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

Выполнил студент группы 08-208 МАИ *Былькова Кристина.*

Условие

Общая постановка задачи: 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. Результатом лабораторной работы является отчёт, состоящий из:

1. Дневника выполонения работы, в котором отражено что и когда делалось, какие средства использовались и какие результаты были достигнуты на каждом шаге выполнения лабораторной работы.

2. Выводов о найденных недочётах.

3. Сравнение работы исправленной программы с предыдущей версией. Общих выводов о выполнении лабораторной работы, полученном опыте.

Минимальный набор используемых средств должен содержать утилиту gprof и библиотеку dmalloc, однако их можно заменять на любые другие аналогичные или более развитые утилиты (например, Valgrind или Shark) или добавлять к ним новые (например, gcov).

Метод решения

Для исследования потребления памяти я использовала утилиту Valgrind. Это инструментальное программное обеспечение, предназначенное для отладки использования памяти, обнаружения утечек памяти, проверки потокобезопасности, а также профилирования. Наиболее используемым инструментом в этой утилите является Memcheck. Проблемы, которые может обнаружить Memcheck, включают в себя: 1. Попытки использования неинициализированной памяти 2. Чтение/запись в память после её освобождения 3. Чтение/запись за границами выделенного блока 4. Утечки памяти

Для отображения профильной статистики, которая накапливается во время приложения я использовала утилиту gprof. Профиллирование позволяет понять, где программа расходует свое время и какие функции вызывали другие функции, пока программа исполнялась. Эта информация может указать на ту часть программы, которая исполняется медленнее, чем ожидалось.

Дневник отладки

Для работы с утилитой Valgrind сначала скомпилируем программу, а затем пропишем в консоли valgrind —leak-check=full ./a.out < test.txt. Флаг —leak-check=full включает функцию обнаружения утечек памяти.

```
==5066== Memcheck, a memory error detector
==5066== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et
    al.
==5066== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright
    info
==5066== Command: ./a.out
==5066==
OK
OK
OK
OK
OK
OK
OK
OK
 Exist
OK
OK
OK
OK
OK
OK: 2
OK: 12
OK
 NoSuchWord
OK: 13
OK
 NoSuchWord
OK
OK
==5066==
==5066== HEAP SUMMARY:
==5066==     in use at exit: 122,880 bytes in 6 blocks
==5066==   total heap usage: 447 allocs, 441 frees, 210,162 bytes
    allocated
==5066==
==5066== LEAK SUMMARY:
==5066==    definitely lost: 0 bytes in 0 blocks
==5066==    indirectly lost: 0 bytes in 0 blocks
==5066==      possibly lost: 0 bytes in 0 blocks
```

```
==5066==    still reachable: 122,880 bytes in 6 blocks
==5066==         suppressed: 0 bytes in 0 blocks
==5066== Reachable blocks (those to which a pointer was found) are not
    shown.
==5066== To see them, rerun with: --leak-check=full
    --show-leak-kinds=all
==5066==
==5066== For lists of detected and suppressed errors, rerun with: -s
==5066== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Итак, в HEAP SUMMARY узнаем, что всего было аллоцировано 210,162 байта. В ERROR SUMMARY ошибки обнаружены не были.

Секция LEAK SUMMARY отвечает за количество утечек памяти, также там прописано о том, что осталось 122,880 bytes in 6 blocks категории still reachable. Память, относящяяся к этой категории не относится к утечкам памяти. Данная категория означает, что блоки не были освобождены, но они могли бы быть теоретически освобождены, потому что программа все еще отслеживала указатели на эти блоки памяти. Благодаря использованию флага --show-reachable=yes можно понять, что это из-за функции ios_base::sync_with_stdio(false);, которая отключает синхронизацию iostream с stdio. После исправления ещё раз запустим Valgrind:

```
==7169== Memcheck, a memory error detector
==7169== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et
    al.
==7169== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright
    info
==7169== Command: ./a.out
==7169==
OK
OK
OK
OK
OK
OK
OK
OK
Exist
OK
OK
OK
OK
OK
OK: 2
OK: 12
OK
NoSuchWord
```

```
OK: 13
OK
NoSuchWord
OK
OK
==7169==
==7169== HEAP SUMMARY:
==7169==     in use at exit: 0 bytes in 0 blocks
==7169==   total heap usage: 443 allocs, 443 frees, 92,402 bytes
    allocated
==7169==
==7169== All heap blocks were freed -- no leaks are possible
==7169==
==7169== For lists of detected and suppressed errors, rerun with: -s
==7169== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Теперь утечек памяти не обнаружено и всё в порядке.

Для отображения профильной статистики, которая накапливается во время работы программы используем утилиту gprof. Данная утилита выводит число вызовов функций при работе программы, определяет время работы каждой функции как обособленно, так и в сравнении с общим временем работы программы, что позволяет найти наиболее часто используемую функцию и в первую очередь оптимизировать именно её.

