УДК 004.424.74

А. В. Аксенов*
старший преподаватель
В. В. Михайличенко*
студент
*Санкт-Петербургский государственный университет аэрокосмического приборостроения

ИССЛЕДОВАНИЕ И РЕАЛИЗАЦИЯ ПОДХОДА К ОРГАНИЗАЦИИ ЭФФЕКТИВНОГО УПРАВЛЕНИЯ ПАМЯТЬЮ В РЯДЕ ВЫЧИСЛИТЕЛЬНЫХ ЗАДАЧ

Предложены идея и реализация специализированного аллокатора памяти, который может быть использован в ряде задач, критичных к эффективности и удовлетворяющих ряду существенных ограничений, среди которых одинаковый размер объектов и отсутствие необходимости их освобождения по запросу.
Ключевые слова: динамическое распределение памяти, управление памятью.

A. V. Aksenov*
Senior Lecturer
V. V. Mikhailichenko*
Student
*St. Petersburg State University of Aerospace Instrumentation

STUDY AND IMPLEMENTATION OF APPROACH TO ORGANIZING AN EFFECTIVE MEMORY MANAGEMENT IN A NUMBER OF COMPUTATIONAL PROBLEMS

Proposed are idea, design and implementation of specialized memory allocator that can be utilized in some applications critical to time efficiency and having a number of crucial limitations amongst which are equal-sized objects and no need to deallocate them on demand.
Keywords: dynamic memory allocation, memory management.

В контексте какой-либо вычислительной задачи нередки случаи, когда размеры и время «жизни» объектов в программе неизвестны на момент разработки. Тогда необходимо прибегнуть к использованию динамического выделения памяти.

При интенсивном выделении и освобождении небольших участков памяти сильно падает производительность приложения. Это связано с недостатками стандартных средств динамического распределения памяти. Каждый раз, когда происходит динамическое выделение памяти, резервируется место в «куче» («heap»). Каждое такое выделение требует обращения к ядру операционной системы, что влечет высокие накладные расходы (это иллюстрирует схема на рис. 1). К тому же память подвергается фрагментации, что также отрицательно сказывается на производительности.

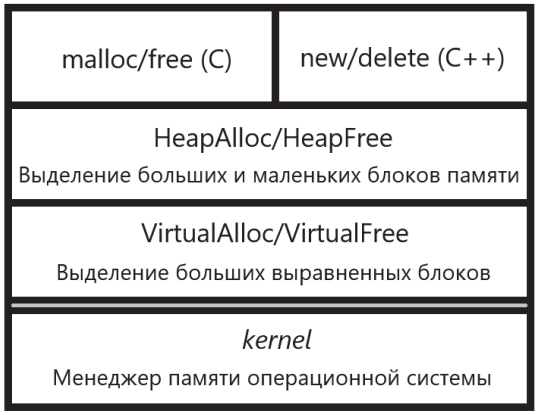

Рис. 1. Упрощенная схема вызовов функций при выделении памяти в MS Windows

В языках C и C++ динамическое выделение памяти выполняется с помощью функции malloc (C) и оператора new (C++). К сожалению, malloc и new присущи вышеперечисленные недостатки. Также стоит отметить, что необходимо самостоятельно отслеживать каждое выделение памяти и своевременно возвращать ее обратно системе, иначе существует риск утечки памяти, что может нарушить работу не только программы, но и операционной системы (ОС).

В свете этого актуальной становится задача разработки средства – менеджера памяти, осуществляющего эффективное ручное распределение и освобождение памяти, а также сводит риск утечек памяти к минимуму. Стоит понимать, что, в отличие от стандартных средств распределения памяти, у разрабатываемого менеджера меньшая область применения – использование имеет смысл только тогда, когда разработчику заранее известно, что время «смерти» всех объектов совпадает. Таким образом, разработчик, принимая во внимание специфику поставленной задачи, может воспользоваться данным средством для оптимизации работы программы.

