UNIVERZITET U BEOGRADU
MATEMATIČKI FAKULTET

Đorđe Todorović

PODRŠKA ZA NAPREDNU ANALIZU PROMENLJIVIH LOKALNIH ZA NITI POMOĆU ALATA GNU GDB

master rad

Beograd, 2018.

Mentor:

dr Mika Mikić, redovan profesor
Univerzitet u Beogradu, Matematički fakultet

Članovi komisije:

dr Ana Anić, vanredni profesor
University of Disneyland, Nedođija

dr Laza Lazić, docent
Univerzitet u Beogradu, Matematički fakultet

Datum odbrane: _____

Mami, tati i dedi

Sadržaj

Glava 1

Uvod

Glava 2

Kako rade debageri?

Debager (eng. *debagger*) je softverski alat koji koriste programeri za testiranje, analizu i otklanjanje grešaka u programima. Sam proces korišćenja takvih alata nazivamo debagovanjem (eng. *debugging*). Debageri mogu startovati neki proces i debagovati ga, ili „nakačiti" se na neki proces koji je već u fazi rada. Kada isti preuzme kontrolu nad programom može ga izvršavati instrukciju po instrukciju, postavljati tačke prekida (eng. *brakpoints*) itd. Neki debageri imaju mogućnost izvršavanja funkcija programa koji se debaguje, uz ograničenje da isti pripada istoj procesorskoj arhitekturi kao i domaćinski sistem na kojem se debager izvršava, ili čak menjati strukturu programa prateći propratne efekte.

Podršku debagerima, u opštem slučaju, daju operativni sistemi, kroz sistemske pozive koji omogućavaju tim alatima da pokrenu i preuzmu kontrolu nad nekim drugim procesom.

U radu će detaljno biti obrađen rad UNIX-olikih, posebno Linux debagera. Windows debageri i programski prevodioci ne prate standard DWARF[2] prilikom baratanja sa debag informacijama namenjene za taj operativni sistem, već isti konsultuju standard *Majkrosoft CodeView*, više informacija možete pronaći na [5].

2.1 Linux debageri

Pre svega, definišimo neke od osnovnih pojmova Linux programiranja koje ćemo često pominjati u radu. Osvrnimo se prvo na fajl format izvršnih fajlova, deljenih biblioteka, objektnih fajlova i datoteka jezgara ELF (eng. *Executable and Linkable Format*)[1]. ELF sadrži razne informacije o samom fajlu koji je podeljen u dva dela: ELF zaglavlje i podaci fajla. ELF zaglavlje sadrži informacije o arhitekturi

za koju je program preveden i definiše da li program koristi 32-bitni ili 64-bitni adresni prostor. Zaglavlje 32-bitnih programa je dužine 52 bajta, dok kod 64-bitnih programa isti je dužine 64 bajta. Podaci fajla mogu sadržati programsku tabelu zaglavlja (eng. *Program header table*), sekcijisku tabelu zaglavlja (eng. *Section header table*) i ulazne tačke prethodne dve tabele. Slika 2.1 prikazuje primer prikaza fajla u formatu ELF.

```
ELF Header:
  Magic:   7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
  Class:                             ELF64
  Data:                              2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              EXEC (Executable file)
  Machine:                           Advanced Micro Devices X86-64
  Version:                           0x1
  Entry point address:               0x4005c0
  Start of program headers:          64 (bytes into file)
  Start of section headers:          7944 (bytes into file)
  Flags:                             0x0
  Size of this header:               64 (bytes)
  Size of program headers:           56 (bytes)
  Number of program headers:         10
  Size of section headers:           64 (bytes)
  Number of section headers:         39
  Section header string table index: 36

Section Headers:
  [Nr] Name              Type             Address           Offset
       Size              EntSize          Flags  Link  Info  Align
  [ 0]                   NULL             0000000000000000  00000000
       0000000000000000  0000000000000000           0     0     0
  [ 1] .interp           PROGBITS         0000000000400270  00000270
       000000000000001c  0000000000000000   A       0     0     1
  [ 2] .note.ABI-tag     NOTE             000000000040028c  0000028c
       0000000000000020  0000000000000000   A       0     0     4
  [ 3] .note.gnu.build-i NOTE             00000000004002ac  000002ac
       0000000000000024  0000000000000000   A       0     0     4
  [ 4] .gnu.hash         GNU_HASH         00000000004002d0  000002d0
       000000000000001c  0000000000000000   A       5     0     8
  [ 5] .dynsym           DYNSYM           00000000004002f0  000002f0
       00000000000000d8  0000000000000018   A       6     1     8
```

Slika 2.1: ELF fajl format počitan alatom *objdump*.

