VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INTELIGENTNÍCH SYSTÉMŮ
DEPARTMENT OF INTELLIGENT SYSTEMS

INSTRUMENTACE C/C++ PROGRAMŮ PŘI PŘEKLADU

INSTRUMENTATION OF C/C++ PROGRAMS DURING COMPILATION

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **KATEŘINA MUŠKOVÁ**
AUTHOR

VEDOUCÍ PRÁCE **Ing. ALEŠ SMRČKA, Ph.D.**
SUPERVISOR

BRNO 2021

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav inteligentních systémů (UITS) Akademický rok 2020/2021

Zadání bakalářské práce

Studentka: **Mušková Kateřina**
Program: Informační technologie
Název: **Instrumentace C/C++ programů při překladu**
Instrumentation of C/C++ Programs during Compilation
Kategorie: Analýza a testování softwaru

Zadání:

1. Nastudujte principy instrumentace programů. Zaměřte se na programy v jazyku C/C++. Seznamte se s infrastrukturou LLVM a s projektem Tforc z platformy Testos.
2. Navrhněte program pro snadnou instrumentaci programů při překladu pomocí nástoje Clang. Zaměřte se na instrumentaci přístupů do paměti a volání funkcí.
3. Implementujte instrumentační program v jazyku C/C++.
4. Ověřte základní funkcionalitu pomocí automatické testovací sady.

Literatura:

- R. Tschüter, J. Ziegenbalg, B. Wesarg, M. Weber, C. Herold, S. Döbel, R. Brendel. An LLVM Instrumentation Plug-in for Score-P. 2017. In Proceedings of LLVM-HPC'17. Denver, CO, USA. doi:10.1145/3148173.3148187
- Dokumentace k překladači LLVM/Clang. Dostupné na URL http://llvm.org/docs/
- A. Vitch, D. Berris, E. Anderson, E. Heintze, N. Wang. XRay: A Function Call Tracing System. 2016-04-05. Dostupné na URL: https://storage.googleapis.com/pub-tools-public-publication-data/pdf/45287.pdf

Pro udělení zápočtu za první semestr je požadováno:

- První dva body zadání

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Smrčka Aleš, Ing., Ph.D.**
Vedoucí ústavu: Hanáček Petr, doc. Dr. Ing.
Datum zadání: 1. listopadu 2020
Datum odevzdání: 12. května 2021
Datum schválení: 11. listopadu 2020

Abstrakt

Tato práce se zabývá návrhem a implementací nástroje TforcTool sloužícího k instrumentaci programů napsaných v jazyce `C++`, a to instrumentaci přístupů do paměti a volání funkcí. Nástroj staví už na existujícím nástroji Tforc poskytující statickou instrumentaci při překladu, jehož funkcionalitu a použitelnost rozšiřuje. Velkou výhodou oproti stávajícím řešením nabízejícím instrumentaci při překladu je možnost použití nástroje bez změny stávajících překladových skriptů (např. Make).

Abstract

This thesis presents design and implementation of the TforcTool offering compile-time instrumentation of memory access and functions. The tool is built on an existing static instrumenting tool Tforc, which was extended in order to provide greater usability and functionality. The advantage of this solution compared to another compile-time tools is that there is no need to change current compile structure of project.

Klíčová slova

instrumentace, C++, testování, instrumentace při překladu, statická instrumentace, LLVM, LLVM IR, Clang, překlad, opt, LLVM zásuvný modul, instrumentace proměnné, instrumentace funkce, nepřímá instrumentace proměnné, formální verifikace

Keywords

instrumentation, C++, testing, compile-time instrumentation, static instrumentation, LLVM, LLVM IR, Clang, compilation, opt, LLVM pass, memory instrumentation, function instrumentation, indirect addresing instrumentation, formal verification

Citace

MUŠKOVÁ, Kateřina. *Instrumentace C/C++ programů při překladu.* Brno, 2021. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Aleš Smrčka, Ph.D.

Instrumentace C/C++ programů při překladu

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracovala samostatně pod vedením pana doktora Aleše Smrčky. Uvedla jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpala.

.....
Kateřina Mušková
10. května 2021

Poděkování

Ráda bych poděkovala panu Ing. Aleši Smrčkovi Ph.D. za vedení mé bakalářské práce a rady, které mi poskytl. Mé díky také patří mé rodině a příteli za podporu po celou dobu studia.

Obsah

1 Úvod **3**

2 Přehledový slovník použitých pojmů **4**

3 Přehled instrumentace programů **6**
3.1 Možnosti použití instrumentace 6
3.2 Základní rozdělení instrumentace 8
3.3 Existující nástroje pro instrumentaci `C/C++` programů 9

4 Použité nástroje **11**
4.1 Technologie LLVM 11
4.2 Tforc 14
4.3 Update-alternatives 21

5 Návrh instrumentačního nástroje TforcTool **23**
5.1 Specifikace požadavků 23
5.2 Přehled použitých nástrojů a jazyků 24
5.3 Struktura programu TforcTool 24

6 Implementace instrumentačního nástroje TforcTool **26**
6.1 Obejití volání originálního překladače 26
6.2 Zpracování vstupních argumentů určeného pro originální překladač 31
6.3 Řízení překladu a instrumentace 31
6.4 Uživatelem definované vstupní soubory 35
6.5 Interní konfigurační soubor 36

7 Evaluace instrumentačního nástroje TforcTool **37**
7.1 Testy 37
7.2 Nalezené chyby 38
7.3 Výkon nástroje 40
7.4 Závislosti 41

8 Závěr **42**

Literatura **43**

A Rozhraní nástroje TforcTool **46**

B Příklad použití **48**

C Seznam jmen nalezených nedeterministických chyb **50**

D Skript sloužící pro výkonnostní test nástroje Tforc **51**

E Obsah přiloženého paměťového média **53**

Kapitola 1

Úvod

V dnešní době je kladen čím dál tím větší důraz na kontrolu kvality implementovaného kódu. Techniky jako vývoj řízený testy (Test Driven Development – TDD), či agilní vývoj, nástroje umožňující ladění kódu, testovací frameworky nebo analyzační nástroje jsou už běžnou součástí procesu tvorby programu. Do této kategorie také spadá nástroj Spectra poskytující programům napsaných v jazyce C formální verifikaci za běhu. To znamená, že kontrola, zda program funguje správně, probíhá přímo na běžícím softwaru na základě specifikovaných formulí.

Kontrola samotná vždy probíhá na specifických bodech, které musí uživatel zatím ručně najít a příslušným způsobem upravit. Do budoucna se však počítá s rozšířením nástroje Spectra o automatickou úpravu těchto bodů, neboli automatickou instrumentaci.

Prvním krokem bylo vytvoření vhodného instrumentačního nástroje, a to nástroje Tforc. Tforc sice umožňuje instrumentaci proměnných a volání funkcí, ale jeho reálné použití je velmi komplikované. A právě tomuto problému se věnuje tato bakalářská práce, tedy zjednodušení použití nástroje tak, aby bylo možné ho v budoucnu propojit s nástrojem Spectra, a v rámci toho poskytnout možnost instrumentace bez nutnosti měnit stávající strukturu překladu. Vytvořený nástroj se jmenuje TforcTool. Jeho vstupem jsou specifikace instrumentace, výstupem pak instrumentovaný program. Posledním krokem k umožnění automatické instrumentace bude vytvoření vazby mezi formálními formulemi a vstupními specifikacemi pro nástroj TforcTool.

V kapitole 2 jsou vypsané základní pojmy použité v rámci bakalářské práce. Na ni navazuje kapitola 3, kde je představena instrumentace včetně možného použití a základního dělení. Závěr kapitoly tvoří několik příkladů existujících instrumentačních nástrojů. Další kapitola 4 se věnuje použitým nástrojům. Jako první je zde popsána technologie LLVM, za ní následuje představení nástroje Tforc a závěr tvoří popis nástroje update-alternatives. Kapitola číslo 5 popisuje návrh nástroje TforcTool spolu se strukturou a požadavky. V návaznosti na ni pokračuje kapitola 6 popisující stěžejní části implementovaného nástroje. Poslední z kapitol obsahuje evaluaci implementovaného nástroje. Nachází se zde popis implementovaných sad testů, výkonnostního testu a závislostí.

Kapitola 2

Přehledový slovník použitých pojmů

Současná kapitola obsahuje přehled základních pojmů použitých v rámci této práce. Definice jsou čerpány z textů k přednáškám Testování a dynamická analýza [27], z knih The Art of Software Testing [25] a Software Instrumentation [20] ze stránky popisující překlad programu [24].

testování	zkoumá software jeho spuštěním za účelem zvýšení jeho kvality; funguje na principu zpracování vstupních dat programem do výstupních dat, která jsou následně porovnána s očekávanými výsledky; slouží k ověření funkčnosti a kvality vyvíjeného programu
system under test	testovaný systém, zkráceně SUT
testovací případ	popis akcí prováděných s cílem ověřit funkcionalitu SUT; skládá ze vstupnıch hodnot, očekávaných výsledků, ale také z hodnot sloužících pro přıpravu programu před testem a vyčištěnı po testu
testovací sada	série testů
pokrytí	míra udávající, jak velká část SUT je už otestována
požadavek na test	specifický element SW, který musı daný test pokrýt nebo splnit
kritérium pokrytí	pravidlo nebo předpis pro systematické generovánı požadavků na test
analýza	získávání komplexních informací o softwaru
analyzátor	nástroj poskytující analýzu. Skládá se ze sondy monitoru a kontroléru.
dynamická analýza	typ analýzy prováděné nad spuštěným softwarem
statická analýza	typ analýzy prováděné bez spuštění softwaru
verifikace	proces kontrolující, zda software vyhovuje specifikacím
validace	proces kontrolující, zda software splňuje zadání nebo účel
sonda	část kódu sledující vybraný artefakt SUT (hodnota proměnné, zatížení CPU, vstupně výstupní operace, . . .)

monitor	jednotka agregující výstupy sond do komplexnějšího popisu stavu SUT
kontrolér	řídící jednotka analyzátoru
instrumentace	proces přidání kódu do stávajícího textu programu, nebo jeho modifikace
statická instrumentace	typ instrumentace probíhající před spuštěním programu
dynamická instrumentace	typ instrumentace probíhající při spuštění programu
instrumentační nástroj	software zajišťující instrumentaci
překlad	převod vyššího programovacího jazyka do nižšího; typicky se skládá z částí preprocesor, překladač, assembler, sestavovací program
preprocesor	součást překladače předzpracovávající symboly, makra a direktivy překladače; angl. preprocessor
překladač	součást překladače převádějící předzpracovaný zdrojový kód do jazyka symbolických instrukcí; angl. compiler
assembler	součást překladače převádějící jazyk symbolických instrukcí do objektového souboru
sestavovací program	součást překladače spojující (linkující) samostatně přeložené moduly, objektové soubory a knihovny; angl. linker
LLVM IR	vnitřní reprezentace kódu (mezikód) používaná v projektu LLVM

Kapitola 3

Přehled instrumentace programů

Tato kapitola poskytuje základní přehled k instrumentaci programu. Nejdříve je vysvětleno, co instrumentace představuje. V další sekci 3.1 jsou uvedeny základní možnosti využití instrumentace za níž následuje sekce 3.2 s klasifikací instrumentace a popisem hlavních principů fungování. Závěrečná sekce 3.3 tvoří krátký přehled současných instrumentačních nástrojů s jejich výhodami a nevýhodami. Hlavním zdrojem celé kapitoly je kniha Instrumentace softwaru [20]. Pokud není uvedeno jinak, je text čerpán právě odtud.

Velmi zjednodušeně můžeme říct, že instrumentace přidává kód navíc do textu programu, nebo ho modifikuje. Tento kód, neboli *sonda*, nám dovoluje sledovat nějaký druh chování programu, a to za účelem například ladění, nebo optimalizací. Dává nám tedy vhled do implementovaného softwaru a přináší dodatečné informace o něm. [28]

Nejjednodušší a nejintuitivnější způsob je manuální instrumentace, tedy zavádění těchto sond ručně do zdrojového kódu. Ta ale není už dlouhou dobu dostačující vzhledem k rostoucí velikosti a komplexitě nejen programů, ale i požadavků na sledování. Jako důsledek vzniklo velké množství analyzačních nástrojů, které jsou schopny provádět automatickou instrumentaci.

Jedna z charakteristik instrumentace je její vedlejší vliv na program, skoro vždy modifikující původní chování. Přidaný kód může například zvýšit čas běhu programu, nebo může ovlivnit vyrovnávací paměť, což změní přístup do paměti. Takovéto rušivé chování může, nebo nemusí být přijatelné v závislosti na účelu instrumentace.

3.1 Možnosti použití instrumentace

Instrumentační techniky se velmi liší jak ve své návrhu, tak v implementaci. V následujících sekcích jsou uvedeny a popsány hlavní účely instrumentace.

Profilování, analýza výkonu a optimalizace programu

Profilování a analýza výkonu vyžadují identifikaci nejvíce výpočetně náročných částí programu. Po nich následuje optimalizace, která tyto sekce částečně, nebo celkově přepíše. Tyto sekce mohou odpovídat rozsáhlým součástím programu jako `funkce`, nebo menším jako *základní blok* (basic blok)[1], *cyklus* (loop), nebo se může jednat o drobné části jako instrukce,

[1] Základní blok je posloupnost maximálního počtu příkazů, pro které platí, že: vstupní bod je na prvním příkazu; výstupní bod je na posledním příkazu; příkazy se prování vždy sekvenčně v pořadí dané posloupností [26].

či příkaz. Požadovaná informace pro analýzu může být získána přidáním počítadla nebo stopek ke sledovanému elementu, jehož hodnota je zvyšována s každým průchodem.

Velká část profilovacích nástrojů používá instrumentaci, jako například profilovací nástroj pro Visual Studio[2], Intel Pin[3], nebo gprof[4].

Kromě manuální optimalizace založené na výsledcích z profilovacích nástrojů, se nabízí i další možnost. Většina dnešních moderních překladačů dokáže využít tyto výsledky a provést automatickou optimalizaci.

Softwarové chyby, jejich detekce a ladění

Bez instrumentace se v podstatě neobejdou žádné nástroje na detekci chyb, především ty, které kontrolují přístup do paměti. Chyby přístupu do paměti jsou velmi častým zdrojem poruch, nebo poklesu výkonu programů u programovacích jazyků jako `C/C++`. Ty jsou schopny detekovat nástroje typu Valgrid[5] či Insure++[6], které zároveň pro tento účel používají instrumentaci.

Jedna z možností implementace je instrumentace každého přístupu do paměti. Jakmile je program instrumentován, přidané instrukce za běhu sledují přidělování a uvolňování paměti a zároveň probíhá jejich kontrola. [13]

Virtualizace

Vizualizace je technika, která napodobuje (emuluje), nebo simuluje aplikaci, operační systém, nebo dokonce celou platformu. Tato část softwaru se zpravidla nazývá *Virtuální stroj* (Virtual machine). Virtuální stroje se často využívají k přenášení, testování a migraci sotwaru na nové, nebo dosud nepodporované platformě či hardwaru. Další běžné využití virtualizace je umožnění zjednodušení implementace snadno přenositelných jazyků (`C#`, `Java`) a vytvoření schopných enginů pro interpretované jazyky jako `Python` a `Java`.

Zajištění kvality produktu a testování

Instrumentace nezřídka pomáhá v úlohách souvisejících se *Zajištěním kvality softwaru* (Software quality control – SQC), které úzce souvisí s testováním.

Dalším užitím instrumentace je analýza *Pokrytí kódu* (Code Coverage). *Kritérií pokrytí*, podle kterých se řídí vytváření testovacích sad je vícero. Pro ilustraci uveďme například *pokrytí všech řádků* (Line Coverage), při kterém jde o to, aby testy zapříčinily spuštění všech řádků testovaného kódu. Úspěšnost takovéto testovací sady se typicky udává v procentech.

