Санкт-Петербургский политехнический университет Петра Великого

Институт компьютерных наук и технологий

Кафедра компьютерных систем и программных технологий

Лабораторная работа

Дисциплина: Проектирование ОС и компонентов

Тема: Анализ и профилирование прикладных приложений в ОС Windows и Linux

Выполнил студент гр. 13541/4 \_\_\_\_\_\_\_\_\_\_ Зорин А.Г.

(подпись)

Руководитель \_\_\_\_\_\_\_\_\_\_ Душутина Е.В.

(подпись)

«\_\_»\_\_\_\_\_\_\_\_ 2017 г.

Санкт –Петербург

2017

**Цель работы:**

В данной работе необходимо выполнить:

* Анализ загрузки приложения в ОЗУ
* Профилирование приложения
* Трассировку

Все эти пункты необходимо выполнить по отношению к утилите, которая была написана ранее. Суть этой утилиты заключается в отслеживании изменений файлов и директорий. Эти пункты выполняются в ОС двух семейств.

**Выполнение работы:**

1. Ubuntu

Информация о системе:

|  |
| --- |
| ➜ ~ uname -a  Linux Arseny-HP-Pavilion-dv6-Notebook-PC 4.4.0-66-generic #87-Ubuntu SMP Fri Mar 3 15:29:05 UTC 2017 x86\_64 x86\_64 x86\_64 GNU/Linux |

Для анализа адресного пространства процесса был использован отладчик gdb. Помимо этого, был проанализирован дизассемблированный код утилиты с помощью objdump.

Для начала был определен адрес точки входа с помощью objdump:

|  |
| --- |
| ➜ inot objdump -d ./bin\_inot > dis.out  ➜ inot head -n 20 dis.out  ./bin\_inot: file format elf64-x86-64  Disassembly of section .init:  00000000004007c0 <\_init>:  **4007c0**: 48 83 ec 08 sub $0x8,%rsp  4007c4: 48 8b 05 2d 08 20 00 mov 0x20082d(%rip),%rax # 600ff8 <\_DYNAMIC+0x1f0>  4007cb: 48 85 c0 test %rax,%rax  4007ce: 74 05 je 4007d5 <\_init+0x15>  4007d0: e8 ab 00 00 00 callq 400880 <atoi@plt+0x10>  4007d5: 48 83 c4 08 add $0x8,%rsp  4007d9: c3 retq  Disassembly of section .plt:  00000000004007e0 <write@plt-0x10>:  4007e0: ff 35 22 08 20 00 pushq 0x200822(%rip) # 601008 <\_GLOBAL\_OFFSET\_TABLE\_+0x8>  4007e6: ff 25 24 08 20 00 jmpq \*0x200824(%rip) # 601010 <\_GLOBAL\_OFFSET\_TABLE\_+0x10> |

Objdump показал, что точка входа в программу располагается по адресу 4008f0. Для того, чтобы удостовериться в этом, был проверен адрес входа с помощью gdb:

|  |
| --- |
| ➜ inot gdb ./bin\_inot /home/arseny/DirForWatching  Reading symbols from ./bin\_inot...(no debugging symbols found)...done.  "/home/arseny/DirForWatching" is not a core dump: Is a directory  (gdb) inspect \_init  $1 = {<text variable, no debug info>} **0x4007c0** <\_init>  (gdb) inspect main  $2 = {int (int, char \*\*)} 0x400986 <main> |

Исходя из листинга результатов выше, можно сделать вывод, что они совпадают с результатами дизасемблированного кода. С помощью gdb можно узнать адреса остальных функций, например main.

Можно убедиться, также в том, что статическая библиотека связывается с исполняемым файлом сразу же. А динамическая — по требованию. Для этого используем inspect с именем функции из библиотеки.

|  |  |
| --- | --- |
| Статическая библиотека | Динамическая библиотека |
| debug\_print\_mask  wd\_path | do\_watch  add\_dir\_watch  add\_dir |

|  |
| --- |
| (gdb) inspect debug\_print\_mask  $3 = {void (uint32\_t)} 0x7ffff7bd4ac8 <debug\_print\_mask>  (gdb) inspect wd\_path  $4 = {void (int, char \*)} 0x7ffff7bd4ca8 <wd\_path> |

Из листинга выше, мы видим адреса функций из статической библиотеки. Для того, чтобы увидеть адреса функций из динамической библиотеки, необходимо запустить программу и остановить ее в том месте, где происходит вызов методов из динамической библиотеки.

