В.П. Ивашенко (Минск, БГУИР)

АЛГОРИТМЫ ПОЛИЛОГАРИФМИЧЕСКОЙ ВРЕМЕННОЙ И ЛОГАРИФМИЧЕСКОЙ ПРОСТРАНСТВЕННОЙ СЛОЖНОСТИ ДЛЯ СИСТЕМЫ ДИНАМИЧЕСКОГО РАСПРЕДЕЛЕНИЯ ЛИНЕЙНО АДРЕСУЕМОЙ ПАМЯТИ С ОДНОРОДНЫМ ДОСТУПОМ К ДАННЫМ

В работе [1] была рассмотрена реализация системы динамического распределения памяти. В данной работе рассматривается улучшение ранее рассмотренной реализации. Удалось достигнуть понижения пространственной сложности алгоритмов до логарифмической за счёт исключения рекурсии и отказа от использования стековых структур для хранения промежуточных данных в алгоритмах системы динамического распределения памяти, а также — значительного сокращения степени внешней фрагментации в наихудшем случае за счёт реализации стратегии «первый подходящий» с использованием декартовых деревьев специального вида [1].

Рассмотрим память размером N бит. Память разбита на S ячеек по d бит каждая ($S \le 2^d$). Каждая ячейка имеет адрес равный двоичному вектору размером d. Среди ячеек F постоянно заняты и хранят служебную информацию. Число возможных различных ситуаций соотношения занятых и свободных ячеек в такой памяти 2^{S-F}. Пусть множество всех таких ситуаций будет $X \subseteq B^A$, где $A = B^d$ и $B = \{0,1\}$. Функция перехода системы [2] имеет вид $c : I \times X \to X$, где $I = A^2$. В работе [1] определены вероятность обнаружения первой ячейки свободного участка памяти и вероятность обнаружения пары из свободной и располагающейся после неё занятой ячейки среди четверти всех пар ячеек, включающей половину пар с занятой и свободной ячейкой. Соответствующие им энтропии обозначены e_1 и e_2. Максимальный размер единовременно выделяемого и высвобождаемого блока обозначен, как m.

В системе динамического распределения памяти решаются задачи двух классов: выделения и высвобождения блока памяти. В задаче выделения блока памяти в качестве исходных данных участвует размер выделяемого блока, а в качестве выходных — адрес выделенного блока и его размер, её целью является исключение выделенного блока из структуры свободных участков памяти. В задаче высвобождения блока памяти в качестве исходных данных участвует адрес первой ячейки и размер высвобождаемого блока, её целью является выявление свободных блоков, включающих прилежащие к высвобождаемому блоку ячейки памяти, исключение их из структуры свободных участков памяти, слияние этих блоков и включение результата слияния в структуру свободных участков памяти. Таким образом, любая из этих двух классов задач может быть однозначно задана предикатом текущей ситуации незанятости ячеек памяти s и парой из I, ко-

гда адрес одной из ячеек A_{null} кодирует незаданный адрес. Существующие подходы к реализации таких систем динамического распределения памяти [4] различаются не менее чем по трём аспектам: способом представления множества свободных участков памяти, стратегией выделения свободных блоков и наличием дополнительных внешних метаинформационных структур обеспечивающих описание и ускоренный доступ к ячейкам памяти. Известны реализации систем динамического распределения памяти TLSF [7], nedmalloc [11], ptmalloc3, mtmalloc, dlmalloc [8], jemalloc, libumem, Hoard [9] и другие. Однако они имеют ряд недостатков [1]. Известны несколько вариантов алгоритмов реализующих стратегию «первый подходящий», предложенных различными авторами [4; 6; 10; 13]: «Алгоритм A», «Алгоритм M», «Алгоритм S» и «Алгоритм N». Алгоритм A использует списки и имеет линейную временную сложность [6; 10]. Алгоритм M использует АВЛ-деревья свободных участков памяти без граничных меток и не поддерживает блоки из менее, чем пяти ячеек [6; 10]. Алгоритм S использует декартово дерево и имеет линейную временную сложность в наихудшем случае [6; 13]. Алгоритм N поддерживает логарифмическую временную сложность, однако использует внешние табличные структуры, увеличивающие его пространственную сложность до линейной [6]. Предлагаемое решение преодолевает эти недостатки.