K tomuto účelu ale potřebuje právě instrumentaci, která je schopná nám poskytnout vnitřní náhled do testovaného softwaru. Má možnost si například vložit na určitá místa sondy, jež kontrolují, zda tato místa byla vykonána. [5]

Mezi nástroje, které poskytují analýzu pokrytí kódu a využívají k tomuto účelu instrumentaci patří například Gcov[7], nebo Intel C++ Compiler[8], překladač, který v sobě tuto analýzu zahrnuje.

[2] https://devblogs.microsoft.com/visualstudio/new-dynamic-instrumentation-profiling/
[3] https://software.intel.com/content/www/us/en/develop/articles/pin-a-dynamic-binary-instrumentation-tool.html
[4] https://ftp.gnu.org/old-gnu/Manuals/gprof-2.9.1/html_mono/gprof.html
[5] https://valgrind.org/
[6] https://www.parasoft.com/products/parasoft-insure/
[7] https://gcc.gnu.org/onlinedocs/gcc/Gcov.html
[8] https://software.intel.com/content/www/us/en/develop/tools/documentation-library.html

3.2 Základní rozdělení instrumentace

Tato kapitola popisuje hlavní typy instrumentací. Zdrojem této sekce je kniha Instrumentace Softwaru [20] a vědecký článek obsahující mimo jiné srovnání statické a dynamické instrumentace [19].

Instrumentační techniky jsou většinou klasifikovány na základě toho, jak a kdy probíhá instrumentace. Nástroje provádějící instrumentaci před spuštěním se označují jako *statické*. Na druhou stranu ty, které modifikují kód za běhu, jsou označovány jako *dynamické*. Každá z těchto metod musí dělat kompromis mezi výkonem a univerzálností použití. Existují i nástroje, které podporují oba typy instrumentace.

Statická instrumentace

Historicky se techniky statické instrumentace objevily jako první. Jak už bylo řečeno, statická instrumentace obecně probíhá před spuštěním programu, například při překladu (více k překladu viz podsekce 4.1). V průběhu překladu je v podstatě možné provést instrumentaci v jakékoliv z jeho fází. Tato technika instrumentace se nazývá *compile-time*. Instrumentace se také dá provést na zdrojovém kódu ještě před překladem – tzv. *source-to-source instrumentace*, nebo naopak na už přeloženém spustitelném souboru. Tento druh instrumentace se nazývá *binary instrumentation*.

Jedním z příkladů nástrojů využívající statickou instrumentaci jsou rozšíření překladače gcc gcov[9] a amudflap[10].

Statická instrumentace na jedné straně šetří čas tím, že v době spuštění už není potřeba instrumentovat a případné zpomalení programu způsobuje už jen volání přidaných instrumentačních funkcí. Na druhou stranu produkuje modifikovaný výsledný kód, což nemusí být vždy přijatelné. Instrumentovat se dá také pouze kód, který je staticky linkován. Dynamické knihovny či další dynamické části se instrumentovat touto metodou nedají.

Dynamická instrumentace

Dynamická instrumentace naproti tomu probíhá při běhu programu, musí být tedy poskytnut binární spustitelný soubor (nebo bytecode). Z tohoto důvodu se také tento druh instrumentace nazývá *Dynamická binární instrumentace* (DBI), ale můžeme se setkat i s pojmem *execution-time*. Používají ji například frameworky Valgrind[11], nebo Intel Pin[12].

Na rozdíl od statické instrumentace, kde výsledný kód běží přímo v daném prostředí, nástroj DBI se vloží mezi běžící aplikaci a hostitelský systém, čímž vytváří mezivrstvu, v rámci které může kontrolovat běh programu z blízka.

Velkou výhodou tohoto přístupu je možnost instrumentovat jakoukoliv běžící část programu, včetně dynamicky generovaného kódu. Vyniká také možností sledování velkého množství informací, čehož se často využívá v oblasti bezpečnosti [17]. Není ale bez omezení. Dynamická instrumentace je velmi časově náročná a může tedy zásadně snížit výkonnost programu. Navíc je obecně náročnější na implementaci.

[9] https://gcc.gnu.org/onlinedocs/gcc/Gcov.html
[10] https://gcc.gnu.org/wiki/Mudflap_Pointer_Debugging
[11] https://valgrind.org/
[12] https://software.intel.com/content/www/us/en/develop/articles/pin-a-dynamic-binary-instrumentation-tool.html

3.3 Existující nástroje pro instrumentaci `C/C++` programů

Následující sekce uvádí přehled několika populárních instrumentačních nástrojů s jejich popisem a funkcionalitou. Pokud není uvedeno jinak, podporuje daný nástroj instrumentaci programů napsaných v jazycích `C` a `C++`.

Intel Pin

Text v této podsekci je čerpán z oficiálních manuálových stránek nástroje Pin [10] a z článku popisující tvorbu analyzačního nástroje s použití nástroje Pin [23]

Pin se řadí mezi dynamické binární instrumentační nástroje, poskytuje tedy instrumentaci za běhu. Je volně dostupný pro nekomerční účely. Rozhraní je povětšinou nezávislé na platformě. Na tomto instrumentačním frameworku jsou postaveny další nástroje jako Intel Inspector[13], nebo Intel Advisor[14].

Pin nabízí dva módy činnosti, tzv. *Just In Time* (JIT), tedy okamžitý mód a *Probe mode* neboli sondový mód. V prvním jmenovaném módu Pin v podstatě představuje JIT překladač, což znamená, že daný kód se kompiluje až je to teprve nezbytné. Jeho vstupem je již přeložený spustitelný binární soubor. Pin je schopen přerušit vykonání první instrukce spouštěného programu a vygenerovat překladem novou sekvenci kódu začínající na pozici původní první instrukce. Vygenerovaná sekvence je velmi podobná té předchozí s tím rozdílem, že si Pin zajistí získání kontroly na konci vykonávání nové sekvence. V tomto módu se provádí pouze generovaný kód, originální slouží jako reference. Pin instrumentuje pouze instrukce, které opravdu byly vykonány, a to bez ohledu na to, ze které části programu pochází (dynamická knihovna, generovaný kód, ...).

Druhý jmenovaný mód místo překladu vkládá do binárního souboru sondy na začátek specifického podprogramu (překlad angl. *subroutine*). Sonda v tomto případě představuje instrukci skoku přesměrovávající tok programu na náhradní obslužnou funkci. Na jednu stranu tento mód tolik výkonnostně nezpomaluje originální program, ale na druhou stranu ne všechna funkcionalita je dostupná stejně jako u JIC módu.

Tento nástroj je velice robustní, propracovaný a nabízí širokou sadu funkcionalit. Zároveň ale citelně zatěžuje svou režií běh programu, čímž ho zpomaluje.

XRay

Informace o nástroji XRay pochází z technické zprávy [16] a z oficiální stránek LLVM s popisem principu instrumentace XRay.

XRay je systém sledující volání funkcí původně vyvíjený jako interní projekt ve společnosti Googl a postavený na technologii LLVM. Dnes už je přímo integrovaný do projektu LLVM s open source licencí. XRay podporuje sledování a instrumentaci následující programovacích jazyků: `C`, `C++`, `Objective-C`, `Objective-C++`.

Účelem tohoto nástroje je primárně výkonnostní analýza. XRay instrumentuje vstupní a výstupní body vybraných funkcí, čímž umožňuje zaznamenávání času provedení těchto dvou bodů. Tyto záznamy mají vysokou přesnost a na jejich základě lze zpětně zjistit, kolik času strávil program v rámci dané funkce.

[13] https://software.intel.com/content/www/us/en/develop/tools/oneapi/components/inspector.html

[14] https://software.intel.com/content/www/us/en/develop/tools/oneapi/components/advisor.html

XRay se skládá ze tří hlavních částí. První vkládá na sledovaná místa sérii nulových operací, čímž si je poznačuje. Pokud je za běhu XRay vypnutý, provádí se originální kód s nulovými operacemi. Přidaná režie je v tomto případě minimální, dle technické zprávy asi 2 %. Se zapnutým XRay vstupuje do hry druhá část, knihovna, která nulové operace přepíše na volání instrumentačního kódu zaznamenávajícího informace o vstupních a výstupných bodech funkcí. Třetí častí je sada nástrojů analyzujících záznamy.

XRay je poměrně rychlý nástroj podporující i další jazyky mimo `C` a `C++`. Podporuje ale jen instrumentaci volání funkcí, ne proměnných.

Score-P

Score-P je rozsáhlá infrastruktura poskytující sadu měřících, profilovacích a analyzačních nástrojů. Informace o tomto nástroji jsou čerpány z oficiální wikipedie projektu [14].

Kromě zmíněné funkcionality umožňuje tento nástroj i několik druhů statické instrumentace, která je postavena na technologii průchodu LLVM (LLVM pass). Samotné použití je umožněno díky tzv. *obálce* (wrapper), která se volá ze souboru Makefile, či příkazové řádky namísto původního překladače s prvním argumentem obsahující jméno tohoto překladače. Score-P nabízí i několik způsobů definice, co bude instrumentováno.

Podporované překladače jsou GCC, IBM překladače, PGI, nebo překladače od Intelu. Vedle jazyků `C` a `C++` nabízí i instrumentaci jazyka `Fortran`. Score-P je primárně vyvíjen pro Linux.

Velkou výhodou je široká podpora překladačů a snadné použití instrumentace. Nástroj bohužel podporuje instrumentaci pouze funkcí, nikoliv však proměnných.

Kapitola 4

Použité nástroje

V rámci této kapitoly jsou popsány hlavní užité nástroje a technologie. První podkapitola 4.1 přibližuje technologii LLVM včetně překladače Clang. V následující kapitole 4.2 je popsán nástroj Tforc, na kterém staví tato bakalářská práce. Celou Kapitolu uzavírá sekce 4.3, jež přibližuje nástroj update-alternatives.

4.1 Technologie LLVM

LLVM je komplexní open source projekt, který v sobě zahrnuje nejen překladač, ale i další infrastrukturu, knihovny a nástroje sloužící k překladu programovacích jazyků, jako například analyzátory, optimalizátory, nebo disassembler. [7]

Původně LLVM začal jako výzkumný projekt na universitě Illinois v roce 2000. LLVM znamenal akronym pro *Low Level Virtual Machine*. Do dnešní doby se však velmi rozrostl. Včetně další podprojektů dnes obsahuje asi 2.5 milionů řádků kód a používá se celosvětově [21]. LLVM je využito u současných populárních projektů jako například Numba[1], Clint[2] nebo CodeChecker[3].

V této kapitole je popsána základní architektura LLVM, frontend pro překlad `C/C++`, tedy Clang, a opt, což je optimalizační a analytický nástroj.

Pokud není řečeno jinak, následující kapitoly čerpají z knihy Getting Started with LLVM Core Libraries [22].

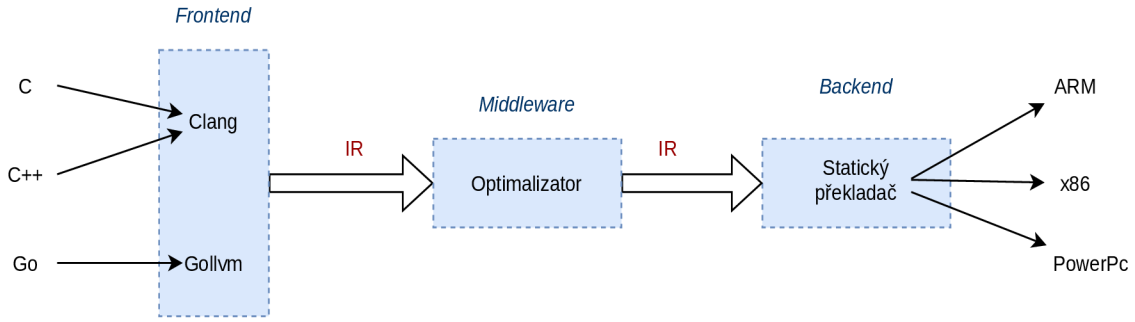
Architektura LLVM

Tato podsekce popisuje základní strukturu LLVM a zodpovědnosti daných součástí.

Oproti klasickým monolitickým překladačům má LLVM výrazně jinou strukturu. Celý projekt se skládá ze tří velkých modulárních částí: frontendu, vnitřního mezikódu (IR) a backendu. Jedna tato větší modulární část pak obsahuje menší moduly. Základní struktura je zobrazena na následujícím obrázku 4.1.

Frontend zajišťuje překlad zdrojových souborů programovacího jazyka (`C`, `C++`, `Objective-C`) do vnitřní reprezentace, tzv. *IR* formátu (Intermediate Reprezentation). Samotnému generování předchází lexikální, syntaktická a sémantická analýza. Jelikož progra-

[1] https://numba.pydata.org/
[2] https://github.com/karimmd/CLint
[3] https://github.com/Ericsson/codechecker



Obrázek 4.1: Struktura LLVM [22].

movací jazyky mají rozdílnou syntaxi a specifickou sémantiku, jeden frontend typicky slouží buď ke zpracování několika podobných jazyků, nebo jen jednoho.

```
$ clang sum.c -emit-llvm -c -o sum.bc
$ clang sum.c -emit-llvm -S -c -o sum.ll
```

Výpis 4.1: Generování IR bitekód a čitelného assembleru

IR formát je prostřední společná část překladů, která spojuje frontend s backendem. Zatímco frontend ji vytváří, backend je zpracovává. IR je také místo, kde probíhá většina optimalizací. LLVM IR existuje ve dvou formách. Jako bytekód, nebo pro člověka čitelný assembler. Generování obou verzí je ilustrováno výpisem 7.2. Mezi oběma variantami lze přecházet pomocí nástroje `llvm-as`.

Backend je část zodpovědná za generování výsledného kódu. Převádí IR na assembler, nebo na objektový binární kód, vše pro danou specifickou architekturu. Ať už se jedná o jakoukoliv architekturu (ARM, Nvidia, XCore, . . .), každá z nich má backend se stejným rozhraním.

Díky tomu, že LLVM nevytváří jeden kompaktní celek, ale jedná se o do značné míry nezávislé celky, dají se tyto celky dobře rozšiřovat. Například při vytvoření nového jazyka nám stačí vytvořit jen frontend. Middleware a backend se dají znovupoužít.

Clang

Clang poskytuje fronted a další infrastrukturu nástrojů pro jazyky příbuzné jazyku `C` (`C++`, `Objective-C/C++`, `OpenCL`, `CUDA` a `RenderScript`). Jedná se v podstatě jen o malý spustitelný soubor, který ovládá další nástroje. Tento ovladač je pak podporován ve dvou verzích, jedné kompatibilní s GNU[4] (`clang`) a jedné kompatibilní s MSVC[5] (`clang-cl.exe`). [2]

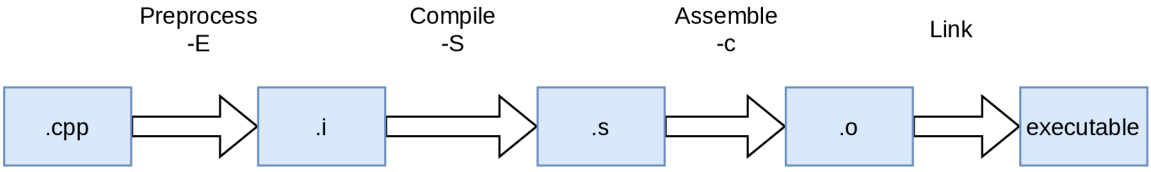
Clang se často používá u překladu aplikací, které jsou kritické z hlediska výkonu jako Chrome[6] nebo Firefox[7] [2].

[4] https://gcc.gnu.org/
[5] https://visualstudio.microsoft.com/vs/features/cplusplus/
[6] https://www.google.com/intl/cs_CZ/chrome/
[7] https://www.mozilla.org/cs/firefox/

Překlad pomocí překladače Clang

Tato kapitola čerpá z oficiálních stránek překladače Clang s příkazy [1] a referenčním manuálem [3] popisujících přepínače a jejich užití.