|  |
| --- |
| (gdb) r  Starting program: /home/arseny/SystemPr/Winter 2017/inot/inot  ^C  Program received signal SIGINT, Interrupt.  0x00007ffff7900680 in \_\_read\_nocancel () at ../sysdeps/unix/syscall-template.S:84  84 ../sysdeps/unix/syscall-template.S: No such file or directory.  (gdb) inspect do\_watch  $5 = {int (int)} 0x7ffff7bd3f00 <do\_watch>  (gdb) inspect add\_dir\_watch  $6 = {int (int, char \*, char \*, int)} 0x7ffff7bd441c <add\_dir\_watch>  (gdb) inspect add\_dir  $7 = {void (int, char \*, int, int)} 0x7ffff7bd4689 <add\_dir> |

Для профилирования утилиты использовалось инструментальное ПО valgrind, которое предназначено для обнаружения утечек памяти, а также профилирования. Запуск анализа производится следующим образом:

|  |
| --- |
| ➜ inot valgrind --tool=massif ./inot  ==4275== Massif, a heap profiler  ==4275== Copyright (C) 2003-2015, and GNU GPL'd, by Nicholas Nethercote  ==4275== Using Valgrind-3.11.0 and LibVEX; rerun with -h for copyright info  ==4275== Command: ./inot  ==4275==  Path: //home//arseny//DirForWatching/  File: Untitled Document  Event: FILE WAS CREATED IN WATCHING DIR  Path: //home//arseny//DirForWatching/  File: Untitled Document  Event: FILE WAS CLOSED  -  Path: //home//arseny//DirForWatching/  File: Untitled Document  Event: FILE WAS MOVED OUTSIDE WATCHING DIR  Path: //home//arseny//DirForWatching/  File: Doc  Event: FILE WAS MOVED INSIDE WATCHING DIR  -  Path: //home//arseny//DirForWatching/  File: .goutputstream-FQO4YY  Event: FILE WAS CREATED IN WATCHING DIR  -  Path: //home//arseny//DirForWatching/  File: .goutputstream-FQO4YY  Event:  Path: //home//arseny//DirForWatching/  File: Doc  Event: FILE WAS CLOSED  -  Path: //home//arseny//DirForWatching/  File: .goutputstream-FQO4YY  Event: FILE WAS MODIFIED  -  Path: //home//arseny//DirForWatching/  File: .goutputstream-FQO4YY  Event: FILE WAS MOVED OUTSIDE WATCHING DIR  Path: //home//arseny//DirForWatching/  File: Doc  Event: FILE WAS MOVED INSIDE WATCHING DIR  -  Path: //home//arseny//DirForWatching/  File: Doc  Event: FILE WAS CLOSED  -  Path: //home//arseny//DirForWatching/  File: Untitled Folder  Event: OBJECT IS A DIR FILE WAS CREATED IN WATCHING DIR  //home//arseny//DirForWatching/Untitled Folder  -  Path: //home//arseny//DirForWatching/  File: Untitled Folder  Event: FILE WAS MOVED OUTSIDE WATCHING DIR OBJECT IS A DIR  Path: //home//arseny//DirForWatching/  File: Fold  Event: FILE WAS MOVED INSIDE WATCHING DIR OBJECT IS A DIR  //home//arseny//DirForWatching/Fold  Path: //home//arseny//DirForWatching/Fold/  File: t  Event: WATCHING DIR WAS MOVED  -  Path: //home//arseny//DirForWatching/  File: Doc  Event: FILE WAS MOVED OUTSIDE WATCHING DIR  Path: //home//arseny//DirForWatching/Fold/  File: Doc  Event: FILE WAS MOVED INSIDE WATCHING DIR  -  Path: //home//arseny//DirForWatching/  File: Fold  Event: FILE WAS MOVED OUTSIDE WATCHING DIR OBJECT IS A DIR  Path: //home//arseny//DirForWatching/Fold/  File: Doc  Event: WATCHING DIR WAS MOVED  -  ^C==4275==  ==4275== Process terminating with default action of signal 2 (SIGINT)  ==4275== at 0x512B680: \_\_read\_nocancel (syscall-template.S:84)  ==4275== by 0x4E323F0: do\_watch (dynamic.c:29)  ==4275== by 0x400ADD: main (inot.c:42)  ==4275== |

