«Профилирование приложения»

**Цель работы:** Проанализировать работу приложения для ОС Linux и ОС Windows с помощью средств профилирования. Пронаблюдать разницу в работе приложений при использовании статических и динамических библиотек.

Используемые системы

Linux

|  |
| --- |
| System**: Linux Mint 18** Sarah x86\_64  Kernel: 4.4.0-34-generic DE: X-Cinnamon Session: cinnamon  ----------------------------------  Processor: Intel(R) Core(TM) i5-3210M CPU @ 2.50GHz  Memory (Gb): 3.7  Video: 00:02.0 VGA compatible controller: Intel Corporation 3rd Gen Core processor Graphics Controller (rev 09) Subsystem: Acer Incorporated [ALI] 3rd Gen Core processor Graphics Controller Kernel driver in use: i915 -- 01:00.0 VGA compatible controller: NVIDIA Corporation GF108M [GeForce GT 630M] (rev a1) Subsystem: Acer Incorporated [ALI] GF108M [GeForce GT 630M] Kernel driver in use: nvidia  ---------------------------------- |

Windows

|  |
| --- |
| C:\Users\Saboteur>systeminfo  Имя узла: SABOTEUR-PC  Название ОС: **Microsoft Windows 7** Максимальная  Версия ОС: 6.1.7601 Service Pack 1 сборка 7601  Изготовитель ОС: Microsoft Corporation  Параметры ОС: Изолированная рабочая станция  Сборка ОС: Multiprocessor Free  Зарегистрированный владелец: Saboteur  Зарегистрированная организация:  Код продукта: 00426-OEM-8992662-00006  Дата установки: 20.09.2015, 2:18:48  Время загрузки системы: 19.12.2015, 11:38:47  Изготовитель системы: Acer  Модель системы: Aspire V3-571G  Тип системы: x64-based PC  Процессор(ы): Число процессоров - 1.  [01]: Intel64 Family 6 Model 58 Stepping 9 Gen  uineIntel ~2501 МГц  Версия BIOS: Acer V1.13, 09.10.2012  Папка Windows: C:\Windows  Системная папка: C:\Windows\system32  Устройство загрузки: \Device\HarddiskVolume1  Язык системы: ru;Русский  Язык ввода: ru;Русский  Часовой пояс: (UTC+04:00) Волгоград, Москва, Санкт-Петербург  Полный объем физической памяти: 3 934 МБ  Доступная физическая память: 1 027 МБ  Виртуальная память: Макс. размер: 7 867 МБ  Виртуальная память: Доступна: 4 672 МБ  Виртуальная память: Используется: 3 195 МБ  Расположение файла подкачки: C:\pagefile.sys |

**Программа работы**

1)ОС Windows. Исследование утилиты, использующей статическую библиотеку

2)OC Windows. Исследование утилиты, использующей динамическую библиотеку

3)ОС Linux. Исследование утилиты, использующей статическую библиотеку

4)ОС Linux. Исследование утилиты, использующей динамическую библиотеку

ОС Windows.

Профилирование — сбор характеристик работы программы, таких как время выполнения отдельных фрагментов. Инструмент, используемый для анализа работы, называют профилировщиком или профайлером.

**Утилита CodeAnalyst**

AMD CodeAnalyst — профилировщик кода с поддержкой графического интерфейса пользователя. Несмотря на то, что CodeAnalyst разрабатывался только для работы с процессорами производства AMD, он также может ограниченно (без поддержки аппаратных событий) работать и с процессорами производства Intel. CodeAnalyst имеет схожий внешний вид и поведение как на платформе Linux, так и на платформе Microsoft Windows.[1]

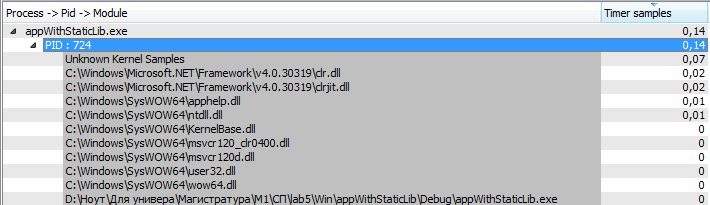


Рис.1 Анализ программы со статической библиотекой

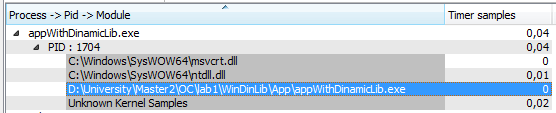


Рис. 2. Анализ программы c динамической библиотекой

Данная программа показывает какие библиотеки были использованы при запуске программы. Эти библиотеки автоматически добавляются в программу при компиляции. Например, из вывода видно, что используются системные динамические библиотеки с расширением .dll из папки SysWOW64.

Папка SysWOW64 — папка системных библиотек для запуска 32-разрядных приложений на 64-разрядной Windows. Все запускаемые 32-разрядные приложения используют библиотеки из этой папки.

Таким образом можно сделать вывод, что приложение скомпилировано под 32-х разрядную ОС

Программа также показывает время, затраченное на исполнение функций из системных библиотек.

Данная утилита предназначена для процессоров от AMD, в то время как на тестируемой машине установлен Intel процессор. Возможно, это является причиной того, что утилита предоставляет малое количество информации.

**Программа AQtime**

AQtime — профилировщик производительности, а также набор инструментов для отслеживания утечек памяти и ресурсов, разработанный компанией AutomatedQA.[2]

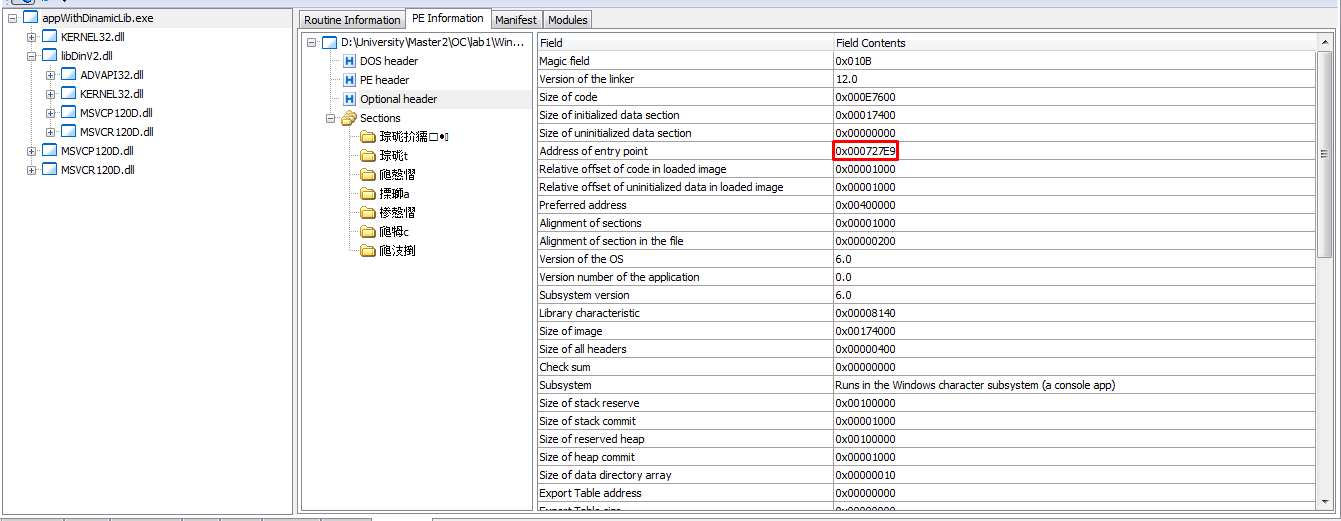


Рис.3. Исследование программы с динамической библиотекой. Первый запуск. Параметры программы

