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

Выполнил студент группы М8о-207Б МАИ *Токарев Никита*.

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. Результатом лабораторной работы является отчёт, состоящий из: Дневника выполнения работы, в котором отражено что и когда делалось, какие средства использовались и какие результаты были достигнуты на каждом шаге выполнения лабораторной работы. Выводов о найденных недочётов. Сравнение работы исправленной программы с предыдущей версии. Общих выводов о выполнении лабораторной работы, полученном опыте.

Метод решения

В данной работе я буду рассматривать такой инструмент как Valgrind. Valgrind является довольно удобным и хорошим инструментом для нахождения ошибок при работе с памятью. Но кроме этого, в его составе имееется несколько других и полезных утилит.

Типы ошибок при работе с memcheck:

1. Difinitely lost - не освободил память, при выходе указателя за область видимости.
2. Possibly lost - valgrind не уверен в том, что указатель на начало области памяти до сих пор существует.
3. Still reachable - нашёл указатель на начало не освобождённого блока памяти.

В ходе выполнения работы мной были допущены ряд ошибок:

Первая ошибка связана с неправильным чтением строки типа char из файла. Решением было выделять на один байт больше, записывая в последнее значение 0(key[len] = 0;).

```
Invalid read of size 1
==14150== at 0x1096CB: LengthOfByte(char*&) (main.cpp:21)
==14150== by 0x10B19C: TPatricia::DifferentBit(char*&, char*&) (main.cpp:375)
==14150== by 0x109F49: TPatricia::Insert(char*&, unsigned long long&) (main.cpp:104)
==14150== by 0x109829: main (main.cpp:427)
==14150==  Address 0x5ba0d72 is 0 bytes after a block of size 2 alloc'd
==14150==  at 0x4C3089F: operator new[](unsigned long)
==14150==  by 0x10A6E0: TPatricia::Load(char const*) (main.cpp:191)
==14150==  by 0x109A17: main (main.cpp:457)
```

Вторая ошибка связана с неверным удалением. С помощью вызова delete я вызывал деконструктор для корневого элемента, где очищался ключ(строка типа char), а сам корневой элемент удален не был и при повторном обращении происходила ошибка при обработке ключа данного узла.

```
Process terminating with default action of signal 11 (SIGSEGV)
==14370==  Access not within mapped region at address 0x0
==14370==    at 0x109698: LengthOfBits(char*&) (main.cpp:12)
==14370==    by 0x10B2B7: TPatricia::KeyCompare(char*&, char*&) (main.cpp:395)
==14370==    by 0x109E04: TPatricia::Find(char*) (main.cpp:77)
==14370==    by 0x109A57: main (main.cpp:466)
```

В результе ключейвой момент:

```
 if((del == head) && (del->branches[0] == head))
            delete del;
            del = nullptr;
            size--;
            return true;
```

В итоге я пришел к выводу заменить код:

```
if((del == head) && (del->branches[0] == head))
            delete head;
            head = nullptr;
            size--;
            return true;
```

С помощью утилиты gprof я также получил статистику вызовов, используя тест, преимущественно с операция поиска добавления и удаления. Так как протокол является довольно объемным выведу самые часто-используемые операции.

```
Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  us/call  us/call  name
 41.69      0.20     0.20  7229090     0.03     0.03  LengthOfBits
 29.18      0.34     0.14   128032     1.09     2.59  TPatricia::Insert
 12.51      0.40     0.06  6909014     0.01     0.04  TPatricia::DesiredBit
 12.51      0.46     0.06    32008     1.88     4.35  TPatricia::Delete
  2.08      0.47     0.01    45483     0.22     0.78  TPatricia::SearchParent
  2.08      0.48     0.01                             main
  0.00      0.48     0.00   505235     0.00     0.00  LengthOfByte
  0.00      0.48     0.00   172566     0.00     0.29  TPatricia::DifferentBit
```

```
  0.00      0.48      0.00   160040      0.00      0.00  Lowercase
  0.00      0.48      0.00   160038      0.00      0.18  TPatricia::KeyCompare
  0.00      0.48      0.00   101332      0.00      0.00  TNode::TNode
  0.00      0.48      0.00   101332      0.00      0.00  TNode::~TNode()
  0.00      0.48      0.00    27216      0.00      0.64  TPatricia::SearchBackward
```

Выводы

Профилирование кода является довольно мощным инструментом, благодаря которому можно заметно увеличить производительность программы. А один из самых важных компонентов успешной и используемой программы – высокая производительность. С помощью углубленного профилиования кода(для которого существуют различные утилиты), можно протестировать функцию и, впринципе, программу на частоту различных вызовов и также изменить порядок выполнения функции или программы, улучшив при этом производительность. Лично мне больше всего понравилось использовать Valgrind и также меня приятно удивило наличие модулей-анализаторов, каждый из которых преследует разные цели. Также можно совместить использование gdb и Valgrind. Также в ходе дальнейшей работы мне бы хотелось подробнее изучить модули-анализаторы Valgrind-a.