Rynsztunek programisty C

Czym wyciągnąć kulę po postrzeleniu się w stopę

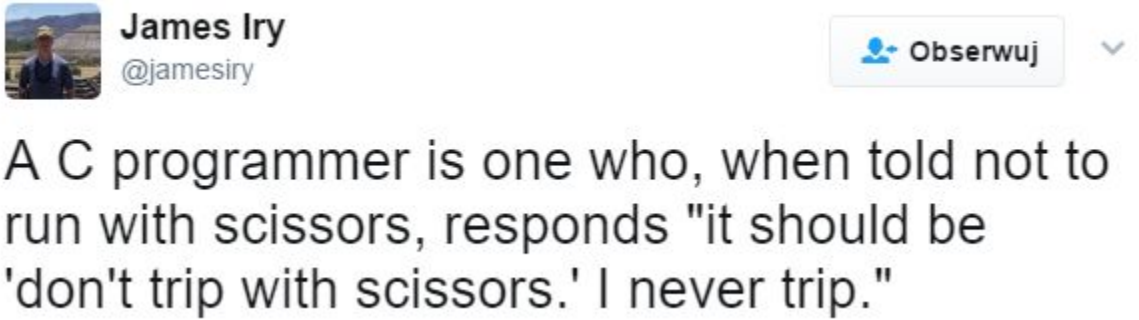
Mateusz Kwiatkowski

Marcin Radomski

Prezentacja: https://goo.gl/IkHIaM

Trochę historii C

- “Przenośny asembler”
  - Duże możliwości
  - Sporo okazji do popełnienia błędu
- Skutecznie u(nie)mila nam życie od 1972
  - sporo czasu na stworzenie narzędzi
  - ...ale czy się udało?



Bezpieczne programowanie w C

~~Bezpieczne~~ programowanie w C

- Nie da się uniknąć błędów
  - ...ale mamy narzędzia do ich wykrywania
    - I nawet czasem działają!
- Różne klasy problemów - różne narzędzia
  - Błędy w logice programu
  - Wycieki zasobów
  - Wielowątkowość
  - Integracja z systemem operacyjnym

Ostrzeżenia kompilatora

$ gcc -o main main.c

$ gcc -Wall -Wextra -o main main.c

C jest trudne

- Preprocesor!
- Kompilacja większych projektów
  - make
  - autoconf/automake
  - CMake
  - ...MSBuild?! Xcodebuild?!
- Zarządzanie zależnościami
  - Java → Maven, (Gradle, SBT, …)
  - Python → pip
  - Ruby → gem
  - C → ???

C jest trudne: preprocesor

```
AVS_LIST_FOREACH_PTR(entry_ptr, &anjay->dm.modules) {
    if ((*entry_ptr)->def == module) {
        return entry_ptr;
    }
}
```

C jest trudne: preprocesor

```
AVS_LIST(anjay_dm_installed_module_t) *
_anjay_dm_module_find_ptr(anjay_t *anjay, const anjay_dm_module_t *module) {
    if (!anjay) {
        return NULL;
    }
    AVS_LIST(anjay_dm_installed_module_t) *entry_ptr;
    AVS_LIST_FOREACH_PTR(entry_ptr, &anjay->dm.modules) {
        if ((*entry_ptr)->def == module) {
            return entry_ptr;
        }
    }
    return NULL;
}
```

C jest trudne: preprocesor

```c
#include <avsystem/commons/list.h>

AVS_LIST(anjay_dm_installed_module_t) *
_anjay_dm_module_find_ptr(anjay_t *anjay, const anjay_dm_module_t *module) {
    if (!anjay) {
        return NULL;
    }
    AVS_LIST(anjay_dm_installed_module_t) *entry_ptr;
    AVS_LIST_FOREACH_PTR(entry_ptr, &anjay->dm.modules) {
        if ((*entry_ptr)->def == module) {
            return entry_ptr;
        }
    }
    return NULL;
}
```

C jest trudne: preprocesor

```
/* avsystem/commons/list.h */
#define AVS_LIST(element_type) element_type*

#define AVS_TYPEOF_PTR(symbol) __typeof__(symbol)

#define AVS_APPLY_OFFSET(type, struct_ptr, offset) \
        ((type *) (((char *) (intptr_t) (struct_ptr)) + (offset)))

#define AVS_LIST_SPACE_FOR_NEXT__ \
offsetof(struct avs_list_space_for_next_helper_struct__, value)

#define AVS_LIST_NEXT(element) \
(*AVS_APPLY_OFFSET(AVS_TYPEOF_PTR(element), element, \
        -AVS_LIST_SPACE_FOR_NEXT__))

#define AVS_LIST_NEXT_PTR(element_ptr) \
((AVS_TYPEOF_PTR(*(element_ptr)) *) &AVS_LIST_NEXT(*(element_ptr)))

#define AVS_LIST_FOREACH_PTR(element_ptr, list_ptr) \
for ((element_ptr) = (list_ptr); \
     *(element_ptr); \
     (element_ptr) = AVS_LIST_NEXT_PTR(element_ptr))
```

C jest trudne: preprocesor

```
anjay_dm_installed_module_t **
_anjay_dm_module_find_ptr(anjay_t *anjay, const anjay_dm_module_t *module) {
    if (!anjay) {
        return NULL;
    }
    anjay_dm_installed_module_t **entry_ptr;
    for (entry_ptr = &anjay->dm.modules;
            *entry_ptr;
            entry_ptr = (__typeof__(*entry_ptr) *)
                &*(__typeof__(*entry_ptr) *) (
                    ((char *) (intptr_t) *entry_ptr)
                    + (-offsetof(struct avs_list_space_for_next_helper_struct__,
                                 value)))) {
        if ((*entry_ptr)->def == module) {
            return entry_ptr;
        }
    }
    return NULL;
}
```

C jest trudne: preprocesor

```
#define AVS_TYPEOF_PTR(symbol) __typeof__(symbol)
```

C jest trudne: preprocesor

```
#if !defined(AVS_CONFIG_TYPEOF) && !defined(AVS_CONFIG_NO_TYPEOF) \
        && !defined(__cplusplus) && __GNUC__
#define AVS_CONFIG_TYPEOF __typeof__
#endif

#ifdef AVS_CONFIG_TYPEOF
#define AVS_TYPEOF_PTR(symbol) AVS_CONFIG_TYPEOF(symbol)
#elif defined(__cplusplus) && \
        (__cplusplus >= 201103L || defined(__GXX_EXPERIMENTAL_CXX0X__))
#include <type_traits>
#define AVS_TYPEOF_PTR(symbol) std::decay<decltype((symbol))>::type
#else
#define AVS_TYPEOF_PTR(symbol) void *
#endif
```

C jest trudne

- Prawidłowe parsowanie kodu wymaga dostępu do **CAŁEGO** projektu wraz z zależnościami!!!
  - Poznanie zależności wymaga parsowania opisu projektu
    - wiele standardów: make, autoconf/automake, CMake, MSBuild, Xcodebuild, …
  - `#define`'y wbudowane w kompilator (`__GNUC__` itp.)
- Konsekwencje dla wielu narzędzi:
  - IDE
  - Cross-referencje
  - Generowanie dokumentacji
  - Autoformatery
- Często stosowane przybliżenia parsujące pojedyncze pliki
  - Z różnym skutkiem...