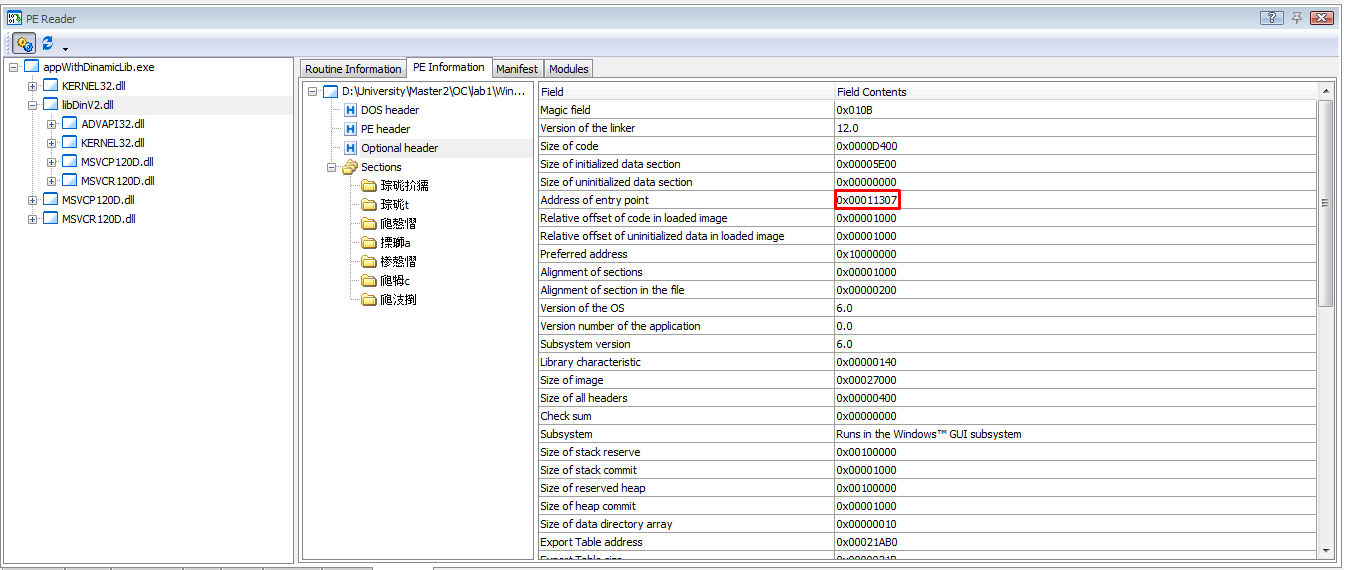


Рис. 4. Исследование программы с динамической библиотекой. Первый запуск. Параметры библиотеки

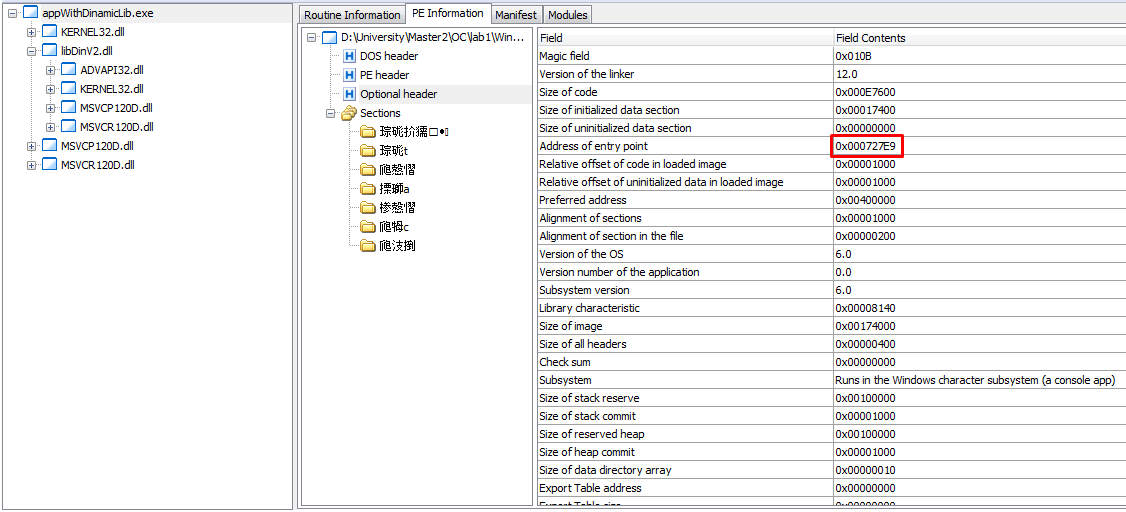


Рис. 5. Исследование программы с динамической библиотекой. Второй запуск. Параметры программы

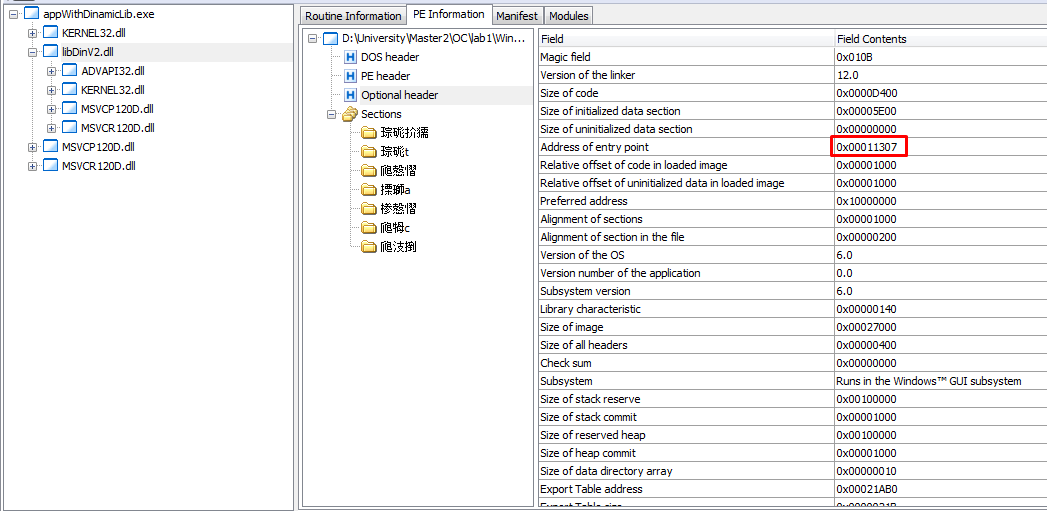


Рис. 6. Исследование программы с динамической библиотекой. Второй запуск. Параметры библиотеки

Как видно, адреса начала программы и динамической библиотеки в памяти не изменились.