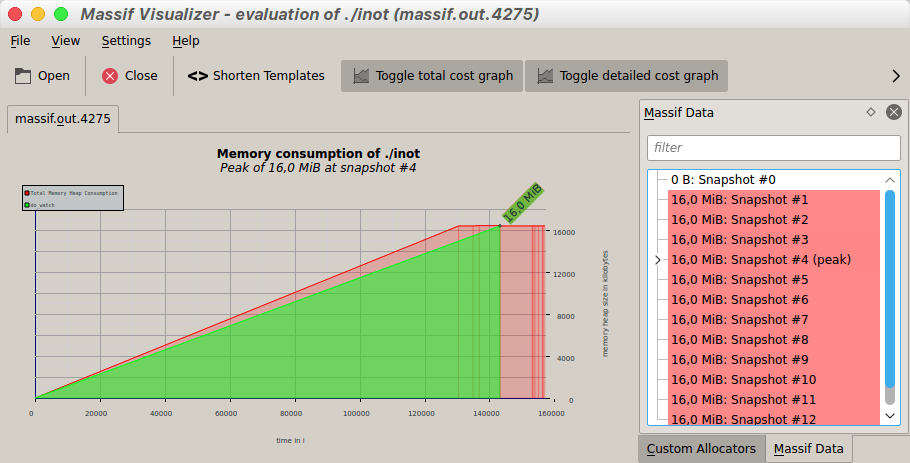
В результате выполнения valgrind был создан файл massif.out.4275. Он выглядит следующим образом:

|  |
| --- |
| desc: (none)  cmd: ./inot  time\_unit: i  #-----------  snapshot=0  #-----------  time=0  mem\_heap\_B=0  mem\_heap\_extra\_B=0  mem\_stacks\_B=0  heap\_tree=empty |

Результат выполнения предыдущих команд сгенерировал такой файл, размером 112 строк. Такой файл тяжелочитаем и получить полезную информацию из него сложно. Поэтому, воспользуемся визуализатором massif-vizualizator.

|  |
| --- |
| ➜ inot sudo massif-visualizer massif.out.4275  Error: "/var/tmp/kdecache-arseny" is owned by uid 1000 instead of uid 0.  massif-visualizer(4492)/kdeui (kdelibs): Attempt to use QAction "toggleDataTree" with KXMLGUIFactory!  massif-visualizer(4492)/kdeui (kdelibs): Attempt to use QAction "toggleAllocators" with KXMLGUIFactory!  Error: "/tmp/kde-arseny" is owned by uid 1000 instead of uid 0.  description: "(none)"  command: "./inot"  time unit: "i"  snapshots: 13  peak: snapshot # 4 after "143211i"  peak cost: "16,0 MiB" heap "4,0 KiB" heap extra "0 B" stacks |

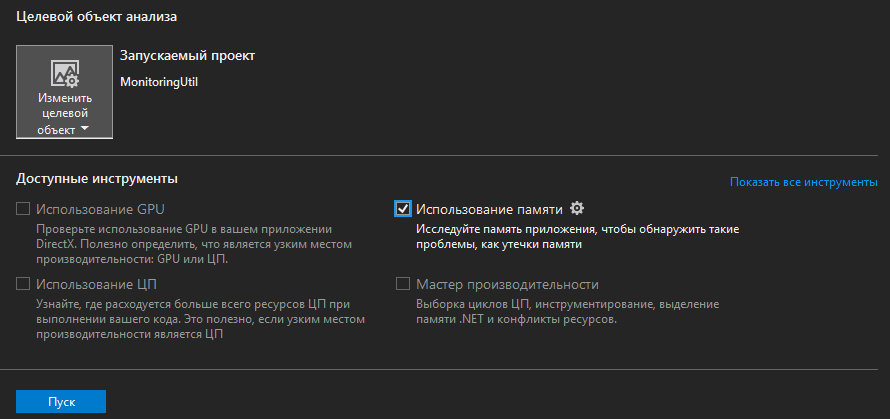
Рисунок ниже показывает, как отображает информацию визуализатор massif-vizualizer. Из данного графика можно видеть, что наша утилита использует, примерно, 16 Мб памяти.