Jak bylo řečeno, při volání překladače Clang jsou spouštěny další nástroje, které mají na starosti různé části překladu. Ty můžete vidět na následujícím obrázku 4.2.



Obrázek 4.2: Fáze překladu s odpovídajícími přepínači a vytvořenými soubory.

Jako první přichází na řadu *předzpracování* (Preprocessing), jenž zpracovává symboly, makra a také direktivy překladače (začínající symbolem #), jako např. `#include`. Vstupem je zdrojový kód, výsledek je pak typicky pojmenován s příponou “.i” (pro `C`), nebo “.ii” (pro `C++`). Pro spuštění pouze této části se používá přepínač `-E`.

Pokud použijeme přepínač `-S`, projde kód dvěma dalšími fázemi překladu. Nejprve proběhne *syntaktická a sémantická analýza* (Parsing a Semantic Analysis). Z toho důvodu také produkuje většinu nalezených varování právě tato část. Produktem této fáze je *Abstraktní Syntaktický strom* (Abstract Syntax Tree – AST). AST je předán *generátoru kódu a optimalizátoru* (Code Generation a Optimization). Generátor překládá AST do nízkoúrovňového mezikódu (LLVM IR) specifického pro danou architekturu. Naprostá většina optimalizací je prováděna zde. Konečný výstup mívá příponu ".s"a jedná se o assembler. Obě dvě fáze dohromady se nazývají *Compilation*, tedy překlad, což je lehce matoucí. V kombinaci s přepínačem `-S` můžeme použít přepínač `-emit-llvm` k vygenerování `IR`.

Předposlední fáze *Assembler* má na starosti překlad assembleru na tzv. objektový soubor s typickou příponou ".o".

Nakonec se musí všechny překládané části sloučit do jedné, což je provedeno pomocí nástroje *Linker*. Výstupem může být jak spustitelný soubor, tak knihovna. Tato část je bez přepínače.

Kompatibilita s GNU

Clang byl původně zamýšlen jako alternativa ke GNU překladačům. Prvotním cílem bylo umožnit uživateli, aby mohl lehce vyměnit Clang za GCC bez nutnosti upravovat stávající systém překladu.

Oba dva projekty jsou v dnešní době velmi komplexní a oba nabízí velké množství přepínačů. V případě Clangu jich je na manuálové stránce [4] popsáno přes 1200, v základním soupisu na stránkách GCC [9] jich najdeme asi 1000. Některé přepínače navíc ani nejsou zdokumentované. Toto vše znamená, že opravdu není v moci Clangu zaručit třeba jen poloviční kompatibilitu. Podporované jsou především základní a nejvíce používané argumenty, které často vystačí v běžných projektech. Projekty vyloženě postavené na jednom druhu překladače, nebo využívající nějaký z nástrojů specifický danému překladači, jsou nepřenositelné.

Lehký problém nastává i tam, kde je sice přepínač pojmenován stejně, ale dokumentace naznačují, že může mít lehce odlišnou funkcionalitu. V takové případě nezbývá než tyto možnosti otestovat.

Nástroj opt

Nástroj opt je LLVM optimalizátor a analyzátor. Jeho vstupem je LLVM IR soubor, výstupem optimalizovaný soubor nebo jeho analýza. Funkcionalita nástroje závisí na tom, jestli je použit přepínač `-analyze`, nebo ne. [8]

Kromě těchto dvou vestavěných funkcionalit, nabízí opt i možnost napsat si svůj vlastní `zásuvný modul` (Dynamic plugin), který rozšiřuje funkcionalitu nástroje. Pro tuto potřebu jsou implementovány podtřídy třídy `Pass`, které umožňují procházet dané struktury kódu. V rámci LLVM jsou k dispozici například:

- ModulePass — průchod celým modulem
- FunctionPass — průchod jednotlivými funkcemi modulu
- LoopPass — prochází cykly vyskytující se v modulu
- BasicBlockPass — prochází všemi základními bloky modulu

Nový modul se do opt nahraje pomocí `-load=<plugin>`. Opt v rámci pluginu povoluje i definování vlastních vstupních argumentů [8].

Pomocí vytvořených modulů lze nástroj opt použít i jako základ instrumentačního nástroje. Je na něm postaven nástroj Score-P[8], nebo Tforc[9].

4.2 Tforc

Tato podkapitola popisuje nástroj Tforc použitý v rámci této bakalářské práce. Jedná se o statický instrumentační framework vytvořený Václavem Ševčíkem sloužící k instrumentaci `C++` programů. Zdrojem této kapitoly je diplomová práce Václava Ševčíka [29] a dokumentace projektu, která je spolu s projektem dostupná ke stažení na serveru výzkumné skupiny Testos – pajda[10].

První část se věnuje funkcionalitě a ovládání programu, druhá implementovanou strukturou. V poslední podsekci je popsán jazyk sloužící k popisu instrumentace.

Funkcionalita

Hlavní funkcionalita spočívá v tom, že Tforc umožňuje sledování přístupu do paměti (zápis, čtení) a volání funkcí, čehož dosahuje pomocí instrumentace při překladu (více k instrumentaci viz kapitola 3). Při instrumentaci proměnných dovoluje instrumentovat i nepřímou adresaci.

Zjednodušeně se dá říct, že nástroji Tforc specifikujeme, co se má sledovat a jak se na to má reagovat. Podle těchto specifikací se při překladu na daná místa, na kterých se má reagovat, vloží kód zajišťující tuto reakci za běhu.

V současném stavu Tforc dovoluje instrumentovat pouze jeden soubor z projektu.

[8] https://www.vi-hps.org/projects/score-p/
[9] https://pajda.fit.vutbr.cz/testos/tforc
[10] https://pajda.fit.vutbr.cz/testos/tforc

Instrumentace funkcí

Obecně se instrumentace funkcí dělí na dvě kategorie, na tzv. *vnější* a *vnitřní*. U vnitřní se vkládaný kód vloží na začátek funkce nebo před každý příkaz `return`. V případě Tforc byla použita vnější instrumentace, která sice vyžaduje nutnost vkládat instrumentační kód na každé místo volání, ale na druhou stranu poskytuje tento typ instrumentace více informací o kontextu.

```
// definice funkce
void func{
    ...
}
...

instrumentacni funkce pred volanim
func();
instrumentacni funkce po volani
```

Výpis 4.2: Pseudokód vnější instrumentace funkce

Tforc podporuje variantu instrumentace jak před voláním funkce, tak po ní. Vložení instrumentační funkce je v obou případech znázorněno ve výpise 4.2.

Instrumentace přístupu do paměti

Instrumentace přístupu do paměti znamená z pohledu programovacího jazyka instrumentaci proměnných. Framework nabízí instrumentaci jak globální, tak lokálních proměnných, kde sleduje jejich zápis, nebo čtení. Na zápis reaguje předem danou akcí díky vložené instrumentační funkci, jejíž pozice závisí na typu sledované akce, viz pseudokód 4.3.

```
int var;
...
instrumentacni funkce pred zapisem
var = 42;
...
foo = var;
instrumentacni funkce po cteni
```

Výpis 4.3: Pseudokód instrumentace zápisu a čtení nástroje Tforc.

V případě zápisu se vloží instrumentační funkce před originální kód zápisu. V případě čtení je vložena až za originální kód.

Nepřímá adresace

Nepřímá adresace je metoda, při které obsahem adresy je další adresa, jež vede ke konečné hodnotě [6]. V kontextu `C/C++` programů se jedná o ukazatele.

Pokud vezmeme v potaz následující příklad 4.4, funkce `increase` nám může v závislosti na daném spuštění a daném překladači ovlivnit nejen proměnnou `first_number`, ale i proměnnou `second_number`. Jestliže dojde k tomuto neúmyslného ovlivnění, lze se zapnutou nepřímou instrumentací v Tforc sledovat i zápis proměnné `second_number` ve funkci `increase`.

```
void increase(int * num){
    num[0]++;
    num[1]++;
}

int main(){
    int first_number = 42;
    int second_number = 7;
    int *num_ptr;
    num_ptr = &first_number;

    increase(num_ptr);
    return 0;
}
```

Výpis 4.4: Ukázka nepřímé instrumentace

Zapnutím nepřímé instrumentace se může skokově zvýšit počet počet vkládaných funkcí oproti přímé instrumentaci, což může mít za následek výrazné zpomalení běhu instrumentovaného programu. Při klasické přímé instrumentaci je zpravidla počet vkládaných funkcí menší, a tedy i program může být rychlejší.

Vícevláknové programy

Nástroj zvládá i instrumentaci vícevláknových programů. Zajištění správného přístupu ke zdrojům je možné pomocí globálního sdíleného zámku.

Struktura

Následný popis struktury je pouze zjednodušený, nerozebírá například dopodrobna knihovnu Tforc.so. Rozepsanou strukturu můžete nalézt v kapitole 5.1 v diplomové práci Václava Ševčíka [29]

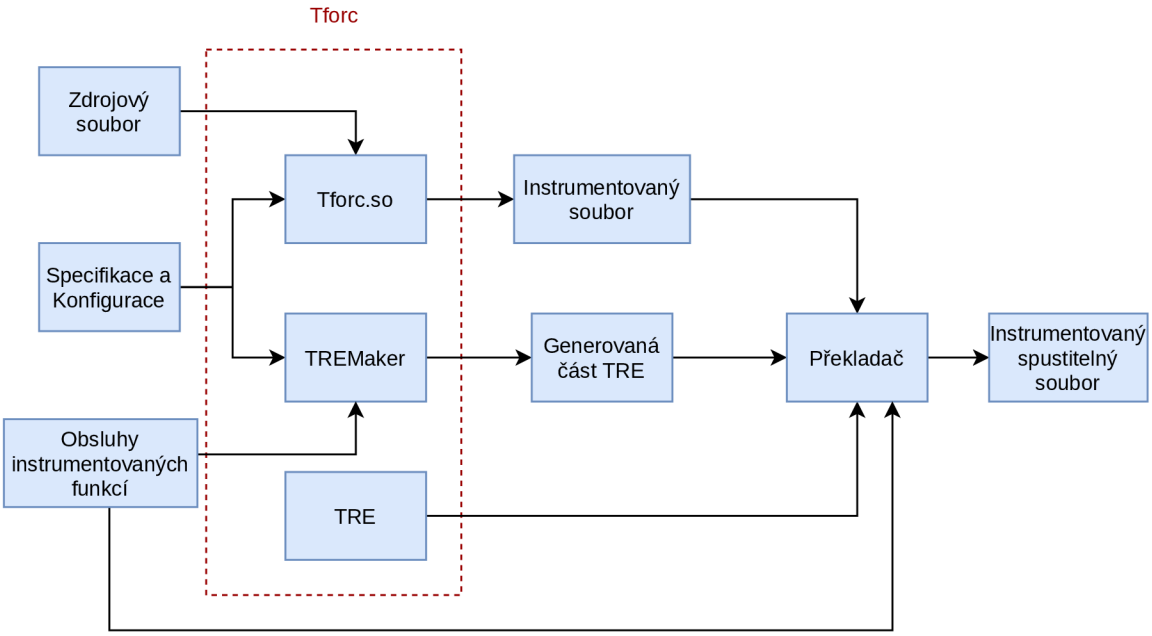
Schéma propojení součástí Tforc je zobrazeno na následující obrázku 4.3.

Hlavním vstupním souborem je zdrojový `C` soubor, který je určen k instrumentaci. Dalším vstupem je položka souhrnně nazvaná `Queries and Configuration`. Jedná se o konfigurační soubor určený pro Tforc obsahující informaci, zda se má provést nepřímá instrumentace a kde se nachází soubor se specifikacemi instrumentace, tzv. *queries*. Konfigurační soubor má zpravidla strukturu zobrazenou na výpise 4.2. Lehce zmatečná je cesta ke specifikacím referovaná jako `configuration`. Druhý soubor s těmito specifikacemi udává požadavky na instrumentaci. Pro jejich popis je použit speciální jazyk, který je vysvětlen v následující podsekci.

```
configuration: config_instrument
indirectAddressing: yes
```

`Callback` obsahuje definice funkcí, které slouží jako obsluhy instrumentovaných funkcí a proměnných.

Tforc jako takový má dvě hlavní části: knihovnu `Tforc.so` a `Tforc Runtime Engine` (`TRE`). Knihovna slouží k instrumentaci vstupního souboru ve formátu `IR`. Při průchodu souborem hledá artefakty, které si přeje uživatel sledovat, a v případě shody tato místa



Obrázek 4.3: Schéma vzájemné propojení a závislostí Tforc frameworku. Převzato z [29].

instrumentuje. Kromě instrumentace samotné je potřebná i správa proměnných a informací o nich. Tu zajišťuje `TRE`. Jedna jeho část je statická a stará se o obsluhu instrumentovaných funkcí. Druhá je dynamická tvořená při překladu skriptem `TREMaker` a slouží pro správu nepřímé adresace. Výstup v podstatě vytváří vyhledávací tabulku (*Look Up Table – LUT*) obsahující informace o provázání nepřímých adres s obslužnými funkcemi. Pokud je nepřímá instrumentace vypnutá, tabulku je prázdná.

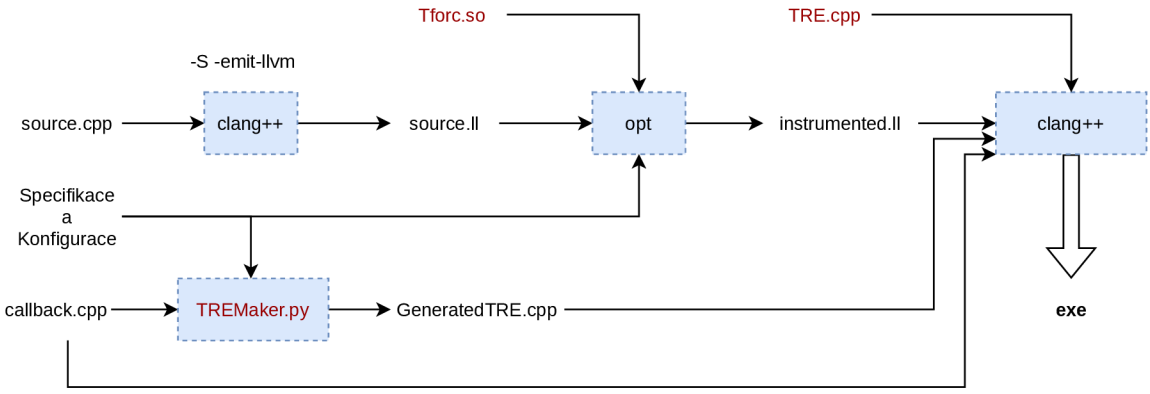
Pro výsledný spustitelný instrumentovaný soubor je potřeba spojit: instrumentovaný soubor, vygenerovaný `TRE`, statický `TRE` a obslužné funkce (`Callback`).

Součástí projektu Tforc je předpřipravný Makefile, který slouží k překladu a instrumentaci. Schéma postupu tohoto souboru je zobrazené na obrázku 4.4. Oproti předešlému je zde rozvedeno použití všech nástrojů v každé fázi překladu. Pokud bychom si chtěli vyzkoušet použití frameworku ručně, postup by byl v podstatě stejný.

Hlavní část, instrumentace, se provádí pomocí LLVM nástroje opt (viz sekce 4.1 popisující nástroj opt), kterému je mu předložen zásuvný modul v podobě knihovny `Tforc.so`. Vstupem je pak zdrojový soubor ve formátu IR vytvořený s pomocí přepínačů `-S -emit-llvm`, výstupem modifikovaný instrumentovaný soubor. Ostatní vytvořené, vygenerované části, nebo statické části, které jsou potřeba pro běh vytvořeného spustitelného souboru, se linkují pomocí Clangu. Jejich formát, nebo fáze překladu mohou být jakékoliv podporované Clangem.

