Alternative alatu Valgrind

Seminarski rad u okviru kursa
Verifikacija softvera
Matematički fakultet

Miloš Miković, 1050/2020
milos.mikovicpos@gmail.com

23. januar 2021.

Sažetak

U ovom radu biće opisan sistem za dinamičku analizu programa Valgrind, njegove karakteristike i alati, a zatim će biti dat pregled nekoliko alternativa ovom alatu. Kroz proces opisivanja alternativnih alata data je uporedna analiza sa Valgrind alatom, a poslednje poglavlje sumira izvedene zaključke.

Sadržaj

1 Uvod

Analiza koda je veoma važna stavka u razvoju softvera koja predstavlja proces dobijanja informacija o programu na osnovu njegovog izvornog koda.[11] Deli se na:

- Statičku
- Dinamičku

Statička analiza predstavlja skup tehnika i metoda za otkrivanje mana izvornog koda bez potrebe da se program izvriši, dok dinamička analiza obuhvata tehnike kojima se prikupljaju podaci o programu tokom njegovog izvršavanja. Neke od najpoznatijih tehnika dinamičke analize koda su:

- Debagovanje
- Profajliranje
- Testiranje

Profajliranje predstavlja vid dinamičke analize programa čiji je cilj da omogući pronalaženje kritičnih tačaka u programu koje smanjuju njegove performanse ili remete njegov rad. Alati koji sprovode ovaj vid dinamičke analize zovu se **profajleri**. Profajleri omogućavaju detaljan uvid o načinu na koji se program izvršava. Neke od mogućnosti koje profajleri pružaju su: uvid u memoriju koju program koristi, praćenje memorije alocirane na hipu (eng. *heap*), praćenje keš memorije, uvid u broj pozivanja neke funkcije kao i vreme i resurse koje ona troši, praćenje ponašanja višenitnih programa u borbi za deljene resurse i slično. U nastavku ovog teksta biće ukratko opisan alat za profajliranje Valgrind, a zatim će biti dat pregled nekih alternativa ovom alatu i njihova uporedna analiza.

2 Valgrind

Valgrind predstavlja jedan od najpopularnijih alata za dinamičku analizu programa na Linux operativnom sistemu koji je javno objavljen februara 2002. godine. Osnovu Valgrind alata čini **jezgro Valgrind-a** na koje se dodaje **alat Valgrind-a**. Jezgro omogućava izvršavanje klijentskog programa a pored toga formira izveštaj koji nastaje kao produkt analize samog programa. Originalni mašinski kod programa se najpre prevodi u međureprezentaciju (eng. *intermediate representation*), a zatim se kod deli u sekvence dužine najviše šezdeset mašinskih instrukcija koje se nazivaju osnovni blokovi. Dalji proces prevođenja i analize mašinskog koda zove se translacija i sačinjena je od osam faza. Sve faze translacije sem instrumentalizacije vrši jezgro Valgrind-a i zato ono troši najviše vremena i resursa potrebnih za dinamičku analizu. Instrumentalizaciju vrši alat Valgrind-a dodajuću u medjureprezentacioni kod dodatne operacije. Ceo proces translacije omogućava korisnicima bogat spektar mogućnosti za analizu i pronalaženje grešaka u programskom kodu, a ceo ovaj proces zavisi od alata koji se koristi.[3] Alati valgrinda:[9]

- Memcheck - prati svako pisanje i čitanje memorije kao i pozive malloc/new/free/delete, primarno se koristi za C i C++ programe
- Cachegrind - simulacija I1, D1 i L2 keš memorije i rada sa njom
- Callgrind - predstavlja dodatak Cachegrind alata i dodatno pruža graf poziva funkcija sa informacijama kao što su vreme izvršavanja ili broj poziva neke funkcije

- Massif - predstavlja hip profajler i koristi se za rad sa dinamički alociranom memorijom
- Helgrind - debager za višenitne programe koji može da detektuje trku za resursima (eng. *race condition*)
- DRD - detektuje greške u višenitnim programima, primarno se koristi za C i C++ programe u radu sa POSIX nitima
- DHAT - prati hip alokacije i najčešće se koristi za vizuelizaciju rezultata
- Lackey, Nulgrind - koriste se za testiranje i demonstrativne svrhe