W czym pisać?

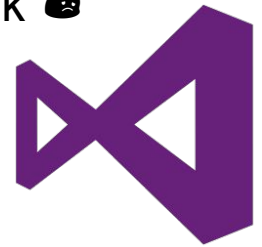
- Dwie szkoły
  - IDE
  - „Proste” edytory na sterydach

IDE

- Komercyjne
  - Microsoft Visual Studio
  - Apple Xcode
  - JetBrains CLion
- Wolne
  - Eclipse CDT
  - NetBeans C/C++ plugin
  - Code::Blocks
  - wiele innych
    - Qt Creator, KDevelop, GNOME Builder, CodeLite, ...
- Wszystkie mają swoje problemy :(

Komercyjne IDE?

- Microsoft Visual Studio
  - Ceny zaczynają się od $499 (ok. 2000 zł) per użytkownik
  - Bezpłatna wersja Community
    - Dla indywidualnych developerów i małych zespołów
  - Pewien standard dla programowania pod Windows
    - Często spotykane w firmach celujących w ten rynek
  - Microsoft zaniedbał „czyste C”
    - W natywnym kompilatorze wciąż brak wsparcia dla C99!
- Apple Xcode
  - Darmowe – ale trzeba mieć Maka
- Problemy z obsługą nienatywnych projektów
  - CMake potrafi generować pliki projektów MSBuild/Xcodebuild



Komercyjne IDE?

- CLion
  - Ceny od €89 (ok. 380 zł) rocznie per użytkownik, różne zniżki
  - Bezpłatne dla edukacji i istniejących, niesponsorowanych projektów open source
  - Napisane w Javie – Windows, Linux, Mac
  - Współpracuje tylko z CMake ☹
    - Ale za to bardzo dobrze! ☺
  - Prawdopodobnie najlepiej na rynku działające podpowiadanie składni, refactoring, ogarnianie kruczków języka (preprocesor, template'y C++)

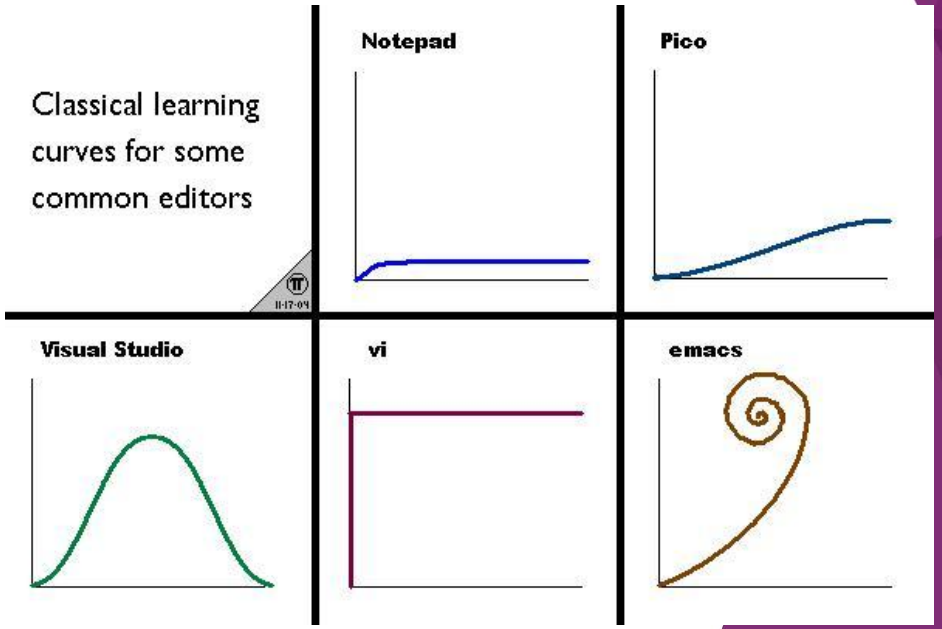
Wolne IDE

- NetBeans
  - Napisane w Javie – Windows, Linux, Mac
  - Dostosowuje się do dowolnego systemu budowania
    - Parsowanie projektu oparte na logu z kompilacji – efekty bywają różne
  - Radzi sobie z większością kruczków języka
    - Często kapituluje przy `auto` z C++11/14
  - Powolne, trochę irytujących bugów
  - Mimo wszystko – prawdopodobnie najlepiej działające darmowe IDE do C/C++
- Eclipse + CDT
  - w miarę działa, ale trochę wyszło z mody
- Code::Blocks, …
  - Z doświadczenia – jeszcze więcej problemów z parsowaniem złożonych projektów

„Proste” edytory

- Vim
- Emacs

- Wymagają czasu
  - http://www.vimgolf.com/
- Dostępne praktycznie wszędzie
- Działają po SSH
- Pluginy na każdą okazję
  - *Prawie* jak IDE
  - Linux jest Twoim IDE



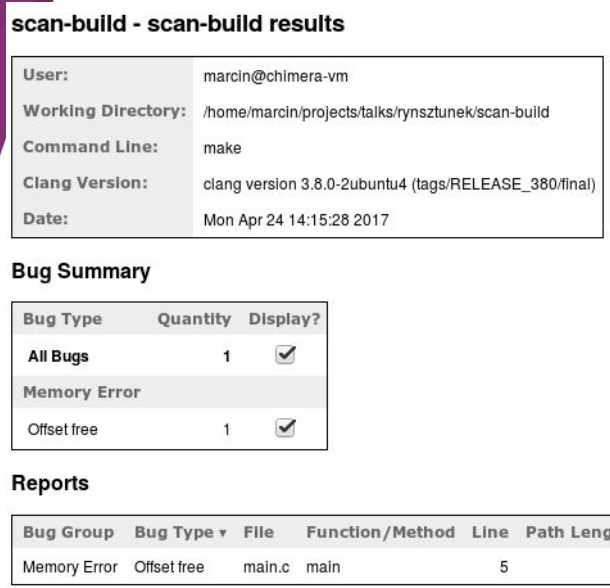
Bonus: Vim

- https://github.com/dextero/dotfiles
- Undo branches
  - Kompletna historia zmian w pliku w formie drzewa
- Persistent undo
  - w niektórych IDE znane jako “local history”
- Przydatne pluginy:
  - YouCompleteMe - podpowiadanie składni dla różnych jezyków
    - C/C++/C#/Python/Go/Rust i jeszcze parę
  - vim-tmux-navigator - integracja z tmuxem
  - syntastic - zaznacza ostrzeżenia/błędy kompilatora w edytorze
  - Gundo/Mundo - undo branches w bardziej przyjaznej formie

