DOTNEXT

А как Вы отлаживаете приложения под mono?

Тонкости отладки mono-приложений

Елизавета Голенок
Ведущий разработчик Gotech Software
mail: golenok-ea5@narod.ru
twitter: @marmothetka

Москва 2017

О чём поговорим?

- С чего всё начиналось

О чём поговорим?

- С чего всё начиналось
- Стандартный отладчик mono

О чём поговорим?

- С чего всё начиналось
- Стандартный отладчик mono
- Удалённая отладка

О чём поговорим?

- С чего всё начиналось
- Стандартный отладчик mono
- Удалённая отладка
- Расширяем кругозор: GDB, LLDB

О чём поговорим?

- С чего всё начиналось
- Стандартный отладчик mono
- Удалённая отладка
- Расширяем кругозор: GDB, LLDB
- Дедлоки

О чём поговорим?

- С чего всё начиналось
- Стандартный отладчик mono
- Удалённая отладка
- Расширяем кругозор: GDB, LLDB
- Дедлоки
- Утечки памяти

О чём поговорим?

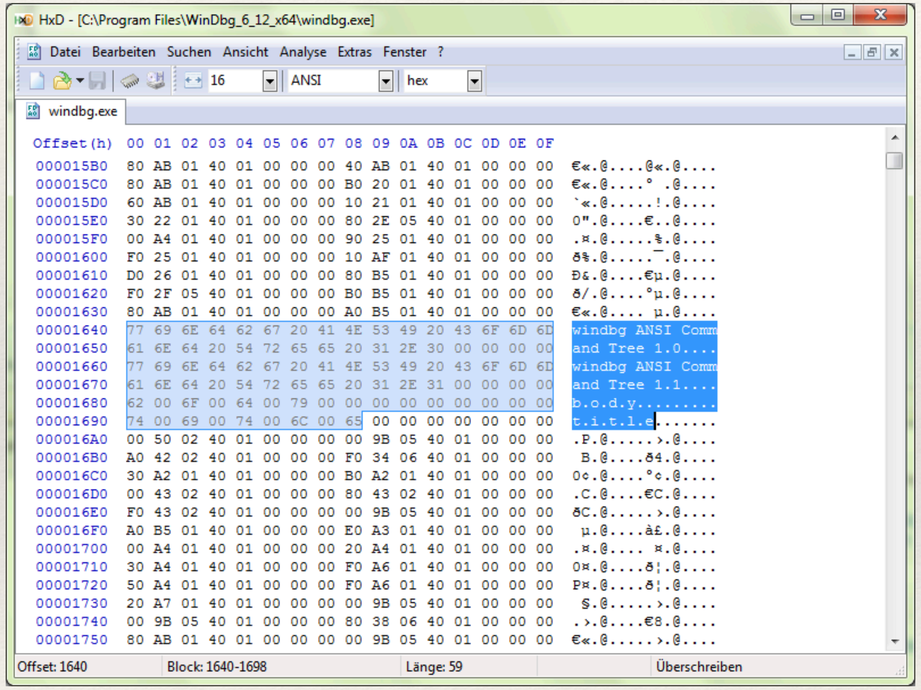
- Mono embedding - как подавать его «горячим»

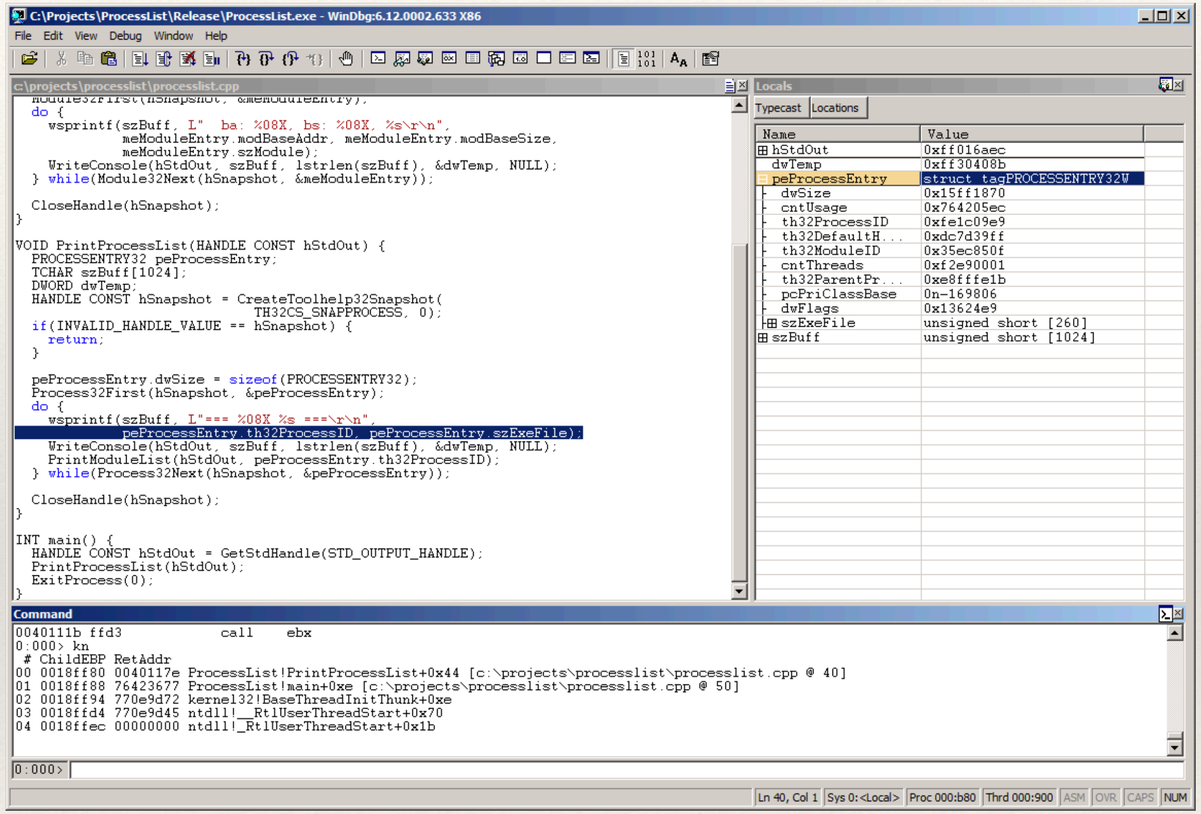
О чём поговорим?

- Mono embedding - как подавать его «горячим»
- Сравним WinDBG с GDB/LLDB

О чём поговорим?

- Mono embedding - как подавать его «горячим»
- Сравним WinDBG с GDB/LLDB





Окружение

- Mac OS Sierra

Окружение

- Mac OS Sierra
- Ubuntu 16.04, 17.04

О структуре материала

- Задача

О структуре материала

- Задача
- Инструменты

О структуре материала

- Задача
- Инструменты
- Решение

О структуре материала

- Задача
- Инструменты
- Решение
- Выводы





Добро пожаловать в мир **MONO_DEBUG=1**

Немного предыстории

- MONO_DEBUG=1

Немного предыстории

- MONO_DEBUG=1
- MDB

Немного предыстории

- MONO_DEBUG=1
- MDB
- Mono Hard Debugger

Немного предыстории

- MONO_DEBUG=1
- MDB
- Mono Hard Debugger
- Консоль

Немного предыстории

- MONO_DEBUG=1
- MDB
- Mono Hard Debugger
- Консоль
- Удалённая отладка через GDB

Немного предыстории

- MONO_DEBUG=1
- MDB
- Mono Hard Debugger
- Консоль
- Удалённая отладка через GDB
- Отладка небезопасного кода

Отладочные символы

MDB

Отладочные символы

MONO 5+

Отладочные символы

PDB

Отладочные символы

MDB И PDB





Удалённая отладка

Удалённая отладка. Задача

- Что у нас есть?

Удалённая отладка. Задача

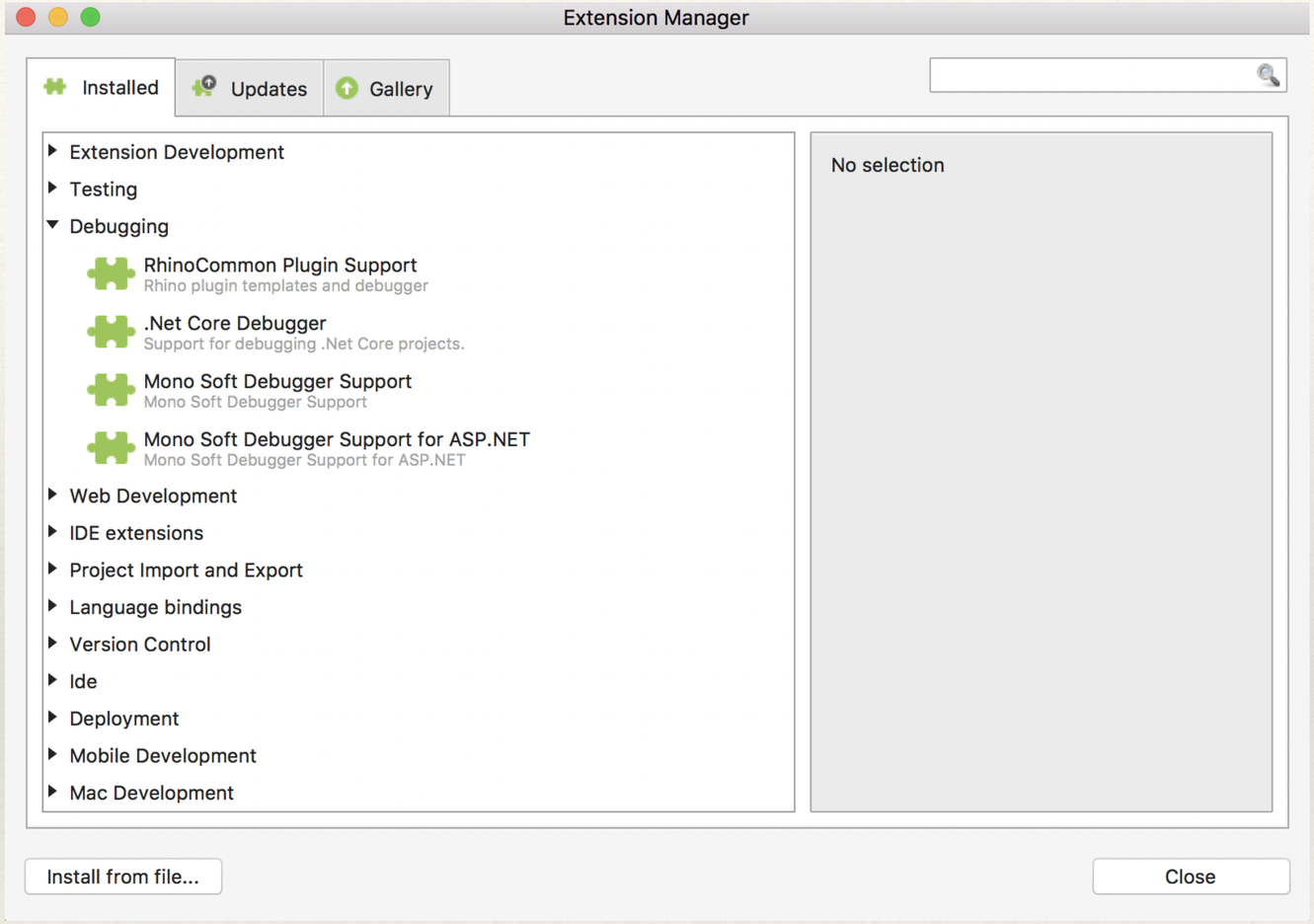
- Что у нас есть?
  - Удалённый сервер

Удалённая отладка. Задача

- Что у нас есть?
  - Удалённый сервер
  - Общая информация об ошибке

Удалённая отладка. Задача

- Что у нас есть?
  - Удалённый сервер
  - Общая информация об ошибке



Удалённая отладка. Инструменты

- Главный инструмент для отладки - Soft Debugger (SDB)

Удалённая отладка. Инструменты

- Главный инструмент для отладки - Soft Debugger (SDB)
  - Встроен в MonoDevelop (Xamarin/VS for Mac)

Удалённая отладка. Инструменты

- Главный инструмент для отладки - Soft Debugger (SDB)
  - Встроен в MonoDevelop (Xamarin/VS for Mac)
  - Может работать из консоли

Удалённая отладка. Инструменты

- Главный инструмент для отладки - Soft Debugger (SDB)
  - Встроен в MonoDevelop (Xamarin/VS for Mac)
  - Может работать из консоли
  - Использует свой протокол Mono.Debugger.Soft.dll

