Verifikacija softvera
6. čas

Ana Vulović *

Matematički fakultet, Univerzitet u Beogradu

1. april 2020

Sadržaj

1 Alat Valgrind **1**
1.1 Helgrind i DRD . . . 1
1.1.1 Loša upotreba intefejsa za programiranje `POSIX` niti . . . 1
1.1.2 Potencijalno blokiranje niti . . . 4
1.1.3 Trka za podacima . . . 6
1.1.4 Zadržavanje katanaca . . . 9

2 Fuzz testiranje **10**
2.1 Korpus . . . 11
2.2 Pokretanje . . . 11

1 Alat Valgrind

1.1 Helgrind i DRD

Helgrind je alat u sklopu programskog paketa Valgrind koji otkriva greške sinhronizacije prilikom upotrebe modela niti `POSIX`.

Glavne apstrakcije modela `POSIX` niti su: grupa niti koji deli zajednički adresni prostor, formiranje niti, čekanje na završetak izvršavanja funkcije niti, izlaz iz funkcije niti, muteks objekti, uslovne promenljive, čitaj-piši zaključavanje i semafori.

Problemi poput ovih imaju kao rezultat nereproduktivne, vremenski zavisne padove, mrtve petlje i druga loša ponašanja programa koja se mogu teško otkriti drugim sredstvima. *Helgrind* je svestan svih apstrakcija niti i prati njihove efekte. Najbolje radi ukoliko program koristi isključivo `POSIX` niti. Moguće ga je koristiti i za programe koji koriste druge standarde za niti, ali je neophodno opisati *Helgrind*-u njihovo ponašanje korišćenjem `ANNOTATE_*` makroa definisanih u `helgrind.h` zaglavlju).

DRD je *Valgrind*-ov alat za detekciju grešaka u *C* i *C++* programima koji koriste više niti. Alat radi za svaki program koji koristi niti POSIX standarda ili koji koriste koncepte koji su nadograđeni na ovaj standard.

	Helgrind	DRD
Mrtve petlje	DA	NE
Trka za podacima	DA	DA
Nepravilno korišćenje API-ja	DA	DA
Zadržavanje podataka	NE	DA

Alati *DRD* i *Helgrind* ne koriste iste algoritme za otkrivanje grešaka, samim tim ne otkrivaju iste tipove grešaka, iako imaju veliki broj poklapanja. Helgrind proizvodi izlaze koji su lakši za interpretaciju, dok *DRD* ima bolje performanse.

1.1.1 Loša upotreba intefejsa za programiranje `POSIX` niti

Mnoge implementacije interfejsa su optimizovane radi kraćeg vremena izvršavanja. Takve implementacije se neće buniti na određene greške (ako muteks otključa neka druga nit, a ne ona koja ga je zaključala).

Greške koje Helgrind i DRD pronalaze su:

*ana_vulovic@matf.bg.ac.rs

- Greške u otključavanju muteksa - kada je muteks nevažeći, nije zaključan ili je zaključan od strane druge niti;
- Greške u radu sa zaključanim muteksom - uništavanje nevažećeg ili zaključanog muteksa, rekurzivno zaključavanje nerekurzivnog muteksa, dealokacija memorije koja sadrži zaključan muteks;
- Prosleđivanje muteksa kao argumenta funkcije koja očekuje kao argument `reader-writer lock` i obrnuto;
- Greške sa `pthread barrier` - nevažeća ili dupla inicijalizacija, uništavanje `pthread barrier` koji nikada nije inicijalizovan ili koga niti čekaju ili čekanje na objekat koji nije nikada inicijalizovan;
- Greške prilikom korišćenja funkcije `pthread_cond_wait` - prosleđivanje nezaključanog, nevažećeg ili muteksa koga je zaključala druga nit;
- `Pthread` funkcija vrati kôd greške koji je potrebno dodatno obraditi;
- Greška kada se nit uništi, a još drži zaključanu promenljivu;
- Nekonzistentne veze između uslovnih promenljivih i njihovih odgovarajućih muteksa.

Ovakve greške mogu da dovedu do nedefinisanog ponašanja programa i do pojave grešaka u programima koje je kasnije veoma teško otkriti. *Helgrind* presreće pozive ka funkcijama biblioteke `pthread`, i zbog toga je u mogućnosti da otkrije veliki broj grešaka. Za sve `pthread` funkcije koje *Helgrind* presreće, generiše se podatak o grešci ako funkcija vrati kôd greške, iako *Helgrind* nije našao greške u kôdu. Provere koje se odnose na mutekse se takođe primenjuju i na `reader-writer lock`. Prijavljena greška prikazuje i primarno stanje steka koje pokazuje gde je detektovana greška. Takođe, ukoliko je moguće ispisuje se i broj linije u samom kôdu gde se greška nalazi. Ukoliko se greška odnosi na muteks, *Helgrind* će prikazati i gde je prvi put detektovao problematični muteks.

Primer 1.1 *01_bad_unlock.c*

Prevodimo program komandom `gcc -g -pthread -Wall 01_bad_unlock.c -o bad_lock`. Opcija `-pthread` je neophodna zbog `POSIX` niti. Naredbom `valgrind --tool=helgrind -v --log-file=bl.log ./bad_lock` pokrećemo *Helgrind*. Opcijom `--log-file=bl.log` zadajemo da nam se izveštaj upiše u datoteku `bl.log`. Vidimo da nam je prijavljen veliki broj grešaka.

```
==24477== ---Thread-Announcement------------------------------------------
==24477==
==24477== Thread #1 is the program's root thread
==24477==
==24477== ----------------------------------------------------------------
==24477==
==24477== Thread #1 unlocked a not-locked lock at 0xFFEFFF720
==24477==    at 0x4C326B4: ??? (in /usr/lib/valgrind/vgpreload_helgrind-amd64-linux.so)
==24477==    by 0x4008DE: nearly_main (01_bad_unlock.c:27)
==24477==    by 0x40096D: main (01_bad_unlock.c:49)
==24477==  Lock at 0xFFEFFF720 was first observed
==24477==    at 0x4C360BA: pthread_mutex_init (in /usr/lib/valgrind/vgpreload_helgrind-amd64-linux.so)
==24477==    by 0x4008B1: nearly_main (01_bad_unlock.c:23)
==24477==    by 0x40096D: main (01_bad_unlock.c:49)
==24477==  Location 0xffefff720 is 0 bytes inside local var "mx1"
==24477==  declared at 01_bad_unlock.c:18, in frame #1 of thread 1
```

Prijavljena nam je greška u da imamo u 27. liniji otključavanje nezaključanog muteksa. Poruka *was first observed* nam dodatno objašnjava gde je taj muteks prvi put primećen od strane *Helgrind*-a.

```
==24477== Thread #2 unlocked lock at 0xFFEFFF750 currently held by thread #1
==24477==    at 0x4C326B4: ??? (in /usr/lib/valgrind/vgpreload_helgrind-amd64-linux.so)
==24477==    by 0x40084D: child_fn (01_bad_unlock.c:11)
==24477==    by 0x4C34DB6: ??? (in /usr/lib/valgrind/vgpreload_helgrind-amd64-linux.so)
==24477==    by 0x4E476B9: start_thread (pthread_create.c:333)
==24477==  Lock at 0xFFEFFF750 was first observed
==24477==    at 0x4C360BA: pthread_mutex_init (in /usr/lib/valgrind/vgpreload_helgrind-amd64-linux.so)
==24477==    by 0x4008F2: nearly_main (01_bad_unlock.c:31)
==24477==    by 0x40096D: main (01_bad_unlock.c:49)
==24477==  Location 0xffefff750 is 0 bytes inside local var "mx2"
==24477==  declared at 01_bad_unlock.c:18, in frame #2 of thread 1
```

Prijavljen nam je pokušaj da nit koja nije zaključala muteks pokušava da ga otključa naredbom u liniji 11.

```
==24477== Thread #1 unlocked an invalid lock at 0xFFEFFF848
==24477==    at 0x4C326B4: ??? (in /usr/lib/valgrind/vgpreload_helgrind-amd64-linux.so)
==24477==    by 0x40094D: nearly_main (01_bad_unlock.c:41)
==24477==    by 0x40096D: main (01_bad_unlock.c:49)
==24477==
==24477== ----------------------------------------------------------------
==24477==
==24477== Thread #1's call to pthread_mutex_unlock failed
==24477==    with error code 22 (EINVAL: Invalid argument)
==24477==    at 0x4C327DA: ??? (in /usr/lib/valgrind/vgpreload_helgrind-amd64-linux.so)
==24477==    by 0x40094D: nearly_main (01_bad_unlock.c:41)
==24477==    by 0x40096D: main (01_bad_unlock.c:49)
```

Greška je u pokušaju otključavanja promenljive koja nije muteks.

```
==24477== Thread #1 unlocked a not-locked lock at 0xFFEFFF720
==24477==    at 0x4C326B4: ??? (in /usr/lib/valgrind/vgpreload_helgrind-amd64-linux.so)
==24477==    by 0x4008DE: nearly_main (01_bad_unlock.c:27)
==24477==    by 0x400990: main (01_bad_unlock.c:50)
==24477==  Lock at 0xFFEFFF720 was first observed
==24477==    at 0x4C360BA: pthread_mutex_init (in /usr/lib/valgrind/vgpreload_helgrind-amd64-linux.so)
==24477==    by 0x4008B1: nearly_main (01_bad_unlock.c:23)
==24477==    by 0x40096D: main (01_bad_unlock.c:49)
==24477==  Location 0xffefff720 is 0 bytes inside local var "mx1"
==24477==  declared at 01_bad_unlock.c:18, in frame #1 of thread 1
```

Greška je u tome što se u liniji 27 otključava već otključani muteks.

```
==24477== Thread #1: Attempt to re-lock a non-recursive lock I already hold
==24477==    at 0x4C320F4: ??? (in /usr/lib/valgrind/vgpreload_helgrind-amd64-linux.so)
==24477==    by 0x400901: nearly_main (01_bad_unlock.c:32)
==24477==    by 0x400990: main (01_bad_unlock.c:50)
==24477==  Lock was previously acquired
==24477==    at 0x4C321BC: ??? (in /usr/lib/valgrind/vgpreload_helgrind-amd64-linux.so)
==24477==    by 0x400901: nearly_main (01_bad_unlock.c:32)
==24477==    by 0x40096D: main (01_bad_unlock.c:49)
```

Prijavljuje nam da ponovno zaključavamo muteks koji je nit već ranije zaključala u liniji 32 funkcije `nearly_main` iz 49. linije `main` fje. Ponovno zaključavanje je nastalo u u liniji 32 funkcije `nearly_main` iz 50. linije `main` fje, jer se tada ponovo poziva funkcija `nearly_main()`. Problematični muteks je ostao zaključan iz prethodnog izvšavanja funkcije. Ako u na kraj funkcije otključamo muteks `mx2`,

```
pthread_mutex_unlock( &mx2 );
```

rešićemo i ovaj i naredni problem.

```
==24477== Thread #1: Exiting thread still holds 1 lock
==24477==    at 0x5129748: _Exit (_exit.c:31)
==24477==    by 0x5096FAA: __run_exit_handlers (exit.c:97)
==24477==    by 0x5097044: exit (exit.c:104)
==24477==    by 0x507D836: (below main) (libc-start.c:325)
```

Greška je što nit završava sa radom, ali nije otključala sve katance koje je držala.

