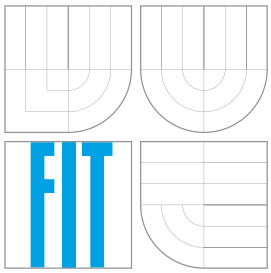
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

JEDNODUCHÉ VÝVOJOVÉ PROSTŘEDÍ PRO C++ NA PLATFORMĚ ANDROID

SIMPLE C++ DEVELOPMENT ENVIRONMENT FOR ANDROID

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

PAVEL REŽŇÁK

VEDOUCÍ PRÁCE
SUPERVISOR

PETR PERINGER, Dr. Ing.

BRNO 2016

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav inteligentních systémů Akademický rok 2015/2016

Zadání bakalářské práce

Řešitel: **Režňák Pavel**
Obor: Informační technologie
Téma: **Jednoduché vývojové prostředí pro C++ na platformě Android**
Simple C++ Development Environment for Android
Kategorie: Softwarové inženýrství

Pokyny:

1. Seznamte se s existujícími nástroji pro vývoj programů v C++ na platformě Android (NDK, c4droid, Terminal-IDE). Seznamte se s vhodnými aplikacemi použitelnými jako komponenty vývojového prostředí (editory, GUI, emulátory terminálu, atd.).
2. Navrhněte a zdokumentujte postup instalace překladače C++ a dalších potřebných nástrojů pro vývoj aplikací přímo na platformě Android.
3. Implementujte potřebné programy, včetně jedné netriviální demonstrační aplikace v C++11 pro testování funkčnosti vytvořeného prostředí. Vytvořené prostředí otestujte také na vhodně zvolené sadě školních příkladů v C/C++.
4. Zhodnoťte dosažené výsledky a navrhněte možná vylepšení.

Literatura:

- Dle pokynů vedoucího.

Pro udělení zápočtu za první semestr je požadováno:

- Splnění prvních dvou bodů zadání.

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Peringer Petr, Dr. Ing.**, UITS FIT VUT
Datum zadání: 1. listopadu 2015
Datum odevzdání: 18. května 2016

L.S.

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav inteligentních systémů
612 66 Brno, Božetěchova 2

doc. Dr. Ing. Petr Hanáček
vedoucí ústavu

Abstrakt

Tato bakalářská práce se zabývá vytvořením prostředí pro překlad aplikací psaných v jazycích C a C++. Vývojové prostředí se skládá z aplikace emulující terminál a balíčků obsahujících open-source překladače a pomocné nástroje, přeložené pro mobilní zařízení s ARM procesorem a operačním systémem Android. Tento proces zahrnuje vygenerování křížového překladače pro systém Android z nástrojů NDK, správné nastavení parametrů během konfigurace, opravy zdrojových souborů těchto open-source překladačů a vytvoření aplikace instalovatelné pod OS Android, která umožní snadné použití těchto nástrojů.

Abstract

This bachelor thesis is about creation of an environment for compiling aplications which has been written in C and C++ languages. This development environment is made of an application emulating terminal window and packages containing open-source compilers and optional tools compiled for mobile devices with ARM processors and Android operating system. This process includes generation of cross-compilers for Android system from NDK tools, correct setting of parameters during configuration, fixes of the source codes of these open-source compilers and creation of an application which is installable under Android OS and which will allow us to easily use these tools.

Klíčová slova

Android, ARM, C, C++, GCC, Clang, Linux

Keywords

Android, ARM, C, C++, GCC, Clang, Linux

Citace

REŽŇÁK, Pavel. *Jednoduché vývojové prostředí pro C++ na platformě Android*. Brno, 2016. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Peringer Petr.

Jednoduché vývojové prostředí pro C++ na platformě Android

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Dr. Ing. Petra Peringera. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Pavel Režňák
18.5.2016

Obsah

Úvod

V poslední době se těší veliké oblibě tzv. chytrá zařízení, která se liší od předchozích zařízení tím, že obsahují operační systém. Mezi populární mobilní operační systémy patří iOS, Windows Phone a Android, ale existují i další (například BlackBerry, Tizen, Firefox OS). V bakalářské práci se budeme zabývat operačním systémem Android, se kterým se můžeme setkat jak u mobilních zařízení, tak i u televizorů, hodinek a miniaturních počítačů. Ve světě existují mobilní telefony a tablety s tímto systémem, které si zachovávají nízkou spotřebu energie a přitom dosahují relativně vysokých výkonů, které se postupem času stále zvyšují nebo dochází ke snížení spotřeby energie. Zároveň dochází ke zvyšování velikosti displejů mobilních telefonů, případně tabletů. Pokud se k těmto zařízením připojí externí klávesnice, mohou se chovat podobně jako počítače. Jejich výhodou je však delší výdrž baterie, která by umožnila vývoj jednodušších aplikací nebo výuku programování bez nutnosti síťového napájení.

Cílem této bakalářské práce je vytvořit prostředí pro překlad aplikací napsaných v jazycích C a C++ přímo v zařízení, na kterém běží operační systém Android. Bude popsán způsob vytvoření nástrojů pro překlad aplikací psaných v těchto jazycích a jejich volitelných rozšíření a bude vytvořena aplikace, která umožní použití těchto nástrojů (emulátor terminálu).

V první kapitole budou definovány důležité termíny a vysvětleny některé pojmy, které je nutné znát o typu procesoru používaném v mobilních zařízení, o operačním systému Android, a o křížovém překladu nástrojů. Následující kapitola bude obsahovat postupy pro překlad jednotlivých nástrojů, které se ve třetí kapitole zabalí do rozšiřujících balíčků a vytvoří se pro ně terminálová aplikace, aby byly tyto nástroje dostupné uživateli. V poslední kapitole otestujeme vytvořené prostředí pomocí několika školních projektů a demonstrační aplikace a provedeme srovnání tohoto prostředí s prostředím na osobním počítači s operačním systémem Linux.

Kapitola 1

Analýza aktuálního stavu

Mobilní zařízení se skládají z relativně výkonného a úsporného procesoru, kombinujícím tyto dvě vlastnosti podle aktuálního vytížení. V bakalářské práci se zaměříme pouze na procesory firmy ARM, na kterých běží operační systém Android. V této kapitole se budeme zabývat prostředím, na kterém bude výsledná aplikace umožňující překlad do binárního kódu spustitelná. Prostředí bude analyzováno jak z pohledu hardwaru, tak i softwaru. do analýzy budou zahrnuty již existující řešení, dostupné v internetovém obchodě **Google Play** [4], který obsahuje aplikace pro operační systém Android.

1.1 ARM procesory

ARM [37] je průmyslový dodavatel, který nabízí širokou škálu mikroprocesorů. Procesory ARM Cortex označují energeticky úspornou architekturu procesorů, která se prosadila zejména v mobilních zařízeních. Pro svůj výkon a rozmanitost je lze použít i do vestavěných systémů a ve firmách. Jedná se o 32 a 64 bitové procesory s RISC [38] architekturou.

Původní procesory neobsahovaly jednotku pro práci s reálnými čísly a bylo nutné tuto činnost emulovat softwarově. V nových procesorech byla tato jednotka implementována a výrazně zrychlila aplikace pracující s reálnými čísly. ARM EABI [9] specifikující způsob komunikace mezi knihovnami a aplikacemi, definuje dvě navzájem nekompatibilní ABI (Application Binary Interface) [1], kdy jedna používá VFP [19] (Vector Floating Point), druhá nikoli. ABI definuje, jakým způsobem má binární kód aplikace komunikovat se systémem při běhu. VFP je způsob architektury procesorů ARM s hardwarovou podporou pro reálná čísla (obsahuje koprocesor). na rozdíl od mnoha dalších architektur, ARM podporuje použití instrukcí pro koprocesor (a jejich výhod) a zároveň si zachovává kompatibilitu i s aplikacemi a knihovnami přeloženými pro základní ABI, kdy koprocesor chybí. Tento způsob má jistá omezení výkonu v porovnání se systémem používající druhou ABI.

