Лабораторная работа №3 по курсу дискретного анализа: исследование качества программ.

Выполнил студент группы М8О-208Б-20 МАИ Примаченко Александр.

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.

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

Для решение данной задачи было использовано две утилиты, такие как:

Программы вроде gprof называются профайлер (Инструмент, используемый для анализа работы). Они предназначены для определения быстродействия вашей программы. Для каждого вызова функции вашей программы профайлер выводит время ее выполнения. Вы как программист анализируете полученную информацию и, если нужно, оптимизируете исходный код вашей программы.

Valgrind — инструментальное программное обеспечение, предназначенное для отладки использования памяти, обнаружения утечек памяти, а также профилирования.

Дневник отладки

Для начала я запустим программу с утилитой gprof. Утилита gprof хороша тем, что позволяет увидеть время работы всех функций, реализованных в программе, количество их вызовов и вычисляет процентное соотношение работы конкретной функции по сравнению с работой всей программы. В таблице представлены наиболее важные из них:

```
sashapaladin@sashapaladin: g++ -o prog main.cpp -pg
sashapaladin@sashapaladin: gprof
./prog Each sample counts as
0.01 seconds.
time seconds seconds calls
Ts/call Ts/call name 27.08 0.81
```

```
  0.81 99999 8.72 13.02 DeleteNode
  2.98 3.06 0.19 99999 1.19 2.01
  InsertNode 12.19 1.29 0.41 main
  2.56 3.04 0.08 99999 0.81 1.53 SearchNode
  0.39 3.32 0.01 159503 0.76 0.07 RotateLeft
  0.01 3.14 0 137908 0 0.01 RotateRight
```

Первый столбец цифр показывает, какое количество времени в процентах от общего времени работы программы выполнялась та или иная функция. Третий столбец показывает количество вызовов различных функций.
Как мы можем увидеть, для каждого метода, наибольшее время выполнения у всех трех основных операций с деревом: вставкой, поиском и удалением.
Далее воспользуемся утилитой valgrind. Одной из самых лучших утилит для поиска всевозможных утечек памяти в программе. Проверять будем со специальным флагом –leak-check=full:

```
sashapaladin@sashapaladin: /Загрузки valgrind –leak-check=full
./prog ==54536== Memcheck, a memory error detector
==54536== Copyright (C) 2002-2017, and GNU GPL’d, by Julian
Seward et al.
==54536== Using Valgrind-3.18.1 and LibVEX; rerun with -h for
copyright info
==54536== Command: ./prog
==54536==
==54536==
==54536== Process terminating with default action of signal 27
(SIGPROF)
==54536== at 0x4BCEBCA: opennocancel(open64nocancel.c:39)
==54536== by 0x4BDD71F: writegmon(gmon.c : 370)
==54536== by 0x4BDDF8E: mcleanup(gmon.c : 444)
==54536== by 0x4AFAA55: cxaf inalize(cxaf inalize.c:83)
==54536== by 0x10A526: ??? (in /home/kristina/Загрузки/prog)
==54536== by 0x400624D: dlf ini(dl – f ini.c : 142)
==54536== by 0x4AFA494: runexithandlers(exit.c:113)
==54536== by 0x4AFA60F: exit (exit.c:143)
==54536== by 0x4ADED96: (below main) (libcstartcallmain.h : 74)
==54536==
==54536== HEAP SUMMARY:
==54536== in use at exit: 220,944 bytes in 8 blocks
==54536== total heap usage: 8 allocs, 0 frees, 220,944 bytes
allocated ==54536==
==54536== LEAK SUMMARY: ==54536== definitely lost: 0 bytes in 0
blocks ==54536== indirectly lost: 0 bytes in 0 blocks ==54536==
possibly lost: 0 bytes in 0 blocks ==54536== still reachable:
220,944 bytes in 8 blocks ==54536== suppressed: 0 bytes in 0
blocks
==54536== Reachable blocks (those to which a pointer was found)
are not shown. ==54536== To see them, rerun with: –leak-
check=full –show-leak-kinds=all ==54536==
==54536== For lists of detected and suppressed errors, rerun
with: -s
```

```
==54536== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0
from 0)
```

Последняя строка показывает, что в данной программе утечек по памяти не обнаружено.

Выводы

Данная лабораторная работа №3 помогла мне закрепить навыки работы с утилитой для нахождения утечек памяти valgrind, а также познакомила меня с очень интересной утилитой gprof, которая помогает пользователю увидеть время выполнения отдельных функций, что очень полезно для огромных программ.