САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
Математико-механический факультет

Кафедра Системного Программирования

Шашкова Елизавета Михайловна

Методы профилирования реализаций кучи

Курсовая работа

Научный руководитель:
к. ф.-м. н. Д.Ю. Булычев

Санкт-Петербург
2013

Оглавление

Введение

Динамическое управление памятью — это управление памятью для объектов, ни размер, ни время жизни которых, вообще говоря, неизвестны до выполнения программы. Под объектом мы будем понимать некоторый неделимый набор байт.

Обычно при динамическом управлении памятью место под объекты выделяется в специальной области памяти — "куче"(heap). Существует две основных операции для управления памятью в куче: выделение памяти и освобождение. При выделении посылается запрос, в котором указывается количество требуемой памяти. При успешном выделении память изымается из кучи, то есть становится недоступна для последующего выделения. Освобождение памяти, наоборот, делает часть кучи доступной для последующего выделения.

Поведение динамической памяти удобно описывать в терминах взаимодействия двух основных сущностей: *аллокатора* и *мутатора*. Аллокатор принимает запросы на выделение памяти в куче и её освобождение. Мутатор посылает данные запросы. Каждый запрос содержит количество памяти для выделения, причем это количество, вообще говоря, становится известно только в момент исполнения программы.

Эффективность работы программы во многом зависит от эффективности аллокатора. На сегодняшний момент существует много способов его реализации; более того, развитие технологий стимулирует создание новых подходов. Поэтому становится важной задача уметь сравнивать разные реализации аллокаторов, находить наиболее подходящие реализации в данном конкретном случае.

Для описания свойств аллокатора можно выделить два полярных подхода:

1. Рассмотрение свойств кучи с точки зрения поведения мутатора. Задача: определить, для какого класса программ данная реализация аллокатора подойдет лучше всего.

2. Сравнительный анализ нескольких реализаций аллокатора. Задача: провести сравнение двух или нескольких аллокаторов при взаимодействии с одним и тем же мутатором.

Основные свойства, которыми можно описать аллокатор в первом подходе:

- Суммарное количество выделенной памяти (в байтах и блоках).

- Наибольшее количество единовременно выделенной памяти за все время работы программы (в байтах и блоках).

- Среднее время жизни блока (в байтах). Это время вычисляется, как количество байт, помещенных в кучу за промежуток времени между созданием и удалением

объекта, поделенное на размер объекта (среднее время жизни одного байта в куче).

Основные свойства, которыми можно описать аллокатор во втором подходе:

- Фрагментация. Фрагментация вычисляется по следующей формуле:

$$F = \frac{free - free_{max}}{free}$$

где $free$ — количество свободных байт в куче, $free_{max}$ — размер (в байтах) наибольшего свободного блока. Так как память в куче выделяется блоками (наборами байтов), может так случиться, что в куче не осталось места для блока требуемого размера, хотя количество свободной памяти еще превышает размер этого блока. С помощью данной формулы фрагментацию можно описать численной величиной. Чем меньше эта величина, тем "лучше" текущее состояние кучи. Фрагментации равна нулю, когда свободное место в куче является одним блоком, и приближается к единице, когда размеры всех свободных блоков в куче малы.

- Время выделения (в единицах времени).

- Порог работоспособности (количество запросов, обработанных аллокатором).

- Зависимость скорости работы аллокатора от нагрузки.

Цель данной курсовой работы заключается в том, чтобы изучить методы оценки реализации аллокатора. Необходимо понять, насколько эффективно использование того или иного аллокатора, определить критерии, по которым различные аллокаторы будут сравниваться. В соответствии с основными подходами, нужно сравнить поведение одного аллокатора при использовании нескольких различных мутаторов, а также сравнить поведение нескольких аллокаторов при работе с одним мутатором. В результате будет создан прототип системы для проведения вышеуказанных сравнений.

1. Существующие подходы и метрики

Для сравнения различных реализаций аллокаторов прежде всего нужно выбрать метрики, по которым будет производиться сравнение. Требуется понять, какие данные нужно собирать при запуске программ с использованием различных аллокаторов.

В статье [1] представлены опытные данные, полученные с помощью шести программ на языке C. Авторы запустили каждую программу на нескольких наборах входных данных и построили распределения для следующих величин:

- размеров объектов;
- времён жизни объектов;
- интервалов времени между запросами.

Для этих распределений были найдены среднее, медиана и мода, которые характеризуют асимметричность распределения. Распределение размеров объектов оказалось асимметричным в сторону маленьких размеров: бóльшая часть объектов, под которые выделялась память, имели размер менее 64 байт. Интересным оказался тот факт, что 90% объектов имеют всего 10 различных размеров. Распределение времени жизни объектов показало, что большинство объектов живет в течение короткого промежутка времени: 40-80 машинных инструкций процессора

Данные об использовании памяти были получены посредством трассировки с помощью методики abstract execution [2]. В качестве тестовых программ для измерения были взяты: программа, находящая факториал больших целых чисел; оптимизатор Espresso; GhostScript[1] — интерпретатор языка PostScript; Gnu Awk[2] — интерпретатор языка AWK; интерпретатор Perl[3]; Chameleon — канальный роутер N-го уровня.