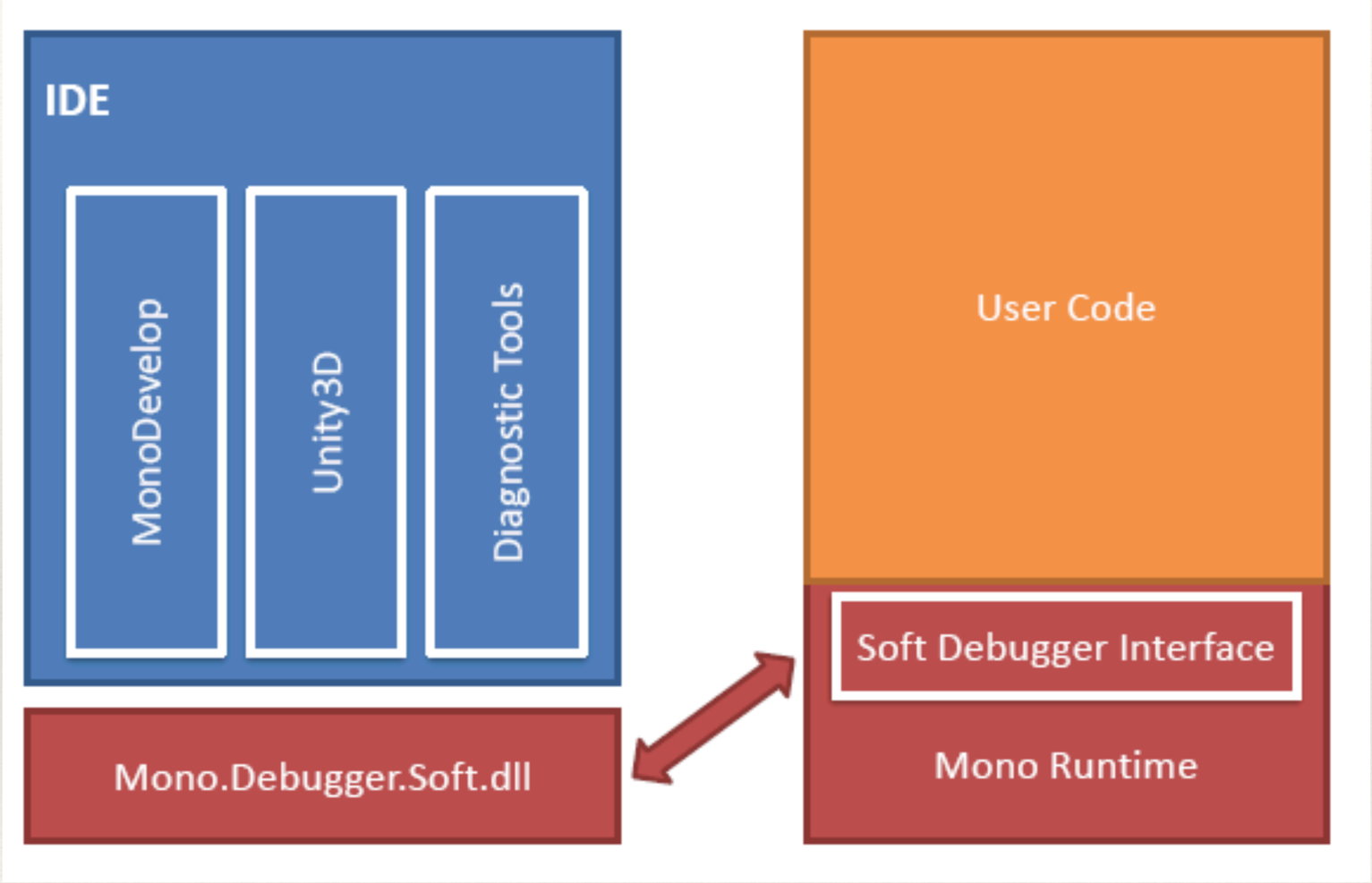
Удалённая отладка. Инструменты

- Главный инструмент для отладки - Soft Debugger (SDB)
  - Встроен в MonoDevelop (Xamarin/VS for Mac)
  - Может работать из консоли
  - Использует свой протокол Mono.Debugger.Soft.dll
  - Не умеет отлаживать unsafe-код

Удалённая отладка. Инструменты

- Главный инструмент для отладки - Soft Debugger (SDB)
  - Встроен в MonoDevelop (Xamarin/VS for Mac)
  - Может работать из консоли
  - Использует свой протокол Mono.Debugger.Soft.dll
  - Не умеет отлаживать unsafe-код
  - Умеет отлаживать удалённые приложения

Удалённая отладка. Инструменты



Удалённая отладка. Решение

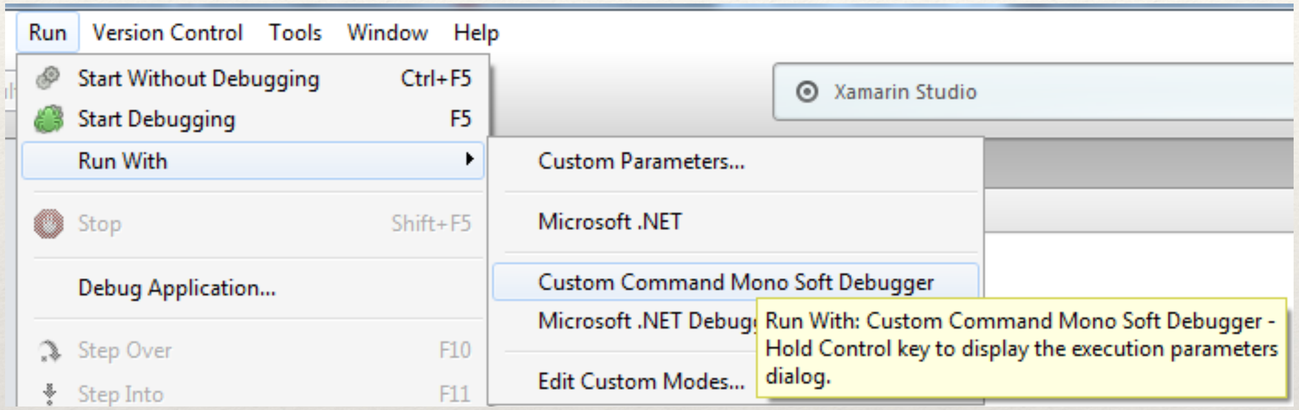
- Запускаем приложение

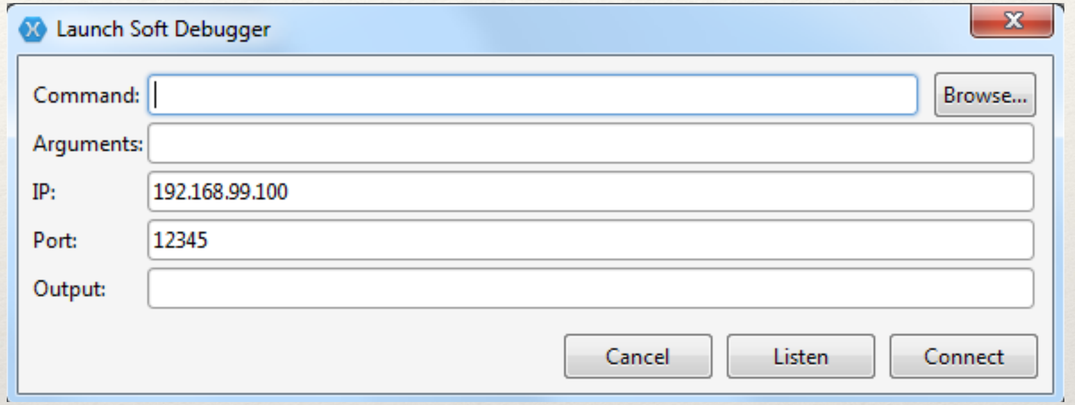
```
mono --debugger-agent=address=192.168.99.100:12345,server=y,transport=dt_socket
--debug ~/Projects/DemoTest/DemoTest/bin/Debug/DemoTest.exe
```

Удалённая отладка. Решение

- Запускаем отладчик

Run -> Run With -> Custom Command Mono Soft Debugger





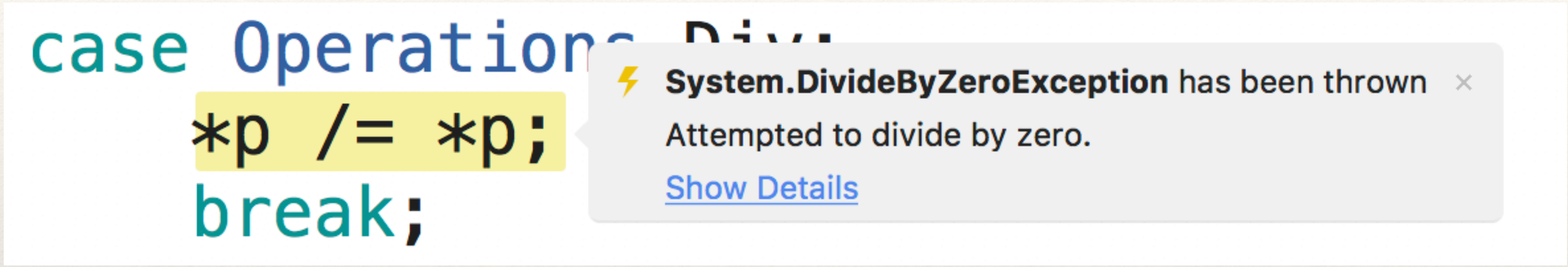
Удалённая отладка. Решение

- Отлаживаем

```
switch(op){
    case Operations.Mul:
        *p *= *p;
        break;
    case Operations.Div:
        *p /= *p;
        break;
```

Удалённая отладка. Решение

- Отлаживаем



Удалённая отладка. Выводы

- Для удалённой отладки необходимо запустить debugger-agent
- Удалённая отладка средствами IDE крайне ограничена
- Для удалённой отладки можно использовать Soft Debugger Console





Отладка unsafe кода

Отладка unsafe-кода. Задача

- Что у нас есть?

Отладка unsafe-кода. Задача

- Что у нас есть?
  - Огромные матрицы

Отладка unsafe-кода. Задача

- Что у нас есть?
  - Огромные матрицы
  - Математические расчёты

Отладка unsafe-кода. Задача

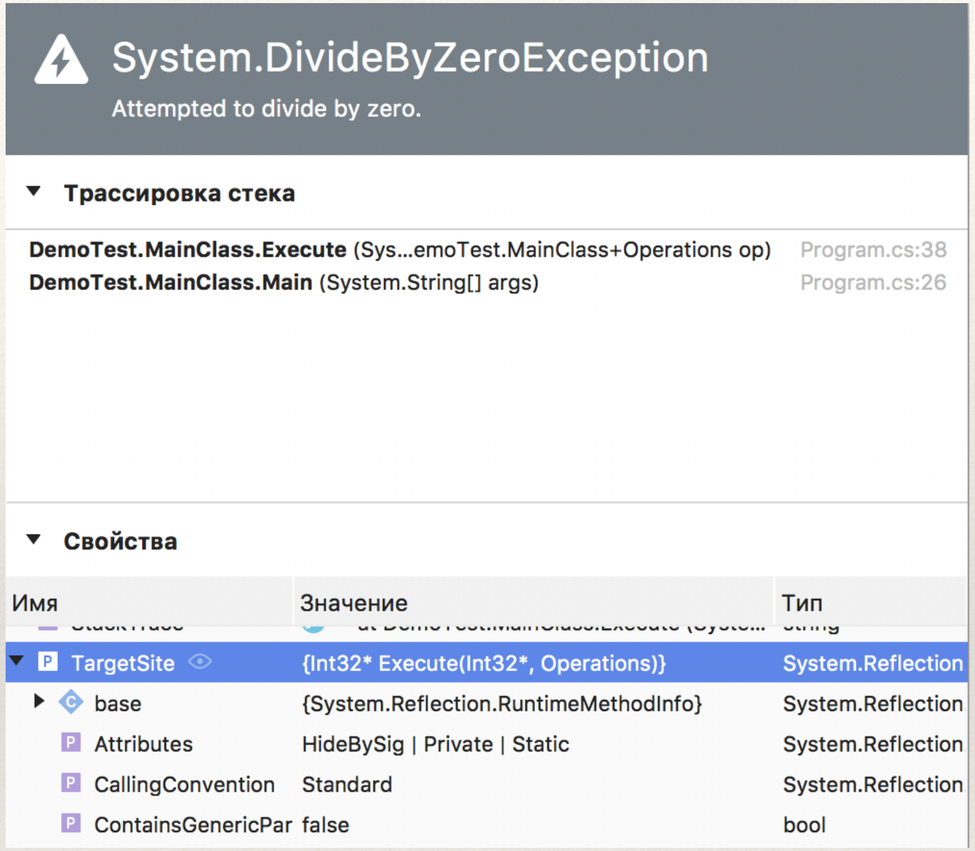
- Что у нас есть?
  - Огромные матрицы
  - Математические расчёты
  - Непонятно откуда приходит ноль