V práci se zaměříme na 32 bitové procesory ARM, které jsou postavené na architektuře ARMv7-A [18]. Aktuálně se prodávají nové procesory, které používají ARMv8-A [10] architekturu a jsou zpětně kompatibilní s ARMv7-A. Dalším předpokladem bude podpora instrukcí NEON [36], které zajišťují akceleraci zpracování multimédií a signálů, jako je kódování videa, 2D/3D grafika, hraní, zvuk, zpracování jazyka, obrazu atd.

1.2 Operační systém Android

Android [49] je moderní operační systém (OS) vyvíjený primárně pro dotykové obrazovky mobilních zařízení firmou Google. Je založený na jádře Linux a kromě moderních telefonů a tabletů jej lze najít v hodinkách (Android Wear [8]), televizorech (Android TV [2]) a dalších zařízeních. Při vývoji softwaru byla brána v potaz omezení vycházející ze zařízeních, na která je tento OS určen, jako je výdrž baterie, menší výkon, omezená dostupná paměť, různé rozlišení displeje či použitá čipová sada. Od svého prvního vydání bylo uvedeno několik verzí. Poslední vydaná verze je označena číslem 6 s názvem *Marshmallow* [3].

Jádro Linux

Operační systém Android byl při svém vzniku založený na Linuxu 2.6.4 [61]. Toto jádro systému bylo upraveno firmou Google několika rozšířeními [54] (systém buzení, binder, ashmem, správa napájení, správce paměti při jejím nedostatku, ladící nástroje jádra pro Android, výpis zpráv a další).

Standardní knihovny

Standardní knihovnou jazyka C pro Android je **Bionic** [61]. Tato knihovna je vyvíjena firmou Google z následujících důvodů:

- Licence: snaha o odstranění GPL [44] z uživatelského prostoru.
- Velikost: musí být načtena pro každý proces, a proto musí být malá.
- Rychlost: s omezeným výkonem procesoru je nutné, aby byla rychlá.

Samotný operační systém obsahuje velice minimální podporu pro C++, kterou zajišťuje statická knihovna `libstdc++.a` [13]. Tento problém řeší pomocné C++ knihovny a dodatečné hlavičkové soubory obsažené v nástroji Android NDK [6]. Je-li projekt psaný v jazyce C++ přeložen se závislostí na jednu z nabízených C++ knihoven, a je-li tato knihovna sdílená, bude ji nutné vložit do výsledné aplikace, protože se v systému nenachází. Android NDK nabízí tyto knihovny, které mají příponu `_shared.so` (sdílená) nebo `_static.a` (statická): `gabi++`, `stlport`, `gnustl`, `c++`.

Android Native Development Kit (Android NDK)

Android NDK [6] je sada nástrojů od společnosti Google, která umožňuje s použitím **křížových překladačů** [27] sestavit aplikaci pro specifickou verzi OS Android psanou v jazycích C a C++. Aplikace přeložené těmito překladači nejsou určeny ke spuštění na stroji, na kterém byly přeloženy. Tato varianta se používá například, pokud cílový stroj má nízký výkon nebo malé množství paměti. Typické použití je pro ARM procesory na miniaturních počítačích, jako je Raspberry PI, nebo pro mobilní zařízení. Nástroje ze sady NDK jsou často používány k optimalizaci částí aplikace, kdy se určité části zdrojového kódu přepíšou z jazyku Java do jazyků C nebo C++ a přeloží do dynamické knihovny, která se poté vloží do aplikace a s pomocí *JNI* [29] (Java Native Interface) se volají jednotlivé funkce.

Android NDK podporuje překlad pro Android API verze 3 až 21 (s výjimkou verzí 7 a 20). Od Android verze 5.0 (API verze 21) je nutné překládat binární soubory formou *Position Independent Executable - PIE* [67]. Toto omezení vzniklo v důsledku zvýšení bezpečnosti, kdy je binární soubor a jeho závislosti nahrán na náhodné místo ve svém virtuálním

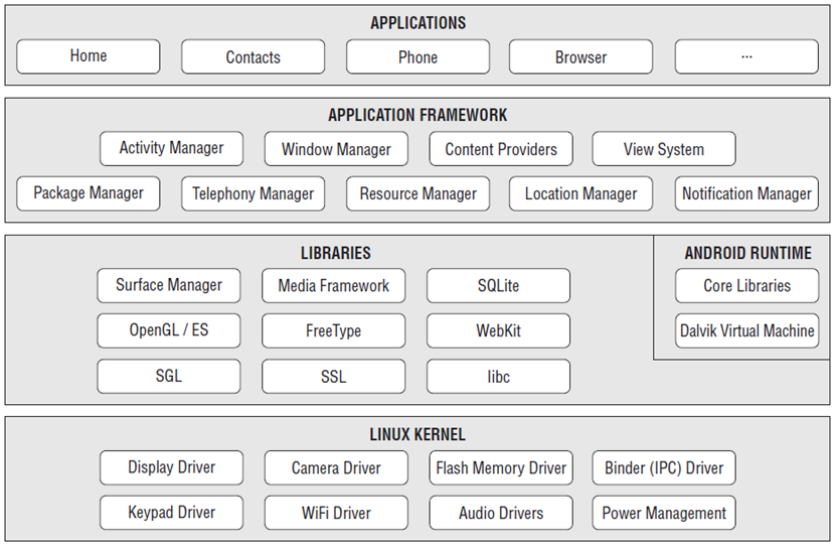
prostoru v paměti. Vzhledem k neúplné knihovně *Bionic* a dalším omezením Android NDK od firmy Google vznikl projekt jménem **CrystaX NDK** [46], který doplňuje některé chybějící funkce standardní knihovny Bionic a také rozšiřuje Android NDK o vlastní příspěvky. Mezi hlavní důvody pro použití CrystaX NDK patří:

- Knihovna libcrystax, která stírá rozdíly mezi různými verzemi OS Android
- Novější verze překladačů (podpora C++11 a C++14)
- Plně funkční standardní C++ knihovna
- Připravené Boost C++ knihovny
- Přidána podpora pro Objective-C a Objective-C++
- Vylepšení a opravy standardní knihovny jazyka C

Tuto sadu nástrojů lze zdarma stáhnout z webových stránek projektu. Po překladu vlastního projektu tímto způsobem je nutné přibalit do výsledné aplikace dynamické knihovny, proti kterým překlad probíhal.

Začlenění práce do struktury OS Android

Všechny aplikace, které budou napsané, přeložené a spuštěné nástroji, o kterých pojednává tato práce, budou mít k dispozici systémové knihovny (zóna označená slovem Libraries na obrázku 1.1), ale výše se v této architektuře nedostanou. Je nutné upozornit, že se nejedná o standardní vývoj aplikace, a autor operačního sytému Android tento způsob nepředpokládal, proto může docházet k problémům. Další důležitá vlastnost tohoto konceptu tvorby aplikací je, že je možné pracovat i s neveřejnou částí API, kterou lze najít na internetu pod názvem Android Open Source Project [7]. Práce s těmito knihovnami a hlavičkovými soubory je často velice obtížná a může vyžadovat odemknuté zařízení (v zařízení existuje tzv. root účet s vyššími právy).



