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: А. Е. Аксёнов
Преподаватель: А. А. Кухтичев
Группа: М8О-308Б

Дата:
Оценка:
Подпись:

Москва, 2020

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: gprof, valgrind.

1 Описание

Выполняя лабраторные работы, мы зачастую не обращаем внимания на производительность наших программ, важнее, чтобы она «зашла», то есть выдавала верные ответы при всех тестах. Однако в промышленной разработке даже минимальное проседание по производительности может сыграть злую шутку. Поэтому помимо того, чтобы программа работала корректно, необходимо, чтобы она ещё работала эффективно.

В задании к данной лабораторной работе указано, что необходимо использовать утилиты dmalloc и gprof, однако, там также было указано, что можно заменить их более известными утилитами. Я воспользуюсь этим допущением, и вместо dmalloc буду использовать valgrind, тем более, как я понимаю, данная утилита успользуется на чекере при тестировании наших работ.

Valgrind является многоцелевым инструментом профилирования кода и отладки памяти. Это позволяет запускать программу в собственной среде Valgrind, что контролирует использование памяти, например, вызовы malloc и free (или new и delete в C++). Если вы используете неинициализированную память, записываете за пределами концов массива, или не освобождаете указатель, Valgrind может это обнаружить. [1]

Gprof также используется для профилирования кода. 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, и которая может быть кандидатом на переписывание, чтобы ускорить выполнение программы. Эта информация также подскажет, какие функции вызывались чаще или реже ожидаемого. Это может помочь отметить ошибки, которые иначе остались бы незамеченными. [2]

2 Дневик отладки

При выполнении предыдущей лабораторной работы чекер сначала выдавал мне ошибку выполнения. Это значит, что в программе есть недочёты, которые скорей всего связаны с памятью и искать их необходимо с помощью утилиты valgrind.

Для того, чтобы найти ошибку в коде, мне не пришлось генерировать большие тесты, оказалось достаточным и теста с парой удалений и парой вставок.

```
==26725== Invalid write of size 1
==26725==    at 0x4C31060: strcpy
(in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==26725==    by 0x4022D8: TPatriciaTrie::Delete(char*) (in /home/me/DA/lab2)
==26725==    by 0x40191F: main (in /home/me/DA/lab2)
==26725==  Address 0x5ab7da2 is 0 bytes after a block of size 2 alloc'd
==26725==    at 0x4C2E80F: operator new[](unsigned long)
(in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==26725==    by 0x402731: TPatriciaTrie::Insert(char*,unsigned long long) (in
/home/me/DA/lab2)
==26725==    by 0x40197F: main (in /home/me/DA/lab2)
==26725==
==26725== Invalid read of size 1
==26725==    at 0x402429: TPatriciaTrie::Delete(char*) (in /home/me/DA/lab2)
==26725==    by 0x40191F: main (in /home/me/DA/lab2)
==26725==  Address 0x100b2c36bc is not stack'd,malloc'd or (recently) free'd
==26725==
==26725==
==26725== Process terminating with default action of signal 11 (SIGSEGV)
==26725==  Access not within mapped region at address 0x100B2C36BC
==26725==    at 0x402429: TPatriciaTrie::Delete(char*) (in /home/me/DA/lab2)
==26725==    by 0x40191F: main (in /home/me/DA/lab2)
==26725==  If you believe this happened as a result of a stack
==26725==  overflow in your program's main thread (unlikely but
==26725==  possible),you can try to increase the size of the
==26725==  main thread stack using the --main-stacksize= flag.
==26725==  The main thread stack size used in this run was 8388608.
==26725== 2 bytes in 1 blocks are definitely lost in loss record 1 of 6
==26725==    at 0x4C2E80F: operator new[](unsigned long)
(in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==26725==    by 0x402731: TPatriciaTrie::Insert(char*,unsigned long long) (in
```

```
/home/me/DA/lab2)
==26725==    by 0x40197F: main (in /home/me/DA/lab2)
==26725==
```

Как оказалось, проблема возникала из-за метода strcpy. Во время удаления узла я не выделял достаточное количество памяти перед копированием. Отсюда возникала ошибка с невалидной записью данных. Однако после исправления этой проблемы никуда не делась потеря одного блока памяти. Как оказалось, это возникло из-за того, что перед выделением необходимого количества памяти я не очищал то, что было до этого. После исправления этой программа работала с памятью корректно.

Далее я столкнулся с другой проблемой – превышением времени работы. Отсюда я сделал вывод, что в каком-то месте моя программа проседает по времени выполнения, и искать проблему я уже буду с помощью gprof'a. В результате профилирования кода я получил приблизительно следующее (приведены лишь первые десять позиций):

% time	cumulative seconds	self seconds	calls	self ms/call	total ms/call	name
20.28	0.16	0.16	998995	0.00	0.00	Insert
13.94	0.27	0.11	6000	0.02	0.02	Index
12.68	0.37	0.10	6000	0.02	0.03	Load
10.14	0.45	0.08	6001	0.01	0.02	DestructRecursive
8.87	0.52	0.07	6000	0.01	0.04	Save
7.61	0.58	0.06	15261651	0.00	0.00	BitGet
6.34	0.63	0.05	7495111	0.00	0.00	ByteLen
6.34	0.68	0.05	999099	0.00	0.00	FirstDifferentBit
3.80	0.71	0.03	5490000	0.00	0.00	TPatriciaTrieNode
2.54	0.73	0.02	5496000	0.00	0.00	Init
2.54	0.75	0.02				main

Меня очень удивил тот факт, что функция main также входит в эту десятку, ведь она по сути только считывает команды, передаёт их в другие функции и просто выводит результат. Тогда я предположил, что это может быть связано с медленным выводом и необходимостью его ускорить. На это меня также натолкнула схожая проблема из первой ЛР. И действительно, после добавления пары строчек, ускоряющих вывод, main оказалась гораздо ниже в топе, а чекер принял моё решение.

3 Выводы

Как было отмечено выше, очень важно, чтобы программа работала оптимально, ведь иначе вряд ли ею будет кто-то пользоваться: например, кому-то не хватит терпения дождаться результатов работы, а кому-то просто-напросто может не хватить мощностей.
В процессе выполнения данной лабораторной работы я познакомился с утилитами для профилирования и отладки кода. На данный момент я могу только предположить, сколько подобная работа может сэкономитть ресурсов в промышленной разработке, ведь, например, до того, как я воспользовался утилитой gprof, время работы программы на некоторых тестах первышала допустимые 160 секунд, после же все тесты выполнялись не более минуты, то есть за счёт добавления двух строчек производительность возросла многократно.
Тоже самое относится и к поиску утечек. В моём небольшом тесте утекло всего 2 байта. Но ведь реальные программы порой работают с невероятными объёмами данных, и даже если утекать будет не больше одного процента, то наберётся солидная цифра.
Конечно, мною были изучены лишь пара утилит, на деле их гораздо больше, на любой вкус и предназначенные для различных нужд. Но я считаю, что и они вместе с отладчиком gdb представляют из себя мощную силу для борьбы с багами и недостатками. Возможно, в промышленной разработке используются более совершенные утилиты, но на данный момент и этих трёх для меня будет достаточно.

Список литературы

[1] *Использование Valgrind для поиска утечек и недопустимого использования памяти.*
URL: `http://cppstudio.com/post/4348/` (дата обращения: 24.09.2020).

[2] *Профилятор gprof.*
URL: `https://www.opennet.ru/docs/RUS/gprof/` (дата обращения: 24.09.2020).