Jazyk frameworku Tforc

Tato podsekce popisuje strukturu jazyka vytvořeného pro potřeby Tforc frameworku. Jazyk je použit pro specifikaci instrumentace, jeho obsahem jsou dané jednotlivé požadavky (*queries*), které mohou být dvojího typu: instrumentace funkce, nebo proměnné.



Obrázek 4.4: Schéma překladu a instrumentace nástrojem Tforc programu v jazyce `C++` s odpovídajícími přepínači. U každého nástroje jsou vyznačeny jeho vstupy a výstupy. Červená barva označuje části z nástroje Tforc.

Obecný předpis pro oba typy instrumentací je znázorněn na dalším obrázku 4.5. Na začátku každého požadavku je definováno, o jaký typ instrumentace se jedná. Za dvojtečkou následuje název proměnné nebo funkce, kterou chceme instrumentovat, následně jméno instrumentující funkce – jednu z funkcí definovanou v souboru s obslužnými funkcemi. Na konci jsou uvedeny dodatečné informace k instrumentaci.

```
instr_type: what_to_instrument instrumenting_fucntion other_information
```

Obrázek 4.5: Obecný předpis jednoho požadavku. Převzato z [29].

Jedna z těchto dodatečných informací pro framework, jež je společná oběma typům instrumentace je klíčové slovo `CCODE` specifikující, že obslužná funkce je v jazyce `C`.

V rámci celého souboru lze využívat jednořádkové komentáře, které se uvozují znakem #. Komentář se může vyskytovat i na řádku za specifikací.

Dekorace `C++` funkcí

Tato podsekce čerpá z dokumentu popisujícího konvence tvorby dekorací u různých `C++` překladačů na různých platformách [15].

Instrumentující funkce ve specifikaci vyžadují speciální zápis pomocí *dekorace* (mangling) v případě užití jazyka `C++`. Tuto metodu využívají `C++` překladače, aby obohatily názvy funkcí a objektů v objektovém souboru. To pak slouží linkeru v případě, že zmiňované položky jsou definovány v jednom souboru, ale vyskytuje se na ně odkaz i v dalších. Obecně umožňují dekorace následující:

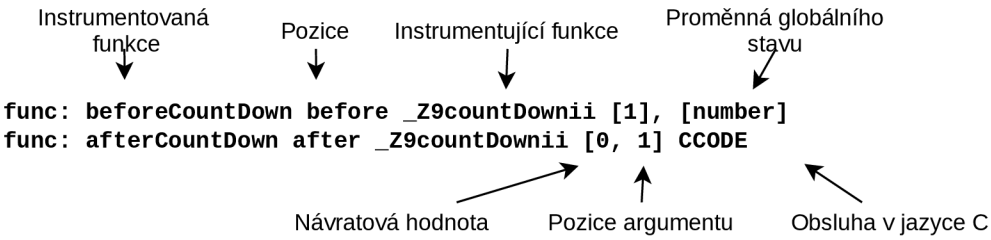
- Umožnění linkeru rozlišit mezi různými verzemi přetížené funkce.
- Umožnění linkeru zkontrolovat, zda jsou objekty a funkce deklarovány správným způsobem ve všech modulech.

- Umožnění linkeru předat všechny potřebné informace v případě neřešitelné reference (unresolved references).

Každá architektura a každý překladač má odlišný styl dekorování. V případě Clang++ lze formu dekorované funkce nalézt v souboru formátu IR, který lze vygenerovat ze zdrojového pomocí přepínače `-S -emit-llvm`. Další možností je použití nástroje `nm`, který vypíše seznam všech použitých dekorovaných jmen ve spustitelném souboru.

Instrumentace funkcí

Příklad dvou požadavků na instrumentaci funkce je uveden na obrázku 4.6. Celá specifikace je uvozena klíčovým slovem `func` a oddělovačem `":"`. Jako první přichází na řadu název instrumentované funkce, která je v klasickém formátu bez dekorací, a to v případě jak jazyka `C` tak `C++`. Za ním následuje instrumentující funkce uvozena pozicí. Pozice může být dvojího typu: před instrumentovanou funkci (`before`) nebo za ní (`after`). V prvních hranatých závorkách jsou poziční argumenty udávající čísla pozičních argumentů, které se mají předávat z instrumentované funkce do vkládané. Index argumentů začíná od 1, protože nula označuje předání návratové hodnoty.



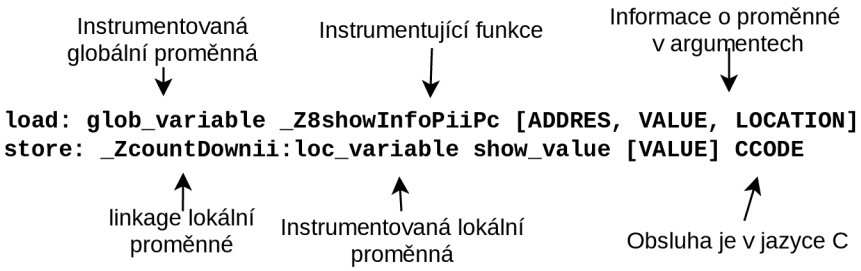
Obrázek 4.6: Příklad specifikace instrumentace funkce. Převzato z [29].

Poslední argument v hranatých závorkách je volitelný. Díky němu lze předat stavy globálních proměnných. Dalším volitelným argumentem je klíčové slovo `LOCATION`, které vytvoří výpis o instrumentované funkci.

Kromě instrumentace jednotlivých funkcí lze instrumentovat i všechny uživatelem definované funkce v daném souboru pomocí žolíka `"*"`. Dva žolíci `"**"`na místě instrumentované funkce způsobí instrumentaci všech volání funkcí dostupný při překladu včetně těch systémových a knihovních.

Instrumentace proměnných

Specifikace instrumentace proměnné (viz obrázek 4.7) je uvozena klíčovým slovem `load`, nebo `store`, podle požadované instrumentace. Operace `load` označuje čtení z paměti, `store` naopak zápis. Pokud chceme sledovat obě operace, musí se definovat obě specifikace. Za tímto klíčovým slovem opět následuje dvojtečka. Hlídaná proměnná může být dvojího druhu. Buď se jedná o globální proměnnou a ve specifikaci stačí uvést pouze její jméno, nebo o lokální proměnnou. V takové případě se zápis skládá z dvojice: název dekorované funkce, v rámci které je proměnná definována (tzv. *linkage*), a název proměnné.



Obrázek 4.7: Příklad specifikace instrumentace proměnné. Převzato z [29].

Jako další následuje opět jméno instrumentující funkce. Ve složených závorkách na konci jsou uvedeny informace, které se mají předat instrumentující funkci. Tato klíčová slova jsou:

- `ADDRESS` – adresa, na které je proměnná uložena
- `VALUE` – hodnota přečtené nebo zapsané proměnné
- `LOCATION` – řetězec s informacemi o pozici operace ve zdrojovém kódu (řádek, funkce a soubor)

U instrumentující funkce se očekává, že typ a pořadí jejích argumentů bude odpovídat specifikaci. Argumenty mají pevně daný typ v jazyce `C`, a to:

- `ADDRESS` – `int *`
- `VALUE` – podle typu proměnné, u ukazatelů jednotně `int *`
- `LOCATION` – `char *`

U argumentu udávající aktuální hodnotu proměnné samozřejmě závisí datový typ na typu sledované proměnné.

Výhody a nevýhody

Tforc poskytuje základní instrumentaci funkcí a proměnných, která zahrnuje i instrumentaci vícevláknových programů. Na rozdíl od komplexních dynamických instrumentačních nástrojů jako například Pin[11] má za běhu mnohem menší režii a nezpůsobuje tedy takové zpomalení. Při vypnuté nepřímé adresaci program zabere dle provedených experimentů průměrně o 14 % více času. Při zapnuté nepřímé adresaci se průměrný čas programu zvýší o 23 %. Užitečnou funkcionalitou je použití žolíků nebo zpětné získání informací z nástroje. Další výhodou je možnost rozšíření instrumentace i programů napsaných v ostatních jazycích, pokud je LLVM schopen je přeložit do formátu LLVM IR.

Tforc má i určitá omezení. Tím, že se jedná o statický compile-time instrumentační nástroj, musí být k dispozici zdrojový kód. Tforc také nebude fungovat v případě, že projekt nelze přeložit pomocí Clangu nebo má příliš složitou strukturu. Použití Tforc je v podstatě závislé na předpřipraveném souboru Makefile. Druhou možností použití je změnit kompletně strukturu překladu. Tforc je také zatím schopen instrumentovat pouze jeden soubor.

[11] https://software.intel.com/content/www/us/en/develop/articles/pin-a-dynamic-binary-instrumentation-tool.html

4.3 Update-alternatives

Tato sekce pojednává o nástroji update-alternatives. Jedná se o jeden z Linuxových nástrojů patřící do skupiny *systém alternativ* (alternativ system) nástrojů, které obecně slouží ke správě symbolických odkazů vedoucích na příkazy. Zdrojem této kapitoly jsou manuálové stránky Linuxu [12] a Ubuntu [11].

Typickým příkladem užití je případ, kdy existují dvě či více verzí toho stejného programu, nebo více programů, které mají podobný účel. S pomocí update-alternatives lze snadno nastavit výchozí program, nebo mezi danými programy přepínat.

Před samotným popisem principu update-alternatives je třeba vysvětlit některé pojmy.

- generic name: Generické jméno, např. /usr/bin/editor, které odkazuje na jeden ze skupiny souborů s podobnou funkcionalitou.
- symlink: Symbolický odkaz ve složce obsahující alternativy.
- alternative: Alternativa, název specifického souboru v souborovém systému, který může být přístupný přes generické jméno použitím systému alternativ.
- alternatives directory: Složka obsahující symbolické odkazy, jako výchozí je nastavena na `/etc/alternatives`.
- administrative directory: Složka obsahující informace o stavu alternativ, jako výchozí je nastavena `/var/lib/alternatives`.
- link group: Skupina odkazů, množina souvisejících symbolických odkazů.
- automatic mode: Pokud je daná skupina odkazů přepnuta do automatického módu, je systémem alternativ zajištěno, že se zvolí odkaz s největší prioritou ve skupině.
- manual mode: Pokud je daná skupina odkazů přepnuta do manuálního módu, systém alternativ výběr udaný uživatelem.

Generické jméno v souborovém systému je sdíleno všemi soubory, které poskytují navzájem zaměnitelnou funkcionalitu. Systém alternativ a uživatel dohromady určují, na který soubor bude nakonec odkazovat generické jméno.

Například, když jsou nainstalovány dvě verze jazyka `Python` na tom stejném systému, systémem alternativ lze zajistit, že generické jméno (`/usr/bin/python`) odkazuje pouze na jednu z verzí s nejvyšší udanou prioritou. Toto lze potlačit, pokud uživatel napevno zadá jednu určitou alternativu, čímž systém alternativ zároveň přejde do manuálního módu.

Obrázek 4.8: Struktura symbolických odkazů vedoucí od generického jména ke konečnému souboru v případě použití update-alternatives. Ukázka na příkladu jazyka python.

Generické jméno není přímý symbolický odkaz na vybranou alternativu. Místo toho vede na jméno ve složce alternativ, což je další symbolický odkaz na už konkrétní soubor. Řetězec těchto odkazů je zobrazen na předchozím obrázku 4.8.

Spravovat systém alternativ je umožněno pouze s administrátorským přístupem. Nové alternativy by měly být vytvářeny s velkou opatrností. Pokud totiž vytvoříme alternativu s novým generickým jménem, které už ale v systému mimo systém alternativ existuje, způsobíme nevratné odstranění původního odkazu nebo souboru. Systém alternativ neposkytuje v tomto případě žádné varování.

Další informace k použití a funkcionalitě tohoto nástroje jsou uvedeny v manuálových stránkách nástroje update-alternatives [12].

Kapitola 5

Návrh instrumentačního nástroje TforcTool

V této kapitole je popsán základní návrh nástroje pojmenovaného TforcTool, který je předmětem této bakalářské práce. V první sekci 5.1 se nachází přehled požadavků na nástroj, v následující podsekci 5.2 jsou vyjmenované použité nástroje a poslední podsekce 5.3 rozebírá základní strukturu.

Projekt řešený v rámci této bakalářské práce staví na nástroji Tforc (viz sekce 4.2). Má za cíl poskytnout snadnou instrumentaci `C++` programů za překladu.

5.1 Specifikace požadavků

Obsah sekce podává přehled požadavků na vyvíjený nástroj. Požadavky vychází ze zadání bakalářské práce, z diskuze s vedoucím práce a z některých současných omezení nástroje Tforc.

- **Snadná konfigurace** (sekce 6.4). Tforc zatím vyžaduje specifikaci dvou konfiguračních souborů bez možnosti jakéhokoliv výchozího nastavení. Konfigurace instrumentace by měla být obsažena v jednom souboru s podporou výchozího nastavení.
- **Instrumentace beze změny stávající struktury překladu** (sekce 6.1). Pro spuštění instrumentace pomocí Tforc je nutné použít předpřipravený Makefile, který se dále modifikuje. Další nástroje pracující na podobné principu jako Tforc často požadují modifikaci stávající struktury překladu. Vyvíjený nástroj by však měl stávající strukturu překladu ponechávat.
- **Instrumentaci přístupů do paměti a volání funkcí** (sekce 4.2). Tento požadavek je splněn výběrem nástroje Tforc.
- **Výkon nástroje** (sekce 7.3). Nástroj by neměl markantně zpomalovat překlad v porovnání s použití samotného nástroje Tforc.
- **Podpora instrumentace programů překládaných pomocí GNU** (sekce 6.2). Program by měl být schopen instrumentovat jednoduché projekty překládané pomocí GNU.

- **Možnost integrace s nástrojem Spectra** (sekce 6.1). Nástroj by měl být otevřený pro budoucí integraci s nástrojem Spectra. To znamená možnost zaznamenat volání funkcí a změnu hodnoty proměnné, a zároveň vyčíst její hodnotu. Další podmínkou je možnost automatického spouštění instrumentace, pokud možno pouze s běžným uživatelským oprávněním.
- **Použití nástroje Clang** (sekce 6.3). Clang je použit už v nástroji Tforc a TforcTool jej taktéž využívá.
- **Vytvoření automatické testovací sady** (sekce 7.1). Program by měl být otestován automatickou testovací sadou.

5.2 Přehled použitých nástrojů a jazyků

Jak už bylo zmíněno, tento program rozšiřuje a využívá původní instrumentační nástroj Tforc (viz sekce 4.2). Hlavní důvod k jeho výběru byl fakt, že pokrývá požadované typy instrumentací. Jelikož byl taktéž vyvíjen v rámci výzkumné skupiny VeriFIT, počítalo se s jeho začleněním do platformy Testos. Současně se jedná o ideální instrumentační nástroj, který lze v budoucnu propojit s nástrojem Spectra, což by umožnilo automatickou verifikaci za běhu.

Jako hlavní jazyk byl vybrán `Python`, a to konkrétně ve verzi 3.8, kterou používá i nástroj Tforc. Nerozšiřují se tím tedy závislosti celého programu.

Dalšími nástroji jsou opt a Clang (viz podsekce 4.1), což jsou nezbytné části pro fungování Tforcu. Posledním nástrojem je make.

Program byl vyvíjen pro platformu Linux.

5.3 Struktura programu TforcTool

Následující text vysvětluje základní strukturu implementovaného nástroje. Detaily implementace a popis řešených problému jsou pak obsaženy až v následující kapitole 6.