Statyczna analiza kodu

- scan-build (clang)
  - `scan-build make`
  - `cmake -DCMAKE_C_COMPILER=/usr/lib/llvm-3.8/libexec/ccc-analyzer .;`
    `scan-build make`
- Coverity
  - Darmowy dla projektów open-source
  - Integracja z Githubem

scan-build



scan-build - scan-build results

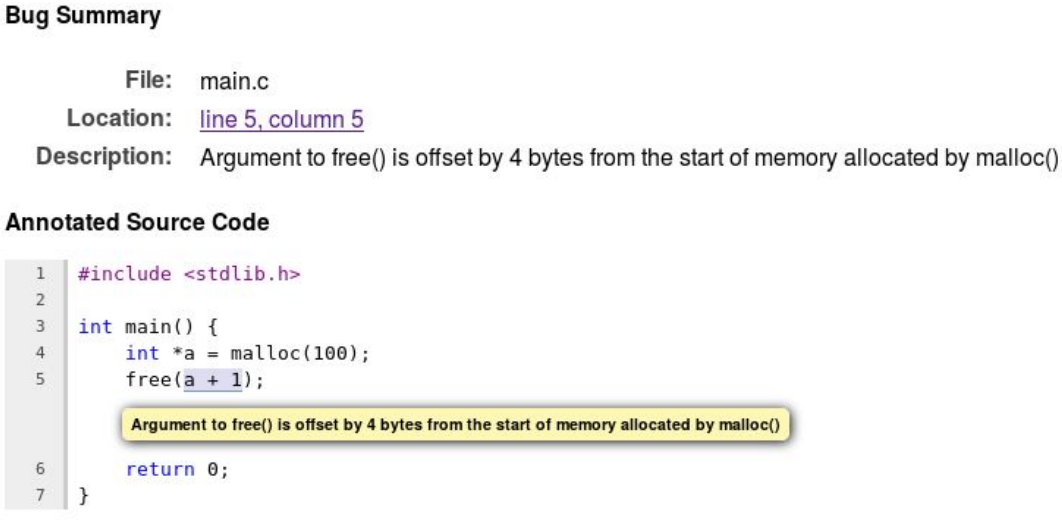
User:	marcin@chimera-vm
Working Directory:	/home/marcin/projects/talks/rynsztunek/scan-build
Command Line:	make
Clang Version:	clang version 3.8.0-2ubuntu4 (tags/RELEASE_380/final)
Date:	Mon Apr 24 14:15:28 2017

Bug Summary

Bug Type	Quantity	Display?
All Bugs	1	☑
Memory Error		
Offset free	1	☑

Reports

Bug Group	Bug Type ▾	File	Function/Method	Line	Path Length	
Memory Error	Offset free	main.c	main	5	1	View Report



Bug Summary

File: main.c
Location: line 5, column 5
Description: Argument to free() is offset by 4 bytes from the start of memory allocated by malloc()

Annotated Source Code

```
1  #include <stdlib.h>
2
3  int main() {
4      int *a = malloc(100);
5      free(a + 1);
       Argument to free() is offset by 4 bytes from the start of memory allocated by malloc()
6      return 0;
7  }
```

Debuggery

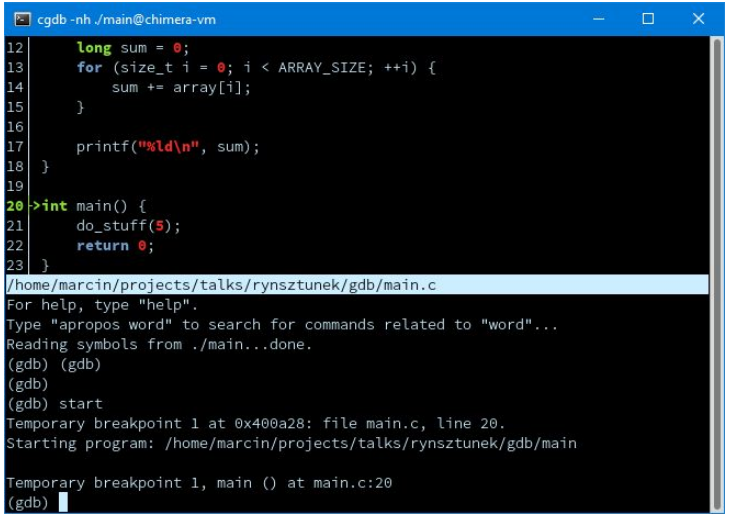
- Uruchamianie programu linia po linii
- Breakpointy
- Watchpointy
- Call stack
- Podglądanie wartości zmiennych i rejestrów procesora
- Funkcje w kodzie asemblera

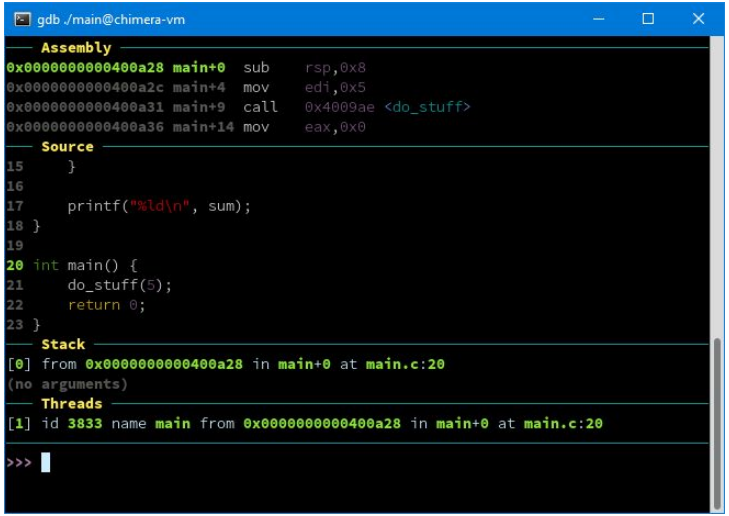
Debuggery

- Zdalne debugowanie (gdbserver)
  - gdbserver :1111 ./program
  - gdb ./program -ex “target remote $IP:1111”
- (GDB 7+/rr) Cofanie się w czasie
  - record
  - reverse-*
  - http://jayconrod.com/posts/28/tutorial-reverse-debugging-with-gdb-7
- CppCon 2016: "GDB - A Lot More Than You Knew"

Nakładki na GDB

- IDE
- Terminal:
  - gdb TUI (wbudowane w GDB:Ctrl-X, A włącza/wyłącza)
  - cgdb
  - gdb-dashboard (skrypt do GDB)





Demo?