Valgrind može da se koristi za analizu programa napisanih na mnogim programskim jezicima kao što su Java, Perl, Python, Fortran, Ada. Fleksibilnost u pogledu analize programa napisanih na različitim programskim jezicima moguća je jer Valgrind radi analizu mašinskog koda programa a ne izvornog koda. Ipak, pretežno se koristi za analizu programa napisanih u C i C++ jeziku.[1]

3 Alternative Valgrind alata

Valgrind je alat koji je prvobitno razvijen za Linux operativni sistem i x86 arhitekturu. Danas se pretežno koristi na uniksoidnim operativnim sistemima (eng. *Unix-like*). U nastavku teksta biće date neke od alternativa Valgrind alata za ostale operativne sisteme.

3.1 Deleaker

Deleaker predstavlja moćan alat za profajliranje memorije razvijen za Windows operativni sistem koji podržava 32-bitnu i 64-bitnu arhitekturu. Pretežno predstavlja alternativu memcheck alatu Valgrind-a ali pruža i dodatne mogućnosti. Deleaker se koristi za profajliranje programa napisanih u C++ i C# programskim jezicima a takodje podržava i .NET platformu i Delphi. Deleaker se može zasebno koristiti kao konzolni alat i tada se rezultat analize dobija u XML formatu. Najčešće dolazi kao dodatak za neko integrisano razvojno okruženje (eng. *Integrated development environment, IDE*). S obzirom da se koristi na Windows operativnom sistemu najčešće je to Visual Studio ali se može koristiti i u QtCreator, C++ Builder i RAD Studio razvojnim okruženjima. U odnosu na Valgrind kod koga je usporenje izvršavanja programa veliko jer umeće dodatan kod, Deleaker ne menja kod programa i time pretežno ne povećava vreme izvršavanja programa. Da bi pronašao curenja memorije i slične greške dovoljno je programski kod prevesti u debag (eng. *debug*) režimu i nije potrebna ponovna izgradnja programa (eng. *rebuild*).[8] U nastavku teksta biće ukratko opisan način korišćenja Deleaker alata u Visual Studio razvojnom okruženju.

3.1.1 Deleaker u Visual Studio razvojnom okruženju

Deleaker je jako dobro integrisan u Visual Studio s obzirom da predstavlja dodatak (eng. *extension*) ovog razvojnog okruženja. S obzirom na to, njegovo korišćenje je jako jednostavno jer Visual Studio pruža udobno i funkcionalno okruženje za rad. Deleaker je povezan sa svakim procesom koji se debaguje, jer praktično ne usporava izvršavanje programa,

mada se ovo svojstvo može ručno ugasiti. U svakom trenutku debagovanja programa mogu se videti alokacije memorije koja nije oslobođena i može prouzrokovati curenje memorije (eng. *memory leak*). Ovi snimci (eng. *snapshot*) koje Deleaker beleži Visual Studio interpretira tabelarno na jako čitljiv način. Neke od informacija koje korisnik dobija su: tip curenja memorije, izvorni fajl gde je greška nastala, veličina memorije, adresa memorije i tako dalje. U svakom trenutku odabirom konkretne greške iz ove tabele Visual Studio prikazuje korisniku mesto u izvornom kodu gde je ona nastala ili gde je memorija alocirana. Nekada je posebno zanimljiva vrednost broj pogodaka (eng. *hit count*) koja ako je velika verovatno ukazuje na to da u programu postoji curenje memorije izazvano alokacijama memorije u petlji. Snimci koje Deleaker beleži mogu da se sačuvaju i kasnije porede i to je njihova osnovna namena. Deleaker i Visual Studio se mogu koristiti i za debagovanje i profajliranje .NET programa.[10]

