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:
Преподаватель: А. А. Кухтичев
Группа: М8О-206Б
Дата:
Оценка:
Подпись:

Москва, 2020

Лабораторная работа №1

Задача: Необходимо провести исследование скорости выполнения и потребления оперативной памяти. В случае выявления 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).

1 Описание

Результатом лабораторной работы является отчёт, состоящий из:

1. Дневника выполнения работы, в котором отражено что и когда делалось, какие средства использовались и какие результаты были достигнуты на каждом шаге выполнения лабораторной работы.
2. Выводов о найденных недочётах.
3. Сравнение работы исправленной программы с предыдущей версией.
4. Общих выводов о выполнении лабораторной работы, полученном опыте.

2 Дневник выполнения работы

Основные этапы создания программы:

1. Реализация необходимых алгоритмов
2. Выявление логических ошибок в коде программы
3. Выявление утечек памяти
4. Выявление и оптимизация неэффективно работающих участков кода
5. Итоговое тестирование

3 Используемые средства

Visual Studio 2017

Visual Studio предоставляет хорошие возможности для отладки программ. Кроме того, в ней достаточно просто собирать проекты, не нужно вручную прописывать makefile.

Для того, чтобы начать пользоваться отладчиком, необходимо запустить программу в режиме «Debug». Далее можно открыть окно просмотра значений переменных: локальных или видимых или же в окне «Контрольные значения» задать необходимую переменную, также можно посмотреть окно стека вызовов и расставить точки остановки. Для базовой отладки этого более чем достаточно.