(gdbserver/reverse-step)

Valgrind

- Uruchamia gotowe programy w kontrolowanym środowisku
  - Wewnętrznie: maszyna wirtualna, rekompilacja JIT
- Śledzi różne rodzaje wywołań
- Modułowa architektura
  - Memcheck – debugowanie odwołań do pamięci
  - Helgrind, DRD – debugowanie programów wielowątkowych
  - Cachegrind – profilowanie wydajności użycia cache procesora
  - Callgrind – śledzenie wywołań funkcji
  - Massif – profilowanie zużycia pamięci
- Wspierane platformy
  - Linux (x86, ARM, PPC, MIPS, …), Android (x86, ARM, MIPS)
  - Solaris (x86)
  - starsze wersje Mac OS X (10.10, częściowo 10.11; x86)

Valgrind: Memcheck

- Wykrywa:
  - Odwołania do niezaalokowanej/zwolnionej pamięci
  - Wycieki pamięci
  - Użycie niezainicjalizowanych wartości
  - Nieprawidłowe i podwójne `free()`
- Używany przy rozwoju wielu znanych projektów
  - Firefox
  - Python
  - LibreOffice
  - libstdc++
  - ...

Valgrind: Memcheck

```c
#include <stdlib.h>

int main() {
    int *tab = (int *) malloc(25 * sizeof(int));
    if (!tab[0]) {
        tab[25] = 42;
    }
    return 0;
}
```

Valgrind: Memcheck

```
$ gcc -g test.c -o test
$ valgrind --leak-check=full ./test
==18277== [...]
==18277== Conditional jump or move depends on uninitialised value(s)
==18277== at 0x1086BE: main (test.c:5)
==18277==
==18277== Invalid write of size 4
==18277== at 0x1086C8: main (test.c:6)
==18277==  Address 0x52000a4 is 0 bytes after a block of size 100 alloc'd
==18277== at 0x4C2CB3F: malloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==18277== by 0x1086B1: main (test.c:4)
==18277==
==18277== HEAP SUMMARY:
==18277==     in use at exit: 100 bytes in 1 blocks
==18277==   total heap usage: 1 allocs, 0 frees, 100 bytes allocated
==18277==
==18277== 100 bytes in 1 blocks are definitely lost in loss record 1 of 1
==18277== at 0x4C2CB3F: malloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==18277== by 0x1086B1: main (test.c:4)
==18277== [...]
```

Valgrind: Helgrind, DRD

- Helgrind
  - **Potencjalne deadlocki** (nie muszą wystąpić żeby je znaleźć!)
  - Niesynchronizowany dostęp do danych
  - Nieprawidłowe użycie API wątków
- DRD
  - Zbyt długie blokowanie muteksów
  - Niesynchronizowany dostęp do danych
  - Nieprawidłowe użycie API wątków
- DRD jest szybszy, jednak wykrywa mniej typów błędów
- Zdarzają się fałszywe alarmy

Valgrind: Helgrind

```c
#include <pthread.h>
pthread_mutex_t mutex1 = PTHREAD_MUTEX_INITIALIZER;
pthread_mutex_t mutex2 = PTHREAD_MUTEX_INITIALIZER;

void *thread1(void *ignored) {
    pthread_mutex_lock(&mutex1);
    pthread_mutex_lock(&mutex2);
    pthread_mutex_unlock(&mutex2);
    pthread_mutex_unlock(&mutex1);
}
void *thread2(void *ignored) {
    pthread_mutex_lock(&mutex2);
    pthread_mutex_lock(&mutex1);
    pthread_mutex_unlock(&mutex1);
    pthread_mutex_unlock(&mutex2);
}
int main() {
    pthread_t tid1;
    pthread_t tid2;
    pthread_create(&tid1, NULL, thread1, NULL);
    pthread_create(&tid2, NULL, thread2, NULL);
    pthread_join(tid1, NULL);
    pthread_join(tid2, NULL);
}
```

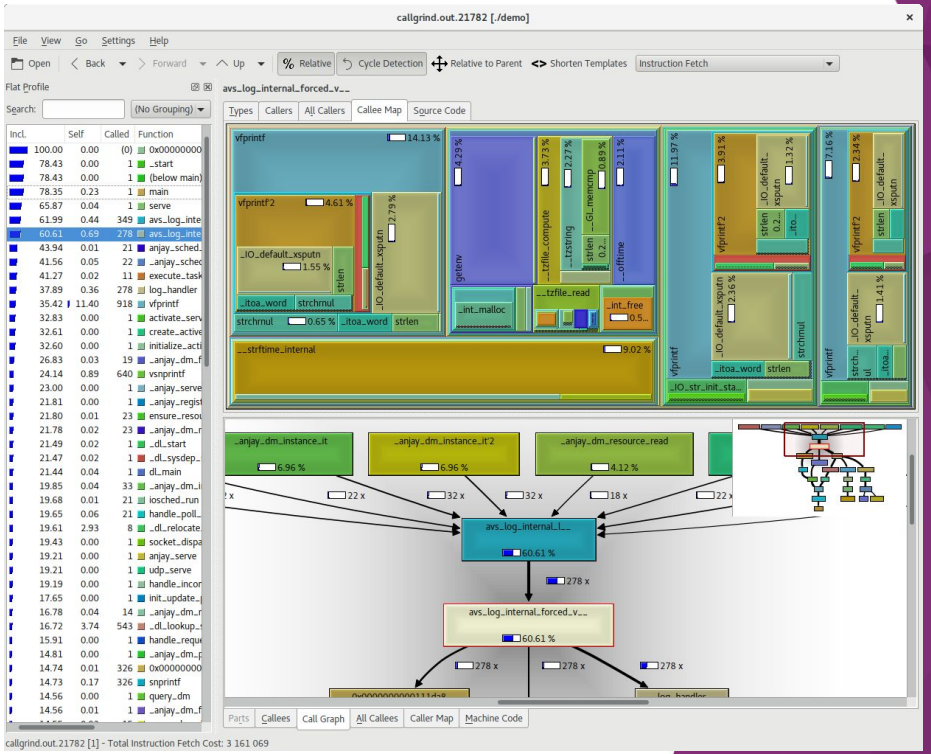
Valgrind: Helgrind

```
$ gcc -g test.c -o test -pthread
$ valgrind --tool=helgrind ./test
==21051== [...]
==21051== Thread #3: lock order "0x309040 before 0x309080" violated
==21051==
==21051== Observed (incorrect) order is: acquisition of lock at 0x309080
==21051== at 0x4C3109C: ??? (in /usr/lib/valgrind/vgpreload_helgrind-amd64-linux.so)
==21051== by 0x108876: thread2 (test.c:13)
==21051==
==21051==  followed by a later acquisition of lock at 0x309040
==21051== at 0x4C3109C: ??? (in /usr/lib/valgrind/vgpreload_helgrind-amd64-linux.so)
==21051== by 0x108882: thread2 (test.c:14)
==21051==
==21051== Required order was established by acquisition of lock at 0x309040
==21051== at 0x4C3109C: ??? (in /usr/lib/valgrind/vgpreload_helgrind-amd64-linux.so)
==21051== by 0x108837: thread1 (test.c:7)
==21051==
==21051==  followed by a later acquisition of lock at 0x309080
==21051== at 0x4C3109C: ??? (in /usr/lib/valgrind/vgpreload_helgrind-amd64-linux.so)
==21051== by 0x108843: thread1 (test.c:8)
==21051== [...]
```