Obrázek 1.1: Architektura OS Android [69]

1.3 Open-source překladače pro C a C++

Mezi dva nejrozšířenější open-source překladače zdrojových kódů napsaných v C/C++ patří **Clang** a **GCC**. Tyto nástroje umožňují kromě překladu i řadu optimalizací a poskytují vývojáři chybová a varovná hlášení.

Během konfigurace překladače jsou často používány termíny **target**, **build** a **host** [27]. Je nutné rozlišit tři základní věci, na kterém stroji probíhá překlad (*build*), na kterém stroji bude nově vytvořený překladač spouštěn (*host*) a pro který stroj je určena výsledná aplikace takto vytvořeného překladače (*target*). Pokud je *host* stejný jako *build*, ale *target* se odlišuje, označíme výsledek tohoto procesu překladu jako **křížový překladač**. Jestli se liší konfigurace ve všech třech parametrech, pak označujeme tento proces jako *Canadian Cross* [27] (křížový překladač, který se přeloží na stroji A, spouští se na zařízení B a produkuje kód pro stroj C).

K identifikaci hardwaru stroje a jeho operačního systému se používá **Target Triplet** [59] (při překladu se generují instrukce volající funkce jádra, nikoli přímo strojové instrukce). Název je složen z trojice *stroj-výrobce-operační systém*. Několik příkladů (výrobce se automaticky doplňuje jako `unknown` pokud není specifikován):

- 64 bitová verze Linuxu na PC: `x86_64-uknown-linux-gnu`
- 32 bitová verze FreeBSD na PC: `i686-uknown-freebsd`
- OS Android s procesorem ARM: `arm-uknown-linux-androideabi`
- Raspberry PI (a další podobné miniaturní počítače) s jádrem Linux s procesorem ARM: `arm-uknown-linux-gnueabi` nebo `arm-uknown-linux-gnueabihf` (`hf` označuje *hard float*)

V našem případě bude probíhat překlad na 64 bitovém operačním systému na jádře Linux architektury x86 za pomocí sady křížových překladačů generovaných z nástrojů NDK (konkrétně CrystaX NDK) do zařízení s operačním systémem Android běžícím na procesoru ARM, tedy:

- `build=x86_64-linux-gnu`
- `target=arm-linux-androideabi`
- `host=arm-linux-androideabi`

Aby konfigurační nástroje mohly najít všechny potřebné programy, hlavičkové soubory a knihovny pro překlad, je nutné mít správně nastavené prostředí (neboli proměnné v prostředí pro překlad musí odkazovat správnými cestami do správných adresářů nebo na správné soubory). Standardně jsou nástroje, které hledají tyto konfigurační skripty, prefixovány, proto například překlad projektu psaném v jazyce C bude probíhat s pomocí nástrojů `arm-linux-androideabi-gcc` a `x86_64-linux-gnu-gcc` (podle nastavení proměnné specifickou hodnotou *Target Triplet*).

Pro přesnější specifikaci je možné k překladu přidat další parametry, které doplňují přesnější označení typu procesoru a dalších parametrů, u ARM specifikujeme práci s desetinnými čísly a podporované instrukční sady.

GNU Compiler Colection (GCC)

Poprvé se tento překladač objevil ve verzi 1.0 v roce 1987 pod jménem GNU C Compiler [45]. V dnešní době umožňuje překlad několika jazyků, jako jsou C, C++, Ada, Fortran, Objective-C, Java, Ada a Go [21]. Základem je řídící program s názvem `gcc`, který zpracovává vstupní argumenty a určuje, jaký jazyk má být použit pro vstupní soubory. Pro každý podporovaný jazyk je k dispozici *front-end*, který převede vstupní soubory do mezikódu v jazyce *GIMPLE* [23]. Tento mezikód založený na tří-adresové reprezentaci je dále zpracováván *back-endem*. Během těchto procesů dochází k optimalizaci kódu, a to na architektuře nezávislé (v jazyce GIMPLE) a na architektuře závislé.

Běžný překlad kompilátoru je zajištěn pomocí nástrojů **Autotools** [42], mezi které patří *Autoconf* a *Automake*. Nástroj Autoconf generuje konfigurační skript `configure` na základě šablony `configure.ac`. Po spuštění skriptu `configure` je prohledán systém, jeho proměnné a je vygenerován soubor `config.status`, který převede vstupní soubory v závislosti na aktuálním systému a parametrech na výstupní (velice často `Makefile.in` na soubor `Makefile`). Zároveň se během konfigurace můžou uvádět dodatečné argumenty sloužící k přizpůsobení následného překladu.

Typické použití těchto nástrojů je následující:

1. Spustí se konfigurační soubor `./configure --arg1 --arg2=XXX ... --argN`
2. Provede se překlad příkazem `make`
3. Následuje většinou instalace přeložených části příkazem `make install`

GCC nabízí obrovské množství přepínačů při konfiguraci, které lze najít v dokumentaci [41]. Mezi nejdůležitější pro tuto práci patří:

`--prefix="cesta"`
: Definuje cestu, která bude sloužit jako předpona pro instalační skript. Standardní instalace binárních souborů se provádí do `/bin`, s prefixem `/userpath` se provede instalace do `/userpath/bin`.

`--host=tt-host --build=tt-build --target=tt-target`
: Definuje Target Triplets pro jednotlivá nastavení. U GCC není nutné specifikovat všechny tyto parametry, protože dokáže dedukovat očekávané Target Triplets s pomocí skriptu `config.guess`.

`--enable-languages=c,c++`
: Specifikuje, pro které jazyky bude tento výsledný překladač určen. Při zakázání některých jazyků se sníží délka doby překladu i výsledná velikost.

`--disable-shared --enable-shared`
: Povolí nebo zakáže generování sdílených knihoven (musí být podporovány na cílovém zařízení). Je možné specifikovat, které knihovny mají být sdílené, aktuálně jsou rozpoznány tyto parametry: `libgcc`, `libstdc++` (ne `libstdc++-v3`), `libffi`, `zlib`, `boehm-gc`, `ada`, `libada`, `libjava`, `libgo`, and `libobjc`.

`--disable-bootstrap`
: Zakáže tzv. *bootstrap*. Jde o 3 fázový překlad, o kterém je možné se více dočíst v dokumentaci [26] (sekce Building a native compiler).

`--disable-nls`
: Zakáže *Native Language Support*, neboli lokalizaci do národního jazyka. Výsledné chybové a varovné zprávy jsou poté ve standardním jazyce (Americká angličtina). Použitím tohoto přepínače se sníží doba nutná k překladu.

`--with-arch=armv7-a`
: Zvolí architekturu vybraného procesoru. Pouze pro ARM procesory.

`--with-float=hard`
: Zvolí způsob práce s desetinnými čísly na ARM procesorech. Mezi možnostmi je `soft` (ABI pro procesory ARM nepoužívající koprocesor pro práci s reálnými čísly), `softfp` (využití koprocesoru a zároveň zachování kompatibility se starší ABI), `hard` (ABI pro ARM s architekturou VFP).

`--with-fpu=neon`
: Zvolí typ instrukcí pro práci s koprocesorem pro reálná čísla u procesorů ARM.

`--with-system-zlib`
: Určí, že se bude používat systémová knihovna zlib místo dodávané s GCC.