3.2 DynamoRIO

DynamoRIO predstavlja sistem za dinamičku analizu programa i pored toga omogućava transformaciju programskog koda za vreme izvršavanja programa. Ovaj sistem korisnicima pruža alate za analizu programa, profajliranje, optimizaciju, translaciju i druge svrhe. DynamoRIO predstavlja virtualnu mašinu koja pravi kopiju originalnog mašinskog koda programa koji se izvšava i zatim u fazi instrumentalizacije svaki alat vrši dodatne akcije kako bi transformisao napravljenu kopiju. S obzirom da se čitav proces izvršava nad kopijom izvršivog programa, nema promena u originalnom programu i nisu potrebne dodatne pripreme originalnog programa. Iz razloga što se analiza programa vrši nad mašinskim kodom, DynamoRIO može da analizira program napisan praktično na bilo kom programskom jeziku. Jako je sličan Valgrind alatu u načinu funkcionisanja i dobar deo alata koje pruža su alternative Valgrind alatima. Vremensko usporenje koje DynamoRIO nameće je oko 11% a čitav proces se odigrava u fazi izvršavanja programa. DynamoRIO je dostupan za Windows, Linux i Android operativne sisteme dok za Mac operativni sistem nema potpunu podršku. Neki od alata ovog sistema su Dr. Memory i Dr. Cachesim koji su detaljnije opisani u narednim potpoglavljima. Pored ovih DynamoRIO nudi još dosta alata a neki od njih su: [6]

- Dr. Cpusim - koristi se za proveru i testiranje programa na legacy procesorima, tj. da li aplikacija izvršava neku instrukciju koja nije podržana na nekom starijem modelu procesora ili ne podržava neku instukciju u legacy modu nekog procesora
- Dr. Strace - prati sve sistemse pozive koje aplikacija vrši, zajedno sa njihovim argumentima i prikazuje korisniku redosled poziva. Ovaj alat podržan je samo na Windows operativnom sistemu
- Dr. Cov - prikupla informacije o pokrivenosti koda
- Dr. Fuzz - dodatak DynamoRIO sistema namenjen fuzz testiranju i predstavlja deo Dr. Memory alata

3.2.1 Dr. Memory

Može se reći da Dr. Memory predstavlja najpoluraniji alat u sklopu DynamoRIO sistema. Koristi se za profajliranje memorije i dostupan je na Windows, Linux, Android i Mac operativnom sistemu. Može se reći da kombinuje funkcionalnosti Memcheck i Massif alata Valgrind-a. Neke od

grešaka koje može da detektuje su: pristupanje neinicijalizovanoj memoriji, pristupanje dinamički alociranoj memoriji van njenih granica (eng. *heap underflow and overflow*), pristupanje već oslobođenoj memoriji, višestruka oslobađanja iste memorije, curenje memorije. Na Windows operativnom sistemu može i da popravi curenja memorije, a takodje može i da detektuje greške u korišćenju Windows GDI interfejsa (*The Microsoft Windows graphics device interface*). Dolazi kao spoljašnji alat (eng. *External Tool*) Visual Studio razvojnog okruženja i podržan je na svim dostupnim verzijama ovog okruženja.[5]

3.2.2 Dr. Cachesim

Dr. Cachesim predstavlja alat koji se sastoji od dve komponente tragač (eng. *tracer*) i analizer (eng. *analyzer*). Tragač je zadužen za prikupljanje informacija o pristupima memoriji za svaku nit i za svaki proces aplikacije. Analizer koristi informacije koje je tragač prikupio i na osnovu njih pravi analizu programa. Ta analiza se primarno svodi na simulaciju keš memorije procesora i rada sa njom, ali ovaj alat može biti proširen za specifične potrebe korisnika. Shodno tome može se reći da Dr. Cachesim predstavlja vid alternative Cachegrind alata Valgrinda ako se koristi njegova uloga u simulaciju keš memorije. U kodu 1 ispod može se videti rezultat izvršavanja alata Dr. Cachesim prilikom simulacije keš memorije procesora. [4]

```
$ bin64/drrun -t drcachesim -- ~/test/pi_estimator
Estimation of pi is 3.142425985001098
---- <application exited with code 0> ----
Cache simulation results:
Core #0 (1 thread(s))
  L1I stats:
    Hits:                          258,433
    Misses:                          1,148
    Miss rate:                        0.44%
  L1D stats:
    Hits:                           93,654
    Misses:                          2,624
    Prefetch hits:                     458
    Prefetch misses:                 2,166
    Miss rate:                        2.73%
Core #1 (1 thread(s))
  L1I stats:
    Hits:                            8,895
    Misses:                             99
    Miss rate:                        1.10%
  L1D stats:
    Hits:                            3,448
    Misses:                            156
    Prefetch hits:                      26
    Prefetch misses:                   130
    Miss rate:                        4.33%
Core #2 (1 thread(s))
  L1I stats:
    Hits:                            4,150
    Misses:                            101
    Miss rate:                        2.38%
  L1D stats:
    Hits:                            1,578
    Misses:                            130
    Prefetch hits:                      25
    Prefetch misses:                   105
    Miss rate:                        7.61%
Core #3 (0 thread(s))
LL stats:
    Hits:                            1,414
    Misses:                          2,844
    Prefetch hits:                     824
```

```
1042    Prefetch misses:                1,577
        Local miss rate:               66.79%
1044    Child hits:                   370,667
        Total miss rate:                0.76%
```