Valgrind: Cachegrind, Callgrind

- Profiluje czas wykonywania programu z uwzględnieniem użycia cache CPU
- KCachegrind – przeglądanie wyników w graficznej formie
  - Wiele dostępnych analiz
- Niestety spowalnia działanie programu ok. 50-krotnie

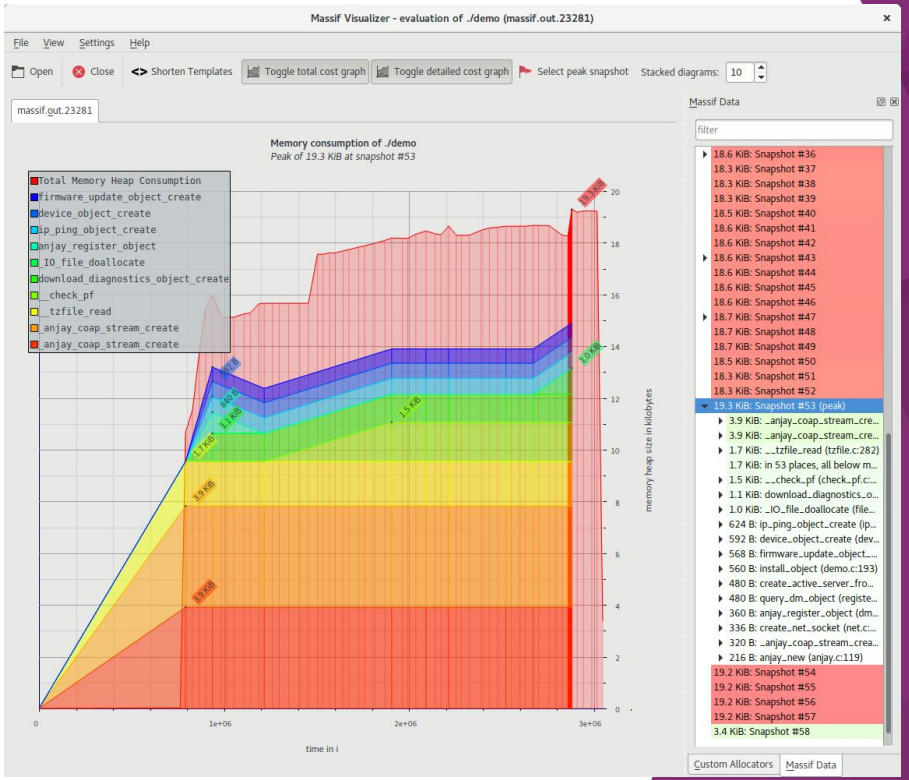
```
$ valgrind --tool=callgrind ./demo
$ kcachegrind callgrind.out.21782
```



Valgrind: Massif

- Analiza zużycia pamięci
  - Sterta
  - Stos (`--stack=yes`)
- Periodyczne zrzuty
  - Konfigurowalna częstotliwość
  - `--detailed-freq`
  - `--max-snapshots`
- Massif Visualizer

```
$ valgrind --tool=massif ./demo
$ massif-visualizer massif.out.23281
```



LLVM Sanitizers

- Stworzone w dużej mierze w Google'u
- Wbudowane w kompilator Clang
  - Dostępne na większość wspieranych przez niego platform
  - W teorii dostępne też w GCC, w praktyce nie zawsze to działa…
- W dużej mierze alternatywa dla Valgrinda…
  - Memcheck ≈
    - AddressSanitizer – dostęp do niezaalokowanej pamięci
    - MemorySanitizer – czytanie niezainicjalizowanych wartości
    - LeakSanitizer – wycieki pamięci
  - ThreadSanitizer ≈ Helgrind + DRD
- UndefinedBehaviorSanitizer
- DataFlowSanitizer – kontrakt dostępu do danych
  - Wymaga „adnotacji" (specjalne API) w kodzie
- SanitizerCoverage, SanitizerStats

LLVM Sanitizers

- Wymagają włączenia na etapie kompilacji
- Nie wszystkie można ze sobą łączyć
  - Konieczność rekompilacji projektu dla różnych testów…

```
$ clang -fsanitize=address test.c -o test
$ ./test
=================================================================
==24295==ERROR: LeakSanitizer: detected memory leaks
Direct leak of 100 byte(s) in 1 object(s) allocated from:
    #0 0x4b62d8  (./test+0x4b62d8)
    #1 0x4e726a  (./test+0x4e726a)
    #2 0x7f27720c83f0  (/lib/x86_64-linux-gnu/libc.so.6+0x203f0)
SUMMARY: AddressSanitizer: 100 byte(s) leaked in 1 allocation(s).

$ clang -fsanitize=address -fsanitize=memory test.c -o test
clang: error: invalid argument '-fsanitize=address' not allowed with
'-fsanitize=memory'
```

UndefinedBehaviorSanitizer

```
$ cat test.c
#include <limits.h>
int main() {
    int tmp = INT_MAX;
    ++tmp;
    return 0;
}

$ clang -fsanitize=undefined -g test.c -o test
$ ./test
test.c:4:5: runtime error: signed integer overflow:
2147483647 + 1 cannot be represented in type 'int'
```

Valgrind, Sanitizers

- Nie zawsze wykrywają wszystkie problemy
- Zdarzają się też fałszywe alarmy
  - Zwłaszcza przy używaniu „sprytnych” hacków
  - Valgrind instrumentuje cały kod, w tym zależności
    - Np. OpenSSL ma DUŻO „sprytnych” hacków
- Analizują tylko rzeczywiście wykonane fragmenty kodu
  - Testy, testy, testy!

Fuzzing

Fuzzing

- `./program </dev/urandom`
  - ...tylko może trochę sprytniej?
- Przykład: AFL fuzzer
  - Kompilacja: `gcc` → `afl-gcc`
  - Uruchamianie: `afl-fuzz -i input -o output -- ./program`
    - Wymaga przynajmniej jednego poprawnego wejścia w `input/`

Demo?
(afl-fuzz)

