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. С. Федоров
Преподаватель: А. А. Кухтичев
Группа: М8О-207Б
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.

Испольщуемые средства: Valgrind, Gprof.

1 Valgrind

Согласно [1] Valgrind — инструментальное программное обеспечение, предназначенное для отладки использования памяти, обнаружения утечек памяти, а также профилирования.

Первая реализация B-дерева обладала некорректным деструктором для дерева, что приводило к утечке памяти. Слияние и разделение узлов не удоляло уже бесполезную память. Происходило обращение к неинициализированной области памяти. Это происходило из-за чтения клчей разной длины.Дерево не было удалено после окончания работы программы.

```
==4648== Invalid read of size 8
==4648==    at 0x10CED2: TVector<Node*>::operator[](long long) (in /home/protaxy/DA/da
==4648==    by 0x1094ED: goAround(Node*&) (in /home/protaxy/DA/da_lab2/a.out)
==4648==    by 0x109555: goAround(Node*&) (in /home/protaxy/DA/da_lab2/a.out)
==4648==    by 0x109555: goAround(Node*&) (in /home/protaxy/DA/da_lab2/a.out)
==4648==    by 0x109555: goAround(Node*&) (in /home/protaxy/DA/da_lab2/a.out)
==4648==    by 0x109555: goAround(Node*&) (in /home/protaxy/DA/da_lab2/a.out)
==4648==    by 0x109555: goAround(Node*&) (in /home/protaxy/DA/da_lab2/a.out)
==4648==    by 0x109555: goAround(Node*&) (in /home/protaxy/DA/da_lab2/a.out)
==4648==    by 0x109555: goAround(Node*&) (in /home/protaxy/DA/da_lab2/a.out)
==4648==    by 0x109555: goAround(Node*&) (in /home/protaxy/DA/da_lab2/a.out)
==4648==    by 0x109555: goAround(Node*&) (in /home/protaxy/DA/da_lab2/a.out)
==4648==    by 0x109555: goAround(Node*&) (in /home/protaxy/DA/da_lab2/a.out)
==4648==  Address 0x5504b48 is 40 bytes inside a block of size 56 free'd
==4648==    at 0x483D1CF: operator delete(void*,unsigned long) (in /usr/lib/x86_64-li
==4648==    by 0x109E9A: Split(Node*&) (in /home/protaxy/DA/da_lab2/a.out)
==4648==    by 0x10A3B6: AddToTree(Node*&,Item) (in /home/protaxy/DA/da_lab2/a.out)
==4648==    by 0x10C9A3: main (in /home/protaxy/DA/da_lab2/a.out)
==4648==  Block was alloc'd at
==4648==    at 0x483BE63: operator new(unsigned long) (in /usr/lib/x86_64-linux-gnu/va
==4648==    by 0x109A7E: Split(Node*&) (in /home/protaxy/DA/da_lab2/a.out)
==4648==    by 0x10A3B6: AddToTree(Node*&,Item) (in /home/protaxy/DA/da_lab2/a.out)
==4648==    by 0x10C9A3: main (in /home/protaxy/DA/da_lab2/a.out)
==4648==

...

==4648==
==4648== Invalid free() / delete / delete[] / realloc()
==4648==    at 0x483D74F: operator delete[](void*) (in /usr/lib/x86_64-linux-gnu/valgr
```

```
==4648==    by 0x10CE74: TVector<Item>::~TVector() (in /home/protaxy/DA/da_lab2/a.out)
==4648==    by 0x10CE02: Node::~Node() (in /home/protaxy/DA/da_lab2/a.out)
==4648==    by 0x10CDD3: Node::~Node() (in /home/protaxy/DA/da_lab2/a.out)
==4648==    by 0x10CDD3: Node::~Node() (in /home/protaxy/DA/da_lab2/a.out)
==4648==    by 0x10CDD3: Node::~Node() (in /home/protaxy/DA/da_lab2/a.out)
==4648==    by 0x10CDD3: Node::~Node() (in /home/protaxy/DA/da_lab2/a.out)
==4648==    by 0x10CDD3: Node::~Node() (in /home/protaxy/DA/da_lab2/a.out)
==4648==    by 0x10CDD3: Node::~Node() (in /home/protaxy/DA/da_lab2/a.out)
==4648==    by 0x10CDD3: Node::~Node() (in /home/protaxy/DA/da_lab2/a.out)
==4648==    by 0x10CDD3: Node::~Node() (in /home/protaxy/DA/da_lab2/a.out)
==4648==    by 0x10CDD3: Node::~Node() (in /home/protaxy/DA/da_lab2/a.out)
==4648==  Address 0x5509110 is 0 bytes inside a block of size 1,056 free'd
==4648==    at 0x483D74F: operator delete[](void*) (in /usr/lib/x86_64-linux-gnu/valgr
==4648==    by 0x10CE74: TVector<Item>::~TVector() (in /home/protaxy/DA/da_lab2/a.out)
==4648==    by 0x10CE02: Node::~Node() (in /home/protaxy/DA/da_lab2/a.out)
==4648==    by 0x109E8D: Split(Node*&) (in /home/protaxy/DA/da_lab2/a.out)
==4648==    by 0x10A3B6: AddToTree(Node*&,Item) (in /home/protaxy/DA/da_lab2/a.out)
==4648==    by 0x10C9A3: main (in /home/protaxy/DA/da_lab2/a.out)
==4648==  Block was alloc'd at
==4648==    at 0x483C583: operator new[](unsigned long) (in /usr/lib/x86_64-linux-gnu/
==4648==    by 0x10D087: TVector<Item>::PushBack(Item) (in /home/protaxy/DA/da_lab2/a
==4648==    by 0x10D3FB: TVector<Item>::Insert(unsigned long long const&,Item
const&) (in /home/protaxy/DA/da_lab2/a.out)
==4648==    by 0x10D6A5: TVector<Item>::OrdinaryInsert(Item const&) (in /home/protaxy/
==4648==    by 0x10A395: AddToTree(Node*&,Item) (in /home/protaxy/DA/da_lab2/a.out)
==4648==    by 0x10C9A3: main (in /home/protaxy/DA/da_lab2/a.out)
==4648==

...

==4648==
==4648== HEAP SUMMARY:
==4648==     in use at exit: 336 bytes in 3 blocks
==4648==   total heap usage: 36,091 allocs,36,091 frees,7,642,616 bytes allocated
==4648==
==4648== 336 (56 direct,280 indirect) bytes in 1 blocks are definitely lost
in loss record 3 of 3
==4648==    at 0x483BE63: operator new(unsigned long) (in /usr/lib/x86_64-linux-gnu/va
==4648==    by 0x109A7E: Split(Node*&) (in /home/protaxy/DA/da_lab2/a.out)
==4648==    by 0x10A3B6: AddToTree(Node*&,Item) (in /home/protaxy/DA/da_lab2/a.out)
==4648==    by 0x10C9A3: main (in /home/protaxy/DA/da_lab2/a.out)
```

```
==4648==
==4648== LEAK SUMMARY:
==4648==    definitely lost: 56 bytes in 1 blocks
==4648==    indirectly lost: 280 bytes in 2 blocks
==4648==      possibly lost: 0 bytes in 0 blocks
==4648==    still reachable: 0 bytes in 0 blocks
==4648==         suppressed: 0 bytes in 0 blocks
==4648==
==4648== For lists of detected and suppressed errors,rerun with: -s
==4648== ERROR SUMMARY: 25 errors from 15 contexts (suppressed: 0 from 0)
```