Ispravimo greške jednu po jednu sve dok nam *Helgrind* prijavljuje nešto. Sada pokrećemo program kroz *Valgrind* sa alatom *drd*.

```
valgrind --tool=helgrind ./a.out
==8174== Helgrind, a thread error detector
==8174== Copyright (C) 2007-2015, and GNU GPL'd, by OpenWorks LLP et al.
==8174== Using Valgrind-3.11.0 and LibVEX; rerun with -h for copyright info
==8174== Command: ./a.out
```

```
==8174==
--------------------
==8174==
==8174== For counts of detected and suppressed errors, rerun with: -v
==8174== Use --history-level=approx or =none to gain increased speed, at
==8174== the cost of reduced accuracy of conflicting-access information
==8174== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)

ana@ana-lenovo:~/Fax/Nastava/2017_18/vs/priprema/06/helgrind i DRD$ valgrind --tool=drd ./a.out
==8394== drd, a thread error detector
==8394== Copyright (C) 2006-2015, and GNU GPL'd, by Bart Van Assche.
==8394== Using Valgrind-3.11.0 and LibVEX; rerun with -h for copyright info
==8394== Command: ./a.out
==8394==
--------------------
==8394== Mutex reinitialization: mutex 0xffefff570, recursion count 0, owner 1.
==8394==    at 0x4C36719: pthread_mutex_init (in /usr/lib/valgrind/vgpreload_drd-amd64-linux.so)
==8394==    by 0x4008A1: nearly_main (01_bad_unlock_fixed.c:23)
==8394==    by 0x40096B: main (01_bad_unlock_fixed.c:51)
==8394== mutex 0xffefff570 was first observed at:
==8394==    at 0x4C36719: pthread_mutex_init (in /usr/lib/valgrind/vgpreload_drd-amd64-linux.so)
==8394==    by 0x4008A1: nearly_main (01_bad_unlock_fixed.c:23)
==8394==    by 0x400948: main (01_bad_unlock_fixed.c:50)
==8394==
==8394== Mutex reinitialization: mutex 0xffefff5a0, recursion count 0, owner 1.
==8394==    at 0x4C36719: pthread_mutex_init (in /usr/lib/valgrind/vgpreload_drd-amd64-linux.so)
==8394==    by 0x4008D3: nearly_main (01_bad_unlock_fixed.c:31)
==8394==    by 0x40096B: main (01_bad_unlock_fixed.c:51)
==8394== mutex 0xffefff5a0 was first observed at:
==8394==    at 0x4C36719: pthread_mutex_init (in /usr/lib/valgrind/vgpreload_drd-amd64-linux.so)
==8394==    by 0x4008D3: nearly_main (01_bad_unlock_fixed.c:31)
==8394==    by 0x400948: main (01_bad_unlock_fixed.c:50)
==8394==
==8394==
==8394== For counts of detected and suppressed errors, rerun with: -v
==8394== ERROR SUMMARY: 2 errors from 2 contexts (suppressed: 14 from 6)
```

Prijavljuje nam da imamo reinicijalizovane mutekse. Prema dokumentaciji reinicijalizovanje inicijalizovanog muteksa ima nedefinisano ponašanje i stoga nam se prijavljuje greška. Muteksi iako su deklarisani u telu funkcije, ne ponašaju se kao obične lokalne promenljive. Dodajemo na kraj funkcije `nearly_main()` sledeće naredbe

```
pthread_mutex_destroy(&mx1);
pthread_mutex_destroy(&mx2);
```

Pokretanje program kroz *Valgrind* sa alatom *drd* ovaj put prolazi bez greške.

1.1.2 Potencijalno blokiranje niti

Helgrind prati redosled kojim niti zaključava promenljive. Na ovaj način *Helgrind* detektuje potencijalne delove kôda koji mogu dovesti do blokiranja niti. Na ovaj način je moguće detektovati greške koje se nisu javile tokom samog procesa testiranja programa, već se javljaju kasnije tokom njegovog korišćenja.

Ilustracija ovakvog problema:

- Pretpostavimo da je potrebno zaključati dve promenljive $M1$ i $M2$ da bi se pristupilo deljenom objekatu O
- Zatim da dve niti $T1$ i $T2$ žele da pristupe deljenoj promenljivoj O. Do blokoranja niti dolazi kada nit $T1$ zaključa $M1$, a u istom trenutku $T2$ zaključa $M2$. Nakon toga nit $T1$ ostane blokirana jer čeka da se otključa $M2$, a nit $T2$ ostane blokirana jer čeka da se otključa $T1$.

Helgrind kreira graf koji predstavlja sve promenljive koje se mogu zaključati, a koje je otkrio u prošlosti. Kada nit naiđe na novu promenljivu koju zaključava, graf se osveži i proverava se da li graf sadrži krug u kome se nalaze zaključane promenljive. Postojanje kruga u kome se nalaze zaključane promenljive je znak da je moguće da će se niti nekada u toku izvršavanja blokirati. Ako postoje više od dve zaključane promenljive u krugu problem je još ozbiljniji. Alat *DRD* ne otkriva ovaj tip grešaka.

Primer 1.2 *dinning_philosophers_deadlock.cpp*

```
/* Naive dining philosophers with inconsistent lock acquisition
   ordering. */

#include <pthread.h>
#include <stdlib.h>
#include <unistd.h>
#include <assert.h>
#include <stdio.h>

static pthread_t phil[5];
static struct {
  pthread_mutex_t m;
  char pad[120 - sizeof(pthread_mutex_t)];
} chop[5];

void* dine ( void* arg )
{
  int i;
  long n = (long)arg;
  long left = (long)arg;
  long right = (left + 1) % 5;


  for (i = 0; i < 3/*arbitrary*/; i++) {
    printf ("Philosopher %ld is thinking\tleft%ld right%ld\n" ,n,left,right);

    pthread_mutex_lock(&chop[left].m);
    printf ("Philosopher %ld took left\n",n);

    pthread_mutex_lock(&chop[right].m);
    printf ("Philosopher %ld took right\n",n);

    /* eating */
    printf ("Philosopher %ld is eating\n",n);
    usleep(15);

    printf ("Philosopher %ld finished eating\n",n);

    pthread_mutex_unlock(&chop[right].m);

    pthread_mutex_unlock(&chop[left].m);

  }
  return NULL;
}

int main ( void )
{
  long i;
  assert(sizeof(pthread_mutex_t) <= 120);

  for (i = 0; i < 5; i++)
    pthread_mutex_init( &chop[i].m, NULL);

  for (i = 0; i < 5; i++)
    pthread_create(&phil[i], NULL, dine, (void*)i );

  sleep(1);
```

```
  for (i = 0; i < 5; i++)
    pthread_join(phil[i], NULL);

  for(i=0;i<5;i++)
    pthread_mutex_destroy(&chop[i].m);

  return 0;
}
```

```
==15457== Thread #6: lock order "0x6010E0 before 0x6012C0" violated
==15457==
==15457== Observed (incorrect) order is: acquisition of lock at 0x6012C0
==15457==    at 0x4C321BC: ??? (in /usr/lib/valgrind/vgpreload_helgrind-amd64-linux.so)
==15457==    by 0x40090D: dine(void*) (02_dinning_philosophers_deadlock.cpp:26)
==15457==    by 0x4C34DB6: ??? (in /usr/lib/valgrind/vgpreload_helgrind-amd64-linux.so)
==15457==    by 0x4E476B9: start_thread (pthread_create.c:333)
==15457==
==15457==  followed by a later acquisition of lock at 0x6010E0
==15457==    at 0x4C321BC: ??? (in /usr/lib/valgrind/vgpreload_helgrind-amd64-linux.so)
==15457==    by 0x400937: dine(void*) (02_dinning_philosophers_deadlock.cpp:27)
==15457==    by 0x4C34DB6: ??? (in /usr/lib/valgrind/vgpreload_helgrind-amd64-linux.so)
==15457==    by 0x4E476B9: start_thread (pthread_create.c:333)
==15457==
==15457==  Lock at 0x6010E0 was first observed
==15457==    at 0x4C360BA: pthread_mutex_init (in /usr/lib/valgrind/vgpreload_helgrind-amd64-linux.so)
==15457==    by 0x400A0E: main (02_dinning_philosophers_deadlock.cpp:48)
==15457==  Address 0x6010e0 is 0 bytes inside data symbol "_ZL4chop"
==15457==
==15457==  Lock at 0x6012C0 was first observed
==15457==    at 0x4C360BA: pthread_mutex_init (in /usr/lib/valgrind/vgpreload_helgrind-amd64-linux.so)
==15457==    by 0x400A0E: main (02_dinning_philosophers_deadlock.cpp:48)
==15457==  Address 0x6012c0 is 480 bytes inside data symbol "_ZL4chop"
```

Ne treba ignorisati činjenicu da nam *Helgrind* prijavljuje narušavanje relacije, čak iako vidimo da se program lepo završava. Ako svaki filozof samo jednom ruča, može se dogoditi da nemamo zaključavanje, ali ako povećamo broj iteracija petlje na 1000 i da svaki filozof želi 1000 puta ručati, stvari se komplikuju.

Problem se rešava uvođenjem uređenja nad izborom viljuške, jer se tada nameće i relacija *„desilo se pre"*. Staviti raspored da filozof uvek bira prvo manju viljušku, pa zatim veću po rednom broju. Kako se sada ponaša program?

Primer 1.3 *deadlock.cpp*

1.1.3 Trka za podacima

Trka za podacima *(eng. Data races)* može da se javi usled nedostatka adekvatnog zaključavanja ili sinhronizacije. Pristup podacima bez adekvatnog zaključavanja ili sinhronizacije se odnosi na problem kada dve ili više niti pristupaju deljenom podatku bez sinhronizacije. Na ovaj način je moguće da dve ili više niti u istom trenutku pristupe deljenom objektu.

Primer 1.4 *Program u kom se koristi promenljiva bez adekvatne sinhronizacije,* `simpleDataRace.c`.

```
#include <pthread.h>
#include <stdio.h>

int var = 0;

void* child_fn ( void* arg ) {
  var=10; /* Unprotected relative to parent */
  return NULL;
}

int main ( void ) {
  pthread_t child1;
```

```
  pthread_create(&child1, NULL, child_fn, NULL);

  var=20; /* Unprotected relative to child */

  pthread_join(child1, NULL);
  printf("VAR = %d\n",var);

  return 0;
}
```

Algoritam detekcije pristupa promenljivoj bez sinhronizacije u okviru alata *Helgrind* implementira *„desilo se pre"* relaciju *(eng. „happens-before" relation)*. Na primer, kao u gore pokazanom programu, nit roditelj kreira nit dete. Zatim obe menjaju vrednost promenljive `var`, a zatim nit roditelja čeka da nit deteta izvrši svoju funkciju. Ovaj program nije dobro napisan jer ne možemo sa sigurnošću da znamo koja je vrednost promenljive `var` prilikom štampanja iste. Ako je nit roditelja brža od niti deteta, onda će biti štampana vrednost 10, u suprotnom će biti 20. Brzina izvršavanja niti roditelja i deteta je nešto na šta programer nema uticaja. Rešenje ovog problema je u zaključavanju promenljive `var`. Na primer, možemo da pošaljemo poruku iz niti roditelj nakon što ona promeni vrednost promenljive `var`, a nit dete neće promeniti vrednost promenljive `var` dok ne dobije poruku. Na ovaj način smo sigurni da će program ispisati vrednost 10. Razmena poruka kreira *„desilo se pre"* zavisnost između dve dodele vrednost: `var = 20;` se događa pre `var = 10;`. Takođe, sada više nemamo pristup promenljivoj bez sinhronizacije. Nije obavezno da šaljemo poruku iz niti roditelj. Možemo poslati poruku iz niti dete nakon što ona izvrši svoju dodelu. Na ovaj način smo sigurni da će se ispisati vrednost 20.