Kod 1: Primer rezultata izvršavanja Dr. Cachesim alata [4]

3.3 Clang sanitizers

Clang je prevodilac otvorenog koda namenjen za C, C++, Objective-C i Objective-C++ programske jezike. Razvijen je od strane firmi Apple, Google, Microsoft, Sony, ARM, AMD i Intel i predstavlja alternativu GCC prevodiocu. Pored uloge prevodioca, Clang pruža i alate[1] namenjene dinamičkoj analizi programskog koda i kao takav predstavlja vid alternative Valgrind alatima. S obzirom da je Clang podržan i na Windows, Linux i Mac operativnom sistemu, ali i na drugim operativnim sistemima, to čini sanitizer alate dosta fleksibilnim i portabilnim. Ovi alati su inicijalno bili deo LLVM/Clang projekta ali su kasnije dodati i u GCC prevodilac. Iako ovi alati nisu moćni kao Valgrind-ovi alati, u odnosu na njih dovode do jako malog usporenja izvršavanja programa i nekada su bolja i brža alternativa. Razlog tome je što se ovi alati linkuju u fazi izgradnje programa, a Valgrindovi alati rade u fazi izvršavanja programa. Za dobijanje boljih rezultata ne preporučuje se korišćenje visokog nivoa optimizacije koda i preporučen nivo optimizacije je -O1, time takođe izbegavamo umetanja (eng. *inline*) koja prevodilac vrši, a koja mogu loše da utiču na proces analize. Neki od Clang alata za dinamičku analizu:

- **Address Sanitizer** - koristi se za rad sa memorijom i sličan je memcheck alatu Valgrind-a. Može da detektuje pristupanje memoriji van njenih granica, duplo ili nepravilno oslobadjanje memorije, korišćenje već oslobodjene memorije i druge vidove grešaka sa memorijom. U proseku nameće usporenje od oko dva puta. Prilikom prevođenja programa potrebno je navesti opciju -fsanitize=address ali postoje i dodatne opcije koje se mogu navesti u zavisnosti od potreba. Primera radi ako se prevodi C++ program, za bolje rezultate dobro je navesti i komandu -fno-omit-frame-pointer. Address Sanitizer se koristi na svim platformama koje podržava od LLVM 3.1 verzije.
- **Thread Sanitizer** - predstavlja alat sličan Helgrind alatu Valgrind-a. Thread Sanitizer može da detektuje trku za podacima u radu sa višenitnim programima i nameće vremensko usporenje od pet do petnaest puta dok je memorijsko usporenje od pet do deset puta. Prilikom prevođenja potrebno je navesti opciju -fsanitize=thread uz opciju -g koja označava da se program prevodi u debag režimu, radi dobijanja dodatnih informacija. Thread Sanitizer je trenutno u beta verziji i podržava rad sa nitima u C++11 verziji korišćenjem biblioteke llvm libc++. Trenutno podržava 64-bitnu arhitekturu dok rad na 32-bitnoj verziji nije u planu.
- **Undefined Behaviour Sanitizer** - se koristi za otkrivanje nedefinisanih ponašanja u programu kao što su: korišćenje null pokaživača, prekoračenje vrednosti označenog celog broja (eng. *Signed integer*

