Московский авиационный институт (национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №3 по курсу «Дискретный анализ»

Студент: А. А. Терво
Преподаватель: А. А. Кухтичев
Группа: М8О-207Б

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.

Используемые утилиты: valgrind, gprof.

1 Valgrind

Как сказано в [1]: «Valgrind – гибкая программа для дебаггинга и профилирования исполняемых файлов Linux».
Valgrind имеет много применений, но я использовал его для проверки корректной работы с памятью: её выделение, освобождение и обращение к ней.
Изначально программа выдавала множество ошибок по памяти, и не выдавала правильного решения. Функции тщательно отлаживались, а часто переписывались сначала, это привело к тому, что память перестала утекать, а программа начала выдавать правильные ответы.

```
[alext@alext-pc solution]$ valgrind --leak-check=full ./solution <test1k.txt
>/dev/null
==7598== Memcheck,a memory error detector
==7598== Copyright (C) 2002-2017,and GNU GPL'd,by Julian Seward et al.
==7598== Using Valgrind-3.17.0 and LibVEX; rerun with -h for copyright info
==7598== Command: ./solution
==7598==
==7598==
==7598== HEAP SUMMARY:
==7598==     in use at exit: 0 bytes in 0 blocks
==7598==   total heap usage: 3,497 allocs,3,497 frees,346,392 bytes allocated
==7598==
==7598== All heap blocks were freed --no leaks are possible
==7598==
==7598== For lists of detected and suppressed errors,rerun with: -s
==7598== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

2 GPROF

Согласно [2], gprof производит профиль исполнения программ на C, Pascal и Fortran77. Эта утилита используется для измерения времени работы отдельных функций программы и общего времени работы.
Профайлер показывает, сколько раз выполнялась конкретная функция и какое время от общего времени работы программы она заняла.
Тестирование проводилось на тесте из 1000 строк с запросами на добавление, поиск и удаление.

```
[alext@alext-pc solution]$ gprof ./solution -p ./gmon.out
Flat profile:

Each sample counts as 0.01 seconds.
%   cumulative   self              self     total
time   seconds   seconds    calls  us/call  us/call  name
50.02      0.01     0.01   538093     0.02     0.02  std::_Sp_counted_base
<(__gnu_cxx::_Lock_policy)2>::_M_add_ref_copy()
50.02      0.02     0.01    78715     0.13     0.13  operator==
(TString const&,TString const&)
0.00      0.02     0.00   554500     0.00     0.00  std::_Sp_counted_base<
(__gnu_cxx::_Lock_policy)2>::_M_release()
0.00      0.02     0.00   366253     0.00     0.00  std::__shared_ptr<TVectorNode
<TString>,(__gnu_cxx::_Lock_policy)2>::get() const
[...]
```

[Дальнейший лог было решено не приводить,поскольку он не несёт какой-либо особенно ценной информации и в то же время является слишком перегруженным для того,чтобы его было комфортно читать.]

Видно, что большую часть времени занимают операции работы с указателями и реализация оператора сравнения для строк.

3 Дневник отладки

В общем на работу над лабораторной №2 ушло более 3-х недель, дневник отладки я не вёл, так как отлаживал при помощи valgrind ещё до того, как узнал про эту лабораторную.
В процессе отладки были выявлены и устранены (возможно частично) следующие ошибки:

- Утечки памяти при malloc() и realloc(). Решена реализацией структуры `Vector` и использованием её вместо массива в узле дерева.
- Нарушение структуры «родитель-ребёнок», когда в ребёнке оставался указатель не на его родителя, а на «дядю» (узел-брат для родителя), после разбиения вершины. Решена исправлением соответсвующих функций.
- Потеря части узлов при удалении лишь одного. Была *частично* решена корректировкой соответсвующих функций, но тем не менее, некоторые узлы по-прежнему теряются, хоть и в гораздо меньших масштабах, чем изначально.

4 Выводы

Выполнив третью лабораторную работу по курсу «Дискретный анализ», я изучил утилиты `gprof` и `valgrind`. Про `valgrind` я знал и ранее, и это действительно полезное средство для диагностики ошибок работы с памятью. Про `gprof` я узнал впервые, и был впечатлён. Я думаю, эта утилита очень полезна в написании эффективных программ.

Список литературы

[1] *man valgrind(1): a suit of tools for debugging and profiling programs.*
URL: `http://manpages.org/valgrind` (дата обращения: 20.12.2020).

[2] *man gprof(1): display call graph profile data.*
URL: `http://manpages.org/gprof` (дата обращения: 20.12.2020).