Такой способ анализа аллокаторов нельзя полностью отнести ни к первому, ни ко второму подходу, т.к. в программах аллокатор и мутатор были фиксированными, изменялись только входные данные. Приоритетом для авторов являлось исследование свойств объектов, для которых выделяется память, а не аллокаторов.

В статье [3] авторы анализировали выделение памяти в нескольких приложениях, чтобы показать, что фрагментацию во время исполнения программы можно уменьшить посредством правильного выбора аллокатора. Было показано, что объекты, которые помещаются в кучу одновременно, удаляются из неё также одновременно. Параметры, измерявшиеся в процессе работы программ, были таковы:

[1] http://www.ghostscript.com/
[2] http://www.gnu.org/software/gawk/
[3] http://www.perl.org/

- время работы;
- суммарное количество выделенных объектов и байт;
- средний размер объектов;
- наибольшее количество объектов и байт, одновременно находящихся в куче;
- среднее время жизни объектов (в байтах).

Также, как и в предыдущей статье, было показано, что динамические объекты имеют обычно несколько определенных размеров. Был сделан вывод, что для уменьшения фрагментации при создании системы выделения памяти нужно опираться прежде всего на данное свойство программ.

В данном исследовании авторы использовали как первый, так и второй подход. Они измеряли фрагментацию по всевозможным комбинациям имеющихся у них мутаторов и аллокаторов.

В [4] авторы описывают созданный ими универсальный мутатор — ACDC Benchmark Tool. Он представляет собой измеритель, который может эмулировать выделение и очистку памяти и показывать различия между реализациями аллокаторов. Основная идея, заложенная в данном измерителе, заключается в том, чтобы рассматривать в качестве метрики время и измерять его в байтах. Это было сделано для работы не только с однопоточными, но и с многопоточными мутаторами.

С помощью данного инструмента можно осуществить измерения в рамках как первого, так и второго подхода.

2. Проведение измерений

В данной работе были реализованы оба подхода к сбору данных. Для первого подхода, который заключается в рассмотрении свойств аллокатора с точки зрения мутатора, измерялись следующие величины:

- суммарное количество выделенной памяти (в байтах и блоках);
- наибольшее количество единовременно выделенной памяти за все время работы программы (в байтах и блоках);
- среднее время жизни блока (в байтах).

Для второго подхода, основная цель которого — сравнить аллокаторы с использованием одного мутатора, измерялись:

- фрагментация;
- время выделения (в единицах времени);
- порог работоспособности (количество запросов, обработанных аллокатором);
- зависимость скорости работы аллокатора от нагрузки.

2.1. Аллокаторы и мутаторы

Сбор данных производился как со стандартной реализации аллокатора в glibc, так и с собственной реализации, созданной Самофаловым А.В. [5]. В его работе было реализовано несколько аллокаторов с различными алгоритмами выделения памяти.

Эти алгоритмы:

- “First fit”— память выделяется в первом свободном блоке;
- “Best fit”— память выделяется в наиболее подходящем блоке;
- “Worst fit”— память выделяется в самом большом свободном блоке;
- “Selfcompact”— самосжимающаяся куча.

В качестве мутаторов были взяты следующие программы:

- архиватор zip[4];
- программа, сортирующая связный список — sort list;
- архиватор tar[5].

[4]http://www.info-zip.org
[5]http://www.gnu.org/software/tar/

2.2. Подробности реализации

В стандартной библиотеке языка C существует специальная структура `mallinfo`, которая описывает состояние кучи. Данная структура имеет следующие поля:

- `int arena` — суммарное пространство, выделенное под кучу (в байтах).
- `int ordblks` — количество свободных блоков.
- `int smblks` — количество свободных блоков “fastbin”. Это блоки, которые были освобождены, однако, не были объединены с соседними свободными блоками. В этих блоках впоследствии можно выделять память под объекты такого же размера.
- `int hblks` — суммарное количество блоков, находящихся в куче.
- `int hblkhd` — суммарное место, выделенное под блоки (в байтах).
- `int usmblks` — наибольшее количество байт, находящихся в куче за время работы программы.
- `int fsmblks` — суммарное место в блоках “fastbin”(в байтах).
- `int uordblks` — суммарное занятое пространство (в байтах).
- `int fordblks` — суммарное свободное пространство (в байтах).
- `int keepcost` — наибольшее количество пространства, которое можно освободить в начале кучи (то есть посредством функции `malloc_trim`) (в байтах).

