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: В. С. Епанешников
Преподаватель: А. А. Кухтичев
Группа: М8О-206Б-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 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 известные утилиты (например, Valgrind или Shark) или добавлять к ним новые (например, gcov)

Вариант дерева: AVl-дерево.

1 Описание

Valgrind. – инструмент для поиска ошибок работы с памятью. Но кроме этого, он также содержит ряд дополнительных утилит для профилирования производительности, поиска ошибок синхронизации в многопоточных программах и анализа потребления памяти. Существуют много разных ошибок, которые можно обнаружить в отчёте valgrind'a. Обычно встречаются следующие ошибки:

Valgrind сообщает о двух типах проблем: об ошибках памяти и утечках памяти. Когда программа динамически выделяет память и забывает позже освободить ее, возникает утечка. Утечка памяти, как правило, не приводит к неправильному поведению программы, сбоям или неправильным ответам. Ошибка памяти - это красное предупреждение. Чтение неинициализированной памяти, запись после конца участка памяти, доступ к освобожденной памяти и другие ошибки памяти могут иметь серьезные последствия, и ошибки необходимо в срочном порядке исправлять. gprof.

Профилирование — это сбор характеристик программы во время ее выполнения. При профилировании замеряется время выполнения и количество вызовов отдельных функций и строк в коде программы. При помощи этого инструмента программист может найти наиболее медленные участки кода и провести их оптимизацию. В качестве профилировщика изспользовал утилиту **gprof**, которая выводит подробный отчёт о результате работы программы.

gcov. Не всегда строчки кода, которые мы пишем, будут выполняться. Чтобы проверить, работает ли функция при работе программы и какая часть кода работает, необходимо код покрыть тестами. Покрытие покажет, какие строчки кода выполнялись при работе программы. После покрытия программист оптимизирует код, исправляя ошибки в невыполненных строчках кода или удаляя их. Буду использовать в качестве инструмента профилирования утилиту gcov.

2 Консоль

Рассмотрим использование утилиты valgrind. Чтобы воспользоваться данной утилитой предварительно необходимо скомпилировать программа с ключом -g. Затем запускаю её с ключами –leak-check=full (для вывода подробной информации о кажлой найденной ошибке) и -show-leak-kinds=all (для возможности просмотра всех найденных ошибок).

```
==13002== Memcheck,a memory error detector
==13002== Copyright (C) 2002-2017,and GNU GPL'd,by Julian Seward et al.
==13002== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==13002== Command: ./solution
==13002==
==13002==
==13002== HEAP SUMMARY:
==13002==     in use at exit: 122,880 bytes in 6 blocks
==13002==   total heap usage: 781,257 allocs,781,251 frees,231,445,584 bytes
allocated
==13002==
==13002== LEAK SUMMARY:
==13002==    definitely lost: 0 bytes in 0 blocks
==13002==    indirectly lost: 0 bytes in 0 blocks
==13002==      possibly lost: 0 bytes in 0 blocks
==13002==    still reachable: 122,880 bytes in 6 blocks
==13002==         suppressed: 0 bytes in 0 blocks
==13002== Rerun with --leak-check=full to see details of leaked memory
==13002==
==13002== For lists of detected and suppressed errors,rerun with: -s
==13002== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Valgrind показал, что в моей программе проблем и ошибкок при работе с памятью не обнаружилось. При написании кода возникали ошибки, связанные с чтением данных из области памяти, указатель на которую был потерян. С помощью отладки программы все возникшие ошибки были исправлены. Сейчас согласно протоколу, в моей программе нет никаких утечек памяти, т.е. программа является безопасной.

Используя инструмент для профилирования gprof прогоню свою программу на 400 тысяч тестах, предварительно скомпилировав с ключом -pg. Профилировщик соберёт необходимую информацию в файл gmon.out и на её основе сформирует отчёт.

```
Flat profile:

Each sample counts as 0.01 seconds.
%   cumulative   self              self     total
time   seconds   seconds    calls  ns/call  ns/call  name
37.58      0.03     0.03   781250    38.48    43.48  TAvlTree::SubInsert(TAvlNode*,cha
long)
25.05      0.05     0.02   781250    25.66    29.77  TAvlTree::SubRemove(TAvlNode*,cha
const*)
12.53      0.06     0.01  5123049     1.96     1.96  TAvlTree::Balance(TAvlNode*)
12.53      0.07     0.01                             TAvlTree::Insert(char*,unsigned
long)
12.53      0.08     0.01                             TAvlTree::Find(char const*)
const
0.00      0.08     0.00   461444     0.00     0.00  TAvlTree::LeftRotate(TAvlNode*)
0.00      0.08     0.00   387290     0.00     1.94  TAvlTree::RemoveMin(TAvlNode*,TAvl
0.00      0.08     0.00   380310     0.00     0.00  TAvlTree::RightRotate(TAvlNode*)
0.00      0.08     0.00   190276     0.00     0.00  TAvlTree::DoubleRightRotate(TAvlNo
0.00      0.08     0.00    90195     0.00     0.00  TAvlTree::DoubleLeftRotate(TAvlNod
0.00      0.08     0.00        1     0.00     0.00  _GLOBAL__sub_I__Z9lower_strPc
0.00      0.08     0.00        1     0.00     0.00  _GLOBAL__sub_I__ZN8TAvlNodeC2Ev
```

По результатам профилирования видно, что 37% времени уходит на вставку узла в дерево, 25% времени занимает удаление узла из дерева и по 12% времени отбирают функции поиска узла в дереве, балансировка.

3 Дневник отладки

- Исследование работы, написанной программы на ошибки при работе с памятью с помощью утилиты valgrind. Утечек памяти не обнаружено.
- Профилирование программы утилитой gprof. Установлено, что 37% времени уходит на вставку узла в дерево, 25% времени занимает удаление узла из дерева.

4 Выводы

Выполнив вторую лабораторную работу по дискретному анализу, я научился пользоваться несомненно важными и полезными утилитами для тестирования и оптимизации программы valgrind, gprov. Данных утилит достаточно для получения важной и объёмной информации о программе, которая показывает, какие есть недочёты и ошибки в коде для будущего исправления и оптимизации. Valgrind поможет с проблемами, связанными с работой с памятью, которые являются частой причиной некорректной работы программы. Утилита gprof может послужить программисту простым и удобным профилировщиком, которая выведет информацию о времени работы каждой функции, а утилита gcov отлично подойдёт для покрытия кода тестами и вывода получившего результата.

Список литературы

[1] *Информация о valgrind.*
URL: `http://alexott.net/en/writings/prog-checking/Valgrind.html`

[2] *Информация о gprof.*
URL: `https://www.opennet.ru/docs/RUS/gprof/`

[3] *Информация о gcov.*
URL: `https://gcc.gnu.org/onlinedocs/gcc/Gcov.html`