Московский авиационный институт (национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №3 по курсу «Дискретный анализ»

Студент: Л. Я. Вельтман
Преподаватель: А. А. Кухтичев
Группа: М8О-207Б

Дата:
Оценка:
Подпись:

Москва, 2018

Лабораторная работа №3

Задача: Для реализации словаря из предыдущей лабораторной работы, необходимо провести исследование скорости выполнения и потребления оперативной памяти. В случае выявления ошибок или явных недочётов, требуется их исправить.

Используемые утилиты: Valgrind, gprof.

1 Описание

Чтобы написать качественную программу, нужно уметь ее отлаживать. Для этого существуют специальные утилиты, которые расширяют наши возможности в отслеживании поведении программы на этапе исполнения.
Профилирование позволяет изучить, где программа расходует свое время и какие функции вызывали другие функции, пока программа исполнялась. Эта информация может указать на ту часть программы, которая исполняется медленнее, чем ожидалось, что служит неким сигналом о возможном переписывании этого участка кода, чтобы ускорить выполнение всей программы. Эта информация также подскажет, какие функции вызывались чаще или реже, чем предполагалось. Это может помочь отметить ошибки, которые иначе остались бы незамеченными. Так как профилятор использует информацию, собранную в процессе реального исполнения программы, он может быть использован для исследования программ, которые слишком большие или слишком сложные, чтобы анализировать их, читая исходный текст.

Gprof - это инструмент анализа производительности для приложений Unix.

Valgrind — инструментальное программное обеспечение, предназначенное для отладки использования памяти, обнаружения утечек памяти, а также профилирования. Особое внимание привлекли memcheck и massif.

- memcheck - основной модуль, обеспечивающий обнаружение утечек памяти, и прочих ошибок, связанных с неправильной работой с областями памяти — чтением или записью за пределами выделенных регионов и т.п.
- massif - позволяет проанализировать выделение памяти различными частями программы.

Мною была использована утилита gprof для поиска наиболее медленных мест программы. Для поиска утечек памяти я использовала Valgrind. Оказалось, что на Mac OS X он работает не совсем корректно, а точнее, из-за особенностей ядра, он указывает на утечки там, где их нет (в некоторых системных библиотеках, например). Пришлось использовать несколько разных систем, чтобы добиться максимальной работоспособности программы.

2 Консоль

На ранних этапах своей работы я использовала только Valgrind для исправления возникавших ошибок и утечек.

```
==77== HEAP SUMMARY:
==77==     in use at exit: 74,226 bytes in 57 blocks
==77==   total heap usage: 59 allocs,2 frees,75,250 bytes allocated
==77==
==77== LEAK SUMMARY:
==77==    definitely lost: 144 bytes in 3 blocks
==77==    indirectly lost: 342 bytes in 15 blocks
==77==      possibly lost: 0 bytes in 0 blocks
==77==    still reachable: 73,740 bytes in 39 blocks
==77==         suppressed: 0 bytes in 0 blocks
==77==
==77== For counts of detected and suppressed errors,rerun with: -v
==77== ERROR SUMMARY: 2 errors from 2 contexts (suppressed: 0 from 0)
Ошибка сегментирования (сделан дамп памяти)
```

Я обращалась к несуществующей области памяти при поиске узла. Это было исправлено введением sentinel узла(nil).

```
==62== HEAP SUMMARY:
==62==     in use at exit: 97,004 bytes in 901 blocks
==62==   total heap usage: 1,011 allocs,110 frees,100,954 bytes allocated
==62==
==62== 24,300 (288 direct,24,012 indirect) bytes in 6 blocks are definitely
lost in loss record 3 of 4
==62==    at 0x4C2E0EF: operator new(unsigned long) (in /usr/lib/valgrind/vgpreload_me
==62==    by 0x401910: TRBTree::Insertion(char*,unsigned long) (in /mnt/c/Users/user/d
==62==    by 0x40284E: main (in /mnt/c/Users/user/desktop/da2/da2)
==62==
==62== LEAK SUMMARY:
==62==    definitely lost: 288 bytes in 6 blocks
==62==    indirectly lost: 24,012 bytes in 894 blocks
==62==      possibly lost: 0 bytes in 0 blocks
==62==    still reachable: 72,704 bytes in 1 blocks
==62==         suppressed: 0 bytes in 0 blocks
==62== Reachable blocks (those to which a pointer was found) are not shown.
==62== To see them,rerun with: --leak-check=full --show-leak-kinds=all
```

```
==62==
==62== For counts of detected and suppressed errors,rerun with: -v
==62== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
```