Программа также показывает вызываемые функции, их ассемблерный код и строит дерево вызовов.

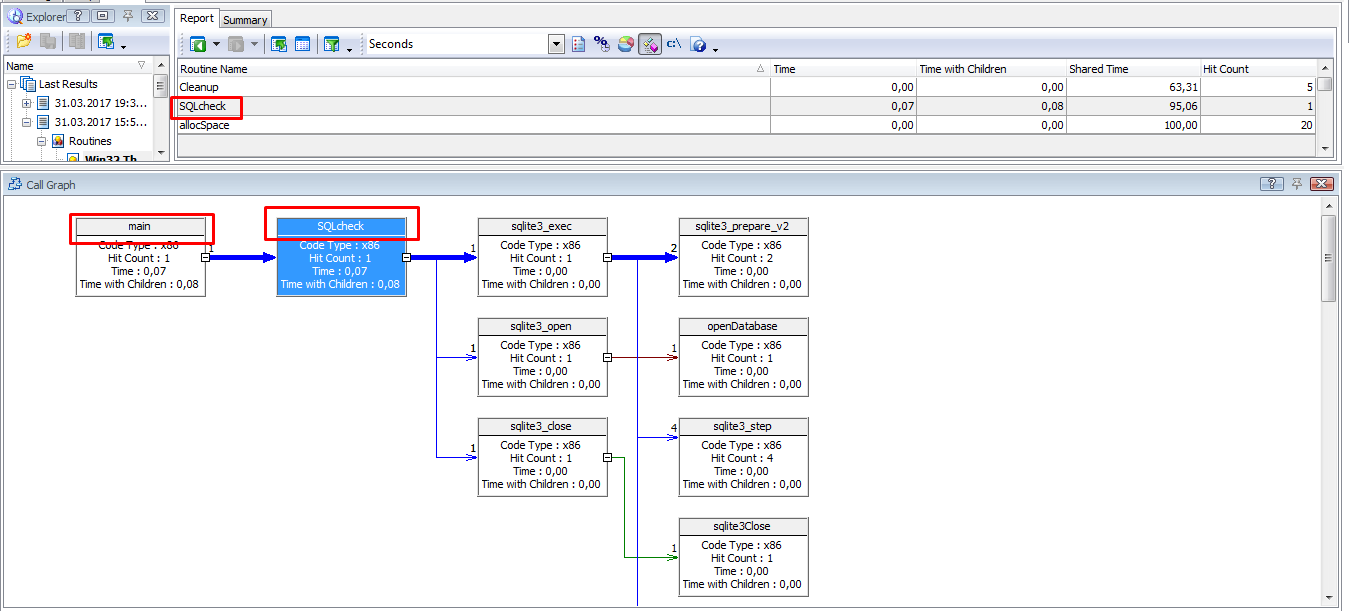


Рис. 7. Исследование программы с динамической библиотекой. Дерево вызовов

При использовании динамической библиотеки AQtime не отображает информацию о функциях, включенных в библиотеку.

Исследование программы со статической библиотекой.

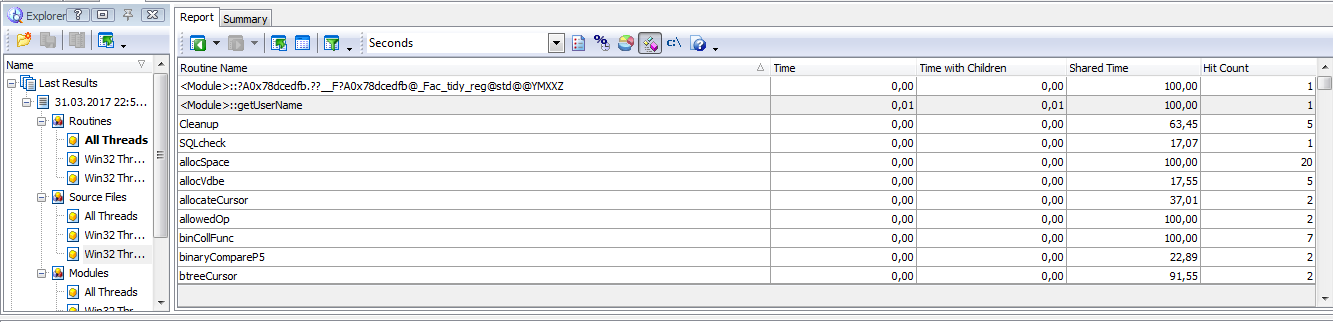


Рис. 8. Исследование программы со статической библиотекой

В данном случае программа отображает функции статической библиотеки в списке вызываемых функций. Однако точка монтирования статической библиотеки не указывается, т.к. она является частью скомпилированной программы. Библиотечные функции выделаются программой AQtime как <Module>.

**Visual Studio**

Среда разработки Visual Studio позволяет отлаживать программы. Параллельно с отладкой IDE позволяет просматривать ассемблерный код и код программы в hex формате.

Интересующие пользователя вкладки, могут быть активированы в меню Debug – Windows

Тестирование работы программы со статической библиотекой.

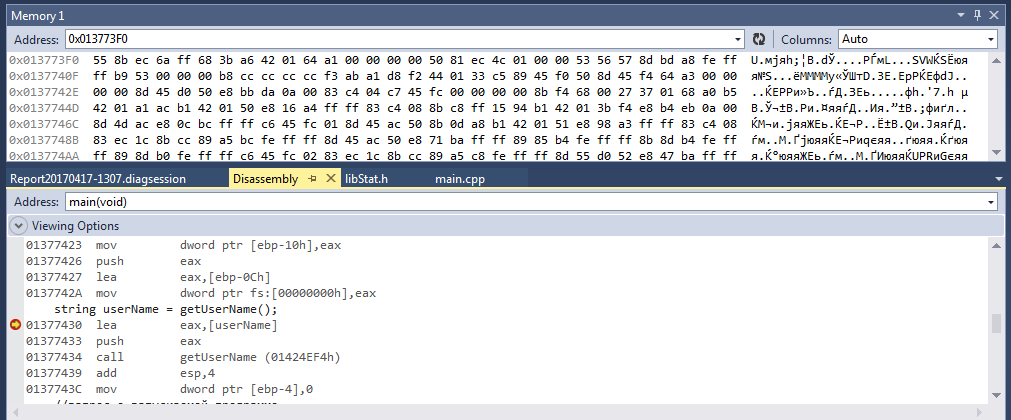


Рис. 9. Visual Studio. Окно Memory и Disassembly при выполнении отладки программы со статической библиотекой.

На рисунке 9, в окне с 16-ричным представлением файла программы виден адрес текущего места выполнения программы. Так же на рисунке 9 виден дизассемблерный код, в котором, к примеру, можно увидеть вызов библиотечной функции с помощью процессорной команды call. При этом среда разработки указывает точку начала функции getUserName, которая подключена из статической библиотеки.

VisualStudio позволяет провести профилирование программ. Вызов профайлера осуществляется из вкладки Analyse – Profiler.

Профайлер запускает программу, а затем предоставляет информацию, собранную при ее запуске.