Функция `mallinfo` возвращает копию структуры `mallinfo`. С её помощью были измерены некоторые величины для стандартной реализации аллокатора. Для осуществления измерений были использованы hook-функции. Переменные `__malloc_hook`, `__realloc_hook` и `__free_hook` являются указателями на функции, которые используют функции `malloc`, `realloc` и `free` соответственно всякий раз, когда они вызываются. Таким образом можно привязывать вызов требуемых функций (в нашем случае — сбор статистики) к запросам на выделение или очистку памяти.

В собственной реализации для сбора статистики был создан аналог структуры `mallinfo` — структура `myinfo`. Её устройство схоже с устройством `mallinfo`, однако, имеются некоторые различия. Она содержит следующие поля:

- `int arena` — суммарное пространство, выделенное под кучу (в байтах);
- `int freemem` — свободное пространство (в байтах);

- `int usdmem` — занятое пространство (в байтах);
- `int freeblks` — количество свободных блоков;
- `nt usdblks` — количество используемых блоков;
- `maxfreeblk` — размер наибольшего свободного блока (в байтах).

Основным отличием данной структуры от структуры `mallinfo` является наличие поля с размером наибольшего свободного блока. Оно позволяет напрямую использовать это значение для вычисления фрагментации (по формуле, упоминавшейся во введении).

3. Результаты

Результаты измерений в первом подходе — данные, усредненные по выполнению программы. Результаты представлены в следующих таблицах. Max(bytes) обозначает наибольшее количество единовременно выделенной памяти(в байтах). Sum(bytes) обозначает общее количество выделенной памяти(в байтах). Mean lifetime(bytes) обозначает среднее время жизни блока(в байтах).

First fit:

Мутатор	Max(bytes)	Sum(bytes)	Mean lifetime(bytes)
zip	21064	25845	1068.48
sort list	808	1592	82.21
tar	52080	58736	446.86

Best fit:

Мутатор	Max(bytes)	Sum(bytes)	Mean lifetime(bytes)
zip	21080	25845	1068.64
sort list	808	1592	82.27
tar	52112	58736	451.89

Worst fit:

Мутатор	Max(bytes)	Sum(bytes)	Mean lifetime(bytes)
zip	21064	25845	1069.85
sort list	808	1592	80.72
tar	52080	58736	455.58

Средние величины для всех аллокаторов получились почти одинаковые. Данный подход к изучению аллокаторов не выявил их различий и особенностей.

Во втором подходе измерения и сравнения производились после каждого запроса, которыми обменивались между собой аллокатор и мутатор. В связи с этим, полученные данные удобно представить в виде графиков.

На Рис. 1 представлено изменение фрагментации для алгоритмов First fit, Best fit и Worst fit при работе с программой zip. Из пяти возможных аллокаторов величину фрагментации с мутатором zip удалось найти только для данных трех. Это связано с тем, что для стандартной реализации аллокатора фрагментацию по нашей формуле найти не удастся, так как нет возможности узнать размер наибольшего свободного блока. А аллокатор selfcompact требует специфичного написания мутатора и его компиляции, поэтому запустить этот аллокатор с мутатором zip не представляется возможным.

