Московский авиационный институт (национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №3 по курсу «Дискретный анализ»

Студент: А. А. Кабанов
Преподаватель: С. А. Михайлова
Группа: М8О-301Б-22

Дата:
Оценка:
Подпись:

Москва, 2024

Лабораторная работа №3

Задача: 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.

1 Дневник выполнения работы

Создадим makefile для выполнения работы:

```
all:
  g++ -O2 -lm -fno-stack-limit -std=c++20 -x c++ ../lab2-patricia/solution.cpp -pg -g
      -o executable
run:
  (./executable < test.in) > test.out
gprof:
  make
  make run
  (gprof executable) > gprof.out
valgrind:
  make
  make run
  valgrind ./executable
clear:
  rm executable
```

Для выполнения работы воспользуемся утилитами gprof и valgrind.

Сегенерируем файл test.in, состоящий из 400000 команд на поиск, вставку и удаление вершин.

Запустим make gprof.

Посмотрим на содержимое файла gprof.out:

```
time seconds seconds calls ns/call ns/call name
88.94 0.08 0.08 main
11.12 0.09 0.01 99487 100.58 100.58 _dl_relocate_static_pie
0.00 0.09 0.00 100000 0.00 0.00 to_lowercase(std::__cxx11::basic_string<char, std::
    char_traits<char>, std::allocator<char> >&)
0.00 0.09 0.00 99485 0.00 0.00 leftmost_bit(std::__cxx11::basic_string<char, std::
    char_traits<char>, std::allocator<char> >&, std::__cxx11::basic_string<char, std::
    char_traits<char>, std::allocator<char> >&)
0.00 0.09 0.00 2 0.00 0.00 std::__cxx11::basic_string<char, std::char_traits<char>,
    std::allocator<char> >::_M_dispose()

index % time self children called name
<spontaneous>
[1] 100.0 0.08 0.01 main [1]
0.01 0.00 99487/99487 _dl_relocate_static_pie [2]
0.00 0.00 100000/100000 to_lowercase(std::__cxx11::basic_string<char, std::char_traits
    <char>, std::allocator<char> >&) [6]
0.00 0.00 99485/99485 leftmost_bit(std::__cxx11::basic_string<char, std::char_traits<
    char>, std::allocator<char> >&, std::__cxx11::basic_string<char, std::char_traits<
    char>, std::allocator<char> >&) [7]
0.00 0.00 2/2 std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<
    char> >::_M_dispose() [8]
-----------------------------------------------
```

```
0.01 0.00 99487/99487 main [1]
[2] 11.1 0.01 0.00 99487 _dl_relocate_static_pie [2]
-----------------------------------------------
0.00 0.00 100000/100000 main [1]
[6] 0.0 0.00 0.00 100000 to_lowercase(std::__cxx11::basic_string<char, std::
    char_traits<char>, std::allocator<char> >&) [6]
-----------------------------------------------
0.00 0.00 99485/99485 main [1]
[7] 0.0 0.00 0.00 99485 leftmost_bit(std::__cxx11::basic_string<char, std::char_traits
    <char>, std::allocator<char> >&, std::__cxx11::basic_string<char, std::char_traits
    <char>, std::allocator<char> >&) [7]
-----------------------------------------------
0.00 0.00 2/2 main [1]
[8] 0.0 0.00 0.00 2 std::__cxx11::basic_string<char, std::char_traits<char>, std::
    allocator<char> >::_M_dispose() [8]
-----------------------------------------------
```

Воспользуемся утилитой valgrind с помощью команды make valgrind:

```
==97772==
==97772== HEAP SUMMARY:
==97772== in use at exit: 0 bytes in 0 blocks
==97772== total heap usage : 10 allocs , 10 frees , 90,274 bytes allocated
==97772==
==97772== All heap blocks were freed == no leaks are possible
==97772==
==97772== For lists of detected and suppressederrors, rerun with : -s
==97772== ERROR SUMMARY: 0 errors from 0 contexts (suppressed : 0 from 0 )
```

2 Выводы

В ходе выполнения лабораторной работы были изучены инструменты для анализа и оптимизации производительности программного кода.

Утилита valgrind позволяет бнаружить и исправить ошибки памяти: утечки, чтение или запись в некорректные области памяти и использование неинициализированных переменных. Утилита позволяет оптимизировать код.

Утилита gprof предназначена для анализа производительности программного кода и определения узких мест. Она позволяет вычислить время выполнения каждой функции и подсчитать количество вызовов каждой функциию

Такие утилиты могут повысить качество и производительность программного кода.