Реализовал Деструктор для узла, который рекурсивно обходит и удаляет всех своих детей. Исправление и доработака операций разбиения и слияния и зануление массива-ключа перед его вводом исправило проблемы.

```
==4978==
==4978== HEAP SUMMARY:
==4978==     in use at exit: 122,880 bytes in 6 blocks
==4978==   total heap usage: 7,466 allocs,7,460 frees,5,484,992 bytes allocated
==4978==
==4978== LEAK SUMMARY:
==4978==    definitely lost: 0 bytes in 0 blocks
==4978==    indirectly lost: 0 bytes in 0 blocks
==4978==      possibly lost: 0 bytes in 0 blocks
==4978==    still reachable: 122,880 bytes in 6 blocks
==4978==         suppressed: 0 bytes in 0 blocks
==4978== Reachable blocks (those to which a pointer was found) are not shown.
==4978== To see them,rerun with: --leak-check=full --show-leak-kinds=all
==4978==
==4978== For lists of detected and suppressed errors,rerun with: -s
==4978== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Лог показал, что больше ошибок в ходе работы программы выявлено не было.

2 Gprof

Согласно [3] Gprof - средство профилирования и измерения работы отдельных фукций программы для Unix систем. Лог Gprof показывает статискику от наиболее вызываемых и долгих фукций до самых быстрых и редковызываемых.