Alat *Helgrind* radi na istom ovom principu. On prati svaki pristup memorijskoj lokaciji. Ako se lokacija, u ovom primeru `var`, pristupa iz dve niti, Helgrind proverava da li su ti pristupi povezani sa *„desilo se pre"* vezom. Ako nisu, alat prijavljuje grešku o pristupu promenljivoj bez sinhronizacije. Ako je pristup deljenoj promenljivoj iz dve ili više programerske niti povezan sa *„desilo se pre"* relacijom, znači da postoji sinhronizacioni lanac između programskih niti koje obezbeđuje da se sam pristup odvija po tačno određenom redosledu, bez obzira na stvarne stope napretka pojedinačnih niti.

Standardne primitive `pthread` niti kreiraju *„desilo se pre"* relaciju:

- Ako je muteks otključan od strane niti $T1$, a kasnije ili odmah zaključan od strane niti $T2$, onda se sav pristup memoriji iz niti $T1$ pre otključavanja muteksa dešava pre nego onih pristupa iz niti $T2$ nakon njenog zaključavanja muteksa.
- Ista ideja se odnosi i na `reader-writer` zaključavanje promenljivih.
- Ako je kondiciona promenljiva signalizirana u funkciji niti $T1$ i ako druga nit $T2$ čeka na taj signal, da bi nastavila sa radom, onda se memorijski pristup u $T1$ dešava pre signalizacije, dok nit $T2$ vrši pristup memoriji nakon što izađe iz stanja čekanja na signal koji šalje nit $T1$.
- Ako nit $T2$ nastavlja sa izvršavanjem nakon što nit $T1$ oslobodi semafor, onda kažemo da postoji *„desilo se pre"* relacija između programskih niti $T1$ i $T2$.

Helgrind presreće sve gore navedene događaje i kreira graf koji predstavlja sve *„desilo se pre"* relacije u programu. Takođe, on prati sve pristupe memoriji u programu. Ako postoji pristup nekoj memorijskoj lokaciji u programu od strane dve niti i *Helgrind* ne može da nađe putanju kroz graf od jednog pristupa do drugog, generiše podatak o grešci u programu koji analizira. *Helgrind* ne proverava da li postoji pristup memorijskoj lokaciji bez sinhronizacije ukoliko se svi pristupi toj lokaciji odnose na čitanje sadržaja te lokacije. Dva pristupa memorijskoj lokaciji su u *„desilo se pre"* relaciji, i ako postoji proizvoljno dugačak lanac sinhronizacije događaja između ta dva pristupa. Ako nit $T1$ pristupa lokaciji M, zatim signalizira nit $T2$, koja kasnije signalizira nit $T3$ koja pristupa lokaciji M, kažemo da su ova dva pristupa između niti $T1$ i $T3$ u *„desilo se pre"* relaciji, iako između njih ne postoji direktna veza.

Naredbom `valgrind --tool=helgrind ./a.out` *Helgrind* nam generiše sledeći izveštaj.

```
==972== Thread #1 is the program's root thread
==972==
==972== ---Thread-Announcement----------------------------------------
==972==
==972== Thread #2 was created
==972==    at 0x51643DE: clone (clone.S:74)
==972==    by 0x4E46149: create_thread (createthread.c:102)
==972==    by 0x4E47E83: pthread_create@@GLIBC_2.2.5 (pthread_create.c:679)
==972==    by 0x4C34BB7: ??? (in /usr/lib/valgrind/vgpreload_helgrind-amd64-linux.so)
```

```
==972==    by 0x400705: main (simpleDataRace.c:12)
==972==
==972== ----------------------------------------------------------------
==972==
==972== Possible data race during read of size 4 at 0x60104C by thread #1
==972== Locks held: none
==972==    at 0x400706: main (simpleDataRace.c:13)
==972==
==972== This conflicts with a previous write of size 4 by thread #2
==972== Locks held: none
==972==    at 0x4006C7: child_fn (simpleDataRace.c:6)
==972==    by 0x4C34DB6: ??? (in /usr/lib/valgrind/vgpreload_helgrind-amd64-linux.so)
==972==    by 0x4E476B9: start_thread (pthread_create.c:333)
==972==  Location 0x60104c is 0 bytes inside global var "var"
==972==  declared at simpleDataRace.c:3
==972==
--972-- warning: evaluate_Dwarf3_Expr: unhandled DW_OP_ 0xf2
==972== ----------------------------------------------------------------
==972==
==972== Possible data race during write of size 4 at 0x60104C by thread #1
==972== Locks held: none
==972==    at 0x40070F: main (simpleDataRace.c:13)
==972==
==972== This conflicts with a previous write of size 4 by thread #2
==972== Locks held: none
==972==    at 0x4006C7: child_fn (simpleDataRace.c:6)
==972==    by 0x4C34DB6: ??? (in /usr/lib/valgrind/vgpreload_helgrind-amd64-linux.so)
==972==    by 0x4E476B9: start_thread (pthread_create.c:333)
==972==  Location 0x60104c is 0 bytes inside global var "var"
==972==  declared at simpleDataRace.c:3
```

Helgrind algoritam za detekciju pristupa memoriji bez sinhronizacije od prikupljeni informacija najpre ispisuje podatke gde su niti koje uzrokuju grešku napravljene. Glavni podatak o grešci počinje sa *„Possible data race during read"*. Zatim se ispisuje adresa gde se nesinhronizovan pristup memoriji dešava, kao i veličina memorije kojoj se pristupa. Prikazuje stek za obe niti, nit roditelja i nit deteta. U nastavku Helgrind ispisuje gde druga nit pristupa istoj lokaciji. Na kraju, *Helgrind* pokrenut sa opcijom `--read-var-info=yes` ispisuje i samo ime promenljive kojoj se pristupa, kao i gde u programu je ta promenljiva deklarisana. Na obe lokacija pristupa promenljivoj *Helgrind* ispisuje i sva zaključavanja koja postoje u tom trenutku, ali ovde ih nema ni na jednom mestu. Takva informacija je uglavnom korisna da se zaključi koja nit nije uspela da zaključa željeni objekat.

Kad imamo 2 poziva na steku kako naći uzrok trke za podacima?

Kao prvo, ispitati lokacije na koje se referiše u svakom steku. Trebalo bi da oba pokazuju na pristup istoj lokaciji, tj. promenljivoj. Potom treba utvrditi kako tu lokaciju obezbediti. Ukoliko je već bilo planirano da bude zaštićena muteksom, treba zaključati i otključati muteks na obe tačke pristupa, čak i ako je na jednom mestu planirano samo čitanje. *Helgrind* u momentu pristupa ispisuje i sve katance koje u tom trenutku drži nit koja pristupa lokaciji. Ukoliko smo zaboravili da zaključamo, tako možemo primetiti.

Možda je namera bila da se lokacija nekim drugim postupkom obezbedi, signalizacijom uslovnih promenljivih, npr. U tom slučaju treba pokušati da se pronađe dokaz da se raniji pristup u relaciji *„desilo se pre"* sa kasnijim pristupom. Činjenica je da *Helgrind* prijavljuje problem sa trkom za podacima ako nije uočio *„desilo se pre"* relaciju između dva pristupa. Ukoliko je u pravu, onda ni mi ne bi trebalo da možemo da nađemo takvu relaciju ni sa detaljnim ispitivanjem izvornog kôda. Ispitivanjem kôda ćemo uočiti i gde bi operacije za sinhronizaciju trebalo da stoje.

DRD ispisuje poruku svaki put kada otkrije da je došlo do trke za podacima u programu. Treba imati u vidu par sledećih stvari prilikom tumačenja ispisa koji nam alat *DRD* daje. Prvo, *DRD* dodeljuje svakoj niti jedinstveni broj `ID`. Brojevi koji se dodeljuju nitima kreću od jedan i nikada se ne koristi isti broj za više niti. Drugo, termin segment se odnosi na sekvencu uzastopnih operacija čuvanja, čitanja i sinhronizacije koje se izvršavaju u jednoj niti. Segment uvek počinje i završava se operacijom sinhronizacije. Analiza trke za podacima se izvršava između dva segmenta umesto između pojedinačnih operacija čitanja i čuvanja podataka, isključivo zbog učinka. Na kraju, uvek postoje bar dva pristupa memoriji prilikom trke za podacima. *DRD* štampa izveštaj o svakom pristupu memoriji koje je dovelo do trke za podacima.

Pokrenućemo za prethodni primer sada alat *DRD* naredbom `valgrind --tool=drd ./a.out`

```
==24411== Conflicting load by thread 1 at 0x0060104c size 4
==24411==    at 0x400706: main (simpleDataRace.c:13)
==24411== Allocation context: BSS section of /home/student/vs/06/helgrind/a.out
==24411== Other segment start (thread 2)
==24411==    at 0x51703E1: clone (clone.S:81)
==24411== Other segment end (thread 2)
==24411==    at 0x516A807: madvise (syscall-template.S:84)
==24411==    by 0x4E5394C: start_thread (pthread_create.c:432)
==24411==
==24411== Conflicting store by thread 1 at 0x0060104c size 4
==24411==    at 0x40070F: main (simpleDataRace.c:13)
==24411== Allocation context: BSS section of /home/student/vs/06/helgrind/a.out
==24411== Other segment start (thread 2)
==24411==    at 0x51703E1: clone (clone.S:81)
==24411== Other segment end (thread 2)
==24411==    at 0x516A807: madvise (syscall-template.S:84)
==24411==    by 0x4E5394C: start_thread (pthread_create.c:432)
```

Pristupi memoriji koji učestvuju u trci za podacima su označeni kao *konfliktni* sa prethodnim pristupom. Mora ih biti bar dva od koji je bar jedan sa namerom za menjanje podatka na memorijskoj lokaciji. Za jedan od pristupa se prikazuje potpun stek poziva, a za ostale delimični stek pozivi. Oni uglavnom uključuju početak i kraj segmenta pristupa. Ove informacije se mogu protumačiti na sledeći način:

- Krenuti od dna oba steka poziva, brojati stek okvire sa istim imenima funkcije, izvorne datoteke i broja linije.
- Naredni viši okvir na steku u oba steka poziva govori između kojih delova izvornog kôda se desio pristup memorijskoj lokaciji.

Problem u ovom primeru je što nemamo sinhronizaciju pristupa promenljivoj `var`. Uvođenjem muteksa i zaključavanjem istog prilikom pristupa promenljivoj, će rešiti problem.

Primer 1.5 *CalculateSum.c*

Primer 1.6 *RaceCondition.zip*

Program prebrojava brojeve manje od 1 000 000, deljive sa 7 ili 13. Pokrenuti program par puta pre profajliranja. Program treba da vrati broj 219782, ali gotovo uvek dobijamo malo manji broj.

Primer 1.7 *vatromet.zip*

Profajliramo program i šaljemo izlaz u log datoteku jer imamo veliki broj grešaka. To radimo komandom:
`valgrind --tool=helgrind -v --log-file=bl.log ./vatromet`
Veliki broj grešaka ima prijavljeno na više mesta `???` za ime funkcije jer su deo biblioteka i nemamo za njih informacije debagera. Pronađimo grešku koja ima veze sa kôdom koji nije bibliotečki i pokušajmo da zaključimo u čemu je stvar.

