Razhroščevalnik

Seminarska naloga pri predmetu Sistemska programska oprema

Jure Jesenšek, januar 2015

1. Uvod

V seminarski nalogi bom predstavil in opisal okvirno delovanje razhroščevalnikov. Razložil bom različne strojne in programske vire, ki omogočajo razhroščevanje. Naštel pa bom tudi nekaj najbolj popularnih in največkrat uporabljenih razhroščevalnikov, ter njihovih »front-end-ov«.

Na koncu bom tudi naredil nekaj kratkih demonstracij uporabe razhroščevalnika GDB iz kompleta programov projekta GNU.

2. Splošno o razhroščevanju in razhroščevalnikih

Razhroščevalnik (debugger) je računalniški program, s katerim testiramo in razhroščujemo druge (ciljne) programe.

2.1. Kratka zgodovina

Uporaba izrazov hrošč (bug) in razhroščevanje (debugging) pri opisovanju iskanja in odpravljanja napak pri računalnikih in računalniških programov sega v 40. leta 20. stoletja. V času velikih elektro-mehaničnih (relejskih) računalnikov se je pogosto zgodilo, da je izvajanje računalnika onemogočil dejanski hrošč, ki se je zavlekel med releje in žice, le-tega pa je bilo seveda treba odstraniti, in s tem "razhroščiti" sistem.

2.2. Postopek razhroščevanja

Najpogosteje se razhroščevanja lotimo tako, da najprej poizkusimo ponoviti stanje, ki je privedlo do napake, kar seveda ni vedno trivialno. Po temu, ko smo ponovili napako v programu, pa nam v nadaljevanju pogosto pomaga, če zmanjšamo obseg oziroma si poenostavimo vhodne podatke. Ko je programer zadovoljen z obsegom tesnih podatkov, pa le-ta uporabi razhroščevalnik, s katerim pregleda stanje programa med samim izvajanjem.

2.3. Pogoste tehnike

- Print (printf) debugging – izpisovanje in pregledovanje stanj programa med izvajanjem
- Post-mortem – razhroščevanje programa, ko je le-ta že prenehal z delovanjem zaradi hrošča. Pogosto se pregleda izpise spomina (memory/core dump)
- »Wolf fence« - razpolavljanje programa dokler ne odkrijemo kje se napaka nahaja
- Delta debugging – uporaba unit testov za avtomatsko odkrivanje in izločanje napak

3. Način delovanja

Razhroščevalniki si pri delovanju pomagajo s pastmi (trap), tj., s prekinitvami, ki se prožijo ob dogodkih, ki potrebujejo posebno pozornost, vir le-teh pa so lahko hrošči (deljenje z nič, nedovoljeni pomnilniški dostop) ali pa ko program naleti na breakpoint – v teh primerih CPE vrne izjemo (exception) oziroma napako (fault). Pri pasti po navadi CPE preklopi v supervisor (kernel) način in s tem z izvajanjem na CPE nadaljuje OS, ki pa nato vrne CPE prvotnemu procesu. Pri razhroščevanju pa pod past najpogosteje štejemo prekinitev, ki poskrbi za začetek predajanja nadzora CPE razhroščevalniku.

Glede na to, kaj razhroščevalnik pokaže, ko se v ciljnem programu sproži past, ločimo dve vrsti razhroščevalnikov:

- source-level debugger ali symbolic debugger - najpogostejši tip razhroščevalnika v integriranih razvojnih okoljih (IDE). Pri ustavitvi pokaže vrstico ustavitve v prvotni izvorni kodi.
- low-level debugger ali machine-language debugger - prikaže vrstico ustavitve v disassembly-ju

4. Strojna podpora

Arhitektura CPE omogoči in olajša razhroščevanje z raznimi tehnologijami, vgrajenimi v samo CPE, kot so:

- podpora za izvajanje enega ukaza (single-stepping), npr. z trap flag, ki vklopi proženje prekinitve po vsakem ukazu
- strojna podpora za breakpoint-e (primerjalniki naslovov in podatkov)
- nabor ukazov, ki omogočajo virtualizacijo za lažje razhroščevanje na isti CPE kot je ciljni program.
- In-system programming - reprogramiranje mikro-krmilnikov, vgrajenih sistemov in podobnih preprostih sistemov, brez spreminjanja strojne opreme (zamenjave čipov)
- JTAG podpora dostopa do strojne opreme (za vgrajene sisteme, FPGA čipe...)