Отладка unsafe-кода. Задача

- Что у нас есть?
  - Огромные матрицы
  - Математические расчёты
  - Непонятно откуда приходит ноль
  - Много ссылок и указателей в unsafe-коде

Отладка unsafe-кода. Задача

- Что у нас есть?
  - Огромные матрицы
  - Математические расчёты
  - Непонятно откуда приходит ноль
  - Много ссылок и указателей в unsafe-коде



Отладка unsafe-кода. Задача

Имя	Значение	Тип
p	0xffffffffbffb94ac	int*
*p	Expression not supported.	

```
⚠ *p = {} Operator * is not supported
⚠ &p = {} Operator & is not supported
⚠ (IntPtr) p = {} Invalid cast.
▶ $exception = {System.DivideByZeroException} {System.DivideByZeroException: Attempted to divide by zero.\n
  op = {DemoTest.MainClass.Operations} Div
  p = {int*} 0x7fff59bdd200
```

Отладка unsafe-кода. Задача

```
unsafe static int* Execute(int* p, Operations op)
{
    switch(op){
        case Operations.Mul:
            *p *= *p;
            break;
        case Operations.Div:
            *p /= *p;
            break;
    }
    MagicWithParams(p);
    return p;
}
```

Отладка unsafe-кода. Инструменты

- Главный инструмент для отладки - Low Level Debugger (LLDB)

Отладка unsafe-кода. Инструменты

- Главный инструмент для отладки - Low Level Debugger (LLDB)
  - Отладчик, отличающийся высокой производительностью

Отладка unsafe-кода. Инструменты

- Главный инструмент для отладки - Low Level Debugger (LLDB)
  - Отладчик, отличающийся высокой производительностью
  - Основан на LLVM

Отладка unsafe-кода. Инструменты

- Главный инструмент для отладки - Low Level Debugger (LLDB)
  - Отладчик, отличающийся высокой производительностью
  - Основан на LLVM
  - Работает на Mac OS как отладчик по умолчанию

Отладка unsafe-кода. Инструменты

- Главный инструмент для отладки - Low Level Debugger (LLDB)
  - Отладчик, отличающийся высокой производительностью
  - Основан на LLVM
  - Работает на Mac OS как отладчик по умолчанию
  - Очень удобен для отладки многопоточных приложений

Отладка unsafe-кода. Инструменты

- Главный инструмент для отладки - Low Level Debugger (LLDB)
  - Отладчик, отличающийся высокой производительностью
  - Основан на LLVM
  - Работает на Mac OS как отладчик по умолчанию
  - Очень удобен для отладки многопоточных приложений
  - Предоставляет возможность просмотра IR

Отладка unsafe-кода. Решение

- Запускаем mono

elizaveta — lldb /Library/Frameworks/Mono.framework/Versions/Current/bin/mono ▸ mono — 99×16

```
(lldb) call mono_method_get_class((MonoMethod *)0x1005e3ba9)
(MonoClass *) $6 = 0x481024448b480000
(lldb) q
Quitting LLDB will kill one or more processes. Do you really want to proceed: [Y/n] Y
iMac-Elizaveta:~ elizaveta$ lldb /Library/Frameworks/Mono.framework/Versions/Current/bin/mono
"monobt" command installed
(lldb) target create "/Library/Frameworks/Mono.framework/Versions/Current/bin/mono"
Current executable set to '/Library/Frameworks/Mono.framework/Versions/Current/bin/mono' (x86_64).
(lldb) run --break 'DemoTest.MainClass:Execute'  ~/Projects/DemoTest/DemoTest/bin/Debug/DemoTest.ex
e
Process 49194 launched: '/Library/Frameworks/Mono.framework/Versions/Current/bin/mono' (x86_64)
warning: (x86_64) /Library/Frameworks/Mono.framework/Versions/5.4.0/lib/mono/4.5/mscorlib.dll.dylib
 empty dSYM file detected, dSYM was created with an executable with no debug info.
Start process 49194
45
Process 49194 stopped
```

Отладка unsafe-кода. Решение

- Запускаем приложение и ставим breakpoint

elizaveta — lldb /Library/Frameworks/Mono.framework/Versions/Current/bin/mono ▸ mono — 100×16

```
iMac-Elizaveta:~ elizaveta$ lldb /Library/Frameworks/Mono.framework/Versions/Current/bin/mono
"monobt" command installed
(lldb) target create "/Library/Frameworks/Mono.framework/Versions/Current/bin/mono"
Current executable set to '/Library/Frameworks/Mono.framework/Versions/Current/bin/mono' (x86_64).
(lldb) run --break 'DemoTest.MainClass:Execute'  ~/Projects/DemoTest/DemoTest/bin/Debug/DemoTest.exe

Process 49194 launched: '/Library/Frameworks/Mono.framework/Versions/Current/bin/mono' (x86_64)
warning: (x86_64) /Library/Frameworks/Mono.framework/Versions/5.4.0/lib/mono/4.5/mscorlib.dll.dylib
empty dSYM file detected, dSYM was created with an executable with no debug info.
Start process 49194
45
Process 49194 stopped
* thread #1, name = 'tid_307', queue = 'com.apple.main-thread', stop reason = EXC_BREAKPOINT (code=E
XC_I386_BPT, subcode=0x0)
    frame #0: 0x00000001005dfba4
->  0x1005dfba4: movq   %r15, 0x10(%rsp)
```

Отладка unsafe-кода. Решение

- Отработал breakpoint

elizaveta — lldb /Library/Frameworks/Mono.framework/Versions/Current/bin/mono ▸ mono — 99×16

```
    frame #9: 0x0000000100002529 mono`main(argc=4, argv=<unavailable>) at main.c:339 [opt]
    frame #10: 0x0000000100002434 mono`start + 52
(lldb) thread list
Process 49194 stopped
* thread #1: tid = 0x99b889, 0x00000001005dfba4, name = 'tid_307', queue = 'com.apple.main-thread',
 stop reason = EXC_BREAKPOINT (code=EXC_I386_BPT, subcode=0x0)
  thread #2: tid = 0x99b89e, 0x00007fffbe587bf2 libsystem_kernel.dylib`__psynch_cvwait + 10, name =
 'SGen worker'
  thread #3: tid = 0x99b89f, 0x00007fffbe587bf2 libsystem_kernel.dylib`__psynch_cvwait + 10, name =
 'SGen worker'
  thread #4: tid = 0x99b8a0, 0x00007fffbe580386 libsystem_kernel.dylib`semaphore_wait_trap + 10, na
me = 'Finalizer'
  thread #5: tid = 0x99b8a1, 0x00007fffbe58844e libsystem_kernel.dylib`__workq_kernreturn + 10
  thread #6: tid = 0x99b8a2, 0x00007fffbe58844e libsystem_kernel.dylib`__workq_kernreturn + 10
  thread #7: tid = 0x99b8a3, 0x00007fffbe58844e libsystem_kernel.dylib`__workq_kernreturn + 10
(lldb)
```

Отладка unsafe-кода. Решение

- *ЛИБО!!!* присоединяемся к процессу

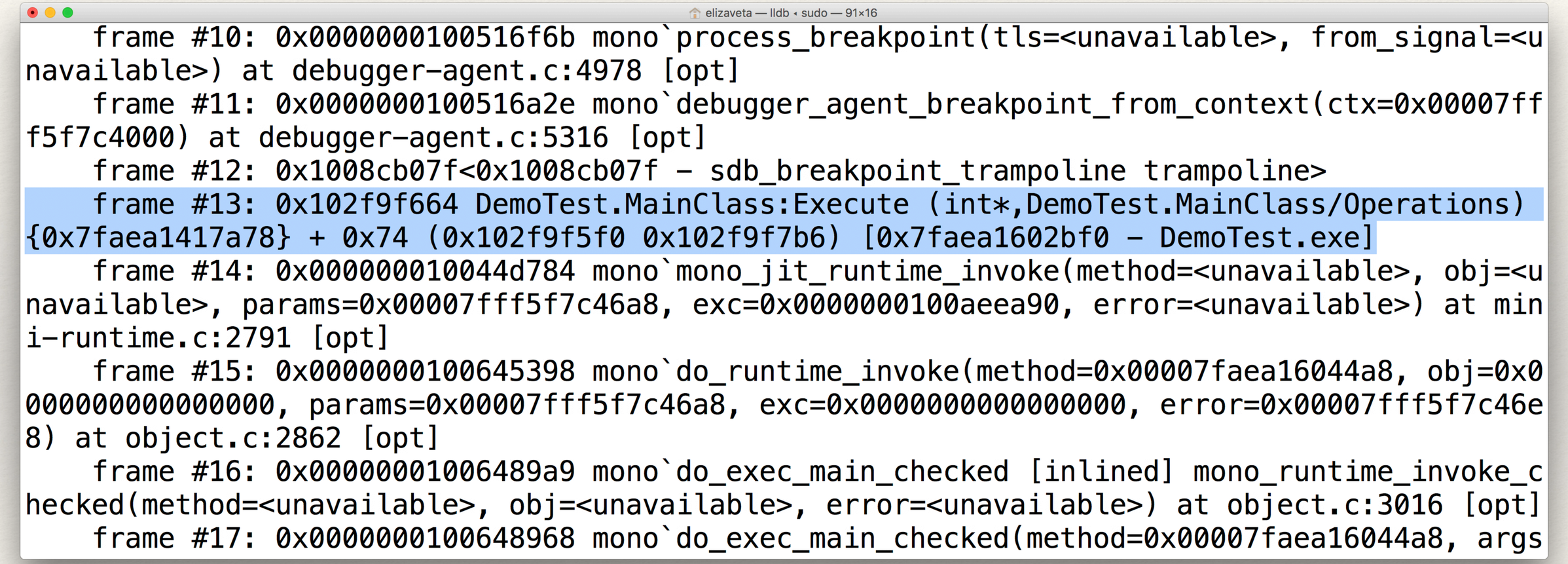
elizaveta — lldb ◂ sudo — 91×16

```
"monobt" command installed
(lldb) process attach --pid 49918
Process 49918 stopped
* thread #1, name = 'tid_307', queue = 'com.apple.main-thread', stop reason = signal SIGSTO
P
    frame #0: 0x00007fffbe589246 libsystem_kernel.dylib`read + 10
libsystem_kernel.dylib`read:
->  0x7fffbe589246 <+10>: jae    0x7fffbe589250            ; <+20>
    0x7fffbe589248 <+12>: movq   %rax, %rdi
    0x7fffbe58924b <+15>: jmp    0x7fffbe580cd4            ; cerror
    0x7fffbe589250 <+20>: retq
Target 0: (mono) stopped.

Executable module set to "/Library/Frameworks/Mono.framework/Versions/Current/bin/mono".
Architecture set to: x86_64h-apple-macosx.
(lldb) c
```