Nakon fajl formata ELF, jako bitan fajl format je DWARF, koji predstavalja format zapisa debag informacija koje debageri koriste kada analiziraju programe. DWARF je od posebne važnosti za rad te će isti biti opisan detaljno u nastavku.

Operativni sistem GNU Linux pruža sistemski poziv `ptrace` [3] koji debagerima omogućava rad. Ovaj sistemski poziv omogućava jednom procesu kontrolu nad izvršavanjem nekog drugog procesa i menjanje memorije i registara istog. Potpis ove funkcije je:

```
long ptrace(enum __ptrace_request request, pid_t pid, void *addr, void
*data);
```

Prvi argument sistemskog poziva predstavlja informaciju kojom operativnom sistemu jedan proces, ne nužno debager, ukazuje na nameru preuzimanja kontrole drugog procesa. Ukoliko isti ima vrednost `PTRACE_TRACEME` to ukazuje na nameru praćenja (eng. *tracing*) određenig procesa, `PTRACE_PEEKDATA` i `PTRACE_POKEDATA` redom ukazuju na nameru čitanja i pisanja memorije, `PTRACE_GETREGS` i `PTRACE_SETREGS` se odnose na čitanje i pisanje registara itd. Drugi argument sistemskog poziva `pid` ukazuje na identifikacioni broj ciljanog procesa. Treći i četvrti argument se po potrebi koriste u zavisnosti od namere korišćenja sistemskog poziva `ptrace` za čitanje ili pisanje sa adrese datom trećim argumentom, pri tom baratajući podacima na adresi zadatoj četvrtim argumentom. To znači ukoliko se koristi `PTRACE_TRACEME` poslednja tri argumenta sistemskog poziva se ignorišu. Ukoliko se koristi `PTRACE_PEEKDATA` sa adrese `addr` se čita jedna reč iz memorije.

Tačke prekida

Postoje dve vrste tačaka prekida (eng. *brakpoints*): softverske i hardverske [4].

Osvrnimo se prvo na softverske tačke prekida. Postavljanje tačaka prekida predstavlja jednu od najkorišćenijih mogućnosti debagera, te stoga navedimo par smernica kako je ista realizovana, u opštem slučaju. Ali takođe treba napomenuti da ne postoji jedinstveni poziv nekog sistemskog poziva za postavljanje tačke prekida, već se ista obavlja kao kombinacija više mogućnosti sistemskog poziva `ptrace`. Opišimo ceo postupak na jednostavnom primeru. Program je preveden za procesorsku arhitekturu Intel x86-64 i asemblerski kod `main` funkcije primera izgelda kao na primeru 2.2.

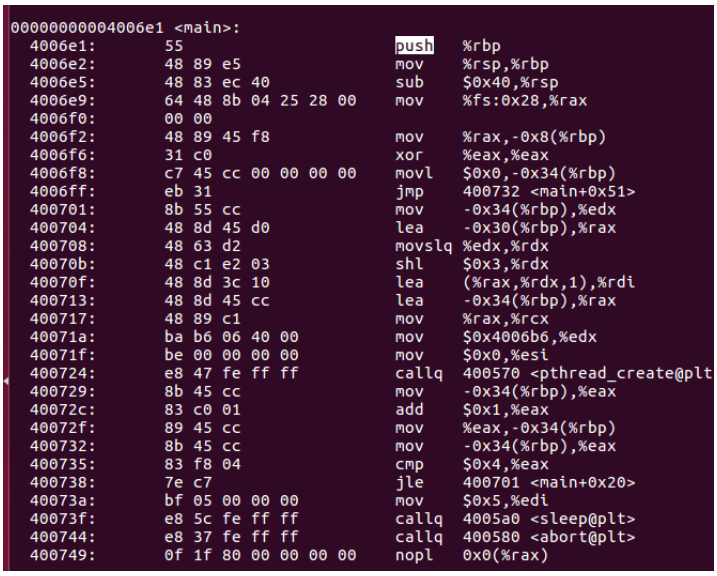
Primera radi, želimo da postavimo tačku prekida na treću po redu instrukciju funkcije `main`:

```
48 89 e5 mov %rsp,%rbp
```

Da bismo to uradili, menjamo prvi bajt instrukcije sa posebnom magičnom vrednošću, obično `0xCC`, i kada izvršavanje dostigne do tog dela koda ono će se zaustaviti na tom mestu.

