

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, Nedodija

dr Laza LAZIĆ, docent
Univerzitet u Beogradu, Matematički fakultet

Datum odbrane: _____

Mami, tati i dedi

Sadržaj

1	Uvod	1
2	Kako rade debageri?	2
2.1	Linux debageri	2
2.2	Windows debageri	8
3	GNU GDB alat	9
4	DWARF format	10
5	Datoteke jezgra	11
6	TLS	12
6.1	Motivacija	12
6.2	Rukovanje TLS-om tokom izvršavanja programa	12
6.3	Arhitekturno specifične zavisnosti	12
6.4	TLS Modeli pristupa	12
7	Implementacija rešenja	13
8	Zaključak	14
	Literatura	15

Glava 1

Uvod

Glava 2

Kako rade debageri?

Debager (eng. *debugger*) je softverski alat koji koriste programeri za testiranje, analizu i otklanjanje grešaka u programima. Sam proces korišćenja takvih alata nazivamo debugovanjem (eng. *debugging*). Debageri mogu startovati neki proces i debugovati 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. *breakpoints*) itd. Neki debageri imaju mogućnost da izvrše neke posebne izraze ili funkcije adresnog prostora programa koji se debuguje, ili čak menjati strukturu programa prateći propratne efekte.

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

2.1 Linux debageri

Pre svega, definišimo neke od osnovnih pojmova Linux programiranja koje ćemo često pominjati u radu. Osvrnimo se prvo na ELF (eng. *Executable and Linkable Format*) fajl format izvršnih fajlova, deljenih biblioteka, objektnih fajlova i datoteka jezgara [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.

```

ELF Header:
  Magic:   7f 45 4c 46 02 01 01 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    0 0 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 ELF fajl formata, jako bitan fajl format je DWARF, koji predstavljala format zapisa debug informacija koje debageri koriste kada analiziraju programe. DWARF [2] je od posebne važnosti za rad te će isti biti opisan detaljnije u nastavku.

GNU Linux operativni sistem pruža sistemski poziv `ptrace` [3] koji zapravo debagerima omogućava rad. Ovaj sistemski poziv omogućava jednom procesu kontrolu nad izvršavanjem nekog drugog procesa i menjanje memroijske i registarske istog.

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

Prvi argument sistemskog poziva predstavlja zapravo informaciju kojom operativnom sistemu jedan proces, ne nužno debager, ukazuje na nameru preuzimanja kontrole drugog procesa. Neki od njih su:

1. `PTRACE_TRACEME` - omogućava praćenje (eng. *tracing*)

2. `PTRACE_PEEKDATA` - čitanje memorije
3. `PTRACE_POKEDATA` - pisanje memorije
4. `PTRACE_GETREGS` - čitanje registara
5. `PTRACE_SETREGS` - pisanje registara

Drugi argument sistemskog poziva `pid` ukazuje na identifikacioni broj ciljanog procesa, dok treći i četvrti argument se po potrebi koriste u zavisnosti od namere korišćenja *ptrace* sistemskog poziva 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 sistemsko 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. *breakpoints*): 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 *ptrace* sistemskog poziva. Opišimo ceo postupak na jednostavnom primeru. Program je preveden za Intel x86-64 procesorsku arhitekturu i asemblerski kod `main` funkcije primera izgleda kao na primeru 2.2.

Primeru 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
```

