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: М. Ю. Курносов
Преподаватель: А. А. Кухтичев
Группа: М8О-208Б

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).

Пришлите отчёт в формате pdf.

1 Valgrind

Valgrind - это программа для поиска утечек памяти, ошибок обращения с ней и профилирования. Для поиска ошибки я использую ключ -g при компиляции программы для того чтобы valgrind мог указать точное место в коде, где работа с памятью осуществляется неверно.
Для начала просто проверим программу из прошлой лабораторной работы на утечки памяти и другие ошибки. Запускаем программу valgrind с флагом -leak-check=full, который включает поиск утечек памяти с подробным описанием:

vorona@DESKTOP:/mnt/c/Users/Max Kar/Desktop/МАИ/ДА/lab2$ valgrind –leak-check=full ./p < test

==730== Memcheck, a memory error detector

==730== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.

==730== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info

==730== Command: ./p

==730==
OK
OK
OK
OK: 11
OK: 8
OK: 7
OK
==730==
==730== HEAP SUMMARY:
==730== in use at exit: 0 bytes in 0 blocks
==730== total heap usage: 10 allocs, 10 frees, 10,579 bytes allocated
==730==
==730== All heap blocks were freed – no leaks are possible
==730==
==730== For lists of detected and suppressed errors, rerun with: -s
==730== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)

Как видно в данном случае утечек памяти не обнаружено, о чём говорит "All heap blocks were freed – no leaks are possible"

Для более подробного разбора ошибок следует использовать флаг –show-leak-kinds=all:

vorona@DESKTOP:/mnt/c/Users/Max Kar/Desktop/МАИ/ДА/lab2$ valgrind –leak-check=full –show-leak-kinds=all ./p < test
==777== Memcheck, a memory error detector
==777== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==777== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==777== Command: ./p
==777==
OK
OK
OK

OK: 11
OK: 8
OK: 7
OK
==777==
==777== HEAP SUMMARY:
==777== in use at exit: 120 bytes in 3 blocks
==777== total heap usage: 10 allocs, 7 frees, 10,579 bytes allocated
==777==
==777== 40 bytes in 1 blocks are indirectly lost in loss record 1 of 3
==777== at 0x483B7F3: malloc (in /usr/lib/x86$_6$4−*linux*−*gnu*/*valgrind*/*vgpreload*$_m$*emcheck*−*amd*64 − *linux.so*)
==777== by 0x10952A: ConstructTNode (solution.c:70)
==777== by 0x1096FE: RInsert (solution.c:123)
==777== by 0x1098F9: TInsert (solution.c:168)
==777== by 0x10A924: main (solution.c:542)
==777==
==777== 40 bytes in 1 blocks are indirectly lost in loss record 2 of 3
==777== at 0x483B7F3: malloc (in /usr/lib/x86$_6$4−*linux*−*gnu*/*valgrind*/*vgpreload*$_m$*emcheck*−*amd*64 − *linux.so*)
==777== by 0x10952A: ConstructTNode (solution.c:70)
==777== by 0x1096FE: RInsert (solution.c:123)
==777== by 0x109796: RInsert (solution.c:146)
==777== by 0x109796: RInsert (solution.c:146)
==777== by 0x1098F9: TInsert (solution.c:168)
==777== by 0x10A924: main (solution.c:542)
==777==
==777== 120 (40 direct, 80 indirect) bytes in 1 blocks are definitely lost in loss record 3 of 3
==777== at 0x483B7F3: malloc (in /usr/lib/x86$_6$4−*linux*−*gnu*/*valgrind*/*vgpreload*$_m$*emcheck*−*amd*64 − *linux.so*)
==777== by 0x10952A: ConstructTNode (solution.c:70)
==777== by 0x1096FE: RInsert (solution.c:123)
==777== by 0x109796: RInsert (solution.c:146)
==777== by 0x1098F9: TInsert (solution.c:168)

==777== by 0x10A924: main (solution.c:542)
==777==
==777== LEAK SUMMARY:
==777== definitely lost: 40 bytes in 1 blocks
==777== indirectly lost: 80 bytes in 2 blocks
==777== possibly lost: 0 bytes in 0 blocks
==777== still reachable: 0 bytes in 0 blocks
==777== suppressed: 0 bytes in 0 blocks
==777==
==777== For lists of detected and suppressed errors, rerun with: -s
==777== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)

Как видно из вывода программы проблемы с памятью начались из-за строки 70, в файле solution.c в функции ConstructTNode, эта функция выделяет память для создания новых узлов в дереве, разобравшись можно понять что память перестала освобождаться.