Pošto `0x48` menjamo sa `0xcc` i na tom mestu u kodu dobijamo instrukciju:

```
cc 89 e5 int3
```



Slika 2.2: Asemblerski kod `main` fukcije test primera na x86-64 platformi.

Instrukcija `int3` je posebna instrukcija procesorske arhitekture Intel x86-64, koja izazva softverski prekid. Kada registar programski brojač (eng. *CPU register pc*) stigne do `int3` instrukcije izvršavanje se zaustavalja na toj tački. Debager je već upoznat od strane korisnika i svestan je da je tačka prekida postavljena te isti čeka na signal koji ukazuje na to da je program dostigao do instrukcije prekida. Operativni sistem prepoznaje `int3` instrukciju, poziva se specijalni obrađivač tog signala (na Linux sistemima `do_int3()`), koji dalje obaveštava debager šaljući mu signal sa kodom `SIGTRAP` koji on obrađuje na željeni način. Treba napomenuti da ovo važi za procesorsku arhitekturu Intel x86-64, instrukcija prekida za arhitekture kao što su ARM, MIPS, PPC itd., se drugačije kodira, ali postupak implementacije tačaka prekida je isti.

Ukoliko želimo da stavimo tačku prekida eksplicitno na funkciju `main`, za to koristimo posrednika u vidu DWARF debug informacija. U tom slučaju debager traži element DWARF stabla koji ukazuje na informacije o `main` funkciji, i odatle dohvata informaciju na kojoj adresi u memoriji se nalazi prva mašinska instrukcija date funkcije. Na slici 2.3 vidimo DWARF element koji opisuje funkciju uz pomoć atributa. Debager će pročitati `DW_AT_low_pc` atribut i na tu adresu postaviti `int3` instrukciju. Napomenimo da DWARF debag simbole generišemo uz pomoć `-g` opcije kompajlera, ali o istim će biti navedena opširna analiza u posebnoj sekciji.

Hardverske tačke prekida su direktno povezane sa hardverom u vidu specijalnih

```
<1><bd>: Abbrev Number: 6 (DW_TAG_subprogram)
   <be>   DW_AT_external    : 1
   <be>   DW_AT_name        : (indirect string, offset: 0x21): main
   <c2>   DW_AT_decl_file   : 1
   <c3>   DW_AT_decl_line   : 15
   <c4>   DW_AT_prototyped  : 1
   <c4>   DW_AT_type        : <0x57>
   <c8>   DW_AT_low_pc      : 0x4006e1
   <d0>   DW_AT_high_pc     : 0x68
   <d8>   DW_AT_frame_base  : 1 byte block: 9c        (DW_OP_call_frame_cfa)
   <da>   DW_AT_GNU_all_tail_call_sites: 1
   <da>   DW_AT_sibling     : <0xfa>
```

Slika 2.3: `main` funkcija prikazana DWARF subprogramom.

registara. Postavlja se na određenu adresu i harverski *watchpoint* montri na zadatu adresu i može signalizirati razne promene na istoj, npr. čitanje, pisanje ili izvršavanje, što im daje prednost u odnosu na softverske tačke prekida. Mane u odnosu na softverske tačke prekida su performanse, gde su iste neuporedivo sporije, i takođe neophodna hardverska podrška za korišćenje hardverskih tačaka prekida.

Koračanje

Pod procesom koračanja (eng. *stepping*) kroz program podrazumevamo izvršavanje programa sekvencu po sekvencu. Sekvenca može biti jedna procesorska instrukcija, linija koda ili pak neka funkcija programa koji se debaguje [4].

