МОДЕЛЬ АЛЛОКАЦИИ ПАМЯТИ В УСЛОВИЯХ ОГРАНИЧЕННЫХ РЕСУРСОВ

Иванов Е.А.[1], Мамонова Т.Е.[2]
[1] Томский политехнический университет, группа А3-36, e-mail: eai13@tpu.ru
[2] Томский политехнический университет, к.т.н., доцент ОИС, e-mail: stepte@tpu.ru

Введение

Устройства в основе которых лежат микроконтроллеры сегодня используются повсеместно. Данный факт обусловлен малыми размерами, низким энергопотреблением и способностью выполнять целый спектр задач. Однако вместе со всеми положительными аспектами у микроконтроллеров имеется ряд вычислительных ограничений. Одним из таковых является малый объем оперативной памяти. Использование стандартных средств языка для динамической аллокации в таком случае может привести к высокой степени фрагментации, утечкам памяти и практической сложности отслеживания свободного ее количества.

Целью данной работы является представление модели аллокации памяти, позволяющей частично или полностью нивелировать вышеописанные недостатки.

Описание общей концепции

В общем случае существует два основных подхода к выделению памяти - статический и динамический [1]. Статическое выделение подразумевает выделение памяти на этапе загрузки программы. Размер статически выделяемой памяти определен заранее и не может быть изменен. Динамически выделяемая память, наоборот, позволяет аллоцировать участки оперативной памяти по ходу выполнения программы. Объем используемой динамической памяти зачастую невозможно предсказать, так как ее выделение может быть не ограничено самой программой.

Использование динамической памяти может привести к бесконтрольному исчерпанию оперативной памяти. В связи с этим, для возможности динамического выделения памяти прибегают к выделению большого участка статической памяти, на базе которого позже выполняется динамическая аллокация. Данный подход иногда называют “динамической аллокацией на статической области” (dynamic allocation on a static pool). Существует ряд способов управления статической областью для получения желаемого результата. Наиболее частым из них является разбиение крупного участка статической памяти на сегменты фиксированного размера (чанки) [2, 3, 4, 5]. Вместе с этим составляется массив, состоящий из флагов. Данные флаги являются индикаторами занятости чанков статической области. Данная модель представлена на рис. 1.

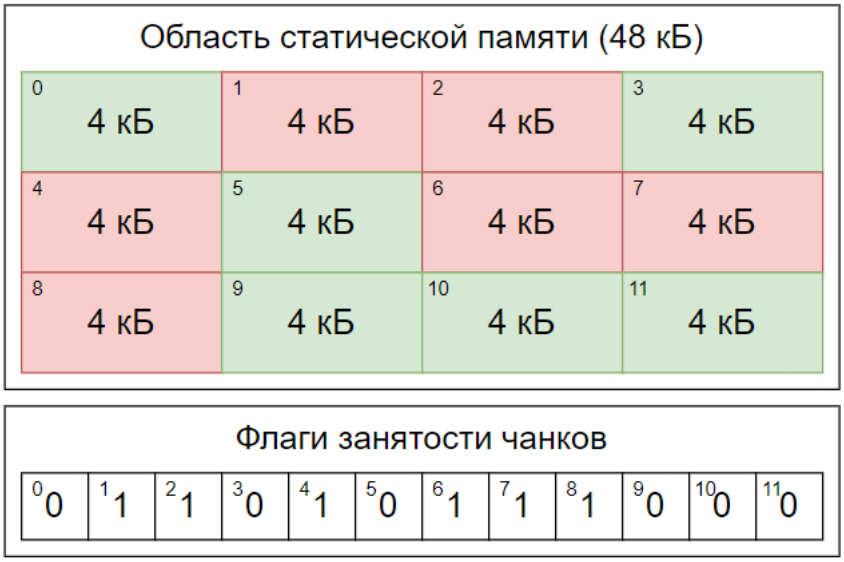

Рис. 1. Базовая модель аллокации памяти
Красным - занятые чанки
Зеленым - свободные чанки

Данный подход позволяет отслеживать объем свободного количества оперативной памяти, однако имеет ряд недостатков. Самым существенным из них является избыточность выделяемой памяти. Независимо от требуемого количества памяти, выделяется всегда участок фиксированного размера. Для наиболее эффективного использования необходимо индивидуально для каждой программы рассчитывать оптимальный размер чанков. Данная модель также не предусматривает возможности дефрагментации. Далее предлагается модель, основанная на вышеописанной, однако предусматривающая компенсацию описанных недостатков.