Valgrind

Valgrind – это достаточно мощный инструмент для отслеживания утечек памяти (и не только), который удобно использовать при работе с консолью. Для его использования необходимо скомпилировать программу с ключом -g, чтобы компилятор добавил отладочную информацию и вызвать valgrind от исполняемого файла. Эта утилита не только сообщит о наличии утечек, но и покажет, где эта память была выделена. Воспользуемся valgrind'ом при поиске утечек на примере лабораторной 3.

```
maria@maria-MS-7817:~/project/da2X$ valgrind --leak-check=full --show-leak-kinds=all
./a.out<test1
==5366== Memcheck,a memory error detector
==5366== Copyright (C) 2002-2017,and GNU GPL'd,by Julian Seward et al.
==5366== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==5366== Command: ./a.out
==5366==
OK
Exist
OK
OK: 18446744073709551615
OK: 1
OK
NoSuchWord
==5366==
==5366== HEAP SUMMARY:
==5366==     in use at exit: 122,880 bytes in 6 blocks
==5366==   total heap usage: 28 allocs,22 frees,197,009 bytes allocated
==5366==
==5366== 8,192 bytes in 1 blocks are still reachable in loss record 1 of 6
==5366==    at 0x4C3289F: operator new[](unsigned long) (in /usr/lib/valgrind/vgpreloa
==5366==    by 0x4F2DFE7: std::basic_filebuf<char,std::char_traits<char>>::_M_allocate
(in /usr/lib/x86_64-linux-gnu/libstdc++.so.6.0.25)
==5366==    by 0x4F2BDC2: ??? (in /usr/lib/x86_64-linux-gnu/libstdc++.so.6.0.25)
==5366==    by 0x4EE1BE0: std::ios_base::sync_with_stdio(bool) (in /usr/lib/x86_64-lin
==5366==    by 0x1098BA: main (da2.cpp:461)
==5366==
==5366== 8,192 bytes in 1 blocks are still reachable in loss record 2 of 6
==5366==    at 0x4C3289F: operator new[](unsigned long) (in /usr/lib/valgrind/vgpreloa
==5366==    by 0x4F2DFE7: std::basic_filebuf<char,std::char_traits<char>>::_M_allocate
(in /usr/lib/x86_64-linux-gnu/libstdc++.so.6.0.25)
==5366==    by 0x4F2BDC2: ??? (in /usr/lib/x86_64-linux-gnu/libstdc++.so.6.0.25)
==5366==    by 0x4EE1C01: std::ios_base::sync_with_stdio(bool) (in /usr/lib/x86_64-lin
==5366==    by 0x1098BA: main (da2.cpp:461)
```

```
==5366==
==5366== 8,192 bytes in 1 blocks are still reachable in loss record 3 of 6
==5366==    at 0x4C3289F: operator new[](unsigned long) (in /usr/lib/valgrind/vgpreloa
==5366==    by 0x4F2DFE7: std::basic_filebuf<char,std::char_traits<char>>::_M_allocate
(in /usr/lib/x86_64-linux-gnu/libstdc++.so.6.0.25)
==5366==    by 0x4F2BDC2: ??? (in /usr/lib/x86_64-linux-gnu/libstdc++.so.6.0.25)
==5366==    by 0x4EE1C22: std::ios_base::sync_with_stdio(bool) (in /usr/lib/x86_64-lin
==5366==    by 0x1098BA: main (da2.cpp:461)
==5366==
==5366== 32,768 bytes in 1 blocks are still reachable in loss record 4 of 6
==5366==    at 0x4C3289F: operator new[](unsigned long) (in /usr/lib/valgrind/vgpreloa
==5366==    by 0x4F2FDCA: std::basic_filebuf<wchar_t,std::char_traits<wchar_t>>::_M_al
(in /usr/lib/x86_64-linux-gnu/libstdc++.so.6.0.25)
==5366==    by 0x4F2BFA2: ??? (in /usr/lib/x86_64-linux-gnu/libstdc++.so.6.0.25)
==5366==    by 0x4EE1C97: std::ios_base::sync_with_stdio(bool) (in /usr/lib/x86_64-lin
==5366==    by 0x1098BA: main (da2.cpp:461)
==5366==
==5366== 32,768 bytes in 1 blocks are still reachable in loss record 5 of 6
==5366==    at 0x4C3289F: operator new[](unsigned long) (in /usr/lib/valgrind/vgpreloa
==5366==    by 0x4F2FDCA: std::basic_filebuf<wchar_t,std::char_traits<wchar_t>>::_M_al
(in /usr/lib/x86_64-linux-gnu/libstdc++.so.6.0.25)
==5366==    by 0x4F2BFA2: ??? (in /usr/lib/x86_64-linux-gnu/libstdc++.so.6.0.25)
==5366==    by 0x4EE1CB1: std::ios_base::sync_with_stdio(bool) (in /usr/lib/x86_64-lin
==5366==    by 0x1098BA: main (da2.cpp:461)
==5366==
==5366== 32,768 bytes in 1 blocks are still reachable in loss record 6 of 6
==5366==    at 0x4C3289F: operator new[](unsigned long) (in /usr/lib/valgrind/vgpreloa
==5366==    by 0x4F2FDCA: std::basic_filebuf<wchar_t,std::char_traits<wchar_t>>::_M_al
(in /usr/lib/x86_64-linux-gnu/libstdc++.so.6.0.25)
==5366==    by 0x4F2BFA2: ??? (in /usr/lib/x86_64-linux-gnu/libstdc++.so.6.0.25)
==5366==    by 0x4EE1CCB: std::ios_base::sync_with_stdio(bool) (in /usr/lib/x86_64-lin
==5366==    by 0x1098BA: main (da2.cpp:461)
==5366==
==5366== LEAK SUMMARY:
==5366==    definitely lost: 0 bytes in 0 blocks
==5366==    indirectly lost: 0 bytes in 0 blocks
==5366==      possibly lost: 0 bytes in 0 blocks
==5366==    still reachable: 122,880 bytes in 6 blocks
==5366==         suppressed: 0 bytes in 0 blocks
==5366==
==5366== For counts of detected and suppressed errors,rerun with: -v
```

```
==5366== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

4 Выводы

Выполнив лабораторную работу , я научилась пользоваться некоторыми средствами поиска утечек памяти и профилирования и поняла, как важно ими пользоваться при разработке программ.

Список литературы

[1] *Особенности профилирования программ на C++ — Хабр.*
URL: `https://habr.com/ru/post/482040/` (дата обращения: 16.12.2020).