Отладка unsafe-кода. Решение

- Проверяем backtrace: monobt



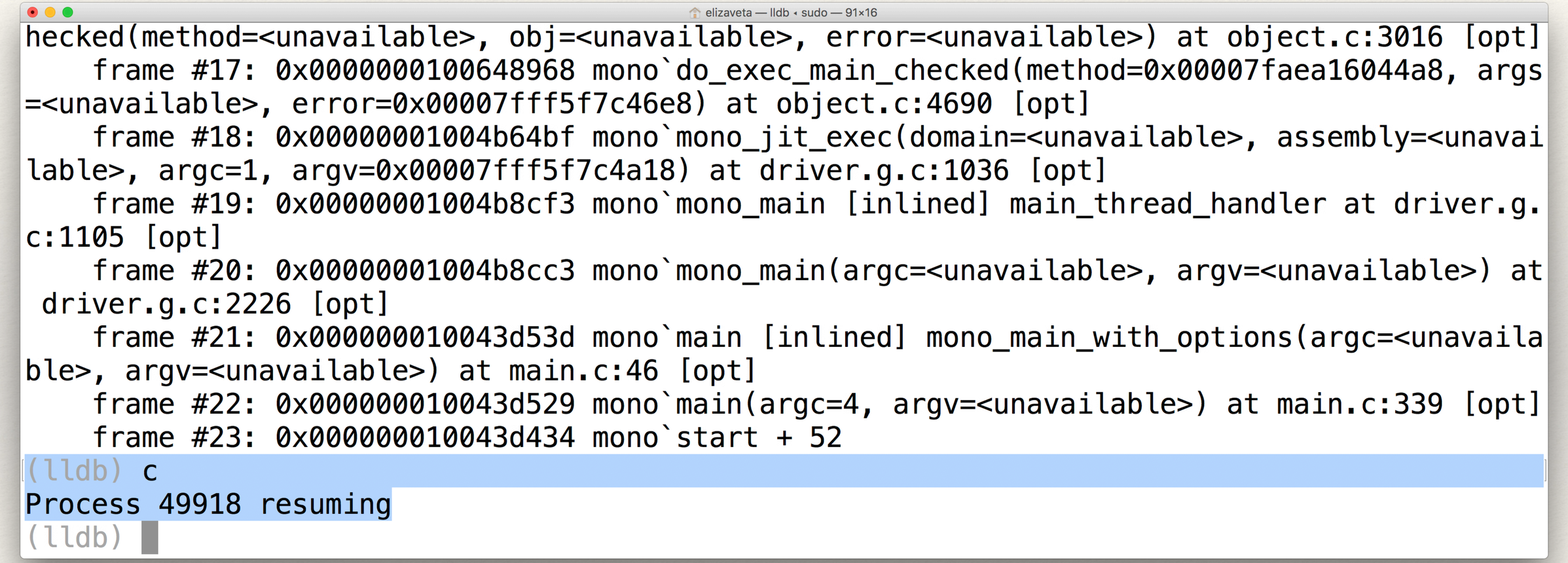
Отладка unsafe-кода. Решение

- Ставим breakpoint

```
b 0x102f9f664
```

Отладка unsafe-кода. Решение

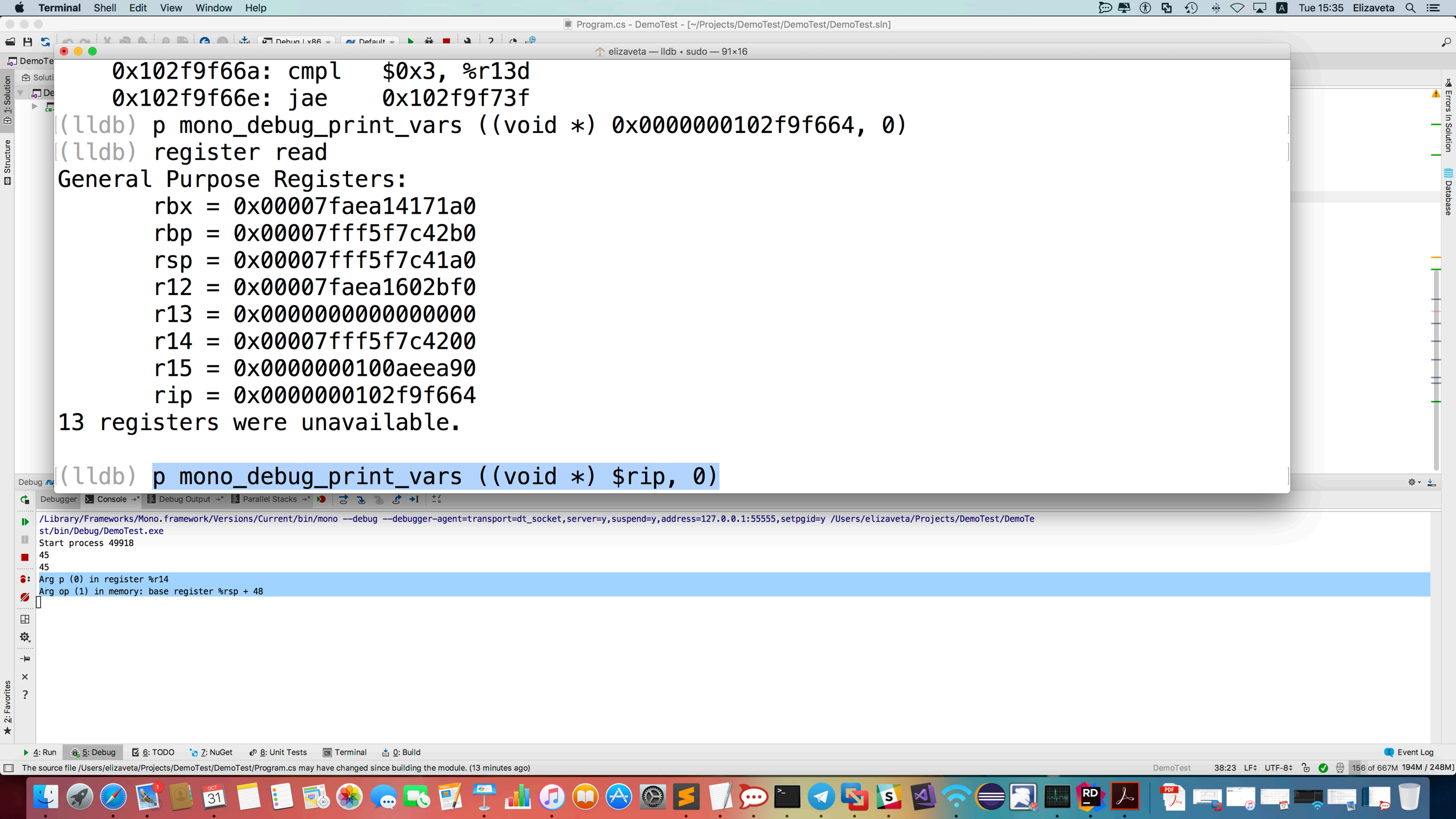
- Продолжаем отладку



Отладка unsafe-кода. Решение

- Проверяем входные параметры

```
(lldb) p mono_debug_print_vars ((void *) $rip, 0)
```


Terminal Shell Edit View Window Help
Tue 15:35 Elizaveta
Program.cs - DemoTest - [~/Projects/DemoTest/DemoTest/DemoTest.sln]
elizaveta — lldb ◂ sudo — 91×16
0x102f9f66a: cmpl $0x3, %r13d
0x102f9f66e: jae 0x102f9f73f
(lldb) p mono_debug_print_vars ((void *) 0x0000000102f9f664, 0)
(lldb) register read
General Purpose Registers:
rbx = 0x00007faea14171a0
rbp = 0x00007fff5f7c42b0
rsp = 0x00007fff5f7c41a0
r12 = 0x00007faea1602bf0
r13 = 0x0000000000000000
r14 = 0x00007fff5f7c4200
r15 = 0x0000000100aeea90
rip = 0x0000000102f9f664
13 registers were unavailable.
(lldb) p mono_debug_print_vars ((void *) $rip, 0)
Debugger Console Debug Output Parallel Stacks
/Library/Frameworks/Mono.framework/Versions/Current/bin/mono --debug --debugger-agent=transport=dt_socket,server=y,suspend=y,address=127.0.0.1:55555,setpgid=y /Users/elizaveta/Projects/DemoTest/DemoTest/bin/Debug/DemoTest.exe
Start process 49918
45
45
Arg p (0) in register %r14
Arg op (1) in memory: base register %rsp + 48
4: Run 5: Debug 6: TODO 7: NuGet 8: Unit Tests Terminal 0: Build
Event Log
The source file /Users/elizaveta/Projects/DemoTest/DemoTest/Program.cs may have changed since building the module. (13 minutes ago)
DemoTest 38:23 LF UTF-8 156 of 667M 194M / 248M

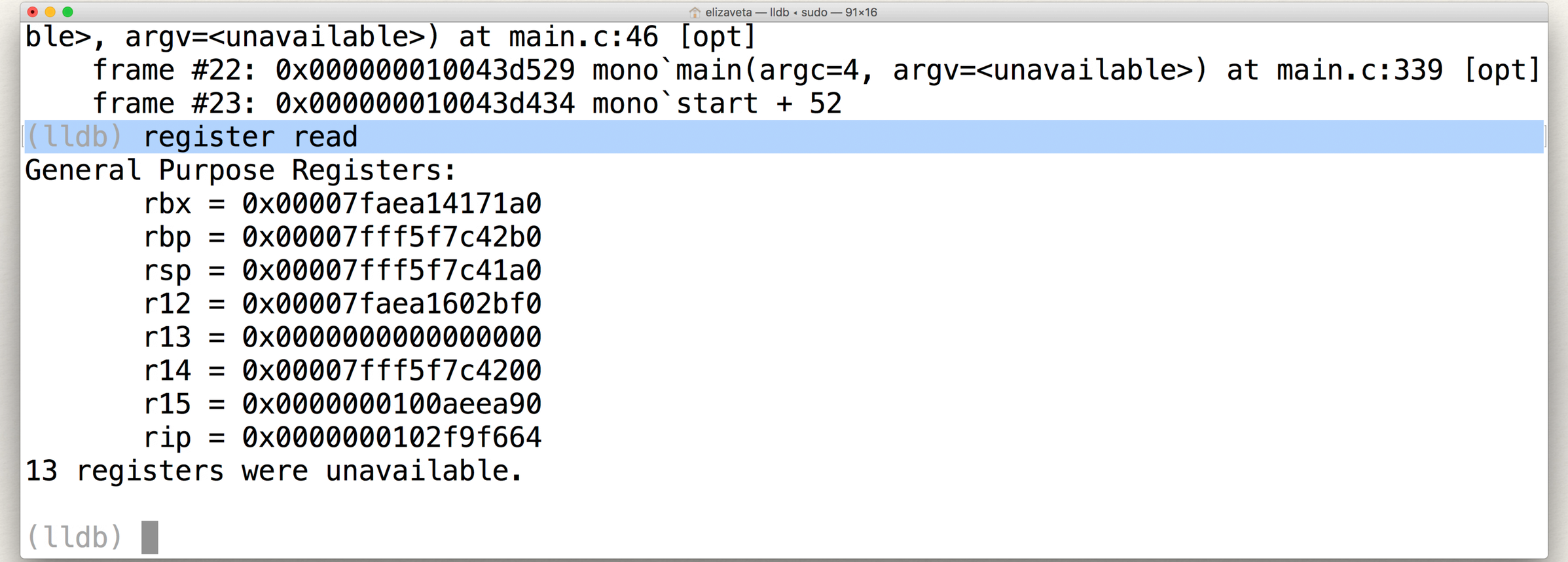
Отладка unsafe-кода. Решение

- Проверяем входные параметры

```
Arg p (0) in register %r14
Arg op (1) in memory: base register %rsp + 48
```

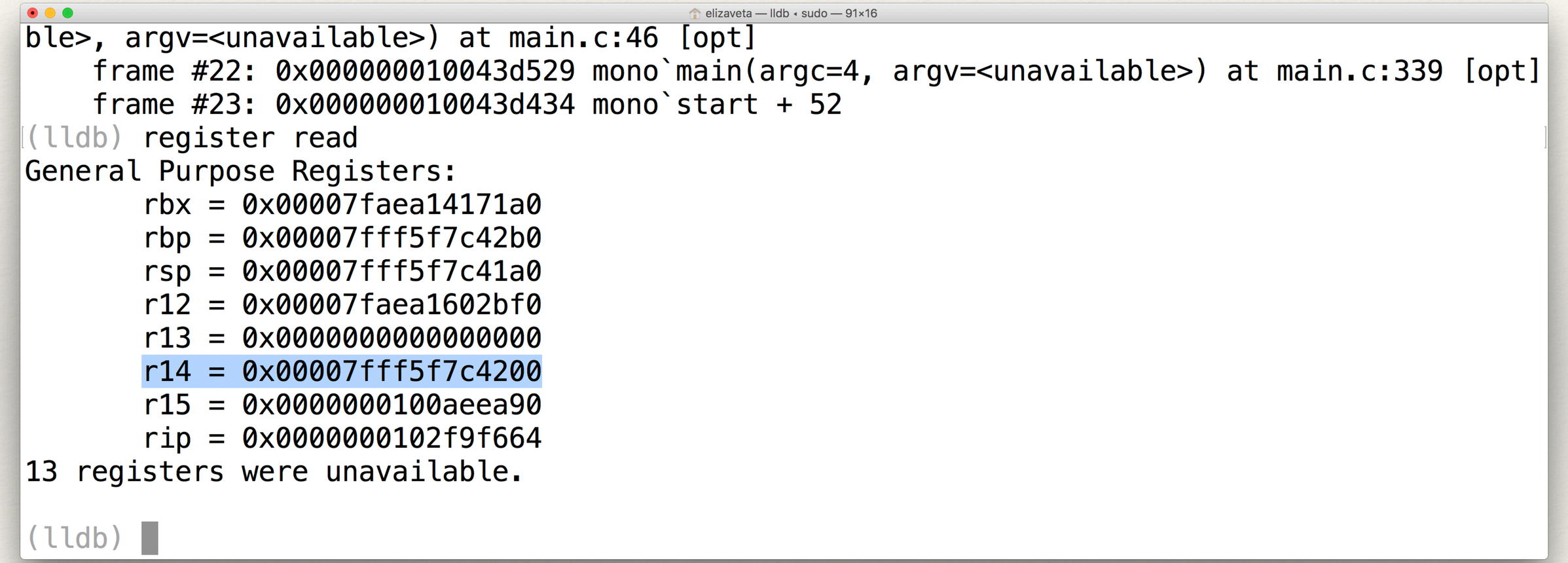
Отладка unsafe-кода. Решение

- Что скрывают эти регистры? Посмотрим?