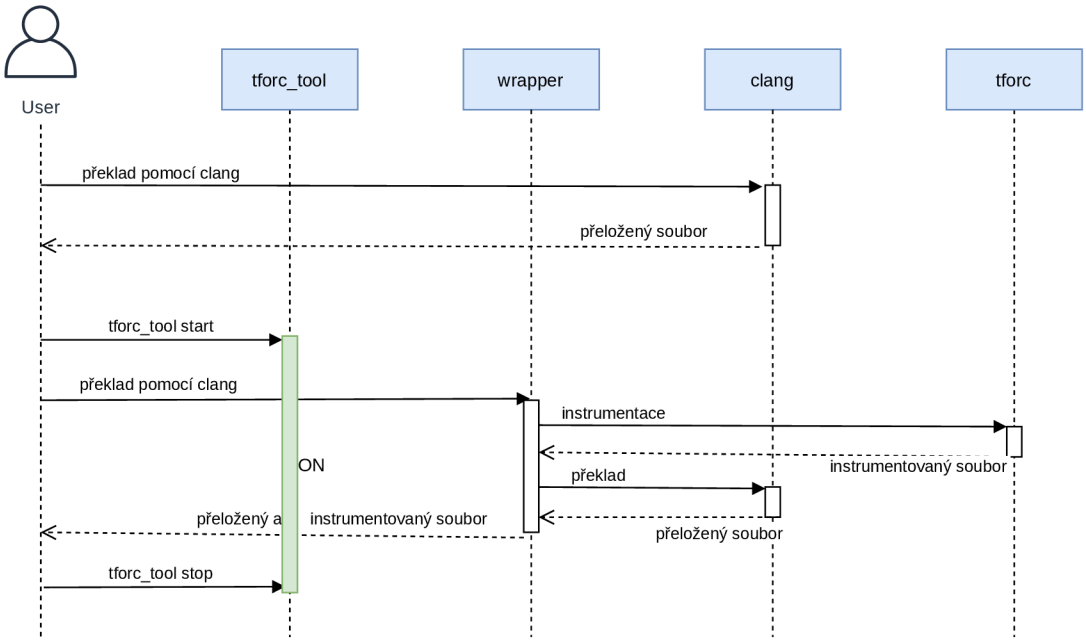
TforcTool můžeme rozdělit do tří částí, a to `tforc_tool`, `wrapper` a `Tforc`. Nástroj zajišťuje instrumentaci pouze zvoleného projektu (souborů v zadané složce) překládaného jedním zvoleným překladačem.

První část pojmenovaná `tforc_tool` zajišťuje spuštění nástroje jako takového. Jelikož jeden z požadavků je zachování stávající struktury překladu bez nutnosti modifikace, je třeba nějakým způsobem obejít originální překladač a zajistit spuštění vlastního software. Metody zajišťující toto chování jsou podrobně popsány v podsekci 6.1. Kromě této hlavní zodpovědnosti zpracovává a ukládá další nastavení specifikované uživatelem (konkrétní popis v sekci 6.4) spolu s interními informacemi, a to do vnitřního souboru, který je využíván zároveň i dalšími částmi nástroje. Jeho podrobný obsah je k nalezení v sekci 6.5.

Zmiňovaný vlastní software, který je volán místo originálního překladače, se v našem případě jmenuje `wrapper` a tvoří druhou část programu. Zajišťuje instrumentaci a překlad. Jejich řízení je dále rozvedeno v sekci 6.3. Kromě toho zpracovává a kontroluje vstupní argumenty, nebo je i překládá, pokud se jedná o argumenty pro GNU překladač. Více k této implementaci je popsáno v sekci 6.2. Má také na starosti nalezení a správu dalších vstupních souborů, či konfigurací a zajišťuje jejich převod do formy přijatelné nástrojem Tforc. Vstupní uživatelské soubory a jejich struktura se mírně liší od těch používaných

Tforcem. Jedná se o konfigurační soubor patřící k danému projektu a soubor s obslužnými funkcemi, jejichž popis je uveden v sekci 6.4.

Třetí částí je již existující nástroj Tforc, který je již popsán v sekci 4.2.



Obrázek 5.1: Sekvenční diagram ukazující volání překladu bez zapnutého nástroje TforcTool a posléze se zapnutým nástrojem TforcTool umožňující kromě překladu i instrumentaci.

Propojení všech tří částí je demonstrováno na příkladu jejich vzájemných volání na obrázku 5.1. Obrázek můžeme rozdělit na dvě sekce. V první je volán překladač Clang bez zapnutého nástroje TforcTool. Dle parametrů je proveden požadovaný překlad nebo jeho část. Ve druhé sekci je nejprve instrumentační nástroj aktivován a jakákoliv další volání už směřují na zástupný `wrapper`. Podle fáze překladu dané volanými argumenty `wrapper` buď provádí překlad pomocí překladače Clang, nebo instrumentuje pomocí Tforcu, případně obojí.

Kapitola 6

Implementace instrumentačního nástroje TforcTool

Tato kapitola popisuje implementaci nejdůležitějších částí nástroje zajišťující klíčovou funkcionalitu. První podkapitola 6.1 seznamuje se způsobem umožňující běh celého nástroje bez potřeby úpravy stávajícího způsobu překladu. Ve druhé sekci 6.2 je popsáno zpracování argumentů použitých pro volání původního překladače, na což navazuje sekce 6.3 zabývající se interním řízením překladu a instrumentací samotnou. Poslední dvě podkapitoly popisují strukturu doprovodných souborů. První z nich 6.4 specifikuje soubory požadované po uživateli, druhá sekce 6.5 se týká interního uchování dat.

Jak `wrapper`, tak `tforc_tool` jsou napsány v jazyce `Python`. Pokud se někde v textu objeví, že tyto části volají některý z Linuxových nástrojů (např. `which`), jedná se buď o funkci v jazyce `Python` ze základních balíků s ekvivalentním chováním (nejčastěji z balíku os[1]), nebo skutečné volání daného nástroje, které umožňují funkce v jazyce `Python` jako `os.system`[2] nebo `subprocess.run`[3].

6.1 Obejití volání originálního překladače

Jeden z požadavků na vyvíjený program je možnost zachování stávajícího způsobu překladu. Dynamické instrumentační nástroje tento požadavek splňují automaticky, jelikož jejich vstupem je už přeložený spustitelný soubor, v případě compile-time nástroje se tak ale neděje. Vznikla zde tedy potřeba určitým způsobem obejít a předběhnout volání stávajícího originálního překladače a zajistit místo něj volání TforcTool části nazvané `wrapper`. Tento `wrapper` je pak pokaždé zkopírován na určité místo a pojmenován po originálním překladači. Jak už bylo řečeno, tuto funkcionalitu zajišťuje část nazvaná `tforc_tool`.

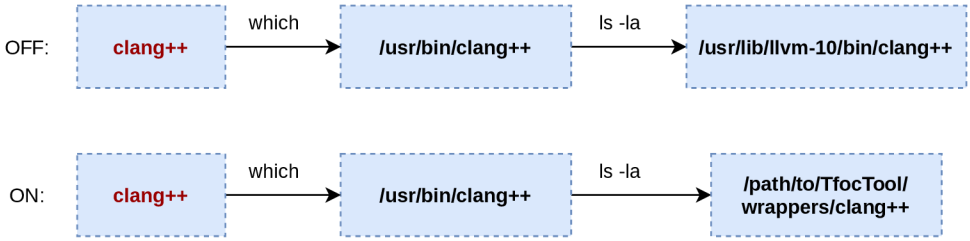
Pro tento účel byly implementovány tři metody, které jsou popsány v následujících podsekcích. Po představení metod je k dispozici krátké shrnutí v poslední podsekci s jejich porovnání. Všechny z následujících metod byly vyvinuty pro platformu Linux.

[1] https://docs.python.org/3/library/os.html
[2] https://docs.python.org/3/library/os.html
[3] https://docs.python.org/3/library/subprocess.html

Metoda nahrazení originálního překladače

První a nejintuitivnější metodou je nahrazení originálního překladače nebo symbolického odkazu na něj v místě jeho volání. Lokace originálního překladače je zjištěna pomocí funkce `which` a následně je tento soubor přejmenován na `<nazev_prekladace>.orig`. Volání překladačů na Linuxu typicky směřuje prvotně do složky `/usr/bin`. Místo překladače se na dané místo vloží symbolický odkaz na nově zkopírovaný `wrapper`. Při deaktivaci nástroje TfocTool se postupuje v opačném pořadí. Nejdříve je smazán symbolický odkaz na `wrapper`, poté přejmenován originální překladač na původní jméno a nakonec smazána vytvořená kopie `wrapper`.

Princip této metody je sice nejpřímější, ale také nejinvazivnější. Jak aktivace, tak deaktivace této metody je implementována tak, aby při přerušení jakékoliv fáze bylo vždy možné obnovit původní překladač.



Obrázek 6.1: Ilustrace funkcionality metody nahrazení pomocí nástrojů `which` a `ls` na příkladě překladače Clang++.

Předchozí obrázek 6.1 ilustruje s pomocí příkazů `which` a `ls` rozdíl mezi systémem s deaktivovanou a aktivovanou metodou.

Metoda upravující systémovou proměnnou PATH

Další z vytvořených metod upravuje a rozšiřuje systémovou proměnnou `PATH` o cestu k nově vytvořenému souboru `wrapper` pojmenovaném přesně jako originální překladač. Při vyhledávání spustitelného souboru se v Linuxu prochází cesty ke složkám uvedené v `PATH` postupně od první položky až do konce, nebo dokud se nenajde požadovaný spustitelný soubor. Položky jsou od sebe oddělené dvojtečkou. V našem případě se přidáním další položky vyskytnou v `PATH` dvě možnosti, proto cesta k složce obsahující `wrapper` musí být první v pořadí.

```
 # Tforc tool 2021-04-16, add wrapper to PATH before original compiler
case ":$PATH:" in
*:/home/user/Bakalarka/TForcTool/tforcetool/wrappers:*)
    ;;
*)
    PATH="/home/user/Bakalarka/TForcTool/tforcetool/wrappers:$PATH"
    ;;
esac
```

Výpis 6.1: Výpis kódu přidaného do uživatelem zvoleného souboru při aktivaci metody upravující PATH

Při spuštění programu z příkazového řádku je tento program spuštěn jako nový proces dědící systémové nastavení od svého rodičovského procesu. Zpětně se ale změny v prostředí nepropisují. Nastavením proměnné `PATH` v rámci běžícího nástroje by se tedy navenek nijak neprojevilo. Proto je místo toho upraven uživatelem zvolený soubor (výchozí je `~/.bashrc`), do kterého je přidán kód obsažený ve výpisu 6.1.

Po uživateli je následně požadováno, aby modifikovaný soubor buď spustil pomocí příkazu `source` (nebo `.`), což způsobí vykonání souboru v současném kontextu (současném terminálu), nebo aby otevřel nové okno terminálu načítající změněný soubor. Podobně při vypnutí musí uživatel vykonat jednu z těchto činností, aby obnovil chod originálního překladače. Toto obnovení je umožněno nahrazením předchozího kódu 6.1 následující sekvencí 6.2 ve stejném souboru.

```
# Tforc tool 2021-04-07, remove wrapper from PATH
PATH=$(echo "$PATH" | \
sed -e 's/\/path\/to\/TforcTool\/tforctool\/wrappers://')
```

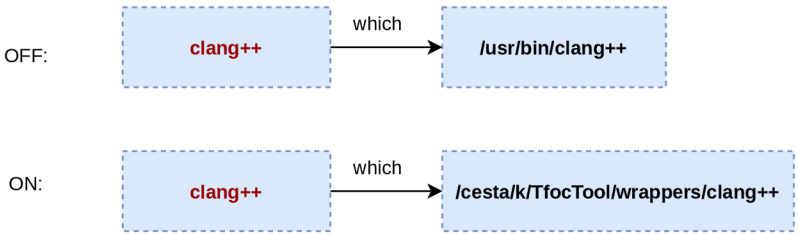
Výpis 6.2: Výpis kódu přidaného do uživatelem zvoleného souboru v rámci deaktivace metody upravující `PATH`

V rámci této metody existují dva režimy. Výše popsaný se nazývá automatický, tím druhým a zároveň výchozím je manuální. Manuální mód v podstatě vůbec nezasahuje do systému, pouze vytvoří vhodně pojmenovaný `wrapper` a uživateli napoví, jak `wrapper` přidat do `PATH` ručně. Příklad této nápovědy je ve výpisu 6.3.

```
Please, add manually wrapper directory
/path/to/TforcTool/tforctool/wrappers
to PATH before original compiler. For example by calling command:
PATH="/path/to/TforcTool/tforctool/wrappers:$PATH"
```

Výpis 6.3: Výpis instruující uživatele, jak přidat `wrapper do PATH`

Velkou výhodou tohoto přístupu je, že není potřeba žádné oprávnění a TforcTool lze spouštět bez oprávnění `root`. Tento výpis se dá také snadno strojově zpracovávat.



Obrázek 6.2: Ilustrace funkcionality metody úpravy `PATH` pomocí nástrojů `which` a `ls` na příkladě překladače Clang++.

Přiložený obrázek 6.2 opět demonstruje strukturu volání a symbolických odkazů při vypnutém a zapnutém nástroji TforcTool požívající popsanou metodu.

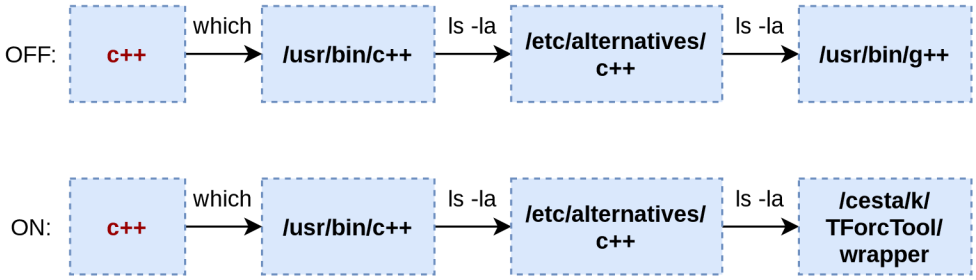
Metoda využívající nástroj Update-alternatives

Poslední z metod využívá Linuxový nástroj update-alternatives (více informací viz 4.3), v rámci kterého vytváří novou alternativu. Všechny následující demonstrace užití příkazů jsou vztaženy k `c++`, alternativě `g++`.

Jak bylo zmíněno ve výše odkazované sekci popisující tento nástroj, vytvořením nové skupiny alternativ můžeme nechtěně odstranit již existující soubor. Tuto metodu lze tedy použít pouze pokud už nějaká alternativa daného generického jména existuje, což je kontrolováno pomocí příkazu `update-alternatives --query compiler_name`.

Před samotným vytvořením alternativy části `wrapper` se nejdřív uloží informace o aktuálním módu skupiny alternativ k danému překladači (manuální, nebo automatický). Nová alternativa se vytvoří příkazem `install`, například takto: `update-alternatives --install /usr/bin/cc cc /cesta/k/TforcTool/tforctool/wrapper.py 10`. Poté se nastaví manuální mód, kterým se zároveň vybere nově vzniklá alternativa: `update-alternatives --set cc /cesta/k/TforcTool/tforctool/wrapper.py`.

Při obnovení se jednoduše zruší vzniklá alternativa pomocí `update-alternatives --remove cc /cesta/k/TforcTool/tforctool/wrapper.py`, případně se znovu obnoví předchozí mód.



Obrázek 6.3: Ilustrace funkcionality metody přidání alternativy pomocí nástrojů `which` a `ls` na příkladě překladače GNU c++.

Přiložený obrázek 6.3 demonstruje strukturu volání a symbolických odkazů při deaktivovaném nástroji a s použitím popsané metody.

Porovnání implementovaných metod

V této podsekci jsou popsaný výhody, nevýhody a použití implementovaných metod. Jejich popis je následně shrnut v tabulce 6.1.

Nejširší využití má metoda nahrazení stávajícího překladače (`replace`). Funguje bez omezení jak při volání příkazu, tak při přímém volání souboru, nejčastěji umístěném v `/usr/bin`. To znamená, že ji lze použít i v kombinaci s nástroji na automatizaci překladu jako například CMake[4]. Navíc nevyžaduje žádný dodatečný zásah po uživateli. Na druhou stranu představuje největší zásah do systému.