`--disable-multilib`
: Zakáže překlad pro podobná zařízení, umožní pouze jeden specifický typ. Sníží dobu překladu i náročnost na místo. Pokud není tato možnost zakázaná, pak například GCC pro 64 bitový systém dokáže generovat aplikace i pro 32 bitový systém.

`--disable-lto`
: Zakáže link-time optimalizace. Opět šetří náročnost na místo výsledného překladače.

`--with-gmp="cesta" --with-mpfr="cesta" --with-mpc="cesta"`
: Nastavení umístění důležitých knihoven pro GCC, bez kterých nelze pokračovat v překladu. Tyto knihovny lze staticky přeložit přímo, pokud se provede tzv. překlad in-tree (zdrojové soubory se zkopírují do kořenového adresáře GCC do složek s názvy GMP, MPFR a MPC).

`--with-stage1-ldflags="flags"`
: Nastaví parametry pro sestavovací program `ld` [43] v první fázi překladu. Tyto parametry jsou použity i když je GCC konfigurováno s přepínačem `--disable-boostrap` a jedná se pouze o jednofázový překlad.

Clang a LLVM

Clang [14] je tzv. *front-end* pro překlad jazyků C, C++, Objective-C, Objective-C++. Jako *back-end* používá **LLVM** [33], a je jeho součástí od verze 2.6 [32]. Projekt LLVM je sadou modulárních a znovupoužitelných překladačů. Slovo LLVM je název celého projektu a nejde o zkratkou utvořenou z prvních písmen několika slov. Zprvu šlo o výzkumný projekt vytvořený v Univerzitě v Illiois, ale od té doby se rozrostl. Clang byl navržen tak, aby byl plnohodnotnou náhradou pro GCC (podporuje většinu běžně používaných přepínačů). Jde o open-source projekt pod licencí *University of Illinois/NCSA License, a permissive free software licence.*

Mezi první velkou firmu, která začala používat LLVM na velkých komerčních projektech, je firma Apple [68]. Nejprve bylo zamýšleno, že front-end podporující jazyky jako je C

a C++ (hlavně Objective-C - podpora firmy Apple) bude GCC, ovšem zdrojové kódy GCC jsou obrovské a je složité pro programátora s nimi pracovat. Dalším argumentem, proč GCC není front-end pro LLVM a tyto jazyky je, že je zaměřeno hlavně na C++ a jazyk Objective-C, nejvíce používaný pro OS od firmy Apple, má nízkou prioritu při vývoji. Třetím argumentem je použitá licence *GPL*, zatímco LLVM má licenci *BSD*, která povoluje použití těchto kódů v softwaru bez dalšího zveřejňování zdrojových kódů.

Při srovnání s GCC [15], Clang je navržen tak, aby bylo jednoduché jej integrovat například do *grafických IDE*, a také, aby zahrnoval mnohem více informací během překladu, pomocí kterých je jednoduší zpětně dohledat chyby ve zdrojových souborech. Tato chybová hlášení obsahují také více informací a jsou jednodušeji zpracovatelná stroji. Clang je nyní součástí i vývojového studia *Microsoft Visual Studio 2015*.

Clang používal pro překlad svých vlastních zdrojových kódů *autotools* (jako je tomu u GCC), ale od verze 3.8.0 ukončil podporu pro překlad těmito nástroji a nyní se spoléhá na **CMake** [16]. CMake je otevřená (software s otevřeným zdrojovým kódem) a multiplatformní sada nástrojů, která slouží k překladu a testování softwaru. Pro proces překladu používá jednoduché a na platformě nezávislé konfigurační soubory, pomocí nichž generuje soubory k překladu pro specifikovanou jinou sadu nástrojů (například *Makefile* [24], *Ninja* [58], *Visual Studio* [66]).

Pomocné nástroje

Pro překladač GCC je nutné před samotným zahájením překladu přeložit nástroje *binutils*, *GMP*, *MPFR*, *MPC* pro cílové zařízení a nastavit cestu, kam byly nainstalovány. GMP, MPFR i MPC lze přeložit přímo v GCC, pokud jsou rozbaleny jejich zdrojové kódy do kořenového adresáře zdrojových kódu GCC do složek se stejným jmény (u verze 5.3.0 pro ARM procesor je s GMP problém, který je řešen v sekci *Překlad GCC*). Clang je závislý pouze na balíčku *binutils*. Mezi volitelné nástroje pak můžeme zařadit *bash*, *cmake*, *git*, *grep*, *gzip*, *less*, *make*, *man*, *nano*, *sed*, *subversion*, *tar*, *unzip*, *vim*, *wget*, *zip* a další.

1.4 Podobná existující vývojová prostředí

Na internetu se již nějakou dobu vyskytují aplikace, které umožňují překlad přímo z mobilního zařízení s OS Android. Největším společným nedostatkem těchto řešení je, že autoři nikde nezveřejnili, jakým způsobem je možné přeložit tyto nástroje do mobilních zařízení s OS Android. V této práci se budeme snažit porozumět přípravě nástrojů potřebných pro překlad aplikací psaných v jazycích C a C++ v přímo v zařízení s OS Android, zdokumentovat postup přípravy nástrojů a opravit chyby, které se objevily během tohoto procesu.

C4droid - C/C++ compiler & IDE

Jako jedna z prvních aplikací pro překlad C a C++ aplikací na serveru obchodu Google Play je C4droid [62]. Jedná se o kvalitní, ale placenou aplikaci (za přibližně 60 Kč), která nabízí překlad přímo na mobilu bez nutnosti internetového připojení, editor zdrojových kódů a několik rozšíření, jako je například SDL2 a Qt. Aktuální verze GCC je 5.3.0. Umožňuje export přeložených aplikací ve formě APK balíku nebo nativního binárního souboru (pouze terminálové aplikace). Hodnocení aplikace se pohybuje na známce 4.7 z 5 a je nainstalována více jak 50 000 uživateli. Pro běžné uživatele je instalace možná pouze na vnitřní uložiště.

Terminal IDE

Terminal IDE [48] je k dispozici zdarma. S o něco horším hodnocením ale s větším počtem stažení oproti C4droid nabízí GCC verzi 4.4.0 a další nástroje, jako je make, Java, ssh, vim, nano, ... Dále nabízí vlastní softwarovou klávesnici, která umožňuje zadávání normálně nedostupných znaků. K dispozici jsou i zdrojové kódy. V popisku aplikace je zdůrazněno, že není kompatibilní s OS Android verzí 5.0 a vyšší.

Termux

Další nástroj, který je dostupný v obchodě Google Play zdarma, je Termux [65]. Má dobré hodnocení od uživatelů (4.7 z 5) a počet stažení (50 000-100 000). Při spuštění neobsahuje nástroje pro překlad, ale umožňuje jejich instalaci příkazem apt install. Nabízí GCC ve verzi 6.1.0 a Clang ve verzi 3.8.0.

CppDroid - C/C++ IDE

CppDroid [55] je aplikace zdarma ke stažení z Google Play se zaměřením na výuku programování v jazycích C a C++ a jejich knihoven. Obsahuje GCC ve verzi 4.8 a několik ukázkových příkladů psaných v C nebo C++. Tato aplikace nabízí doplňováním kódu, diagnostiku v reálném čase, statickou analýzu a více. Při prvním spuštění aplikace potřebuje dalších 150MB na SDK a následně pro návody 190MB vnitřní paměti.

Terminal Emulator for Android

Terminal Emulator [60] je aplikace s otevřeným zdrojovým kódem bez přidané podpory překladu C a C++. Jde o základní terminál, ověřený mnoha uživateli, který bude součástí řešení této práce. Je hodnocen známkou 4.4 a stažen více jak 10 000 000 uživateli.