1.1.4 Zadržavanje katanaca

Prilikom rada niti može doći do pojave zadržavanja katanca *(eng. lock contention)*, pri kojoj jedna nit ne može da radi zbog blokiranja drugih niti. Dešava se da nit mora da čeka da muteks ili sinhronizacioni reader-write objekat budu otključani od strane druge niti. Ovakva pojava je nepoželjna u višenitnim sistemima, alat *DRD* otkriva ovaj tip problema. Zadržavanje katanaca stvara kašnjenja, koja bi trebalo da budu što je moguće kraća. Opcije `--exclusive-threshold=<n>` i `--shared-threshold=<n>` omogućavaju da DRD otkrije preterano zadržavanje katanca, tako što će prijaviti svako zadržavanje katanca koje je duže od zadatog praga. Alat *Helgrind* ne otkriva ovakav tip grešaka.

Primer 1.8 *hold_lock.c*

Pokrećemo komandu
`valgrind --tool=drd --exclusive-threshold=10 ./hard_lock -i 500`
Program dobija preko argumenata komandne linije `-i 500` da bi između zaključavanja i otključavanja muteksa bio uspavan 500ms.

DRD nam generiše izveštaj o nitima koje su prekoračile vreme od 10ms koje smo zadali opcijom
`--exclusive-threshold=10`

```
Locking mutex ...
==25008== Acquired at:
==25008==    at 0x4C3725B: pthread_mutex_lock (in /usr/lib/valgrind/vgpreload_drd-amd64-linux.so)
==25008==    by 0x400E1C: main (07_hold_lock.c:51)
==25008== Lock on mutex 0xffefff8f0 was held during 505 ms (threshold: 10 ms).
==25008==    at 0x4C3818C: pthread_mutex_unlock (in /usr/lib/valgrind/vgpreload_drd-amd64-linux.so)
==25008==    by 0x400E4D: main (07_hold_lock.c:55)
==25008== mutex 0xffefff8f0 was first observed at:
==25008==    at 0x4C36719: pthread_mutex_init (in /usr/lib/valgrind/vgpreload_drd-amd64-linux.so)
==25008==    by 0x400E04: main (07_hold_lock.c:49)
==25008==
Locking rwlock exclusively ...
==25008== Acquired at:
==25008==    at 0x4C3F484: pthread_rwlock_wrlock (in /usr/lib/valgrind/vgpreload_drd-amd64-linux.so)
==25008==    by 0x400E94: main (07_hold_lock.c:61)
==25008== Lock on rwlock 0xffefff920 was held during 502 ms (threshold: 10 ms).
==25008==    at 0x4C40955: pthread_rwlock_unlock (in /usr/lib/valgrind/vgpreload_drd-amd64-linux.so)
==25008==    by 0x400EAD: main (07_hold_lock.c:63)
==25008== rwlock 0xffefff920 was first observed at:
==25008==    at 0x4C3E705: pthread_rwlock_init (in /usr/lib/valgrind/vgpreload_drd-amd64-linux.so)
==25008==    by 0x400E88: main (07_hold_lock.c:60)
==25008==
Locking rwlock shared ...
Done.
```

Ovako dolazimo do saznanja u kojoj liniji je zaključan neki od katanaca i koliko se dugo držao dok nije otključan.

2 Fuzz testiranje

Jedna od tehnika pronalaženja grešaka u programu koja se pokaza vrlo efikasna je tehnika *faz testiranja* (eng. *fuzzing*). Prilikom takvog testiranja, programu se šalju neočekivani i loši ulazni podaci, u cilju otkrivanja ulaza koji izaziva greške. Da bi se kreirali faz testovi, standardni fazer alati će ili mutirati postojeće testove ili generisati testove na osnovu definisane gramatike ili skupa pravila. Još efikasniji način je faz testiranje vođeno pokrivenošću kôda (eng. *coverage guided fuzzing*). Prate se putanje prilikom izvršavanja da bi se generisali još efikasniji testovi u cilju postizanja maksimalne pokrivenosti kôda, tako da svaka grana u kôdu bude pokrivena. Tako rade postojeći alati za faz testiranje, kao što su American Fuzzy Lop (AFL), LLVM libFuzzer i HonggFuzz.

U najjednostavnijim slučajevima potrebno je da izolujemo funkcionalnost koju bismo da testiramo i ispišemo par pratećih linija kôda i kompajliramo i pokremo fazer. Fazer će izvršiti hiljade ili desetine hiljada testova po sekundi i sačuvaće one intersantne koji povećavaju pokrivenosti ili izazivaju određeno ponašanje.

Prvi korak u upotrebi *libFuzzer*-a na nekoj biblioteci koju bi trebalo testirati je kreiranje funkcije koja se u literaturi zove *fuzz target*. Ta funkcija prihvata niz bajtova i na njih primenjuje neko funkcije iz biblioteke koju testiramo.

```
//fuzz_me.cpp
extern "C" int LLVMFuzzerTestOneInput(const uint8_t *Data, size_t Size) {
  DoSomethingInterestingWithMyAPI(Data, Size);
  return 0;  // Non-zero return values are reserved for future use.
}
```

Ova funkcija ne zavisi ni u kojoj meri od *libFuzzer*-a, pa ju je moguće i poželjno koristiti i sa drugim alatima za faz testiranje npr. AFL i/ili Radamsa.

Važne činjenice o faz funkciji (fuzz taget):

- Alat za faz testiranje će izvršiti ovu funkciju mnogo puta sa različitim ulazima u okviru istog procesa.
- Mora tolerisati sve vrste ulaznih podataka (prazne, velike, loše formatirane, itd.)
- Ni za jedan ulaz se ne sme pozivati `exit()`.
- Može koristiti niti, ali idealno bi bilo da sve niti budu spojene na kraju funkcije.
- Mora biti deterministička, koliko god je moguće. Nedeterminizam, kao što su nasumične odluke nezavisne od ulaznih bajtova, učiniće faz testiranje neefikasnim.

- Mora biti brza. Pokušajte zaobići kubne i veće složenosti, logovanje ili preveliko trošenje memorije.
- Idealno bi bilo da ne modifikuje nijedno globalno stanje (iako to nije strogo neophodno).
- Obično je što uža funkcija to bolja, u smislu, ako funkcija može da parsira više formata ulaza, treba je izdeliti na nekoliko faz funkcija, po jedna po formatu.

LibFuzzer koristi sanitajzer za pokrivenost kôda kojim se instumentalizuje kôd da bi fazer mogao da dobija informacije o pokriventosti koje navode dalje faz testiranje. Može se podešavati opcijom `-fsanitize-coverage` *clang* kompajlera.

Trebalo bi uključiti još neki od sanitajzera, koji mogu pomoći u otkrivanju grešaka prilikom izvršavanja programa:

AddressSanitizer (ASAN) detektuje grešeke sa memorijom, uključuje se opcijom `-fsanitize=address`.

UndefinedBehaviorSanitizer (UBSAN) detektuje upotrebu različitih mogućnosti C/C++ koje su eksplicitno izdvojene jer uzrokuju nedefinisano ponašanje. Uključiti opcijom
`-fsanitize=undefined -fno-sanitize-recover=undefined`
ili nekom drugom *UBSAN* proverom, npr.
`-fsanitize=signed-integer-overflow -fno-sanitize-recover=undefined`
Može se kombinovati sa *ASAN* u istom prevođenju.

MemorySanitizer (MSAN) detektuje upotrebu neinicijalizovanih vrednosti, kôd čije ponašanje zavisi od sadržaja neinicijalizovane memorijske lokacije. Uključuje se opcijom
`-fsanitize=memory`
MSAN ne sme se kombinovati sa drugim sanitajzerima i treba ga koristiti u zasebnom prevođenju.

```
clang++  -fsanitize=fuzzer,address fuzz_me.cpp
```

2.1 Korpus

Fazeri, kao *libFuzzer*, koji koriste pokrivenost kôda za generisanje testova, oslanjaju se na korpus prethodno datih testova za test koji se testira. Korpus treba da bude idealno odemerena kolekcija validnih i nevalidnih ulaznih podataka. Fazer generiše nasumičnim mutacijama nove testove od datih iz trenutnog korpusa. Ako mutacija aktivira izvršavanje prethodno nepokrivene putanje u kôdu koji se testira, tada se mutacija čuva kao nov test u korpus i služi za buduće varijacije.

LibFuzzer može da radi bez inicijalnog test primera, ali će biti manje efikasan ako bibilioteka koja se testira prihvata kompleksne i struktuirane ulaze. Korpus može da posluži i za proveru, da li svi ranije pronađeni testovi i dalje rade bez problema kada se propuste kroz kôd.

Ako imamo veliki korpus, bilo da je dobijen faz testiranje ili drugačije, ukoliko želimo da ga smanjimo tako da zadržimo pokrivenost, to možemo uraditi upotrebom opcije `-merge=1`.

```
mkdir NEW_CORPUS_DIR  # Store minimized corpus here.
./my_fuzzer -merge=1 NEW_CORPUS_DIR FULL_CORPUS_DIR
```

Ista opcija se može koristiti da se doda još testova u postojeći korpus. Samo oni testovi koji povećavaju pokrivenost će biti dodati u prvi navedeni korpus.

```
./my_fuzzer -merge=1 CURRENT_CORPUS_DIR NEW_POTENTIALLY_INTERESTING_INPUTS_DIR
```

2.2 Pokretanje

Da bi se pokrenuo fazer sa korpusom testova, prvo treba kreirati direktorijum sa inicijalnim test primerima i potom ga pokrenuti:

```
mkdir CORPUS_DIR
cp /some/input/samples/* CORPUS_DIR

./my_fuzzer CORPUS_DIR  # -max_len=1000 -jobs=20 ...
```

Novi testovi biće dodavani u direktorijum korpusa.

Podrazumevano ponašanje je da fazer nastavlja da radi beskonačno ili bar dok greška nije pronađena. Svaki pad ili greška koju detektuje sanitajzer biće prijavljeni, fazer zaustavljen i test primer koji je prouzrokovao grešku biće sačuvan u datoteku (obično, `crash-<sha1>`, `leak-<sha1>` ili `timeout-<sha1>`).