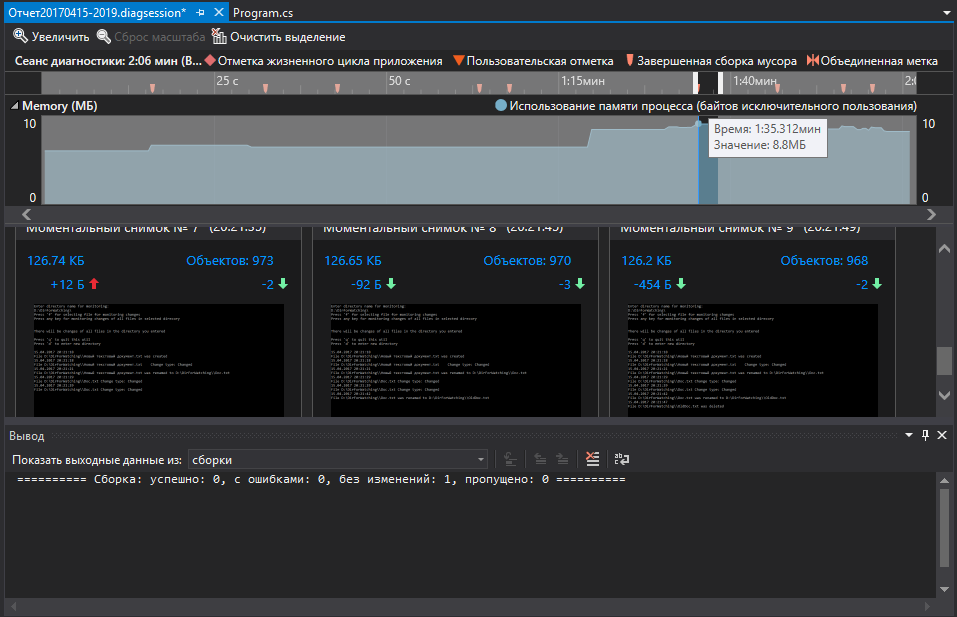
2. Windows 10.

|  |
| --- |
| C:\Users\zorin>systeminfo  Имя узла: DESKTOP-1LD1IT8  Название ОС: Майкрософт Windows 10 Pro (Registered Trademark)  Версия ОС: 10.0.14393 Н/Д построение 14393  Изготовитель ОС: Microsoft Corporation  Параметры ОС: Изолированная рабочая станция  Построение ОС: Multiprocessor Free  Зарегистрированный владелец: Пользователь Windows |

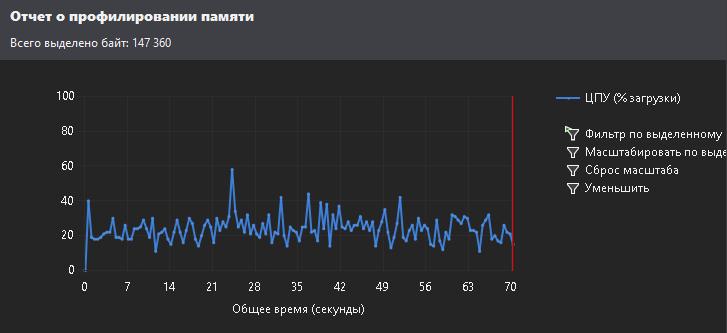
Для профилирования утилиты в системе Windows, можно использовать встроенные методы Visual Studio. Эта IDE предоставляет возможность провести несколько видов профилирования. От использования памяти до использования GPU. Чтобы запустить профилирование, во вкладке «Анализ» выбираем «Профилировщик производительности». Но, в первую очередь, установим конфигурацию проекта в Release, т.к. в версию Debug, установленную по умолчанию, встраивается дополнительная информация для отладки приложения, и она плохо скажется на точности результатов профилирования. Далее выберем проверку использования памяти.

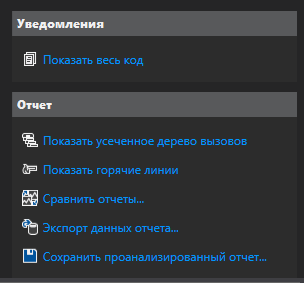


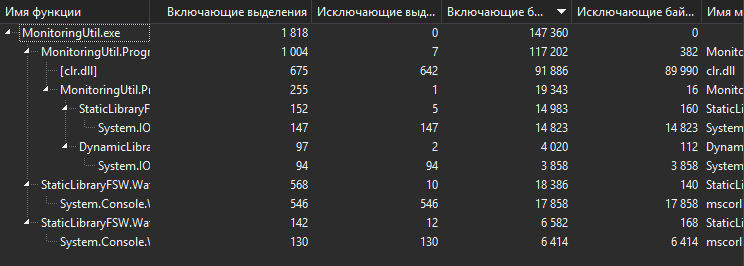
Выбираем запускаемый проект и инструмент, который хотим использовать. После запуска профилирования, мы получаем отчет расширения .diagsession c результатами проверки. Из этого файла можно увидеть, что максимальное значение используемой памяти не превышает 8.8 Мб. В ходе профилирования можно создавать, так называемые, «снимки», которые покажут подробную информацию об используемой памяти в ходе его создания.