Kapitola 2

Překlad nástrojů pro systém Android

V této kapitole se zaměříme na vlastní překlad nástrojů, jako je překladač GCC, Clang a jejich závislosti pro OS Android. Nakonec přeložíme i některé volitelné nástroje, které zjednodušují nebo automatizují překlad. Po dokončení této kapitoly bude možné tyto nástroje nahrát do zařízení a otestovat například pomocí ADB [5] (Android Debug Bridge). Jedná se o nástroj příkazové řádky, který umožňuje komunikaci mezi PC a připojeným zařízením přes USB.

2.1 Nastavení prostředí pro vývoj

Překlad těchto nástrojů probíhal na 64 bitovém PC s operačním systémem Linux Mint 17.3 (vycházející z Ubuntu 14.04). Před samotným překladem nástrojů je nutné mít nainstalované následující aplikace: `make`, `cmake`, `gcc`, `g++` (většinu těchto nástrojů nainstalujeme s balíkem `build-essential`) a staženy NDK (v našem případě CrystaX NDK i Android NDK). na disku budeme potřebovat přibližně 8GB na rozbalení CrystaX NDK, 5GB pro Android NDK a 500MB pro přípravu křížového překladače [39], který bude vygenerován z CrystaX NDK. Zdrojové kódy GCC zaberou přibližně 800MB a pro překlad je vhodné mít minimálně 1GB volného místa. Zdrojové soubory pro Clang zabírají po rozbalení přibližně 250MB (i s LLVM), a pro překlad je nutné si připravit přibližně 300MB místa. Výsledný soubor obsahující GCC nebo Clang zabalený do jednoho souboru bude mít přibližně 100MB.

Adresářová struktura projektu

Kořenový adresář projektu obsahuje základní skripty a je strukturován do adresářů `build`, `scripts`, `src`, `toolchains` a `install`.

`clean.sh`
: Uvede projekt do původního stavu tím, že odstraní soubory a složky, které doprovází tvorbu nástrojů. V kořenovém adresáři ponechá vytvořené rozšiřující balíčky.

`configure.sh`
: Skript se základním nastavením. Zde je nutné nastavit cestu k NDK a je možné upravit další parametry, jako je instalační složka pro vytvořené nástroje, id aplikace,

pro kterou se bude překládat, verze a typ křížového překladače pro vygenerování z NDK.

`make-module.sh`
: Základní skript pro stažení, rozbalení, překlad a instalaci modulu. Jména modulů jsou uvedena ve složce scripts a toto jméno se předává jako první parametr při spuštění skriptu. Po úspěšném provedení skriptu je modul nainstalován do složky install (lze změnit v configure.sh).

`make-package.sh`
: Zabalení nainstalovaných modulů do rozšiřujícího balíčku. Operační systém Android nabízí formát OBB [30] (Opaque Binary Blob), který lze připojit do systému a přistupovat k němu přímo. Tento formát může být zašifrovaný pomocí hesla a je vázaný právě na jednu aplikaci. Před zabalením souborů do tohoto formátu se na každý soubor ve složce `bin` a na každou sdílenou knihovnu ve složce `lib` spustí program `chrpath` s parametrem `-d`, který odebere proměnné RPATH a RUNPATH z těchto souborů. Pokud binární soubor obsahuje tyto proměnné, zobrazuje se od verze Android 5.0 upozornění `WARNING: linker: unused DT entry: type 0x1d`. Aby vzniklé soubory zabíraly co nejméně místa na disku, je před zabalením odstraněna složka `share` a na binární soubory a sdílené knihovny se spustí nástroj `strip` s parametrem `--strip-all` (na spustitelné soubory) a `--strip-debug` (na knihovny) pro odstranění nepotřebných informací. Velikost binárních souborů bez těchto informací se může značně lišit, program `cc1`, který je součástí GCC, zmenšil svoji velikost z přibližně 100 MB na 16 MB.

`make-toolchain.sh`
: Skript pro vygenerování křížového překladače z NDK. V souboru configure.sh je možné nastavit cestu k NDK, cestu, kam se překladač vygeneruje i verzi překladače.

`build/`
: Obsahuje přeložené binární soubory tříděné do složek `modul-verze`.

`scripts/`
: Obsahuje připravené skripty a opravné soubory pro jednotlivé moduly, pokud modul nepotřebuje dodatečné soubory, je specifikován jedním souborem `modul.sh` (modul = jméno, které používá skript pro překlad k nalezení tohoto souboru). Pokud modul obsahuje opravné soubory, je nutné vytvořit složku s názvem modulu a do ní umístit `modul.sh` pro překlad a soubory s koncovkou `.patch` pro automatickou aplikaci těchto patch souborů na zdrojové soubory.

`src/`
: Do této složky jsou staženy potřebné soubory z internetu, které se rozbalí do předem známého umístění. Jména složek se zdrojovými soubory jsou definovány ve stylu `modul-verze`.

`toolchains/`
: Složka obsahující vygenerovaný křížový překladač z předem definovaného NDK (Android nebo CrystaX), který je specifický pro verzi Android API. Uvnitř nalezneme důležité soubory pro křížový překlad, jako jsou hlavičkové soubory, knihovny, a další soubory pro danou verzi Android API. Obsahuje také GCC překladač (nebo Clang), který je předpřipravený pro křížový překlad.

`install/`
: Tato složka bude obsahovat nainstalované moduly. Bude uváděna jako parametr `--prefix` při konfiguraci modulů.

2.2 Překlad GCC

Překlad GCC není triviální záležitostí, obsahuje veliké množství přepínačů během konfigurace a trvá relativně dlouho. V této sekci bude shrnuto, jaké přepínače použít při sestavování tohoto programu s pomocí křížového překladače, jak vhodně nastavit prostředí, aby překlad proběhl bez problémů a našel potřebné soubory. Seznámíme se se závislostmi, bez kterých nelze GCC přeložit, a s problémy, které se během překladu objevily.

Prerekvizity pro GCC

Před samotným překladem je nutné mít stažené zdrojové soubory pro *MPC* (1.0.3), *MPFR* (3.1.3), *GMP* (6.1.0), *binutils* (2.26) a *GCC* (4.9.3 / 5.3.0). O stažení těchto balíků je postaráno s pomocí nástroje `wget`, který získá *URL* adresu pro stažení z nastavení modulu (soubor `scripts/modul.sh` nebo `scripts/modul/modul.sh`). Velice důležité je správně nastavit prostředí, ve kterém bude překlad probíhat (např. proměnná `PATH` musí odkazovat do `toolchains/křížový_překladač/bin`, ve kterém se nacházejí programy prefixované pro OS Android `arm-linux-androideabi-*`). O základní nastavení prostředí se stará soubor `make-module.sh`.

Samotný překlad má vždy velice podobný průběh, provede se `configure && make && make install`. Skript `configure` se spouští s předem definovanými parametry stejnými pro všechny nástroje, může být rozšířen o další parametry, které jsou specifické k určitému modulu. Pro automatizaci tohoto procesu je ve skriptu `make-module.sh` vše předem připraveno, pokud žádá modul o další parametry, nastaví proměnnou `PKG_EXTRA_CONFIGURE`, stejně tak může modul přidat parametry k překladu s pomocí proměnné `PKG_EXTRA_MAKE`.

Konfigurační parametry

`--prefix="install-dir"`
: Parametr prefix definuje cestu, která bude sloužit jako předpona proti standardní instalaci. Tato cesta je nastavena v souboru `configure.sh` a ve skriptech se používá proměnná z tohoto souboru.