Manuální mód metody upravující `PATH` (`path manual`) nepředstavuje žádný zásah do systému a tedy nepotřebuje ani další oprávnění, vyžaduje ale zároveň největší dodatečnou

[4] https://cmake.org/

metoda	bez root	překladac	/usr/bin/překladac	bez interakce uživatele
replace	✗	✓	✓	✓
path auto	✓	✓	✗	✗
path manual	✗	✓	✗	✗
alternative	✗	✓	✓	✓

Tabulka 6.1: Porovnání metod nahrazení překladače.

interakci uživatele. Ten si musí sám ohlídat, že `PATH` obsahuje správné položky. U automatického módu (`path auto`) je tato interakce pouze jednorázová. Uživatel musí ale počítat s tím, že zvolený soubor bude modifikován.

Zlatou střední cestou je použití metody vytvářející novou alternativu (`alternative`). Pokud tedy název překladače, který chceme použít, už nějakou alternativou disponuje. Stejně jako metoda nahrazení umožňuje použití nástrojů na automatizaci překladu.

Porovnávané vlastnosti v tabulce 6.1 jsou:

- zda lze metodu využít bez oprávnění `root`
- zda metoda funguje při volání názvu překladače
- zda metoda funguje při volání souboru umístěném ve složce s příkazy
- zda není po uživateli požadována žádná další činnost, aby mohla dané metoda fungovat

6.2 Zpracování vstupních argumentů určeného pro originální překladač

Argumenty původně určené pro originální překladač jsou zpracovány částí `wrapper`. Druhů argumentů je velké množství. Ty, které nás zajímají, jsou především argumenty ovlivňují pozdější řízení fází překladu nebo instrumentaci, tedy argumenty `-c, -E, -S`.

Do argumentů spadají i názvy překládaných souborů, které nemusí být jen ve formátu zdrojovém (`.cpp`, `.cc`, ...), ale mohou být už částečně přeloženy. Příklad přípon těchto souborů je např. `.o`, nebo `.s`. Ze seznamu souborů je nutné vyčlenit ty, které nejsou přímo určené k překladu, ale slouží jako další vstup, např. soubory uvozené argumentem `-I`. Všechny ostatní poskytnuté argumenty jsou v rámci nějaké části vždy použity.

Existuje i seznam zakázaných argumentům jako `-x` a `-emit-llvm`, které nástroj z různých důvodu nepodporuje. Při jejich použití skončí nástroj s nenulovým kódem a příčinou vzniku problému.

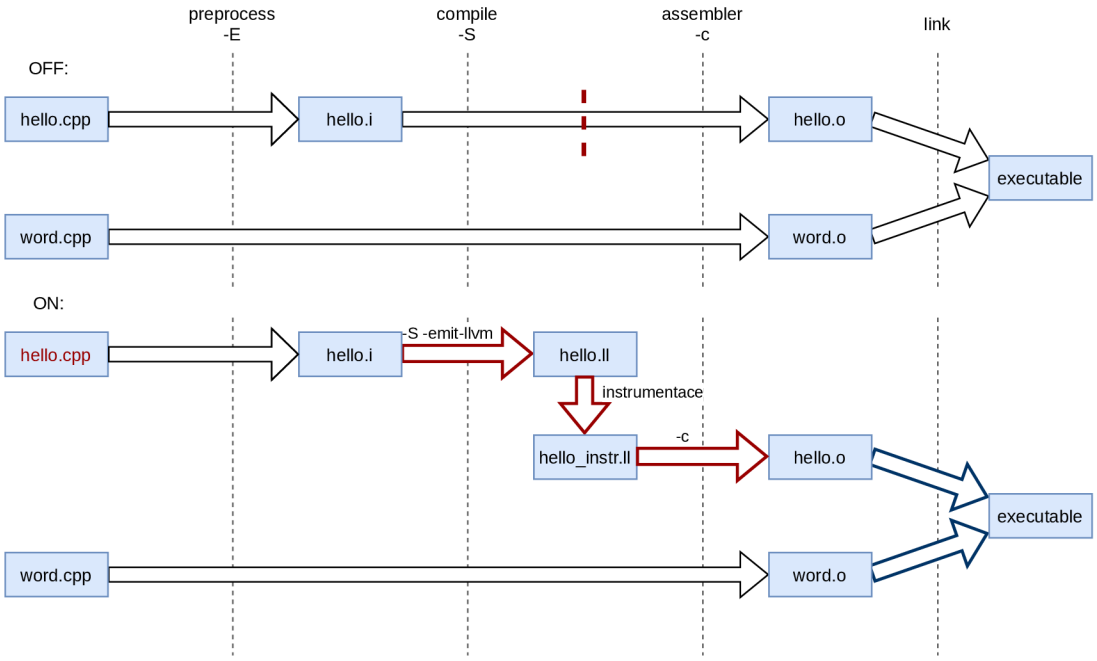
Výše popsané platí pro překlad určený překladačům v rámci skupiny Clang. V případě GNU překladače byly vytvořeny oddělené seznamy se zakázanými, ignorovanými a povolenými argumenty. U těch povolených pak existuje slovník překládající základní a notoricky používané argumenty.

6.3 Řízení překladu a instrumentace

Jak už bylo zmíněno, řízení instrumentace a překladu má na starosti `wrapper`, vykonávání pak Makefile. První dvě podsekce popisují řízení mimo fázi linkování a ve fázi linkování. Dále pak následující sekce podrobněji rozebírá, jak funguje vykonávání s pomocí souboru Makefile. Závěrečná podsekce dává předchozí více do souvislostí a ukazuje, jak se bude nástroj chovat při různých vstupních souborech a fázích překladu.

Následující obrázek 6.4 ilustruje nejdříve překlad bez zapnutého instrumentačního nástroje a následně s jeho aktivací. Na tomto obrázku budou v následujících podsekcích demonstrovány základní principy řízení. Příkazy zajišťující tento překlad jsou vypsány ve výpise 6.4.

V první části, kde je nástroj TforcTool vypnutý vidíme, že první ze souborů `hello.cpp` je nejdříve předzpracován (`preprecess`), poté jsou oba dopřeloženy do objektového souboru a nakonec se spojují pomocí `linkeru`.



Obrázek 6.4: Sekvence překladů nebo instrumentace při deaktivovaném a aktivovaném nástroji TforcTool.

```
$ clang++ -E hello.cpp -o hello.i
$ clang++ -c hello.i -o hello.o
$ clang++ -c word.cpp -o word.o
$ clang++ word.o hello.o -o executable
```

Výpis 6.4: Sekvence příkazů zajišťují překlad ilustrovaný na obrázku 6.4

Ve druhé části vstupuje do hry instrumentace. Ta bude blíže popsána v následujících sekcích.

Řízení instrumentace a překladu mimo fázi linkování

Tforc používá k instrumentaci LLVM nástroj opt, který je schopný ji vykonat pouze nad souborem typu LLVM IR (Tforc je popsán v sekci 4.2, více k LLVM IR viz podkapitola 4.1). Instrumentace přichází tedy v úvahu pouze tehdy, pokud vstupní soubor má překročit fázi `assembler`, v obrázku 6.4 vyznačenou červenou přerušovanou čárou.

Nestačí ale pouze kontrolovat fázi překladu podle specifických argumentů, je potřeba zanalyzovat i typ souboru. Mohlo by se stát, že uživatel zadá příkaz `clang++ hello.o -S`. Díky své robustnosti by Clang pouze vytiskl upozornění, překlad ignoroval a skončil s návratovým kódem 0, což by mohlo nástroji očekávajícímu například daný výstupní soubor způsobit problémy.

Pro spuštění instrumentace je také potřeba nalézt oba uživatelem definované pomocné soubory – konfigurační a obsahující obslužné funkce. Jejich bližší specifikace je popsána v sekci 6.4. Identifikace instrumentace se dá shrnout do následujících tří pravidel:

1. Je překročena fáze překladu `compiler`.
2. Fáze vstupního souboru v kombinaci s požadovanou fází překladu je validní.
3. Je nalezen konfigurační soubor a soubor s obslužnými funkcemi.

V případě našeho příkladu na obrázku vidíme, že `wrapper` kromě instrumentace samotné vykonal ještě dvě dodatečné fáze překladu s využitím přepínačů `-S` a `-c`. První slouží k vytvoření LLVM IR, druhá dopřekládá soubor do požadovaného formátu. Použití obou dodatečných překladů závisí na konkrétním případu.

TforcTool podporuje překlad vícero souborů naráz. Kromě fáze linkování je zpracováván a vyhodnocen každý soubor zvlášť.

Řízení linkování

Mohlo by se zdát, že kromě vyjmenovaných změn je jinak vždy proveden původní překlad. To ale neplatí pro spojovací fázi (`link`), pokud je jejím vstupem instrumentovaný soubor. Tforc ke svému běhu potřebuje ještě další soubory, konkrétně soubor s obslužnými funkcemi a dynamickou a statickou část `TRE` (`Tforc Runtime Engine`), které se musí přidat k výslednému spustitelnému souboru, což znamená úpravu linkování. Pokud žádný ze spojovaných souborů nebyl instrumentován, provede se tato fáze beze změny.

Jestliže probíhá pouze fáze linkování na objektových souborech, musí se zjistit, zda některý ze vstupních souborů byl opravdu instrumentován, a je tedy třeba modifikovat linkování. TforcTool v tomto případě zjišťuje, zda jsou dostupné soubory se specifikacemi a obslužnými funkcemi a zda byla vygenerována dynamická část `TRE`. Pokud ano, přidávají se do linkování i ostatní nezbytné soubory pro běh instrumentovaného programu.

Makefile

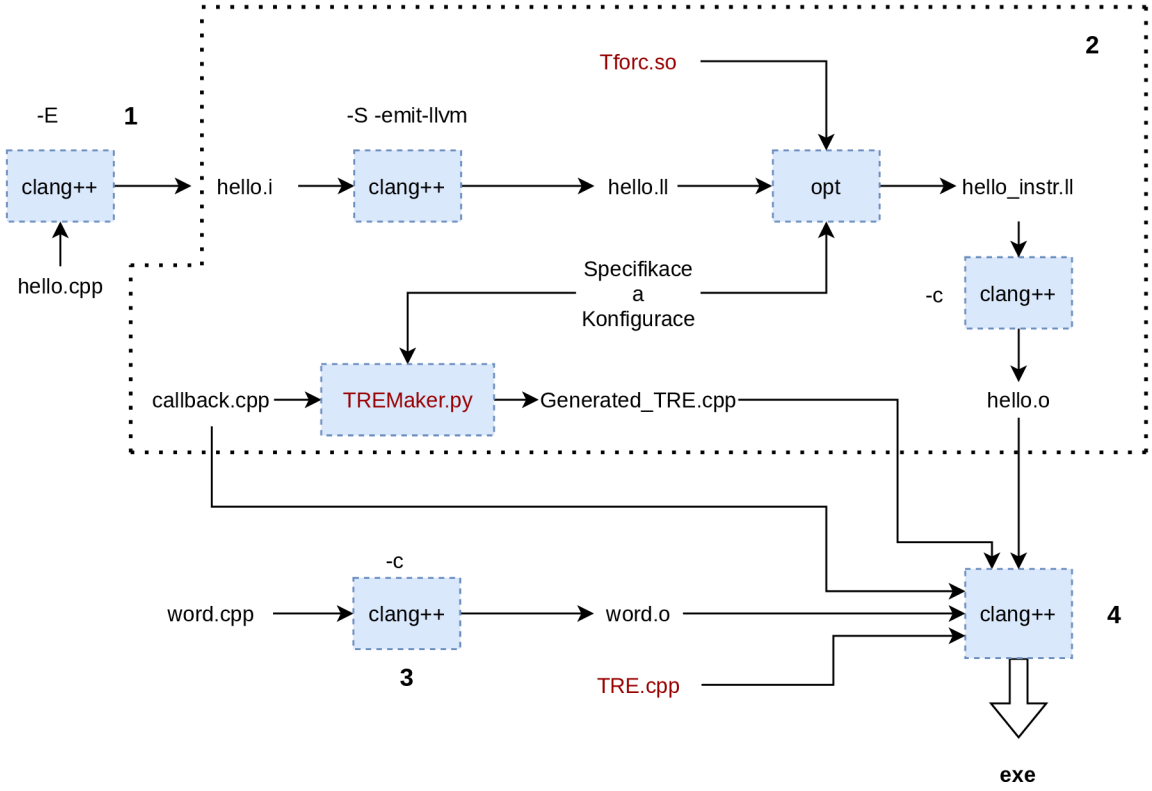
Řízení má na starosti `wrapper`, nicméně samotné vykonávání je prováděno pomocí souboru Makefile, který je touto částí ovládán. V rámci současné podsekce je podrobněji popsáno, co se během jednotlivých fází instrumentace a překladu děje. Opět se jedná o překlad projektu ilustrovaném v minulých částech (viz výpis 6.4).

Souhrnný obrázek 6.5 ukazuje, jak Makefile postupně aplikuje akce požadované částí `wrapper`. Očíslované části odpovídají pořadí příkazů ve výpisu 6.4. U příkazu číslo jedna a tři se jedná o klasický překlad, který by proběhl stejně i bez aktivace TforcToolu.

V rámci příkazu číslo dva, v obrázku vyznačeném přerušovanou čarou, nám do hry vstupuje už zmíněná instrumentace. Po přeložení vstupního souboru do formátu LLVM IR, je vstupní soubor instrumentován s pomocí knihovny Tforc a nástroje opt. Dalším vstupem instrumentace jsou soubory obsahující konfigurace a specifikace instrumentace. V rámci druhé fáze je rovnou vytvořena i dynamická část `TRE` zajišťující spolu se statickým `TRE` nepřímou instrumentaci proměnných. Výstupem původní příkazu má být objektový soubor, proto je třeba provést ještě jednu fázi překladu k zajištění korektního výsledku.

Posledním příkazem se díky souboru Makefile spojují všechny potřebné soubory do spustitelného. Kromě `hello` a `word` se jedná ještě o obě části `TRE` a soubor s obslužnými funkcemi.

Vyznačená knihovna `Tforc.so` je sestavena při prvním spuštění části `wrapper`.



Obrázek 6.5: Podrobné schéma překladu a instrumentace prováděné souborem Makefile. Červené části označují Tforc.

Příklady řízení překladu, instrumentace a linkování

Tato podsekce dává do souvislostí předchozí a ukazuje, jak se bude nástroj chovat při vstupních souborech v různých fázích a různých fázích překladu v případě jednoho souboru, který má být v konečném důsledku instrumentován. Všechny dále popisované varianty jsou zobrazeny v tabulce 6.2.

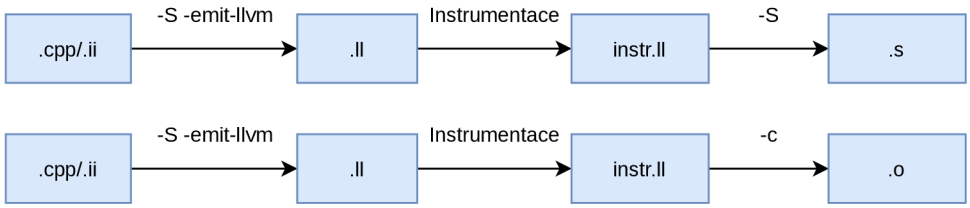
Šedě jsou v tabulce 6.2 vyznačena pole, kde je požadovaná fáze překladu nižší než fáze vstupního souboru. V takovém případě je proveden originální překlad, do kterého není nijak zasaženo.

	-E	-S	-c	x
.cpp	originální překlad	instrumentace	instrumentace	instrumentace a linkování
.ii	špatná fáze	instrumentace	instrumentace	instrumentace a linkování
.s	špatná fáze	špatná fáze	originální překlad	linkování
.o	špatná fáze	špatná fáze	špatná fáze	linkování

Tabulka 6.2: Chování nástroje při různých vstupních souborech a fázích překladu.