5. Prekinitvene točke (Breakpoint)

Breakpoint (prekinitvena točka) je lokacija v programu, na kateri želimo da CPE ustavi izvajanje programa, z namenom razhroščevanja. V grobem jih delimo na dve vrsti prekinitvenih točk. Prekinitveni točki, ki se izvede vedno, ko CPE pride do ukaza kjer smo postavili prekinitveno točko pravimo instruction breakpoint. V nasprotju pa conditional breakpoint ustavi program le pri pogoju, ki mu ga

nastavimo, npr. branje, pisanje ali spreminjanje neke lokacije v pomnilniku. Alternativna imena so tudi data breakpoint in watchpoint.

Prekinitvene točke pa lahko delimo tudi na Hardware (strojne) in Software (programske) breakpoint-e. Kot je že razvidno iz imena, so strojni implementirani na nivoju strojne opreme, medtem ko pa za delovanje programske skrbi programska oprema.

5.1. Strojne prekinitvene točke

Strojne prekinitvene točke omogoča sam CPE, v katerem se nahajajo namenski registri (DR – debug register). Ker za delovanje strojnih prekinitvenih točk potrebujemo namenske strojne vire, je število strojnih prekinitvenih točk pogosto omejeno (x86 podpira le 4 strojne prekinitvene točke).

Debug registre v arhitekturi x86 sestavlja šest registrov. V registre DR0 do DR3 razhroščevalnik piše linearni pomnilniški naslov prekinitvene točke. DR6 in DR7 pa se uporabljata za statuse prekinitev in nadzor strojnih prekinitvenih točk.

Strojne prekinitvene točke se najpogosteje uporabljajo kot »pomnilniške« prekinitvene točke - če se pri pisanju in/ali branju iz pomnilnika, naslov v ukazu ujema z naslovom shranjenem v enem izmed debug registrov, se proži prekinitev, ki »zbudi« razhroščevalnik.

Strojne prekinitvene točke pa se lahko uporabljajo pri razhroščevanju vhodno-izhodnih naprav, a je uporabnost tega načina omejena.

5.2. Programske prekinitvene točke

Pri navadni uporabi razhroščevalnika pa največkrat in najpogosteje uporabljamo programske prekinitvene točke. Le-te od razhroščevalnika zahtevajo nekoliko več dela.

Pri postavitvi programske prekinitvene točke v kodi, razhroščevalnik dotični ukaz (opcode) shrani v tabelo prekinitvenih točk, sam opcode v kodi pa zamenja z ukazom, ki bo ob izvedbi sprožil izjemo (exception), torej past, deljenje z nič ali kakšen drug podoben ukaz. Ko CPE javi to izjemo, v izvajanje vskoči razhroščevalnik, ki ustavi izvajanje ciljnega programa. Ob nadaljevanju izvajanja ciljnega programa, razhroščevalnik v kodi obnovi prvoten opcode, izvede en ukaz v programu in zopet v kodo vstavi prvotno past. Pred izvedbo ukaza s prvotnim opcode-om pa mora razhroščevalnik poskrbeti še za »star« ukaz, ki je ostal v cevovodu CPE. Pred izvajanjem mora tako cevovod izprazniti in ponovno izvesti prevzem ukaza.

Pri programskih prekinitvenih točkah lahko pride do težav, saj mora razhroščevalnik pisati v kodo programa. Pri tem se lahko pojavijo težave kadar ne moremo pisati v pomnilnik, kjer je shranjen ciljni program (npr. v ROM-u).

Problematična je lahko tudi velikost ukaza, ki proži izjemo, saj le-ta ne sme biti daljši od najmanjšega ukaza (npr. jump), saj bi lahko v nasprotnem primeru prepisali sosednji ukaz v programu.

Še ena slabost programskih prekinitvenih točk je tudi v tem, da močno upočasnijo izvajanje programa, saj za isto izvajanje potrebuje več virov na procesorju.

6. Razhroščevanje v Javi

Razhroščevanje programov napisanih v Javi in izvajanih v JVM (Java Virtual Machine) je zaradi odsotnosti naslovov in strojne kode drugačen. Pri javi se zato uporablja vmesnik (interface) Java Virtual Machine Tool Interface (JVM TI), ki je implementiran na nivoju navideznega stroja. JVM TI definira storitve, ki jih mora VM omogočati v namen razhroščevanja, med katerimi so zahteve za informacije (stanje spremenljivk), akcije (postavljanje prekinitvene točke) in obvestila (sporočilo, ko program doseže prekinitveno točko).

