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: М. А. Инютин
Преподаватель: А. А. Кухтичев
Группа: М8О-207Б-19

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: valgrind, gprof.

1 Valgrind

Согласно [1], Valgrid — это программа для поиска ошибок обращения с памятью, поиска утечек памяти и профилирования. Для поиска ошибки я использовал средство memcheck.

Я обнаружил, что моя первая версия программы может выдавать неправильный ответ. Для поиска ошибки я использовал Valgrind.

```
engineerxl@engineerxl:~/DA/lab2$ valgrind --tool=memcheck --leak-check=full
./solution <testing/test75.txt >/dev/null
==6371== Memcheck,a memory error detector
==6371== Copyright (C) 2002-2017,and GNU GPL'd,by Julian Seward et al.
==6371== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==6371== Command: ./solution
==6371==
==6371== Invalid read of size 8
==6371==    at 0x10A8B8: TVector<TItem>::operator[](unsigned long long const&)
(in /home/engineerxl/DA/lab2/solution)
==6371==    by 0x10C50C: void EraseNode<TItem>(TBtreeNode<TItem>*&,TItem const&)
(in /home/engineerxl/DA/lab2/solution)
==6371==    by 0x10B478: TBtree<TItem>::Erase(TItem const&) (in /home/engineerxl/DA/la
==6371==    by 0x109E84: main (in /home/engineerxl/DA/lab2/solution)
==6371==  Address 0x4dc7c80 is 0 bytes inside a block of size 48 free'd
==6371==    at 0x483D1CF: operator delete(void*,unsigned long) (in /usr/lib/x86_64-lin
==6371==    by 0x10D5C8: void EraseNode<TItem>(TBtreeNode<TItem>*&,TItem const&)
(in /home/engineerxl/DA/lab2/solution)
==6371==    by 0x10C4F0: void EraseNode<TItem>(TBtreeNode<TItem>*&,TItem const&)
(in /home/engineerxl/DA/lab2/solution)
==6371==    by 0x10B478: TBtree<TItem>::Erase(TItem const&) (in /home/engineerxl/DA/la
==6371==    by 0x109E84: main (in /home/engineerxl/DA/lab2/solution)

... Лог очень большой ...

==6371== HEAP SUMMARY:
==6371==     in use at exit: 0 bytes in 0 blocks
==6371==   total heap usage: 155 allocs,155 frees,158,208 bytes allocated
==6371==
==6371== All heap blocks were freed --no leaks are possible
==6371==
==6371== For lists of detected and suppressed errors,rerun with: -s
==6371== ERROR SUMMARY: 261 errors from 4 contexts (suppressed: 0 from 0)
```

Диагностика показала, что программа неверно обращается с удалённой памятью, из-за чего и возникает ошибка.

Я неправильно реализовал удаление из B-дерева. Если программа удаляла элемент из нетерминального узла, то она искала наименьшее большее значение, удаляла его, и пыталась записать это значение в тот же узел. Дерево могло перестроиться и указатель мог указывать на что-то другое после удаления. Для исправления я сделал поиск исхоного удаляемого значения и изменил значение в найденном узле.

```
engineerxl@engineerxl:~/DA/lab2$ valgrind --tool=memcheck --leak-check=full
./solution <testing/test75.txt >/dev/null
==6386== Memcheck,a memory error detector
==6386== Copyright (C) 2002-2017,and GNU GPL'd,by Julian Seward et al.
==6386== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==6386== Command: ./solution
==6386==
==6386==
==6386== HEAP SUMMARY:
==6386==     in use at exit: 0 bytes in 0 blocks
==6386==   total heap usage: 175 allocs,175 frees,176,464 bytes allocated
==6386==
==6386== All heap blocks were freed --no leaks are possible
==6386==
==6386== For lists of detected and suppressed errors,rerun with: -s
==6386== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

После иправления ошибки программа стала выдавать правильный ответ и Valgrind показал чистый результат.

2 GPROF

Как сказано в [3], Gprof — средство профилирования в Unix системах. Используется для измерения времени работы отдельных функций программы и общего времени работы 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 10000 строк с запросами на поиск, добавление и удаление.

```
engineerxl@engineerxl:~/DA/lab2/gprof$ gprof ./solution -p ./gmon.out
Flat profile:

Each sample counts as 0.01 seconds.
%
time    calls  name
75.04   27903  TItem::operator=(TItem const&)
25.01   23463  Clear(char*)
0.00  168425  unsigned short Min<unsigned short>(unsigned short const&,unsigned
short const&)
0.00  146858  TVector<TItem>::operator[](unsigned long long const&)
0.00  109344  operator<(TItem const&,TItem const&)
0.00   80572  TVector<TItem>::Size()
0.00   52618  TVector<TBtreeNode<TItem>*>::operator[](unsigned long long const&)
0.00   39119  unsigned long long BinSearch<TItem>(TVector<TItem>&,TItem const&)
0.00   28638  operator==(TItem const&,TItem const&)
0.00   13463  TItem::TItem()
0.00   10147  void FindNode<TItem>(TBtreeNode<TItem>*,TBtreeNode<TItem>*&,unsigned
long long&,TItem const&)
0.00   10000  TBtree<TItem>::Find(TBtreeNode<TItem>*&,unsigned long long&,TItem
const&)
0.00    8997  TVector<TBtreeNode<TItem>*>::Size()
0.00    6916  TVector<TBtreeNode<TItem>*>::PushBack(TBtreeNode<TItem>* const&)
0.00    4772  TVector<unsigned long long>::PushBack(unsigned long long const&)
0.00    3387  TVector<TBtreeNode<TItem>*>::PopBack()
0.00    2330  unsigned long long Max<unsigned long long>(unsigned long long
const&,unsigned long long const&)
0.00    2195  TVector<TItem>::PushBack(TItem const&)
0.00    2123  TVector<TItem>::Insert(unsigned long long const&,TItem const&)
0.00    2123  TVector<TBtreeNode<TItem>*>::Insert(unsigned long long const&,TBtreeNode
```

```
const&)
0.00    2030  TVector<TBtreeNode<TItem>*>::~TVector()
0.00    1899  TVector<unsigned long long>::Size()
0.00    1899  TVector<unsigned long long>::operator[](unsigned long long const&)
0.00    1826  TBtree<TItem>::Insert(TItem const&)
0.00    1826  TVector<TItem>::PopBack()
0.00    1825  void InsertNode<TItem>(TBtreeNode<TItem>*&,TBtreeNode<TItem>*&,TItem
const&)
0.00    1635  TVector<TBtreeNode<TItem>*>::TVector()
0.00    1635  TVector<unsigned long long>::TVector()
0.00    1635  TVector<unsigned long long>::~TVector()
0.00    1633  TVector<TItem>::Erase(unsigned long long const&)
0.00    1633  TVector<TBtreeNode<TItem>*>::Erase(unsigned long long const&)
0.00    1612  TVector<unsigned long long>::PopBack()
0.00    1488  void EraseNode<TItem>(TBtreeNode<TItem>*&,TItem const&)
0.00    1488  TBtree<TItem>::Erase(TItem const&)
0.00     395  TBtreeNode<TItem>::TBtreeNode()
0.00     395  TVector<TItem>::TVector(unsigned long long)
0.00     395  TVector<TItem>::~TVector()
0.00     395  TVector<TBtreeNode<TItem>*>::TVector(unsigned long long)
0.00     370  TVector<TBtreeNode<TItem>*>::Resize(unsigned long long const&)
0.00     359  TBtreeNode<TItem>::~TBtreeNode()
0.00     298  TVector<TItem>::Resize(unsigned long long const&)
0.00     147  TItem::TItem(TItem const&)
0.00      82  TVector<unsigned long long>::Resize(unsigned long long const&)
0.00       1  TBtree<TItem>::TBtree()
0.00       1  TBtree<TItem>::~TBtree()
```

Видно, что программа тратит большую часть времни не на операции с деревом, а на операции с ключом. Действительно, моя реализация B-дерева использует динамический массив в узлах, что требует большого количества копирований при вставке и удалении. Исходя из диагностика, наиболее приоритетным для оптимизации является работа со структурой «ключ-значение».

3 Дневник отладки

1. 05.11.2020 Использовал утилиту Valgrind для отладки программы на предмет утечек памяти. Найденная ошибка была устранена.
2. 06.11.2020 Ознакомился с дополнительными возможностями Valgrind
3. 13.11.2020 Изучил принцин работы с утилитой Gprof
4. 14.11.2020 Произвёл профилирование программы с помощью Gprof

4 Выводы

Выполнив третью лабораторную работу по курсу «Дискретный анализ», я изучил и применил средства диагностики и профилирования, в частности Valgrind и Gprof. Исходя из их популярности, понятно, что они часто используются на практике.

Valgrind помог мне найти неправильное обращение с памятью в программе и исправить эту ошибку.

Gprof показал, какие функции дольше всего выполняет моя программа, а именно работа со строками.

Эти утилиты помогают оптимизировать код с целью уменьшения потребляемой памяти и времени работы программы. Учитывая, что это основные ресурсы компьютера, то таким образом можно в целом улучшить качество работы и быстродейтсвие программы.

Список литературы

[1] *Valgrind — Wikipedia*
URL: `https://en.wikipedia.org/wiki/Valgrind` (дата обращения: 6.11.2020).

[2] *Ловим утечки памяти в C/C++ / Хабр — Habr.com*
URL: `https://habr.com/ru/post/480368/` (дата обращения: 6.11.2020).

[3] *Gprof — Wikipedia*
URL: `https://en.wikipedia.org/wiki/Gprof` (дата обращения: 13.11.2020).

[4] *Профилирование уже запущенных программ / Хабр — Habr.com*
URL: `https://habr.com/ru/post/167837/` (дата обращения: 14.11.2020).