Primer 2.1 *Primer* ***fuzz_me.cpp*** *prevodimo korišćenjem* clang-*a*

```
clang++  -fsanitize=fuzzer,address fuzz_me.cpp -o fuzz_me
```

Kreiramo direktorijum u koji će nam se čuvati generisani testovi.

```
mkdir testovi
```

i pokrećemo program

```
./fuzz_me  testovi/
```

Testovi se generišu variranjem postojećih iz direktorijuma testovi ukoliko ih ima. Novi testovi se dodaju u korpus samo ukoliko povećavaju pokrivenost kôda nakon svih već ranije generisanih testova. Ukoliko želimo da nam se generišu testovi samo od ASCII karaktera prilikom pozivanja programa treba proslediti i opciju `-only_ascii=1`, 0 je podrazumevana.

```
./fuzz_me -only_ascii=1 testovi/
```

Testovi se generišu u direktrijum, nama se prikazuje na ekran sledeći izlaz

```
INFO: Seed: 2050712183
INFO: Loaded 1 modules (7 guards): [0x77be60, 0x77be7c),
INFO: -max_len is not provided; libFuzzer will not generate inputs larger than 4096 bytes
INFO: A corpus is not provided, starting from an empty corpus
#0 READ units: 1
#2 INITED cov: 3 ft: 3 corp: 1/1b exec/s: 0 rss: 30Mb
#5 NEW    cov: 4 ft: 4 corp: 2/4b exec/s: 0 rss: 31Mb L: 3 MS: 3 CopyPart-ChangeBit-InsertByte-
#1114 NEW    cov: 5 ft: 5 corp: 3/122b exec/s: 0 rss: 33Mb L: 118 MS: 2 ShuffleBytes-InsertRepeatedBytes-
#176228 NEW    cov: 6 ft: 6 corp: 4/242b exec/s: 0 rss: 235Mb L: 120 MS: 1 CMP- DE: "U\x00"-
#555573 NEW    cov: 7 ft: 7 corp: 5/366b exec/s: 555573 rss: 401Mb L: 124 MS: 1 CMP- DE: "Z\x00\x00\x00"-
#1048576 pulse  cov: 7 ft: 7 corp: 5/366b exec/s: 524288 rss: 409Mb
#2097152 pulse  cov: 7 ft: 7 corp: 5/366b exec/s: 419430 rss: 409Mb
================================================================
==26069==ERROR: AddressSanitizer: heap-buffer-overflow on address 0x602000023bb3 at pc 0x000000539e73 bp 0
READ of size 1 at 0x602000023bb3 thread T0
.....
==26069==ABORTING
MS: 2 ChangeBinInt-CrossOver-; base unit: 1162a64ba4eb59dab49cfb3a8bb87abe708c3bb1
0x46,0x55,0x5a,
FUZ
artifact_prefix='./'; Test unit written to ./crash-0eb8e4ed029b774d80f2b66408203801cb982a60
Base64: RlVa
```

Opcijom `-max-len=N` možemo postaviti maksimalnu dužinu ulaza na neku drugu veličinu. Poruku `INFO: A corpus is not provided, starting from an empty corpus` označava da smo počeli sa praznim korpusom, kao što i jesmo. `READ` prati broj pročitanih testova iz korpusa, svaki novo dodati test je obelezen sa `NEW`. Broj na početku linije predstavlja redni broj generisanog testa. Ako je `NEW` u nastavku onda je i dodat u korpus, ako stoji `pulse`, tada je taj red ispisan samo zbog programera da bude obavešten da fazer i dalje radi. Vidimo da je fazer generisao 2097152 testova pre nego što je naišao na greku koja mu je prekinula rad. Dogodilo se prekoračenje memorije na hipu i generisan nam je izveštaj o prekidu rada `./crash-0eb8e4ed029b774d80f2b66408203801cb982a60` sa sve ulazom koji je grešku izazvao `FUZ`.

Pojašnjenje kôdova događaja i prateće statistike:

READ Fazer je pročitao sve od datih testova iz direktorijuma sa korpusima.

INITED Fazer je završio inicijalizaciju, što uključuje izvršavanje svakog inicijalnog testa kroz kôd koji se testira.

NEW Fazer je kreirao nov test koji pokriva do sad ne pokrivene delove kôda koji se testira. Ovaj test će biti sačuvan u direktorijumu glavnog korpusa.

REDUCE Fazer je pronašao bolji (kraći) test koji izaziva prethodno otkriveno ponašanje. Može se isključiti opcijom `-reduce_inputs=0`.

pulse Fazer je generisao 2n testotva. Poruka se generiše periodično da bi se korisnik uverio da fazer i dalje radi.

DONE Fazer je završio sa radom jer je došao da limita broja iteracija (zadaje se opcijom `-runs`) ili vremenskog limita (zadaje se opcijom `-max_total_time`).

RELOAD Fazer periodično ponovno učitava testove iz direktorijuma sa korpusom. Ovo omogućava fazeru da uzme u obzir testove koji su otkriveni drugim procesima fazera, prilikom paralelnog faz testiranja.

Svaka izlazna linija sadrži i izveštaj sa sledećim statistikama, kada nisu 0:

cov: Ukupan broj blokova kôda ili ivica pokrivenih izvršavanjem trenutnog korpusa.

ft: *libFuzzer* koristi različite signale da proceni pokrivenost kôda: pokrivenost ivica, brojače ivica, profajleri vrednosti, indirektne parove pozivaoc/pozvani, itd. Ovi signali su kombinovani i obeleženi sa (ft:).

corp: Broj testova u trenutnom korpusu u memoriji i njegova veličina u B.

lim: Trenutni limit u dužini novih ulaznih podataka u korpusu. Povećava se tokom vremena, dok ne dostigne maksimalnu dužinu. Ona se može zadati opcijom `-max_len`.

exec/s : Broj iteracija fazera u sekundi.

rss: Trenutno iskorišćeno memorije.

Za nove (NEW) događaje, izlazna linija sadrži i informacije:

L: Veličina novog testa u bajtovima.

MS: <n><operations > Broj i lista operacija mutacija koje su korišćene da bi se generisao nov ulaz.

Primer 2.2 *Dat je program koji nalazi rešenje jednačine čiji se koeficijenti učitavaju sa ulaza. Dopuniti kôd* `jednacina.c` *sa funkcijom neophodnom za faz testiranje, i zatim datu* `main` *funkciju staviti pod komentar. Pri testiranju generisane test primere sačuvati u poddirektorijum korpus.*

Program: `06_sredjeno/libFuzzer_primeri/jednacina.c`

```c
#include <assert.h>
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>

// C function for extended Euclidean Algorithm
int gcdExtended(int a, int b, int *x, int *y)
{
    // Base Case
    if (a == 0)
    {
        *x = 0;
        *y = 1;
        return b;
    }

    int x1, y1; // To store results of recursive call
    int gcd = gcdExtended(b%a, a, &x1, &y1);

    // Update x and y using results of recursive
    // call
    *x = y1 - (b/a) * x1;
    *y = x1;

    return gcd;
}

int find_any_solution (int a, int b, int c, int *x0, int *y0, int *g) {
  *g = gcdExtended (abs(a), abs(b), x0, y0);
  if (c % *g != 0)
    return 0;
  *x0 *= c / *g;
  *y0 *= c / *g;
  if (a < 0)   *x0 *= -1;
  if (b < 0)   *y0 *= -1;
  return 1;
}

// Napisati fuzz funkciju koja ce pozivati prethodnu
// i testirati je na fuzz podacima, po ugledu na predlozenu main funkciju, na primer.

int main(){
```

```
43    int a,b,c;
      int x, y, g;
45
      scanf("%d%d%d",&a,&b,&c);
47
      printf(" %d x + %d y = %d\n",a,b,c);
49
51    if( find_any_solution(a,b,c, &x, &y, &g) ){
        printf("resenje je: x = %d   y = %d\n",x,y);
53      printf("GCD %d\n",g);
        assert(a*x+b*y==c);
55    }
      else{
57      printf("Nema resenja\n");
        printf("GCD %d\n",g);
59      assert(a*x+b*y!=c);
      }
61
      return 0;
63  }
```

Dodajemo traženu funkciju. Prvi argument predstavlja adresu na koju se smeštaju generisani test podaci, a drugi argument je ukupna veličina svih generisanih podataka u bajtovima.

Program: `06_sredjeno/libFuzzer_primeri/jednacina_v1.c`

```
38  // Fuzz funkcija -- Prva verzija
    int LLVMFuzzerTestOneInput(const uint32_t *data, size_t size){
40    if (size < 3*sizeof(int))
        return 0;
42
      int a,b,c;
44
      a = data[0];
46    b = data[1];
      c = data[2];
48
      int x,y,g;
50
      if( find_any_solution(a,b,c,&x, &y, &g))
52      assert(a*x+b*y == c);
      else
54      assert(a*x+b*y != c);

56    return 0;
    }
```

Stavili smo da je niz `data` niz 32-bitnih podataka, potrebno je da je `size>= 3* sizeof(int)` jer nam je neophodno da pročitamo 3 parametra jednačine. Kako je data niz 32-bitnih podataka, možemo imati jednostavne dodele, poput `a = data[0]`.

Prevodimo sa informacijama debagera, da bismo u slučaju greške imali više informacija o lokaciji greške. Pokrećemo fazer:

```
mkdir korpus
clang -fsanitize=fuzzer,address  -g jednacina_v1.c
./a.out korpus
```