`--host=arm-linux-androideabi`
: Přeložené aplikace budou spustitelné v prostředí běžícím na procesorech ARM s OS Android.

`--disable-rpath`
: Výsledná instalace nebude obsahovat cesty k systémovým knihovnám. Bude zapotřebí nastavit proměnné, jako je LD_LIBRARY_PATH pro jejich vyhledání.

Následující parametry byly již vysvětleny v sekci *Open-source překladače pro C a C++*:

`--disable-static`
`--enable-shared`
`--disable-nls`

`--with-arch=armv7-a`
`--with-fpu=neon`
`--with-float=hard`

Proměnné prostředí

`PATH=$PATH:toolchains/křížový_překladač/bin`
: Nastavení cesty k binárním souborům křížového překladače, který byl vygenerován z NDK. Tyto binární soubory mají předponu arm-linux-androideabi- (například arm-linux-androideabi-gcc pro program gcc). Nastavení této cesty je nutné, aby se během konfigurace našly všechny potřebné soubory.

`LDFLAGS`
: Parametry pro sestavovací program křížového překladače (arm-linux-androideabi-ld), budeme potřebovat:

  `-pie`
  : Sestavení programu bude probíhat formou Position Independent Executable. Tento formát je vyžadován od Android verze 5.0.

  `-march=armv7-a`
  : Výsledná aplikace bude sestavena pro architekturu ARMv7-A.

  `-Wl,-no-warn-mismatch`
  : Potlačí varování o míchání dvou nekompatibilních ABI v přeložených objektových souborech.

  `-lm_hard`
  : Sestavení bude probíhat se statickou matematickou knihovnou, která využívá ABI s koprocesorem.

`CFLAGS a CXXFLAGS`
: Parametry pro překlad křížovým překladačem. CFLAGS ovlivňuje překlad zdrojových souborů psaných v jazyce C (arm-linux-androideabi-gcc, arm-linux-androideabi-clang) a CXXGLAGS je pro překlad zdrojových souborů psaných v jazyce C++ (arm-linux-androideabi-g++, arm-linux-androideabi-clang++).

  `-fPIE`
  : Překlad bude probíhat formou Position Independent Executable, tato metoda je nutná pro spuštění aplikace v operačním systému Android 5.0 a vyšší.

  `-march=armv7-a`
  : Překládat se bude pro architekturu ARMv7-A.

  `-mfpu=neon`
  : Během překladu se budou generovat instrukce v instrukční sadě NEON.

  `-mhard-float`
  : Zvolí se ABI, která podporuje použití koprocesoru pro práci s reálnými čísly. Je možné použít parametr `-mfloat-abi=hard`. Konfigurace umožňuje specifikovat 3 možnosti pro práci s reálnými čísly: hard, soft a softfp (kompatibilní se soft, ale může používat koprocesor pro reálná čísla, pomalejší než hard).

  `-D_NDK_MATH_NO_SOFTFP=1`
  : Definice převzatá z NDK Programmer's Guide [35].

`-Os`
: Výsledný binární soubor bude optimalizován pro rychlost, pokud to nenavýší jeho velikost.

Jména programů
: Proměnné prostředí mohou ovlivnit skript, který provádí konfiguraci (configure). Při křížovém překladu je vhodné nastavit tyto proměnné tak, aby odkazovaly na jména binárních souborů, které chceme při tomto překladu použít [47]. Nastavíme tedy:

```
AR=arm-linux-androideabi-ar
AS=arm-linux-androideabi-as
CC=arm-linux-androideabi-gcc
CPP=arm-linux-androideabi-cpp
CXX=arm-linux-androideabi-g++
LD=arm-linux-androideabi-ld
OBJDUMP=arm-linux-androideabi-objdump
RANLIB=arm-linux-androideabi-ranlib
READELF=arm-linux-androideabi-readelf
STRIP=arm-linux-androideabi-strip
```

Překlad GNU Binutils

GNU Binutils [51] je sada binárních nástrojů. Mezi hlavní patří `ld` (*GNU Linker*) a `as` (*GNU Assembler*), ale obsahuje i další nástroje. Samotný překlad upravují 3 patch soubory:

`ld/configure.tgt`
: Soubor upravuje cesty ke standardním knihovnám systému. V OS Android se systémové knihovny nachází v umístění /system/lib.

`ld/emultempl/elf32.em`
: Oprava chyby unsupported flags DT_FLAGS_1=0x8000000'.

`ld/ldmain.c`
: Zakáže výpis varování o tom, že všechny funkce nepoužívají VFP registry. Není nutné používat parametr `-Wl,-no-warn-mismatch` během sestavování aplikace v zařízení s OS Android při použití programu `ld`.

Konfigurace potřebuje přidat k již uvedeným parametrům ještě `--disable-werror`, aby bylo možné aplikaci úspěšně přeložit. Jinak probíhá překlad s parametrem `-Werror`, který označí všechna varování jako chyby během překladu. Při překladu je použit parametr `tooldir="install-dir"` [31]. Překlad modulu binutils provedeme v projektu příkazem `make-module.sh binutils`.

Překlad GMP, MPFR a MPC

Sadu **GMP** [63] překládáme jako první kvůli závislostem. Jedná se o volně dostupnou knihovnu, která se stará o výpočty s libovolnou přesností. Provádí operace nad celými

a racionálními čísly i čísly s plovoucí desetinou čárkou a našla si uplatnění v kryptografii. Konfiguraci a překlad provedeme bez přidání dalších parametrů.

Poté přeložíme **MPFR** [25]. MPFR je knihovna jazyka C pro vícenásobnou přesnost a správné zaokrouhlování při práci s čísly s plovoucí desetinou čárkou. Jelikož je tato knihovna postavena na knihovně GMP, konfigurace vyžaduje jeden parametr navíc (`--with-gmp="cesta-ke-gmp"`), který definuje cestu k instalaci předchozí knihovny přeložené pro OS Android.

Nakonec přeložíme **MPC** [34]. Jedná se o knihovnu jazyka C pro práci s komplexními čísly s libovolně vysokou přesností a správným zaokrouhlením výsledku. Tato knihovna je postavena na knihovně MPFR a proto ji vyžaduje předem nainstalovanou. Během konfigurace se přidají 2 parametry, `--with-gmp="cesta-ke-gmp"` a `--with-mpfr="cesta-k-mpfr"`.

Výsledné knihovny se po instalaci nainstalují do složky `lib`, a to s příponou `.so`, pokud jsme překládali sdílené knihovny, nebo s příponou `.a` pro statické knihovny. Mezi vzniklými soubory jsou i soubory s koncovkou `.la`, které obsahují textový popis důležitý pro sestavení aplikace s konkrétní knihovnou pro nástroj *GNU Libtool* [56].

Tuto sadu tří matematických knihoven můžeme vložit i přímo do zdrojových souborů GCC a provést tzv. překlad *in-tree*, kdy konfigurace GCC zajistí i konfiguraci těchto modulů. Výsledné knihovny se staticky přeloží přímo do GCC. Zde je nutná jedna úprava pro GMP, kdy po konfiguraci celého GCC je nutné upravit soubor `Makefile` například příkazem (jedná se o chybu GCC):

```
sed -i 's/none-/arm-/' Makefile
```

Dojde k úpravě dvou řádků, které změní typ procesoru z neznámý na ARM. Bez této opravy se během překladu vyskytuje chyba `__gmpn_invert_limb`.