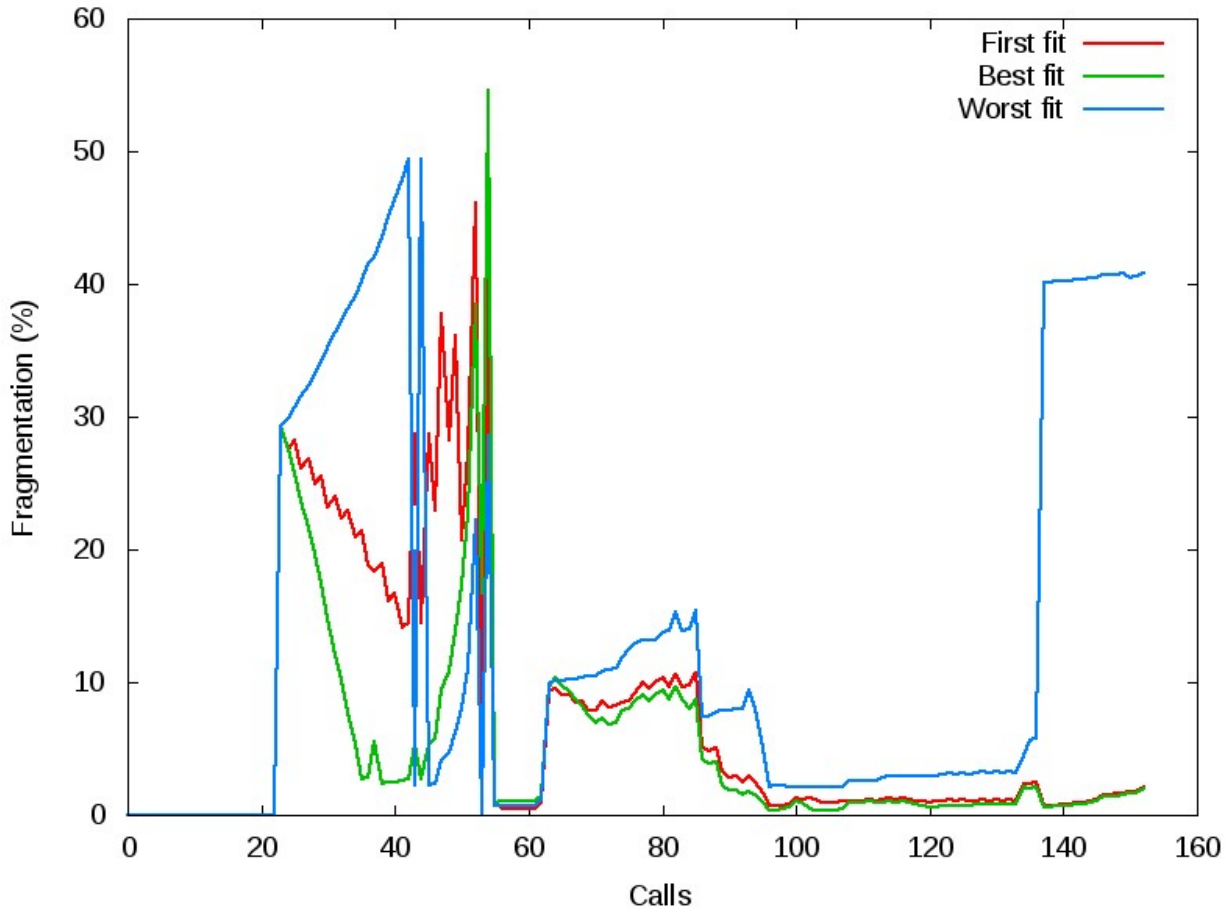

Рис. 1: Величина фрагментации с мутатором zip

Можно заметить, что первом участке у всех трех аллокаторов величина фрагментации равна 0. Это связано с тем, что освобождения памяти еще не производились и объекты находятся в куче последовательно, без пропусков. Затем фрагментация начинает возрастать. Как и предполагалось теоретически, почти на всех участках графика фрагментация аллокатора Worst fit больше фрагментации остальных аллокаторов, а фрагментация аллокатора Best fit меньше фрагментации остальных аллокаторов.

Можно отметить резкий скачок, а затем резкое уменьшение фрагментации в районе 60-го запроса. Для объяснения такого поведения, рассмотрим график размера кучи, например, для аллокатора Worst fit(см. Рис. 2). На нем представлены размер кучи, размер свободного пространства в куче и величина максимального свободного блока. Тогда становится понятным, что резкое изменение фрагментации в районе 60-го запроса связано с увеличением размера кучи (добавляется большой свободный блок). Подобное повторяется в районе 82-го и 98-го запросов, однако, там фрагментация изменяется не столь резко. Это связано с тем, что общий размер кучи к тому моменту уже значительно увеличился.