Ошибка сегментирования пропала, но количество выделений памяти и ее освобождения намного различались. Это означало, что что-то не так с удалением узлов. Проблема была в деструкторах. Я неправильно освобождала память, а точнее вообще не освобождала. Исправив данную ситуацию, я запустила Valgrind еще раз.

```
==1417== HEAP SUMMARY:
==1417== in use at exit: 72,704 bytes in 1 blocks
==1417== total heap usage: 1,012 allocs,1,011 frees,101,211 bytes allocated
==1417==
==1417== Searching for pointers to 1 not-freed blocks
==1417== Checked 106,456 bytes
==1417==
==1417== 72,704 bytes in 1 blocks are still reachable in loss record 1 of 1
==1417== at 0x4C2DB8F: malloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==1417== by 0x4EC3EFF: ??? (in /usr/lib/x86_64-linux-gnu/libstdc++.so.6.0.21)
==1417== by 0x40106B9: call_init.part.0 (dl-init.c:72)
==1417== by 0x40107CA: call_init (dl-init.c:30)
==1417== by 0x40107CA: _dl_init (dl-init.c:120)
==1417== by 0x4000C69: ??? (in /lib/x86_64-linux-gnu/ld-2.23.so)
==1417==
==1417== LEAK SUMMARY:
==1417== definitely lost: 0 bytes in 0 blocks
==1417== indirectly lost: 0 bytes in 0 blocks
==1417== possibly lost: 0 bytes in 0 blocks
==1417== still reachable: 72,704 bytes in 1 blocks
==1417== suppressed: 0 bytes in 0 blocks
==1417==
==1417== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
==1417== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Насколько я поняла, 1 неосвобожденный блок не у меня в коде, а в стандартных библиотечных функциях. Все же ошибки пропали. Несмотря на отсутствие ошибок, моя программа не проходила на 4 тесте (time limit exceeded). Было предпринято решение начать оптимизировать код. Изменив все std::cin.get() и std::cout на getchar_unlocked() и printf(), я подсчитала время работы программы, оно сократилось. Отправив ее на чекер, я была удивлена тем, что вернулась на шаг назад, на 3 тест (wrong answer). Посмотрев на свои предыдущие тесты, я заметила, что они написаны только для вставки, решила переделать алгоритм удаления. Это не помогло.

Очень долго думая над этой ошибкой, мне подсказали, что я поставила пробел в одной строчке, когда выводила «NoSuchWord», если слово было не найдено. Исправив, ошибка возникла опять на 4 тесте(wrong answer). Запустила утилиту gprof, которая показывает производительность программы.

```
Flat profile:

Each sample counts as 0.01 seconds.
no time accumulated

%   cumulative   self              self     total
time   seconds   seconds    calls  Ts/call  Ts/call  name
0.00      0.00     0.00  3250000     0.00     0.00  LowerCase(char)
0.00      0.00     0.00   500001     0.00     0.00  Parser(char&,char*,unsigned
long long&)
0.00      0.00     0.00   500000     0.00     0.00  TRBTree::NodeSearch(char*)
0.00      0.00     0.00   250000     0.00     0.00  TNode::TNode(char*,unsigned
long long,bool)
0.00      0.00     0.00   250000     0.00     0.00  TRBTree::FixInsert(TNode*)
0.00      0.00     0.00   250000     0.00     0.00  TRBTree::Insertion(char*,unsigned
long long,bool)
0.00      0.00     0.00   174500     0.00     0.00  TRBTree::RightRotate(TNode*)
0.00      0.00     0.00     6500     0.00     0.00  TRBTree::LeftRotate(TNode*)
0.00      0.00     0.00        1     0.00     0.00  _GLOBAL__sub_I__ZN5TNodeC2Ev
0.00      0.00     0.00        1     0.00     0.00  __static_initialization_and_destru
0.00      0.00     0.00        1     0.00     0.00  TNode::TNode()
0.00      0.00     0.00        1     0.00     0.00  TRBTree::TRBTree()
0.00      0.00     0.00        1     0.00     0.00  TRBTree::~TRBTree()
```

Если приглядеться, то функции удаления вообще нет, хотя профилирование производилось на тесте, где удаление слов присутствовало. На вход подавалось 500000 слов, 250000 должны были добавиться, и приблизительно 250000 удалиться. Заново переписала функцию удаления узла. Оказавшись в очередной раз на 4 тесте (time limit excedeed), посмотрела на профилирование программы и заметила, что метод NodeSearch вызывается ровно столько раз, сколько слов подается на обработку программе. Дело в том, что каждый раз при добавлении нового узла, вызывается функция поиска, чтобы определить существует такой узел, если нет, то происходит вставка вершины в дерево, но при этом в самой функции Insertion существует аналогичный цикл поиска для вставки. Изменила эту void функцию на bool, чтобы она возвращала false, если такое слово уже есть в дереве, и true, если слова нет и узел был добавлен в дерево. Таким образом, функция поиска вызывалась только для функции удаления. Мне показалось, что это может серьезно ускорить программу. Но все же главная

проблема заключалась в другом, я перепутала ссылки при балансировке дерева после удаления, нужно было ссылаться на родителя от действующего узла, а не от его узла-брата. Перескочила на time limit exceeded на 14 тесте. Сделала повторное профилирование с помощью gprof.

```
Flat profile:

Each sample counts as 0.01 seconds.
no time accumulated

%   cumulative   self              self     total
time   seconds   seconds    calls  Ts/call  Ts/call  name
0.00      0.00     0.00  3250000     0.00     0.00  LowerCase(char)
0.00      0.00     0.00   500001     0.00     0.00  Parser(char&,char*,unsigned
long long&)
0.00      0.00     0.00   259750     0.00     0.00  TRBTree::TransPlant(TNode*,TNode*)
0.00      0.00     0.00   250000     0.00     0.00  TRBTree::NodeSearch(char*)
0.00      0.00     0.00   250000     0.00     0.00  TRBTree::Insertion(char*,unsigned
long long,bool)
0.00      0.00     0.00   244500     0.00     0.00  TRBTree::RightRotate(TNode*)
0.00      0.00     0.00   228750     0.00     0.00  TNode::TNode(char*,unsigned
long long,bool)
0.00      0.00     0.00   228750     0.00     0.00  TNode::~TNode()
0.00      0.00     0.00   228750     0.00     0.00  TRBTree::Deletion(TNode*)
0.00      0.00     0.00   228750     0.00     0.00  TRBTree::FixInsert(TNode*)
0.00      0.00     0.00   218250     0.00     0.00  TRBTree::FixDelete(TNode*)
0.00      0.00     0.00   158250     0.00     0.00  TRBTree::LeftRotate(TNode*)
0.00      0.00     0.00    34500     0.00     0.00  TRBTree::MinValueNode(TNode*)
0.00      0.00     0.00        1     0.00     0.00  _GLOBAL__sub_I__ZN5TNodeC2Ev
0.00      0.00     0.00        1     0.00     0.00  _GLOBAL__sub_I__ZN7TRBTree10Saving
0.00      0.00     0.00        1     0.00     0.00  _GLOBAL__sub_I_main
0.00      0.00     0.00        1     0.00     0.00  __static_initialization_and_destru
0.00      0.00     0.00        1     0.00     0.00  __static_initialization_and_destru
0.00      0.00     0.00        1     0.00     0.00  __static_initialization_and_destru
0.00      0.00     0.00        1     0.00     0.00  TNode::TNode()
0.00      0.00     0.00        1     0.00     0.00  TRBTree::TRBTree()
0.00      0.00     0.00        1     0.00     0.00  TRBTree::~TRBTree()
```

Действительно, теперь функция поиска использовалась в 2 раза реже.

С помощью утилиты massif из пакета Valgrind посмотрела на количество памяти, которое выделяется при выполнении программы в зависимости от времени.

