Московский авиационный институт (национальный исследовательский университет)

Факультет компьютерных наук и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №3 по курсу «Дискретный анализ»

Студент: С. Ю. Свиридов
Преподаватель: А. А. Кухтичев
Группа: М8О-206Б-22

Дата:
Оценка:
Подпись:

Москва 2024

Лабораторная работа №3

Задача: А. Исследование качества программ 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.

Результатом лабораторной работы является отчёт, состоящий из:

Дневника выполнения работы, в котором отражено что и когда делалось, какие средства использовались и какие результаты были достигнуты на каждом шаге выполнения лабораторной работы. Выводов о найденных недочётах. Сравнение работы исправленной программы с предыдущей версией. Общих выводов о выполнении лабораторной работы, полученном опыте. Минимальный набор используемых средств должен содержать утилиту `gprof` и библиотеку `dmalloc`, однако их можно заменять на любые другие аналогичные или более развитые утилиты (например, `Valgrind` или `Shark`) или добавлять к ним новые (например, `gcov`)

1 Описание работы утилиты gprof

Первой утилитой на очереди оказалась утилита gprof. Она позволяет пользователю получить подробный отчет о выполнении программы, в котором будет сказано, о том, сколько времени выполнялась каждая функция, сколько раз она вызывалась и т.п. Для корректной работы этой утилиты, требуется разработать сложную, долговыполняющуюся программу или нагрузить не очень сложную программу большим количеством вызовов функций. Для такого случая, был написан генератор случайных строк, которые затем добавлялись в дерево

```
1  std::string generateRandomString(int length) {
2
3      const std::string charset = "abcdefghijklmnopqrstuvwxyz";
4
5      std::mt19937 rng(std::random_device{}());
6      std::uniform_int_distribution<> dist(0, charset.length() - 1);
7
8      std::string result;
9      result.reserve(length);
10     for (int i = 0; i < length; ++i) {
11         result += charset[dist(rng)];
12     }
13
14     return result;
15 }
```

Вот, что выдала утилита gprof:

```
stepan@stepan-ASUS:~/Рабочий стол/учеба/prog4sem/discran/laba2/src/bench$ g++
-O2 -lm -fno-stack-limit -std=c++20 -x c++ tests.cpp -o executable -pg
stepan@stepan-ASUS:~/Рабочий стол/учеба/prog4sem/discran/laba2/src/bench$ ./executable
stepan@stepan-ASUS:~/Рабочий стол/учеба/prog4sem/discran/laba2/src/bench$ gprof
executable >output.txt
stepan@stepan-ASUS:~/Рабочий стол/учеба/prog4sem/discran/laba2/src/bench$ cat
output.txt
Flat profile:

Each sample counts as 0.01 seconds.
%   cumulative   self              self     total
time   seconds   seconds    calls  us/call  us/call  name
40.74      0.11     0.11    50001     2.20     2.20  std::mersenne_twister_engine...
33.33      0.20     0.09    50001     1.80     4.60  generateRandomString...
7.41       0.22     0.02 12800256     0.00     0.01  Patricia::Remove...
7.41       0.24     0.02    50000     0.40     0.40  Patricia::search...
3.70       0.25     0.01 12800256     0.00     0.01  std::mersenne_twister_engine...
```

```
3.70      0.26     0.01    50001     0.20     0.80  Patricia::Add...
3.70      0.27     0.01    50000     0.20     0.20  Patricia::Insert...
0.00      0.27     0.00   200004     0.00     0.00  frame_dummy
...
```

