Московский авиационный институт (национальный исследовательский университет)

Факультет компьютерных наук и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №3 по курсу «Дискретный анализ»

Студент: А. А. Каримов
Преподаватель: А. А. Кухтичев
Группа: М8О-206Б-22

Дата:
Оценка:
Подпись:

Москва, 2024

Лабораторная работа №3

Задача: Задача: Для реализации словаря из предыдущей лабораторной работы необходимо провести исследование скорости выполнения и потребеления оперативной памяти. В случае выявления ошибок или явных недочетов исправить.

Используемые утилиты: `valgrind, gprof`

1 Valgrind

Valgrind — это инструментальная платформа для создания инструментов динамического анализа. Существуют инструменты Valgrind, которые могут автоматически обнаруживать многие ошибки управления памятью и многопоточностью, а также подробно профилировать ваши программы. Вы также можете использовать Valgrind для создания новых инструментов.

При использовании этой утилиты, я получил следующий вывод:

```
1 karseny99karseny99:/study/DA/lab2/new$ valgrind -tool=memcheck -leak-check=full
    ./a.out < ../tests/01.t > /dev/null==683== Memcheck, a memory error
    detector==683== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et
    al.==683== Using Valgrind-3.18.1 and LibVEX; rerun with -h for copyright
    info==683== Command: ./a.out==683====683====683== HEAP SUMMARY:==683== in use at
    exit: 0 bytes in 0 blocks==683== total heap usage: 7 allocs, 7 frees, 195,584
    bytes allocated==683====683== All heap blocks were freed -no leaks are
    possible==683====683== For lists of detected and suppressed errors, rerun with:
    -s==683== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)Этот вывод
    говорит о том, что моя программа чисто работает с памятью.
```

2 GPROF

GPROF - утилита, которая 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

```
Flat profile:

Each sample counts as 0.01 seconds.
  % cumulative self self total
 time seconds seconds calls us/call us/call name
 28.57 0.04 0.04 81353 0.49 0.90 string_to_lower_case(std::__cxx11::basic_string<char,
     std::char_traits<char>, std::allocator<char> >&)
 10.71 0.06 0.01 20953510 0.00 0.00 std::__cxx11::basic_string<char, std::char_traits<
    char>, std::allocator<char> >::operator[](unsigned long)
  7.14 0.07 0.01 26560605 0.00 0.00 std::__cxx11::basic_string<char, std::char_traits<
     char>, std::allocator<char> >::_M_data() const
  7.14 0.07 0.01 10476755 0.00 0.00 to_lower_case(char)
  7.14 0.09 0.01 2564493 0.00 0.00 std::_Sp_counted_base<(__gnu_cxx::_Lock_policy)2>::
     _M_add_ref_copy()
  7.14 0.10 0.01 1055052 0.01 0.01 std::__cxx11::basic_string<char, std::char_traits<
     char>, std::allocator<char> >::data() const
  7.14 0.10 0.01 561440 0.02 0.02 std::__cxx11::basic_string<char, std::char_traits<
     char>, std::allocator<char> >::_M_dispose()
  7.14 0.12 0.01 557960 0.02 0.02 std::char_traits<char>::copy(char*, char const*,
     unsigned long)
  7.14 0.12 0.01 482391 0.02 0.02 std::__cxx11::basic_string<char, std::char_traits<
     char>, std::allocator<char> >::_M_create(unsigned long&, unsigned long)
  7.14 0.14 0.01 436396 0.02 0.06 void std::__cxx11::basic_string<char, std::
     char_traits<char>, std::allocator<char> >::_M_construct<char*>(char*, char*, std
     ::forward_iterator_tag)
  3.57 0.14 0.01 490603 0.01 0.03 std::__cxx11::basic_string<char, std::char_traits<
     char>, std::allocator<char> >::~basic_string()
  0.00 0.14 0.00 14934472 0.00 0.00 std::__cxx11::basic_string<char, std::char_traits<
     char>, std::allocator<char> >::size() const
  0.00 0.14 0.00 6003728 0.00 0.00 std::__shared_ptr<NMap::TRBTree<NMap::TPair<std::
     __cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> >,
     unsigned long> >::TNode, (__gnu_cxx::_Lock_policy)2>::get() const
  0.00 0.14 0.00 5943780 0.00 0.00 std::__shared_ptr_access<NMap::TRBTree<NMap::TPair<
     std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> >,
     unsigned long> >::TNode, (__gnu_cxx::_Lock_policy)2, false, false>::_M_get()
     const
  0.00 0.14 0.00 5943780 0.00 0.00 std::__shared_ptr_access<NMap::TRBTree<NMap::TPair<
     std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> >,
     unsigned long> >::TNode, (__gnu_cxx::_Lock_policy)2, false, false>::operator->()
```

```
    const
21  0.00 0.14 0.00 4040979 0.00 0.00 std::__shared_ptr<NMap::TRBTree<NMap::TPair<std::
        __cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> >,
        unsigned long> >::TNode, (__gnu_cxx::_Lock_policy)2>::~__shared_ptr()
22  0.00 0.14 0.00 4040979 0.00 0.00 std::__shared_count<(__gnu_cxx::_Lock_policy)2>::~
        __shared_count()
23  0.00 0.14 0.00 3292172 0.00 0.00 std::shared_ptr<NMap::TRBTree<NMap::TPair<std::
        __cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> >,
        unsigned long> >::TNode>::~shared_ptr()
24  0.00 0.14 0.00 2723561 0.00 0.00 std::shared_ptr<NMap::TRBTree<NMap::TPair<std::
        __cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> >,
        unsigned long> >::TNode>::shared_ptr(std::shared_ptr<NMap::TRBTree<NMap::TPair<
        std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> >,
        unsigned long> >::TNode> const&)
```

3 Выводы

В ходе выполнения лабораторной работы я смог поиспользовать уже известную мне утилиту valgrind. При написании кода для решение олимпиадных задач или написания небольших проектов я использовал valgrind, чтобы отлавливать потенциальные баги, которые неочевидны на первый взгляд.

Список литературы

[1] *Профилирование уже запущенных программ.*
URL: `https://habr.com/ru/post/167837/`

[2] *Ловим утечки памяти в C/C++*
URL: `https://habr.com/ru/articles/480368/`