Основной стратегией разрабатываемого менеджера памяти является запрос больших фрагментов памяти, организации пулов памяти и дальнейшее распределение выделенной памяти внутри приложения. Следовательно, обращение к стандартным функциям ОС будет происходить только один раз при запуске программы, а не каждый раз при добавлении нового объекта.

В соответствии со стандартом C++ размер указателя зависит от конкретной реализации компилятора и не связан напрямую с разрядностью используемой платформы. Однако на большинстве современных ОС общего назначения (настольные UNIX-совместимые системы, MS Windows) используются модели данных, в которых размер указателя соответствует разрядности адресной шины у архитектуры этих платформ. Ширина шины адреса определяет объем адресуемой памяти. Например, для Visual Studio 2017 под управлением Windows 10 64-bit размер указателя равен 8 байтов или 64 бита. В разрабатываемом менеджере за счет того, что адресация происходит внутри пула, работа с памятью может быть инкапсулирована, а размер указателя уменьшен. Вместо 2^{64} байт памяти мы адресуем только память пула.

Для выбора оптимального размера указателя необходимо принять во внимание выравнивание данных. Процессоры в качестве основной единицы при работе с памятью используют машинное слово, размер которого всегда равен нескольким (как правило 2^n) байтам. Это означает, что структура размером 3 байта займет в памяти все 4 – последний байт будет игнорироваться процессором и не будет использован. Отсюда следует, что размер указателя должен быть целочисленной степенью числа 2. Адресация 4294967296 объектов (для указателя размером 4 байта – 2^{32}) – явное излишество, а значит оптимальный размер указателя составляет 2 байта или 16 бит [1]. Тогда в пуле могут храниться 65536 (2^{16}) объектов. Для объекта размером 32 байта размер пула составит 2 Мбайта (32 x 65536 = 2097152 байта).

Разработка менеджера памяти

Прежде всего необходимо получить у ОС большой блок памяти для создания пула объектов. В ОС MS Windows есть функция VirtualAlloc, которая выделяет большие выровненные блоки памяти (рис. 1). Выравнивание в данном контексте подразумевает, что выделенный блок будет иметь размер, кратный 4 Кбайтам, а адрес начала блока памяти, который вернет функция, 64 Кбайтам. Это означает, что при попытке выделить 1 байт памяти ОС все равно выделит 64 Кбайта. Из этого следует, что выделение страниц, размер которых меньше 64 Кбайт, – пустая трата памяти. Так как VirtualAlloc является самой низкоуровневой функцией, доступной разработчику, ее использование позволит обойти лишний уровень абстракции – использование «кучи», что положительно скажется на производительности, а также сократит накладные расходы [2].

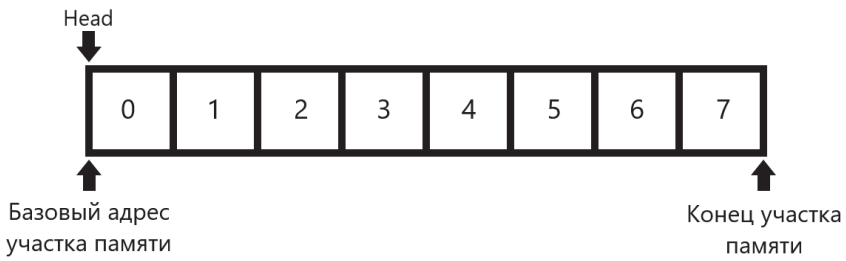

Рис. 2. Начальное состояние пула

С помощью функции VirtualAlloc необходимо получить базовый адрес – начало выделенного участка памяти. Этот участок делится на равные блоки, доступ к которым осуществляется по индексу, – базовый адрес + индекс (арифметика указателей C++). Head – индекс текущего свободного блока, изначально совпадает с базовым адресом (рис. 2). При попытке выделения аллокатор возвращает значение Head – индекс, по которому можно получить доступ к объекту, после чего Head увеличивается на единицу (сдвигается на следующий объект) (рис. 3).

