Московский авиационный институт (национальный исследовательский университет)

Институт информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №3 по курсу «Дискретный анализ»

Студент: Л. В. Ивенкова
Преподаватель: Н. С. Капралов
Группа: М8О-208Б-19

Дата:
Оценка:
Подпись:

Москва, 2021

Лабораторная работа №3

Задача: 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 недочётов.

Сравнение работы исправленной программы с предыдущей версии.

Общих выводов о выполнении лабораторной работы, полученном опыте.

Минимальный набор используемых средст должен содержать утилиту gprof и библиотеку dmalloc, однако их можно заменять на любые другие аналогичные или более известные утилиты (например, Valgrind или Shark) или добавлять к ним новые (например, gcov).

1 Поиск утечек памяти Valgrind

Согласно ([1]): Valgrind — инструментальное программное обеспечение, предназначенное для отладки использования памяти, обнаружения утечек памяти, а также профилирования.

Я обнаружила, что одна из моих первых верий программы хотя работала правильно, в ней всё же происходили большие утечки памяти:

```
parsifal@DESKTOP-3G70RV4:~/DA/Lab2$ valgrind --tool=memcheck --leak-check=full
./a.out <test01.t

==1141== Memcheck,a memory error detector
==1141== Copyright (C) 2002-2017,and GNU GPL'd,by Julian Seward et al.
==1141== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==1141== Command: ./a.out
==1141==
==1141== error calling PR_SET_PTRACER,vgdb might block
==1141== Invalid write of size 1
==1141==    at 0x4842B63: memmove (in /usr/lib/x86_64-linux-gnu/valgrind/
vgpreload_memcheck-amd64-linux.so)
==1141==    by 0x4915664: std::basic_istream<char,
std::char_traits<char>>& std::operator>><char,std::char_traits<char>>
(std::basic_istream<char,std::char_traits<char>>&,char*)
(in/usr/lib/x86_64-linux-gnu/libstdc++.so.6.0.28)
==1141==    by 0x10A990: main (in /home/parsifal/DA/Lab2/a.out)
==1141==  Address 0x4dc3e00 is 0 bytes inside a block of size 257 free'd
==1141==    at 0x483D74F: operator delete[](void*)
(in/usr/lib/x86_64-linux-gnu/valgrind/vgpreload_memcheck-amd64-linux.so)
==1141==    by 0x10AE7E: TAVL::TNode::~TNode() (in /home/parsifal/DA/Lab2/a.out)
==1141==    by 0x10B4A2: TAVL::Remove(TAVL::TNode*,char*)
(in /home/parsifal/DA/Lab2/a.out)
==1141==    by 0x10AA78: main (in /home/parsifal/DA/Lab2/a.out)
==1141==  Block was alloc'd at
==1141==    at 0x483C583: operator new[](unsigned long) (in
/usr/lib/x86_64-linux-gnu/valgrind/vgpreload_memcheck-amd64-linux.so)
==1141==    by 0x10A955: main (in /home/parsifal/DA/Lab2/a.out
....
==1141==
```

```
==1141==
==1141== HEAP SUMMARY:
==1141==     in use at exit: 125,450 bytes in 16 blocks
==1141==   total heap usage: 20 allocs,5 frees,198,491 bytes allocated
==1141==
==1141== 2,570 bytes in 10 blocks are definitely lost in loss record 1 of 7
==1141==    at 0x483C583: operator new[](unsigned long)
(in/usr/lib/x86_64-linux-gnu/valgrind/vgpreload_memcheck-amd64-linux.so)
==1141==    by 0x10ACA3: main (in /home/parsifal/DA/Lab2/a.out)
==1141==
==1141== LEAK SUMMARY:
==1141==    definitely lost: 2,570 bytes in 10 blocks
==1141==    indirectly lost: 0 bytes in 0 blocks
==1141==      possibly lost: 0 bytes in 0 blocks
==1141==    still reachable: 122,880 bytes in 6 blocks
==1141==         suppressed: 0 bytes in 0 blocks
==1141== Reachable blocks (those to which a pointer was found) are not shown.
==1141== To see them,rerun with: --leak-check=full --show-leak-kinds=all
==1141==
==1141== For lists of detected and suppressed errors,rerun with: -s
==1141== ERROR SUMMARY: 31 errors from 18 contexts (suppressed: 0 from 0)
```

Здесь мы можем увидеть, что проблема в строке $TKey * key = newchar[SIZE_KEY + 1]$; в цикле *while* - в переобъявлении указателей. У нас терялась память прошлого объявленного указателя.