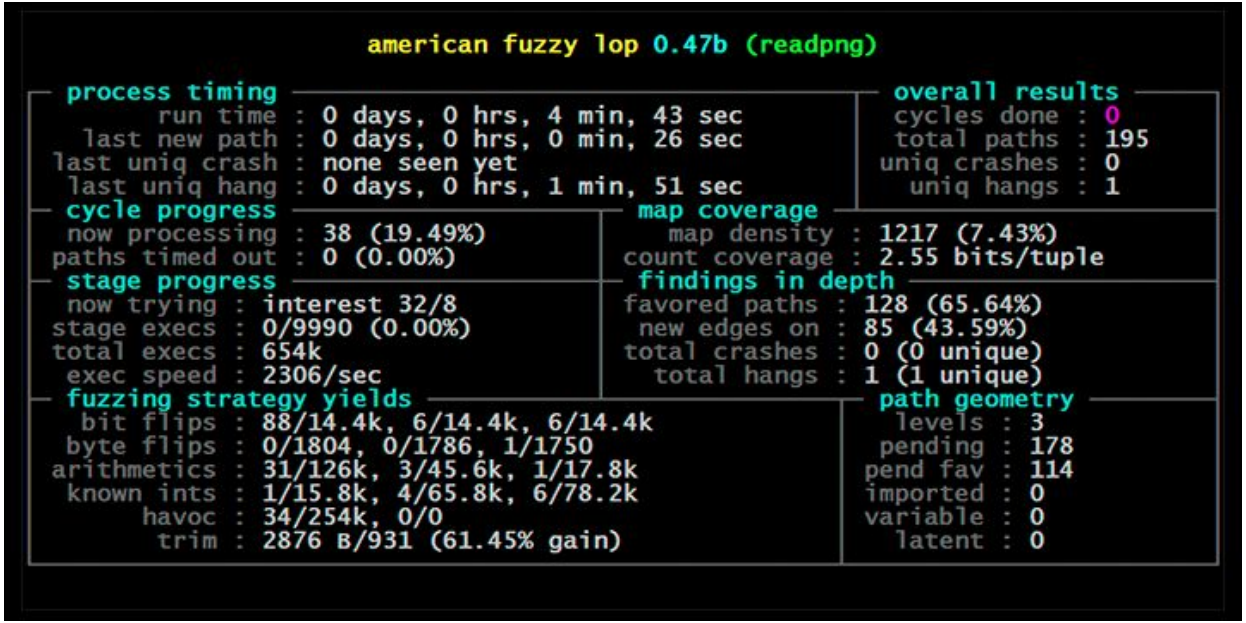
Fuzzing: AFL demo

```c
#include <stdio.h>

int main() {
    int a, b;
    scanf("%d %d", &a, &b);
    printf("%d\n", a / b);
    return 0;
}
```

- echo “10 2” > input/simple
- make CC=afl-gcc
- afl-fuzz -i input -o output -- ./main

Fuzzing: AFL

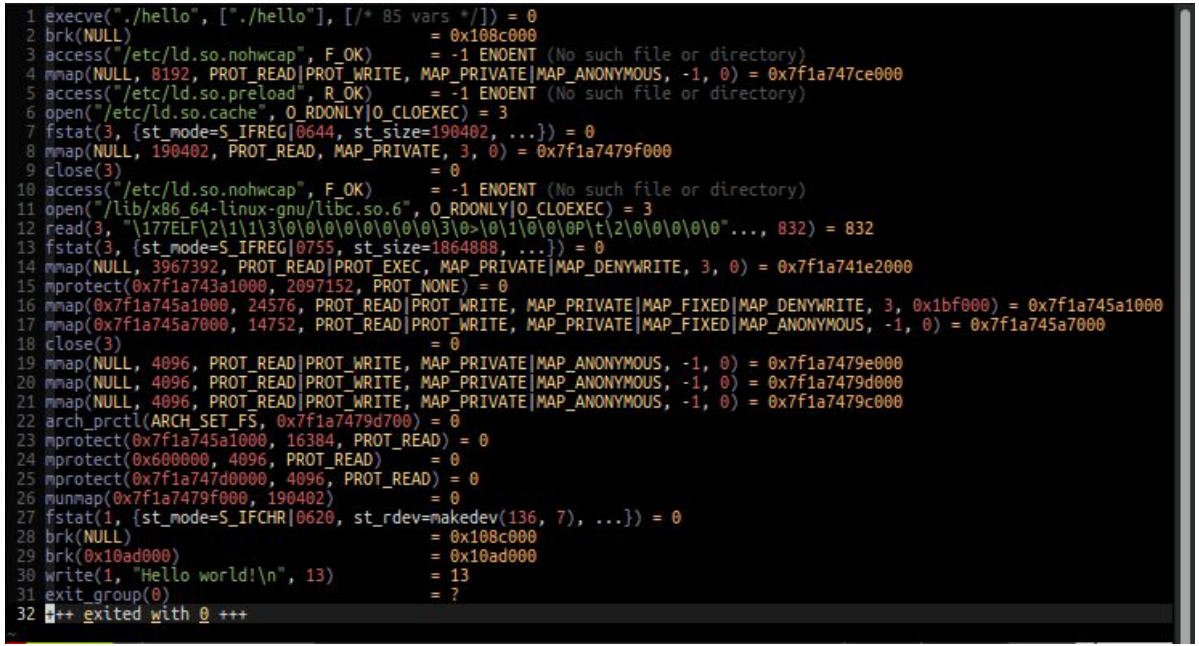

american fuzzy lop 0.47b (readpng)
process timing
run time : 0 days, 0 hrs, 4 min, 43 sec
last new path : 0 days, 0 hrs, 0 min, 26 sec
last uniq crash : none seen yet
last uniq hang : 0 days, 0 hrs, 1 min, 51 sec
overall results
cycles done : 0
total paths : 195
uniq crashes : 0
uniq hangs : 1
cycle progress
now processing : 38 (19.49%)
paths timed out : 0 (0.00%)
map coverage
map density : 1217 (7.43%)
count coverage : 2.55 bits/tuple
stage progress
now trying : interest 32/8
stage execs : 0/9990 (0.00%)
total execs : 654k
exec speed : 2306/sec
findings in depth
favored paths : 128 (65.64%)
new edges on : 85 (43.59%)
total crashes : 0 (0 unique)
total hangs : 1 (1 unique)
fuzzing strategy yields
bit flips : 88/14.4k, 6/14.4k, 6/14.4k
byte flips : 0/1804, 0/1786, 1/1750
arithmetics : 31/126k, 3/45.6k, 1/17.8k
known ints : 1/15.8k, 4/65.8k, 6/78.2k
havoc : 34/254k, 0/0
trim : 2876 B/931 (61.45% gain)
path geometry
levels : 3
pending : 178
pend fav : 114
imported : 0
variable : 0
latent : 0

Fuzzing

- `cat /dev/urandom | ./program`
  - ...tylko może trochę sprytniej?
- Przykład: AFL http://lcamtuf.coredump.cx/afl/
  - Kompilacja: `gcc` → `afl-gcc`
  - Uruchamianie: `afl-fuzz -i input -o output -- ./program`
    - Wymaga przynajmniej jednego poprawnego wejścia w `input/`