[1] Zapravo sanitezer nije alat Clang prevodioca u pravom smislu te reči, ipak u tekstu je zbog jednostavnosti korišćen naziv alat kako bi se izbeglo ponavljanje reči sanitizer svuda gde je to potrebno i radi lakšeg poređenja sa Valgrind-ovim alatima

overflow), konverzije koje mogu da dovedu do prekoračenja i drugih. Prilikom prevođenja potrebno je navesti opciju -fsanitize=*, gde * predstavlja nedefinisano ponašanje koje želimo da Thread Sanitizer prati. Neke od opcija su signed-integer-overflow, null, alignment i one mogu biti zasebno navedene ali se može i kombinovati više njih. Undefined Behaviour Sanitizer se koristi na svim platformama koje podržava od LLVM 3.3 verzije.

Pored navedenih, postoje još i Memory Sanitizer, Data Flow Sanitizer, LeakSanitizer, SanitizerCoverage, SanitizerStats. Iako ovi alati predstavljaju deo Clang i GCC prevodioca i ne predstavljaju deo nekog zasebnog alata za dinamičku analizu programa, često se koriste u praksi za analizu C i C++ programa. Njihova velika prednost je to što omogućavaju dosta bržu analizu programa od Valgrinda zbog već pomenutih razloga, ali su ograničeni na analizu programa koje može da prevede Clang prevodilac. [2]

4 Poređenje navedenih alata sa Valgrind alatom

Postoje razne karakteristike po kojima možemo porediti navedene alata, a neke od njih su vremensko usporenje koje nameću izvršavanju programa, lakoća rada sa njima, operativni sistemi na kojima su podržani, kvalitet analize koju pružaju i druge. U tabeli 1 može se videti dostupnost navedenih alata na različitim operativnim sistemima. Valgrind i DynamoRIO predstavljaju sisteme koji imaju jako sličnu arhitekturu i način korišćenja, rade nad mašinskim kodom programa i samim tim su dosta fleksibilni u pogledu analize programa napisanih na različitim programskim jezicima. Sa druge strane njihov faktor vremenskog usporenja je visok i trebaju se pažljivo koristiti onda kada je njihova upotreba zaista neophodna. Oba sistema pružaju jako kvalitetnu i detaljnu analizu programa i daju širok spektar alata. Valgrind je nešto popularniji na Linux operativnom sistemu dok se DynamoRIO više koristi na Windows i Android operativnom sistemu. Dr. Memory se može koristiti na Mac operativnom sistemu, iako sistem DynamoRIO ne pruža potpunu podršku za MacOS i mnogi alati se ne mogu koristiti. Sa druge strane ako opsežna analiza programa nije potrebna, mogu se koristiti i Clang Sanitizer alati. Dosta su efikasni i brzi, daju prilično kvalitetnu analizu i ne nameću veliko vremensko usporenje. Kada je upitanju profajliranje C++ i C# programa na Windows operativnom sistemu Deleaker predstavlja jako dobru opciju jer skoro da ne nameće vremensko usporenje i pritom dolazi kao dodatak Visual Studio razvojnog okruženja. Pored toga, kao što je već rečeno, Vi-

Tabela 1: Pregled dostupnosti alata na različitim operativnim sistemima

	Linux	Windows	MacOS	Android
Valgrind	da	ne	da	da
Dr. Memory	da	da	da	da
Dr. Cachesim	da	da	ne	da
Deleaker	ne	da	ne	ne
Clang sanitizers	da	da	da	da

sual Studio podatke analize interpretira na jako čitljiv način i stoga nije potrebno koristiti spoljašnje alate za interpretaciju rezultata.

Kada je u pitanju profajliranje C# programa vredno je napomenuti i dotMemory alat koji je razvila firma JetBrains. Jako dugo bio je dostupan samo za Windows operativni sistem, ali juna 2020. godine dolazi i kao komandni alat na Linux i Mac operativnom sistemu. Podržava praktično sve novije verzije .NET CORE frejmvorka (eng. *framework*) i pruža jako kvalitetnu dinamičku analizu memorije.