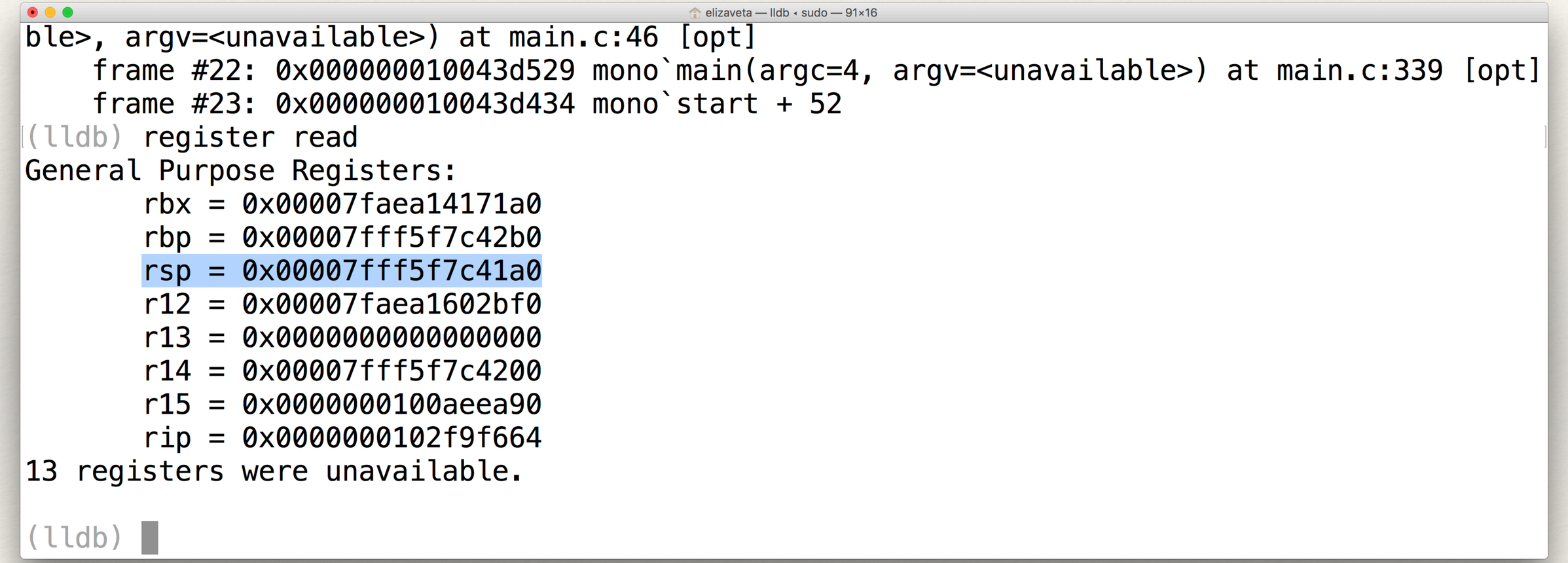
Отладка unsafe-кода. Решение

- Что скрывают эти регистры? Посмотрим? Arg p(0)=r14



Отладка unsafe-кода. Решение

- Что скрывают эти регистры? Посмотрим? Arg op (1) = rsp + 48



Отладка unsafe-кода. Решение

- Проверяем входные параметры

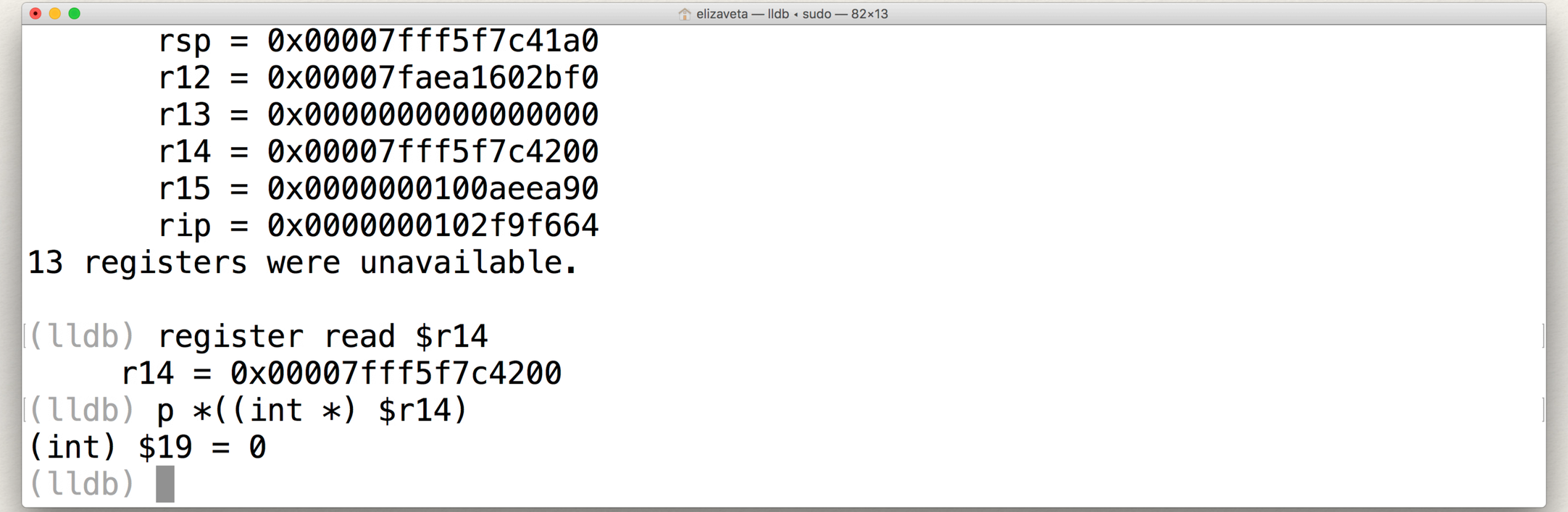
```
Arg p (0) in register %r14
Arg op (1) in memory: base register %rsp + 48
```

Отладка unsafe-кода. Решение

- Нас интересует регистр r14

Отладка unsafe-кода. Решение

- Нас интересует регистр r14
- Выведем его значение:



Отладка unsafe-кода. Выводы

- Soft Debugger не умеет отлаживать unsafe

Отладка unsafe-кода. Выводы

- Soft Debugger не умеет отлаживать unsafe
- IDE позволяет отлаживать unsafe-код с помощью GDB/LLDB

Отладка unsafe-кода. Выводы

- Soft Debugger не умеет отлаживать unsafe
- IDE позволяет отлаживать unsafe-код с помощью GDB/LLDB
- Но!!! Максимум, что Вы увидите - АДРЕС

Отладка unsafe-кода. Выводы

- Soft Debugger не умеет отлаживать unsafe
- IDE позволяет отлаживать unsafe-код с помощью GDB/LLDB
- Но!!! Максимум, что Вы увидите - АДРЕС
- Хотите увидеть больше?

Отладка unsafe-кода. Выводы

- Soft Debugger не умеет отлаживать unsafe
- IDE позволяет отлаживать unsafe-код с помощью GDB/LLDB
- Но!!! Максимум, что Вы увидите - АДРЕС
- Хотите увидеть больше?
  - Используйте GDB или LLDB





Отладочные символы Mono native crash

Отладочные символы. Задача

- Не работает soft debugger

Отладочные символы. Задача

- Не работает soft debugger
- В процессе работы GC происходит native crash

Отладочные символы. Задача

- Не работает soft debugger
- В процессе работы GC происходит native crash
- Что делать?

Отладочные символы. Инструменты

- Главный инструмент для отладки - GNU Debugger (GDB)

Отладочные символы. Инструменты

- Главный инструмент для отладки - GNU Debugger (GDB)
  - Стандартный GNU-отладчик

Отладочные символы. Инструменты

- Главный инструмент для отладки - GNU Debugger (GDB)
  - Стандартный GNU-отладчик
  - Есть поддержка mono

Отладочные символы. Инструменты

- Главный инструмент для отладки - GNU Debugger (GDB)
  - Стандартный GNU-отладчик
  - Есть поддержка mono
  - Можно останавливать отдельный thread, другие thread будут продолжать работать

Отладочные символы. Решение

- Присоединяемся к процессу

```
gdb -p 77123

GNU gdb (GDB) 8.0
Copyright (C) 2017 Free Software Foundation, Inc.
……..
This GDB was configured as "x86_64-apple-darwin16.6.0".
Type "show configuration" for configuration details.
……..
Attaching to process 77123
……..
Reading symbols from /usr/local/bin/mono-sgen…
done.
warning: unhandled dyld version (15)
0x00007fffb959c563 in ?? ()
Mono support loaded.
```

Отладочные символы. Решение

- Проверяем backtrace

```
(gdb) where
#0 0x00007fffb959c563 in ?? ()
#1 0x000000010ec39000 in ?? ()
#2 0x0000000000000008 in ?? ()
#3 0x000000010ec3ca00 in ?? ()
#4 0x000000010ec3de00 in ?? ()
#5 0x00007fff51488bd0 in ?? ()
#6 0x00007fffb96a3472 in ?? ()
#7 0x0aa69892cc260046 in ?? ()
#8 0x0aa69892cc260046 in ?? ()
```

Отладочные символы. Решение

- Символы загружены корректно



Отладочные символы. Решение

- Символы загружены корректно
- Но почему же дебаг так и не работает при отрабатывании GC???





Отладочные символы. Решение

- Что скрывается под этими адресами?
  - call mono_pmip(0x00007fffb95c6423)



Отладочные символы. Решение

- Запускается новое окно терминала! И мы там видим..



Отладочные символы. Решение

LLDB

Отладочные символы. Решение

- GDB и call-вызовы в Mac OS = обертка над LLDB
- Ошибки от приложения перед вызовом lldb
- Native stacktrace:

```
0 mono 0x000000010e828eb1 mono_handle_native_crash + 257
1 mono 0x000000010e890086 altstack_handle_and_restore + 70
2 ??? 0x000000010000a970 0x0 + 4295010672
```

Отладочные символы. Решение

- На старт, внимание…

```
Debug info from gdb:
(lldb) command source -s 0 '/tmp/mono-gdb-commands.ajX1c8'
Executing commands in '/tmp/mono-gdb-commands.ajX1c8'.
(lldb) process attach --pid 77123
error: libmini-static.a(libmini_static_la-mini-exceptions.o) failed to load objfile for
/Users/elizaveta/mono/mono-5.4.0.135/mono/mini/./.libs/libmini-static.a
error: libmini-static.a(libmini_static_la-exceptions-amd64.o) failed to load objfile for
/Users/elizaveta/mono/mono-5.4.0.135/mono/mini/./.libs/libmini-static.a
Process 77123 stopped
```

Отладочные символы. Выводы

- Для того, чтобы понять, почему произошёл native crash, используйте GDB или LLDB

Отладочные символы. Выводы

- Для того, чтобы понять, почему произошёл native crash, используйте GDB или LLDB
- Не используйте GDB для отладки mono под Mac OS





Deadlock. Задача 1
lock(object)

Deadlock. Задача



- Что у нас есть?