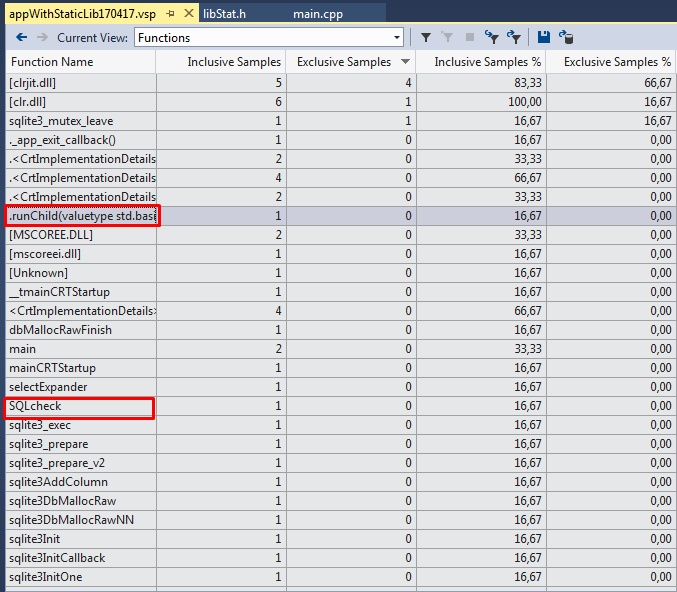


Рис.10. Профайлер Visual Studio. Окно Functions

На рисунке 10 можно увидеть список выполняемых функций, реди которых есть функции запуска нового процесса runChild из подключенной статической библиотеки, своя функция SQLcheck и множество функций статической библиотеки sqlLite.

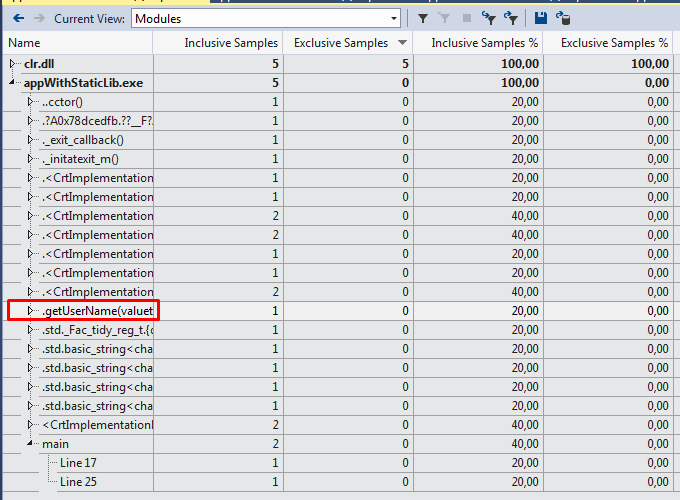


Рис.11 Профайлер Visual Studio. Окно Modules

Каждый запуск профайлера несколько отличается от предыдущего. На рисунке 11 видно появление функции getUserName, которая при предыдущем запуске не была отображена.

OS Linux

Исполняемый файл состоит из нескольких секций (машинный код, данные: переменные, константы, список внешних библиотечных функций при динамической компоновке, отладочную и прочую информацию), на основе которых операционная система создает исполняемый процесс.

Формат ELF (Execution & Linkable Format) - основной двоичный формат Linux и многих Unix-систем.[3]

Утилита readelf

Утилита позволяет получить информацию о структуре и составе исполняемого файла формата ELF..

Ключ --header отображает все заголовки в файле:

elf-заголовок отображает информацию о системе, для которой была скомпилированна программа. Эти данные могут быть использованы при компиляции новой программы для той же платформы что и та, с которой получен файл.

Разбор утилиты с динамической библиотекой.

Команда:

readelf -h starterDin

|  |
| --- |
| ELF Header:  Magic: 7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00  Class: ELF64  Data: 2's complement, **little endian**  Version: **1 (current)**  OS/ABI: **UNIX - System V**  ABI Version: 0  Type: EXEC (Executable file)  Machine: Advanced Micro Devices **X86-64**  Version:  **0x1**  Entry point address: **0x4022a0**  Start of program headers: 64 (bytes into file)  Start of section headers: 930560 (bytes into file)  Flags: 0x0  Size of this header: 64 (bytes)  Size of program headers: 56 (bytes)  Number of program headers: 9  Size of section headers: 64 (bytes)  Number of section headers: 32  Section header string table index: 29 |

Из этих данных можно получить следующую информацию:

little endian – порядок бит, используемый в процессоре. Существуюе так же big endian

UNIX - System V – тип операционной системы

Advanced Micro Devices X86-64 – тип процессора

0x4022a0 – точка монтирования программы

Для просмотра секций исполняемого файла используется команда:

readelf -S starterDin

|  |
| --- |
| Section Headers:  [Nr] Name Type Address Offset Size EntSize Flags Link Info Align  [ 0] NULL 0000000000000000 00000000 0000000000000000 0000000000000000 0 0 0  [ 1] .interp PROGBITS 0000000000400238 00000238 000000000000001c 0000000000000000 A 0 0 1  [ 2] .note.ABI-tag NOTE 0000000000400254 00000254 0000000000000020 0000000000000000 A 0 0 4  [ 3] .note.gnu.build-i NOTE 0000000000400274 00000274 0000000000000024 0000000000000000 A 0 0 4  [ 4] .gnu.hash GNU\_HASH 0000000000400298 00000298 0000000000000064 0000000000000000 A 5 0 8  [ 5] .dynsym DYNSYM 0000000000400300 00000300 0000000000000930 0000000000000018 A 6 1 8  [ 6] .dynstr STRTAB 0000000000400c30 00000c30 000000000000082a 0000000000000000 A 0 0 1  [ 7] .gnu.version VERSYM 000000000040145a 0000145a 00000000000000c4 0000000000000002 A 5 0 2  [ 8] .gnu.version\_r VERNEED 0000000000401520 00001520 00000000000000e0 0000000000000000 A 6 5 8  [ 9] .rela.dyn RELA 0000000000401600 00001600 0000000000000210 0000000000000018 A 5 0 8  [10] .rela.plt RELA 0000000000401810 00001810 0000000000000630 0000000000000018 AI 5 25 8  [11] .init PROGBITS 0000000000401e40 00001e40 000000000000001a 0000000000000000 AX 0 0 4  [12] .plt PROGBITS 0000000000401e60 00001e60 0000000000000430 0000000000000010 AX 0 0 16  [13] .plt.got PROGBITS 0000000000402290 00002290 0000000000000008 0000000000000000 AX 0 0 8  [14] .text PROGBITS 00000000004022a0 000022a0 00000000000a7e40 0000000000000000 AX 0 0 16  [15] .fini PROGBITS 00000000004aa0e0 000aa0e0 0000000000000009 0000000000000000 AX 0 0 4  [16] .rodata PROGBITS 00000000004aa100 000aa100 000000000000bce0 0000000000000000 A 0 0 32  [17] .eh\_frame\_hdr PROGBITS 00000000004b5de0 000b5de0 0000000000003cf4 0000000000000000 A 0 0 4  [18] .eh\_frame PROGBITS 00000000004b9ad8 000b9ad8 000000000000f5fc 0000000000000000 A 0 0 8  [19] .gcc\_except\_table PROGBITS 00000000004c90d4 000c90d4 0000000000000070 0000000000000000 A 0 0 1  [20] .init\_array INIT\_ARRAY 00000000006c9db8 000c9db8 0000000000000010 0000000000000000 WA 0 0 8  [21] .fini\_array FINI\_ARRAY 00000000006c9dc8 000c9dc8 0000000000000008 0000000000000000 WA 0 0 8  [22] .jcr PROGBITS 00000000006c9dd0 000c9dd0 0000000000000008 0000000000000000 WA 0 0 8  [23] .dynamic DYNAMIC 00000000006c9dd8 000c9dd8 0000000000000220 0000000000000010 WA 6 0 8  [24] .got PROGBITS 00000000006c9ff8 000c9ff8 0000000000000008 0000000000000008 WA 0 0 8  [25] .got.plt PROGBITS 00000000006ca000 000ca000 0000000000000228 0000000000000008 WA 0 0 8  [26] .data PROGBITS 00000000006ca240 000ca240 0000000000001790 0000000000000000 WA 0 0 32  [27] .bss NOBITS 00000000006cb9e0 000cb9d0 0000000000000980 0000000000000000 WA 0 0 32  [28] .comment PROGBITS 0000000000000000 000cb9d0 0000000000000034 0000000000000001 MS 0 0 1  [29] .shstrtab STRTAB 0000000000000000 000e31e2 000000000000011e 0000000000000000 0 0 1  [30] .symtab SYMTAB 0000000000000000 000cba08 000000000000d2f0 0000000000000018 31 1917 8  [31] .strtab STRTAB 0000000000000000 000d8cf8 000000000000a4ea 0000000000000000 0 0 1  Key to Flags:  W (write), A (alloc), X (execute), M (merge), S (strings), l (large)  I (info), L (link order), G (group), T (TLS), E (exclude), x (unknown)  O (extra OS processing required) o (OS specific), p (processor specific) |

