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: Д. С. Ляшун
Преподаватель: А. А. Кухтичев
Группа: М8О-207Б

Дата:
Оценка:
Подпись:

Москва, 2021

Лабораторная работа №3

Задача: провести диагностику программы на правильное выделение и освобождение динамической памяти, а также её профилирование, измерение частоты вызова и скорости выполнения отдельных функций, воспользовавшись соответствующими утилитами.

Рассматриваемая программа: реализация Б-дерева.

Используемые утилиты: Valgrind, Gprof.

1 Valgrind

Как сказано в [1], Valgrind — инструментальное программное обеспечение, предназначенное для отладки использования памяти, обнаружения утечек памяти, а также профилирования. В поставку Valgrind входят следующие модули-анализаторы [2]:

1. Memcheck – основной модуль, обеспечивающий обнаружение утечек памяти, и прочих ошибок, связанных с неправильной работой с областями памяти – чтением или записью за пределами выделенных регионов и т.п.

2. Cachegrind – анализирует выполнение кода, собирая данные о непопаданиях в кэш, и точках перехода (когда процессор неправильно предсказывает ветвление). Эта статистика собирается для всей программы, отдельных функций и строк кода

3. Callgrind – анализирует вызовы функций, используя примерно ту же методику, что и модуль cachegrind. Позволяет построить дерево вызовов функций, и соответственно, проанализировать узкие места в работе программы.

4. Massif – позволяет проанализировать выделение памяти различными частями программы

5. Helgrind – анализирует выполняемый код на наличие различных ошибок синхронизации, при использовании многопоточного кода, использующего POSIX Threads.

Пример использования: в ходе отладки программы работы со словарем возникла проблема в утечке памяти, поскольку Valgrind показывал следующее:

```
dmitry@dmitry-VirtualBox:~/DA_labs/lab2/lab3$ valgrind
--tool=memcheck --leak-check=full ./solution <test.txt >a.txt
==3117== Memcheck,a memory error detector
==3117== Copyright (C) 2002-2017,and GNU GPL'd,by Julian Seward et al.
==3117== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==3117== Command: ./solution
==3117==
==3117==
==3117== HEAP SUMMARY:
==3117==     in use at exit: 528 bytes in 2 blocks
==3117==   total heap usage: 1,048 allocs,1,046 frees,185,080 bytes allocated
==3117==
==3117== 528 bytes in 2 blocks are definitely lost in loss record 1 of 1
==3117==    at 0x483BE63: operator new(unsigned long)
```

```
(in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_memcheck-amd64-linux.so)
==3117==    by 0x109A09: SplitChild(TNode*,int const&)
(in /home/dmitry/DA_labs/lab2/lab3/solution)
==3117==    by 0x10A04F: Insert(TNode*&,char*,unsigned long long const&)
(in /home/dmitry/DA_labs/lab2/lab3/solution)
==3117==    by 0x109456: main
(in /home/dmitry/DA_labs/lab2/lab3/solution)
==3117==
==3117== LEAK SUMMARY:
==3117==    definitely lost: 528 bytes in 2 blocks
==3117==    indirectly lost: 0 bytes in 0 blocks
==3117==      possibly lost: 0 bytes in 0 blocks
==3117==    still reachable: 0 bytes in 0 blocks
==3117==         suppressed: 0 bytes in 0 blocks
==3117==
==3117== For lists of detected and suppressed errors,rerun with: -s
==3117== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
```

Как видно, количество выделений памяти (allocs) и её освобождений (frees) не совпадает, что указывает на проблему. При вызове утилиты с параметрами –leak-check=full выводится подробная информации о месте выделения памяти, которая не была затем освобождена. В данном случае это функция SplitChild, она выводится в стеке вызовов функций.

Спустя время отладки я выяснил, что данная проблема возникала в связи с тем, что в Б-дереве при слиянии вершин одну из них программа удаляла только из массива указателей, при этом не освобождая то, на что он указывает, из-за этого и происходила утечка памяти.

