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

Выполнил студент группы 08-208 МАИ *Левштанов Денис*.

Условие

1. 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

1. gprof

Для проверки своей программы утилитой gprof, я использовал флаг компиляции -pg, затем запускал исполняемый файл с тестами, в результате чего создавался файл gmon.out с данными о времени работы функций программы, затем с помощью самой утилиты gprof, эти данные представляются в понятном виде.

Проверив свою программу на 200 000 строк я получил такие результаты:

```
%   cumulative   self              self  total
time   seconds   seconds    calls  us/call  us/call  name
38.43    0.58  0.58 12584062  0.05  0.05  TSmartPointer<TElem>::operator->()
38.43    1.16  0.58   715833  0.81  1.62  FindElem(TVector<TSmartPointer<TElem> >&, char*)
 5.96    1.25  0.09  4955722  0.02  0.02  TSmartPointer<TElem>::operator=(TSmartPointer<TElem> const&)
 4.31    1.32  0.07 11726987  0.01  0.01  TVector<TSmartPointer<TElem> >::operator[](unsigned long)
 2.65    1.36  0.04  4883980  0.01  0.01  TSmartPointer<TNode>::operator->()
 1.99    1.39  0.03   146093  0.21  0.56  TVector<TSmartPointer<TElem> >::Erase(unsigned long)
 1.66    1.41  0.03  1279834  0.02  0.02  TVector<TSmartPointer<TElem> >::Size()
 1.33    1.43  0.02  5464522  0.00  0.00  TSmartPointer<TElem>::RemovePtr()
 1.33    1.45  0.02   242126  0.08  0.08  PrintMessage(int)
 1.33    1.47  0.02   106110  0.19  2.66  TBTree::DelInLeaf(TSmartPointer<TNode>&, int, char*)
 0.66    1.48  0.01  1521231  0.01  0.01  TVector<TSmartPointer<TNode> >::operator[](unsigned long)
```

Отсюда видно, что намного больше времени, чем все остальные занимают функции operator-> и FindElem. Если operator-> особо ускорить не получится, то функцию FindElem я ускорил, заменив обычный поиск, работающий за O(n), на бинарный, работающий за O(logn). Запустив исправленную программу с этими же тестами я получил:

```
Each sample counts as 0.01 seconds.
  %   cumulative   self              self  total
 time   seconds   seconds    calls  us/call  us/call  name
 30.01    0.21  0.21  4955722  0.04  0.06  TSmartPointer<TElem>::operator=(TSmartPointer<TElem> const&)
 20.01    0.35  0.14   715833  0.20  0.40  FindElem(TVector<TSmartPointer<TElem> >&, char*)
 18.58    0.48  0.13  9631914  0.01  0.01  TSmartPointer<TElem>::operator->()
 10.00    0.55  0.07  5464522  0.01  0.01  TSmartPointer<TElem>::RemovePtr()
  6.43    0.60  0.05  4883980  0.01  0.01  TSmartPointer<TNode>::operator->()
  5.00    0.63  0.04  8774839  0.00  0.00  TVector<TSmartPointer<TElem> >::operator[](unsigned long)
  4.29    0.66  0.03   146093  0.21  1.08  TVector<TSmartPointer<TElem> >::Erase(unsigned long)
  1.43    0.67  0.01  1279834  0.01  0.01  TVector<TSmartPointer<TElem> >::Size()
  1.43    0.68  0.01   715833  0.01  0.03  CheckExist(TVector<TSmartPointer<TElem> >&, char*, int)
  1.43    0.69  0.01   170632  0.06  0.06  TSmartPointer<TNode>::RemovePtr()
  1.43    0.70  0.01   106258  0.09  2.88  TBTree::DeleteNonEmpty(TSmartPointer<TNode>&, char*)
  0.00    0.70  0.00  1521231  0.00  0.00  TVector<TSmartPointer<TNode> >::operator[](unsigned long)
```

Тем самым программа ускорилась практически в 4 раза.

2. valgrind

Valgrind - инстринструментальное программное обеспечение, предназначенное для отладки использования памяти, обнаружения утечек памяти. Запустив свою программму с помощью valgrind –leak-check=full –show-leak-kinds=all, что в некоторых местах не очищается память, тем самым нагружая систему. Для решения этой проблемы и предотвращения ее появления в будущем я написал умные указатели, с помощью которых память освобождается сама.

```
==2403== 72,704 bytes in 1 blocks are still reachable in loss record 7 of 7
==2403==    at 0x4C2BBCF: malloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64-
linux.so)
==2403==    by 0x4EC21FF: ??? (in /usr/lib/x86_64-linux-gnu/libstdc++.so.6.0.21)
==2403==    by 0x40105C9: call_init.part.0 (dl-init.c:72)
==2403==    by 0x40106DA: call_init (dl-init.c:30)
==2403==    by 0x40106DA: _dl_init (dl-init.c:120)
==2403==    by 0x4000D09: ??? (in /lib/x86_64-linux-gnu/ld-2.21.so)
==2403==
==2403== LEAK SUMMARY:
==2403==    definitely lost: 0 bytes in 0 blocks
==2403==    indirectly lost: 0 bytes in 0 blocks
==2403==      possibly lost: 0 bytes in 0 blocks
==2403==    still reachable: 195,584 bytes in 7 blocks
==2403==         suppressed: 0 bytes in 0 blocks
==2403==
==2403== For counts of detected and suppressed errors, rerun with: -v
==2403== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Также я заметил баг в GCC который теряет 72,704 байт памяти.

Выводы

Утилита gprof помогает в оптимизации программы, с помощью неё можно узнать функции, в которых она долго работает и локально ее ускорить, не тратя время на ускорение незначительных функций. Утилита valgrind поможет узнать об утечках памяти, об освобождении уже освобожденной памяти и многие друие ошибки, тем самым не произойдет так, что после долгой работы програмы закончится оперативная память на копьютере.