Dobijamo sledeći izveštaj na ekran:

```
1   ./a.out korpus/
    INFO: Seed: 317551772
3   INFO: Loaded 1 modules   (13 inline 8-bit counters): 13 [0x562fb0, 0x562fbd),
    INFO: Loaded 1 PC tables (13 PCs): 13 [0x53ea38,0x53eb08),
5   INFO:        0 files found in korpus/
    INFO: -max_len is not provided; libFuzzer will not generate inputs larger than 4096 bytes
7   INFO: A corpus is not provided, starting from an empty corpus
    #2  INITED cov: 2 ft: 3 corp: 1/1b lim: 4 exec/s: 0 rss: 27Mb
9     NEW_FUNC[1/2]: 0x5218c0 in gcdExtended /home/ana/Fax/nastava_matf/Nastava/vs/vezbe/06/06_sredjeno/
        libFuzzer_primeri/jednacina_v1.c:8
      NEW_FUNC[2/2]: 0x521d50 in find_any_solution /home/ana/Fax/nastava_matf/Nastava/vs/vezbe/06/06
        _sredjeno/libFuzzer_primeri/jednacina_v1.c:27
11  #1051 NEW    cov: 9 ft: 11 corp: 2/15b lim: 14 exec/s: 0 rss: 28Mb L: 14/14 MS: 4 ChangeByte-
        InsertRepeatedBytes-ChangeBit-CopyPart-
```

```
#1052 NEW    cov: 9 ft: 13 corp: 3/29b lim: 14 exec/s: 0 rss: 28Mb L: 14/14 MS: 1 ChangeBit-
#1053 NEW    cov: 9 ft: 16 corp: 4/43b lim: 14 exec/s: 0 rss: 28Mb L: 14/14 MS: 1 CrossOver-
#1054 NEW    cov: 11 ft: 18 corp: 5/57b lim: 14 exec/s: 0 rss: 28Mb L: 14/14 MS: 1 ChangeBit-
#1055 NEW    cov: 11 ft: 19 corp: 6/71b lim: 14 exec/s: 0 rss: 28Mb L: 14/14 MS: 1 ChangeBinInt-
#1060 NEW    cov: 11 ft: 22 corp: 7/85b lim: 14 exec/s: 0 rss: 28Mb L: 14/14 MS: 5 ChangeByte-
    CopyPart-ChangeByte-ChangeBinInt-ChangeByte-
#1062 NEW    cov: 12 ft: 24 corp: 8/99b lim: 14 exec/s: 0 rss: 28Mb L: 14/14 MS: 2 CopyPart-
    ShuffleBytes-
#1071 NEW    cov: 12 ft: 27 corp: 9/113b lim: 14 exec/s: 0 rss: 28Mb L: 14/14 MS: 4 ShuffleBytes-
    EraseBytes-InsertByte-CrossOver-
#1073 NEW    cov: 13 ft: 28 corp: 10/127b lim: 14 exec/s: 0 rss: 28Mb L: 14/14 MS: 2 ChangeBinInt-
    ChangeBinInt-
#1076 NEW    cov: 13 ft: 29 corp: 11/141b lim: 14 exec/s: 0 rss: 28Mb L: 14/14 MS: 3 ShuffleBytes-
    ChangeBit-ShuffleBytes-
#1089 NEW    cov: 13 ft: 30 corp: 12/155b lim: 14 exec/s: 0 rss: 28Mb L: 14/14 MS: 3 ChangeBit-
    ChangeBinInt-CopyPart-
#1090 NEW    cov: 13 ft: 31 corp: 13/169b lim: 14 exec/s: 0 rss: 28Mb L: 14/14 MS: 1 ShuffleBytes-
#1101 NEW    cov: 13 ft: 33 corp: 14/183b lim: 14 exec/s: 0 rss: 28Mb L: 14/14 MS: 1 ChangeBinInt-
#1102 NEW    cov: 13 ft: 34 corp: 15/197b lim: 14 exec/s: 0 rss: 28Mb L: 14/14 MS: 1 ChangeBit-
#1110 NEW    cov: 13 ft: 35 corp: 16/211b lim: 14 exec/s: 0 rss: 28Mb L: 14/14 MS: 3 ChangeBit-
    CopyPart-ChangeBinInt-
#1117 NEW    cov: 13 ft: 36 corp: 17/225b lim: 14 exec/s: 0 rss: 28Mb L: 14/14 MS: 2 ChangeBit-CMP-
    DE: "\x01\x00\x00\x00\x00\x00\x00\x00"-
AddressSanitizer:DEADLYSIGNAL
=================================================================
==3937==ERROR: AddressSanitizer: FPE on unknown address 0x000000521ea3 (pc 0x000000521ea3 bp 0
    x7ffdaa24e430 sp 0x7ffdaa24e340 T0)
    #0 0x521ea2 in find_any_solution /home/ana/Fax/nastava_matf/Nastava/vs/vezbe/06/06_sredjeno/
    libFuzzer_primeri/jednacina_v1.c:29:9
    #1 0x5224e8 in LLVMFuzzerTestOneInput /home/ana/Fax/nastava_matf/Nastava/vs/vezbe/06/06_sredjeno
    /libFuzzer_primeri/jednacina_v1.c:51:7
    #2 0x42c70a in fuzzer::Fuzzer::ExecuteCallback(unsigned char const*, unsigned long) (/home/ana/
    Fax/nastava_matf/Nastava/vs/vezbe/06/06_sredjeno/libFuzzer_primeri/a.out+0x42c70a)
    #3 0x42bf05 in fuzzer::Fuzzer::RunOne(unsigned char const*, unsigned long, bool, fuzzer::
    InputInfo*, bool*) (/home/ana/Fax/nastava_matf/Nastava/vs/vezbe/06/06_sredjeno/libFuzzer_primeri
    /a.out+0x42bf05)
    #4 0x42deb9 in fuzzer::Fuzzer::MutateAndTestOne() (/home/ana/Fax/nastava_matf/Nastava/vs/vezbe
    /06/06_sredjeno/libFuzzer_primeri/a.out+0x42deb9)
    #5 0x42eb85 in fuzzer::Fuzzer::Loop(std::vector<std::__cxx11::basic_string<char, std::
    char_traits<char>, std::allocator<char> >, fuzzer::fuzzer_allocator<std::__cxx11::basic_string<
    char, std::char_traits<char>, std::allocator<char> > > > const&) (/home/ana/Fax/nastava_matf/
    Nastava/vs/vezbe/06/06_sredjeno/libFuzzer_primeri/a.out+0x42eb85)
    #6 0x424780 in fuzzer::FuzzerDriver(int*, char***, int (*)(unsigned char const*, unsigned long))
     (/home/ana/Fax/nastava_matf/Nastava/vs/vezbe/06/06_sredjeno/libFuzzer_primeri/a.out+0x424780)
    #7 0x446af2 in main (/home/ana/Fax/nastava_matf/Nastava/vs/vezbe/06/06_sredjeno/
    libFuzzer_primeri/a.out+0x446af2)
    #8 0x7fed4ac56b6a in __libc_start_main (/lib/x86_64-linux-gnu/libc.so.6+0x26b6a)
    #9 0x41d929 in _start (/home/ana/Fax/nastava_matf/Nastava/vs/vezbe/06/06_sredjeno/
    libFuzzer_primeri/a.out+0x41d929)

AddressSanitizer can not provide additional info.
SUMMARY: AddressSanitizer: FPE /home/ana/Fax/nastava_matf/Nastava/vs/vezbe/06/06_sredjeno/
    libFuzzer_primeri/jednacina_v1.c:29:9 in find_any_solution
==3937==ABORTING
MS: 1 CopyPart-; base unit: 2c83c7c64e0a333d2a44787079c9b2e4c6af9cb6
0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0xe,0x2f,0x0,0x8,
\x00\x00\x00\x00\x00\x00\x00\x00\x00\x0e/\x00\x08
artifact_prefix='./'; Test unit written to ./crash-8580b18e503cfb9328f4dbaa74369db8ec11fd22
Base64: AAAAAAAAAAAAA4vAAg=
```

Iz 30. linije izveštaja saznajemo da imamo problem sa FloatingPointException (FPE) u 29. liniji kôda. Pretpostavljamo da je u pitanju deljenje sa 0. Ukoliko se želimo uveriti možemo se pogledati koji je test izazvao ovu grešku i pokušati da debagujemo program. U 45. liniji izveštaja vidimo test primer

```
0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0xe,0x2f,0x0,0x8,
```

Vratimo se na potpis faz funkcije

```
int LLVMFuzzerTestOneInput(const uint32_t *data, size_t size){
```

Iz njega se vidi da mi posmatramo generisani niz podataka kao niz 32-bitnih podataka, ukupne veličine `size` bajtova. Svaki generisani heksadekadni broj nam je jedan bajt. Kako su prvi 8 sve 0, dakle i a i b u programu biće postavljeni na 0, pa će za njihov najveći zajednički delilac gcd(a,b) vratiti 0.

Da bismo uklonili problem, možemo u funkciji `find_any_solution` dodati proveru pre računanja ostatka pri deljenju u liniji 29. Ako su a i b jednaki 0, da bi jednačina imala rešenja, neophodno je da i c bude 0, inače nema rešenja.

Program: 06_sredjeno/libFuzzer_primeri/jednacina_v2.c

```
int find_any_solution (int a, int b, int c, int *x0, int *y0, int *g) {
  *g = gcdExtended (abs(a), abs(b), x0, y0);
  if(*g == 0)
    return c == 0;
```

Pokrećemo ponovo fazer:

```
clang -fsanitize=fuzzer,address  -g jednacina_v2.c
./a.out korpus
```

Dobijamo novi izveštaj o grešci nakon dužeg kreiranja testova:

```
 ./a.out korpus
INFO: Seed: 3069155692
INFO: Loaded 1 modules   (14 inline 8-bit counters): 14 [0x562fb0, 0x562fbe),
INFO: Loaded 1 PC tables (14 PCs): 14 [0x53ea38,0x53eb18),
INFO:       31 files found in korpus
INFO: -max_len is not provided; libFuzzer will not generate inputs larger than 4096 bytes
INFO: seed corpus: files: 31 min: 1b max: 14b total: 408b rss: 27Mb
#32 INITED cov: 13 ft: 43 corp: 21/268b lim: 4 exec/s: 0 rss: 28Mb
#60 REDUCE cov: 13 ft: 43 corp: 21/267b lim: 4 exec/s: 0 rss: 28Mb L: 13/14 MS: 3 EraseBytes-
    ChangeBit-CopyPart-
#71 REDUCE cov: 13 ft: 43 corp: 21/265b lim: 4 exec/s: 0 rss: 28Mb L: 12/14 MS: 1 CrossOver-
```

```
#2097152  pulse  cov: 14 ft: 49 corp: 26/301b lim: 4096 exec/s: 299593 rss: 570Mb
a.out: jednacina_v2.c:56: int LLVMFuzzerTestOneInput(const uint32_t *, size_t): Assertion `a*x+b*y
    != c' failed.
==20562== ERROR: libFuzzer: deadly signal
    #0 0x4fb027 in __sanitizer_print_stack_trace /build/llvm-toolchain-8-F3l7P1/llvm-toolchain-8-8/
    projects/compiler-rt/lib/asan/asan_stack.cc:38:3
    #1 0x44633b in fuzzer::PrintStackTrace() (/home/ana/Fax/nastava_matf/Nastava/vs/vezbe/06/06
    _sredjeno/libFuzzer_primeri/a.out+0x44633b)
    #2 0x42b3e8 in fuzzer::Fuzzer::CrashCallback() (/home/ana/Fax/nastava_matf/Nastava/vs/vezbe
    /06/06_sredjeno/libFuzzer_primeri/a.out+0x42b3e8)
    #3 0x42b3af in fuzzer::Fuzzer::StaticCrashSignalCallback() (/home/ana/Fax/nastava_matf/Nastava/
    vs/vezbe/06/06_sredjeno/libFuzzer_primeri/a.out+0x42b3af)
    #4 0x7f70d98b5f3f  (/lib/x86_64-linux-gnu/libpthread.so.0+0x13f3f)
    #5 0x7f70d957fed6 in gsignal (/lib/x86_64-linux-gnu/libc.so.6+0x43ed6)
    #6 0x7f70d9561534 in abort (/lib/x86_64-linux-gnu/libc.so.6+0x25534)
    #7 0x7f70d956140e in __tls_get_addr (/lib/x86_64-linux-gnu/libc.so.6+0x2540e)
    #8 0x7f70d9571011 in __assert_fail (/lib/x86_64-linux-gnu/libc.so.6+0x35011)
    #9 0x522801 in LLVMFuzzerTestOneInput /home/ana/Fax/nastava_matf/Nastava/vs/vezbe/06/06_sredjeno
    /libFuzzer_primeri/jednacina_v2.c:56:5
    #10 0x42c70a in fuzzer::Fuzzer::ExecuteCallback(unsigned char const*, unsigned long) (/home/ana/
    Fax/nastava_matf/Nastava/vs/vezbe/06/06_sredjeno/libFuzzer_primeri/a.out+0x42c70a)
    #11 0x42bf05 in fuzzer::Fuzzer::RunOne(unsigned char const*, unsigned long, bool, fuzzer::
    InputInfo*, bool*) (/home/ana/Fax/nastava_matf/Nastava/vs/vezbe/06/06_sredjeno/libFuzzer_primeri
    /a.out+0x42bf05)
    #12 0x42deb9 in fuzzer::Fuzzer::MutateAndTestOne() (/home/ana/Fax/nastava_matf/Nastava/vs/vezbe
    /06/06_sredjeno/libFuzzer_primeri/a.out+0x42deb9)
    #13 0x42eb85 in fuzzer::Fuzzer::Loop(std::vector<std::__cxx11::basic_string<char, std::
    char_traits<char>, std::allocator<char> >, fuzzer::fuzzer_allocator<std::__cxx11::basic_string<
    char, std::char_traits<char>, std::allocator<char> > > > const&) (/home/ana/Fax/nastava_matf/
    Nastava/vs/vezbe/06/06_sredjeno/libFuzzer_primeri/a.out+0x42eb85)
    #14 0x424780 in fuzzer::FuzzerDriver(int*, char***, int (*)(unsigned char const*, unsigned long)
    ) (/home/ana/Fax/nastava_matf/Nastava/vs/vezbe/06/06_sredjeno/libFuzzer_primeri/a.out+0x424780)
    #15 0x446af2 in main (/home/ana/Fax/nastava_matf/Nastava/vs/vezbe/06/06_sredjeno/
    libFuzzer_primeri/a.out+0x446af2)
    #16 0x7f70d9562b6a in __libc_start_main (/lib/x86_64-linux-gnu/libc.so.6+0x26b6a)
    #17 0x41d929 in _start (/home/ana/Fax/nastava_matf/Nastava/vs/vezbe/06/06_sredjeno/
    libFuzzer_primeri/a.out+0x41d929)

NOTE: libFuzzer has rudimentary signal handlers.
      Combine libFuzzer with AddressSanitizer or similar for better crash reports.
SUMMARY: libFuzzer: deadly signal
MS: 4 ChangeBinInt-ChangeByte-InsertByte-CMP- DE: "\xf6\xe7/-"-; base unit:
    c66745c9da9499dfa0abf427cf7623e6369f0962
0x7e,0x13,0x4,0x84,0x0,0xe,0x1,0x0,0xf6,0xe7,0x2f,0x5f,0xff,
~\x13\x04\x84\x00\x0e\x01\x00\xf6\xe7/_\xff
artifact_prefix='./'; Test unit written to ./crash-3243026878bd940d69a0db70b5749699fbc2b44b
Base64: fhMEhAAOAQD25y9f/w==
```