Секции .data и .text соответствуют секциям исходного файла, а секции .comment и .shstrtab добавлены компилятором.

Секции могут быть прочитаны в 16-ричном формате с выделением кодируемых символов с помощью команды:

readelf -x .comment starterDin

|  |
| --- |
| Hex dump of section '.comment':  0x00000000 4743433a 20285562 756e7475 20352e34 GCC: (Ubuntu 5.4  0x00000010 2e302d36 7562756e 7475317e 31362e30 .0-6ubuntu1~16.0  0x00000020 342e3429 20352e34 2e302032 30313630 4.4) 5.4.0 20160  0x00000030 36303900 609. |

Из вывода видно, что программа скомпилирована с помощью gcc для Ubuntu 16.04.4

В секции .dynstr можно увидеть функции из подключаемых динамических библиотек, в том числе и из StarterDin.o

|  |
| --- |
| user3@user-Aspire-V3-571G ~/Univer/DinLibApp $ readelf -x .dynstr startertDin  Hex dump of section '.dynstr':  0x00400c30 006c6962 44696e2e 736f005f 5f676d6f .libDin.so.\_\_gmo  0x00400c40 6e5f7374 6172745f 5f005f4a 765f5265 n\_start\_\_.\_Jv\_Re  0x00400c50 67697374 6572436c 61737365 73005f49 gisterClasses.\_I  0x00400c60 544d5f64 65726567 69737465 72544d43 TM\_deregisterTMC  0x00400c70 6c6f6e65 5461626c 65005f49 544d5f72 loneTable.\_ITM\_r  0x00400c80 65676973 74657254 4d436c6f 6e655461 egisterTMCloneTa  0x00400c90 626c6500 5f5a3872 756e4368 696c644e ble.\_Z8**runChild**N  0x00400ca0 5374375f 5f637878 31313132 62617369 St7\_\_cxx1112basi  0x00400cb0 635f7374 72696e67 49635374 31316368 c\_stringIcSt11ch  0x00400cc0 61725f74 72616974 73496345 53614963 ar\_traitsIcESaIc  0x00400cd0 45454500 5f5a3131 67657455 7365724e EEE.\_Z11**getUserN**  0x00400ce0 616d6542 35637878 31317600 5f66696e **ame**B5cxx11v.\_fin  0x00400cf0 69006c69 62646c2e 736f2e32 00646c63 i.libdl.so.2.dlc |

readelf также может вывести список динамических функций в более читаемом формате.

|  |
| --- |
| user3@user-Aspire-V3-571G ~**/Univer/DinLibApp $ readelf -r startertDin**  Relocation section '.rela.dyn' at offset 0x1600 contains 22 entries:  Offset Info Type Sym. Value Sym. Name + Addend  0000006c9ff8 004e00000006 R\_X86\_64\_GLOB\_DAT 0000000000000000 \_\_gmon\_start\_\_ + 0  ...  0000006cbb40 005700000005 R\_X86\_64\_COPY 00000000006cbb40 \_ZSt3cin@GLIBCXX\_3.4 + 0  Relocation section '.rela.plt' at offset 0x1810 contains 66 entries:  Offset Info Type Sym. Value Sym. Name + Addend  0000006ca018 000100000007 R\_X86\_64\_JUMP\_SLO 0000000000000000 \_\_errno\_location@GLIBC\_2.2.5 + 0  0000006ca020 000200000007 R\_X86\_64\_JUMP\_SLO 0000000000000000 \_ZNSt7\_\_cxx1112basic\_s@GLIBCXX\_3.4.21 + 0  0000006ca028 000300000007 R\_X86\_64\_JUMP\_SLO 0000000000000000 pthread\_mutex\_trylock@GLIBC\_2.2.5 + 0  **0000006ca030 000500000007 R\_X86\_64\_JUMP\_SLO 0000000000000000 \_Z11getUserNameB5cxx11 + 0**  0000006ca038 000600000007 R\_X86\_64\_JUMP\_SLO 0000000000000000 \_ZNKSt7\_\_cxx1112basic\_@GLIBCXX\_3.4.21 + 0  ...  0000006ca098 001b00000007 R\_X86\_64\_JUMP\_SLO 0000000000000000 \_ZNSt7\_\_cxx1112basic\_s@GLIBCXX\_3.4.21 + 0  **0000006ca0a0 001c00000007 R\_X86\_64\_JUMP\_SLO 0000000000000000 \_Z8runChildNSt7\_\_cxx11 + 0**  0000006ca0a8 001d00000007 R\_X86\_64\_JUMP\_SLO 0000000000000000 sysconf@GLIBC\_2.2.5 + 0  ...  0000006ca220 006100000007 R\_X86\_64\_JUMP\_SLO 0000000000402280 \_ZNSt8ios\_base4InitD1E@GLIBCXX\_3.4 + 0 |