Было сгенерировано 50000 строк, которые затем были добавлены в дерево. Как видно из отчета, строки генерируются дольше, чем добавляются в дерево. А сама функция вставки нового узла в дерево занимает лишь около 11 процентов времени работы всей программы. Так же видно, что около 11 процентов времени занимает выполнение функции `Search`. Это происходит из-за того, что в функции добавления узла в дерево, каждый раз происходит проверка на то, существует ли в дереве добавляемый узел. Так же столько же времени занимает функция удаления элемента из дерева. Я заметил, что она вызывается столько же раз, сколько и генератор псевдослучайных чисел Mersenne Twister (MT). Скорее всего это связно с функцией генерациии строк, однако компиляция со всеми флагами происходит без ошибок и предупреждений. Более того при проверке работы программы на простых тестах, без генераци случайных строк, функция `Remove` не вызывается без вмешательства пользователя. В целом, даже на таком тесте видно насколько быстро происходит добавление новых вершин в дерево.

```
stepan@stepan-ASUS:~/Рабочий стол/учеба/prog4sem/discran/laba2/src/bench$ g++
-O2 -lm -fno-stack-limit -std=c++20 -x c++ tests.cpp -o executable -pg
stepan@stepan-ASUS:~/Рабочий стол/учеба/prog4sem/discran/laba2/src/bench$ ./executable
stepan@stepan-ASUS:~/Рабочий стол/учеба/prog4sem/discran/laba2/src/bench$ gprof
executable >output.txt
stepan@stepan-ASUS:~/Рабочий стол/учеба/prog4sem/discran/laba2/src/bench$ cat
output.txt
Flat profile:

Each sample counts as 0.01 seconds.
%   cumulative   self              self     total
time   seconds   seconds    calls  us/call  us/call  name
33.33      0.08     0.08    50001     1.60     4.19  generateRandomString...
33.33      0.16     0.08    50001     1.60     1.60  std::mersenne_twister_engine...
12.50      0.19     0.03 12800256     0.00     0.01  std::mersenne_twister_engine...
8.33       0.21     0.02 12825257     0.00     0.01  Patricia::Remove...
4.17       0.22     0.01    50001     0.20     0.60  Patricia::Add...
4.17       0.23     0.01    49999     0.20     0.20  Patricia::Insert...
4.17       0.24     0.01    49999     0.20     0.20  Patricia::search...
0.00       0.24     0.00   200004     0.00     0.00  frame_dummy
0.00       0.24     0.00    25001     0.00     0.00  Patricia::Search_Parent...
...
```

В следующем тесте была проверена работа функции `Remove`. В дерево так же добавлялось 50000 строк и каждая вторая из них сразу же удалялась из дерева. В результате из функций дерева самая долгая оказалась функция удаления, при этом количество вывовов не изменилось, то есть эта функция вызывалась столько же раз, сколько и генератор псевдослучайных чисел Mersenne Twister (MT) плюс еще 25001 вызов для удаления половины элементов. В целом, из двух вышепроведенных тестов можно сделать вывод о том, что функции в дереве работают достаточно быстро даже в сравнении с генератором случайных строк.

2 Описание работы утилиты valgrind

Утилита Valgrind предназачена для поиска утечек памяти в программе. Вот, что выдала утилита Valgrind при первом запуске программы:

```
stepan@stepan-ASUS:~/Рабочий стол/учеба/prog4sem/discran/laba2/src$ valgrind
--leak-check=full ./exec
==7719== Memcheck,a memory error detector
==7719== Copyright (C) 2002-2022,and GNU GPL'd,by Julian Seward et al.
==7719== Using Valgrind-3.22.0 and LibVEX; rerun with -h for copyright info
==7719== Command: ./exec
==7719==
+ a 0
OK
+ b 1
OK
+ c 3
OK
-b
OK
! Save ./filetest.txt
OK
+ q 13
OK
! Load ./filetest.txt
OK
q
NoSuchWord
==7719==
==7719== HEAP SUMMARY:
==7719==     in use at exit: 176 bytes in 4 blocks
==7719==   total heap usage: 20 allocs,16 frees,92,828 bytes allocated
==7719==
==7719== 24 bytes in 1 blocks are definitely lost in loss record 1 of 4
==7719==    at 0x4E06F73: operator new[](unsigned long) (vg_replace_malloc.c:725)
==7719==    by 0x10AF87: Patricia::Search_Parent(std::__cxx11::basic_string<char,std:
const (main.cpp:230)
==7719==    by 0x10B16A: Patricia::Remove(std::__cxx11::
basic_string<char,std::char_traits<char>,std::
allocator<char>>const&) (main.cpp:272)
==7719==    by 0x10BD92: main (main.cpp:473)
```

```
==7719==
==7719== 24 bytes in 1 blocks are definitely lost in loss record 2 of 4
==7719==    at 0x4E06F73: operator new[](unsigned long) (vg_replace_malloc.c:725)
==7719==    by 0x10AF87: Patricia::Search_Parent(std::__cxx11::basic_string<char,std:
const (main.cpp:230)
==7719==    by 0x10B342: Patricia::Remove(std::__cxx11::
basic_string<char,std::char_traits<char>,
std::allocator<char>>const&) (main.cpp:317)
==7719==    by 0x10BD92: main (main.cpp:473)
==7719==
==7719== 128 (64 direct,64 indirect) bytes in 1 blocks
are definitely lost in loss record 4 of 4
==7719==    at 0x4E05833: operator new(unsigned long) (vg_replace_malloc.c:483)
==7719==    by 0x10AC3A: Patricia::Add(std::__cxx11::basic_string<char,std::char_trait
std::allocator<char>>const&,unsigned long long) (main.cpp:188)
==7719==    by 0x10B84F: Patricia::LoadFromFile(std::__cxx11::basic_string<char,std::c
(main.cpp:403)
==7719==    by 0x10BEB3: main (main.cpp:491)
==7719==
==7719== LEAK SUMMARY:
==7719==    definitely lost: 112 bytes in 3 blocks
==7719==    indirectly lost: 64 bytes in 1 blocks
==7719==      possibly lost: 0 bytes in 0 blocks
==7719==    still reachable: 0 bytes in 0 blocks
==7719==         suppressed: 0 bytes in 0 blocks
==7719==
==7719== For lists of detected and suppressed errors,rerun with: -s
==7719== ERROR SUMMARY: 3 errors from 3 contexts (suppressed: 0 from 0)
```