U 48. liniji izveštaja, saznajemo da ne prolazi assert `a*x+b*y != c`, kada smo zapravo zaključili da jednačina nema rešenje. Potrebno je ponovo da protumačimo generisani test primer:

```
0x7e,0x13,0x4,0x84,0x0,0xe,0x1,0x0,0xf6,0xe7,0x2f,0x5f,0xff,
```

Da bismo dobro protumačili test primer neophodno je da poznajemo arhitekturu sistema na kom radimo. Virtulna mašina na kojoj radimo je little-endian. Što znači da ako se 4B koriste za 1 int podataka. Najniži bajt se zapisuje na nižoj adresi, tj prvi. Da bismo se uverili u tu činjenicu o arhitekturi dovoljno je da pokrenemo sledeći jednostavan program:

Program: `06_sredjeno/libFuzzer_primeri/test_za_endianess.c`

```c
#include <stdio.h>

int main(){
    int num = 1;

    if (*(char *)&num == 1)
    {
        printf("Little-Endian\n");
    }
    else
    {
        printf("Big-Endian\n");
    }
}
```

Na osnovu toga za parametar a, koriste se sledeći bajtovi: `0x7e,0x13,0x4,0x84`. Želimo da dobijemo odgovarajući heksadekadni broj koji bismo mogli da postavimo u `main` funkciji kao vrednost od `a` prilikom debugovanja. Ta vrednosti bi bila: `0x8404137e`. Analogno za `b` bi od `0x0,0xe,0x1,0x0` bilo `0x00010e00` i od `0xf6,0xe7,0x2f,0x5f` za `c`, `0x5f2fe7f6`.

Postavimo vrednosti ili u `main` funkciju polaznog koda ili u sam fazer i tako ignorisemo sve podatke koje on bude generisao.

Prevedemo ponovo program. Debagovanje ćemo vršiti uz pomoć QtCreatora, startovanjem i pokretanjem debagera za eksternu aplikaciju (Debug→Start Debugger→Start and Debug External Application) i štikliramo da stavi tačku prekida na početku `main` funkcije. Stavljamo dodatnu tačku prekida u liniji gde puca program zbog `assert`.

Selektujemo deo koda `a*x + b*y` i menija dobijenog na desni klik, izaberemo *Add Expression Evaluator* da bi nam pored lokalnih promenljivih sračunao i tu vrednost.

Vidimo da je sledeće vrednosti:

```
Locals
a -2080107650 int
b 69120 int
c 1596975094 int
g 10 int
x 1733 int
y -52153162 int
Inspector
Expressions
a*x+b*y 1596975094 int
Return Value
Tooltip
```

Dakle, zaista je vrednost sa leve strane jednaka `c`. To je posledica prekoračenja opsega celih brojeva.

Sad kada smo na to posumnjali, možemo pozvati fazer dodajući mu i `signed-integer-overflow` sanitajzer.

```
clang -fsanitize=fuzzer,address,signed-integer-overflow  -g jednacina_v2.c
./a.out korpus
```

Dobijamo sledeći izveštaj:

```
./a.out korpus/
INFO: Seed: 3110428304
INFO: Loaded 1 modules   (45 inline 8-bit counters): 45 [0x564190, 0x5641bd),
INFO: Loaded 1 PC tables (45 PCs): 45 [0x53fa00,0x53fcd0),
INFO:       59 files found in korpus/
INFO: -max_len is not provided; libFuzzer will not generate inputs larger than 4096 bytes
INFO: seed corpus: files: 59 min: 1b max: 14b total: 752b rss: 27Mb
jednacina_v2.c:33:7: runtime error: signed integer overflow: 15325 * 234881024 cannot be represented
    in type 'int'
SUMMARY: UndefinedBehaviorSanitizer: undefined-behavior jednacina_v2.c:33:7 in
```

```
jednacina_v2.c:34:7: runtime error: signed integer overflow: -62950811 * 234881024 cannot be
    represented in type 'int'
```

```
#21602  REDUCE cov: 38 ft: 91 corp: 27/313b lim: 177 exec/s: 0 rss: 31Mb L: 12/12 MS: 1 EraseBytes-
jednacina_v2.c:31:9: runtime error: division of -2147483648 by -1 cannot be represented in type 'int
    '
SUMMARY: UndefinedBehaviorSanitizer: undefined-behavior jednacina_v2.c:31:9 in
AddressSanitizer:DEADLYSIGNAL
=================================================================
==20116==ERROR: AddressSanitizer: FPE on unknown address 0x000000522271 (pc 0x000000522271 bp 0
    x7ffd7412d170 sp 0x7ffd7412d020 T0)
    #0 0x522270 in find_any_solution /home/ana/Fax/nastava_matf/Nastava/vs/vezbe/06/06_sredjeno/
    libFuzzer_primeri/jednacina_v2.c:31:9
    #1 0x522d5c in LLVMFuzzerTestOneInput /home/ana/Fax/nastava_matf/Nastava/vs/vezbe/06/06_sredjeno
    /libFuzzer_primeri/jednacina_v2.c:53:7
    #2 0x42c70a in fuzzer::Fuzzer::ExecuteCallback(unsigned char const*, unsigned long) (/home/ana/
    Fax/nastava_matf/Nastava/vs/vezbe/06/06_sredjeno/libFuzzer_primeri/a.out+0x42c70a)
    #3 0x42bf05 in fuzzer::Fuzzer::RunOne(unsigned char const*, unsigned long, bool, fuzzer::
    InputInfo*, bool*) (/home/ana/Fax/nastava_matf/Nastava/vs/vezbe/06/06_sredjeno/libFuzzer_primeri
    /a.out+0x42bf05)
    #4 0x42deb9 in fuzzer::Fuzzer::MutateAndTestOne() (/home/ana/Fax/nastava_matf/Nastava/vs/vezbe
    /06/06_sredjeno/libFuzzer_primeri/a.out+0x42deb9)
    #5 0x42eb85 in fuzzer::Fuzzer::Loop(std::vector<std::__cxx11::basic_string<char, std::
    char_traits<char>, std::allocator<char> >, fuzzer::fuzzer_allocator<std::__cxx11::basic_string<
    char, std::char_traits<char>, std::allocator<char> > > > const&) (/home/ana/Fax/nastava_matf/
    Nastava/vs/vezbe/06/06_sredjeno/libFuzzer_primeri/a.out+0x42eb85)
    #6 0x424780 in fuzzer::FuzzerDriver(int*, char***, int (*)(unsigned char const*, unsigned long))
     (/home/ana/Fax/nastava_matf/Nastava/vs/vezbe/06/06_sredjeno/libFuzzer_primeri/a.out+0x424780)
    #7 0x446af2 in main (/home/ana/Fax/nastava_matf/Nastava/vs/vezbe/06/06_sredjeno/
    libFuzzer_primeri/a.out+0x446af2)
    #8 0x7ff84b341b6a in __libc_start_main (/lib/x86_64-linux-gnu/libc.so.6+0x26b6a)
    #9 0x41d929 in _start (/home/ana/Fax/nastava_matf/Nastava/vs/vezbe/06/06_sredjeno/
    libFuzzer_primeri/a.out+0x41d929)

AddressSanitizer can not provide additional info.
SUMMARY: AddressSanitizer: FPE /home/ana/Fax/nastava_matf/Nastava/vs/vezbe/06/06_sredjeno/
    libFuzzer_primeri/jednacina_v2.c:31:9 in find_any_solution
==20116==ABORTING
MS: 2 ChangeByte-PersAutoDict- DE: "\x00\x00\x00\x80"-; base unit:
    c16ab7c0bd7afcaa222b7cb4563eb381b41f5fa5
0x2f,0xff,0xfe,0xff,0x0,0x0,0x0,0x80,0x0,0x0,0x0,0x80,
/\xff\xfe\xff\x00\x00\x00\x80\x00\x00\x00\x80
artifact_prefix='./'; Test unit written to ./crash-f75c56271890234fb3f7b560f75959963d6aeae8
Base64: L//+/wAAAIAAAACA
```

Primećujemo da je sanitajzer pronašao više situacija kada se dogodi prekoračenje, pa čak i situaciju kada broj ne može biti predstavljen kao int. Problem je izazvala kombinacija brojeva takva da resenja, nema ali kada se izračuna leva strana jednačine, napravi se baš toliko prekoračenje da je jednako desnoj strani. Možemo ukloniti assert naredbe, da bismo izbegli prekoračenje u računjanju, ali to ne može da ukloni mogućnost da se napravi test koji će oboriti program na isti način. Eventualno bi moglo da se razmisli o drugačijem pristupu računanju. Ovde se naša analiza završava.

Faz funkcija je mogla biti zadata drugačijim potpisom. Na primer:

Program: `06_sredjeno/libFuzzer_primeri/jednacina_v3.c`

```
// Fuzz funkcija -- Druga verzija
int LLVMFuzzerTestOneInput(const char *data, size_t size){

  // printf("\n\n\%ld\n", size);

  if (size < 3*sizeof(int))
    return 0;

  int a,b,c;

  a = *((int*)data);
  b = ((int*)data)[1];
  c = ((int*)data)[2];


  int x,y,g;

  if( find_any_solution(a,b,c,&x, &y, &g))
    printf("Imamo resenje, %d,%d\n",x,y);
```

```
  else
    printf("Nemamo resenje\n");

  return 0;
}
```

Tada je samo potrebno malo komplikovanije izdvojiti podatke iz genereisanog niza. Što se tiče tumačenja test primera, ostaje is to, da se bajtovi tumače kao što odgovara little-endian arhitekturi. Ako je test primer na primer `0xd,0x0,0x0,0x2d,0x0,0x0,0x0,0x80,0x0,0x0,0x0,0x80`, možemo staviti niz `data` u main funkciju i da se a, b i c dobijaju na isti način kao u faz funkciji.

```
int a,b,c;
char data[]= {0xd,0x0,0x0,0x2d,0x0,0x0,0x0,0x80,0x0,0x0,0x0,0x80};

a = *((int*)data);
b = ((int*)data)[1];
c = ((int*)data)[2];
```

Prilikom debagovanja kroz QtCreator, možemo staviti tačku prekida nakon dodeljivanja vrednostima parametrima i videti njihove decimalne vrednosti.

```
Locals
a 754974733 int
b -2147483648 int
c -2147483648 int
```

Desnim klikom na promenljivu `a` iz bočnog prikaza vrednosti lokalnih promenljivih, možemo izabrati *Change Value Display Format* i izabrati da nam se vrednost promenljive prikazuje kao heksadecimalna vrednost. Ponovimo to i za `b` i `c`.

```
Locals
a (hex) 2d00000d int
b (hex) ffffffff80000000 int
c (hex) ffffffff80000000 int
```

Primer 2.3 *Hearthbleed*

Heartbleed je popularo ime za grešku koja je postojala u biblioteci OpenSSL i konkretno na curenje memorije u implementaciji Heartbeat protokola u okviru OpenSSL bibilioteke.

