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: Л. А. Постнов
Преподаватель: А. А. Кухтичев
Группа: М8О-206Б-22

Дата: 03.06.2024
Оценка:
Подпись:

Москва, 2024

Лабораторная работа №3

Задача: Требуется исследовать программу написанную при выполнении лабораторной работы №2 на утечки памяти и провести профилирование. В случае обнаружения ошибок, исправить их.

Используемые утилиты: valgrind, gprof

1 Описание

Valgrind - это GPL-система для отладки и профилирования программ Linux. С помощью Valgrind можно автоматически обнаруживать множество ошибок в управлении памятью и потоковой передаче, избегая многочасового поиска ошибок и повышая стабильность своих программ.

Gprof - это инструмент анализа производительности для приложений Unix.

Он использует гибрид инструментария и выборки и был создан как расширенная версия старого инструмента prof. В отличие от prof, gprof способен собирать и печатать ограниченный график вызовов.

Выходные данные GPROF состоят из двух частей:

1. Плоский профиль. Показывает общее время выполнения, затраченное на каждую функцию, и его процент от общего времени выполнения. Также сообщается количество вызовов функций.

2. Граф вызовов. Показывает для каждой функции, кто ее вызвал (родительская функция) и кого она вызвала (дочерние подпрограммы).

2 Работа с утилитой valgrind

После первого использования утилиты мы получили следующую картину: Как мож-

```
==169250==
==169250== HEAP SUMMARY:
==169250==     in use at exit: 123,616 bytes in 29 blocks
==169250==   total heap usage: 589 allocs, 560 frees, 474,944 bytes allocated
==169250==
==169250== LEAK SUMMARY:
==169250==    definitely lost: 736 bytes in 23 blocks
==169250==    indirectly lost: 0 bytes in 0 blocks
==169250==      possibly lost: 0 bytes in 0 blocks
==169250==    still reachable: 122,880 bytes in 6 blocks
==169250==         suppressed: 0 bytes in 0 blocks
==169250== Rerun with --leak-check=full to see details of leaked memory
==169250==
==169250== For lists of detected and suppressed errors, rerun with: -s
==169250== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Рис. 1: Результаты до исправления ошибок

но заметить, по полю *definitely lost*, после окончания работы программы было потеряно 736 байт памяти. Используя флаг *–leak-check=full* удалось обнаружить следующие ошибки при работе с памятью:

1. В функции *Node::merge*, я очищаю уже ненужные данные вершины *right child*, однако забываю очистить память выделенную непосредственно под саму вершину.

2. Аналогичную ошибку я допускаю в функции *BTree::remove* при очистке памяти, выделенной под старый корень дерева.

Кроме того, можно заметить утечку в 122800 байт, которая помечена полем *still reachable*, это связано с тем, что в моей программе используются следующие строчки:

```
1  std::ios_base::sync_with_stdio(false);
2  std::cin.tie(NULL);
3  std::cout.tie(NULL);
```

Эти строчки помогают ускорить работу программы, так как с их помощью отключается синхронизация между потоками ввода и вывода, а также не производяться лишние сбросы буфера. В поле *still reachable* valgrind показывает, что есть объекты, которые не были освобождены до завершения программы. В данном случае это связано как раз с отключением синхронизации, и, как следствие, утилита помечает

некоторые ресурсы, так как они не очщаются при обычном завершении программы, однако операционная система очщает их самостоятельно.

После проведения всех испроавлений, а также удаления вышеописанных строчек, мы получаем следующий результат, где видно, что утечек больше нет.

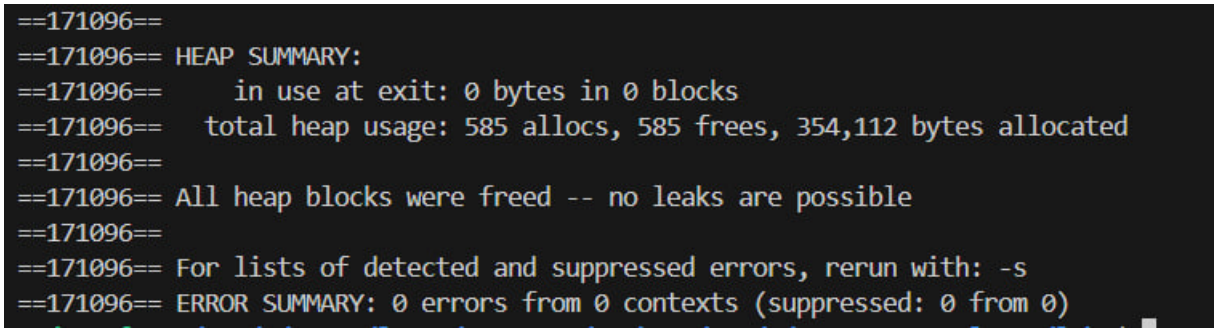

Рис. 2: Результаты после исправления ошибок

3 Работа с утилитой gprof

Рассмотрим вывод утилиты gprof на программе отработавшей с входными данными, состоящими из 10000 строк.