После исправления программы я запустил Valgrind снова:

```
dmitry@dmitry-VirtualBox:~/DA_labs/lab3$ valgrind --tool=memcheck
--leak-check=full ./solution <test.txt >a.txt
==3253== Memcheck,a memory error detector
==3253== Copyright (C) 2002-2017,and GNU GPL'd,by Julian Seward et al.
==3253== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==3253== Command: ./solution
==3253==
==3253==
==3253== HEAP SUMMARY:
==3253==     in use at exit: 0 bytes in 0 blocks
==3253==   total heap usage: 1,048 allocs,1,048 frees,185,080 bytes allocated
==3253==
```

```
==3253== All heap blocks were freed --no leaks are possible
==3253==
==3253== For lists of detected and suppressed errors,rerun with: -s
==3253== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Теперь утилита показывает, что количество выделений и освобождений динамической памяти совпадает, и программа выполняется правильно.

2 Gprof

Как указано в [3], Gprof – GNU профилятор для диагностики времени работы программ в Unix. Профилирование позволяет изучить, где программа расходует своё время и какие функции вызывали другие функции, пока программа исполнялась. Эта информация может указать на ту часть программы, которая исполняется медленнее, чем ожидалось, и которая может быть кандидатом на переписывание, чтобы ускорить выполнение программы. Эта информация также подскажет, какие функции вызывались чаще или реже, чем предполагалось, что может помочь отметить ошибки, которые иначе остались бы незамеченными.

Для работы с Gprof сперва необходимо скомпилировать исходный код программы с ключом -pg, тем самым разрешив её профилирование. Далее необходимо выполнить обычный вызов программы, после которого будет создан файл gmon.out, использующийся для профилирования. Далее вызывается сама утилита Gprof с указанием имени исполняемого файла в числе аргументов, а также файла gmon.out. Ниже приведён пример диагностики работы программы для теста из 10^5 запросов:

```
dmitry@dmitry-VirtualBox:~/DA_labs/lab3$ g++ -pg main.cpp -o main
dmitry@dmitry-VirtualBox:~/DA_labs/lab3$ ./main <test.txt >a.txt
dmitry@dmitry-VirtualBox:~/DA_labs/lab3$ gprof ./main -p -b ./gmon.out
Flat profile:
Each sample counts as 0.01 seconds.
%  cumulative self         self     total
time   sec   sec   calls   us/call us/call      name
45.51  0.10  0.10 1615738  0.06  0.06  StrCompare(char*,char*)
22.76  0.15  0.05  390542  0.13  0.33  BinSearch(TItem**,int const&,char*)
9.10  0.17  0.02    20962   0.96  0.96  TItem::TItem()
9.10  0.19  0.02                             main
4.55  0.20  0.01    33285   0.30  2.11  EraseInRoot(TNode*&,char*)
4.55  0.21  0.01    33285   0.30  1.81  Erase(TNode*,char*)
4.55  0.22  0.01    33190   0.30  1.80  Search(TNode*,char*)
0.00  0.22  0.00   100000   0.00  0.00  Convert(char*)
0.00  0.22  0.00    33525   0.00  2.10  InsertNonFull(TNode*,char*,unsigned
long long const&)
0.00  0.22  0.00    33525   0.00  2.10  Insert(TNode*&,char*,unsigned long
long const&)
0.00  0.22  0.00    33525   0.00  2.10  TBtree::Insert(char*,unsigned long
long const&)
0.00  0.22  0.00    33285   0.00  2.11  TBtree::Erase(char*)
0.00  0.22  0.00    33190   0.00  1.80  TBtree::Search(char*)
0.00  0.22  0.00    26343   0.00  0.00  void Insert<TItem>(TItem**,int&,TItem*,int)
```