Для просмотра таблицы сегментов (информации о том, как загружаемые секции файла могут быть расположены в виртуальном адресном пространстве) необходимо ввести команду

readelf -l starterDin

|  |
| --- |
| Elf file type is EXEC (Executable file)  Entry point 0x4022a0  There are 9 program headers, starting at offset 64  Program Headers:  Type Offset VirtAddr PhysAddr  FileSiz MemSiz Flags Align  PHDR 0x0000000000000040 0x0000000000400040 0x0000000000400040  0x00000000000001f8 0x00000000000001f8 R E 8  INTERP 0x0000000000000238 0x0000000000400238 0x0000000000400238  0x000000000000001c 0x000000000000001c R 1  [Requesting program interpreter: /lib64/ld-linux-x86-64.so.2]  LOAD 0x0000000000000000 0x0000000000400000 0x0000000000400000  0x00000000000c9144 0x00000000000c9144 R E 200000  LOAD 0x00000000000c9db8 0x00000000006c9db8 0x00000000006c9db8  0x0000000000001c18 0x00000000000025a8 RW 200000  DYNAMIC 0x00000000000c9dd8 0x00000000006c9dd8 0x00000000006c9dd8  0x0000000000000220 0x0000000000000220 RW 8  NOTE 0x0000000000000254 0x0000000000400254 0x0000000000400254  0x0000000000000044 0x0000000000000044 R 4  GNU\_EH\_FRAME 0x00000000000b5de0 0x00000000004b5de0 0x00000000004b5de0  0x0000000000003cf4 0x0000000000003cf4 R 4  GNU\_STACK 0x0000000000000000 0x0000000000000000 0x0000000000000000  0x0000000000000000 0x0000000000000000 RW 10  GNU\_RELRO 0x00000000000c9db8 0x00000000006c9db8 0x00000000006c9db8  0x0000000000000248 0x0000000000000248 R 1  Section to Segment mapping:  Segment Sections...  00  01 .interp  02 .interp .note.ABI-tag .note.gnu.build-id .gnu.hash .dynsym .dynstr .gnu.version .gnu.version\_r .rela.dyn .rela.plt .init .plt .plt.got .text .fini .rodata .eh\_frame\_hdr .eh\_frame .gcc\_except\_table  03 .init\_array .fini\_array .jcr .dynamic .got .got.plt .data .bss  04 .dynamic  05 .note.ABI-tag .note.gnu.build-id  06 .eh\_frame\_hdr  07  08 .init\_array .fini\_array .jcr .dynamic .got |

Вывод списка используемых динамических библиотек

|  |
| --- |
| **user3@user-Aspire-V3-571G ~/Univer/StatLibApp $ readelf -d ../DinLibApp/startertDin**  Dynamic section at offset 0xc9dd8 contains 29 entries:  Tag Type Name/Value  **0x0000000000000001 (NEEDED) Shared library: [libDin.so]**  0x0000000000000001 (NEEDED) Shared library: [libdl.so.2]  0x0000000000000001 (NEEDED) Shared library: [libstdc++.so.6]  0x0000000000000001 (NEEDED) Shared library: [libgcc\_s.so.1]  0x0000000000000001 (NEEDED) Shared library: [libpthread.so.0]  0x0000000000000001 (NEEDED) Shared library: [libc.so.6]  0x000000000000000c (INIT) 0x401e40  ...  0x0000000000000000 (NULL) 0x0 |

Чтение elf заголовка динамической библиотеки

|  |
| --- |
| user3@user-Aspire-V3-571G ~/Univer/DinLibApp $ readelf -h libDin.so  ELF Header:  Magic: 7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00  Class: ELF64  Data: 2's complement, little endian  Version: 1 (current)  OS/ABI: UNIX - System V  ABI Version: 0  Type: **DYN (Shared object file)**  Machine: Advanced Micro Devices X86-64  Version: 0x1  Entry point address: **0xa80**  Start of program headers: 64 (bytes into file)  Start of section headers: 11344 (bytes into file)  Flags: 0x0  Size of this header: 64 (bytes)  Size of program headers: 56 (bytes)  Number of program headers: 7  Size of section headers: 64 (bytes)  Number of section headers: 30  Section header string table index: 27 |

Из приведенного листинга видно, что файл уже имеет метку динамической библиотеки.

В списке динамических функций для динамической библиотеки функций самой библиотеки нет

|  |
| --- |
| user3@user-Aspire-V3-571G ~/Univer/DinLibApp $ **readelf -r libDin.so | grep -E "runChild|getUserName"**  user3@user-Aspire-V3-571G ~/Univer/DinLibApp $ |

Разбор утилиты со статической библиотекой.

Вывод команды readelf –header appStatLib

|  |
| --- |
| ELF Header:  Magic: 7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00  Class: ELF64  Data: 2's complement, little endian  Version: 1 (current)  OS/ABI: UNIX - System V  ABI Version: 0  Type: EXEC (Executable file)  Machine: Advanced Micro Devices X86-64  Version: 0x1  Entry point address: **0x402360**  Start of program headers: 64 (bytes into file)  Start of section headers: 931040 (bytes into file)  Flags: 0x0  Size of this header: 64 (bytes)  Size of program headers: 56 (bytes)  Number of program headers: 9  Size of section headers: 64 (bytes)  Number of section headers: 32  Section header string table index: 29  Section Headers:  [Nr] Name Type Address Offset Size EntSize Flags Link Info Align  [ 0] NULL 0000000000000000 00000000 0000000000000000 0000000000000000 0 0 0  [ 1] .interp PROGBITS 0000000000400238 00000238 000000000000001c 0000000000000000 A 0 0 1  ...  [23] .dynamic DYNAMIC 00000000006c9de8 000c9de8 0000000000000210 0000000000000010 WA 6 0 8  [24] .got PROGBITS 00000000006c9ff8 000c9ff8 0000000000000008 0000000000000008 WA 0 0 8  ...  [30] .symtab SYMTAB 0000000000000000 000cba28 000000000000d3c8 0000000000000018 31 1919 8  [31] .strtab STRTAB 0000000000000000 000d8df0 000000000000a5cd 0000000000000000 0 0 1  Key to Flags:  W (write), A (alloc), X (execute), M (merge), S (strings), l (large)  I (info), L (link order), G (group), T (TLS), E (exclude), x (unknown)  O (extra OS processing required) o (OS specific), p (processor specific)  Program Headers:  Type Offset VirtAddr PhysAddr  FileSiz MemSiz Flags Align  PHDR 0x0000000000000040 0x0000000000400040 0x0000000000400040  0x00000000000001f8 0x00000000000001f8 R E 8  INTERP 0x0000000000000238 0x0000000000400238 0x0000000000400238  0x000000000000001c 0x000000000000001c R 1  [Requesting program interpreter: /lib64/ld-linux-x86-64.so.2]  LOAD 0x0000000000000000 0x0000000000400000 0x0000000000400000  0x00000000000c9404 0x00000000000c9404 R E 200000  LOAD 0x00000000000c9dc8 0x00000000006c9dc8 0x00000000006c9dc8  0x0000000000001c28 0x00000000000025b8 RW 200000  DYNAMIC 0x00000000000c9de8 0x00000000006c9de8 0x00000000006c9de8  0x0000000000000210 0x0000000000000210 RW 8  NOTE 0x0000000000000254 0x0000000000400254 0x0000000000400254  0x0000000000000044 0x0000000000000044 R 4  GNU\_EH\_FRAME 0x00000000000b602c 0x00000000004b602c 0x00000000004b602c  0x0000000000003d04 0x0000000000003d04 R 4  GNU\_STACK 0x0000000000000000 0x0000000000000000 0x0000000000000000  0x0000000000000000 0x0000000000000000 RW 10  GNU\_RELRO 0x00000000000c9dc8 0x00000000006c9dc8 0x00000000006c9dc8  0x0000000000000238 0x0000000000000238 R 1  Section to Segment mapping:  Segment Sections...  00  01 .interp  02 .interp .note.ABI-tag .note.gnu.build-id .gnu.hash .dynsym .dynstr .gnu.version .gnu.version\_r .rela.dyn .rela.plt .init .plt .plt.got .text .fini .rodata .eh\_frame\_hdr .eh\_frame .gcc\_except\_table  03 .init\_array .fini\_array .jcr .dynamic .got .got.plt .data .bss  04 .dynamic  05 .note.ABI-tag .note.gnu.build-id  06 .eh\_frame\_hdr  07  08 .init\_array .fini\_array .jcr .dynamic .got |