Překlad GCC 4.9.3

Konfigurace GCC ve verzi 4.9.3 obsahuje několik přidaných přepínačů k základní sadě a je nutné použít několik opravných souborů, které upraví zdrojové soubory.

Konfigurace: Většina parametrů byla popsána v sekci Open-source překladače pro C a C++ a proto se zde nebudeme opakovat. Pokud jsme překládali matematické knihovny (GMP, MPFR a MPC) samostatně, je nutné použít tyto parametry a správně nastavit cesty k nim:

```
--with-gmp="cesta k nainstalované knihovně GMP"

--with-mpfr="cesta k nainstalované knihovně MPFR"

--with-mpc="cesta k nainstalované knihovně MPC"
```

Další parametry důležité při konfiguraci jsou tyto:

```
--enable-languages=c,c++

--with-system-zlib

--disable-multilib

--disable-lto
```

Prostředí: Tato úprava zahrnuje přepis proměnných z původního jména (např. CFLAGS) pro cíl (CFLAGS_FOR_TARGET). K předchozím proměnným (CFLAGS, CXXFLAGS, LDFLAGS, AR, AS, CC, ...) přidáme příponu _FOR_TARGET a odebereme původní proměnné z prostředí příkazem unset. Například:

```
export CFLAGS_FOR_TARGET="$CFLAGS"

export CC_FOR_TARGET="$CC"

unset CFLAGS

unset CC
```

Patch soubory: GCC ve verzi 4.9.3 potřebuje upravit 6 zdrojových souborů.

`gcc/Makefile.in`

`gcc/double-int.h`
: Úpravy pro *gengtype*.

`libcpp/files.c`

`libcpp/macro.c`
: Oprava jednoduchého implicitního přetypování, které se během křížového překladu muselo explicitně povolit.

`libiberty/configure`

`libiberty/configure.ac`
: Na OS Android je definovaná funkce getpagesize v souboru unistd.h jako static inline. Během konfigurace však nebyla nalezena a musí být nastavena manuálně.

Po konfiguraci nebudeme překládat celou sadu GCC (nebudeme překládat například standardní knihovny jazyka C ani C++), ale jen nástroj **GCC** a k němu důležitou knihovnu **LIBGCC**. Překlad neproběhne příkazem `make` bez parametrů, ale dvěma příkazy `make all-gcc` a `make all-target-libgcc`. Standardní knihovny již máme přeložené v zařízení nebo v `křížový_překladač/arm-linux-androideabi/lib` a pro hard float ještě dále ve složce `armv7-a/hard`.

Po překladu je důležité do složky, do které se provedla instalace, zkopírovat hlavičkové soubory, knihovny a soubory s příponou `.o` (`crtbegin_dynamic.o`, `crtbegin_so.o`, `crtbegin_static.o`, `crtend_android.o`, `crtend_so.o`), které se nacházejí v Android NDK v adresáři `platforms/android-XX/arch-arm/usr` (XX je verze Android API). Nástroj CrystaX NDK obsahuje upravené verze hlavičkových souborů i knihoven a při použití těchto souborů v zařízení docházelo k chybám, kdy aplikace byly po spuštění ukončeny chybou *Segmentation Fault*. Toto chování bylo objeveno během testování u aplikací `fold` a `wordcount-dynamic`. Hlavičkové soubory se nakopírují do složky `include`, knihovny a další soubory patří do složky `lib`. Aby bylo možné využít výhod, plynoucích z překladu s VFP, je nutné zkopírovat i statickou matematickou knihovnu `libm_hard.a`.

Dále je nutné přidat podporu pro aplikace psané v jazyce C++, která se nenachází v zařízeních ale v sadě NDK. Zkopírujeme hlavičkové soubory, které se nachází v adresáři v Android NDK (`sources/cxx-stl/gnu-libstdc++/4.9/include`) a nezapomeneme

na hlavičkové soubory pro naši verzi, jedná se o soubory `.h` ve složce bits (ARMv7-A a hard float) `../libs/armeabi-v7a-hard/include/bits/`.

Na závěr zkopírujeme standardní knihovnu C++ (konkrétně GNU C++), která se nachází ve složce `křížový_překladač/arm-linux-androideabi/lib/armv7-a/hard` (pokud hledáme knihovny přeložené pro architekturu VFP) se jménem `libgnustl_shared.so`, do složky `install/lib`. Během překladu C++ zdrojových souborů je nutné tuto knihovnu registrovat u programu pro sestavení výsledného binárního souboru (ld) příkazem `-lgnustl_shared`.

Překlad GCC 5.3.0

Překlad novější verze probíhá téměř totožně, není ovšem zapotřebí tolik souborů pro opravu zdrojových kódů. Opravné soubory pro `libcpp/files.c` a `libcpp/macro.c` zůstávají. Nový soubor obsahující opravu je pro hlavičkový soubor `gcc/system.h`, který odebere deklarace funkcí `fopen_unlocked`, `putc_unlocked`, `getc_unlocked` a dalších s příponou `_unlocked`, protože křížový překladač nemůže najít jejich implementaci.

Překlad GCC 6.1.0

Dne 27. 4. 2016 byla vydána nová verze GCC [20]. Tuto verzi bylo možné přeložit stejně jako verzi 5.3.0.

2.3 Překlad Clang

V této sekci si ukážeme, jak je možné přeložit sadu nástrojů *Clang*, *LLVM* a *compiler-rt knihovny* pro OS Android. Fáze překladu jsou podobné jako u GCC, největším rozdílem je použití nástroje CMake pro vygenerování skriptů pro překlad (namísto autotools). V našem případě použijeme pro sestavení nástroj **Ninja** [58]. Jedná se o malý nástroj se zaměřením na rychlost. Spoléhá na vygenerované skripty (v tomto případě pro něj generuje tyto skripty CMake) a snaží se pracovat co nejrychleji. Skripty pro tento nástroj jsou sice čitelné pro člověka, ale není pohodlné je psát. Jeho aktuální verze je 1.6.0 a je nutné jej mít předem nainstalovaný v hostitelském PC.

Prerekvizity pro překlad

Před samotným překladem je nutné mít stažené zdrojové soubory pro *binutils*, *LLVM*, *Clang* a *compiler-rt*. Stažení, konfigurace, překlad a instalace balíku binutils je již zmíněna v sekci Překlad GCC. Zdrojové soubory pro LLVM, Clang a comiler-rt musí být rozbaleny do předem dané adresářové struktury [22]. Kořenový adresář obsahuje zdrojové soubory LLVM, Clang se rozbalí do podsložky `tools/clang` a compiler-rt do podsložky `projects/compiler-rt`. Opět budeme muset správně nastavit proměnné prostředí.

Samotný překlad je tentokrát závislý na systému CMake a Ninja. Pro konfiguraci se používá příkaz `cmake -G Ninja "cesta ke zdrojovým souborům" parametry`. Parametr `-G` může nabývat těchto hodnot: `Unix Makefiles` pro generování tradičních *Makefile* souborů pro Unix, `Visual Studio` pro *Microsoft Visual Studio*, `Xcode` pro *Apple Xcode*. Další validní hodnoty pro tento přepínač lze najít v dokumentaci [50].

Po konfiguraci probíhá fáze překladu, která se spouští příkazem `ninja`. Po skončení překladu je možné přeložené soubory nainstalovat příkazem `ninja install`.

Konfigurační parametry

`-DCMAKE_AR="which arm-linux-androideabi-ar"`
: Nastavíme cestu k programu `arm-linux-androideabi-ar`. Program which [52] je Unixový nástroj, který najde a poté vypíše cestu k hledanému programu. Prohledává při tom adresáře definované v proměnné PATH.

