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: К. А. Калугин
Преподаватель: А. А. Кухтичев
Группа: М8О-206Б

Дата:
Оценка:
Подпись:

Москва, 2021

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.

Использованные утилиты: `gprof`, `valgrind`.

1 Использование утилиты valgrind

Valgrind — инструментальное программное обеспечение, предназначенное для отладки использования памяти, обнаружения утечек памяти, а также профилирования [1]. Я использовал valgrind, чтобы найти утечки памяти в одной из промежуточных версий программы.

netter@netter-VirtualBox: /Загрузки$ valgrind –tool=memcheck -s –leak-check=full ./a.out
==2641== Memcheck, a memory error detector
==2641== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==2641== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==2641== Command: ./a.out
==2641==
==2641==
==2641== HEAP SUMMARY:
==2641== in use at exit: 304 bytes in 1 blocks
==2641== total heap usage: 4 allocs, 3 frees, 75,056 bytes allocated
==2641==
==2641== 304 bytes in 1 blocks are definitely lost in loss record 1 of 1
==2641== at 0x483BE63: operator new(unsigned long) (in /usr/lib/x8664-linux-gnu/valgrind/vgpreloa
amd64-linux.so)
==2641== by 0x109DA9: TPatricia::TPatricia() (in /home/netter/Загрузки/a.out)
==2641== by 0x1094E1: main (in /home/netter/Загрузки/a.out)
==2641==
==2641== LEAK SUMMARY:
==2641== definitely lost: 304 bytes in 1 blocks
==2641== indirectly lost: 0 bytes in 0 blocks
==2641== possibly lost: 0 bytes in 0 blocks
==2641== still reachable: 0 bytes in 0 blocks
==2641== suppressed: 0 bytes in 0 blocks
==2641==
==2641== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)

Как видно, утечка происходит из-за "незакрытого"оператора new. В этой версии программы я не удалял корень дерева, что и вызывало ошибку. После добавления дестркутора, ошибка исчезла.

netter@netter-VirtualBox: /Загрузки$ valgrind –tool=memcheck -s –leak-check=full ./a.out
==3724== Memcheck, a memory error detector

==3724== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==3724== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==3724== Command: ./a.out
==3724==
0==3724==
==3724== HEAP SUMMARY:
==3724== in use at exit: 0 bytes in 0 blocks
==3724== total heap usage: 4 allocs, 4 frees, 75,056 bytes allocated
==3724==
==3724== All heap blocks were freed – no leaks are possible
==3724==
==3724== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
Как видно, valgrind показывает 0 ошибок.

2 Использование утилиты gprof

Согласно [2], gprof - инструмент, позволяющий анализировать эффективность программ для Unix. Одна из версий программы не проходила чекер из-за слишком медленной работы.

```
netter@netter-VirtualBox:~/Загрузки\$ g++ -pg message.cpp  -o test-gprof
netter@netter-VirtualBox:~/Загрузки\$ cat 14.t | ./test-gprof >/dev/null
netter@netter-VirtualBox:~/Загрузки\$ gprof test-gprof gmon.out
Flat profile:

Each sample counts as 0.01 seconds.

%   cumulative   self              self     total

time   seconds   seconds    calls  ms/call  ms/call  name

50.28      1.22     1.22  1048576     0.00     0.00

TPatricia::Find(std::__cxx11::basic_string<char,std::char_traits<char>,

std::allocator<char>>const&)
```

```
13.19      1.54     0.32  1048576     0.00     0.00
ltu(std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>&)
11.95      1.83     0.29        1   290.41   290.41
TPatricia::RecParcer(TPatricia::TNode*,_IO_FILE*)
8.45       2.04     0.21 41507260     0.00     0.00
bitcheck(int,std::__cxx11::basic_string<char,std::char_traits<char>,
std::allocator<char>>const&)
7.83       2.23     0.19        1   190.27   190.27
TPatricia::Clean(TPatricia::TNode*)
3.71       2.32     0.09  1040489     0.00     0.00
TPatricia::ParFind(std::__cxx11::basic_string<char,std::char_traits<char>,
std::allocator<char>>const&,TPatricia::TNode**,int const&,bool&)
2.27       2.37     0.06 41499173     0.00     0.00
code(char)
1.24       2.40     0.03  1048576     0.00     0.00
TPatricia::Add(TKV&)
1.24       2.43     0.03
main
0.00       2.43     0.00  1048583     0.00     0.00
bool std::operator==<char,std::char_traits<char>,std::allocator<char>
>(std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>
```

```
const&,char const*)
0.00      2.43     0.00  1048576     0.00     0.00
__gnu_cxx::__enable_if<std::__is_char<char>::__value,bool>::__type
std::operator==<char>(std::__cxx11::basic_string<char,std::char_traits<char>,
std::allocator<char>>const&,std::__cxx11::basic_string<char,std::char_traits<char>,
std::allocator<char>>const&)
0.00      2.43     0.00  1040491     0.00     0.00
TKV::TKV()
0.00      2.43     0.00  1040491     0.00     0.00
TKV::~TKV()
0.00      2.43     0.00  1040490     0.00     0.00
TPatricia::TNode::TNode()
0.00      2.43     0.00  1040490     0.00     0.00
TPatricia::TNode::~TNode()
0.00      2.43     0.00  1040489     0.00     0.00
TKV::operator=(TKV const&)
0.00      2.43     0.00    12126     0.00     0.00
std::char_traits<char>::compare(char const*,char const*,unsigned long)
0.00      2.43     0.00        1     0.00     0.00
_GLOBAL__sub_I__Z3ltuRNSt7__cxx1112basic_stringIcSt11char_traitsIcESaIcEEE
0.00      2.43     0.00        1     0.00     0.00
```

```
__static_initialization_and_destruction_0(int,int)

0.00      2.43     0.00        1     0.00   290.41

TPatricia::Save(std::__cxx11::basic_string<char,std::char_traits<char>,

std::allocator<char>>const&)

0.00      2.43     0.00        1     0.00     0.00

TPatricia::TPatricia()

0.00      2.43     0.00        1     0.00     0.00

TPatricia::~TPatricia()
```