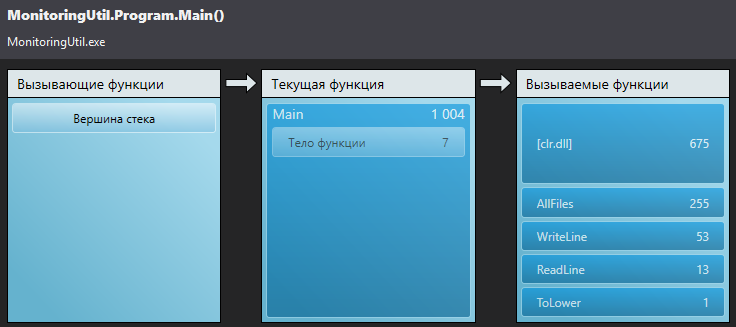
Для дальнейшего профилирования был произведен повторный запуск провилировщика производительности. Но на этот раз был выбран «Мастер производительности». После того, как было проведено профилирование, в качестве результата был получен файл .vsp. В нем можно посмотреть отчет о профилировании памяти, дерево вызовов или горячие линии.



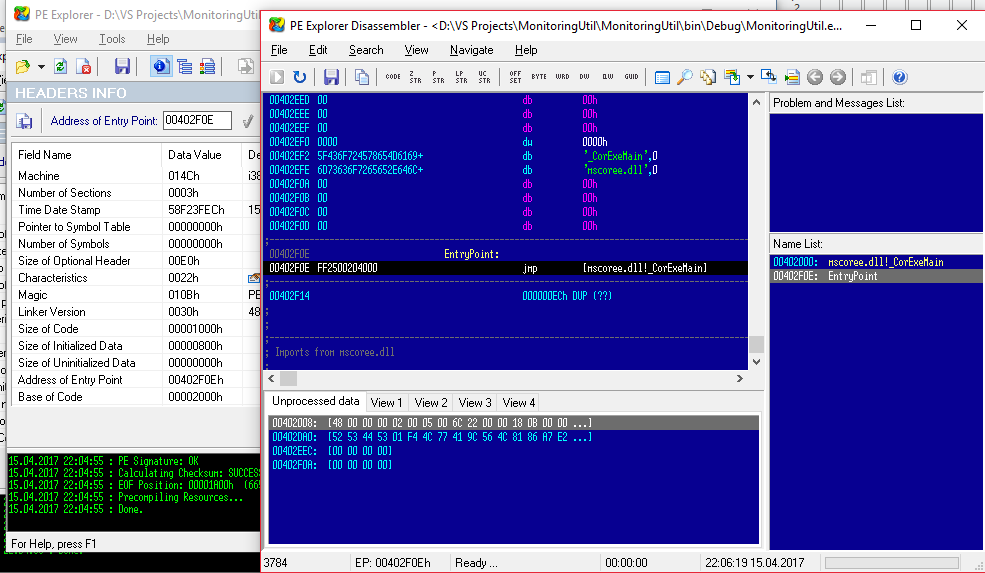




На рисунке выше показан граф вызовов для утилиты. В ней можно увидеть, что статическая и динамическая библиотека успешно подключаются к проекту. Здесь же можно увидеть кол-во байт, выделенное функцией и ее потомками (4-ая колонка). Можно посмотреть вызываемые методы:



Для просмотра адреса входа программы была использована утилита PE Explorer, которая позволяет произвести дизассемблирование программы и посмотреть адрес входа.



**Вывод:**

В данной работе были изучены основы анализа прикладных утилит в ОС Windows и Linux. Сначала было проанализировано адресное пространство программы. С помощью инструментов отладки и дизассемблирования были определены адреса точек входа программ. Так же было проанализировано распределение статических и динамических библиотек в адресном пространстве процесса и определено, что статические библиотеки изначально располагаются в памяти процесса, а динамические подгружаются в память во время выполнения. Результаты в обеих ОС были схожи.

Далее было произведено профилирование программ. Профилирование показало, что наиболее ресурсоемкими функциями в программах являются те, которые производят вызовы функций WinAPI или системных вызовов Linux.

**Список литературы:**

1. Сайт gdb: https://www.gnu.org/software/gdb/
2. Профилирование приложений в Visual Studio: https://habrahabr.ru/post/98361/
3. Valgrind: http://www.valgrind.org