$$O\left(\log\left(N\right)\right) \tag{1}$$

$$O\left(f\left(N\right)*\log^2\left(N\right)\right) \tag{2}$$

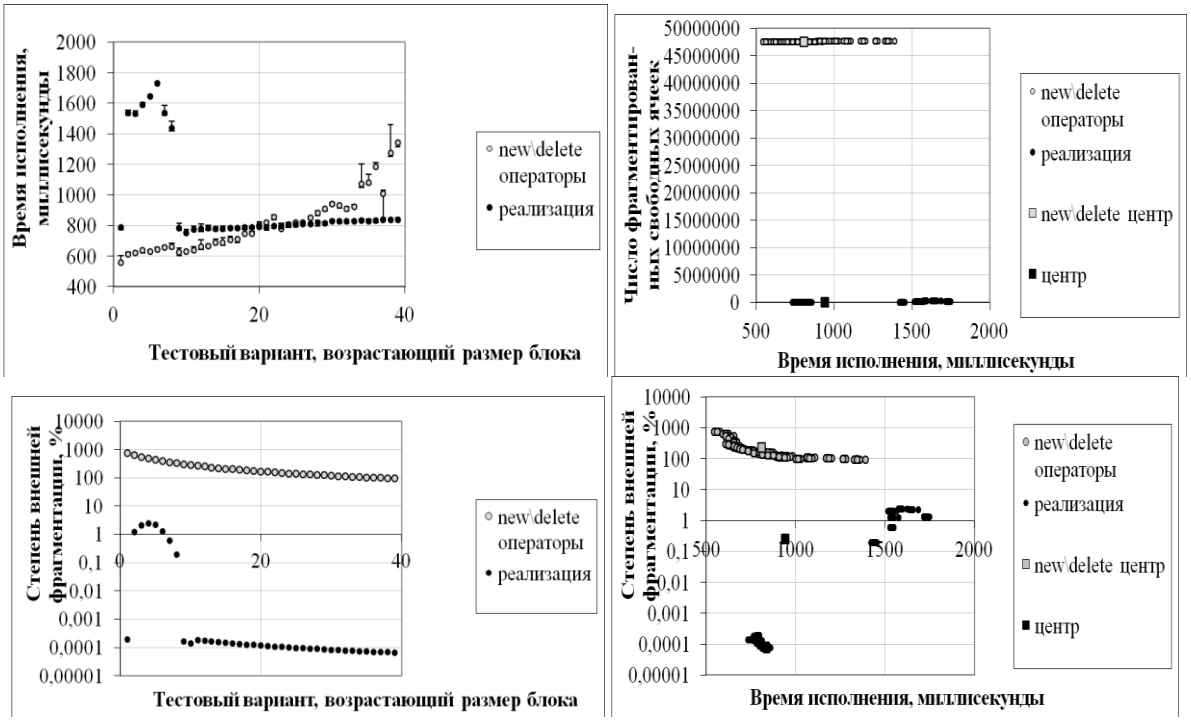
```
allocate(A_allocating) do
 A_left ← S
 A_right ← −S
 A_current_address ← A_root_address
 while (valueOf(A_current_address) ≠ A_null) do
  A_current ← valueOf(A_current_address)
  A_center ← (A_left + A_right)/2
  if (A_allocating < A_current) then do
   A_current_address ← addressOfRight(A_current)
   A_left ← A_center
  end else if (A_allocating > A_current) then do
   A_current_address ← addressOfLeft(A_current)
   A_right ← A_center
  end else do
   A_right ← nearestRight(right(A_current))
   A_left ← nearestLeft(left(A_current))
   if ((A_left = A_null) or ((A_right ≠ A_null) and
       (distance(A_center, A_right) < distance(A_center, A_left))))
   then allocateRight(A_current_address)
   else allocateLeft(A_current_address)
   break
  end
 end
 return valueOf(A_current_address)
end
```

```
free(A_freeing) do
 A_left ← S
 A_right ← −S
 A_current_address ← A_root_address
 while (valueOf(A_current_address) ≠ A_null) do
  A_current ← valueOf(A_current_address)
  A_center ← (A_left + A_right)/2
  if (distance(A_center, A_freeing) <
      distance(A_center, A_current)) then do
   replace(A_current_address, A_freeing)
   A_freeing ← A_current
   A_current ← valueOf(A_current_address)
  end
  if (A_current > A_freeing) then do
   A_current_address ← addressOfRight(A_current)
   A_left ← A_center
  end else if (A_current < A_freeing) then do
   A_current_address ← addressOfLeft(A_current)
   A_right ← A_center
  end else break
 end
 replace(A_current_address, A_freeing)
 return
end
```

