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

Выполнил студент группы М8О-212Б-22 МАИ *Юрков Евгений*.

Условие

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 недочётах.
- Сравнение работы исправленной программы с предыдущей версией.
- Общих выводов о выполнении лабораторной работы, полученном опыте.

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).

Метод решения

В рамках выполнения лабораторной работы я буду использовать следующие утилиты:

- Анализ времени работы: gprof
- Анализ потребления памяти: valgrind

gprof

Утилита gprof позволяет измерить время работы всех функций в программе, а также количество их вызовов и долю от общего времени работы программы в процентах.

Для работы с утилитой gprof было необходимо скомпилировать программу с ключом `-pg` После запуска полученного исполняемого файла появился файл gmon.out, в котом содержалась информация предоставленная для анализа программы. Далее этот файл был обработан gprof для получения текстового файла с подробной информацией о времени работы и вызовах всех функций и операторов, которые использовались в программе.

% time	self seconds	calls	name
33.35	0.01	3448130	`map<...>::__data::operator!=(...)`
33.35	0.01	3397716	`bool std::operator< (...)`
33.35	0.01	75095	`rb_tree<...>::__find_parent(...)`

Время работы остальных функций по данным gprof заняло меньше 0.01 секунды, поэтому они не были внесены в таблицу. Как можно заметить из таблицы, больше всего времени заняли функции сравнения ключей и поиска в дереве. Это связано с тем, что сравнение строк, являющихся ключами, происходит за линейное время, и функция сравнения вызывается на каждой итерации при поиске в дереве.

valgrind

Valgrind является утилитой для поиска ошибок в работе с памятью в программе, таких как утечки памяти и выход за границу массива.

В результате исследования программы valgrind была выявлена утечка памяти в функции очистки дерева. Для её устранения был написан деструктор для структуры `__node`, удаляющий также правого и левого потомков.

Все утечки были устранены. Результат работы valgrind:

```
==2555== Memcheck, a memory error detector
==2555== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==2555== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==2555== Command: ./build/Lab2/lab2
==2555==
==2555==
==2555== HEAP SUMMARY:
==2555== in use at exit: 122,880 bytes in 6 blocks
```

```
==2555== total heap usage: 25,073 allocs, 25,067 frees, 2,000,336 bytes allocated
==2555==
==2555== LEAK SUMMARY:
==2555== definitely lost: 0 bytes in 0 blocks
==2555== indirectly lost: 0 bytes in 0 blocks
==2555== possibly lost: 0 bytes in 0 blocks
==2555== still reachable: 122,880 bytes in 6 blocks
==2555== suppressed: 0 bytes in 0 blocks
==2555== Reachable blocks (those to which a pointer was found) are not shown.
==2555== To see them, rerun with: -leak-check=full -show-leak-kinds=all
==2555==
==2555== For lists of detected and suppressed errors, rerun with: -s
==2555== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Сообщение `still reachable: 122,880 bytes in 6 blocks` происходит из-за использования `ios::sync_with_stdio()`, это не является утечкой памяти.

Выводы

В ходе выполнения лабораторной работы я научился выявлять узкие места и проблемы в работе с памятью. Анализ времени работы программы с помощью утилиты gprof позволяет определить, какие функции занимают больше всего времени, что помогает в оптимизации кода для улучшения производительности. Анализ работы с памятью с помощью valgrind помогает выявить утечки памяти, неправильное использование указателей и другие проблемы, которые могут привести к непредсказуемому поведению программы.

Исследование качества программ необходимо для создания надежного, эффективного и безопасного программного обеспечения. Это позволяет улучшить пользовательский опыт, уменьшить количество ошибок и сбоев, повысить производительность и снизить затраты на поддержку.