Московский авиационный институт
(национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №1 по курсу «Дискретный анализ»

Студент: М. А. Волков
Преподаватель: А. А. Кухтичев
Группа: М8О-207Б
Дата: 1 декабря 2020 г.
Оценка:
Подпись:

Москва, 2020

Лабораторная работа №3

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

Используемые утилиты: valgrind, gprof.

1 Valgrind

Согласно [1], Valgrind – это программа для поиска ошибок обращения с памятью, поиска утечек памяти и профлирования. Дл япоиска ошибки я использовал средство ключ компиляции $-g$.

Благодаря valrgind я обнаружил, что моя четвертая рабочая версия программы потребляет много памяти.

```
==10261== Memcheck,a memory error detector
==10261== Copyright (C) 2002-2017,and GNU GPL'd,by Julian Seward et al.
==10261== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==10261== Command: ./DA_lab2
==10261==

... ВВОД ...

a==10261== Mismatched free() / delete / delete []
==10261==    at 0x483CFBF: operator delete(void*) (in /usr/lib/x86_64-linux-gnu/valgr
==10261==    by 0x10B6DD: ~TNode (main.cpp:141)
==10261==    by 0x10B6DD: TPatricia<char>::Delete(char*) (main.cpp:314)
==10261==    by 0x10A80F: main (main.cpp:453)
==10261==  Address 0x4dc4ff0 is 0 bytes inside a block of size 3 alloc'd
==10261==    at 0x483C583: operator new[](unsigned long) (in /usr/lib/x86_64-linux-gnu
==10261==    by 0x10B36A: Initialize (main.cpp:117)
==10261==    by 0x10B36A: TPatricia<char>::Insert(char*,unsigned long long)
(main.cpp:248)
==10261==    by 0x10A6E5: main (main.cpp:446)
==10261==

... ВЫВОД ...

==10261==
==10261== HEAP SUMMARY:
==10261==     in use at exit: 122,923 bytes in 8 blocks
==10261==   total heap usage: 14 allocs,6 frees,195,728 bytes allocated
==10261==
==10261== 43 (40 direct,3 indirect) bytes in 1 blocks are definitely lost in
loss record 2 of 8
==10261==    at 0x483BE63: operator new(unsigned long) (in /usr/lib/x86_64-linux-gnu/v
==10261==    by 0x10B2F1: TPatricia<char>::Insert(char*,unsigned long long)
(main.cpp:241)
```

```
==10261==    by 0x10A6E5: main (main.cpp:446)
==10261==
==10261== LEAK SUMMARY:
==10261==    definitely lost: 40 bytes in 1 blocks
==10261==    indirectly lost: 3 bytes in 1 blocks
==10261==      possibly lost: 0 bytes in 0 blocks
==10261==    still reachable: 122,880 bytes in 6 blocks
==10261==         suppressed: 0 bytes in 0 blocks
==10261== Reachable blocks (those to which a pointer was found) are not shown.
==10261== To see them,rerun with: --leak-check=full --show-leak-kinds=all
==10261==
==10261== For lists of detected and suppressed errors,rerun with: -s
==10261== ERROR SUMMARY: 2 errors from 2 contexts (suppressed: 0 from 0)
```

Благодаря особому ключу компиляции $-g$, valgrind показал мне конкретную строчку, где может происходить утечка памяти. Это оказался диструктер.

Он был неправильно реализован. Сначала это было обычное применение default диструктора, то есть был написан диструктер так:

```
1 ~TPatricia() = default;
```

Но потом был сделан метод $CleanUp()$, который проходится рекурсивно по всему дереву и удаляет каждый элемент.

Данная запись помогла мне локализовать ошибку:

```
a==10261== Mismatched free() / delete / delete []
==10261==    at 0x483CFBF: operator delete(void*) (in /usr/lib/x86_64-linux-gnu/valgr
==10261==    by 0x10B6DD: ~TNode (main.cpp:141)
==10261==    by 0x10B6DD: TPatricia<char>::Delete(char*) (main.cpp:314)
==10261==    by 0x10A80F: main (main.cpp:453)
==10261==  Address 0x4dc4ff0 is 0 bytes inside a block of size 3 alloc'd
==10261==    at 0x483C583: operator new[](unsigned long) (in /usr/lib/x86_64-linux-gnu
==10261==    by 0x10B36A: Initialize (main.cpp:117)
==10261==    by 0x10B36A: TPatricia<char>::Insert(char*,unsigned long long)
(main.cpp:248)
==10261==    by 0x10A6E5: main (main.cpp:446)
```

2 Gprof

Как сказано в [3], Gprof – едство профилирования в Unix системах. Используется для измерения времени работы отдельных функций программы и общего временироботы программы.

Профилировщик показывает, сколько процентов от общего времени работы программы работает и сколько раз вызывается каждая функция (и ещё много данных, которые не так важны для нашей задачи).

Диагностика проводилась на тесте из 25000 строк с запросами на добавление, удаление, поиск, загрузку и сохранение.

```
Flat profile:

Each sample counts as 0.01 seconds.
%   cumulative   self              self     total
time   seconds   seconds    calls  us/call  us/call  name
50.16      0.06     0.06                             TPatricia<char>::Load(std::basic_
16.72      0.08     0.02     3239     6.19     6.19  TPatricia<char>::Enumerate(TNode<
16.72      0.10     0.02                             TPatricia<char>::Save(std::basic_
16.72      0.12     0.02                             TPatricia<char>::CleanUp(TNode<ch
0.00      0.12     0.00        1     0.00     0.00  _GLOBAL__sub_I__ZN9TPatriciaIcE4Sa
```

Видно, что программа тратит большую часть времени не на операции с деревом, а на загрузку дерева из памяти. Это можно легко объянить: в данном тесте очень много операций загрузки.

3 Дневник отладки

1. 24.11.20 Воспользовался утилитой valgrind для поиска причины возникновения ошибки превышения памяти.
2. 25.11.20 Ошибка была найдена и устранена.
3. 30.11.20 Изучил принцип работы с утилитой Gprof.
4. 01.12.20 Произвёл профилирование программы с помощью Gprof.

4 Выводы

Выполнив третью лабораторную работу по курсу «Дискретный анализ», я узнал об утилите Gprof и на практике научился ей пользоваться. Узнал о великолепном ключе компиляции $-g$, который позволяет при помощи Valgrind показать конкретную строчку, в которой возможна утечка памяти. Без этого ключа я очень долго и безуспешно искал ошибку.

Список литературы

[1] *Valrgind — Википедия.*
URL: `https://en.wikipedia.org/wiki/Valgrind` (дата обращения: 06.11.2020).

[2] *Ловим утечки памяти в C/C++ / Хабр – Habr.com*
URL: `https://habr.com/ru/post/480368/` (дата обращения: 20.11.2020).

[3] *Gprof – Википедия*
URL: `https://en.wikipedia.org/wiki/Gprof` (дата обращения: 20.11.2020).

[4] *рофилирование уже запущенных программ / Хабр – Habr.com*
URL: `https://habr.com/ru/post/167837/` (дата обращения: 20.11.2020).

[5] Список использованных источников оформлять нужно по ГОСТ Р 7.05-2008