Deadlock. Задача



- Что у нас есть?
  - Зависшая программа на удалённой машине

Deadlock. Задача



- Что у нас есть?
  - Зависшая программа на удалённой машине
  - Нет дампов

Deadlock. Задача



- Что у нас есть?
  - Зависшая программа на удалённой машине
  - Нет дампов
  - Нет доступа к серверу

Deadlock. Задача



- Что у нас есть?
  - Зависшая программа на удалённой машине
  - Нет дампов
  - Нет доступа к серверу
  - Ubuntu Desktop

Deadlock. Задача



Deadlock. Задача



АНАЛИЗИРОВАТЬ КОД

63

Deadlock. Инструменты

- Основной инструмент - Valgrind

Deadlock. Инструменты

- Основной инструмент - Valgrind
  - Динамический анализатор проблем с памятью

Deadlock. Инструменты

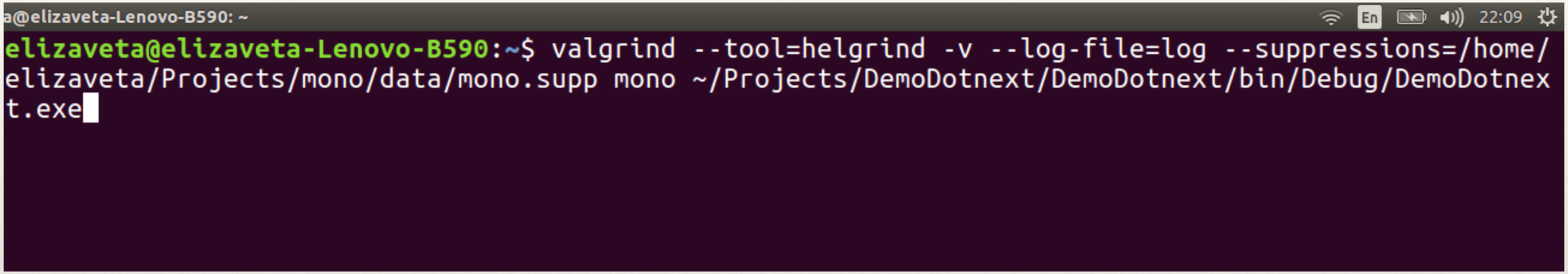
- Основной инструмент - Valgrind
  - Динамический анализатор проблем с памятью
  - Работает с IR

Deadlock. Инструменты

- Основной инструмент - Valgrind
  - Динамический анализатор проблем с памятью
  - Работает с IR
  - Представляет собой совокупность инструментов: memcheck, massif, helgrind, cachegrind, callgrind, SGcheck

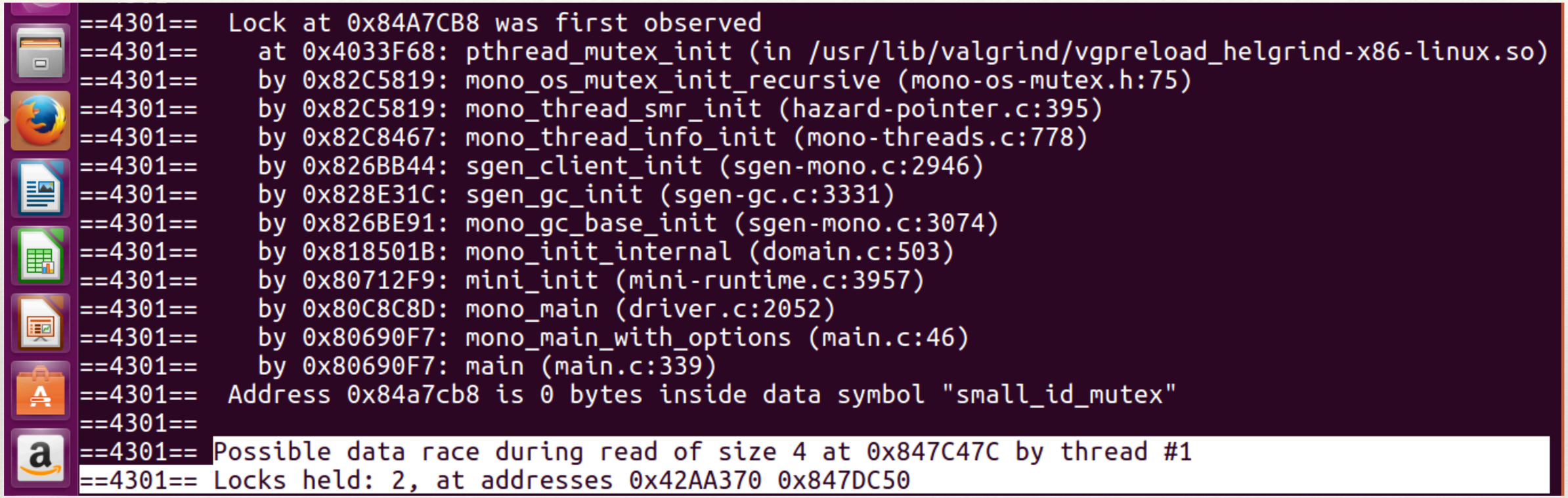
Deadlock. Решение

- Для анализа дедлоков будем использовать helgrind



Deadlock. Решение

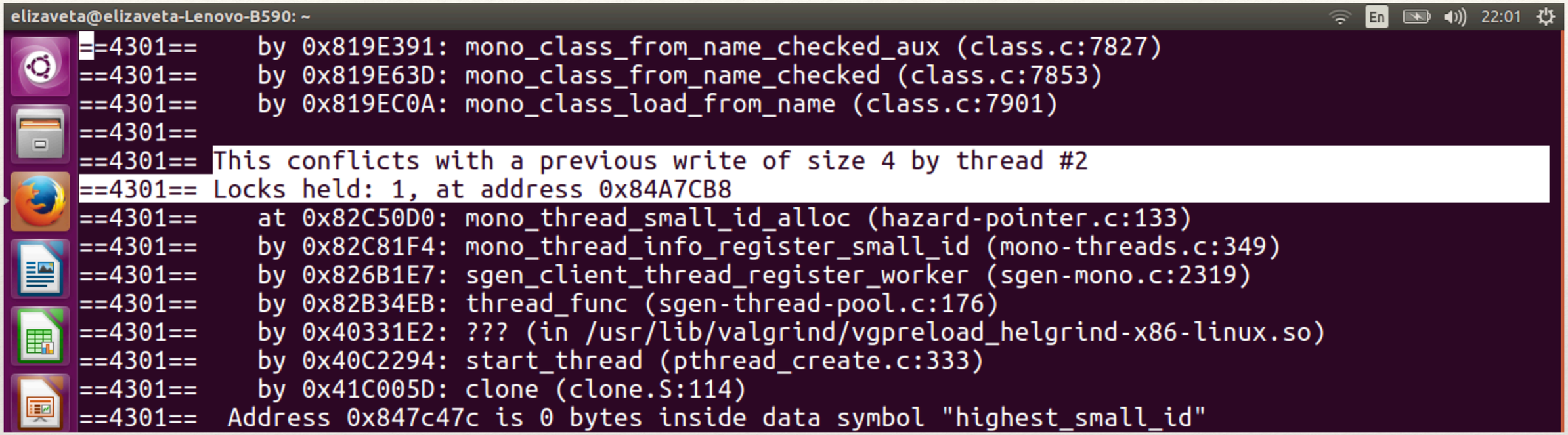
- Открываем файл с логами



```
==4301==  Lock at 0x84A7CB8 was first observed
==4301==    at 0x4033F68: pthread_mutex_init (in /usr/lib/valgrind/vgpreload_helgrind-x86-linux.so)
==4301==    by 0x82C5819: mono_os_mutex_init_recursive (mono-os-mutex.h:75)
==4301==    by 0x82C5819: mono_thread_smr_init (hazard-pointer.c:395)
==4301==    by 0x82C8467: mono_thread_info_init (mono-threads.c:778)
==4301==    by 0x826BB44: sgen_client_init (sgen-mono.c:2946)
==4301==    by 0x828E31C: sgen_gc_init (sgen-gc.c:3331)
==4301==    by 0x826BE91: mono_gc_base_init (sgen-mono.c:3074)
==4301==    by 0x818501B: mono_init_internal (domain.c:503)
==4301==    by 0x80712F9: mini_init (mini-runtime.c:3957)
==4301==    by 0x80C8C8D: mono_main (driver.c:2052)
==4301==    by 0x80690F7: mono_main_with_options (main.c:46)
==4301==    by 0x80690F7: main (main.c:339)
==4301==  Address 0x84a7cb8 is 0 bytes inside data symbol "small_id_mutex"
==4301==
==4301== Possible data race during read of size 4 at 0x847C47C by thread #1
==4301== Locks held: 2, at addresses 0x42AA370 0x847DC50
```

Deadlock. Решение

- Открываем файл с логами



Deadlock. Решение

- Открываем файл с логами

```
==4301==    by 0x80F8655: mini_jit_info_table_find (mini-exceptions.c:1396)
==4301==    by 0x80FA163: mini_add_method_trampoline (mini-trampolines.c:313)
==4301==    by 0x80FA8D3: common_call_trampoline (mini-trampolines.c:720)
==4301==    by 0x80FAED5: mono_magic_trampoline (mini-trampolines.c:839)
==4301==    by 0x403F187: ???
==4301==    by 0x43BDD2F: ???
==4301==  Address 0x84a8570 is 80 bytes inside data symbol "mono_stats"
==4301==
--4301--
--4301-- used_suppression:   1629 helgrind-glibc2X-005 /usr/lib/valgrind/default.supp:945
==4301==
==4301== ERROR SUMMARY: 57 errors from 17 contexts (suppressed: 1629 from 253)
```

Deadlock. Решение

- Если вы собираете mono сами, то можете воспользоваться инструментами для анализа кода/памяти Clang

Deadlock. Решение

- Если вы собираете mono сами, то можете воспользоваться инструментами для анализа кода/памяти Clang
- ThreadSanitizer (data races) интегрирован в Clang

Deadlock. Решение

- Если вы собираете mono сами, то можете воспользоваться инструментами для анализа кода/памяти Clang
- ThreadSanitizer (data races) интегрирован в Clang
- Быстрее в 4-6 раз, чем valgrind (8мин VS 35 мин)

Deadlock. Решение

- Если вы собираете mono сами, то можете воспользоваться инструментами для анализа кода/памяти Clang
- ThreadSanitizer (data races) интегрирован в Clang
- Быстрее в 4-6 раз, чем valgrind (8мин VS 35 мин)

- ./autogen.sh —prefix=/usr/local CC=clang CFLAGS=-fsanitize=thread LDFLAGS=-fsanitize=thread CXX=clang





Что делать?

Deadlock. Решение

- Если у нас есть *логи* - мы счастливые *люди*

Deadlock. Решение

- Если у нас есть логи - мы счастливые люди
- Mono Log Profiler - профилировщик mono

Deadlock. Решение

- Если у нас есть логи - мы счастливые люди
- Mono Log Profiler - профилировщик mono
  - Все данные записываются в файл
  - События, которые регистрируются в файле
    - Вход/выход из метода
    - Распределение объектов в памяти
    - Сборка мусора
    - JIT-компиляция и др.