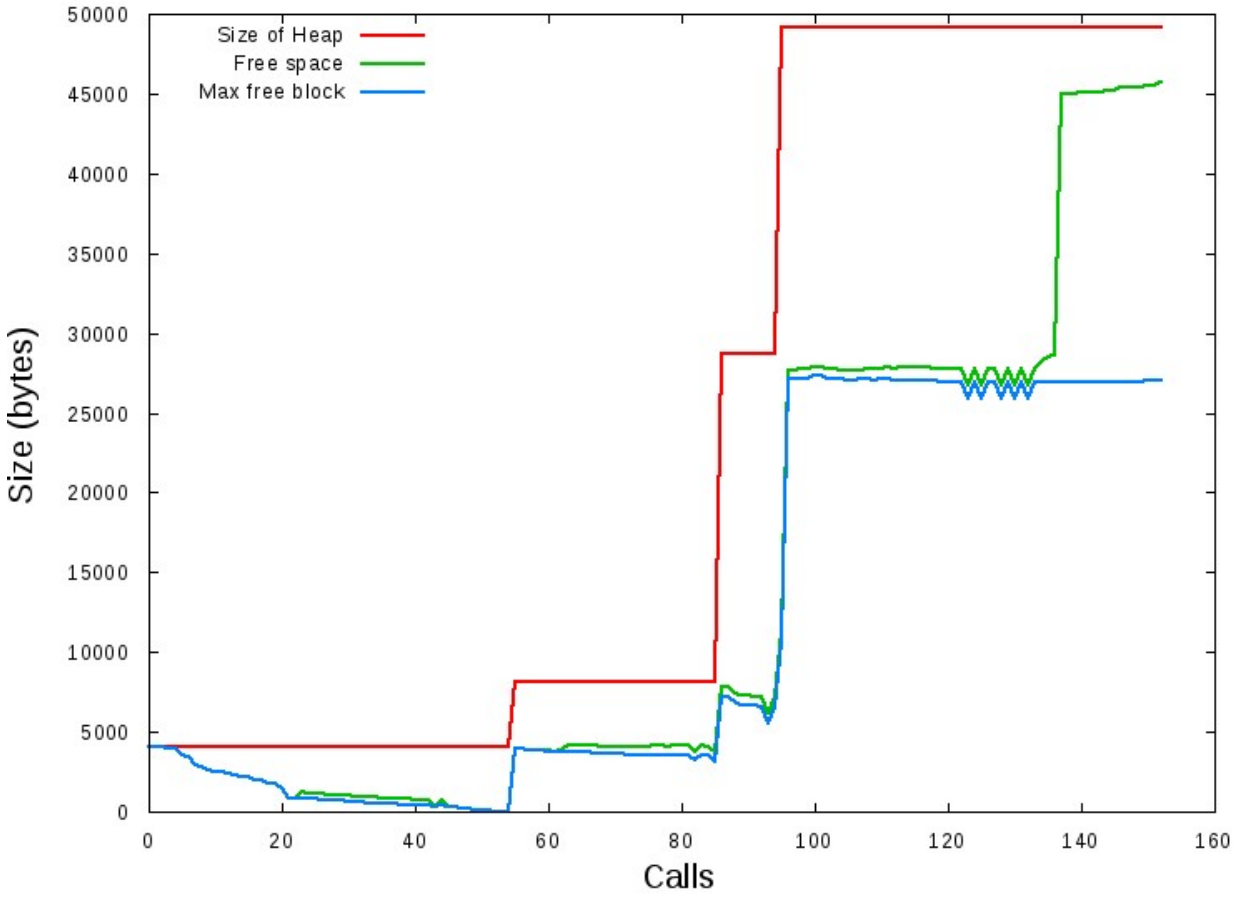

Рис. 2: Состояние кучи с аллокаторм Worst fit и мутатором zip

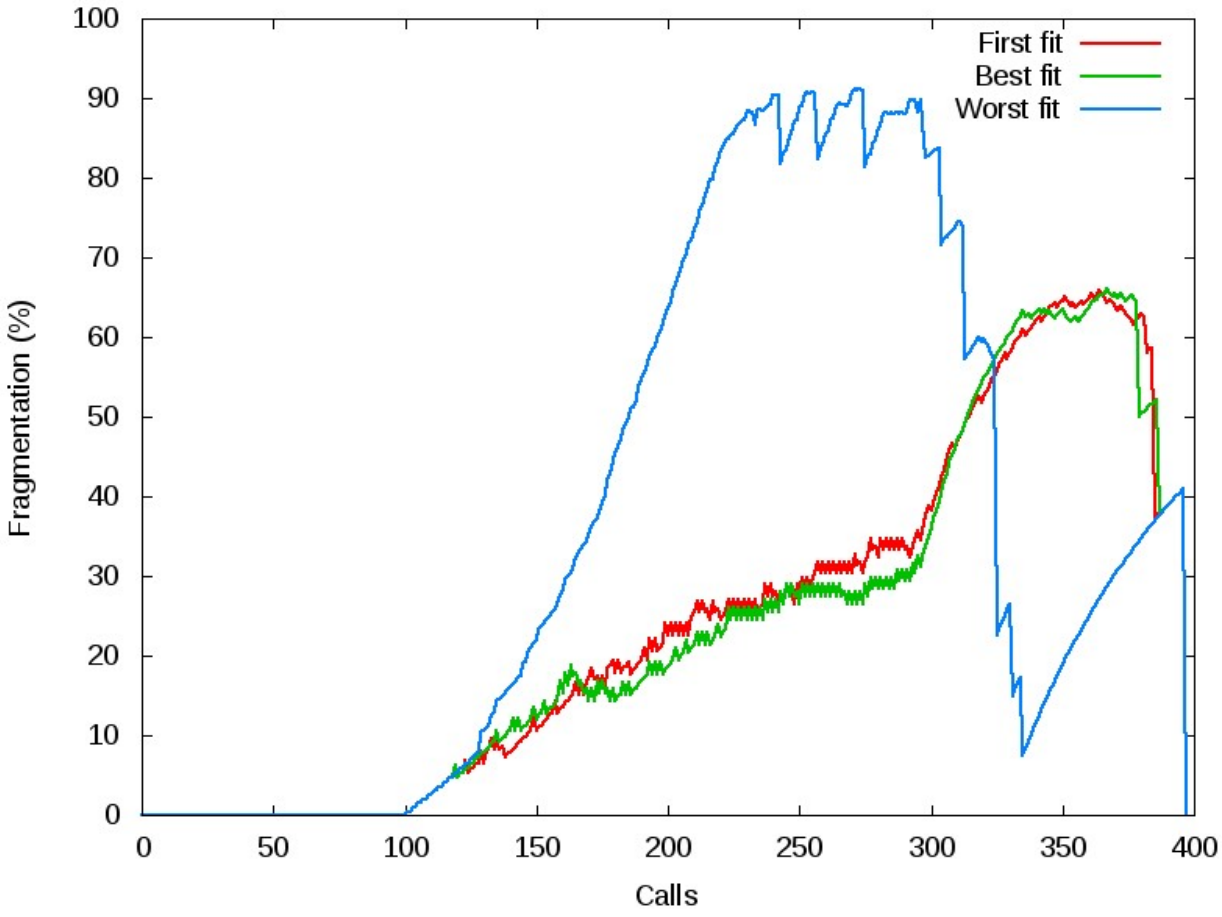

Рис. 3: Величина фрагментации с мутатором sort list

На Рис. 3 представлена величина фрагментации в процессе исполнения программы — сортировки связного списка — для четырех аллокаторов. На первом участке графика видно, что для всех аллокатором величина фрагментации равна 0. На этом участке список создается, объекты располагаются в куче последовательно и все свободное пространство является одним блоком.

Затем, в процессе сортировки, некоторые объекты удаляются, под новые выделяется память. Здесь сразу можно заметить, что фрагментация алгоритма Worst fit растет быстрее фрагментации других алгоритмов. Около 300-го запроса можно отметить увеличение фрагментации у алгоритмов First fit и Best fit.

Хочется отметить, что сбор статистики помог выявить ошибку в первоначальной реализации аллокатора First fit. На Рис. 4 представлено, как изменялась фрагментация на одной из первых тестовых программ. Такое резкое изменение фрагментации показалось нам неестественным. Благодаря этому была проведена перепроверка реализации и выявлена ошибка.

Также для различных алгоритмов измерялось время, затраченное аллокаторами на выполнение запросов. На Рис. 5 представлены результаты измерения времени в процессе работы программы. Можно заметить, что при некоторых запросах, время в несколько раз превышает среднее время на выделение. Это происходит на запросах о выделении памяти, и, вероятно, это связано с поиском подходящего свободного блока.

На Рис. 6 представлена зависимость времени выделения от заполненности кучи. Наиболее выражена эта зависимость для алгоритма First fit. Для остальных реализаций время выделения увеличивается, однако, не настолько явно.