Чтобы избавиться от этой ошибки я заменила $TKey * key = newchar[SIZE_KEY + 1]$; на $key = newchar[SIZE_KEY + 1]$;.

```
==1301== Memcheck,a memory error detector
==1301== Copyright (C) 2002-2017,and GNU GPL'd,by Julian Seward et al.
==1301== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==1301== Command: ./a.out <test01.t
==1301==
==1301== error calling PR_SET_PTRACER,vgdb might block
...
==1301==
==1301== HEAP SUMMARY:
==1301==     in use at exit: 124,165 bytes in 11 blocks
==1301==   total heap usage: 14 allocs,3 frees,197,166 bytes allocated
==1301==
```

```
==1301== LEAK SUMMARY:
==1301==    definitely lost: 1,285 bytes in 5 blocks
==1301==    indirectly lost: 0 bytes in 0 blocks
==1301==      possibly lost: 0 bytes in 0 blocks
==1301==    still reachable: 122,880 bytes in 6 blocks
==1301==         suppressed: 0 bytes in 0 blocks
==1301== Rerun with --leak-check=full to see details of leaked memory
==1301==
==1301== For lists of detected and suppressed errors,rerun with: -s
==1301== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

2 Оценка времения работы программы Callgrind

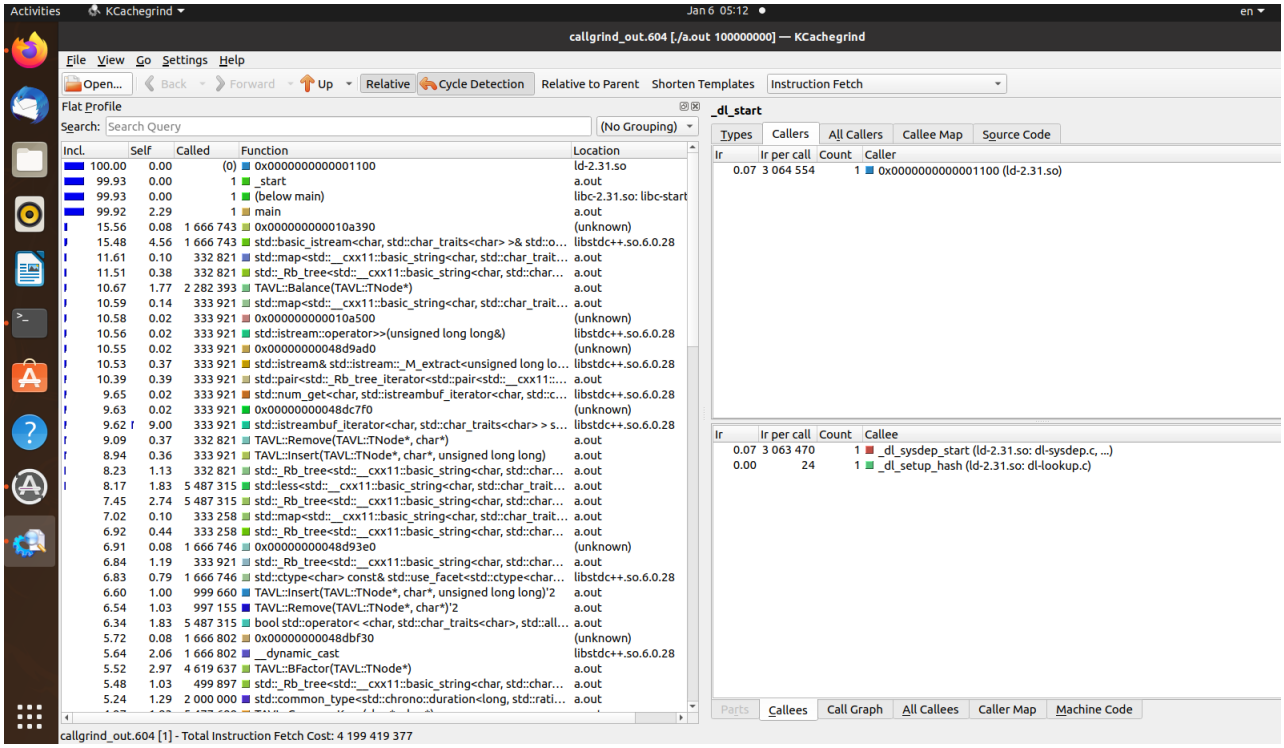
Callgrind находится в составе Valgrind'а. Является одной из наиболее удобных утилит, а вкупе с утилитой kcachegrind, наглядных инструментов. Благодаря простому интерфейсу можно быстро найти, сколько раз вызывается та или иная функция, сколько выполняется по времени, сколько процентов времени от общего вызова она занимает и прочее.

Для того, чтобы получить сведения о производительности программы, сначала ее скомпилируем (следует это делать с флагом компиляции -g). После этого запустим профайлер следующим образом:

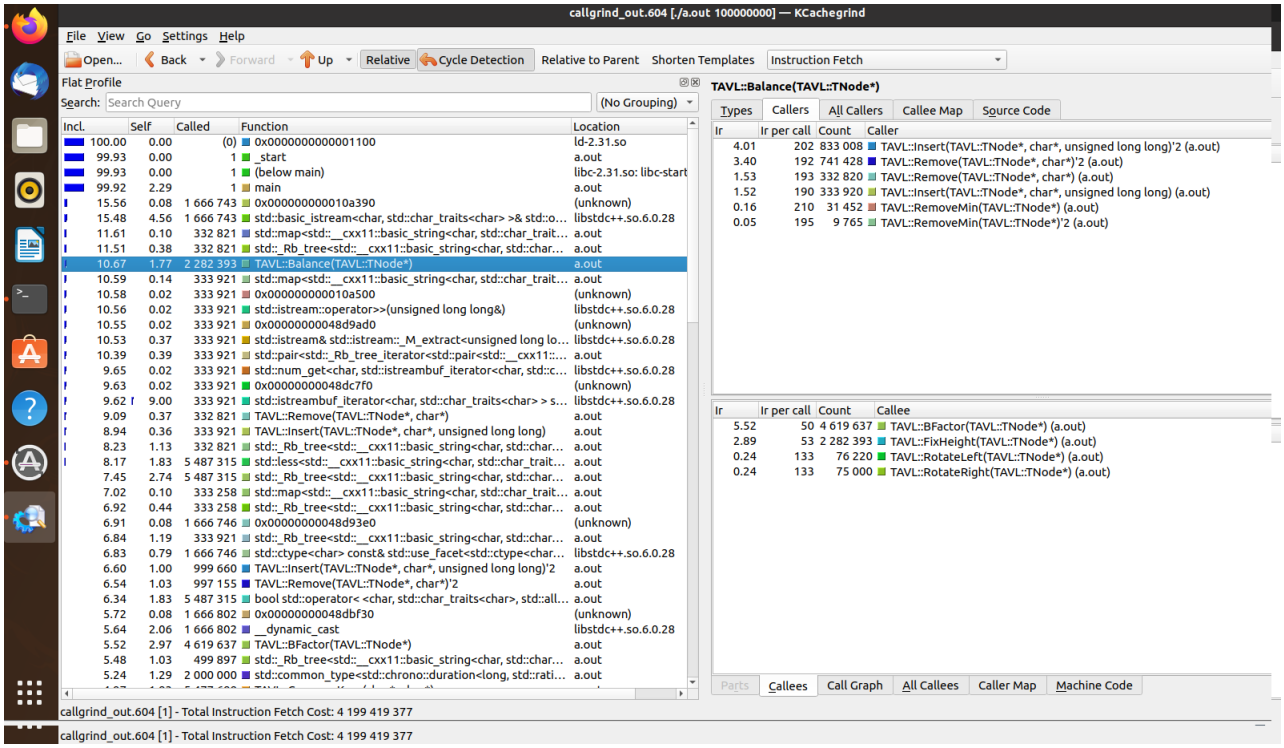
```
parsifal@ubuntu:~/Downloads$ LD_BIND_NOW=1 valgrind -tool=callgrind ./a.out
<test01.t
```

Таким образом сгенерировался файл callgrind.out.XXXX. Откроем:

```
parsifal@ubuntu:~/Downloads$ kcachegrind callgrind_out.604
```



Слева отображается таблица функций, отранжированных по умолчанию по времени выполнения по убыванию. Нас интересуют функции, написанные нами — возможно, анализ данных функций, позволит нам понять, каким образом мы можем оптимизировать нашу программу. Выбрав одну из функций, например TAVL::Balance, справа можно наблюдать целый набор инструментов, предоставляющий подробную информацию о ней:



Одни из самых хорошо интерпретируемых полей здесь, как и упоминалось выше, callee map и call graph. Судя по данным графикам, наиболее «долгая» функция — встроенная функция записи в файл. Её оптимизировать, к сожалению, на данном этапе моего обучения не предоставляется возможным.

3 Выводы

В этой лабораторной работе я проанализировала код программы и саму программу, которую написала к предыдущей работе, и выделила для себя наиболее удобные инструменты профилирования. Callgrind вкупе с kcachegrind дает достаточно полную информацию о производительности всей программы и ее частей в частности. Также мне понравился Valgrind с его богатым инструментарием и подробным описанием ошибок и их причины. Борьба с неправильным распределением ресурсов памяти и низкой производительностью - это одна из главных частей в разработке ПО. Не своевременное устранение ошибок и багов программы могжет привести к сбою. Инстументы, изученные мной в этой лабораторной работе, в будущем помогут мне в профилировании других программ.

Список литературы

[1] *Valgrind — Википедия.*
URL: `https://ru.wikipedia.org/wiki/Valgrind` (дата обращения: 5.01.2021).