```
root@lina:/mnt/c/Users/user/desktop# g++ -g -std=c++11 main.cpp -o ./da2
root@lina:/mnt/c/Users/user/desktop# valgrind --tool=massif ./da2 <test >/dev/null
==1425== Massif,a heap profiler
==1425== Copyright (C) 2003-2015,and GNU GPL'd,by Nicholas Nethercote
==1425== Using Valgrind-3.11.0 and LibVEX; rerun with -h for copyright info
==1425== Command: ./da
==1425==
==1425== error calling PR_SET_PTRACER,vgdb might block
==1425==
root@lina:/mnt/c/Users/user/desktop# ms_print massif.out.1424 | head --lines=35
--------------------------------------------------------------------------------
Command:            ./da
Massif arguments:   (none)
ms_print arguments: massif.out.1424
--------------------------------------------------------------------------------


KB
73.48^                                                                       :
|                                                                    ::::#:@
|                                                                    :  :#:@
|                                                                    :  :#:@
|                                                                    :  :#:@
|                                                                    :  :#:@
|                                                                    :  :#:@
|                                                                    :  :#:@
|                                                                    :  :#:@
|                                                                    :  :#:@
|                                                                    :  :#:@
|                                                                    :  :#:@
|                                                                    :  :#:@
|                                                                    :  :#:@
|                                                                    :  :#:@
|                                                                    :  :#:@
|                                                                    :  :#:@
|                                                                    :  :#:@
|                                                                    :  :#:@
|                                                                    :  :#:@
0 +-------------------------------------------------------------------------->Mi
0                                                                     2.209
```

```
Number of snapshots: 52
Detailed snapshots: [9,19,25 (peak),35,45]
```

Диаграмма показалась мне странной. Потом я вспомнила, что добавила std::cin.peek() для проверки следующего символа в потоке, эта функция использовалась каждый раз при считывании очередного символа, представив, какой объем данных может подаваться на вход и что std::cin.peek() работает для каждого символа, учитывая, что она предназначалась лишь для одного случая (если встретился конец файла, то прекратить считывание последующих символов), пришла к решению убрать ее. Добавление этой функции в код было сделано из-за моей невнимательности, так как тестируя программу, забыла сделать переход на новую строку после последнего слова в файле, из-за этого возникал segmentation fault, нашла единственный выход, как раз применяя эту функцию. Запустив massif еще раз, получилась вот такая диаграмма.

```
root@lina:/mnt/c/Users/user/desktop# g++ -g -std=c++11 main.cpp -o ./da2
root@lina:/mnt/c/Users/user/desktop# valgrind --tool=massif ./da2 <test >/dev/null
==1433== Massif,a heap profiler
==1433== Copyright (C) 2003-2015,and GNU GPL'd,by Nicholas Nethercote
==1433== Using Valgrind-3.11.0 and LibVEX; rerun with -h for copyright info
==1433== Command: ./da
==1433==
==1433== error calling PR_SET_PTRACER,vgdb might block
==1433==
root@lina:/mnt/c/Users/user/desktop# ms_print massif.out.1433 | head --lines=35
--------------------------------------------------------------------------------
Command:            ./da
Massif arguments:   (none)
ms_print arguments: massif.out.1433
--------------------------------------------------------------------------------