Как видно, большое количество времени тратится на Find. После оптимизации программы были получены следующие данные.

```
netter@netter-VirtualBox:~/Загрузки$ gprof test-gprof gmon.out
Flat profile:

Each sample counts as 0.01 seconds.

%   cumulative   self              self     total

time   seconds   seconds    calls  ms/call  ms/call  name

47.03      1.55     1.55  1048576     0.00     0.00

TPatricia::Find(TString&)

15.17      2.05     0.50 389725873     0.00     0.00

TString::operator[](int)

9.41      2.36     0.31

main
```

```
8.19      2.63     0.27        1   270.32   274.84
TPatricia::RecParcer(TNode*,_IO_FILE*,unsigned long long&,unsigned long long&)
5.46      2.81     0.18 41507210     0.00     0.00
TString::Size()
5.01      2.98     0.17 41507210     0.00     0.00
bitcheck(int,TString&)
4.85      3.14     0.16        1   160.19   160.19
TPatricia::Clean(TNode*)
1.82      3.20     0.06  1040489     0.00     0.00
TPatricia::ParFind(TString&,TNode**,int const&,bool&)
1.52      3.25     0.05 41499123     0.00     0.00
code(char)
0.91      3.28     0.03  1048576     0.00     0.00
TPatricia::Add(TKV&)
0.61      3.30     0.02  1048576     0.00     0.00
TString::operator==(TString&)
0.15      3.30     0.01
frame_dummy
0.00      3.30     0.00  1048577     0.00     0.00
TString::Clear()
```

```
0.00      3.30     0.00  1040492     0.00     0.00

TString::TString()

0.00      3.30     0.00  1040491     0.00     0.00

TKV::TKV()

0.00      3.30     0.00  1040490     0.00     0.00

TNode::TNode()

0.00      3.30     0.00        1     0.00     0.00

_GLOBAL__sub_I__Z4codec

0.00      3.30     0.00        1     0.00     0.00

__static_initialization_and_destruction_0(int,int)

0.00      3.30     0.00        1     0.00     0.00

TString::Pushback(char)

0.00      3.30     0.00        1     0.00   274.84

TPatricia::Save(TString&)

0.00      3.30     0.00        1     0.00     0.00

TPatricia::TPatricia()

0.00      3.30     0.00        1     0.00     0.00

TPatricia::~TPatricia()
```

Как видно, оптимизация дала выигрыш в скорости.

3 Выводы

Выполнив третью лабораторную работу, я научился: 1) Находить и устранять утечки памяти и другие неприятные ошибки при помощи утилиты valgrind. 2) Производить оценку эффективности программы с помощью gprof.

Список литературы

[1] URL: https://ru.wikipedia.org/wiki/Valgrind (дата обращения: 18.01.2021)

[2] URL: https://en.wikipedia.org/wiki/Gprof (дата обращения: 19.01.2021).