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: С. Я Симонов
Преподаватель: А. А. Кухтичев
Группа: М8О-206Б

Дата:
Оценка:
Подпись:

Москва, 2020

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: gprof, valgrind.

1 Описание

Профилирование позволяет увидеть поведение программы во время её выполнения: изучить стек вызовов функций, время выполнения каждой. Это позволяет определять, как долго и как часто выполняются те или иные части программы, чтобы идентифицировать неэффективные части программы с целью оптимизации кода.

Отладка использования памяти позволяет обнаруживать утечки памяти и другие ошибки, связянные с использованием памяти: выход за границы массива, отсутствие деаллокации памяти после завершения работы программы и так далее.

2 Дневник выполнения работы

1. Ознакомился с gprof на практических примерах, после чего нашёл ключевые части программы, нуждающиеся в оптимизации.

2. Произвёл тестирование с помощью Valgrind. Не обнаружил утечек памяти и других ошибок, после чего искусственно создал две ошибки и протестировал программу на этом коде.

3 gprof

Данная утилита позволяет увидеть профильную статистику, которая накапливается программой во время её работы.

Результаты работы утилиты gprof с программой, реализующей работу AVL-дерева:

```
Flat profile:

Each sample counts as 0.01 seconds.
 no time accumulated
```

% time	cumulative seconds	self seconds	calls	self Ts/call	total Ts/call	name
0.00	0.00	0.00	18	0.00	0.00	AVLTree::GetHeight(AVLTree::Node*)
0.00	0.00	0.00	16	0.00	0.00	ToLower(char)
0.00	0.00	0.00	13	0.00	0.00	IsLetter(char)
0.00	0.00	0.00	7	0.00	0.00	GetKey(char*)
0.00	0.00	0.00	7	0.00	0.00	AVLTree::Search(Data&)
0.00	0.00	0.00	6	0.00	0.00	AVLTree::BalFact(AVLTree::Node*)
0.00	0.00	0.00	3	0.00	0.00	GetVal(unsigned long long&)
0.00	0.00	0.00	3	0.00	0.00	AVLTree::Balance(AVLTree::Node*)
0.00	0.00	0.00	3	0.00	0.00	AVLTree::SearchSon(AVLTree::Node*, Data&)
0.00	0.00	0.00	3	0.00	0.00	AVLTree::SetHeight(AVLTree::Node*)
0.00	0.00	0.00	2	0.00	0.00	AVLTree::CreateNode(Data)
0.00	0.00	0.00	2	0.00	0.00	AVLTree::Insert(Data)
0.00	0.00	0.00	1	0.00	0.00	_GLOBAL__sub_I_NODE
0.00	0.00	0.00	1	0.00	0.00	__static_initialization_and_destruction_0(int, int)
0.00	0.00	0.00	1	0.00	0.00	AVLTree::DeleteTree(AVLTree::Node*&)
0.00	0.00	0.00	1	0.00	0.00	AVLTree::DeleteTree()
0.00	0.00	0.00	1	0.00	0.00	AVLTree::Min(AVLTree::Node*)
0.00	0.00	0.00	1	0.00	0.00	AVLTree::Delete(Data)
0.00	0.00	0.00	1	0.00	0.00	AVLTree::DeleteMin(AVLTree::Node*)
0.00	0.00	0.00	1	0.00	0.00	AVLTree::InsertSon(AVLTree::Node*, Data)
0.00	0.00	0.00	1	0.00	0.00	AVLTree::AVLTree()
0.00	0.00	0.00	1	0.00	0.00	AVLTree::~AVLTree()

Из полученных данных можно сделать вывод, что обработка команд для осуществления преобразования дерева и балансировки занимает большую часть времени, поэтому важно уделить время оптимизации именно этой части программы.

4 Valgrind

Valgrind - утилита для профилирования и отладки использования памяти. Она идентифицирует ошибки, связанные с памятью, а также позволяет обнаружить переполенние стека. Vagrind анализирует работу программы, выполняя её на эмуляторе, проводит анализ процессов и выводит статистику. Время работы программы с Valgrind в среднем в 4 раза дольше, чем выполнение программы без использования дополнительных утилит.