Описание предлагаемой реализации

В предлагаемой реализации также используется статическая область памяти. Однако, данная область не будет разбита на отдельные участки. Вместо этого в данной области будет формироваться двусвязный список. Каждый элемент списка имеет структуру, представленную на рис. 2.

Название поля	Адрес предыдущего участка	Адрес следующего участка	Размер текущего участка	Выделяемый участок данных
Возможный размер	UINT8 или UINT16 или UINT32 или UINT64	UINT8 или UINT16 или UINT32 или UINT64	UINT8 или UINT16 или UINT32 или UINT64	UINT8 [*]

Рис. 2. Структура элемента списка

Данная структура позволяет выделять произвольные по размеру участки памяти без избыточности. В таком подходе формирования отдельных чанков избыточность устраняется за счет четкого выделения необходимого количества памяти. Единственной избыточностью в данном подходе является формирование заголовка для каждого отдельного чанка. В худшем случае это добавляет к “полезному” размеру 24 байта. Однако данный параметр настраиваем и может быть адаптирован под отдельные случаи. Так, например, в случаях, если общий объем статической области не превышает 256 байт, то становится возможным использование размеров добавочных полей, равных 1 байту и избыточность составит всего 3 байта.

По сравнению с подходом с фиксированного размера участками, данный подход также позволяет эффективно выполнять автоматическую дефрагментацию при излишней раздробленности статической области памяти.

Основная идея автоматической дефрагментации заключается в использовании для выделения памяти, вместо обычного указателя, двойного указателя. Двойной указатель указывает на место хранения указателя на выделенный участок. Такой подход позволяет скрыть от пользователя подмену указателя на адрес выделенного чанка.

Это осуществимо дополнением структуры каждого элемента полем указания на адрес начала выделяемого участка (рис. 3).

Название поля	Адрес предыдущего участка	Адрес следующего участка	Размер текущего участка	Индекс выделенного участка	Выделяемый участок данных
Возможный размер	UINT8 или UINT16 или UINT32 или UINT64	UINT8 или UINT16 или UINT32 или UINT64	UINT8 или UINT16 или UINT32 или UINT64	UINT8 или UINT16 или UINT32 или UINT64	UINT8 [*]

Рис. 3. Структура элемента списка с поддержанием возможности автоматической дефрагментации

Также необходимо завести отдельный массив, который будет хранить указатели на выделенные участки памяти. Индексы данного массива связаны со структурами элементов списка. Общая структура примера модели представлена на рис. 4.

Как можно заметить, данная модель строится на базе двусвязного списка. В заголовке каждого выделенного участка памяти хранится индекс элемента из массива, содержащего указатели на выделенные участки памяти. При выделении памяти пользователю возвращается указатель на элемент массива, хранящего адрес выделенного участка памяти.

Таким образом, пользователь работает не напрямую с адресами выделенных участков памяти, а с указателями на области, где хранится указатель на выделенную область памяти. Это позволяет, скрыто от пользователя, выполнять дефрагментацию вручную пользователем, либо во время выделения/освобождения памяти по заданному критерию степени фрагментации (например, когда есть определенное количество малых фрагментов невыделенной памяти).

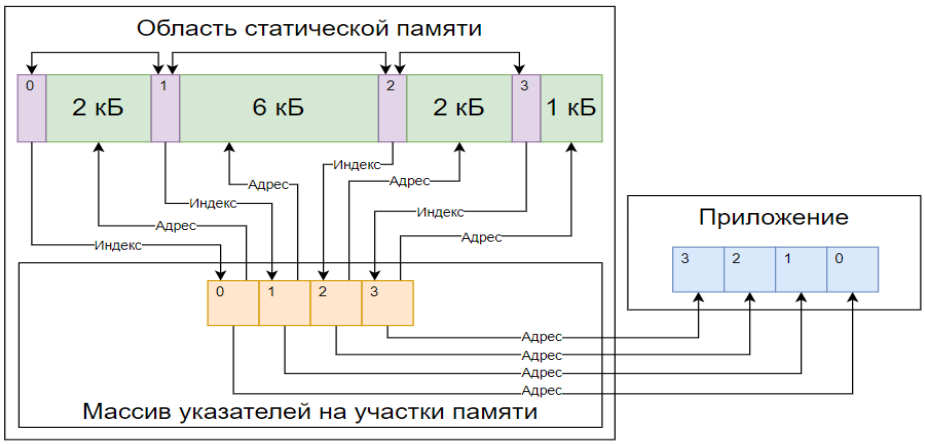

Рис. 4. Пример общей структуры работы модели аллокатора:
Зеленый - используемая пользователем выделяемая область памяти
Фиолетовый - заголовки каждого выделенного участка памяти
Оранжевый - массив, хранящий адреса начала выделенных участков памяти
Голубой - адреса элементов массива указателей на выделенные участки памяти

Особое внимание в данном случае стоит уделить работе с данными областями памяти. В классическом подходе работа с выделенной областью памяти происходит напрямую, через прямой указатель на нее. Однако, для текущей модели такой подход неприемлем, так как доступ к области памяти осуществляется посредством двойного указателя.