Stejně tak je proveden originální překlad v případě kombinací souboru s fází `.cpp -E` a `.s -c`. U první jmenované překlad nedojde k fázi, kde se instrumentuje. U druhé naopak tuto fázi přejdeme.

Modře jsou obarvená pole, kde instrumentace probíhá. Nejedná se samozřejmě jen o instrumentaci. Mimo ní probíhá i překlad do formátu `IR` a následný dopřeklad do konečného požadovaného typu souboru. Detailní schéma je zobrazeno na obrázku 6.6.



Obrázek 6.6: Podrobné schéma překladu a instrumentace souborů ve formátu .cpp/.ii s výstupní fází assembler a objektový soubor.

Žlutou část tabulky zaujímají dvě varianty, kde je vykonáno pouze linkování. V případě dřívější instrumentace je adekvátním způsobem toto linkování modifikováno.

Poslední sekci tvoří varianty, kdy proběhne jak instrumentace, tak linkování. Vše popsané k variantě, kde je linkování a instrumentace se vztahuje i na tyto varianty.

6.4 Uživatelem definované vstupní soubory

Ke správnému běhu jsou potřeba dva uživatelem definované soubory, a to `<nazev_instrumentovaneho_souboru>_CB.cpp`, který obsahuje definice obslužných funkcí, a `<nazev_instrumentovaneho_souboru>.query` s definicemi požadavků na instrumentaci. Pokud nejsou oba nalezeny, neproběhne ani instrumentace, ale pouze nezměněný překlad.

Oba dva jsou implicitně očekávány v korespondujících složkách `tforc_query` a `tforc_callback` nacházející se v kořenovém adresáři projektu, pro který byl TfocTool spuštěn. Obě složky lze změnit pomocí nastavení systémových proměnných `TFORC_QUERY_DIR` a `TFORC_CALLBACK_DIR`.

Obsah souboru s obslužnými funkcemi je identický s tím používaným nástrojem Tforc. Soubor se specifikacemi se skládá ze dvou částí: z hlavičky (výpis 6.5) s nastavením a ze samotných specifikací. Nepovinná hlavička může obsahovat dvě klíčová slova. První z nich, `callBackFunctions`, může měnit implicitní název souboru s obslužnými funkcemi, druhé, `indirectAddressing`, udává, zda se má použít nepřímá adresace, nebo ne. Ve výchozím nastavení je vypnutá.

```
callBackFunctions: my_main_CB.cpp
indirectAddressing: no
```

Výpis 6.5: Příklad hlavičky v souboru se specifikacemi

Pro zápis samotných požadavků na instrumentaci, neboli specifikací, je použit speciální jazyk vyvinutý pro Tforc popsaný v podsekci 4.2.

`Wrapperu` tento posledně jmenovaný soubor rozděluje na dva, a to na konfiguraci vycházející z hlavičky, kam jsou doplněny další povinné údaje, a na specifikaci samotnou, tak aby oba byly přijatelné nástrojem `Tforc`.

6.5 Interní konfigurační soubor

Mezi částmi `tforc_tool` a `wrapper` je potřeba si předávat informace, jako například, který překladač bude použit u daného projektu. Někam se musí ukládat i informace sloužící k obnově volání původního překladače, jako je název použité metody. K tomuto účelu slouží vnitřní konfigurační soubor ve formátu `json`.

Klíče vyskytující se v tomto souboru jsou následující:

- `compiler`: název vybraného překladače, který se bude nahrazovat
- `compiler_family`: typ překladače, Clang, nebo GNU
- `language`: jazyk vybraného překladače, zatím podporován pouze jazyk `C`
- `method`: číslo použité metody
- `directory`: absolutní cesta k projektu, který má být instrumentován
- `indirect_addressing`: výchozí nastavení typu adresování
- `verbose`: úroveň logování
- `orig_clang_path`: cesta k originálnímu Clang překladači, bez ní by nebylo možné v průběhu překladu a instrumentace volat originální překladač

V rámci metody `PATH` se přidává klíč `auto_mode`, který obsahuje informaci, zda se jedná o automatický režim, či nikoliv. Dále také klíč `shell_file`, za nímž následuje cesta k souboru, ve kterém byla změněna `PATH`.

Kapitola 7

Evaluace instrumentačního nástroje TforcTool

Tato kapitola zahrnuje nejdříve popis implementované sady testů v podkapitole 7.1 a v návaznosti na ně nalezené chyby popsané v kapitole 7.2. Následně je v podkapitole 7.3 popsán výkonnostní test provedený na nástroji TforcTool. Poslední část 7.4 popisuje jeho závislosti.

Testy i měření výkonu probíhalo na zařízení s konfigurací:

- Operační systém: Ubuntu 20.04.1 LTS
- Procesor: Intel Core i5-8250U
- Operační paměť: 15.5 GiB
- L1 Cache: 128K KiB
- L2 Cache: 1 MiB
- L3 Cache: 6 MiN

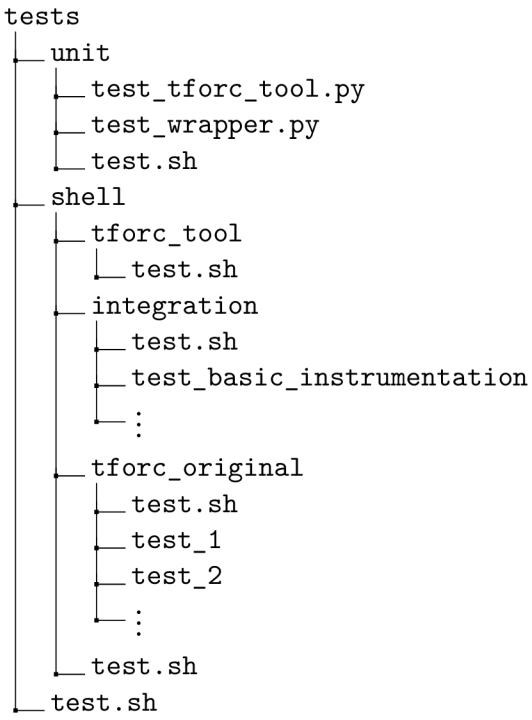
7.1 Testy

V rámci této sekce jsou popsány testy implementované k nástroji TforcTool.

Implementované testy mají strukturu zobrazenou na obrázku 7.1. Podrobný popis testů je popsán v `Readme.md` v kořenovém adresáři. Každá část nebo podčást obsahuje skript `test.sh`, který spouští a vyhodnocuje testy v současné složce anebo podsložkách. Na nejnižší úrovni, kde už je spuštěna pouze jedna sada testů, je vždy možnost spustit izolovaně pouze jeden test pomocí přepínače `-t název_testu`. Všechny testy naráz je možné spustit příkazem `make test` v kořenovém adresáři.

První část `unit` obsahuje jednotkové testy (`unittest`[1]) pro část `wrapper` a `tforc_tool`. Jelikož TforcTool velkou měrou interaguje s okolním prostředím, jsou povětšinou pokryty jen funkce, jenž buď nekomunikují a nezasahují do systému vůbec, nebo se dají tyto činnosti snadno emulovat s použitím tzv. `mock`[2] objektů.

[1] https://docs.python.org/3/library/unittest.html
[2] https://docs.python.org/3/library/unittest.mock.html



Obrázek 7.1: Struktura testů nástroje TforcTool.

Druhá část testů, (`shell`), testuje TforcTool zvenčí pomocí shell scriptů.

Ve složce `tforc_tool` se nachází testy, které zkoumají, zda nástroj při aktivaci opravdu umožňuje volání části `wrapper` místo původního překladače a zda při deaktivaci je opravdu obnoven původní stav. Toto je zkoumáno pro všechny tři metody obejití původního překladače a to s různými parametry.

Testy obsažené ve složce `integration` zkoumají TforcTool jako celek. V jednotlivých složkách (`test_basic_instrumentation`, `test_mutiple_stages`, ...) jsou projekty o několika souborech spolu se skriptem `test.sh`, který spouští instrumentaci, nebo překlad a sleduje výsledek této akce. Testy zahrnují použití nástroje TforcTool buď při instrumentaci nebo jen překladu daného projektu. Mezi testy jsou i všechny varianty fází vstupních souborů a výstupních fází překladu sepsané v tabulce 6.2.

Poslední série testů se nachází ve složce `tforc_original`. Jedná se o testy, které byly původně vytvořeny k nástroji Tforc a následně upraveny tak, aby se daly použít při testování nástroje TforcTool.

7.2 Nalezené chyby

Nedeterministická chyba při instrumentaci a linkování

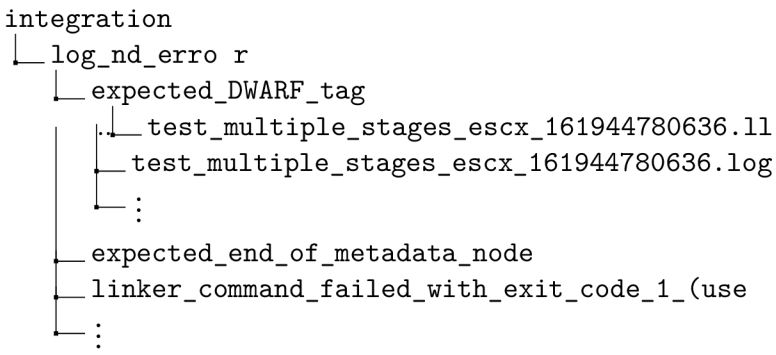
Při testování nástroje jako celku v rámci testů `integration` se objevuje nedeterministická chyba, a to při použití přepínačů `-c` a `-S` a pouze u projektů skládajících se pouze z jednoho souboru. Konkrétně se jedná o pět testů:

- test_multiple_stages_cx
- test_multiple_stages_ecx

- test_multiple_stages_escx
- test_multiple_stages_sx
- test_cmake

Chyba se projevuje z pravidla na dvou místech, a to při dalším překladu už instrumentovaného souboru ve formátu `IR` (.ll), nebo při linkování instrumentovaného souboru s ostatními. V obou případech jsou jak instrumentovaný soubor, tak příkazy použité pro instrumentaci, nebo překlad totožné s případem, kdy se chyba neprojeví.

Funkcionalita testovacího skriptu `test.sh` ve složce `intergration` byla rozšířena o možnost spustit buď vybraný test, nebo sadu testů opakovaně za sebou. Pokud v rámci některého spuštění test selže, je jeho záznam spolu s instrumentovaným souborem uložen do složky `log_nd_error`. Tento prostor je rozdělen na podsložky s názvem generované chyby, jak lze vidět na obrázku 7.2. Oba soubory jsou zkopírovány do příslušné složky a pojmenovány názvem skládajícím se z názvu testu, časové značky a příslušné přípony. Názvy všech nalezených chyb lze nalézt v příloze C.



Obrázek 7.2: Struktura adresáře `log_nd_error`.

Nedeterministická chyba při linkování instrumentovaného souboru

Tento druh chyby vzniká při linkování už instrumentovaného souboru s ostatními. Formát této chyby je zobrazen ve výpise 7.1. Samotnému selhání předchází varování `overriding the module target triple with x86_64-pc-linux-gnu [-Woverride-module]`. Projevuje se pouze u testů `test_multiple_stages_cx` a `test_multiple_stages_escx`. U obou testů bylo zaznamenáno selhání i na druhém místě.

```
/usr/bin/ld: /usr/bin/../lib/gcc/x86_64-linux-gnu/9/../../../
x86_64-linux-gnu/crt1.o: in function `_start':
(.text+0x24): undefined reference to `main'
```

Výpis 7.1: Příklad výpisu při projevu nedeterministické chyby během linkování

Podobné varování `warning: overriding the module target triple with x86_64-unknown-linux-gnu [-Woverride-module]` je zmíněno na stránkách GitHub repozitáře Scala Native v sekci Issues [18]. Jeden uživatele poukazuje, že varování znamená, že není specifikována cílová platforma generovaného kódu, což se dá napravit pomocí `llvm-config --host-target`.

Tímto se eliminovaly všechny nedeterministické chyby související s fází spojování. Přestaly se také projevovat veškeré chyby u testu `test_cmake`, přestože ani jedna z nich nesouvisela s linkováním.

Nedeterministická chyba při následném překladu instrumentovaného souboru

Druhý typ chyby se projevuje při překladu instrumentovaného souboru buď do assembleru, nebo objektového souboru, tedy při použití přepínačů `-c` a `-S`. Při překladu na spustitelný soubor se už neprojevuje. Nejčastější chybové hlášky je `expected_field_label_here` a `unterminated_attribute_group`. Oba druhy chyb se vyskytují u všech výše zmíněných testů kromě `test_cmake`. Po opravení chyby při linkování je procento výskytů chyby zhruba 20 %.

Většina chyb je syntaktického rázu (`unterminated_attribute_group`, `expected_type`), ale najde se tu i sémantická chyba jako `use_of_undefined_metadata_!717`.

Jak už bylo zmíněno výše, obsah instrumentovaného souboru, při kterém se neobjevila chyba je stejný jako ten, kde se chyba projevila. Stejně tak použité příkazy odpovídají těm, které jsou volány v případě, kdy test uspěje.

7.3 Výkon nástroje

V rámci této sekce je popsán způsob měření výkonu nástroje TforcTool v porovnání s nástrojem Tforc a dosažené výsledky měření.

Struktura výkonnostních testů

Měření probíhalo na sadě testů nástroje Tforc a na sadě testů obsažených ve složce `tforc_original`, což jsou testy nástroje Tforc přepsané pro účely TforcTool. Obě sady obsahují 62 testů, přičemž každý z nich obsahuje jeden zdrojový soubor.

Pro účely měření byla rozšířena funkcionalita skriptu `test.sh` ve složce `tforc_original` o přepínač `-p n`, který celou sadu testů vykoná n-krát a vypíše průměrnou dobu jednoho běhu. Samotné měření času probíhá v rámci každého testu zvlášť a je měřen pouze výkon instrumentace či překladu. Doba potřebná k aktivaci a deaktivaci nástroje TforcTool započítána není. Měření se tedy vztahuje pouze na část `wrapper`.

Při měření výkonu nástroje Tforc byly využity nejen originální testy, ale i originální `Makefile`. K němu byl implementován malý skript volající pravidla souboru `Makefile` a zaznamenávající čas. Opět byla měřena pouze doba nutná k instrumentaci nebo překladu. Tento skript je přiložen v příloze D.

V obou případech probíhalo měření pomocí nástroje `date`[3] s přesností na nanosekundy a v obou případech byla každá sada spuštěna 30 krát za sebou.

Výsledky výkonnostních testů

V tabulce 7.1 je přehled dosažených výsledků pro průměrný běh jedné sady. Pro přehled je zde poskytnuto i srovnání s během testů bez instrumentace. Zpomalení v případě použití nástroje TforcTool oproti Tforc je o 8.82 %. Při porovnání obou nástrojů s překladem bez instrumentace zpomaluje Tforc překlad asi 4.42 násobně, TforcTool pak 4.76 násobně.

[3] https://man7.org/linux/man-pages/man1/date.1.html

nástroj	průměrná délka běhu [s]
TforcTool	79.56
Tforc	73.11
bez instrumentace	16.54

Tabulka 7.1: Porovnání výkonu nástroje Tforc a TforcTool.

Při porovnání překladu bez instrumentace a s instrumentací je potřeba brát v potaz, že překlady a instrumentace probíhají na minimálním zdrojovém souboru, a proto je poměr instrumentovaného kódu velmi vysoký.

7.4 Závislosti

Tato podkapitola sumarizuje software potřebný ke správnému běhu nástroje TforcTool.

Pro běh nástroje Tforc je potřeba hlavně sada nástrojů LLVM ve verzi 9, nebo 10. Nižší verzi nejde použít a vyšší bohužel také ne, jelikož mezi verzemi 10 a 11 proběhly zásadní změny znemožňující kompatibilitu, jako například přejmenování hlavičkových souborů[4]. TforcTool navíc vyžaduje překladač Clang, nástroj make a Python3.8, nebo vyšší. Pro testování je potřeba mít nainstalován nástroj bc, a date.