Instrukcijsko koračanje na platformi Intel x86-64 je direktno omogućeno kroz sistemski poziv `ptrace`:

```
ptrace(PTRACE_SINGLESTEP, debuggee_pid, nullptr, nullptr);
```

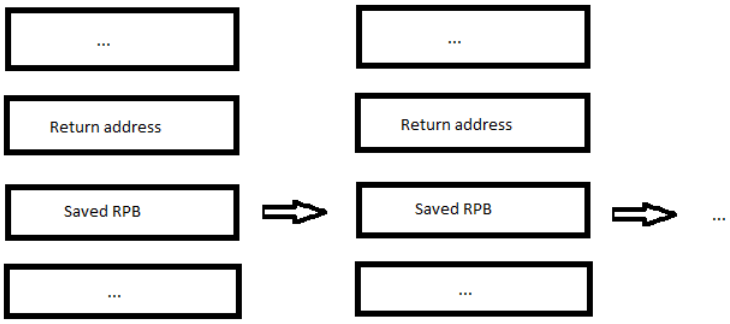
Operativni sistem će dati signal kada je korak izvršen.

Pored instrukcijkog koračanja pomenućemo još jednu vrstu zakoračavanja u neku funkciju koja je pozvana `call` ili `jump` instrukcijom. Komanda koja nam to omogućava jeste `step in`.

Treba napomenuti da postoje arhitekture za koje ovo ne važi, kao npr. platforma ARM, koja nema hardversku podršku za instrukcijsko koračanje i za njih se koračanje implementira na drugačiji način, uz pomoć emulacije instrukcija, ali u ovom radu neće biti opširno o tome.

Izlistavanje pozivanih funkcija

Objasnimo komandu izlistavanje pozivanih funkcija (eng. *backtrace*) posmatrajući organizaciju stek okvira (eng. *stack frames*) na platformi Intel x86-64 [4].



Slika 2.4: Primerređanja stek okvira na x86-64 platformi.

Na slici 2.4 navedeni su stek okviri za dva funkcijska poziva. Pre povratne vrednosti funkcije obično seređaju argumenti funkcije. Sačuvana adresa u registru `RBP` jeste adresa stek okvira svog pozivaoca. Prateći iste kao elemente povezane liste dolazimo do svih pozivanih funkcija do zadate tačke. Ako se pitamo kako debager ima informaciju o imenu funkcije odgovor je u tome što pretražuje DWARF stablo sa debag informacijama, tražeći `DW_TAG_subprogram` sa odgovarajućom povratnom adresom, pritom čitajući `DW_AT_name` atribut tog elementa.

Čitanje vrednosti promenljivih

Za čitanje vrednosti promenljivih u programu, debager pretražuje DWARF stablo tražeći promenljivu sa zadatim imenom. U slučaju lokalnih promenljivih, traži se `DW_TAG_variable` element čiji `DW_AT_name` odgovara navedenoj promenljivoj. Kada se ista pronađe konsultuje se `DW_AT_location`, koji ukazuje na lokaciju gde se vred-

nost promenljive nalazi. Ukoliko ovaj atribut nije naveden debager će vrednost takve promenljive smatrati kao optimizovanu prijavljujući informaciju o tome [4].

Glava 3

GNU GDB alat

Glava 4

DWARF format

Glava 5

Datoteke jezgra

Glava 6

TLS

6.1 Motivacija

6.2 Rukovanje TLS-om tokom izvršavanja programa

6.3 Arhitekturalno specifične zavisnosti

6.4 TLS Modeli pristupa

Glava 7

Implementacija rešenja

Glava 8

Zaključak

Literatura

[1] *ELF Format*. on-line at: http://www.cs.cmu.edu/afs/cs/academic/class/15213-s00/doc/elf.pdf. 1992.

[2] Free Software Foundation. *DWARF Format*. on-line at: http://dwarfstd.org/. 1992.

[3] Linux Foundation. *ptrace*. on-line at: http://man7.org/linux/man-pages/man2/ptrace.2.html.

[4] *GNU GDB*. on-line at: https://www.gnu.org/software/gdb/.

[5] Reid Kleckner. *CodeView*. on-line at: https://llvm.org/devmtg/2016-11/Slides/Kleckner-CodeViewInLLVM.pdf.

Biografija autora