НОВЫЙ ПОДХОД К БЫСТРОМУ ВЫДЕЛЕНИЮ ПАМЯТИ В ПРОГРАММАХ НА ЯЗЫКЕ C++

Веретенников А.Б.

Управление выделением динамической памяти является одной из ключевых задач программирования. При реализации многих структур данных, как-то: очереди, списки, различные древовидные структуры, активно используется динамическая память. Вместе с тем хорошо известно, что функции языка C malloc/free (и соответственно операции C++ new/delete) в силу универсального характера их реализации очень медленно работают при выделении/освобождении большого количества малых по размеру блоков памяти. Эта проблема присутствует и во многих других языках программирования и средах. Необходимо отметить, что упомянутые структуры требуют многократного выделения небольших блоков памяти. Это наводит на мысль о необходимости разработки функций управления памятью, которые эффективно работают с блоками небольшого размера.

А. Александреску в [1, гл. 4] дал вариант решения подобной задачи, но описанный им алгоритм иногда может работать неэффективно. В данной работе описан другой алгоритм, который решает задачу более эффективно и обладает гарантированной производительностью.

Предлагаемый алгоритм при выделении блока, размер которого меньше некоторого фиксированного числа (например, 1 Кб), всегда выделяет и освобождает память за небольшое константное число операций, то есть имеет для таких блоков вычислительную сложность O(1). Для блоков большего размера он просто передает управление обычным функциям.

Предполагается, что операционная система предоставляет нам функции управления памятью, которые работают для любых размеров блоков, но возможно медленно. С их помощью мы выделяем блок большого размера. Далее наша функция для выделения блока малого размера использует часть выделенного большого блока.

Реализация алгоритма автора статьи была выполнена на языке C++, при этом были учтены следующие аспекты:

1) Какие функции операционной системы использовать для выделения больших блоков: есть стандартные функции языка C malloc и free, но операционная система может предлагать свои функции.

2) Выделение памяти может происходить как в однопоточных программах, так и в многопоточных.

Основная идея быстрого выделения памяти заключается, как и в [1], в реализации выделения блоков памяти фиксированного размера, но предлагаемый алгоритм абсолютно иной.

Пусть у нас есть функции операционной системы для выделения блоков некоторого размера BlockSize, которые мы называем страницами, организуем хранение маленьких блоков в этих страницах. Страница описывается структурой LIST. Все страницы хранятся в виде циклического списка. Свободные блоки хранятся в виде общего для всех страниц циклического списка, при этом для каждой страницы определены указатели FreeFirst и FreeLast, которые указывают на ту часть общего циклического списка маленьких блоков, которая относиться к конкретной странице, свободные блоки для конкретной страницы располагаются в списке последовательно. Также для страницы определен счетчик Count – число выделенных блоков. Когда мы говорим, что работаем со списком страницы, это означает что мы работаем с частью общего списка используя FreeFirst и FreeLast.

Пусть мы организуем выделение памяти для блоков имеющих некоторый размер S. Каждый блок будет описываться структурой

```
struct ITEM {
   union { char Data[S];
      struct {ITEM *Next; ITEM *Prev; }; };
   LIST *Block; };
```

Когда блок заполнен, данные его хранятся в поле Data, когда он свободен поля Next и Prev используются для организации циклического списка. Указатель Next указывает на следующий элемент списка, а Prev — на предыдущий элемент списка. Поле Block указывает на заголовок страницы, который располагается в ее начале, остальная часть страницы заполнена блоками (далее будет отмечено, что поле Block не обязательно).

Общее состояние системы описывается переменными: BlockSize — размер страницы; FirstFreeItemSize — это число блоков, свободных и еще не выделявшихся в последней выделенной странице; list — указатель на последний из созданных элементов списка страниц; freeitem — указатель на один из элементов списка свободных блоков.

Использование параметра FirstFreeItemSize позволяет не добавлять все свободные блоки страницы в список свободных блоков при

выделении новой страницы. Всего блоков у страницы (BlockSize - размер(LIST)) / размер(ITEM). В список свободных блоков помещается только один блок – с номером 0, этот блок мы переносим в переменную freeitem. При выделении памяти блоки будут выделяться из страницы list пока FirstFreeItemSize>1. При этом нулевой блок, помещенный в переменную freeitem, будет выделен последним.

Выделение блока:

1. Переменная X обозначает выделяемый блок

2. Если список свободных блоков не пуст (указатель freeitem не нулевой), то переходим на шаг 2.1, иначе на шаг 3.

2.1. Получаем страницу P блока freeitem

2.2. Если FirstFreeItemSize>1 то выполняем шаг 2.2.1 иначе шаг 2.2.2.

2.2.1. Уменьшаем FirstFreeItemSize на 1 и устанавливаем в переменную X блок страницы list с номером FirstFreeItemSize. Заполняем переменную Block у X. Переходим на шаг 6.