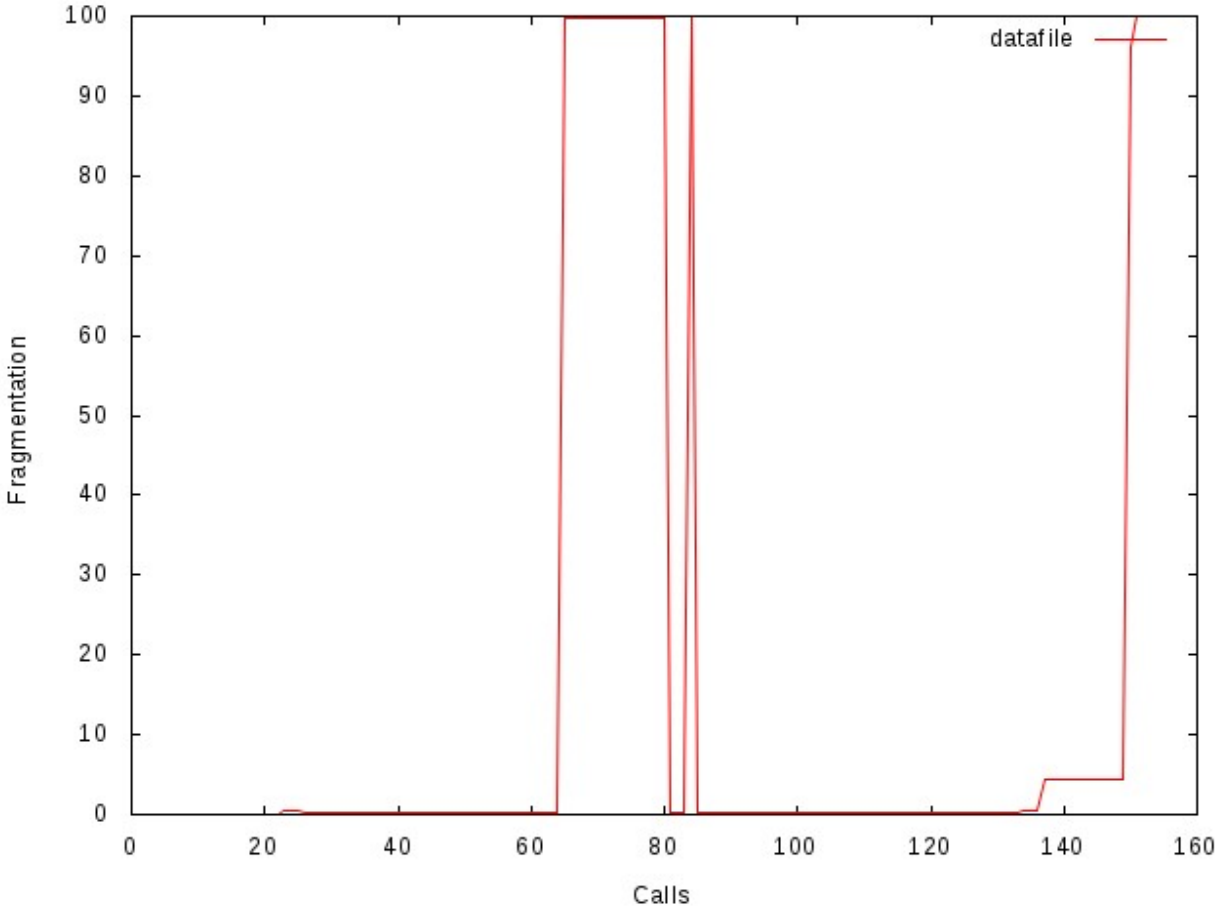

Рис. 4: Величина фрагментации аллокатора First fit

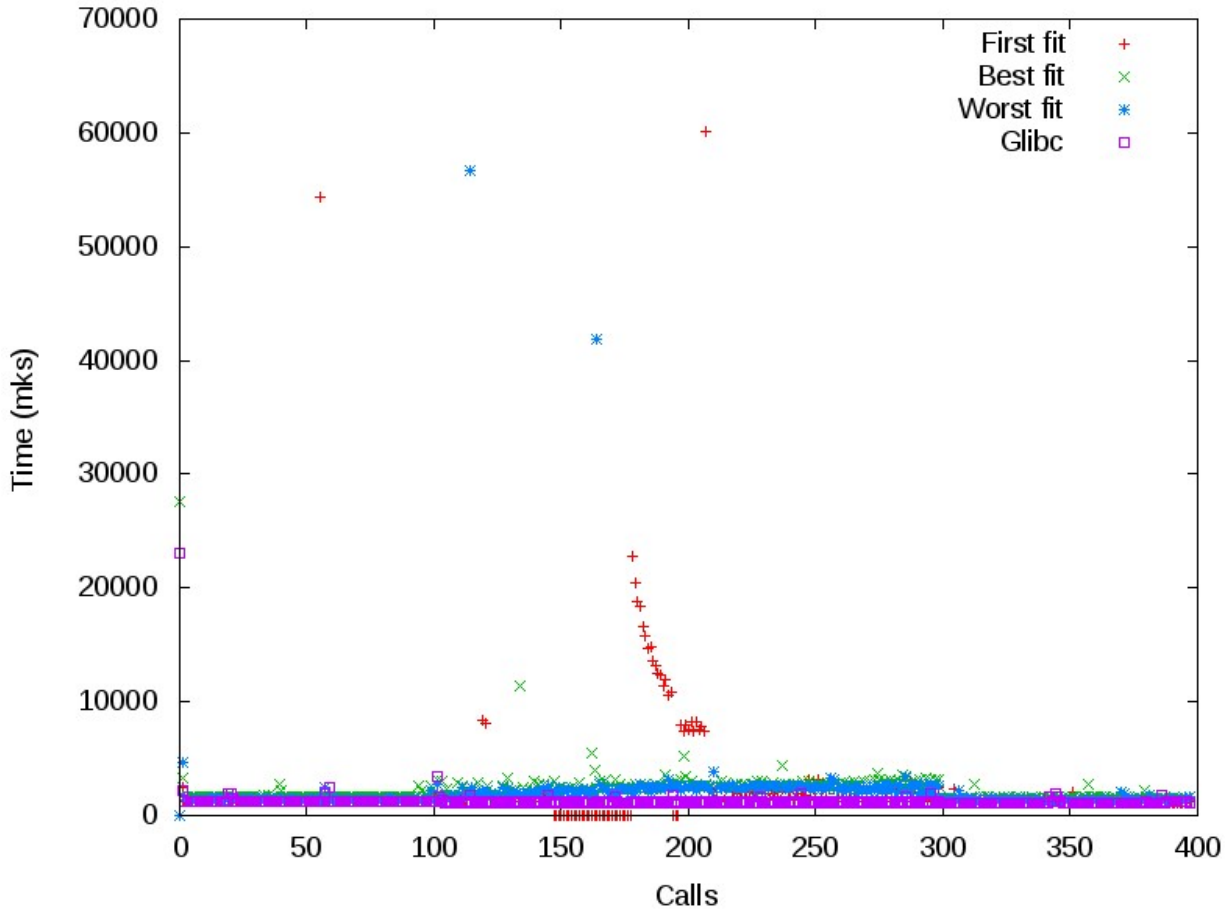

Рис. 5: Время выполнения запросов

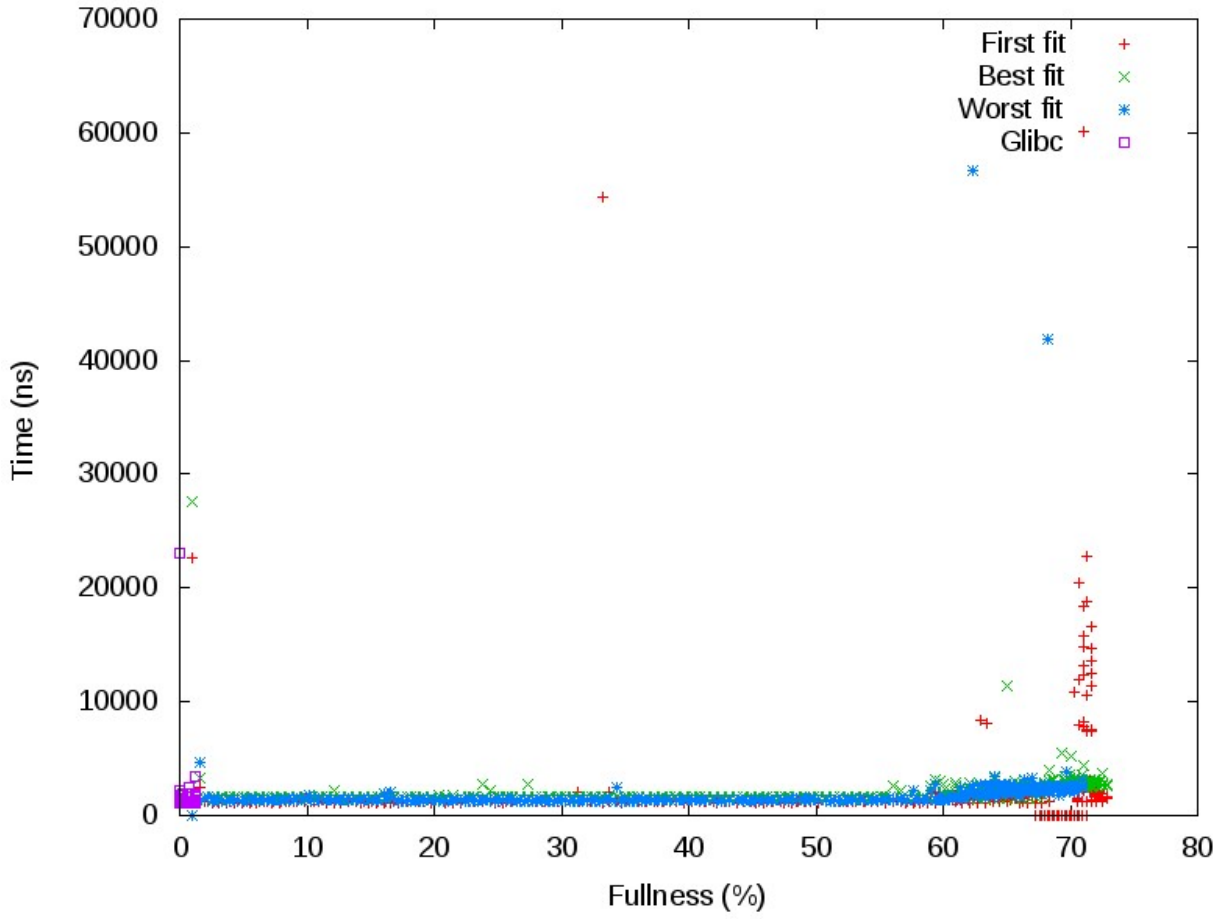

Рис. 6: Зависимость времени выполнения запросов от заполненности кучи

4. Заключение

В результате работы над данной курсовой работой:

- были изучены существующие метрики для сравнения реализаций кучи;
- были изучены подходы для проведения измерений;
- была создана система сравнения различных реализаций;
- была выявлена ошибка в реализации кучи.

Список литературы

[1] Benjamin Zorn Dirk Grunwald. Empirical Measurements of Six Allocation-intensive C Programs. — University of Colorado, Boulder, 1992.

[2] Larus James R. Abstract Execution: a Technique for Efficiently Tracing Programs. — Software — Practice and Experience, 20(12):1241—1258, 1991.

[3] Mark S. Johnstone Paul R. Wilson. The Memory Fragmentation Problem: Solved? — The University of Texas at Austin, 1997.

[4] Martin Aigner. Christoph M. Kirsch. Towards a Universal Mutator for Benchmarking Heap Management Systems. — University of Salzburg, 2012.

[5] А.В. Самофалов. Способы реализации кучи и их свойства. — СПбГУ. Курсовая работа, 2013.