, забезпечених операційною системою;

б) звільнення блоку пам'яті, що більше не потрібний, для подальшого використання;

в) склеювання між собою блоків пам'яті малого розміру для найбільш ефективного їх використання в майбутньому.

Існує ряд універсальних алгоритмів динамічного управління пам'яттю. Основні параметри, що їх характеризують, - це:

- час процесора, що використовується менеджером пам'яті під час роботи програми;
- обсяг пам'яті, що потрібен для реалізації стратегії динамічного управління пам'яттю;
- сумісність реалізацій; оскільки деякі платформи мають специфічні проблеми з обслуговуванням пам'яті.

Аналіз попередніх досліджень. Існує ряд стратегій динамічного управління пам'яттю. Більшість з них ґрунтуються на методології, що використовує модель поведінки "типової" програми, запропонованої в 1961 році . Ця методологія легко реалізується і дозволяє уникати специфічної поведінки деяких програм.

Стратегії динамічного управління пам'яттю можуть бути згруповані в три категорії:

1. Стратегії послідовної відповідності (наприклад, стратегія першого відповідного блоку, наступного відповідного і найкращого блоку) [2, 3].
2. Стратегії, що використовують вільні виділені списки: просте виділене зберігання, виділений придатний і т.п. [4, 5].
3. Методи близнят: двійкові і подвоєні ієрархічні системи і т.п. [6].

Проте, за умови невеликого об'єму даних, ці стратегії мають певні недоліки. Наприклад, стратегії послідовної відповідності спричиняють значну фрагментацію. Прикладом такої фрагментації є такі, коли виділяють більші блоки пам'яті на короткий час, і малі – на довгий. У цьому разі вивільнений великий блок буде, швидше за все, відразу використаний для виділення пам'яті відповідно до запиту на малий обсяг і після цього не вивільниться найближчим часом.

Недоліком стратегій, що використовують прості виділені списки, є те, що вони не дозволяють змінювати розмір блоків (розбивати на менші або поєднувати) [7]. Тобто якщо програма запитуватиме тільки блоки одного розміру, система із вичерпуванням їхнього запасу розширюватиме динамічну пам'ять.

Методи близнят також мають ряд недоліків. Хоча за продуктивністю вони і випереджають інші алгоритми динамічного програмування, але вони неефективні для малого об'єму даних через те, що пам'ять розбивають на блоки, розмір яких є

степенем числа 2, що при невеликому об'ємі даних призводить до значної внутрішньої фрагментації, наслідок якої – неекономне використання обчислювальних ресурсів [8].

Мета роботи. Створення алгоритму динамічного розподілу ресурсів, при якому за невеликого об'єму даних, досягається значна оптимізація затрат обчислювальних ресурсів і реалізація оптимального дотримання наступних вимог:

- мінімізація використання пам'яті;
- максимізація сумісності;
- мінімізація часових витрат.

Матеріал і результати дослідження. Розроблений алгоритм базується на сукупності двох структур даних:

- структура даних, яка зберігає інформацію про усі вільні блоки, що забезпечує швидкий пошук підходящого блоку (для алгоритму пошуку);
- структура даних, що зберігає додаткову інформацію про блоки пам'яті (розмір попереднього і поточного блоків, інформацію, чи вільний блок), - для алгоритму склейки.

Створюється зв'язний список усіх вільних блоків, в якому блоки відсортовані по зростанню розміру. На початку кожного з блоків знаходиться необхідна інформація про блок. Список об'єднує між собою дві структури, перша з них містить вказівники на наступний та попередній блоки (для просування по списку до наступного чи попереднього блоку).

Використовуючи синтаксис мови С, друга структура виглядає так:

```
typedef
struct _block {
    int last_size;
    int size;
    bool is_allocated;
} block;
```

Перша структура має наступний вигляд:

```
typedef
struct _free_block {
        struct _block block;
        struct _free_block *last, *next;
} free_block;
```

Оскільки операція пошуку елемента в списку, який забезпечує перша структура, має лінійну складність, що в нашому випадку неприпустимо, тому вводиться ще одна структура, що є на зразок hash-масиву. Це - масив покажчиків на вільні блоки, індексом якого є функція від розміру блоку.

Для ще більшого скорочення часу пошуку застосовується вирівнювання розміру до кратності 8 байтам (на сучасних процесорах така кратність є першою необхідністю для оптимізації); в кожній комірці масиву (крім останньої) зберігається покажчик на елемент списку блоків, розмір якого не менший заданого.

Для пошуку потрібного блоку використовується стратегія пошуку найкращого блоку. Стратегія вибору найкращого блоку зводиться до виділення пам'яті із вільного блоку, розмір якого найближчий до необхідного розміру пам'яті. Після такого виділення у пам'яті залишатимуться найменші вільні фрагменти (рис.1).

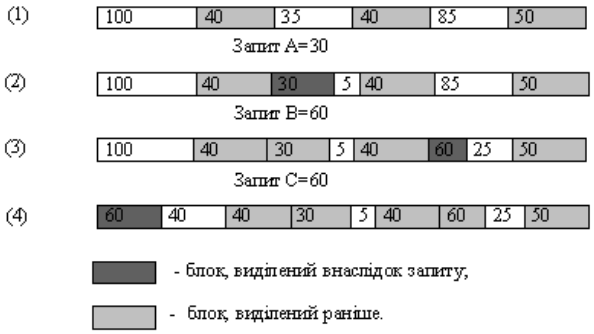

Рисунок 1 – Алгоритм найкращого блоку

Для виділення блоку в масиві знаходиться покажчик на той блок, розмір якого є найбільш близьким до потрібного. Потім, якщо розмір знайденого блоку збігається з необхідним розміром, блок видаляється зі списку і позначається як зайнятий. Якщо розмір знайденого блоку більший, то блок розбивається на 2 блоки, один із яких необхідного розміру, позначається як зайнятий і видаляється зі списку, другий же вставляється у відповідну позицію списку (алгоритм виділення блоку).