2.2.2. Устанавливаем X равным freeitem. Удаляем freeitem из списка свободных блоков страницы P (меняем ее указатели FreeFirst, FreeLast). Удаляем freeitem из общего списка свободных блоков; устанавливаем переменную freeitem на следующий элемент в списке или обнуляем freeitem, если список состоял из одного элемента. Переходим на шаг 6.

3. Список свободных блоков пуст. Выделяем новую страницу, помещаем ее в список страниц, устанавливаем freeitem на нулевой блок страницы.

4. Устанавливаем значение переменной FirstFreeItemSize равным числу блоков в странице, значение переменной Count равным 0. Помещаем нулевой блок страницы в список свободных блоков (также меняем указатели FreeFirst и FreeLast у страницы).

5. Уменьшаем FirstFreeItemSize на 1 и устанавливаем в переменную X блок новой страницы с номером FirstFreeItemSize. Заполняем переменную Block у блока.

6. Увеличиваем счетчик Count у страницы блока X и возвращаем указатель на поле Data блока X.

Освобождение блока:

1. По адресу блока определяем указатель на структуру ITEM блока, получаем страницу блока P.

2. Уменьшаем счетчик Count у P.

3. Если Count равен 0, то нужно освободить страницу, удалив также все свободные блоки этой страницы из списка свободных блоков. Переходим на шаг 3.1, иначе на шаг 4.

3.1. Удаляем страницу P из списка страниц.

3.2. Если freeitem равен значению поля FreeFirst страницы P, то устанавливаем значение FirstFreeItemSize равным 1 и freeitem на свободный блок другой страницы (если он есть, то он найдется как следующий за блоком FreeLast страницы P блок в списке свободных блоков)

3.3. Для удаления всех свободных блоков страницы P из списка свободных блоков достаточно изменить указатели у следующего блока за FreeLast и предыдущего блока у FreeFirst.

3.4. Выход из процедуры

4. Страница не удаляется из памяти, нужно передать освобождаемый блок в список свободных блоков и обновить указатели FreeFirst и FreeLast у страницы P.

Подобная организация позволяет выделить и освободить блоки за константное O(1) число операций (без учета выделения страниц, используя функции операционной системы). Все операции ограничены простой арифметикой. Этот алгоритм был реализован в виде класса CConstantMemoryManager.

Для реализации выделения блоков размера меньшего или равного выбранного значения SMALL_SIZE необходимо создать SMALL_SIZE объектов, подобных CConstantMemoryManager. При этом указатель на конкретный экземпляр соответствующего блоку ITEM класса CConstantMemoryManager нужно поместить в структуру LIST.

Следует немного модифицировать структуру ITEM:

```
struct ITEM {
   LIST *Block;
   union { char Data[1];
      struct { ITEM *Next; ITEM *Prev; }; }; };
```

При выделении блока мы возвращаем поле Data, а при освобождении блока смотрим на память перед ним и получаем указатель Block на структуру LIST. Если выделяется блок размером большим, чем SMALL_SIZE, то для различия блоков следует выделить блок большего размера, чем указано, на размер указателя и поместить в

начало 0 (то есть обнулить указатель Block). Впоследствии можно таким образом различать большие и маленькие блоки.

Какой выбрать размер страницы? Для организации выделения памяти по небольшим страницам, например по 4Кб, можно использовать уже описанную схему CConstantMemoryManager, а уже здесь использовать большие страницы, например 1 - 10 Мб. Также полезно при освобождении страницы удерживать одну страницу в оперативной памяти и использовать ее при необходимости повторно. Модель выделения страниц оформлена как отдельный класс.

Можно избавиться и от указателя Block в структуре ITEM за счет выравнивания страниц на определенную границу, но в таком случае при освобождении блока нужно будет указывать его размер (только для того, чтобы различать, больше он чем SMALL_SIZE или нет).

В итоге созданы классы CExtendedConstantMemoryManager, имеющий методы Alloc(size_t Size) и Free(void *A), и CExtendedConstantMemoryManagerAlignment, имеющий метод Alloc(size_t Size) и Free(void *A, size_t Size). Для использования их с контейнерами STL созданы вспомогательные классы.

Реализованный алгоритм был сравнен с алгоритмами библиотеки Loki [2], разработанной А. Александреску, и работал более чем на 30% быстрее, и в 5-7 раз быстрее стандартных функций. При этом выяснилось, что при работе с большим числом блоков (более 10 миллионов) алгоритм Loki начинает работать весьма медленно, а описанный нами алгоритм сохраняет свою эффективность. При тестировании использовался компилятор Microsoft Visual C++ 7.1.

Список литературы

[1]. *Александреску А.* Современное проектирование на C++. М.: Вильямс, 2002.

[2]. http://www.awl.com/cseng/titles/0-201-70431-5.