Лабораторная работа № 3 по курсу дискретного анализа: исследование качества программ

Выполнила студентка группы 08-208 МАИ *Понагайбо Анастасия.*

Условие

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.

Дневник отладки

Valgrind

Была произведена диагностика реализации словаря с помощью Valgrind - инструментального программного обеспечения, предназначенного для отладки использования памяти, обнаружения утечек памяти, а также профилирования. Программа была запущена с помощью вызова **valgrind –log-file=log.txt –leak-resolution=high –leak-check=full –track-origins=yes ./lab2 <test.txt >out.txt**. Ключ **–log-file** задает имя файла, в который будет выводиться отчет о работе, ключ **–leak-resolution=high** сравнивает полный стек вызова функций, **–leak-check=full** включает функцию обнаружения утечек памяти, **–track-origins=yes** позволяет отслеживать неинициализированные данные. Были выявлены утечки памяти:

```
==4393== Memcheck, a memory error detector
==4393== Copyright (C) 2002-2013, and GNU GPL'd, by Julian Seward et al.
==4393== Using Valgrind-3.10.1 and LibVEX; rerun with -h for copyright info
==4393== Command: ./lab2
==4393== Parent PID: 4347
==4393==
==4393==
==4393== HEAP SUMMARY:
==4393==     in use at exit: 18,698 bytes in 1,753 blocks
==4393==   total heap usage: 13,585 allocs, 11,832 frees, 252,658 bytes allocated
==4393==
==4393== 2 bytes in 1 blocks are definitely lost in loss record 1 of 1,019
==4393==    at 0x402CDAC: operator new[](unsigned int) (in /usr/lib/valgrind/vgpreloa
==4393==    by 0x804A465: node::node(char*, unsigned long long) (lab2.cpp:22)
==4393==    by 0x80491F6: Insert(char*, unsigned long long, node*) (lab2.cpp:121)
==4393==    by 0x80492D8: Insert(char*, unsigned long long, node*) (lab2.cpp:138)
==4393==    by 0x80492D8: Insert(char*, unsigned long long, node*) (lab2.cpp:138)
==4393==    by 0x8049ECB: main (lab2.cpp:372)
```

```
==4393==
==4393== 2 bytes in 1 blocks are definitely lost in loss record 4 of 1,019
==4393==    at 0x402CDAC: operator new[](unsigned int) (in /usr/lib/valgrind/vgpreloa
==4393==    by 0x804A465: node::node(char*, unsigned long long) (lab2.cpp:22)
==4393==    by 0x8049787: Remove(char*, node*) (lab2.cpp:246)
==4393==    by 0x80495AD: Remove(char*, node*) (lab2.cpp:205)
==4393==    by 0x80495AD: Remove(char*, node*) (lab2.cpp:205)
==4393==    by 0x804966C: Remove(char*, node*) (lab2.cpp:220)
==4393==    by 0x8049F48: main (lab2.cpp:377)
==4393==
==4393== LEAK SUMMARY:
==4393==    definitely lost: 18,698 bytes in 1,753 blocks
==4393==    indirectly lost: 0 bytes in 0 blocks
==4393==      possibly lost: 0 bytes in 0 blocks
==4393==    still reachable: 0 bytes in 0 blocks
==4393==         suppressed: 0 bytes in 0 blocks
==4393==
==4393== For counts of detected and suppressed errors, rerun with: -v
==4393== ERROR SUMMARY: 1019 errors from 1019 contexts (suppressed: 0 from 0)
```

Исправив ошибку при удалении, удалось избавиться от утечек:

```
==4468== Memcheck, a memory error detector
==4468== Copyright (C) 2002-2013, and GNU GPL'd, by Julian Seward et al.
==4468== Using Valgrind-3.10.1 and LibVEX; rerun with -h for copyright info
==4468== Command: ./lab2
==4468== Parent PID: 4347
==4468==
==4468==
==4468== HEAP SUMMARY:
==4468==     in use at exit: 0 bytes in 0 blocks
==4468==   total heap usage: 13,585 allocs, 13,585 frees, 252,658 bytes allocated
==4468==
==4468== All heap blocks were freed -- no leaks are possible
==4468==
==4468== For counts of detected and suppressed errors, rerun with: -v
==4468== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

gprof

Предварительно скомпилировав программу с ключом **-pg** и получив после ее исполнения файл gmon.out, была произведена диагностика с помощью утилиты gprof. Были получены следующие данные:

```
Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
time   seconds   seconds    calls  ms/call  ms/call  name
56.86     0.58     0.58   624964     0.00     0.00  Insert(char*, unsigned long long,
17.65     0.76     0.18   124967     0.00     0.00  Remove(char*, node*)
15.20     0.92     0.15   250067     0.00     0.00  Find(char*, node*)
3.92      0.95     0.04        1    40.00    40.00  DeleteTree(node*)
2.94      0.98     0.03                             main
0.98      1.00     0.01   285629     0.00     0.00  FixBalance(node*)
0.98      1.01     0.01    46732     0.00     0.00  MinNode(node*, node*)
0.49      1.02     0.01                             Serialise(node*, _IO_FILE*)
0.00      1.02     0.00   620719     0.00     0.00  node::node(char*, unsigned long l
0.00      1.02     0.00   213076     0.00     0.00  RotateToLeft(node*)
0.00      1.02     0.00   212313     0.00     0.00  RotateToRight(node*)
0.00      1.02     0.00    46732     0.00     0.00  RemoveMin(node*)
```

Таким образом, дольше всего работают функции Insert и Remove. Вынеся общие части из блоков в функциях FixBalance, Insert и Remove, удалось немного их ускорить:

```
Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
time   seconds   seconds    calls  ms/call  ms/call  name
48.05     0.37     0.37   624964     0.00     0.00  Insert(char*, unsigned long long,
18.18     0.51     0.14   250067     0.00     0.00  Find(char*, node*)
15.58     0.63     0.12   124967     0.00     0.00  Remove(char*, node*)
5.19      0.71     0.04                             main
2.60      0.73     0.02    46732     0.00     0.00  MinNode(node*, node*)
1.95      0.75     0.01   620719     0.00     0.00  node::node(char*, unsigned long l
1.30      0.76     0.01    46732     0.00     0.00  RemoveMin(node*)
1.30      0.77     0.01        1    10.00    10.00  DeleteTree(node*)
0.00      0.77     0.00   285629     0.00     0.00  FixBalance(node*)
0.00      0.77     0.00   213076     0.00     0.00  RotateToLeft(node*)
0.00      0.77     0.00   212313     0.00     0.00  RotateToRight(node*)
```

Выводы

Утилита Valgrind помогает предотвратить чтение и запись за границами выделенного блока, попытки использования неинициализированной памяти, а также помогает найти утечки памяти, что дает возможность избежать многих ошибок.

Утилита gprof предоставляет информацию о времени работы каждой функции, частоте ее вызова и взаимодействии функций друг с другом, которая может быть в дальнейшем использовано для оптимизации программы.