Рис. 1. Алгоритмы выделения и высвобождения блока из двух ячеек

Предлагаемая реализация имеет следующие характеристики: отсутствие внешней фрагментации для блоков памяти, содержащих целое число ячеек; логарифмическое ограничение пространства (1), полилогарифмическая временная сложность (2) в случае константного времени доступа к адресуемой ячейке $f(N)$ [1] и логарифмическая временная сложность в случае, когда размер выделяемых и высвобождаемых блоков памяти кратен натуральному числу большему единицы; работоспособность в многопоточной среде; использование только восьми ячеек для хранения внешней структуры с дополнительной информацией в виде массива адресов корней деревьев, хранящих свободные участки памяти; поддержка стратегий «наилучший подходящий» и «первый подходящий»; возможность поддержки других стратегий.

Для хранения свободных участков памяти используется восемь структур: четыре структуры для одноячеечных блоков, деревья для блоков из двух, трёх, четырёх и свыше четырёх ячеек. Деревья для хранения блоков из двух, трёх и четырёх ячеек организованы по дихотомическому принципу [1]. Для хранения одноячеечных блоков используется аналогичная структура, а для хранения блоков из пяти и более ячеек используется сбалансированное декартово дерево [10]. На рис. 1 приведены алгоритмы добавления и удаления блока из дерева блоков из двух ячеек. На рис. 2 приведены результаты тестирования реализованного алгоритма. Реализация проведена на языке C++ с использованием шаблонов, исходные тексты проекта занимают порядка 150КиБ.



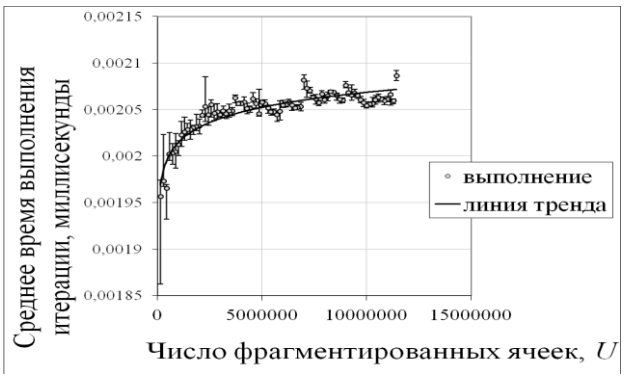


Рис. 2. Сравнительные результаты тестирования реализованного алгоритма и реализации C++ операторов new и delete

Рис. 3. Увеличение требуемого для решения задачи времени и пространства в условиях динамического распределения памяти

Характеристики предлагаемой реализации важны для систем реального времени и масштабируемых систем. Вообразим прикладную интеллектуальную систему, планирующую решение задачи, запись условия которой занимает *D*d* бит. Пусть оценки сложности на уровне абстрактной модели (исключая время на распределение памяти) требуют для её решения время и память соответственно $O(t(D))$ и $O(s(D))$ (чёрный квадрат на Рис. 3). Тогда, используя предлагаемые алгоритмы, полное решение задачи, включая время на распределение памяти, будет иметь сложность по пространству и времени не хуже, чем $O\left(t(D)*\log_2^2(N)\right)$ и $O\left(s(D)*\left(1.06+\log_2(s(D))\right)\right)$ (серый прямоугольник на Рис. 3). Также прикладная система, способна оценить допустимое значение *D*:

$$D \le s^{-1}\left(\frac{W\left(2^{2.085*\frac{N}{d}}\right)}{2.085}\right) \qquad (6)$$