Deadlock. Решение

- Для анализа используем логи профайлера

```
ENTER: TestDeadlock:normal_order ()([0xb49ffb40: 0,04048 1] LEAVE: (wrapper write-barrier) object:wbarrier_noconc (intptr))
……..
[0xb4c1ab40: 0,04069 2] LEAVE: (wrapper synchronized) System.IO.TextWriter/SyncTextWriter:WriteLine (string)
[0xb49ffb40: 0,04071 1] ENTER: TestDeadlock:reverse_order ()()
……..
[0xb49ffb40: 0,54096 2] LEAVE: (wrapper managed-to-native) System.Threading.Thread:SleepInternal (int)
```

Deadlock. Решение

- Переходим к отладчику - SDB
  - 1. Необходимо запустить агент отладчика, чтобы можно было подключиться удаленно

```
Inferior process '2523' ('DemoDotnext.exe') suspended
#0 [0x00000026] TestDeadlock.reverse_order at
/home/elizaveta/Projects/DemoDotnext/DemoDotnext/Program.cs:24
lock (lockA) {
```

Deadlock. Решение

- Переходим к отладчику - SDB
  - 2. Проверяем backtrace

```
(sdb) bt
#0 [0x00000026] TestDeadlock.reverse_order at /home/elizaveta/Projects/DemoDotnext/DemoDotnext/
Program.cs:24
lock (lockA) {
#1 [0x00000022] System.Threading.ThreadHelper.ThreadStart_Context at /build/mono-oWqW1z/mono-
4.2.1.102+dfsg2/external/referencesource/mscorlib/system/threading/thread.cs:68
#2 [0x00000083] System.Threading.ExecutionContext.RunInternal at /build/mono-oWqW1z/mono-
4.2.1.102+dfsg2/external/referencesource/mscorlib/system/threading/executioncontext.cs:581
```

Deadlock. Решение

- Переходим к отладчику - SDB
  - 3. Проверяем состояние threads:

```
(sdb) thread list
Thread #3 ''
[0x00000026] TestDeadlock.reverse_order at /home/elizaveta/Projects/DemoDotnext/DemoDotnext/Program.cs:
24
lock (lockA) {
Thread #4 'Finalizer'
Thread #1 ''
[0x00000032] System.Threading.Thread.JoinInternal
Thread #2 ''
[0x00000026] TestDeadlock.normal_order at /home/elizaveta/Projects/DemoDotnext/DemoDotnext/Program.cs:14
lock (lockB) {
```

Deadlock. Выводы

- Valgrind & Thread Sanitizer удобно использовать для поиска дедлоков в unsafe или P/Invoke - коде

Deadlock. Выводы

- Valgrind & Thread Sanitizer удобно использовать для поиска дедлоков в unsafe или P/Invoke - коде
- Для решения простых проблем с дедлоками поможет Mono Log Profiler

Deadlock. Выводы

- Valgrind & Thread Sanitizer удобно использовать для поиска дедлоков в unsafe или P/Invoke - коде
- Для решения простых проблем с дедлоками поможет Mono Log Profiler
- Что делать, если нет логов?





Deadlock. Задача 2 AsParallel и ForEach

Deadlock. Задача и инструменты

- Что у нас есть?

Deadlock. Задача и инструменты

- Что у нас есть?
  - Зависшая программа на удалённой машине

Deadlock. Задача и инструменты

- Что у нас есть?
  - Зависшая программа на удалённой машине
  - Дамп памяти процесса

Deadlock. Задача и инструменты

- Что у нас есть?
  - Зависшая программа на удалённой машине
  - Дамп памяти процесса
  - LLDB и отладочная информация mono

Deadlock. Решение

- Делаем дамп ядра зависшего процесса

```
lldb -p 3737
process save-core.3737
```

Deadlock. Решение

- **Работаем по инструкции**

https://youtu.be/cYqN8-T695s



Deadlock. Решение

- **Исследуем thread**

https://youtu.be/R141B3Dnc4M



Deadlock. Решение

- А вот и код!

```
public void Execute(ManualResetEventSlim mre){
    Enumerable.Range(0, Environment.ProcessorCount * 100).AsParallel().ForAll((j) =>
    {
        if (j == Environment.ProcessorCount)
        {
            Console.WriteLine("Set on {0} with value of {1}", Thread.CurrentThread.ManagedThreadId, j);
            mre.Set();
        }
        else
        {
            Console.WriteLine("Waiting on {0} with value of {1}", Thread.CurrentThread.ManagedThreadId, j);
            mre.Wait();
        }
    });
}
```

Deadlock. Выводы

- LLDB и GDB позволяют работать с дампами памяти

Deadlock. Выводы

- LLDB и GDB позволяют работать с дампами памяти
- LLDB и GDB с помощью mono embedding позволяют оперативно найти объекты, попавшие в дедлок

Deadlock. Выводы

- LLDB и GDB позволяют работать с дампами памяти
- LLDB и GDB с помощью mono embedding позволяют оперативно найти объекты, попавшие в дедлок
- Но бывают ситуации, где мы вынуждены вручную анализировать backtraces





Утечки памяти

Утечки памяти. Задача

- Что у нас есть?

Утечки памяти. Задача

- Что у нас есть?
  - Старый код, работает на .Net 4.6.1

Утечки памяти. Задача

- Что у нас есть?
  - Старый код, работает на .Net 4.6.1
  - Не работает на mono

Утечки памяти. Задача

- Что у нас есть?
  - Старый код, работает на .Net 4.6.1
  - Не работает на mono
  - Вызывает OutOfMemoryException

Утечки памяти. Решение

- А вот и код

```
public class ByteReader
{
    byte[] bf;
    public ByteReader()
    {
        bf = new byte[1024 * 1024 * 100];//100MB
    }

    //todo: read data
    public byte[] Read(){
        return bf;
    }
}
```

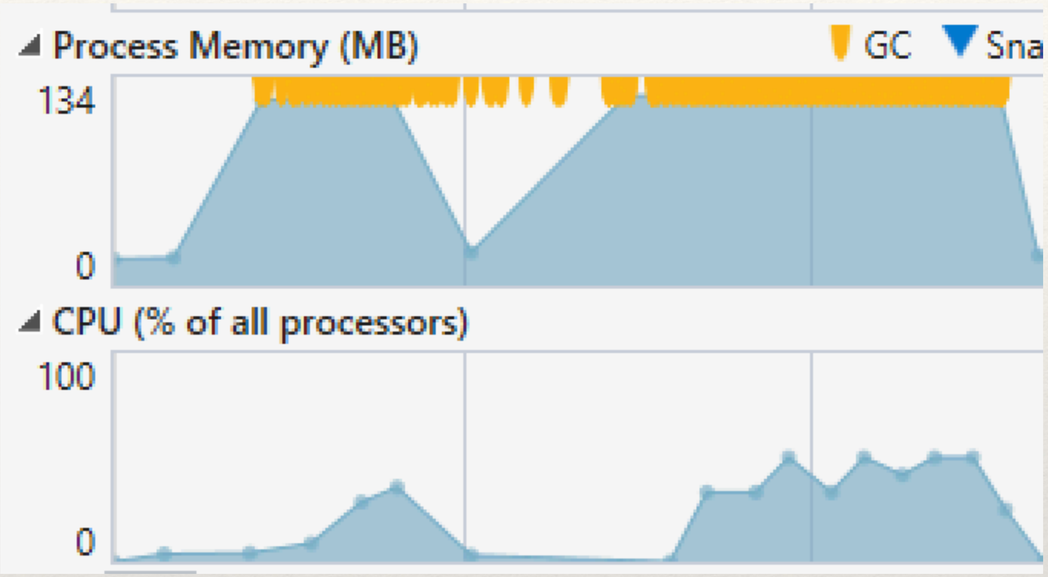
Утечки памяти. Решение

- А вот и код

```
public static void Execute()
{
    if (ParamA == ParamB)
    {
        ByteReader byteReader = new ByteReader();
        byteReader.Read();
    }
}
```

Утечки памяти. Решение

- Под .Net Framework



Утечки памяти. Решение

- Под mono

```
21              byte[] bf;
22              public ByteReader()
23              {
24                  bf = new byte[1024 * 1024 * 100];//100MB
25              }
26
27              //todo: read data
28              public byte[] Read(){
```

System.OutOfMemoryException has been thrown

Out of memory

Show Details

Утечки памяти. Инструменты

- Главный инструмент - HeapShot

Утечки памяти. Инструменты

- Главный инструмент - HeapShot
  - Используется для отладки проблем с памятью

Утечки памяти. Инструменты

- Главный инструмент - HeapShot
  - Используется для отладки проблем с памятью
  - Основан на Mono Logs Profiler
    - --profile=log:heapshot

Утечки памяти. Инструменты

- Главный инструмент - HeapShot
  - Используется для отладки проблем с памятью
  - Основан на Mono Logs Profiler
    - --profile=log:heapshot
  - Можно сравнивать снимки памяти

Утечки памяти. Инструменты

- Запускаем HeapShot

```
iMac-Elizaveta:~ elizaveta$ mono --gc=sgen --profile=log:heapshot
~/Projects/MemoryLeakDotnext/MemoryLeakDotnext/bin/Debug/MemoryLeakDotnext.exe

iMac-Elizaveta:~ elizaveta$ mprof-report --alloc-sort=bytes output.mlpd
```

Утечки памяти. Инструменты

- Запускаем HeapShot

```
iMac-Elizaveta:~ elizaveta$ mono --gc=sgen --profile=log:heapshot
~/Projects/MemoryLeakDotnext/MemoryLeakDotnext/bin/Debug/MemoryLeakDotnext.exe

iMac-Elizaveta:~ elizaveta$ mprof-report --alloc-sort=bytes output.mlpd
```

Утечки памяти. Инструменты

- Получаем результат

```
Mono log profiler data
Profiler version: 1.1
Data version: 13
Arguments: log:heapshot
Architecture: x86-64
Operating system: osx
Mean timer overhead: 16 nanoseconds
Program startup: Thu Sep 7 14:38:39 2017
Program ID: 6768
Server listening on: 53856
```

Утечки памяти. Инструменты

- Нас интересует HeapShot Summary

Heap shot 0 at 0.436 secs: size: 16618832, object count: 381465, class count: 50, roots: 0

Bytes	Count	Average	Class name
8186288	95276	85	System.Byte[]
4572672	190528	24	System.Int32
3810640	95266	40	System.Collections.Generic.List<System.Object

Утечки памяти. Инструменты

- Нас интересует HeapShot Summary

Heap shot 1 at 0.907 secs: size: 32075648, object count: 760412, class count: 50, roots: 0

Bytes	Count	Average	Class name
15306160	190009	80	System.Byte[] (bytes: +7119872, count: +94733)
9120096	380004	24	System.Int32 (bytes: +4547424, count: +189476)
7600160	190004	40	System.Collections.Generic.List<System.Object> (bytes: +3789520, count: +94738)

Утечки памяти. Инструменты

- Нас интересует trace-report

```
iMac-Elizaveta:~ elizaveta$ mprof-report --traces --maxframes=2 output.mlpd

Bytes       Count  Average Class name
15306160 190009 80          System.Byte[] (bytes: +7119872, count: +94733)

12 root references (0 pinning)
190004 references from: System.Collections.Generic.List<System.Object>
```

Утечки памяти. Инструменты

- **Что у нас есть?**
  - Mac OS и leaks

https://youtu.be/vPMu1v0iXag



Утечки памяти. Выводы

- Анализируем утечки с помощью HeapShot
- Анализ утечек памяти занимает значительное время
- Под Mac OS можно анализировать утечки памяти с помощью Leaks





Как отлаживать ASP.NET?

Как отлаживать ASP.NET?

https://youtu.be/WDN0O4DCc5s







WinDBG & SOS

против

GDB/LLDB & mono

Проведём аналогию с WinDBG

Описание	WinDBG & SOS	GDB & mono	LLDB & mono
Посмотреть потоки	**!threads** **!threadpool**	**info threads**	**thread list**
Работа со стеком	!clrstack !dumpstack	mono_stack mono_backtrace	monobt
Посмотреть переменные	!clrstack -a	call mono_print_debug_var call mono_object_describe call mono_class_describe_statics ……	
Информация об объекте	!do (!dumpobject)	Получаем адрес объекта Обращаемся по указателю к Vtable ((MonoObject *)0x123456)->VTable Работаем с объектом VTable	

Проведём аналогию с WinDBG

Описание	WinDBG & SOS	GDB & mono	LLDB & mono
Посмотреть потоки	!threads !threadpool	info threads	thread list
Работа со стеком	**!clrstack** **!dumpstack**	**mono_stack** **mono_backtrace**	**monobt**
Посмотреть переменные	!clrstack -a	call mono_print_debug_var call mono_object_describe call mono_class_describe_statics ……	
Информация об объекте	!do (!dumpobject)	Получаем адрес объекта Обращаемся по указателю к Vtable ((MonoObject *)0x123456)->VTable Работаем с объектом VTable	

Проведём аналогию с WinDBG

Описание	WinDBG & SOS	GDB & mono	LLDB & mono
Посмотреть потоки	!threads !threadpool	info threads	thread list
Работа со стеком	!clrstack !dumpstack	mono_stack mono_backtrace	monobt
Посмотреть переменные	**!clrstack -a**	**call mono_print_debug_var** **call mono_object_describe** **call mono_class_describe_statics** **.....**	
Информация об объекте	!do (!dumpobject)	Получаем адрес объекта Обращаемся по указателю к Vtable ((MonoObject *)0x123456)->VTable Работаем с объектом VTable	

Проведём аналогию с WinDBG

Описание	WinDBG & SOS	GDB & mono	LLDB & mono
Посмотреть потоки	!threads !threadpool	info threads	thread list
Работа со стеком	!clrstack !dumpstack	mono_stack mono_backtrace	monobt
Посмотреть переменные	!clrstack -a	call mono_print_debug_var call mono_object_describe call mono_class_describe_statics ……	
Информация об объекте	**!do (!dumpobject)**	**Получаем адрес объекта** **Обращаемся по указателю к Vtable** **((MonoObject *)0x123456)->VTable** **Работаем с объектом VTable**	