Из вывода видно, что точки монтирования разных программ не совпадают

Попытка найти в списке динамических функций функции из подключаемой статической библиотеки:

|  |
| --- |
| user3@user-Aspire-V3-571G ~/Univer/StatLibApp $ readelf -r appStatLib | grep -E "runChild|getUserName"  user3@user-Aspire-V3-571G ~/Univer/StatLibApp $ |

Как видно из вывода утилиты, в разделе динамических функций, функций runChild и getUserName нет.

Вывод списка используемых динамических библиотек

|  |
| --- |
| user3@user-Aspire-V3-571G ~/Univer/StatLibApp $ readelf -d appStatLib  Dynamic section at offset 0xc9de8 contains 28 entries:  Tag Type Name/Value  0x0000000000000001 (NEEDED) Shared library: [libdl.so.2]  0x0000000000000001 (NEEDED) Shared library: [libstdc++.so.6]  0x0000000000000001 (NEEDED) Shared library: [libgcc\_s.so.1]  0x0000000000000001 (NEEDED) Shared library: [libpthread.so.0]  0x0000000000000001 (NEEDED) Shared library: [libc.so.6]  0x000000000000000c (INIT) 0x401e98  ...  0x0000000000000000 (NULL) 0x0 |

Как видно, в списке нет статической библиотеки.

Рассмотрим статическую библиотеку

|  |
| --- |
| user3@user-Aspire-V3-571G ~/Univer/StatLibApp $ readelf -h libStat.a  File: libStat.a(StarterLib.o)  ELF Header:  Magic: 7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00  Class: ELF64  Data: 2's complement, little endian  Version: 1 (current)  OS/ABI: UNIX - System V  ABI Version: 0  Type: **REL (Relocatable file)**  Machine: Advanced Micro Devices X86-64  Version: 0x1  Entry point address: **0x0**  Start of program headers: 0 (bytes into file)  Start of section headers: 2352 (bytes into file)  Flags: 0x0  Size of this header: 64 (bytes)  Size of program headers: 0 (bytes)  Number of program headers: 0  Size of section headers: 64 (bytes)  Number of section headers: 14  Section header string table index: 11 |

Из вывода видно, что файл имеет тип “перемещаемый” (Relocatable) и точку монтирования 0x0, что дает нам понять о том что это библиотека

Вывод

Исходя из проделанных экспериментов, можно сделать вывод, что статическая библиотека никак не выдает свое присутсвие в скомпилированном файле программы, в то время как ссылки на динамическую библиотеку всегда можно найти.

Профилирование

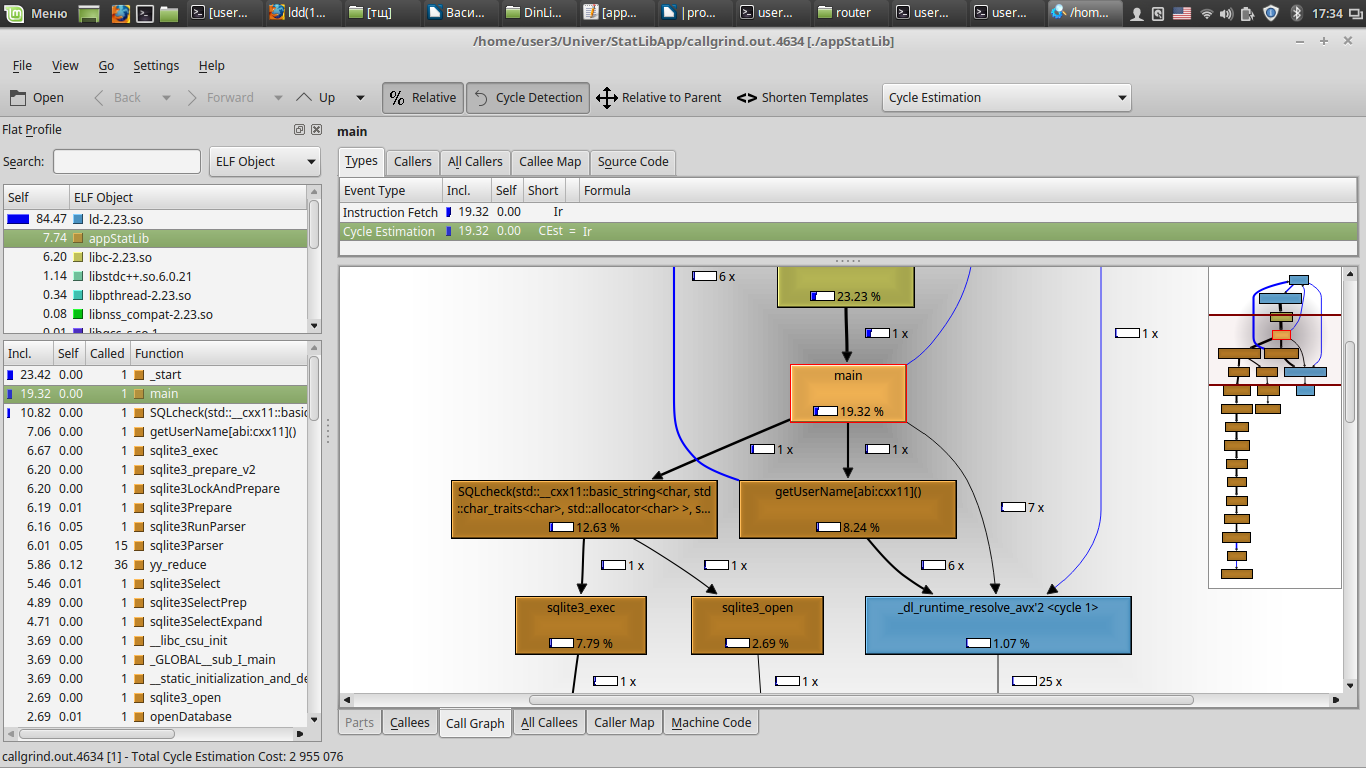
Утилита valgrind

Valgrind хорошо известен как мощное средство поиска ошибок работы с памятью. Но кроме этого, в его составе имеется некоторое количество дополнительных утилит, предназначенных для профилирования программ, анализа потребления памяти и поиска ошибок связанных с синхронизацией в многопоточных программах.[4]

|  |
| --- |
| user3@user-Aspire-V3-571G ~/Univer/StatLibApp $ valgrind --tool=callgrind ./appStatLib  ==4634== Callgrind, a call-graph generating cache profiler  ==4634== Copyright (C) 2002-2015, and GNU GPL'd, by Josef Weidendorfer et al.  ==4634== Using Valgrind-3.11.0 and LibVEX; rerun with -h for copyright info  ==4634== Command: ./appStatLib  ==4634==  ==4634== For interactive control, run 'callgrind\_control -h'.  Please, enter program path  /usr/bin/firefox  Opened database successfully  Operation done successfully  ==4634==  ==4634== Events : Ir  ==4634== Collected : 2955076  ==4634==  ==4634== I refs: 2,955,076 |