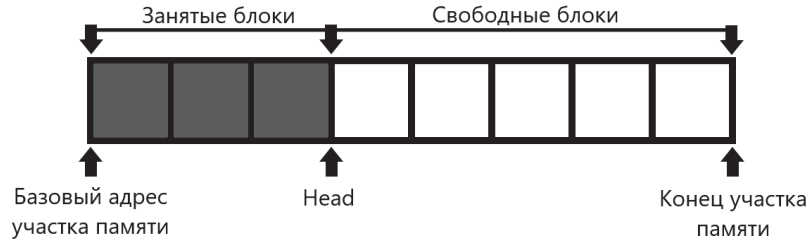

Рис. 3. Состояние пула после выделения памяти для трех объектов

Важной отличительной чертой данного аллокатора является невозможность освобождения блоков по отдельности (возврат их в пул). С другой стороны, нет необходимости в отслеживании свободных или занятых ячеек, что позволяет значительно увеличить скорость выделения. К тому же есть возможность освободить весь пул целиком, просто вернув память назад операционной системе.

Сравнительный анализ

Для проведения анализа были разработаны две программы. В первой (далее – std_alloc) использовались стандартные средства динамического распределения памяти языка C++ операторы new и delete. Во второй (далее – custom_alloc) использовался разработанный менеджер памяти. На вход программам подается целое число – количество объектов, память для которых выделится и освободится в процессе выполнения. Алгоритм следующий: сначала поштучно выделяется память для 65536 объектов, затем освобождается. Цикл повторяется до тех пор, пока число выделений/освобождений не сравняется со входным значением.

Программы были скомпилированы тремя различными компиляторами: GCC, Clang, MSVC. Профилирование проводилось программой AMD μProf, с помощью которой было замерено время выполнения разработанных программ.

Результаты тестов приведены ниже.

По приведенной диаграмме (рис. 4) видно, что программа std_alloc, скомпилированная с помощью Clang, показывает в среднем более высокие результаты в своем сегменте. Однако программа custom_alloc показывает лучшие результаты в своем сегменте (рис. 5) в случае, когда скомпилирована с помощью MSVC. Для наглядного сравнения будем использовать результаты программ, скомпилированных с помощью Clang.

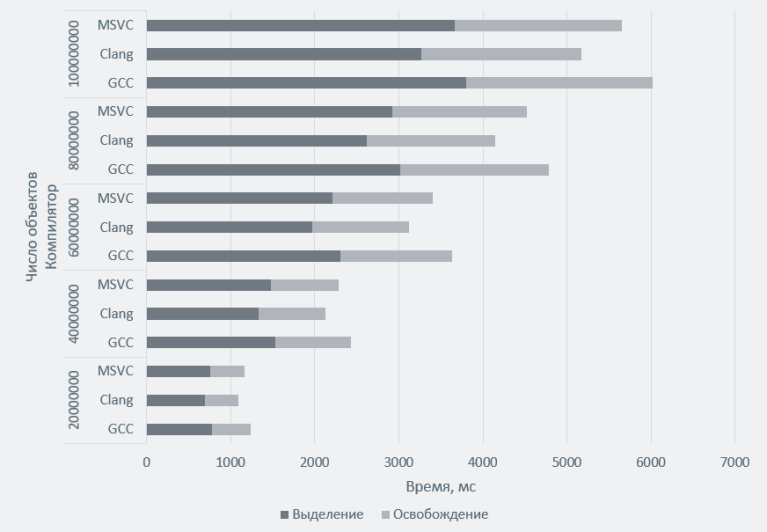

Рис. 4. Результаты работы программы std_alloc

Как видно из диаграммы выше (рис. 6), время на выделение и освобождение памяти можно многократно сократить за счет использования подходящего под задачу аллокатора.

Преимущества такого подхода очевидны, но ограничения весьма существенны. Использование разработанного аллокатора осмысленно только в весьма ограниченном круге задач, когда:

– время «смерти» всех объектов в пуле совпадает;
– все объекты имеют одинаковый размер;
– нет ограничения на объем памяти, которым располагает целевая система.