```
protaxy@protaxY:~/DA/da_lab2$ gprof a.out gmon.out -p
Flat profile:
Each sample counts as 0.01 seconds.
%   cumulative   self              self     total
time   seconds   seconds    calls  us/call  us/call  name
29.25      0.19     0.19   399734     0.48     0.48  Item::operator=(Item)
26.17      0.36     0.17                             main
21.55      0.50     0.14   262250     0.53     0.53  Item::Item()
10.78      0.57     0.07  1507464     0.05     0.05  operator<(Item const&,Item
const&)
3.08       0.59     0.02    84995     0.24     2.44  TVector<Item>::PushBack(Item)
1.54       0.60     0.01  1835714     0.01     0.01  TVector<Item>::operator[](long
long)
1.54       0.61     0.01   376508     0.03     0.03  operator==(Item const&,Item
const&)
1.54       0.62     0.01    36845     0.27     6.11  TVector<Item>::Insert(unsigned
long long const&,Item const&)
1.54       0.63     0.01    33345     0.30     1.14  DeleteFromTree(Node*&,Item)
1.54       0.64     0.01    33245     0.30     1.13  SearchInTree(Node*,Item)
1.54       0.65     0.01     3126     3.20    47.74  Split(Node*&)
0.00       0.65     0.00   920464     0.00     0.00  TVector<Item>::Size()
0.00       0.65     0.00   754823     0.00     0.00  TVector<Node*>::operator[](long
long)
0.00       0.65     0.00   162858     0.00     0.00  TVector<Node*>::Size()
0.00       0.65     0.00    91878     0.00     0.00  TVector<Node*>::PushBack(Node*)
0.00       0.65     0.00    33410     0.00    11.58  AddToTree(Node*&,Item)
0.00       0.65     0.00    33410     0.00     6.27  TVector<Item>::OrdinaryInsert(Ite
const&)
0.00       0.65     0.00     6883     0.00     0.00  Node::Node()
0.00       0.65     0.00     6883     0.00     0.00  TVector<Item>::TVector()
0.00       0.65     0.00     6883     0.00     0.00  TVector<Item>::~TVector()
0.00       0.65     0.00     6883     0.00     0.00  TVector<Node*>::TVector()
0.00       0.65     0.00     6883     0.00     0.00  TVector<Node*>::~TVector()
0.00       0.65     0.00     3440     0.00     0.00  Node::~Node()
0.00       0.65     0.00     3435     0.00     0.00  TVector<Node*>::Insert(unsigned
```

```
long long const&,Node* const&)
0.00      0.65     0.00        1     0.00     0.00  _GLOBAL__sub_I__ZltRK4ItemS1_
0.00      0.65     0.00        1     0.00     0.00  __static_initialization_and_destru
```

Как видно из лога, программа тратит большую часть времени на копирование и вызов коструктора структруы Item. Это логично так как все операции слияния и разбиения узлов требуют поэлементного копирования. На работу программы это валияет существенно, оптимизация разбиения и слияния узлв B-дерева является наиболее приоритетной.

3 Дневник Отладки

1. 14.11.20 Выявил и исправил некорректную работу деструктора дерева с помошью утлиты Valgrind.
2. 18.11.20 Выявил и исравил некорректное слияние и разбиение узлов B-дерева.
3. 20.11.20 Успешная посылка на чекер
4. 22.11.20 Исследовал скорость работы отдельных фукций программы с помошью утилиты Gprof.
5.

4 Выводы

В ходе выполненения лабораторной работы я изучил стредства отладки и профилирования Valgrind и Gprof. Применил их для отладки программы для лабораторной работы №2.

Подобные утилиты значительно ускоряют отладку программ, так как автоматически находят ее уязвимые места. Учитывая сколько времени могут сэкономить подобные средства, полагаю, что на практике эти утилиты применяются постоянно.

Время и память - основные ресурсы компьютера. Наиболее рациональное их использование повышает быстродействие и общее качество работы программы.

Список литературы

[1] *Valgrind - википедия*
URL: `https://ru.wikipedia.org/wiki/Valgrind` (дата обращения: 14.11.2020)

[2] *Что такое valgrind и зачем он нужен*
URL: `http://alexott.net/ru/writings/prog-checking/Valgrind.html`
(дата обращения: 14.11.2020)

[3] *Gprof - Wikipedia*
URL: `https://en.wikipedia.org/wiki/Gprofl`
(дата обращения: 16.11.2020).

[4] *Ускорение кода при помощи GNU-профайлера*
URL: `https://www.ibm.com/developerworks/ru/library/l-gnuprof/l`
(дата обращения: 18.11.2020).