Instrukcija `int3` je posebna instrukcija Intel x86-64 procesorske arhitekture, koja izazva softverski prekid. Kada registar programski brojač (eng. *CPU register pc*)


```

0000000004006e1 <main>:
4006e1: 55                push    %rbp
4006e2: 48 89 e5          mov     %rsp,%rbp
4006e5: 48 83 ec 40       sub     $0x40,%rsp
4006e9: 64 48 8b 04 25 28 00 mov     %fs:0x28,%rax
4006f0: 00 00
4006f2: 48 89 45 f8       mov     %rax,-0x8(%rbp)
4006f6: 31 c0             xor     %eax,%eax
4006f8: c7 45 cc 00 00 00 00 movl    $0x0,-0x34(%rbp)
4006ff: eb 31             jmp     400732 <main+0x51>
400701: 8b 55 cc          mov     -0x34(%rbp),%edx
400704: 48 8d 45 d0       lea     -0x30(%rbp),%rax
400708: 48 63 d2          movslq  %edx,%rdx
40070b: 48 c1 e2 03       shl     $0x3,%rdx
40070f: 48 8d 3c 10       lea     (%rax,%rdx,1),%rdi
400713: 48 8d 45 cc       lea     -0x34(%rbp),%rax
400717: 48 89 c1          mov     %rax,%rcx
40071a: ba b6 06 40 00    mov     $0x4006b6,%edx
40071f: be 00 00 00 00    mov     $0x0,%esi
400724: e8 47 fe ff ff    callq   400570 <pthread_create@plt>
400729: 8b 45 cc          mov     -0x34(%rbp),%eax
40072c: 83 c0 01          add     $0x1,%eax
40072f: 89 45 cc          mov     %eax,-0x34(%rbp)
400732: 8b 45 cc          mov     -0x34(%rbp),%eax
400735: 83 f8 04          cmp     $0x4,%eax
400738: 7e c7             jle     400701 <main+0x20>
40073a: bf 05 00 00 00    mov     $0x5,%edi
40073f: e8 5c fe ff ff    callq   4005a0 <sleep@plt>
400744: e8 37 fe ff ff    callq   400580 <abort@plt>
400749: 0f 1f 80 00 00 00 00 nopl    0x0(%rax)

```

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

stigne do `int3` instrukcije izvršavanje se zaustavlja 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 Intel x86-64 procesorsku arhitekturu, 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 debug simbole generišemo uz pomoć `-g` opcije kompajlera, ali o istim će biti opširnije u posebnoj sekciji.

Hardverske tačke prekida su direktno povezane sa hardverom u vidu specijalnih registara. Postavlja se na određenu adresu i hardverski *watchpoint* montri na zadatu adresu i može signalizirati razne promene na istoj, npr. čitanje, pisanje ili izvrša-

```

<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.

vanje, što im daje prednost u odnosu na softverske tačke prekida. Mane u odnosu na softverske tačke prekida svakako jesu performanse, gde su neuporedivo sporije, i takođe neophodna hardverska podrška za korišćenje istih.

Koraćanje

Pod procesom koraćanja (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 debuguje [4].

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

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

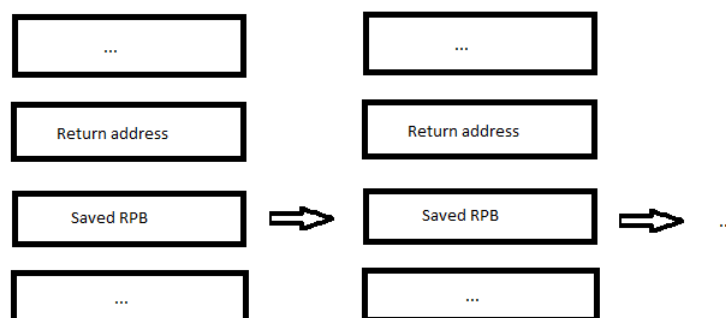
Operativni sistem će dati signal kada je korak izvršen.

Pored instrukcijskog koraćanja pomenućemo još jednu vrstu zakoraćavanja 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. ARM platforma, koja nema hardversku podršku za instrukcijsko koraćanje i za njih se koraćanje implementira na drugačiji način, uz pomoć emulacije instrukcija, ali u ovom radu neće biti opširnije o tome.

Izlistavanje pozvanih funkcija

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



Slika 2.4: Primer redanja stek okvira na x86-64 platformi.

Na slici 2.4 navedeni su stek okviri za dva funkcijska poziva. Pre povratne vrednosti funkcije obično se ređaju 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 debug 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 vrednost promenljive nalazi. Ukoliko ovaj atribut nije naveden debager će vrednost takve promenljive smatrati kao optimizovanu prijavljujući informaciju o tome [4].

2.2 Windows debageri

Windows debageri i programski prevodioci ne prate DWARF standard prilikom baratanja sa debug informacijama namenjene za taj operativni sistem, već isti konsultuju *CodeView* Majkrosoft standard, više informacija možete pronaći na [5].

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 Arhitekturno 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