Všechny zmíněné nástroje lze nainstalovat v případě balíčkovacího systému apt příkazem:

```
$ apt get install llvm llv-dev clang make python3 bc
```

Výpis 7.2: Instalace potřebných nástrojů

[4] https://github.com/llvm/llvm-project/commit/2c24051bacd2d0eb7141fc4adb870281aec4e714

Kapitola 8

Závěr

V této práci byl představen návrh a implementace nástroje TforcTool, který poskytuje instrumentaci programů napsaných v jazyce `C++` při překladu, a to konkrétně instrumentaci přístupů do paměti a volání funkcí. TforcTool vychází z již existujícího nástroje Tforc. Přestože se jedná o statickou instrumentaci při překladu, nevyžaduje nástroj úpravu stávající skriptů pro překlad, ani použití předpřipravého souboru Makefile, jak tomu bylo u nástroje Tforc. Tyto skripty zůstávají neměnné. TforcTool také podporuje jakékoliv ze základních argumentů překladače určujících fázi překladu.

Funkcionalita vytvořeného nástroje byla ověřena několika sadami testů, jenž zahrnují i testy vytvořené původně pro nástroj Tforc a testy porovnávající výkon nástroje Tforc a TforcTool ukazující, že zpomalení překladu a instrumentace oproti nástroje Tforc je pouze 8.82 %.

Díky testům byla nalezena nedeterministická chyba, kterou je třeba nejdříve blíže lokalizovat a opravit před dalším vývojem. Oprava chyby byla už nad rámec zadání. TforcTool je aktuálně schopen instrumentovat pouze jeden soubor, a to z důvodu omezení na straně nástroje Tforc, což ponechává prostor pro další rozšíření. Struktura nástroje TforcTool je pro zmíněné rozšíření již připravena. Stejně tak TforcTool počítá s podporou instrumentace dalších jazyků, které lze přeložit s pomocí LLVM překladače do formátu `IR`. Posledním z navrhovaných rozšíření je automatický převod instrumentujících funkcí do dekorované formy (manglink).

V budoucnu se pak počítá s propojením nástroje TforcTool s existujícím nástrojem Spectra, což by umožnilo automatizovanou verifikaci za běhu programu.

Literatura

[1] *Clang - the Clang C, C++, and Objective-C compiler* [online]. [cit. 2021-01-10]. Dostupné z: https://clang.llvm.org/docs/CommandGuide/clang.html.

[2] *Clang: a C language family frontend for LLVM* [online]. [cit. 2021-01-05]. Dostupné z: https://clang.llvm.org/.

[3] *Clang command line argument reference* [online]. LLVM Project [cit. 2021-01-010]. Dostupné z: https://clang.llvm.org/docs/ClangCommandLineReference.html#actions/.

[4] *Clang Compiler User's Manual* [online]. LLVM Project [cit. 2021-01-10]. Dostupné z: https://clang.llvm.org/docs/UsersManual.html.

[5] *Gcov* [online]. [cit. 2020-01-16]. Dostupné z: https://gcc.gnu.org/onlinedocs/gcc/Gcov.html.

[6] *Indirect addressing* [online]. [cit. 2021-01-17]. Dostupné z: https://www.encyclopedia.com/computing/dictionaries-thesauruses-pictures-and-press-releases/indirect-addressing.

[7] *The LLVM Compiler Infrastructure* [online]. [cit. 2021-01-02]. Dostupné z: https://llvm.org/.

[8] *Opt - LLVM optimizer* [online]. [cit. 2021-01-12]. Dostupné z: https://llvm.org/docs/CommandGuide/opt.html.

[9] *Option Summary* [online]. GCC team [cit. 2021-01-10]. Dostupné z: https://gcc.gnu.org/onlinedocs/gcc/Option-Summary.html.

[10] *Pin 3.18 User Guide*. Intel Corporation [cit. 2020-03-20]. Dostupné z: https://software.intel.com/sites/landingpage/pintool/docs/98332/Pin/html/.

[11] *Update-alternatives - maintain symbolic links determining default commands*. Ubuntu Manpage Repository [cit. 2020-02-01]. Dostupné z: http://manpages.ubuntu.com/manpages/trusty/man8/update-alternatives.8.html.

[12] *Update-alternatives(8) - Linux man page*. Die.net [cit. 2020-01-15]. Dostupné z: https://linux.die.net/man/8/update-alternatives.

[13] *Valgrind* [online]. [cit. 2020-02-16]. Dostupné z: https://valgrind.org/.

[14] *Score-P* [online]. 2020. Dostupné z: https://hpc-wiki.info/hpc/Score-P.

[15] *Calling conventions for different C++ compilers and operating systems* [online]. Agner Fog. Public, listopad 2021 [cit. 2020-01-15]. Dostupné z: https://www.agner.org/optimize/calling_conventions.pdf.

[16] Berris, D. M., Veitch, A., Heintze, N., Anderson, E. a Wang, N. *XRay: A Function Call Tracing System* [online]. Duben 2016. Dostupné z: https://storage.googleapis.com/pub-tools-public-publication-data/pdf/45287.pdf.

[17] D'Elia, D. C., Coppa, E., Nicchi, S., Palmaro, F. a Cavallaro, L. SoK: Using Dynamic Binary Instrumentation for Security (And How You May Get Caught Red Handed). In:. Association for Computing Machinery, 2019, s. 15–27. DOI: 10.1145/3321705.3329819. ISBN 9781450367523. Dostupné z: https://doi.org/10.1145/3321705.3329819.

[18] Guillaume, M. *Warning: overriding the module target triple with x86_64-unknown-linux-gnu [-Woverride-module]* [online]. GitHub, 2016 [cit. 2020-04-15]. Dostupné z: https://github.com/scala-native/scala-native/issues/414.

[19] Iskhodzhanov, T., Kleckner, R. a Stepanov, E. Combining compile-time and run-time instrumentation for testing tools. *Programmnye produkty i sistemy*. 2013, č. 3, s. 224–231. Dostupné z: http://swsys.ru/index.php?page=article&id=3593&lang=en.

[20] Kempf, T., Karuri, K. a Gao, L. Software Instrumentation. In: *Wiley Encyclopedia of Computer Science and Engineering*. American Cancer Society, 2008, s. 1–11. DOI: https://doi.org/10.1002/9780470050118.ecse386. ISBN 9780470050118. Dostupné z: https://onlinelibrary.wiley.com/doi/abs/10.1002/9780470050118.ecse386.

[21] Larabel, M. *LLVM Is At Nearly 2.5 Million Lines Of Code* [online]. [cit. 2021-01-02]. Dostupné z: https://www.phoronix.com/scan.php?page=news_item&px=MTU1MzY#:~:text=LLVM%20Is%20At%20Nearly%202.5%20Million%20Lines%20Of%20Code%20%2D%20Phoronix.

[22] Lopes, B. C. a Auler, R. *Getting Started with LLVM Core Libraries*. Packt Publishing, 2014. Community experience distilled. ISBN 978-1-78216-692-4.

[23] Luk, C.-K., Cohn, R., Muth, R., Patil, H., Klauser, A. et al. Pin: Building Customized Program Analysis Tools with Dynamic Instrumentation. In: *Proceedings of the 2005 ACM SIGPLAN Conference on Programming Language Design and Implementation*. New York, NY, USA: Association for Computing Machinery, 2005, s. 190–200. PLDI '05. DOI: 10.1145/1065010.1065034. ISBN 1595930566. Dostupné z: https://doi.org/10.1145/1065010.1065034.

[24] Martínek, D. *Překlad programu* [online]. Září 2011 [cit. 2020-5-1]. Dostupné z: http://www.fit.vutbr.cz/~martinek/clang/gcc.html#compiler.

[25] Myers, C. *The Art of Software Testing*. 3. vyd. Wiley Publishing, 2011. ISBN 1118031962.

[26] Nielson Chris Hankin, F. N.-H. a. *Principles of Program Analysis*. Springer Berlin, 2015. ISBN 978-3-662-03811-6.

[27] Smrčka, A. *Testování a dynamická analýza.* 2020. Texty k přednáškám.

[28] Walls, C. *Software Instrumentation* [online]. Elsevier SciTech Connect., březen 2017 [cit. 2020-02-15]. Dostupné z: http://scitechconnect.elsevier.com/software-instrumentation/.

[29] Ševčík, V. *Rozvoj instruemntace programu při překladu.* Brno, CZ, 2020. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Dostupné z: https://www.vutbr.cz/studenti/zav-prace/detail/129259.

Příloha A

Rozhraní nástroje TforcTool

TforcTool poskytuje rozhraní pro tři základní příkazy: start (aktivace), stop (deaktivace), status (současný stav), clean (vyčištění). U každého z těchto příkazů lze nastavit `verbose` režim pomocí přepínače `-v`.

start

Aktivace nástroje a umožnění instrumentace za překladu. V jednom okamžiku lze mít spuštěn TforcTool pouze pro jeden projekt.

```
tforc_tool.py start [-h] -c compiler [-d directory] [-m method] [-i] [-a]
[-f shell_file]
```

Výpis A.1: Argumenty příkazu start

- `-h`: nápověda
- `-c`: název překladače (clang++, g++, ...)
- `-d`: kořenová složka projektu, výchozí je současný adresář
- `-m`: použitá metoda k obejití původního překladače (replace, alias, path), výchozí je path
- `-i`: použití nepřímé adresace při instrumentaci proměnných
- `-a`: automatický mód u metody nazvané path
- `-f`: pouze při použití metody path v automatickém módu; úplná cesta k souboru, v rámci kterého bude systémová proměnná PATH upravena

stop

Deaktivace nástroje a obnovení původního stavu

```
tforc_tool.py stop [-h]
```

Výpis A.2: Argumenty příkazu stop

status

Výpis aktuálního stavu. Pokud je nástroj aktivován, tak i název překladače a aktuální projekt.

```
tforc_tool.py status [-h]
```

Výpis A.3: Argumenty příkazu status

clean

Odstranění všech dočasných a pomocných souborů, především ve složce tforc_buid.

```
tforc_tool.py clean [-h]
```

Výpis A.4: Argumenty příkazu clean

Příloha B

Příklad použití

Následující použití je demonstrováno na projektu obsaženém ve složce example. Pro zjednodušení je dále použita proměnná `TFORC_TOOL` s cestou k souboru `tforc_tool.py`

```
TFORC_TOOL=/cesta/k/tforc_tool.py
```

Výpis B.1: Nastavení proměnné TFORC_TOOL

Vytvoření potřebných souborů

Struktura projektu určeném k instrumentaci může vypadat následovně:

```
example
├── main.cpp
├── Makefile
├── tforc_callback
│   └── main_CB.cpp
└── tforc_query
    └── main.query
```

Obrázek B.1: Struktura projektu v adresáři example

V tomto případě je instrumentován soubor `main.cpp`. Přestože je možné instrumentovat pouze jeden soubor, projekt samotný se může skládat z vícero souborů. Aby bylo možné instrumentaci vykonat, je třeba vytvořit soubor `<jméno_souboru>.query` ve složce `tforc_query`, kde se nachází informace, co se bude instrumentovat, tedy na co se bude reagovat. Obsah těchto specifikací je popsán v dokumentaci nástroje Tforc nebo v sekci 4.2. A pak také soubor ve složce `tforc_callback`, který obsahuje definice funkcí, které jsou volány na základě reakcí popsaných ve specifikacích (queries). Specifikují tedy JAK se bude reagovat. Jak psát tyto obslužné funkce si můžete přečíst taktéž v dokumentaci Tforc, nebo v sekci 4.2 .

Oba dva druhy souborů jsou očekávány ve složkách pojmenovaných `tforc_callback` a `tforc_query`, které jsou obě v kořenovém adresáři projektu. Obě cesty lze měnit pomocí systémových proměnných `TFORC_CALLBACK_DIR=/moje/cesta` a `TFORC_QUERY_DIR=/moje/cesta`.

Aktivace Instrumentace

Pro základní použití lze nástroj spustit příkazem z výpisu B.2.

```
$TFORC_TOOL -d /cesta/k/projektu/example -c clang++ -m replace
```

Výpis B.2: Příklad spuštění nástroje

kde přepínačem `-d` se specifikuje cesta ke kořenovému adresáři projektu a přepínačem `-c` název překladače volaného při překladu. Přepínač `-m` specifikuje metodu, která zajišťuje možnost instrumentace, ne pouhého překladu. Více k metodám je popsáno v sekci 6.1.

Spuštění překladu a instrumentace

Spusťte překlad tak, jak jste zvyklý. Například s použitím nástroje make, nebo přímo z terminálu.

```
clang++ main.cpp -o executable
make
```

Výpis B.3: Příklad spuštění instrumentace a překladu

Vygenerovaný spustitelný soubor bude už nainstrumentovaný.

Spuštění

```
./executable
```

Výpis B.4: Příklad spuštění instrumentovaného spustitelného souboru

Deaktivace Instrumentace

Pro vypnutí instrumentace a obnovu původního překladače proveďte příkaz

```
$TFORC_TOOL stop
```

Výpis B.5: Příklad zastavení nástroje

Příloha C

Seznam jmen nalezených nedeterministických chyb

Následující seznam udává jména chyb hlášených nástrojem Clang při nedeterministické chybě.

- expected_DWARF_tag
- expected_end_of_metadata_node
- expected_field_label_here
- expected_(_herea
- expected_integer
- expected_metadata_operand
- expected_metadata_type
- expected_string_constant
- expected_type
- expected_value_token
- invalid_debug_info_flag_flag_DIFlagPrototyp
- invalid_DWARF_tag_DW_TAG_imported_declara
- linker_command_failed_with_exit_code_1_(use
- unterminated_attribute_group
- use_of_undefined_metadata_!717

Příloha D

Skript sloužící pro výkonnostní test nástroje Tforc

```
#!/usr/bin/env bash

round_count=1
runtime_total=0

function processArgs() {
  usage="Usage: [-h] [-p n] "
  while getopts ":hp:" opt; do
    case ${opt} in
    h)
      echo "Performance test for Tforc"
      echo $usage
      exit 0
      ;;
    p)
      echo "Performance test for Tforc"
      round_count=${OPTARG}
      ;;
    *)
      echo $usage
      exit 1
      ;;
    esac
  done
}

function run() {
  for ((i = 1; i <= round_count; i++)); do
    printf "\n######## ROUND %s ################\n" ${i}
    make clean
    make prepare_directories
```

```
    start=$(date +%s.%N)
    make prepare_files
    end=$(date +%s.%N)

    runtime=$(echo "$end - $start" | bc -l)
    runtime_total=$(echo "$runtime_total + $runtime" | bc -l)

  done

  avg_time=$(echo "$runtime_total / $round_count" | bc -l)
  LC_NUMERIC=C
  printf "\n\nAverage time for one run: %f s\n" "$avg_time"

}

processArgs "$@"
run
```

Výpis D.1: Zdrojový soubor skriptu pro výkonnostní testování nástroje Tforc

Příloha E

Obsah přiloženého paměťového média

Na SD disku se nachází kompletní projekt obsahující kromě zdrojových souborů (`tforctool`) i příklad (`example`), testy (`tests`) a technickou zprávu v podobě zdrojových souborů (`thesis_latex`) a formátu pdf (`thesis.pdf` a `thesis_print.pdf`). V kořenovém adresáři se také nachází `README.md` s krátkým popisem a ovládáním aplikace a `Makefile` sloužící ke generování programové dokumentace a spuštění přiloženého příkladu.

```
├── example
├── tforctool
│   ├── lib
│   ├── tforc_tool.py
│   ├── wrapper.py
│   ├── Makefile
│   └── wrappers
├── tests
├── README.md
├── Makefile
├── thesis_latex
└── thesis.pdf
```

Obrázek E.1: Obsah přiloženého média