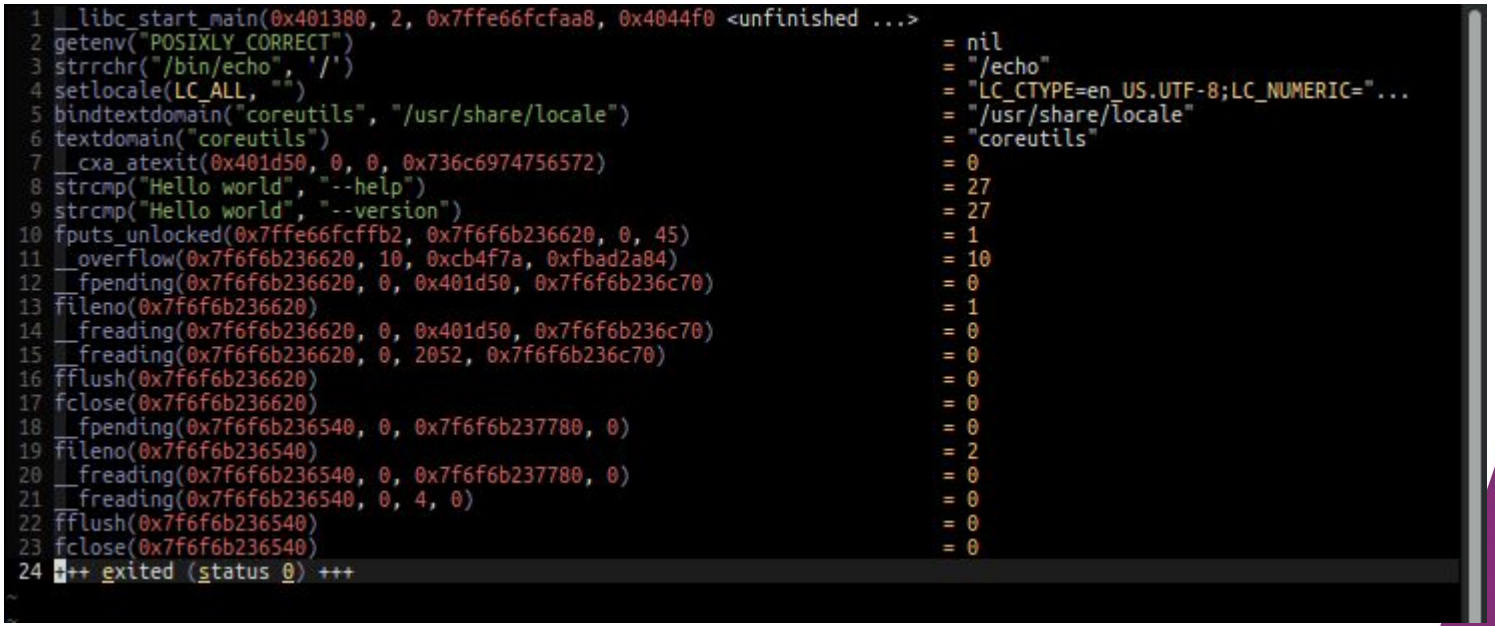
strace

- Podgląd syscalli wykonywanych przez proces: `strace ./hello`
  - Zrzucanie do pliku: `strace -o hello.strace -- ./hello`
  - Vim potrafi pokolorować `*.strace`



ltrace

- Podgląd wywołań funkcji z bibliotek .so: `ltrace ./hello`
  - Zrzucanie do pliku: `ltrace -o hello.ltrace -- /bin/echo “Hello world”`
  - Vim potrafi pokolorować `*.ltrace`



DTrace

- Dostępny na bardziej hipsterskich systemach
  - Solaris, macOS, *BSD
  - Eksperymentalne porty na Linuksa
- Skryptowalne śledzenie procesów!
- `dtruss` – dołączony skrypt podobny do strace/ltrace
- Więcej informacji: https://en.wikipedia.org/wiki/DTrace

DTrace

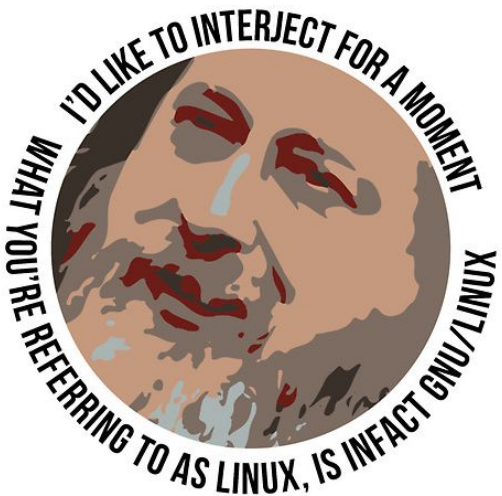
```
$ sudo dtrace -n 'syscall:::entry { @num[probefunc] = count(); }' -c './echo Hello, world'
dtrace: description 'syscall:::entry ' matched 522 probes
Hello, world
dtrace: pid 1394 has exited
  __disable_threadsignal                                          1
  __semwait_signal                                                1
  bsdthread_ctl                                                   1
  bsdthread_terminate                                             1
  exit                                                            1
  fstat64                                                         1
  getrlimit                                                       1
  psynch_cvsignal                                                 1
  psynch_mutexwait                                                1
  kevent_qos                                                      2
  psynch_cvwait                                                   2
  read                                                            2
  workq_kernreturn                                                2
  psynch_cvbroad                                                  3
  select                                                          3
  write_nocancel                                                  3
  madvise                                                         4
  sigaction                                                       4
  sysctl                                                          4
  getrusage                                                       6
  ioctl                                                          20
```

DTrace

```
$ sudo dtruss ./echo 'Hello, world'
SYSCALL(args)    = return
Hello, world
open("/dev/dtracehelper\0", 0x2, 0x7FFF5D6D8980)    = 3 0
ioctl(0x3, 0x80086804, 0x7FFF5D6D8908)    = 0 0
close(0x3)    = 0 0
thread_selfid(0x3, 0x80086804, 0x7FFF5D6D8908)    = 12794 0
bsdthread_register(0x7FFFC765B1F0, 0x7FFFC765B1E0, 0x2000)    = 1073741919 0
ulock_wake(0x1, 0x7FFF5D6D802C, 0x0)    = -1 Err#2
issetugid(0x1, 0x7FFF5D6D802C, 0x0)    = 0 0
mprotect(0x102533000, 0x88, 0x1)    = 0 0
mprotect(0x102535000, 0x1000, 0x0)    = 0 0
mprotect(0x10254B000, 0x1000, 0x0)    = 0 0
mprotect(0x10254C000, 0x1000, 0x0)    = 0 0
mprotect(0x102562000, 0x1000, 0x0)    = 0 0
mprotect(0x10252A000, 0x1000, 0x1)    = 0 0
mprotect(0x102533000, 0x88, 0x3)    = 0 0
mprotect(0x102533000, 0x88, 0x1)    = 0 0
getpid(0x102533000, 0x88, 0x1)    = 1465 0
stat64("/AppleInternal/XBS/.isChrooted\0", 0x7FFF5D6D7EE8, 0x1)    = -1 Err#2
stat64("/AppleInternal\0", 0x7FFF5D6D7F80, 0x1)    = -1 Err#2
csops(0x5B9, 0x7, 0x7FFF5D6D7A10)    = -1 Err#22
csops(0x5B9, 0x7, 0x7FFF5D6D72F0)    = -1 Err#22
getrlimit(0x1008, 0x7FFF5D6D9878, 0x7FFF5D6D72F0)    = 0 0
fstat64(0x1, 0x7FFF5D6D9898, 0x7FFF5D6D72F0)    = 0 0
ioctl(0x1, 0x4004667A, 0x7FFF5D6D98DC)    = 0 0
```