Чтобы воспользоваться gprof, необходимо ввести следующие команды в консоль:

1. Компилирование программы с флагом профилирования: g++ -pg lab2.cpp
2. Исполнение программы для порождения файла данных о профиле: ./a.out
3. Запуск 'gprof' для анализа данных о профиле: gprof ./a.out -p ./gmon.out

```
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  us/call  us/call  name
100.14      0.01     0.01    49806     0.20     0.20
    TString::TString(TString const&)
  0.00      0.01     0.00   492032     0.00     0.00
    TString::operator[](int)
  0.00      0.01     0.00   246977     0.00     0.00
    TString::PushBack(char)
  0.00      0.01     0.00   246977     0.00     0.00  TString::Size()
    const
  0.00      0.01     0.00   246016     0.00     0.00  ToLowerChar(char)
  0.00      0.01     0.00    52842     0.00     0.00
    TAVLTree<TString, unsigned long
    long>::GetHeight(TAVLTree<TString, unsigned long long>::TNode*)
```

```
0.00      0.01      0.00     50770      0.00      0.00
   TString::~TString()
0.00      0.01      0.00     16814      0.00      0.00
   TAVLTree<TString, unsigned long long>::BFactor(TAVLTree<TString,
   unsigned long long>::TNode*)
0.00      0.01      0.00     12463      0.00      0.40
   operator<(TString const&, TString const&)
0.00      0.01      0.00     12440      0.00      0.40
   operator>(TString const&, TString const&)
0.00      0.01      0.00      9607      0.00      0.00
   TAVLTree<TString, unsigned long long>::Max(unsigned char,
   unsigned char)
0.00      0.01      0.00      9607      0.00      0.00
   TAVLTree<TString, unsigned long
   long>::FixHeight(TAVLTree<TString, unsigned long long>::TNode*)
0.00      0.01      0.00      8301      0.00      0.00
   TAVLTree<TString, unsigned long long>::Balance(TAVLTree<TString,
   unsigned long long>::TNode*)
0.00      0.01      0.00      1922      0.00      0.00  TString::Clear()
0.00      0.01      0.00      1922      0.00      0.00
   operator>>(std::istream&, TString&)
0.00      0.01      0.00       964      0.00      0.00
   TString::TString()
0.00      0.01      0.00       961      0.00      0.00  ToLower(TString&)
0.00      0.01      0.00       961      0.00      0.00
   TString::operator=(TString const&)
0.00      0.01      0.00       961      0.00      5.21
   TAVLTree<TString, unsigned long
   long>::InsertNode(TAVLTree<TString, unsigned long long>::TNode*,
   TAVLTree<TString, unsigned long long>::TNode*)
0.00      0.01      0.00       961      0.00      0.00
   TAVLTree<TString, unsigned long long>::TNode::TNode(TString
   const&, unsigned long long const&)
0.00      0.01      0.00       961      0.00      0.00
   TAVLTree<TString, unsigned long long>::TNode::~TNode()
0.00      0.01      0.00       961      0.00     10.42
   TAVLTree<TString, unsigned long long>::Insert(TString const&,
   unsigned long long const&)
0.00      0.01      0.00       961      0.00      5.22
   TAVLTree<TString, unsigned long long>::FindTree(TAVLTree<TString,
   unsigned long long>::TNode*, TString const&)
0.00      0.01      0.00       961      0.00      0.00
   TString::operator==(char const*) const
0.00      0.01      0.00       327      0.00      0.00
   TAVLTree<TString, unsigned long
```

```
    long>::LeftRotate(TAVLTree<TString, unsigned long long>::TNode*)
0.00        0.01        0.00        326        0.00        0.00
    TAVLTree<TString, unsigned long
    long>::RightRotate(TAVLTree<TString, unsigned long long>::TNode*)
0.00        0.01        0.00        257        0.00        0.00
    TString::Resize(unsigned long)
0.00        0.01        0.00          1        0.00        0.00
    _GLOBAL__sub_I__ZN7TStringC2Ev
0.00        0.01        0.00          1        0.00        0.00
    __static_initialization_and_destruction_0(int, int)
0.00        0.01        0.00          1        0.00        0.00
    TAVLTree<TString, unsigned long
    long>::DeleteTree(TAVLTree<TString, unsigned long long>::TNode*)
0.00        0.01        0.00          1        0.00        0.00
    TAVLTree<TString, unsigned long long>::TAVLTree()
0.00        0.01        0.00          1        0.00        0.00
    TAVLTree<TString, unsigned long long>::~TAVLTree()
```

Благодаря данному отчету, можно увидеть, какие функции использовались чаще всего и сколько времени это занимало.

Выводы

В результате данной лабораторной работы я познакомилась с такими утилитами, как valgrind и grof, которые позволяют отлаживать программы: находить ошибки, утечки памяти. Благодаря этим утилитам легче разобраться, что не так в коде и устранить недочеты, так как набор сведений, полученный этими утилитами, дает подробные сведения о программе, а также о её недостатках, которые можно исправить или устранить, тем самым оптимизировав код.