Лабораторная работа №3 по курсу Дискретного Анализа: Инструментирование и профилирование

Выполнил студент группы 08-208 МАИ Жерлыгин Максим Андреевич

Введение

Для отладки различных приложений программисту крайне часто приходится пользоваться дополнительными программами или утилитами, которые позволяют находить логические ошибки или баги. Но, помимо этого, при разработке программ, написанных на языках, где программист может собственноручно оперировать памятью вычислительной машины требуется следить, чтобы при отработке определённых блоков программы выполнялось освобождение ранее использованной памяти. Кроме того, есть потребность в получении данных о производительности программы.

Задание

Реализовать Абстрактный тип данных под названием б-дерево, которое поддерживает следующие операции – вставка, удаление, поиск, сохранение и загрузка в файл.

Метод решения

Для реализации данного типа данных для начала требуется ознакомится с самой структурой. Все реализации методов для данного дерева были взяты из книги «Алгоритмы и структуры данных» под авторством Кормена, Лейзерсона, Ривеста и Штайна. Последовательно реализуем алгоритмы поиска ключа, объединения двух дочерних узлов, вставки в дерево, далее удаление с присущими ему операциями. Далее требуется провести тестирование программы, проверить линейность основных операции а также наличие утечек памяти. Для этого будем использовать утилиты valgrind, gprof.

Отладка и проверка программы

В ходе выполнения всей лабораторной работы я столкнулся с рядом проблем, вызванных выходом за пределы массива или утечек памяти, однако большинство из них решались довольно просто и не требовали использования дополнительных утилит.

Но необошлось и без довольно серьезных проблем, из-за которых мне пришлось использовать средства отладки и поиска ошибок. Первая проблема появилась на самом раннем этапе моего написания данной программы.

```
valgrind ./a.out
==2268== Memcheck, a memory error detector
==2268== Copyright (C) 2002−2017, and GNU GPL'd, by Julian Sewardetal.
```

==2268== Using Valgrind −3.13.0 and LibVEX ; rerun with −h for copyright info
==2268== Command: ./a.out
==2268==
+ abababa 123
OK
==2268==
==2268== HEAP SUMMARY:
==2268== in use at exit : 122 ,880 bytes in 6 blocks
==2268== total heap usage : 10 allocs, 4 frees , 222,552 bytes allocated
==2268==
==2268== LEAK SUMMARY:
==2268== definitely lost: 0 bytes in 0 blocks
==2268== indirectly lost: 0 bytes in 0 blocks
==2268== possibly lost: 0 bytes in 0 blocks
==2268== still reachable: 122,880 bytes in 6 blocks
==2268== suppressed: 0 bytes in 0 blocks
==2268== Rerun with −−leak−check=full to see details of leaked memory
==2268==
==2268== For counts of detected and suppressed errors, rerun with: −v
==2268== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)

Как мы видим по итогу выполнения программы всегда остается доступная для использования память. Для устранения этой проблемы попросим valgrind вывести более подробную информацию с помощью ключей –leak-check=full –show-leak-kinds=all. Стало очевидным, что данная утечка возникает при отключении синхронизации между двумя потоками.

Вывод

При помощи средств valgrind удалось обнаружить утечку памяти. Однако на этапе тестирования стало понятно, что сам фреймворк valgrind сильно тормозит работу моей программы из-за чего стало неудобным проводить дальнейшие исправления.
Как альтернатива valfrind'y, можно использовать утилиту ASAN, которая использует немного иной режим работы при сборе информации во время работы программы.
Address Sanitizer довольно прост в использовании и подключается при помощи ключа -fsanitize=address.

==========2490==
ERROR: LeakSanitizer : detected memory leaks Direct leak of 91920 byte(s) in 1149 object(s) allocated from:
#0 0x7fd2a6e0f618 in operator new[](unsigned long) (/usr/lib/x86_64−linux−gnu/libasan.so.4+0xe0618)
#1 0x560c3b22bf37 in BTree<int,5>::BTreeNode<int,5> ::BTreeNode() (/mnt/c/Users/mma

xi/Desktop/coursera/DA_ex/lab2/a.out+0x1f37)
#2 0x560c3b22c2c6 in BTree<int,5>::BTreeNode<int,5> ::BTreeSplitChild(BTree<int,5>::BTree Node<int,5>∗,int) (/mnt/c/Users/mmaxi/Desktop/coursera/DA_ex/lab2/a.out+0x22c6)
#3 0x560c3b22cd8b in BTree<int,5>::BTreeNode<int,5> ::BTreeInsertNonfull(int)(/mnt/c/Users /mmaxi/Desktop/coursera/DA_ex/lab2/a.out+0x2d8b)
#4 0x560c3b22ce72 in BTree<int,5>::BTreeNode<int,5> ::BTreeInsertNonfull(int)(/mnt/c/Users /mmaxi/Desktop/coursera/DA_ex/lab2/a.out+0x2e72)
#5 0x560c3b22ce72 in BTree<int,5>::BTreeNode<int,5>::BTreeInsertNonfull(int)(/mnt/c/Users /mmaxi/Desktop/coursera/DA_ex/lab2/a.out+0x2e72)
6 0x560c3b22ce72 in BTree<int,5>::BTreeNode<int,5>::BTreeInsertNonfull(int)(/mnt/c/Users/m maxi/Desktop/coursera/DA_ex/lab2/a.out+0x2e72)