7. Seznam razhroščevalnikov in front-end-ov

7.1. Razhroščevalniki:

- GDB (GNU)
- LLDB (LLVM)
- MS Visual Studio Debugger
- WDW (OpenWatcom)
- dbx (Berkeley Unix)
- Intel debugger (vsi zgoraj C/C++)
- Valgrind (memory debugging)
- WinDbg (Windows)
- CodeView (MS-DOS)
- sdb (stari UNIX)
- jdb (Java)
- Python debugger
- DBG (PHP)
- ARM DS-5

7.2. GUI front-end:

- DDD (GDB, jdb, python, Perl...)
- Emacs (GDB)
- xxgdb (X-window in dbx)
- Qt Creator (GDB, CDB, LLDB)
- Eclipse
- CodeBlocks

... in seveda mnogi drugi.

8. GDB (GNU Debugger)

GNU Debugger (GDB ali gdb, po imenu same izvršljive datoteke) je bil eden izmed prvih programov, ki ga je Richard Stallman napisal za kolekcijo programov, ki sestavljajo sistem GNU. GDB je tako standardni razhroščevalnik GNU operacijskih sistemov in ga najdemo na praktično vseh GNU/Linux operacijskih sistemih. Začetne verzije programa so temeljile na razhroščevalniku DBX, iz distribucije Berkeley Unix.

GDB podpira programske jezike Ada, C, C++, Objective-C, Pascal, Fortan, Java. Delna podpora obstaja tudi za druge programske jezike. Nekatere izmed arhitektur, ki jih GDB podpira so: ARM, x86 in x86-64, Itanium procesorje, PowerPC, Motorola 68000, itd.

GNU Debugger omogoča sledenje in spreminjanje izvajanja programov. Uporabnik lahko izpisuje in spreminja vrednosti spremenljivk v programu, ter kliče funkcije, ki so neodvisno od stanja v programu. Ponuja tudi možnost remote razhroščevanja, ki se uporablja pri iskanju in odpravljanju hroščev v programih na vgrajenih sistemih. V tem načinu GDB teče na drugem računalniku kot pa ciljni program, komunikacija pa poteka preko TCP/IP. Ta način se tudi uporablja pri razhroščevanju Linux jedra med izvajanjem.

GDB sam po sebi nima grafičnega vmesnika, saj komunikacija z uporabnikom poteka preko ukazne vrstice. Seveda obstaja precej grafičnih »front-end-ov«, kot so na primer: DDD (Data Display Debugger), xxgdb, kdbg, itn. Integrirana razvojna okolja (IDE) pogosto podpirajo razhroščevanje z GDB, med njimi so: CodeBlocks, Geany, Qt Creator, Eclipse, NetBeans kot tudi MS Visual Studio. GNU Emacs podpira način za izvajanje z GDB.

9. Prikaz uporabe razhroščevalnika GDB

Prikazal bom nekaj osnovnih primerov ukazov v programu gdb, ter kako se z razhroščevalnikom priklopimo na že izvajajoč proces.

10. Viri:

https://en.wikipedia.org/wiki/Debugging

https://en.wikipedia.org/wiki/Debugger

http://www.nynaeve.net/?p=80

https://en.wikipedia.org/wiki/Exception_handling

http://processors.wiki.ti.com/index.php/How_Do_Breakpoints_Work

https://en.wikipedia.org/wiki/GNU_Debugger

https://sourceware.org/gdb/wiki/Internals/

https://en.wikipedia.org/wiki/Data_Display_Debugger

http://www.unknownroad.com/rtfm/gdbtut/gdbtoc.html

https://en.wikipedia.org/wiki/List_of_debuggers

https://en.wikipedia.org/wiki/In-system_programming

https://en.wikipedia.org/wiki/Trap_flag

https://en.wikipedia.org/wiki/Delta_Debugging

http://docs.oracle.com/javase/7/docs/technotes/guides/jpda/architecture.html

http://docs.oracle.com/javase/7/docs/platform/jvmti/jvmti.html

10.1. Programi, uporabljeni pri predstavitvi:

- GCC: https://gcc.gnu.org/
- GDB: https://www.gnu.org/software/gdb/
- DDD: https://www.gnu.org/software/ddd/