Raspakovati arhivu `heartbleed.zip` i iz raspakovanog direktorijuma izvršiti komandu [1]

```
./openssl-1.0.1f/build.sh
```

koja će skinuti izvorni kôd bildovati i njega i kreirati `openssl-1.0.1f-libfuzzer`

Alternativno, možemo zaobići bildovanje i samo raspakovati arhivu `heartbleed_already_built.zip`.

Pokrenimo *libFuzzer* na faz funkciji `target.cc` iz direktorijuma `openssl-1.0.1f`, komandom:

```
./openssl-1.0.1f-libfuzzer
```

LibFuzzer nam za nekoliko sekundi nalazi grešku. [2]

```
...
#48364 REDUCE cov: 1666 ft: 802 corp: 40/419b lim: 21 exec/s: 12091 rss: 390Mb L: 12/21 MS:
4 ChangeByte-ChangeBit-CopyPart-ChangeByte-
=================================================================
==28875==ERROR: AddressSanitizer: heap-buffer-overflow on address 0x629000009748 at pc 0x0000004c67ec
 bp 0x7ffd0fa619c0 sp 0x7ffd0fa61170
READ of size 20560 at 0x629000009748 thread T0
  #0 0x4c67eb in __asan_memcpy (/home/student/vs/06/06_sredjeno/libFuzzer_primeri/heartbleed_already
_built/openssl-1.0.1f-libfuzzer+0x4c67eb)
```

[1] Potrebno napraviti sledeće simboličke linkove da bi bildovanje prošlo:
`sudo ln -s /usr/bin/clang-5.0 /usr/bin/clang`
`sudo ln -s /usr/bin/clang++-5.0 /usr/bin/clang++`

[2] Ukoliko ne vidimo imena datoteka ili brojeve linija u kojima je detektovana greška, potrebno je kreirati sledeći simbolički link da bi AddressSanitizer dobijao sve te informacije od llvm-a:
`sudo ln -s /usr/bin/llvm-symbolizer-5.0 /usr/bin/llvm-symbolizer`

```
  #1 0x524882 in tls1_process_heartbeat /home/student/vs/06/fuzzer/heartbleed/BUILD/ssl/t1_lib.c:2586:3
  #2 0x595229 in ssl3_read_bytes /home/student/vs/06/fuzzer/heartbleed/BUILD/ssl/s3_pkt.c:1092:4
  #3 0x599a01 in ssl3_get_message /home/student/vs/06/fuzzer/heartbleed/BUILD/ssl/s3_both.c:457:7
  #4 0x562bb9 in ssl3_get_client_hello /home/student/vs/06/fuzzer/heartbleed/BUILD/ssl/s3_srvr.c:941:4
  #5 0x55ec62 in ssl3_accept /home/student/vs/06/fuzzer/heartbleed/BUILD/ssl/s3_srvr.c:357:9
  #6 0x51811c in LLVMFuzzerTestOneInput /home/student/vs/06/fuzzer/heartbleed/./openssl-1.0.1f/target.cc:3
  #7 0x84fd24 in fuzzer::Fuzzer::ExecuteCallback(unsigned char const*, unsigned long) /home/student/vs/06/
fuzzer/heartbleed/./Fuzzer/FuzzerLoop.cpp:517:13
  #8 0x84f588 in fuzzer::Fuzzer::RunOne(unsigned char const*, unsigned long, bool, fuzzer::InputInfo*, boo
 /home/student/vs/06/fuzzer/heartbleed/./Fuzzer/FuzzerLoop.cpp:442:3
  #9 0x850c45 in fuzzer::Fuzzer::MutateAndTestOne() /home/student/vs/06/fuzzer/heartbleed/./Fuzzer/
FuzzerLoop.cpp:650:19
  #10 0x8515a5 in fuzzer::Fuzzer::Loop(std::vector<std::__cxx11::basic_string<char, std::char_traits<char>
std::allocator<char> >, fuzzer::fuzzer_allocator<std::__cxx11::basic_string<char, std::char_traits<char>,
std::allocator<char> > > > const&) /home/student/vs/06/fuzzer/heartbleed/./Fuzzer/FuzzerLoop.cpp:773:5
  #11 0x84731c in fuzzer::FuzzerDriver(int*, char***, int (*)(unsigned char const*, unsigned long))
/home/student/vs/06/fuzzer/heartbleed/./Fuzzer/FuzzerDriver.cpp:754:6
  #12 0x842a60 in main /home/student/vs/06/fuzzer/heartbleed/./Fuzzer/FuzzerMain.cpp:20:10
  #13 0x7f3df719982f in __libc_start_main /build/glibc-Cl5G7W/glibc-2.23/csu/../csu/libc-start.c:291
  #14 0x41cb18 in _start (/home/student/vs/06/06_sredjeno/libFuzzer_primeri/heartbleed_already_built/opens
1.0.1f-libfuzzer+0x41cb18)

0x629000009748 is located 0 bytes to the right of 17736-byte region [0x629000005200,0x629000009748)
allocated by thread T0 here:
...
```

Krenimo od informacije

```
#1 0x524882 in tls1_process_heartbeat /home/student/vs/06/fuzzer/heartbleed/BUILD/ssl/t1_lib.c:2586:3
```

koja nas upućuje na konkretnu liniju 2586 u konkretnoj datoteci. Otvorimo navedenu datoteku u *QT Creator* i pronalazimo konkretnu liniju i pomenutu `memcpy` naredbu
`memcpy(bp, pl, payload);` Problem koji nam se prijavljuje je čitanje van memorije koja je alocirana na hipu. Stavimo tačku prekida u pomenutoj liniji. Namestićemo *Qt Creator* da debaguje aplikaciju koju ćemo pokrenuti van njega. Biramo iz *Debug* menija :
Debug $\longrightarrow$ *Start Debugging* $\longrightarrow$ *Attach to Unstarted Application*
Izaberemo da nam izvršni program koji se debaguje bude naša aplikacija za faz testiranje : *./openssl-1.0.1f-libfuzzer* i kliknemo *Start Watching*.

Vraćamo se na konzolu i pokrećemo ponovo: *./openssl-1.0.1f-libfuzzer*. *Qt Creator* prekida izvršavanje na tački prekida. Dodajemo izraz koji želimo da pratimo, desnim klikom na pogled na vrednosti lokalnih vrednosti i izaberemo *Add Expression Evaluator* i unesemo sledeći izraz `s->s3->rbuf.len > payload`. Taj izraz uvek mora biti tačan jer je `s->s3->rbuf.len` veličina alocirane memorije na adresi koju čuva `pl`, a `payload` je veličina koja se iz `pl` želi prekopirati na adresu `bp`. To je suštinski problem bio sa ovom greškom. Što se kao odgovor može dobiti duži odgovor od poslatog.

\openssl-1.0.1\ssl\t1_lib.c

```
/* Read type and payload length first */
hbtype = *p++;
n2s(p, payload);
pl = p;

if (s->msg_callback)
    s->msg_callback(0, s->version, TLS1_RT_HEARTBEAT
        &s->s3->rrec.data[0], s->s3->rrec.length,
        s, s->msg_callback_arg);
```

\openssl-1.0.1g\ssl\t1_lib.c

```
if (s->msg_callback)
    s->msg_callback(0, s->version, TLS1_RT_HEARTBEAT,
        &s->s3->rrec.data[0], s->s3->rrec.length,
        s, s->msg_callback_arg);

/* Read type and payload length first */
if (1 + 2 + 16 > s->s3->rrec.length)
    return 0; /* silently discard */
hbtype = *p++;
n2s(p, payload);
if (1 + 2 + payload + 16 > s->s3->rrec.length)
    return 0; /* silently discard per RFC 6520 sec. 4 */
pl = p;
```

Ispravka u liniji 2570 `heartbleed/SRC/ssl/t1_lib.c`, kojom se proverava da li je dužina odgovora duža od poruke i odbacuje ukoliko ne odgovara.

```
if (s->msg_callback)
    s->msg_callback(0, s->version, TLS1_RT_HEARTBEAT,
                    &s->s3->rrec.data[0], s->s3->rrec.length,
        s, s->msg_callback_arg);

  /**********************/
  //   FIX
  if (1 + 2 + 16 > s->s3->rrec.length)
    return 0;  /* zero lenght, silently discard*/
  /* Read type and payload length first */
  hbtype = *p++;
  n2s(p, payload);

  if (1 + 2 + payload + 16 > s->s3->rrec.length)
    return 0;  /* exceeds lenght of a message, silently discard*/

  pl = p;

  /**********************/

if (hbtype == TLS1_HB_REQUEST){
```

Zbog prirode naše skripte *build.sh* , ova ispravka mora da se uradi u datoteci `heartbleed/SRC/ssl/t1_lib.c` i da se ponovo pokrene build.sh iz direkorijuma *openssl-1.0.1f*.

```
./openssl-1.0.1f/build.sh
```

Proveriti u pomenutoj skripti da li je pod komentarom naredba za `git`. Ukoliko nije, skinuće nam ponovo repozitorijum i izgubićemo našu ispravku.

Nakon build-ovanja pokrećemo ponovo libfuzzer, ovaj put on nastavlja da radi i radi, bez prijavljivanja greške. Prekidamo ga posle nekog vremena sa `CTRL+c`.

Primer 2.4 *woff*

Raspakovati arhivu `woff.zip` i iz raspakovanog direktorijuma izvršiti komandu

```
./woff2-2016-05-06/build.sh
```

koja će skinuti izvorni kôd bildovati i njega i kreirati `woff2-2016-05-06-libfuzzer`

Alternativno, možemo zaobići bildovanje i samo raspakovati arhivu `woff_already_built.zip`.

Pokrenimo *libFuzzer* na faz funkciji `target.cc` iz direktorijuma `woff2-2016-05-06`, komandom:

```
./woff2-2016-05-06-libfuzzer
```

Pratimo pokrivenost blokova kôda koja se postiže korpusom testova koje gradi (broj pored `cov:`), vidimo da se vrlo brzo dostiže neki broj i da svi naredni testovi vrlo sporo, povećavaju pokrivenost.

Sada kreirajmo direktorijum `MOJ_KORPUS` i sad pokrenimo fazer tako da nam u novi direktorijum generiše testove, ali ćemo mu i proslediti direktorijum `seeds` sa inicijalni testovima.

```
mkdir MOJ_KORPUS
./woff2-2016-05-06-libfuzzer MOJ_KORPUS/ seeds/
```

Primetimo koliko se brže postiže desetostruko veća pokrivenost ukoliko su zadati dobri inicijalni test primeri.

Literatura

[1] Helgrind priručnik

[2] DRD priručnik

[3] Karadžić A., Alat Valgrind - Implementacija FPXX za arhitekturu MIPS, master rad, Matematički fakultet, Beograd, 2018

[4] https://releases.llvm.org/5.0.0/docs/LibFuzzer.html

[5] https://github.com/google/fuzzer-test-suite/blob/master/tutorial/libFuzzerTutorial.md

[6] https://en.wikipedia.org/wiki/Heartbleed

[7] Chandra B., A technical view of theOpenSSL 'Heartbleed', [link]