GNU binutils

- https://www.gnu.org/software/binutils/
- `ldd` - wyświetla używane biblioteki .so
- `strip` - wycina symbole ze skompilowanych plików
  - Mniejszy rozmiar
  - Trudniejsze debugowanie
- `objdump` - wyświetla zawartość plików .a/.o/.so/ELF
  - `objdump -T libfoo.so` - lista symboli
- `strings` - fragmenty tekstu w dowolnym binarnym pliku
  - ostrożnie z niezaufanymi plikami!
- `readelf` - podgląd nagłówków plików ELF



GNU ld.so: triki

- `ld.so` - dynamiczny linker
  - Niezbędny żeby biblioteki .so zadziałały
- Przydatne zmienne środowiskowe
  - LD_LIBRARY_PATH - dodatkowe ścieżki do plików .so
    - `LD_LIBRARY_PATH=. ./program`
  - LD_PRELOAD - wymuszanie załadowania .so
    - Nadpisywanie funkcji bibliotecznych!
    - `LD_PRELOAD=libmymalloc.so ./program`
  - Więcej info: `man ld.so`

bloaty

- https://github.com/google/bloaty
- Profilowanie pod względem rozmiaru pliku wykonywalnego
  - Obsługuje pliki ELF (Linux, Solaris, *BSD) i Mach-O (macOS)
- Profilowanie po:
  - Sekcji pliku wykonywalnego (kod vs. dane vs. …)
  - Jednostkach translacji
  - Symbolach

bloaty

```
$ bloaty demo
      VM SIZE                    FILE SIZE
 --------------                --------------
   0.0%      0 .debug_info       484Ki  34.0%
  60.9%   338Ki .text            338Ki  23.7%
   0.0%      0 .debug_line       115Ki   8.1%
   0.0%      0 .debug_str       86.9Ki   6.1%
   0.0%      0 .symtab          63.4Ki   4.5%
  11.4%  63.4Ki .rodata          63.4Ki   4.5%
  11.2%  62.4Ki .eh_frame        62.4Ki   4.4%
   0.0%      0 .debug_abbrev    57.7Ki   4.1%
   0.0%      0 .strtab          51.5Ki   3.6%
   5.6%  31.0Ki .data.rel.ro     31.0Ki   2.2%
   3.0%  16.6Ki .rela.dyn        16.6Ki   1.2%
   2.8%  15.4Ki .eh_frame_hdr    15.4Ki   1.1%
   1.8%  10.2Ki .dynsym          10.2Ki   0.7%
   1.5%  8.18Ki .dynstr          8.18Ki   0.6%
   0.9%  5.19Ki [Other]          7.05Ki   0.5%
   0.0%      0 .debug_aranges    4.59Ki   0.3%
   0.0%     60 [Unmapped]        3.53Ki   0.2%
   0.1%    568 [ELF Headers]     2.80Ki   0.2%
   0.4%  2.36Ki .gnu.hash        2.36Ki   0.2%
   0.3%  1.70Ki .bss                  0   0.0%
   0.0%     24 [None]                 0   0.0%
 100.0%   555Ki TOTAL           1.39Mi 100.0%
```

bloaty

```
$ bloaty -d compileunits demo
      VM SIZE                                                    FILE SIZE
 --------------                                                --------------
  39.2%   217Ki [None]                                          1.06Mi  76.3%
  31.4%   174Ki [Other]                                          174Ki  12.2%
   3.1%  17.1Ki src/dm.c                                        17.1Ki   1.2%
   2.2%  12.5Ki avs_commons/git/net/src/net.c                   12.5Ki   0.9%
   2.2%  12.4Ki src/observe.c                                   12.4Ki   0.9%
   2.0%  11.4Ki avs_commons/git/net/src/mbedtls.c               11.4Ki   0.8%
   1.7%  9.52Ki src/anjay.c                                     9.52Ki   0.7%
   1.6%  8.71Ki modules/attr_storage/src/attr_storage.c         8.71Ki   0.6%
   1.5%  8.50Ki src/servers/connection_info.c                   8.50Ki   0.6%
   1.5%  8.20Ki demo/objects/firmware_update.c                  8.20Ki   0.6%
   1.4%  8.01Ki src/coap/stream/server.c                        8.01Ki   0.6%
   1.4%  7.90Ki src/interface/bootstrap.c                       7.90Ki   0.6%
   1.4%  7.79Ki avs_commons/git/rbtree/src/rbtree.c             7.79Ki   0.5%
   1.4%  7.54Ki src/dm/handlers.c                               7.54Ki   0.5%
   1.3%  7.05Ki demo/demo_args.c                                7.05Ki   0.5%
   1.2%  6.65Ki src/interface/register.c                        6.65Ki   0.5%
   1.2%  6.40Ki src/coap/stream/stream.c                        6.40Ki   0.4%
   1.1%  6.38Ki modules/access_control/src/handlers.c           6.38Ki   0.4%
   1.1%  5.97Ki modules/at_sms/src/at_commands.c                5.97Ki   0.4%
   1.0%  5.71Ki demo/demo.c                                     5.71Ki   0.4%
   1.0%  5.65Ki demo/objects/test.c                             5.65Ki   0.4%
 100.0%   555Ki TOTAL                                          1.39Mi 100.0%
```

bloaty

```
$ bloaty -d symbols demo
      VM SIZE                                    FILE SIZE
 --------------                                --------------
  31.8%   176Ki [Unmapped]                     1.02Mi  73.4%
  60.0%   333Ki [Other]                         332Ki  23.3%
   4.3%  24.1Ki DEFAULT_CMDLINE_ARGS           24.1Ki   1.7%
   0.7%  3.64Ki demo_parse_argv                3.64Ki   0.3%
   0.1%     568 [ELF Headers]                  2.80Ki   0.2%
   0.3%  1.76Ki print_option_help              1.76Ki   0.1%
   0.3%  1.67Ki add_instance                   1.67Ki   0.1%
   0.3%  1.52Ki demo_init                      1.52Ki   0.1%
   0.2%  1.21Ki dev_read                       1.21Ki   0.1%
   0.2%  1.21Ki test_resource_read             1.21Ki   0.1%
   0.2%  1.03Ki rb_detach_fix                  1.03Ki   0.1%
   0.2%  1.03Ki _anjay_time_diff               1.03Ki   0.1%
   0.2%     1024 LOOKUP_TABLE.5671                0   0.0%
   0.2%      972 dm_discover                    972   0.1%
   0.2%      957 configure_socket               957   0.1%
   0.2%      948 ip_ping_handler                948   0.1%
   0.2%      939 avs_rbtree_detach__            939   0.1%
   0.2%      936 _anjay_dm_res_read_i64         936   0.1%
   0.2%      936 send_update                    936   0.1%
   0.2%      894 anjay_unregister_object        894   0.1%
   0.0%      126 [None]                           0   0.0%
 100.0%   555Ki TOTAL                         1.39Mi 100.0%
```

Pytania?

Prezentacja: https://goo.gl/IkHIaM

Więcej naszych wykładów:
https://github.com/AVSystem/Wyklady