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: Меджидли И. И. о
Преподаватель: Михайлова С. А
Группа: М8О-201Б-21

Дата: 07.09.2023
Оценка:
Подпись:

Москва, 2023

Лабораторная работа №3

Задача:

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. Составить дневник выполнения работы, в котором отражено что и когда делалось, какие средства использовались и какие результаты были достигнуты на каждом шаге выполнения лабораторной работы и сформулировать выводы.

2) 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).

1 Описание

1) Отладка при помощи определяющего профилирования (при помощи утилиты gprof):

Это способ, когда целевая программа модифицируется (в нашем случае компилируется с ключом –pg для компилятора g++). В код внедряются вызовы библиотеки профилирования, собирающие информацию о выполняемой программе и передающие её профилировку с помощью утилиты gprof.

2) Отладка при помощи Valgrind:

Valgrind — инструментальное ПО, предназначенное для контроля использования памяти и обнаружения её утечек. С помощью этой утилиты можно обнаружить попытки обращений к неициализированной памяти, работу с памятью после её освобождения и другое.

2 Тесты

Я воспользовался инструментом leak-check (аналогичен memcheck) в утилите Valgrind для контроля утечек памяти:

imedzhidli@imedzhidli: /Desktop/DA/LABA2$g++ lab2.cpp -Wall -Wextra$

imedzhidli@imedzhidli: /Desktop/DA/LABA2./$a.out$

imedzhidli@imedzhidli: /Desktop/DA/LABA2$valgrind --leak-check=full ./a.out$

==9666== Memcheck, a memory error detector

==9666== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.

==9666== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info

==9666== Command: ./a.out

==9666==

==9666==

==9666== HEAP SUMMARY:

==9666== in use at exit: 384 bytes in 6 blocks

==9666== total heap usage: 13 allocs, 7 frees, 75,439 bytes allocated

==9666==

==9666== 384 (64 direct, 320 indirect) bytes in 1 blocks are definitely lost in loss record 2 of 2

==9666== at 0x483BE63: operator new(unsigned long) (in /usr/lib/x86$_6$4$-linuxgnu/valgrind/vgpreload_memcheck-amd64-linux.so$)

==9666== by 0x10A8D4: Patricia::Add(std::$_{cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>const,u}$

==9666== by 0x10BEC7: main (in /home/imedzhidli/Desktop/DA/LABA2/a.out)

==9666==

==9666== LEAK SUMMARY:

==9666== definitely lost: 64 bytes in 1 blocks

==9666== indirectly lost: 320 bytes in 5 blocks

==9666== possibly lost: 0 bytes in 0 blocks

==9666== still reachable: 0 bytes in 0 blocks

==9666== suppressed: 0 bytes in 0 blocks

==9666==

==9666== For lists of detected and suppressed errors, rerun with: -s

==9666== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)

Видно, что происходит утечка данных. Все дело в том, что после прекращения подачи команд дерево не чистится перед завершением программы. После добавления строки "imedy.Clear();"в конец программы проблема исчезает:

imedzhidli@imedzhidli: /Desktop/DA/LABA2$g++lab2.cpp -Wall -Wextra$

imedzhidli@imedzhidli: /Desktop/DA/LABA2./$a.out$

imedzhidli@imedzhidli: /Desktop/DA/LABA2$valgrind --leak-check=full./a.out$

==9161== Memcheck, a memory error detector

==9161== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.

==9161== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info

==9161== Command: ./a.out

==9161==

==9161==

==9161== HEAP SUMMARY:

==9161== in use at exit: 0 bytes in 0 blocks

==9161== total heap usage: 13 allocs, 13 frees, 75,439 bytes allocated

==9161==

==9161== All heap blocks were freed – no leaks are possible

==9161==

==9161== For lists of detected and suppressed errors, rerun with: -s

==9161== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)

В последней строке Valgrind сообщает, что ошибок не обнаружено.Значит Valgrind по сути заменяет стандартное выделение памяти собственными методами, что позволяет утилите обнаруживать утечки. Теперь воспользуемся утилитой GNU Profiler (gprof), которая способна отобразить некоторую статистику отработки программы: Следует заметить, что программа изначально должна компилироваться с дополнительным флагом -pg

imedzhidli@imedzhidli: /Desktop/DA/LABA2$g++lab2.cpp-pgimedzhidli@imedzhidli: /Desktop/D$
./a.out imedzhidli@imedzhidli: /Desktop/DA/LABA2$gprof./a.out$

Flat profile:

Each sample counts as 0.01 seconds. no time accumulated

% time	cumulative seconds	self seconds	calls	self Ts/call	total Ts/call	name
0.00	0.00	0.00	54	0.00	0.00	Прочее
0.00	0.00	0.00	45	0.00	0.00	Patricia::Node
0.00	0.00	0.00	30	0.00	0.00	Прочее

0.00	0.00	0.00	30	0.00	0.00	Прочее
0.00	0.00	0.00	30	0.00	0.00	Прочее
0.00	0.00	0.00	27	0.00	0.00	Прочее
0.00	0.00	0.00	12	0.00	0.00	Прочее
0.00	0.00	0.00	12	0.00	0.00	Прочее
0.00	0.00	0.00	11	0.00	0.00	Прочее
0.00	0.00	0.00	9	0.00	0.00	Patricia::Add()
0.00	0.00	0.00	9	0.00	0.00	Patricia::Search()
0.00	0.00	0.00	9	0.00	0.00	Сравнение символов
0.00	0.00	0.00	9	0.00	0.00	Прочее
0.00	0.00	0.00	8	0.00	0.00	Patricia::Node::Node()
0.00	0.00	0.00	8	0.00	0.00	Patricia::Node:: Node()
0.00	0.00	0.00	7	0.00	0.00	Patricia::Insert()
0.00	0.00	0.00	3	0.00	0.00	Patricia::SearchE()
0.00	0.00	0.00	3	0.00	0.00	Прочее
0.00	0.00	0.00	3	0.00	0.00	Прочее
0.00	0.00	0.00	3	0.00	0.00	Прочее
0.00	0.00	0.00	3	0.00	0.00	Прочее
0.00	0.00	0.00	3	0.00	0.00	Прочее
0.00	0.00	0.00	3	0.00	0.00	Прочее
0.00	0.00	0.00	3	0.00	0.00	Прочее
0.00	0.00	0.00	3	0.00	0.00	Прочее
0.00	0.00	0.00	3	0.00	0.00	Прочее
0.00	0.00	0.00	3	0.00	0.00	Прочее
0.00	0.00	0.00	3	0.00	0.00	Прочее
0.00	0.00	0.00	3	0.00	0.00	Прочее
0.00	0.00	0.00	2	0.00	0.00	Patricia::Erase()
0.00	0.00	0.00	2	0.00	0.00	Прочее
0.00	0.00	0.00	2	0.00	0.00	Прочее
0.00	0.00	0.00	2	0.00	0.00	Прочее
0.00	0.00	0.00	2	0.00	0.00	Прочее
0.00	0.00	0.00	2	0.00	0.00	Прочее
0.00	0.00	0.00	2	0.00	0.00	Прочее
0.00	0.00	0.00	2	0.00	0.00	Прочее

0.00	0.00	0.00	2	0.00	0.00	Прочее
0.00	0.00	0.00	2	0.00	0.00	Прочее
0.00	0.00	0.00	2	0.00	0.00	Прочее
0.00	0.00	0.00	2	0.00	0.00	Прочее
0.00	0.00	0.00	1	0.00	0.00	Прочее
0.00	0.00	0.00	1	0.00	0.00	Прочее
0.00	0.00	0.00	1	0.00	0.00	Patricia::Clear()
0.00	0.00	0.00	1	0.00	0.00	Patricia::ClearNode()
0.00	0.00	0.00	1	0.00	0.00	Patricia:: Patricia()
0.00	0.00	0.00	1	0.00	0.00	Patricia::At()
0.00	0.00	0.00	1	0.00	0.00	Прочее
0.00	0.00	0.00	1	0.00	0.00	Прочее

В таблице 1-й столбец (% time) показывает процент от общего времени выполнения программы, используемого конкретной функцией.

2-й, 3-й, 5-й и 6-е столбцы – данные о времени выполнений функций. Вероятно, в таблице времена не указаны (no time accumulated) из-за малого количества действий.

4-й столбец (calls) отображает число вызовов функции за всё время работы программы.

В тестовом примере было 9 операций вставки и поиска, 2 удаления. В таблице количество вызовов этих функций совпадает с данными числами.

Как можно заметить, все работает верно.

3 Выводы

При выполнении четвертой лабораторной работы по курсу «Дискретный анализ» я познакомился с профилированием, крайне необходимым для качественной разработки, изучил возможные методы работы с ним, применив их на практике. Научился пользоваться Valgrind для контроля утечек памяти. Нашел утечку памяти в своей программе и исправил этот недочет. Также научился использовать утилиту-профайлер GNU Profiler (gprof), который 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. Надеюсь, мне понадобятся эти знания в будущем.