KB
143.9^#
|# :    :   ::    :  @      :     :    :    :     :    :    :     :    :
|# ::   :   :     : :@      :     :   ::    :    ::    :    :    ::    :
|#::::  :   :    :: :@:     :    ::   ::    :@   ::   ::    ::   ::   ::
|#::::  :   :    :: :@:     :::  ::   ::   ::@   ::   ::   :::   ::   ::
|#::::: :   :    ::::@: :  :::  :::   :::  ::@  :::   :::  :::  :::  @:::
|#::::: ::: : : :::::@: :  :::  :::@  :::  ::@: :::  :::: ::::  :::  @:::
|#::::: : ::: : :::::@: : :::: ::::@  ::: :::@:::::@ :::: :::: @:::: @:::
```

```
|#::::::: ::: : :::::@:::::::: ::::@ ::::::::@:::::@:::::::::: @:::::@:::
|#::::::: ::: : :::::@:::::::: ::::@ ::::::::@:::::@:::::::::::@:::::@:::
|#::::::: ::: :::::::@:::::::: ::::@:::::::::@:::::@:::::@:::::@:::::@:::
|#::::::: ::: :::::::@:::::::: ::::@:::::::::@:::::@:::::@:::::@:::::@:::
|#::::::: ::: :::::::@:::::::: ::::@:::::::::@:::::@:::::@:::::@:::::@:::
|#::::::: ::: :::::::@:::::::: ::::@:::::::::@:::::@:::::@:::::@:::::@:::
|#::::::: ::: :::::::@:::::::: ::::@:::::::::@:::::@:::::@:::::@:::::@:::
|#::::::: ::: :::::::@:::::::: ::::@:::::::::@:::::@:::::@:::::@:::::@:::
|#::::::: ::: :::::::@:::::::: ::::@:::::::::@:::::@:::::@:::::@:::::@:::
|#::::::: ::: :::::::@:::::::: ::::@:::::::::@:::::@:::::@:::::@:::::@:::
|#::::::: ::: :::::::@:::::::: ::::@:::::::::@:::::@:::::@:::::@:::::@:::
|#::::::: ::: :::::::@:::::::: ::::@:::::::::@:::::@:::::@:::::@:::::@:::
0 +----------------------------------------------------------------------->Mi
0                                                                   752.4

Number of snapshots: 98
Detailed snapshots: [1 (peak),22,39,52,62,72,82,92]

--------------------------------------------------------------------------------
```

После я еще измерила время выполнения предыдущей версии программы и новой, сократилос почти вдвое, поэтому отправила на чекер и получила долгожданный OK.

3 Выводы

Valgrind и gprof - простые и мощные инструменты для профилирования и отладки программ. Но это не единственные утилиты, которые были использованы мной. Они не были включены в отчет, потому что их функционал очень схож. Библиотека dmallocxx имеет ограниченные возможности для отладки программы по сравнению с Valgrind. Утилита gcov была интересна тем, что показывала сколько раз какая конкретно строка была использована, что очень похоже на утилиту gprof, которая показывает количество вхождений в функции. Понравилась визуализация работы программы при помощи pprof и massif. Решила включить в отчет только график, полученный последним профилировщиком.

Исследование качества программы и ее оптимизация занимают, как мне кажется, очень важную роль в программировании. Это требует особого внимания, особенно для очень больших проектов, где любая незначительная ошибка может существенно повлиять на всю работу и привести к неприятным последствиям.

Список литературы

[1] *Valgrind.*
URL: `https://ru.wikipedia.org/wiki/Valgrind`.
(дата обращения: 7.09.2018).

[2] *Профилятор gprof.*
URL: `https://www.opennet.ru/docs/RUS/gprof/`.
(дата обращения: 23.11.2018).

[3] *Gcov.*
URL: `https://ru.wikipedia.org/wiki/Gcov`.
(дата обращения: 23.11.2018).