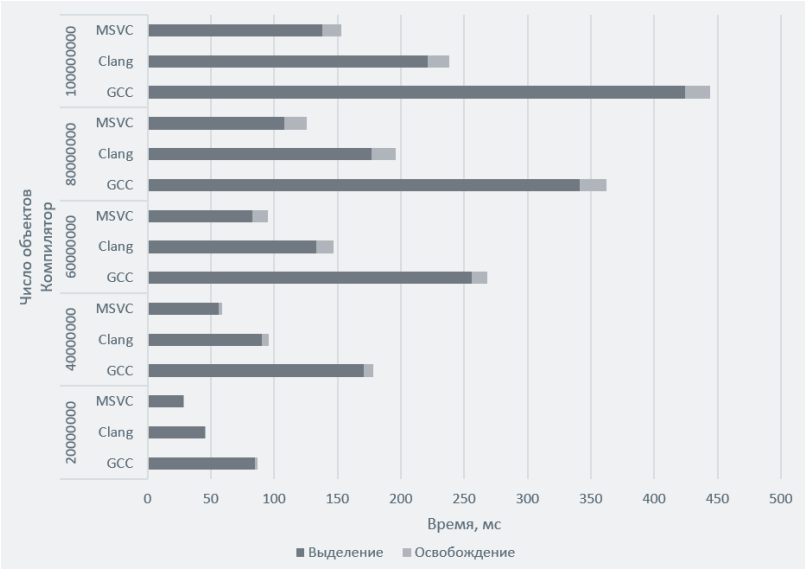

Рис. 5. Результаты работы программы custom_alloc

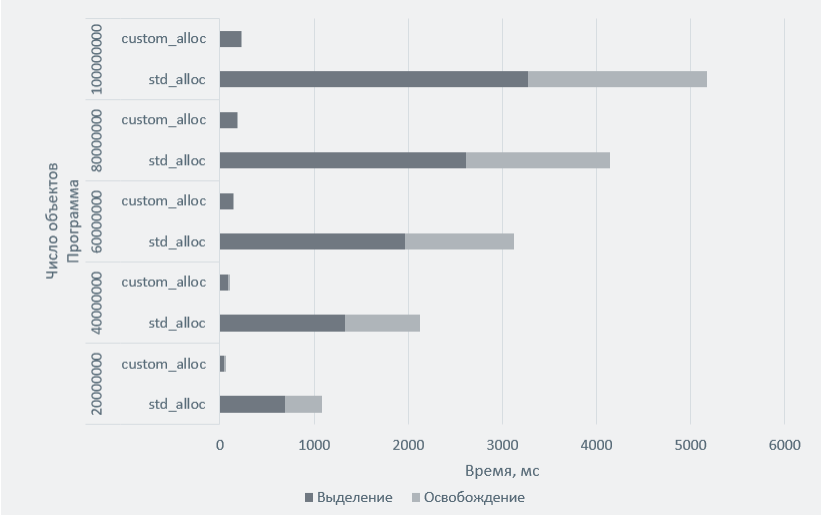

Рис. 6. Сравнение работы программ, скомпилированных с помощью Clang

Поясняя последний пункт, отметим, что аллокатор выделяет память сразу для 65536 объектов. Если необходимое число объектов значительно меньше, то большая часть памяти будет выделена впустую. В ряде задач такое пренебрежение памятью недопустимо. Тем не менее в современных системах, где объем памяти достаточно высок и есть возможность закрыть глаза на повышенное потребление памяти приложением, можно воспользоваться разработанным средством для повышения производительности программы.

Библиографический список

1. Rentzsch J. Data alignment: Straighten up and fly right. URL: https://developer.ibm.com/articles/pa-dalign (дата обращения: 10.04.2019).
2. Programming reference for Windows API. URL: https://docs.microsoft.com/en-us/windows/desktop/api/index (дата обращения: 10.04.2019).