2 Gprof

Gprof-это инструмент анализа производительности для приложений Unix. Он использовал гибрид инструментария и выборки[1] и был создан как расширенная версия более старого инструмента "prof". В отличие от prof, gprof способен к ограниченному сбору и печати графиков вызовов.

Профилировщик показывает сколько процентов от общего времени работы программы занимает и сколько раз вызывается каждая функция и другие данные.

```
{\obeylines\textbf{vorona@DESKTOP:/mnt/c/Users/Max Kar/Desktop/МАИ/ДА/lab2\$
gprof ./p gmon.out >prof.txt
vorona@DESKTOP:/mnt/c/Users/Max Kar/Desktop/МАИ/ДА/lab2\$ cat prof.txt
Flat profile:

Each sample counts as 0.01 seconds.
%   cumulative   self              self     total
time   seconds   seconds    calls  Ts/call  Ts/call  name
100.17      0.04     0.04                             main
0.00      0.04     0.00   100000     0.00     0.00  TFind
0.00      0.04     0.00     1259     0.00     0.00  IsR
0.00      0.04     0.00      100     0.00     0.00  ConstructTNode
```

```
0.00      0.04     0.00      100     0.00     0.00  Dig
0.00      0.04     0.00      100     0.00     0.00  RInsert
0.00      0.04     0.00      100     0.00     0.00  TInsert
0.00      0.04     0.00       58     0.00     0.00  Rotate1
0.00      0.04     0.00       17     0.00     0.00  Rotate2
0.00      0.04     0.00        1     0.00     0.00  DelTree

%         the percentage of the total running time of the
time       program used by this function.

cumulative a running sum of the number of seconds accounted
seconds   for by this function and those listed above it.

self      the number of seconds accounted for by this
seconds    function alone.  This is the major sort for this
listing.

calls      the number of times this function was invoked,if
this function is profiled,else blank.

self      the average number of milliseconds spent in this
ms/call    function per call,if this function is profiled,
else blank.

total     the average number of milliseconds spent in this
ms/call    function and its descendents per call,if this
function is profiled,else blank.

name       the name of the function.  This is the minor sort
for this listing. The index shows the location of
the function in the gprof listing. If the index is
in parenthesis it shows where it would appear in
the gprof listing if it were to be printed.


Copyright (C) 2012-2020 Free Software Foundation,Inc.

Copying and distribution of this file,with or without modification,
are permitted in any medium without royalty provided the copyright
notice and this notice are preserved.
```

```
Call graph (explanation follows)


granularity: each sample hit covers 2 byte(s) for 24.96% of 0.04 seconds

index % time    self  children    called     name
<spontaneous>
[1]    100.0    0.04    0.00                 main [1]
0.00    0.00  100000/100000      TFind [2]
0.00    0.00     100/100         Dig [5]
0.00    0.00     100/100         TInsert [7]
0.00    0.00       1/1           DelTree [10]
-----------------------------------------------
667225              TFind [2]
0.00    0.00  100000/100000      main [1]
[2]      0.0    0.00    0.00  100000+667225  TFind [2]
667225              TFind [2]
-----------------------------------------------
0.00    0.00    1259/1259        RInsert [6]
[3]      0.0    0.00    0.00    1259         IsR [3]
-----------------------------------------------
0.00    0.00     100/100         RInsert [6]
[4]      0.0    0.00    0.00     100         ConstructTNode [4]
-----------------------------------------------
0.00    0.00     100/100         main [1]
[5]      0.0    0.00    0.00     100         Dig [5]
-----------------------------------------------
544             RInsert [6]
0.00    0.00     100/100         TInsert [7]
[6]      0.0    0.00    0.00     100+544     RInsert [6]
0.00    0.00    1259/1259        IsR [3]
0.00    0.00     100/100         ConstructTNode [4]
0.00    0.00      24/58          Rotate1 [8]
0.00    0.00      17/17          Rotate2 [9]
544             RInsert [6]
-----------------------------------------------
0.00    0.00     100/100         main [1]
[7]      0.0    0.00    0.00     100         TInsert [7]
0.00    0.00     100/100         RInsert [6]
-----------------------------------------------
```

```
0.00    0.00      24/58          RInsert [6]
0.00    0.00      34/58          Rotate2 [9]
[8]      0.0    0.00    0.00      58         Rotate1 [8]
-----------------------------------------------
0.00    0.00      17/17          RInsert [6]
[9]      0.0    0.00    0.00      17         Rotate2 [9]
0.00    0.00      34/58          Rotate1 [8]
-----------------------------------------------
200             DelTree [10]
0.00    0.00       1/1           main [1]
[10]     0.0    0.00    0.00       1+200     DelTree [10]
200             DelTree [10]
-----------------------------------------------
```

This table describes the call tree of the program,and was sorted by
the total amount of time spent in each function and its children.