```
0.00  0.22  0.00    22124   0.00  0.00  Assign(char*,char*)
0.00  0.22  0.00    20962   0.00  0.00  TItem::~TItem()
0.00  0.22  0.00    16488   0.00  0.00  void Delete<TItem>(TItem**,int&,int)
0.00  0.22  0.00     1955   0.00  0.00  void Insert<TNode>(TNode**,int&,TNode*,int)
0.00  0.22  0.00     1268   0.00  0.00  TNode::TNode()
0.00  0.22  0.00     1254   0.00  0.00  SplitChild(TNode*,int const&)
0.00  0.22  0.00     1047   0.00  0.00  void Delete<TNode>(TNode**,int&,int)
0.00  0.22  0.00      357   0.00  0.00  TNode::~TNode()
0.00  0.22  0.00      346   0.00  0.00  MergeChild(TNode*,int const&,int const&)
0.00  0.22  0.00        1   0.00  0.00  TBtree::TBtree()
0.00  0.22  0.00        1   0.00  0.00  TBtree::~TBtree()
```

Как видно, в результате работы утилиты генерируется таблица, содержащая все вызванные программой функции и информацию о времени работы каждой из них, со следующими полями [3] :

1. '% time' – процент от общего времени исполнения программы, затраченный на выполнение этой функции. Сумма по всем строкам должна составлять 100%.

2. 'cumulative seconds' – общее время в секундах, которое затратил компьютер на выполнение этой функции, плюс время, затраченное на выполнение всех функций, перечисленных выше в этой таблице.

3. 'self seconds' – количество секунд, подсчитанных только для этой функции. Листинг простого профиля сперва упорядочивается по этому количеству.

4. 'calls' – общее количество вызовов этой функции – сколько раз она была вызвана.

5. 'self ms call' – представляет собой среднее количество миллисекунд, затраченных этой функцией на вызов, если эта функция профилируется.

6. 'total ms call' – представляет собой среднее количество миллисекунд, затраченных этой функцией и её подпрограммами на вызов, если эта функция профилируется.

7. 'name' – имя функции. Листинг простого профиля упорядочивается по этому полю в алфавитном порядке после упорядочения по полю 'self seconds'.

Исходя из данных приведённой таблицы можно сделать вывод о том, что самыми часто вызываемыми операциями являются функции работы со строковыми ключами, а основное время работы программы занимают функции сравнения строк и бинарного поиска в массиве элементов. В случае требования ускорения времени работы программы необходимо начать оптимизацию именно с этих функций.

3 Дневник отладки

Дата	Время	Возникшая проблема	Действия по устранению
20.11.2020	15:40	В ходе запуска Valgrind была обнаружена утечка памяти: узел, создаваемый в функции SplitChild не уничтожается.	Исправить работу функции MergeChild, чтобы после слияния вершин в одну происходило освобождение памяти той вершины, из которой перекидывались данные.
20.11.2020	16:20	При запуске Valgrind сообщил об ошибке – записи во внешний файл неинициализированных значений.	Данная проблема была связана с тем, что при записи ключей элементов во внешний файл символьный массив сохранялся целиком, без учета нулевого символа на конце строки. Для исправления необходимо изменить конструктор элемента, чтобы при создании ключа происходила также его предварительная инициализация каким-нибудь значением, например, заполнением нулевыми символами.

4 Выводы

Выполнив третью лабораторную работу по курсу «Дискретный анализ», я познакомился со средствами диагностики и профилирования программ Gprof и Valgrind.

Valgrind помог мне определить, возникают ли утечки памяти в ходе работы моей программы, и в случае этого показать место выделения не уничтожающихся объектов. В свою очередь Gprof вывел мне подробную информацию о времени работы различных функций в программе и количестве их вызовов.

Использовать данные утилиты было довольно удобно, и я уверен, что при выполнении последующих лабораторных работ буду продолжать активно их применять.

Список литературы

[1] *Valgrind - Википедия*
URL: `https://ru.wikipedia.org/wiki/Valgrind` (дата обращения: 23.11.2020).

[2] *Выявление ошибок работы с памятью при помощи valgrind / OpenNET*
URL: `https://www.opennet.ru/base/dev/valgrind_memory.txt.html` (дата обращения: 23.11.2020).

[3] *Профилятор Gprof / OpenNET*
URL: `https://www.opennet.ru/docs/RUS/gprof/gprof.html` (дата обращения: 24.11.2020).