Пример работы Valgrind с использованием флага −−leak−check=full для предоставления полный информации, связанной с утечками памяти (вывод имён функций и строк кода, работа которых повлекла за собой ошибки):

```
 sergey@sergey-RedmiBook-14:~/labs/DA/lab2$  valgrind --leak-check=full ./a.out < test.txt
==5733== Memcheck, a memory error detector
==5733== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==5733== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==5733== Command: ./a.out
==5733==
OK
Exist
OK
OK: 18446744073709551615
OK: 1
OK
NoSuchWord
OK
OK
OK
NoSuchWord
NoSuchWord
OK
NoSuchWord
OK
OK
OK
NoSuchWord
==5733==
==5733== HEAP SUMMARY:
==5733==     in use at exit: 0 bytes in 0 blocks
==5733==   total heap usage: 21 allocs, 21 frees, 78,160 bytes allocated
==5733==
==5733== All heap blocks were freed -- no leaks are possible
==5733==
==5733== For counts of detected and suppressed errors, rerun with: -v
==5733== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

В программе не обнаружено ошибок, связанных с памятью. Искусственно создадим пару ошибок и запустим Valgrind ещё раз.

```
 sergey@sergey-RedmiBook-14:~/labs/DA/lab2$  valgrind --leak-check=full ./a.out < test.txt
==5870== Memcheck, a memory error detector
==5870== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==5870== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==5870== Command: ./a.out
```

```
==5870==
OK
Exist
OK
OK: 18446744073709551615
OK: 18446744073709551615
OK
NoSuchWord
OK
OK
OK
NoSuchWord
NoSuchWord
OK
NoSuchWord
OK
OK
OK
NoSuchWord
==5870==
==5870== HEAP SUMMARY:
==5870==     in use at exit: 224 bytes in 2 blocks
==5870==   total heap usage: 21 allocs, 19 frees, 78,160 bytes allocated
==5870==
==5870== 80 bytes in 1 blocks are definitely lost in loss record 1 of 2
==5870==    at 0x4C2FB0F: malloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==5870==    by 0x1094F8: TNode::TNode(unsigned long long) (main.cpp:175)
==5870==    by 0x109194: TBTree::TBTree(unsigned long long) (main.cpp:69)
==5870==    by 0x10B4D5: main (main.cpp:742)
==5870==
==5870== 144 bytes in 1 blocks are definitely lost in loss record 2 of 2
==5870==    at 0x4C2FB0F: malloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==5870==    by 0x1094DD: TNode::TNode(unsigned long long) (main.cpp:174)
==5870==    by 0x109194: TBTree::TBTree(unsigned long long) (main.cpp:69)
==5870==    by 0x10B4D5: main (main.cpp:742)
==5870==
==5870== LEAK SUMMARY:
==5870==    definitely lost: 224 bytes in 2 blocks
==5870==    indirectly lost: 0 bytes in 0 blocks
==5870==      possibly lost: 0 bytes in 0 blocks
==5870==    still reachable: 0 bytes in 0 blocks
==5870==         suppressed: 0 bytes in 0 blocks
==5870==
==5870== For counts of detected and suppressed errors, rerun with: -v
==5870== ERROR SUMMARY: 2 errors from 2 contexts (suppressed: 0 from 0)
```

В ходе работы программы Valgrind нашел область памяти, на которую нет указателей, то есть программист не освободил память при выходе из программы.

5 Выводы

Выполнив вторую лабораторную работу по курсу «Дискретный анализ», я поработал с утилитами Valgrind и gprof. Каждая из них помогает отлаживать и оптимизировать программы. Valgrind помогает бороться с утечками памяти и ошибками во время исполнения программы. Исходя из собственного опыта, я могу сказать, что пользуюсь Valgrind очень часто. Утилита gprof анализирует время работы и количество вызовов функций. Помогает в оптимизации работы программы путём выделения самых существенных их частей.

Список литературы

[1] *Профилирование программ в среде UNIX*
URL: `http://www.opennet.ru/docs/RUS/gprof/gprof-4.html` (дата обращения: 10.02.2020)

[2] *Valgrind*
URL: `http://valgrind.org/docs/manual` (дата обращения: 10.02.2020).