5 Zaključak

Dinamička analiza, iako jako skup proces, neophodna je kada softver postane kompleksan i glomazan. Pored traženja grešaka, što je i primarni razlog analize, u velikoj meri se vrši i verfikacija softvera. Valgrind predstavlja jako kvalitetan i sveobuhvatan sistem za dinamičku analizu i kao takav predstavlja jako dobar izbor za korisnike Linux i Mac operativnog sistema. Njegova prednost je što kombinuje veliki broj alata i korisniku pruža fleksibilniju i moćniju analizu u poređenju sa alatima koji primarno vrše analizu memorije, kao što je na primer Deleaker. DynamoRIO sistem jako je sličan Valgrind-u i u pogledu arhitekture i u pogledu na sveobuhvatnost analize koju pruža. S obzirom da Valgrind nije dostupan na Windows operativnom sistemu, DynamoRIO je jako dobra alternativa za korisnike Windows sistema. Na Mac operativnom sistemu nisu dostupni svi alati DynamoRio sistema i tada je na korisniku da odluči koje alate će koristiti i kombinovati sa Valgrind-ovim alatima. Sa druge strane Clang sanitizer alati su dosta jednostavni i laki za korišćenje i kada je u pitanju analiza C i C++ programa predstavljaju jako dobar izbor na svim prethodno navedenim operativnim sistemima. Trenutno postoji najviše alata za analizu memorije, i dosta drugih koji vrše analizu ostalih segmenata programa. S obzirom na to, izbor nekog alata zavisi od operativnog sistema, programskog jezika koji se koristi, kvaliteta analize, udobnosti u radu i često zavisi od lične odluke korisnika jer ne postoji najbolji alat među navedenim. Tek njihovim integracionim korišćenjem, zajedno sa sistemima kao što su Valgrind i DinamoRyo, korisnik dobija kvalitetnu i sveobuhvatnu analizu programa.

Literatura

[1] About valgrind. https://www.valgrind.org/info/about.html.

[2] Clang v12 documentation. https://clang.llvm.org/docs/index.html.

[3] The design and implementation of valgrind. https://www.valgrind.org/docs/manual/mc-tech-docs.html#mc-tech-docs.intro.

[4] Dr. cachesim documentation. http://dynamorio.org/dynamorio_docs/page_drcachesim.html.

[5] Dr. memory documentation. https://dynamorio.org/drmemory_docs/index.html.

[6] Dynamorio. https://dynamorio.org/.

[7] Supported platforms. https://www.valgrind.org/info/platforms.html.

[8] Valgrind for windows. https://www.deleaker.com/blog/2020/01/04/valgrind-for-windows/.

[9] Valgrind's tool suite. https://www.valgrind.org/info/tools.html.

[10] Visual studio memory leak detection with the help of deleaker. https://www.deleaker.com/docs/deleaker/tutorial.html.

[11] David Binkley. Source Code Analysis: A Road Map. *Future of Software Engineering(FOSE '07)*, pages 2–5, 2007.

[12] Derek Bruening and Qin Zhao. Practical memory checking with dr. memory. https://www.researchgate.net/publication/224236404_Practical_memory_checking_with_Dr_Memory.

A Dodatak

Platforme koje Valgrind podržava su: x86/Linux, AMD64/Linux, PPC32/Linux, PPC64/Linux, PPC64LE/Linux, S390X/Linux, ARM/Linux, ARM64/Linux, MIPS32/Linux, MIPS64/Linux, X86/Solaris, AMD64/Solaris, X86/illumos, AMD64/illumos, X86/Darwin (10.10, 10.11), AMD64/Darwin (10.10, 10.11), ARM/Android, ARM64/Android, MIPS32/Android, X86/Android.[7]

Na testu koji je izvršen na 32-bitnom Linux sistemu korišćenjem benchmark standarda SPEC CPU2006 Dr. Memory se pokazao dvostruko bržim od Memcheck alata Valgrind-a, a u samostalnom radu i do četiri puta bržim.[12]