Проведём аналогию с WinDBG

Описание	WinDBG & SOS	GDB & mono	LLDB & mono
Информация о куче	**!eeheap**	**mono_gc_max_generation, mono_gc_get_generation (MonoObject *object), mono_gc_get_heap_size, mono_gc_get_used_size, mono_gc_walk_heap, ……**	
Точки останова	!bmpd name.dll Namespace.Class.Method	b ‘Namespace.Class:Method’ --break ‘Namespace.Class:Method’	
Блокировки	!syncblock !analyze -hang -v -f	call mono_locks_dump call mono_object_describe call mono_class_describe_statics ……	

*Расширение для VTables:
https://monoruntime.wordpress.com/2009/04/22/interface-method-dispatch-im-table-and-thunks/

Проведём аналогию с WinDBG

Описание	WinDBG & SOS	GDB & mono	LLDB & mono
Информация о куче	!eeheap	mono_gc_max_generation, mono_gc_get_generation (MonoObject *object), mono_gc_get_heap_size, mono_gc_get_used_size, mono_gc_walk_heap, ……	
Точки останова	**!bmpd name.dll Namespace.Class.Method**	**b 'Namespace.Class:Method' --break 'Namespace.Class:Method'**	
Блокировки	!syncblock !analyze -hang -v -f	call mono_locks_dump call mono_object_describe call mono_class_describe_statics ……	

*Расширение для VTables:
https://monoruntime.wordpress.com/2009/04/22/interface-method-dispatch-im-table-and-thunks/

Проведём аналогию с WinDBG

Описание	WinDBG & SOS	GDB & mono	LLDB & mono
Информация о куче	!eeheap	mono_gc_max_generation, mono_gc_get_generation (MonoObject *object), mono_gc_get_heap_size, mono_gc_get_used_size, mono_gc_walk_heap, ……	
Точки останова	!bmpd name.dll Namespace.Class.Method	b ‘Namespace.Class:Method’ --break ‘Namespace.Class:Method’	
Блокировки	**!syncblock** **!analyze -hang -v -f**	**call mono_locks_dump** **call mono_object_describe** **call mono_class_describe_statics** ……	

*Расширение для VTables:
https://monoruntime.wordpress.com/2009/04/22/interface-method-dispatch-im-table-and-thunks/





Сравним функциональность отладчиков

Сравним функциональность отладчиков

Возможности	SDB	GDB	LLDB
Присоединение по PID	-	+	+
Удалённая отладка	+	+	+
Отладка managed	+	С ограничениями	С ограничениями
Unsafe, P/Invoke	-	+	+

Сравним функциональность отладчиков

Возможности	SDB	GDB	LLDB
Присоединение по PID	-	+	+
Удалённая отладка	**+**	**+**	**+**
Отладка managed	+	С ограничениями	С ограничениями
Unsafe, P/Invoke	-	+	+

Сравним функциональность отладчиков

Возможности	SDB	GDB	LLDB
Присоединение по PID	-	+	+
Удалённая отладка	+	+	+
Отладка managed	+	**С ограничениями**	**С ограничениями**
Unsafe, P/Invoke	-	+	+

Сравним функциональность отладчиков

Возможности	SDB	GDB	LLDB
Присоединение по PID	-	+	+
Удалённая отладка	+	+	+
Отладка managed	+	С ограничениями	С ограничениями
Unsafe, P/Invoke	**-**	**+**	**+**

Сравним функциональность отладчиков

Возможности	SDB	GDB	LLDB
Точки останова, трассировочные точки	+	+	**С ограничениями**
Дизассемблер	-	+	+
Backtrace	Только managed	+	+
Расширения	[Command] в .cs	Python	Python

Сравним функциональность отладчиков

Возможности	SDB	GDB	LLDB
Точки останова, трассировочные точки	+	+	С ограничениями
Дизассемблер	**-**	**+**	**+**
Backtrace	Только managed	+	+
Расширения	[Command] в .cs	Python	Python

Сравним функциональность отладчиков

Возможности	SDB	GDB	LLDB
Точки останова, трассировочные точки	+	+	С ограничениями
Дизассемблер	-	+	+
Backtrace	**Только managed**	**+**	**+**
Расширения	[Command] в .cs	Python	Python

Сравним функциональность отладчиков

Возможности	SDB	GDB	LLDB
Точки останова, трассировочные точки	+	+	С ограничениями
Дизассемблер	-	+	+
Backtrace	Только managed	+	+
Расширения	**[Command] в .cs**	**Python**	**Python**





Подведём итоги

Подведём итоги

- Приложения на mono отлаживать можно и нужно!

Подведём итоги

- Приложения на mono отлаживать можно и нужно!
  - SDB и IDE спасут Вас от 70% проблем

Подведём итоги

- Приложения на mono отлаживать можно и нужно!
  - SDB и IDE спасут Вас от 70% проблем
  - GDB/LLDB покроют ещё 20% проблем с unsafe, дедлоками и дампами
    - mono embedding - без неё не обойдется GDB/LLDB

Подведём итоги

- Приложения на mono отлаживать можно и нужно!
  - SDB и IDE спасут Вас от 70% проблем
  - GDB/LLDB покроют ещё 20% проблем с unsafe, дедлоками и дампами
    - mono embedding - без неё не обойдется GDB/LLDB
  - Valgrind/CLang Sanitizers/HeapShot покроют оставшиеся 10% проблем

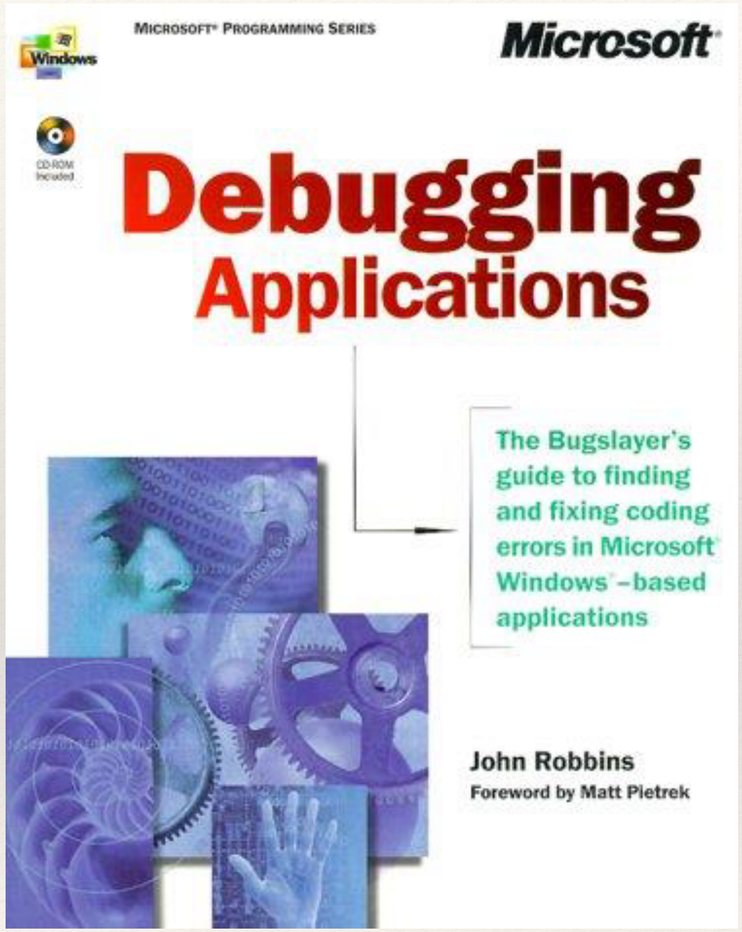


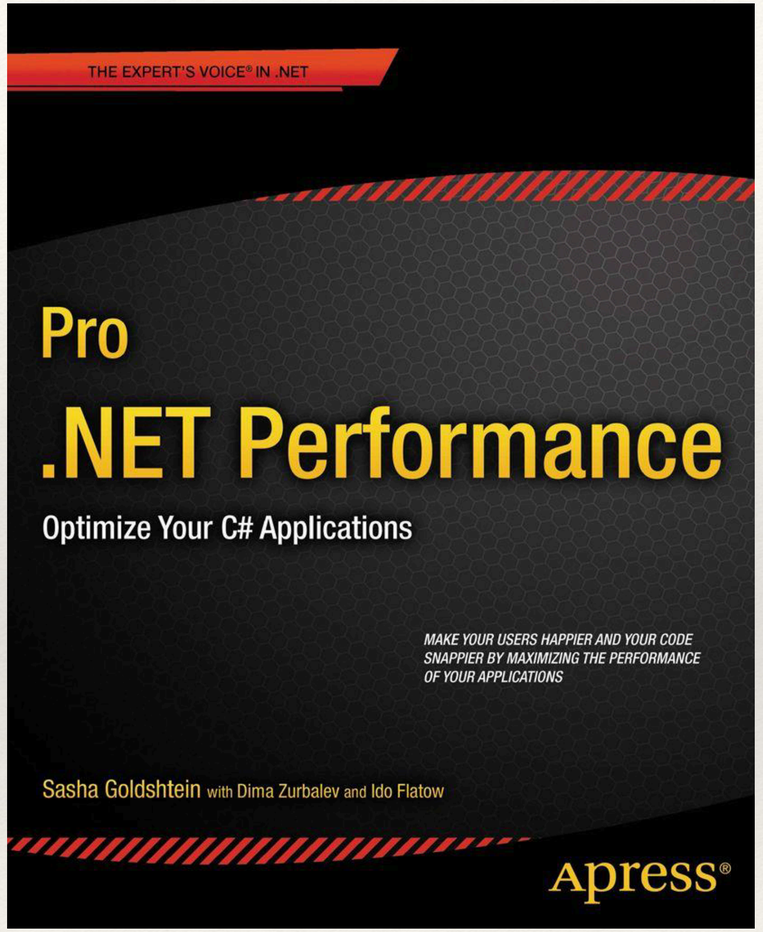


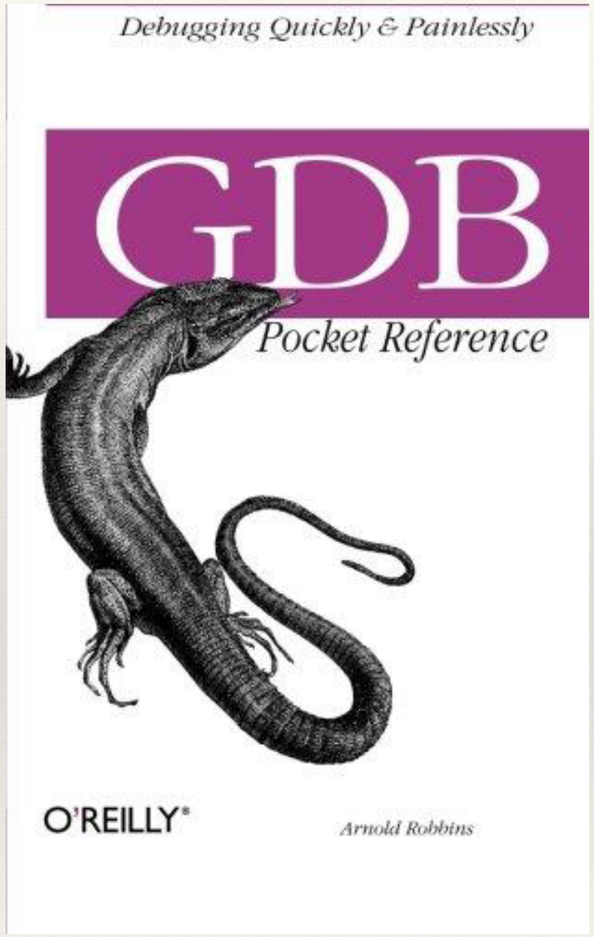
Литература

Литература

http://www.mono-project.com/docs/













Ваши вопросы?

Елизавета Голенок

golenok-ea5@narod.ru

@marmothetka





А как Вы отлаживаете свои приложения под MONO?

Ссылки на изображения

- https://www.infoniac.ru/news/Udivitel-nye-chlenistonogie-domashnie-pitomcy.html
- https://pxhere.com/ru/photo/1116276
- https://eladnava.com/debug-remote-mono-apps-via-xamarin-studio/
- http://eax.me/windbg/
- https://habrahabr.ru/post/327128/
- http://www.mono-project.com/docs/advanced/runtime/docs/soft-debugger/