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: М. О. Чапалда
Преподаватель: С. А. Михайлова
Группа: М8О-201Б-22

Дата: _____
Оценка: _____
Подпись: _____

Москва, 2024

Условие

Для реализации словаря из предыдущей лабораторной работы, необходимо провести исследование скорости выполнения и потребления оперативной памяти. В случае выявления ошибок или вливших предосторов, требуется их исправить.

Методы решения

Изучение утилит *valgrind* и *gprof* для исследования качества программ и использование их для оптимизации программы.

- **Valgrind** — инструментальное ПО, предназначенное в основном для контроля использования памяти и обнаружения с утечек. С помощью этой утилиты можно обнаружить попытки использования (обращения) к неинициализированной памяти, работа с памятью после её освобождения и некоторые другие.
- Утилита **gprof** позволяет измерить время работы всех функций, методов и операторов программы, количество их вызовов и долю от общего времени работы программы в процентах.

Valgrind

Valgrind — инструментальное программное обеспечение, предназначенное для отладки использования памяти, обнаружения утечек памяти, а также профилирования. Первоначальная версия программы не содержала ошибок:

```
   mariyaach@LAPTOP-HSJSPL2E:~/MAI_DA/lab2$ valgrind -leak-check=full ./a.out <
code.txt | cat > output.txt

==3384== Memcheck, a memory error detector
==3384== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==3384== Using Valgrind-3.18.1 and LibVEX; rerun with -h for copyright info
==3384== Command: ./a.out
==3384==
==3384== HEAP SUMMARY:
==3384==     in use at exit: 0 bytes in 0 blocks
==3384==   total heap usage: 5 allocs, 5 frees, 80,988 bytes allocated
==3384==
==3384== All heap blocks were freed -- no leaks are possible
==3384==
==3384== For lists of detected and suppressed errors, rerun with: -s
==3384== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Gprof

Используя утилиту gprof, можно отследить, где и сколько времени проводила программа, тем самым выявляя слабые участки. Возьмем достаточно большой тест и вызовем gprof:

```
mariyaach@LAPTOP-HSJSPL2E:~/MAI_DA/lab3$ g++ -pg -o a.out main.cpp
mariyaach@LAPTOP-HSJSPL2E:~/MAI_DA/lab3$ ./a.out < test.txt | cat > output.txt
mariyaach@LAPTOP-HSJSPL2E:~/MAI_DA/lab3$ gprof a.out | cat > profile.info
```

% time	cumulative seconds	self seconds	calls	self Ts/call	total Ts/call	name
0.00	0.00	0.00	902	0.00	0.00	__gnu_cxx::__normal_iterator<char*
0.00	0.00	0.00	451	0.00	0.00	bool __gnu_cxx::operator!=<char*,
0.00	0.00	0.00	363	0.00	0.00	__gnu_cxx::__normal_iterator<char*
0.00	0.00	0.00	363	0.00	0.00	__gnu_cxx::__normal_iterator<char*
0.00	0.00	0.00	341	0.00	0.00	bool std::operator< <char, std::ch
0.00	0.00	0.00	287	0.00	0.00	__gnu_cxx::__enable_if<std::__is_c
0.00	0.00	0.00	153	0.00	0.00	bool std::operator==<char, std::ch
0.00	0.00	0.00	88	0.00	0.00	toLowerCase(std::__cxx11::basic_st
0.00	0.00	0.00	88	0.00	0.00	find(TreeNode*, std::__cxx11::basi
0.00	0.00	0.00	46	0.00	0.00	std::char_traits<char>::compare(ch
0.00	0.00	0.00	23	0.00	0.00	std::numeric_limits<long>::max()
0.00	0.00	0.00	22	0.00	0.00	insert(TreeNode*&, TreeNode*)
0.00	0.00	0.00	22	0.00	0.00	TreeNode::TreeNode(std::__cxx11::b
0.00	0.00	0.00	22	0.00	0.00	TreeNode::~TreeNode()
0.00	0.00	0.00	11	0.00	0.00	split(TreeNode*, TreeNode*&, TreeN
0.00	0.00	0.00	5	0.00	0.00	merge(TreeNode*, TreeNode*)
0.00	0.00	0.00	5	0.00	0.00	remove(TreeNode*&, std::__cxx11::b
0.00	0.00	0.00	1	0.00	0.00	__static_initialization_and_destru
0.00	0.00	0.00	1	0.00	0.00	clear(TreeNode*)

1. Производительность и оптимизация: Некоторые функции вызываются множество раз, что указывает на потенциальные точки для оптимизации производительности, особенно если время выполнения этих функций значительно.

2. Часто вызываемые функции: Функции типа

```
_gnu_cxx::new_allocator, std::allocator_traits, и std::_List_node
```

вызываются чрезвычайно часто, что может указывать на интенсивное использование структур данных, требующих динамического выделения памяти. Это может быть признаком потенциально неэффективного управления памятью.

Выводы

В рамках лабораторной работы я подробно изучила инструменты профилирования Gprof и Valgrind, которые значительно помогают в оптимизации программ.

Gprof предоставил детальную информацию о времени выполнения функций и их частоте вызовов, помогая идентифицировать функции с наибольшими затратами времени для дальнейшей оптимизации.

Valgrind был полезен для выявления утечек памяти и других ошибок использования памяти, что улучшило стабильность и эффективность программы.

Оба инструмента оказались незаменимыми для улучшения производительности и надежности программ, и я планирую использовать их в дальнейших проектах для повышения качества разработки программного обеспечения.