Модуль callgrind позволяет собрать информацию о дереве вызова функций в программе. Информация сохраняется в файл callgrind.out.\*

Просматривать сохраненную информацию можно в графическом варианте. Для этого необходимо установить дополнительную утилиту Kcachgrind

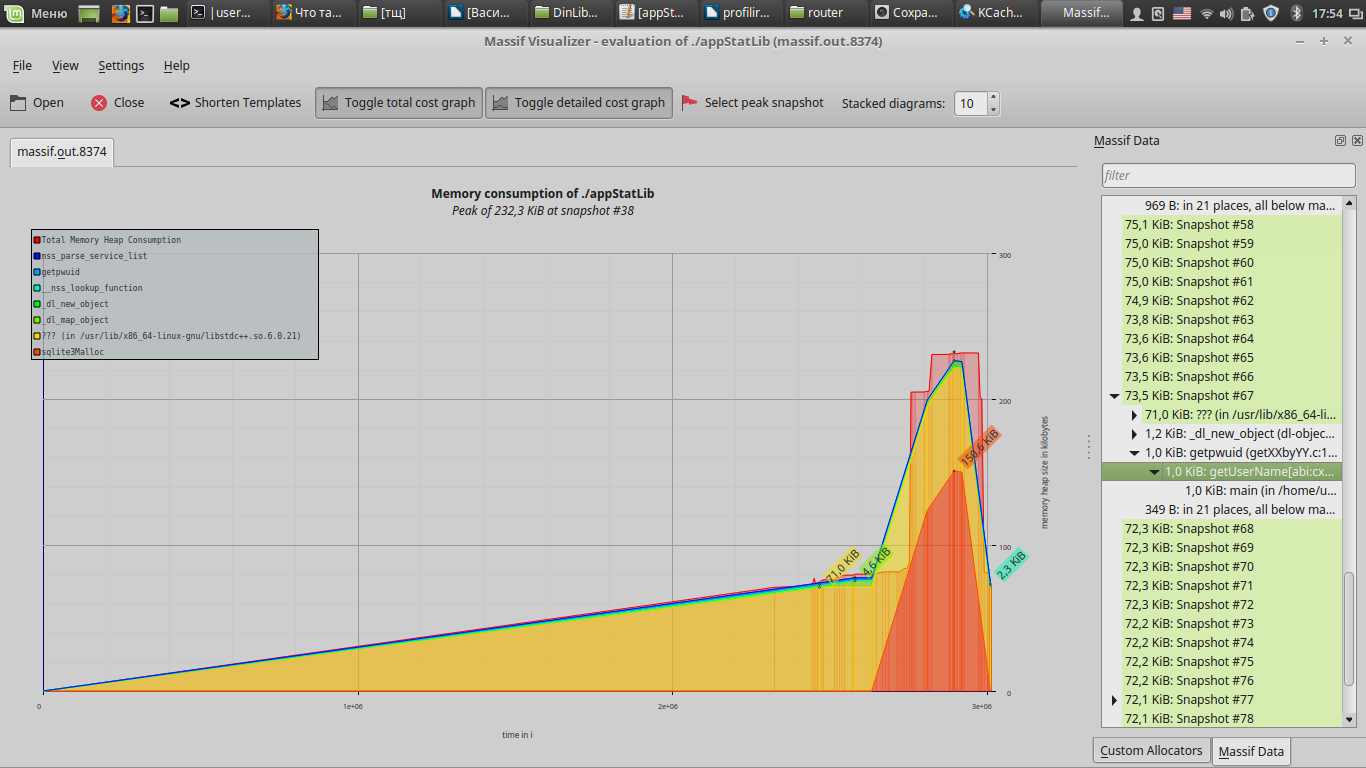
Рис. 11. Утилита KcachGrind

В левом верхнем окне отображаются компоненты программы: сама программа и используемые динамические библиотеки. В нижнем левом окне отображен список функций каждого из компонентов программы. Для каждой функции указано число выховов и затраченное время. В центральном огне можно отобразить граф вызовов. Утилита также отображает процентное отношение используемого времени для каждой функции. Так функция getUserName занимает 7% используемого времени.

Для анализа выделения памяти в программах используется модуль massif. Он собирает сведения не только о размерах блоков, выделяемых программой, но также и о том, сколько дополнительной памяти потребуется для хранения служебной информации.

|  |
| --- |
| user3@user-Aspire-V3-571G ~/Univer/StatLibApp $ valgrind --tool=massif ./appStatLib  ==8374== Massif, a heap profiler  ==8374== Copyright (C) 2003-2015, and GNU GPL'd, by Nicholas Nethercote  ==8374== Using Valgrind-3.11.0 and LibVEX; rerun with -h for copyright info  ==8374== Command: ./appStatLib  ==8374==  Please, enter program path  /usr/bin/firefox  Opened database successfully  Operation done successfully  ==8374== |

Данные о работе утилиты сохраняются в файл massif.out.\*. Данные так же могут быть визуализированы с помощью утилиты massif-vizualizer.

Рис.12 Утилита massif-vizualizer.

Утилита massif-visualizer отображает красным цветом заднего фона максимально занимаемую память: 232.3 Кб. Остальные цвета отображают работу библиотек. Например желтый – работа библиотеки libstdc++.so, а красный цвет на переднем фоне – sqlite.

Утилита позволяет найти отдельно взятую функцию и посмотреть используемую ею память.

Вывод

В данной работе были рассмотрены утилиты позволяющие проводить профилирование программ, а также утилиты позволяющие получить информацию о программе на основе ее исполняемого файла.

Для ОС Windows рассмотрены такие утилиты как AQTime и CodeAnalyst. С их помощью проведено профилирование и рассмотрены некоторые параметры программ. Так же для профилирования была использована среда разработки Visual Studio и ее отладчик.

В ОС Linux рассмотрены утилиты readelf и valgrind. Обе утилиты позволяют получить обширную информацию о программе.

В ОС Linux профилирование является более легким процессом, т.к. существуют удобные и проверенные программы, а также сама система нацелена на полный контроль над тем что в ней происходит, в том числе и понимание того какая программа выполняется.

Список используемых источников

# CodeAnalyst. URL: http://developer.amd.com/tools-and-sdks/archive/compute/amd-codeanalyst-performance-analyzer/amd-codeanalyst-performance-analyzer-for-linux/

# AQtime 8. URL: <https://smartbear.com/product/aqtime-pro/overview/>

# Анатомия динамических библиотек Linux. URL:<https://www.ibm.com/developerworks/ru/library/l-dynamic-libraries/>

# Что такое valgrind и зачем он нужен. URL: <http://alexott.net/ru/linux/valgrind/Valgrind.html>