Было обнаружено 3 утечки памяти в функциях `LoadFromFile, Add и Remove`. Были введены некоторые правки в код, которые устранили утечки памяти. Вот, что теперь выдает утилита `Valgrind` после небольшого теста программы:

```
stepan@stepan-ASUS:~/Рабочий стол/учеба/prog4sem/discran/laba2/src$ g++ -g
-o exec main.cpp
stepan@stepan-ASUS:~/Рабочий стол/учеба/prog4sem/discran/laba2/src$ valgrind
--leak-check=full ./exec
==9026== Memcheck,a memory error detector
==9026== Copyright (C) 2002-2022,and GNU GPL'd,by Julian Seward et al.
==9026== Using Valgrind-3.22.0 and LibVEX; rerun with -h for copyright info
==9026== Command: ./exec
```

```
==9026==
0x547fc80
+ a 0
OK
+ b 1
OK
+ c 3
OK
-b
OK
! Save ./file.txt
OK
+ q 123
OK
q
OK: 123
! Load ./file.txt
OK
q
NoSuchWord
==9026==
==9026== HEAP SUMMARY:
==9026==     in use at exit: 128 bytes in 2 blocks
==9026==   total heap usage: 20 allocs,18 frees,92,828 bytes allocated
==9026==
==9026== 128 (64 direct,64 indirect) bytes in 1 blocks are definitely lost
in loss record 2 of 2
==9026==    at 0x4E05833: operator new(unsigned long) (vg_replace_malloc.c:483)
==9026==    by 0x10AC9A: Patricia::Add(std::__cxx11::basic_string<char,std::char_trait
long long) (main.cpp:188)
==9026==    by 0x10B913: Patricia::LoadFromFile(std::__cxx11::basic_string<char,std::c
(main.cpp:408)
==9026==    by 0x10BFB0: main (main.cpp:498)
==9026==
==9026== LEAK SUMMARY:
==9026==    definitely lost: 64 bytes in 1 blocks
==9026==    indirectly lost: 64 bytes in 1 blocks
==9026==      possibly lost: 0 bytes in 0 blocks
==9026==    still reachable: 0 bytes in 0 blocks
==9026==         suppressed: 0 bytes in 0 blocks
==9026==
```

```
==9026== For lists of detected and suppressed errors,rerun with: -s
==9026== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
```

Как видно, осталась лишь одна утечка, которая связана с созданием корня дерева. Мои попытки исправить ситуацию оказались тщетны, но я намереваюсь узнать у преподавателя о том, почему же эта утечка остается.

3 Выводы

Выполнив третью лабораторную работу по курсу «Дискретный анализ», я поработал с некоторыми утилитами, которые анализируют работу моей программы. Теперь мне намного легче узнать о том, сколько времени выполняется моя программа и какие функции в ней самые долгие и том, есть ли в моей программе утечки памяти. На мой взгляд, наиболее полезной оказалась утилита `Valgrind`, потому что она помогает выявляеть и устранять проблемы с памятью в коде, которые являются одной из самых серьезных проблем для разработчика на `c++`.

Список литературы

[1] *Gprof*
URL: `https://www.sourceware.org/binutils/docs/gprof.html`

[2] *Valgrind*
URL: `https://poby.medium.com/memory-profiling-using-valgrind-b30514f35117`