Для возможности работы с такой моделью аллокации предлагается введение отдельных структур данных, аналогичных классическим указателям и позволяющим выполнять операции классической арифметики указателей. Такая структура должна содержать в себе непосредственно двойной указатель на выделенную область памяти, а также смещение, которое будет обеспечивать выполнение арифметики указателей. Для удобства работы с данными сущностями создаются наборы функций. Пользовательский интерфейс работы с описываемой моделью аллокации представлена на рис. 5.

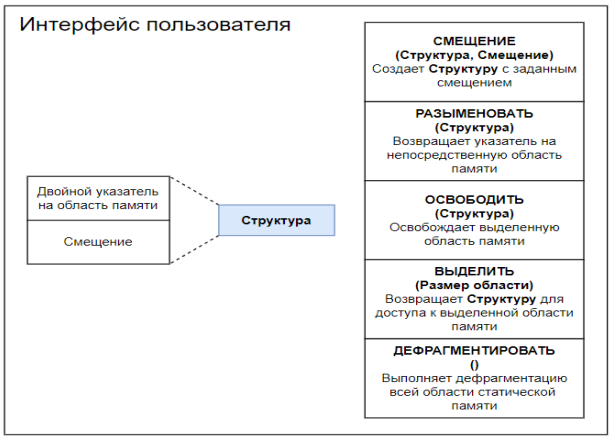

Рис. 5. Интерфейс пользователя для работы с предлагаемой моделью статической аллокации

Как итог, предлагаемая модель для работы с динамической аллокацией на статической области памяти позволяет выделять произвольные по размеру участки памяти. За счет использования заголовков для каждого выделяемого участка памяти удается добиться алгоритмической сложности поиска свободного участка памяти равного *O(N)*, где *N* - количество выделенных участков памяти. Сложность также возможно оптимизировать за счет кеширования адреса заголовка последнего выделенного участка памяти, в таком случае удастся добиться константной алгоритмической сложности *O(1)*. Кроме того, предлагаемая модель позволяет наиболее эффективно использовать всю доступную область памяти за счет периодической дефрагментации, позволяющей увеличить плотность хранения выделенных участков памяти. Однако, алгоритм дефрагментации имеет сложность, в лучшем случае, *O(N)*, где *N* - количество выделенных участков памяти.

Тем не менее, данная модель также содержит ряд недостатков. Один из наиболее существенных - большая стоимость доступа к элементам выделенного области памяти. Это обусловлено необходимостью доступа к выделенной области памяти посредством двойного разыменования адреса, в то время как при классическом подходе разыменование выполняется единожды. Помимо этого, поскольку для доступа к элементам выделенной памяти, необходимо использование интерфейса, может стать затруднительным использование некоторых сторонних библиотек, в которых предполагается использование стандартных механизмов адресации.

Заключение

В данной работе был предложен подход, позволяющий выполнять динамическое выделение памяти на статической области. Описан ряд преимуществ, которые привносит данная модель аллокации, среди которых возможность выделения произвольных по размеру участков памяти, а также возможность автоматической или ручной дефрагментации. Также был описан ряд недостатков, наиболее существенными среди которых являются увеличенные затраты ресурсов вычислительного устройства при получении доступа к выделенной области памяти, а также потенциальная сложность использования данного подхода при работе с внешними библиотеками.

В дальнейшем предполагается доработка модели аллокации, в частности введение возможности настройки отдельных ее компонент.

Список использованных источников

1. Scott M. Programming Language Pragmatics, Third Edition – 2018.

2. Yongwei’s Programming Page // Design and Implementation of a Static Memory Pool: сайт. – URL: http://wyw.dcweb.cn/static_mem_pool.html.

3. Compilers // Writing a Pool Allocator: сайт. – URL: http://dmitrysoshnikov.com/compilers/writing-a-pool-allocator/#block-allocation.

4. Medium // Memory Pool Techniques in C++: сайт. – URL: https://medium.com/@threehappyer/memory-pool-techniques-in-c-79e01f6d2b19.

5. Pink Squirrel Labs // Fixed Memory Pool Design: сайт. – URL: http://www.pinksquirrellabs.com/blog/2018/01/31/-fixed-memory-pool-design/.