В соответствии с максимально возможной степенью внешней фрагментации для реализованной стратегии [12] безопасное значение размера

полезной занятой памяти U может быть выражено из (7), где C — размер полезного постоянно занятого пространства.

$$(S-U)/U \geq \left((U-C)*(1.06+\log_2(m))+C-U\right)/U \quad (7)$$

В этом случае энтропия при размере адресного пространства 2^{32} может достигать значений не менее $e_1 \approx 0.22$, $e_2 \approx 0.359$ и не более $e_1 \approx 0.352$, $e_2 \approx 0.54$. Следует отметить, что в наихудшем случае степень внешней фрагментации при стратегии «первый подходящий» близка к наилучшему значению в наихудшем случае для произвольно взятой стратегии [12].

Ограничения на количество участков свободной памяти доступное за время, не превышающее заданную границу T, при предлагаемой реализации стратегии «первый подходящий» и при реализации, использующей списковое представление, соответствуют (8) и $O(T)$.

$$O\left(2^{\sqrt{\frac{T}{k}}}\right) \quad (8)$$

Отсюда можно вывести, что при $k < T * \log_T^2 2$ и соответствующем значительном объёме памяти, недоступной за время T при последовательном доступе по списку, следует отдавать предпочтение предлагаемой реализации, иначе — использовать списки. Таким образом, предлагаемое решение имеет сравнимые по абсолютным значениям основные характеристики, обеспечивая не худшую или лучшую по классу вычислительную сложность и увеличение гарантируемого объёма доступной памяти, не смотря на возрастающие, вследствие этого, значениях энтропии.

ЛИТЕРАТУРА

1. Ивашенко, В.П. Вордомацкий, Ю.А. Алгоритмы полилогарифмической временной и пространственной сложности для системы динамического распределения линейно адресуемой памяти с однородным доступом к данным. // материалы 5-й конф. «Карповские чтения». — Минск, 2011.

2. Берталанфи, Л. фон. Общая теория систем — Критический обзор // Исследования по общей теории систем. — М., 1969.— С. 23–82.

3. Шеннон, К. Работы по теории информации и кибернетике. — М., 1963.

4. Wilson, P.R., Johnstone, M.S., Neely, M., Boles, D. Dynamic Storage Allocation: A Survey and Critical Review. University of Texas at Austin. 1995.

5. Гомон, С.В., Лапицкий, А.В. Алгоритмы логарифмической временной сложности для поиска в декартовом дереве специального вида. // материалы 47-й науч. конф. аспирантов, магистрантов и студентов (25–29 апреля 2011 года). — Минск, 2011. — С. 28.

6. Brent, R.P. Efficient implementation of the first-fit strategy for dynamic storage allocation. ACM Transactions on Programming Languages and Systems. 11 (3) (1989). — PP. 388–403.

7. TLSF: A New Dynamic Memory Allocator for Real-Time Systems (M. Masmano, I. Ripoll, A. Crespo and J. Real). 16th Euromicro Conference on Real-Time Systems (ECRTS 2004).

8. Lea, D. A Memory Allocator. — Unix/Mail, 6/96, 1996.

9. Emery, D.B., Blumofe, R.D. 1999. *H*oard: A Fast, Scalable, and Memory-Efficient Allocator for Shared-Memory Multiprocessors. University of Texas at Austin. UTCS TR99-22.

10. Кнут, Д. Искусство программирования. — Вильямс, 2000.

11. Nedmalloc Homepage [Электронный ресурс]. — Режим доступа: http://www.ned prod.com/programs/portable/nedmalloc/index.html. — Дата доступа: 1.09.2010.

12. Robson, J. M. Worst case fragmentation of first fit and best fit storage allocation strategies. Comput. J. 20,3 (Aug. 1977).

13. Stephenson, C. J. New methods for dynamic storage allocation (Fast Fits), Proceedings of the ninth ACM symposium on Operating systems principles. — Bretton Woods, New Hampshire, United States. — 10–13 October 1983. — PP. 30–32.