```
Flat profile:

Each sample counts as 0.01 seconds.
 no time accumulated

  %   cumulative   self              self     total
 time   seconds   seconds    calls  Ts/call  Ts/call  name
  0.00      0.00     0.00    16691     0.00     0.00  std::basic_istream<char, std::char_traits<char> >& std::o
perator>><char, std::char_traits<char> >(std::basic_istream<char, std::char_traits<char> >&, char*)
  0.00      0.00     0.00    10000     0.00     0.00  to_lover(char*)
  0.00      0.00     0.00     9999     0.00     0.00  btree::Node::search(char*)
  0.00      0.00     0.00     5085     0.00     0.00  Pair::Pair()
  0.00      0.00     0.00     3354     0.00     0.00  btree::BTree::insert(Pair)
  0.00      0.00     0.00     3353     0.00     0.00  btree::Node::insert_non_full(Pair)
  0.00      0.00     0.00     3336     0.00     0.00  btree::BTree::remove(char*)
  0.00      0.00     0.00     3310     0.00     0.00  btree::BTree::search(char*)
  0.00      0.00     0.00      565     0.00     0.00  btree::Node::Node(int, bool)
  0.00      0.00     0.00      560     0.00     0.00  btree::Node::split_child(int, btree::Node*)
  0.00      0.00     0.00        1     0.00     0.00  __static_initialization_and_destruction_0(int, int)
  0.00      0.00     0.00        1     0.00     0.00  btree::Node::destroy_node()
  0.00      0.00     0.00        1     0.00     0.00  btree::BTree::BTree(int)
  0.00      0.00     0.00        1     0.00     0.00  btree::BTree::~BTree()
```

Рис. 3: Flat profile

По выводу видно, какие функции вызываются чаще всего, и сколько они работают. Чаще всего вызывается функция *std::opertor»*, отвечающая за ввод данных. А сразу полсе идет функция, за перевод строки в нижний регистр, которая была введена, чтобы ключи в словаре были регистро независимыми.

Кроме того, утилита показывает, что ни одна из функций не занимает значительное время выполнения (все функции имеют 0.00 секунд), хотя вызываются часто, что сказывается на времени работы программы.

4 Дневник выполнения работы

1. 01.06.2024 Изучил работу утилиты valgrind и использовал ее для поиска утечек памяти в программе.
2. 02.06.2024 Изучил работу утилиты gprof и провел профилирование с ее помощью

5 Выводы

Выполняя данную лабораторную работу я поработал с двумя утилитами улучшающими работу программы, а именно valgrind и gprof, с помомщью первой удалось отловить утечку памяти возникщую из-за ошибки по невнимательности, а с помощью второй - провести профилирование программы. Я получил очень ценный опыт работы с этими утилитами и буду и впредь использовать их в дальнейшем.

Список литературы

[1] Томас Х. Кормен, Чарльз И. Лейзерсон, Рональд Л. Ривест, Клиффорд Штайн. *Алгоритмы: построение и анализ, 2-е издание.* — Издательский дом «Вильямс», 2007. Перевод с английского: И. В. Красиков, Н. А. Орехова, В. Н. Романов. — 1296 с. (ISBN 5-8459-0857-4 (рус.))

[2] *gprof - opennet*
URL: `https://www.opennet.ru/docs/RUS/gprof/` (дата обращения: 02.06.2024).

[3] *valgrind*
URL: `https://valgrind.org/docs/manual/manual.html` (дата обращения: 01.06.2024).