Для повернення блоку системі блок позначається як вільний, вставляється у потрібну позицію в списку, використовуючи для знаходження цієї позиції hash-масив, який потім відповідно коректується (алгоритм вивільнення блоку).

Алгоритм склеювання блоків є найбільш складним і витрато містким за часом, але принциповим з погляду мінімізації фрагментації. На основі додаткової інформації про звільнений блок (його розмір, розмір попереднього блоку) легко знаходяться сусідні блоки. Якщо один або обидва сусідніх блоки також вільні, то виконується склеювання. Після склеювання коректується список блоків і hash-масив, що є найбільш складною операцією, яка потребує багато часу для здійснення. Склеювання блоків може виконуватись як відразу після звільнення блоку, так і через якийсь проміжок часу, наприклад, коли нагромадилася певна кількість блоків.

Описаний алгоритм був реалізований для операційної системи Linux. Швидкодія алгоритму порівнювалася із швидкодіями інших розповсюджених методів. Для оцінки швидкодії була написана програма, яка випадковим чином виділяла і повертала через певний час блоки розміром від 5 до 2000 байт. Оцінювався як і загальний час виконання програми, так і окремо середній час виконання операцій виділення та повернення блоків.

Для визначення загального часу виконання використовувалася стандартна утиліта time. Для дослідження швидкодії окремих операцій використовувалась програма GNU profiler. Результати наведені в таблиці 1.

Таблиця 1 – Результати тестування різних методів реалізації динамічного розподілу пам'яті

№	Реалізація	Загальний час, с	Час виділення блоку, мкс	Час повернення блоку, мкс	Використано пам'яті, байт
1	Doug Lea's malloc	4,78	2,01	2,38	2752512
2	Hans Boehm's garbage collector	15,74	---	---	6967296
3	Запропонований підхід	4,58	1,63	2,09	2770856
4	Mike Haerthel's malloc	4,21	0,58	2,01	3596288

Аналізуючи дані таблиці 1, запропонований метод 3 має значно кращі показники, в порівнянні з методом 2: метод 2 використовує на 151,5% більше пам'яті та на 243,7% більше часу (див. рис. 1). Метод 4, хоч і показує на 8,1% менші витрати за часу, але використовує на 29,8% більше пам'яті, тому також значно програє, порівняно з методом 3, оскільки більша швидкодія досягається завдяки використанню надмірної кількості пам'яті. Досить непогані показники, порівняно з методом 2 та 4, має метод 1, але порівняно методом 3 він хоч і показує на 0,7% менші затрати пам'яті, але затрати часу збільшуються на 4,4%.

З наведених даних видно, що існує співвідношення між швидкодією і пам'яттю, яка використовується, і тому, як правило, більш швидкий алгоритм використовує більше пам'яті. Але на прикладі розробленого алгоритму було продемонстровано, що підвищення швидкодії для певного типу задач є можливим і при відносно сталому використанні пам'яті. Це добре видно на прикладі методів 1 та 3.

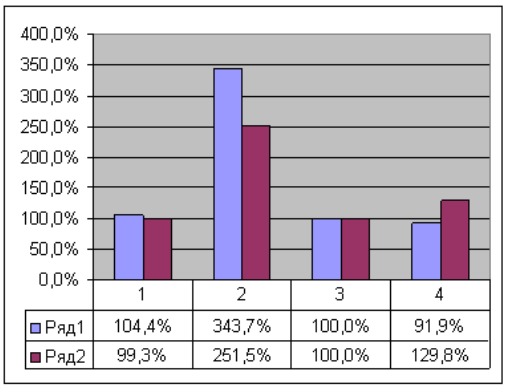

Рисунок 2 – Результат порівняння ефективності методів реалізації динамічного розподілу пам'яті за даними таблиці 1

Висновки. В статті проведено огляд існуючих методів розв'язку задачі динамічного виділення пам'яті і запропоновано власний алгоритм, який при невеликому об'ємі даних дозволяє досягнути оптимізації затрат обчислювальних ресурсів, таких як час та пам'ять і доводить, що навіть при досить високій швидкодії алгоритму можливо звести використання пам'яті до константного.

ЛІТЕРАТУРА

1. Christopher W. Fraser. A Retargetable C compiler: /Christopher W. Fraser. David R.Hanson. Design and Implementation. – Benjamin/Cummings Publishing, 1995.

2. A Memory Allocator. [Electronic resource] / Doug Lea // Hamser Verlag. – 1996. – Access mode: http://www.g.oswego.edu/dl/html/malloc.html.

3. Stephenson C.J. Fast fits: New method for dynamisc storage allocation // Operating Systems Review 17(5), 1982, pp. 30-32.

4.J.L.Petersonlangnp1033, T.A. langnp1033Norman. langnp1033Buddy systems. Communications of the ACM, 20(6): 421-431, 1877.

5. G.O. Collins. Experience in automatic storage allocation. Communications of the ACM, 4(10): 436-440, October 1961.

6. Mark S. Johnstone, Paul R. Wilson, "The Memory Fragmentation Problem: Solved.

7. Donald E. Knuth. The Art of Computer Programming, volume 1: Fundamental Algorithms, Addison-Wesley, Reading, Massachusetts, 1973.

8. Шеховцов В.А. Операційні системи. – К.: Видавнича група BHV, 2005. – 256 с.

Стаття надійшла 15.03.2009 р.
Рекомендовано до друку д.т.н., проф.
Чорним О.П.