Each entry in this table consists of several lines. The line with the
index number at the left hand margin lists the current function.
The lines above it list the functions that called this function,
and the lines below it list the functions this one called.
This line lists:
index A unique number given to each element of the table.
Index numbers are sorted numerically.
The index number is printed next to every function name so
it is easier to look up where the function is in the table.

% time This is the percentage of the 'total'time that was spent
in this function and its children. Note that due to
different viewpoints,functions excluded by options,etc,
these numbers will NOT add up to 100%.

self This is the total amount of time spent in this function.

children This is the total amount of time propagated into this
function by its children.

called This is the number of times the function was called.
If the function called itself recursively,the number
only includes non-recursive calls,and is followed by
a '+'and the number of recursive calls.

name The name of the current function. The index number is printed after it. If the function is a member of a cycle,the cycle number is printed between the function's name and the index number.

For the function's parents,the fields have the following meanings:

self This is the amount of time that was propagated directly from the function into this parent.

children This is the amount of time that was propagated from the function's children into this parent.

called This is the number of times this parent called the function '/'the total number of times the function was called. Recursive calls to the function are not included in the number after the '/'.

name This is the name of the parent. The parent's index number is printed after it. If the parent is a member of a cycle,the cycle number is printed between the name and the index number.

If the parents of the function cannot be determined,the word '<spontaneous>'is printed in the 'name'field,and all the other fields are blank.

For the function's children,the fields have the following meanings:

self This is the amount of time that was propagated directly from the child into the function.

children This is the amount of time that was propagated from the child's children to the function.

called This is the number of times the function called this child '/'the total number of times the child was called. Recursive calls by the child are not listed in the number after the '/'.

```
name       This is the name of the child.  The child's index
number is printed after it.  If the child is a
member of a cycle,the cycle number is printed
between the name and the index number.

If there are any cycles (circles) in the call graph,there is an
entry for the cycle-as-a-whole.  This entry shows who called the
cycle (as parents) and the members of the cycle (as children.)
The '+'recursive calls entry shows the number of function calls that
were internal to the cycle,and the calls entry for each member shows,
for that member,how many times it was called from other members of
the cycle.


Copyright (C) 2012-2020 Free Software Foundation,Inc.

Copying and distribution of this file,with or without modification,
are permitted in any medium without royalty provided the copyright
notice and this notice are preserved.


Index by function name

[4] ConstructTNode          [6] RInsert                 [7] TInsert
[10] DelTree                 [8] Rotate1                 [1] main
[5] Dig                     [9] Rotate2
[3] IsR                     [2] TFind
vorona@DESKTOP:/mnt/c/Users/Max Kar/Desktop/МАИ/ДА/lab2\$}}
```

Данное исследование времени работы функций в программе было проведено на тесте из 100100 строк, как видно из результата профилирования основное время работы занимает функция main, что логично, так как все взаимодействия с пользователем проходят именно через неё и функции так же запускются из неё.

3 Дневник отладки

1. 19.12.20 Воспользовался утилитой Valgrind для поиска утечек при тестировании программы
2. 20.12.20 Успешно нашёл и устранил неполадку
3. 20.12.20 Изучил основы работы gprof
4. 20.12.20 Отпрофилировал программу с помощью gprof

4 Выводы

В результате выполнения лабораторной работы по курсу «Дискретный анализ», я научился работать с утилитой gprof и закрепил свои навыки в использовании valgrind, а так же разобрался в поиске утечек по информации о строках в которых они произошли при помощи valgrind и компиляции с -g.

Список литературы

[1] *Как установить и использовать профилировщик Gprof в Linux*
URL:http://rus-linux.net/MyLDP/algol/install-and-use-gprof.html
(дата обращения: 19.12.2020).

[2] *Использование Valgrind для поиска утечек и недопустимого использования памяти*
URL: http://cppstudio.com/post/4348/ (дата обращения: 19.12.2020).