`-DCMAKE_BUILD_TYPE=MinSizeRel`
: Nastavíme typ překladu na *release* verzi se snahou o minimální velikost, mezi další parametry patří `Debug` a `Release` pro sestavení s ladícími informacemi nebo bez nich. Překlad s volbou `Debug` je vysoce časově i paměťově náročný.

`-DCMAKE_CROSSCOMPILING=True`
: Nastavíme typ překladu jako křížový.

`-DCMAKE_CXX_FLAGS="$CXXFLAGS -lgnustl_shared"`
: Specifikujeme dodatečné argumenty pro překlad C++ kódů, jako je architektura (`march`), typ koprocesoru pro práci s desetinnými čísly (`mfpu`), typ ABI (`hard`) a další.

`-DCMAKE_INSTALL_PREFIX="install-dir"`
: Cesta ke složce, do které proběhne výsledná instalace.

`-DCMAKE_LINKER="which arm-linux-androideabi-ld"`
: Nastavíme cestu k sestavovacímu programu.

`-DCMAKE_RANLIB="which arm-linux-androideabi-ranlib"`
: Nastavíme cestu pro program `ranlib`.

`-DCMAKE_SYSTEM_NAME=Linux`
: Název OS pro který bude probíhat překlad (*target*).

`-DLLVM_TABLEGEN="host/bin/llvm-tblgen"`
: Cesta k předem přeloženému programu `llvm-tblgen`. Tento program je spouštěn na hostujícím počítači a proto je nutné jej přeložit před samotným překladem nástroje Clang.

`-DLLVM_DEFAULT_TARGET_TRIPLE="armv7a-linux-androideabihf"`
: Definice cílového zařízení. Je možné specifikovat přímo rodinu procesoru a práci s koprocesorem pro desetinná čísla.

`-DLLVM_TARGET_ARCH=ARM`
: Definice cílové architektury.

`-DLLVM_TARGETS_TO_BUILD=ARM`
: Nastavení cílů, pro které takto vytvořený překladač bude schopen generovat aplikace. Pro naše řešení postačí překlad pro ARM procesory, ušetříme místo i čas překladu.

`-DLLVM_ENABLE_PIC=ON`
: Přidá parametr `-fPIC` [17] překladači, pokud jej podporuje. Tento přepínač povolí generování kódu nezávislého na své pozici.

`-DLLVM_INCLUDE_TESTS=OFF`
: Zakázání testů.

`-DCLANG_TABLEGEN="host/bin/clang-tblgen"`
: Cesta k předem přeloženému programu `clang-tblgen`. Tento program je spouštěn na hostujícím počítači a proto je nutné jej přeložit před samotným překladem nástroje Clang.

`-DC_INCLUDE_DIRS="install-dir/include"`
: Cesta k hlavičkovým knihovnám během překladu.

`-DBUILD_SHARED_LIBS=ON`
: Povolení překladu sdílených knihoven.

`-DCOMPILER_RT_BUILD_SANITIZERS=OFF`
: Zakázaní překladu *sanitizer runtimes*, jde o část `compiler-rt`, která způsobovala chyby během překladu.

Proměnné prostředí

Tyto proměnné jsou totožné jako při překladu GCC. Je nutné nastavit `PATH`, `CPPFLAGS`, `LDFLAGS`, `CFLAGS` a `CXXFLAGS`.

Překlad Clang 3.8.0

Samotný překlad se skládá z více fází. Před překladem verze pro OS Android je nutné přeložit, případně sehnat, programy `llvm-tblgen` a `clang-tblgen`. Tyto soubory jsou překládané pro hostitelský PC a tudíž se budou překládat odlišně. K překladu poslouží standardně nastavené prostředí, konfiguraci je možné spustit příkazem:

```
cmake -G Ninja <zdrojové soubory>

-DCMAKE_BUILD_TYPE=Release

-DLLVM_TARGETS_TO_BUILD="ARM;X86"
```

Přeložit bude stačit pouze zmíněné programy, k tomu poslouží 2 příkazy:

```
ninja bin/llvm-tblgen

ninja bin/clang-tblgen
```

Jako prerekvizitu je nutné nejdříve přeložit a nainstalovat sadu **binutils**. Překlad binutils je již zmíněný v sekci *Překlad GCC, Překlad GNU Binutils*. Clang 3.8.0 a LLVM během křížového překladu pro OS Android obsahuje pouze jeden opravný soubor, který opravuje chybu `__asm __volatile("svc 0x0": "=r"(start_reg))`. Tato chyba se nachází v `compiler-rt`. Spustíme konfiguraci s přepínači zmíněnými výše a překlad příkazem `ninja`. Instalaci provedeme příkazem `ninja install`.

Po instalaci, podobně jako u GCC, je nutné do instalační složky zkopírovat soubory, které se budou používat při překladu na samotném zařízení. Zkopírujeme hlavičkové soubory, systémové knihovny a některé další důležité soubory. Tyto soubory najdeme v nástroji Android NDK v umístění `platforms/android-XX/arch-arm/usr` (XX je verze Android API). Adresář `include` bude obsahovat hlavičkové soubory, adresář `lib` všechny ostatní. na rozdíl od GCC, je nutné zkopírovat i knihovnu `libgcc.a`, která je umístěna v adresáři `křížový_překladač/lib/gcc/.../armv7-a/hard/`. Stejně jako u GCC, přidáme podporu pro C++. Celý tento proces je automatizován při překladu balíku příkazem `make-module.sh clang380`.

2.4 Překlad volitelných nástrojů

Tyto nástroje nejsou nedílnou součástí během překladu, ale ulehčují jej. Mezi nejdůležitější nástroj patří GNU Make [24], který kontroluje generování binárních a dalších souborů ze zdrojových souborů. Podařilo se také přeložit pro ARM procesor s OS Android i další nástroje, jako je GNU Tar [53] pro vytváření archívů a GNU sed [64] pro filtraci textu.

GNU Make
: Překlad probíhá standardně. Je možné upravit cestu ke standardnímu programu *shell* v souboru `job.c` (OS Android má standardní shell v umístění `/system/bin/sh`).

GNU Tar
: Standardní překlad bez úprav.

GNU sed
: Standardní překlad bez úprav.

Kapitola 3

Návrh a implementace aplikací

Nyní se zaměříme na vývoj aplikací. První vytvořená aplikace bude sloužit jako emulátor terminálového prostředí, který bude umožňovat použití vytvořených nástrojů z předchozí kapitoly. Druhá aplikace bude demonstrovat funkčnost těchto nástrojů a poukáže na omezení ze strany OS Android (například při vytváření nativního okna).

3.1 Aplikace Terminál

Aplikace bude vycházet z již existujícího terminálu, který upravíme pro účely této práce. Původní aplikace se jmenuje **Terminal Emulator for Android**, je na Google Play hodnocena velice dobře a má dostupné zdrojové kódy na serveru GitHub [28] pod licencí Apache v2.0. Doplníme funkcionalitu pro načítání rozšiřujících balíčků, jejich připojení do systému a práci s nimi. Tyto rozšiřující balíčky budou zabaleny ve formátu OBB, tento formát je v operačním systému Android oficiálně podporován a Google Play nabízí prostor až 2GB pro uložení 2 balíků k jedné aplikaci na serveru. Velikou nepříjemností je, že soubor OBB vychází ze souborového systému *FAT16*, který má jistá omezení, a proto je občas nutné se těmto omezením přizpůsobit.

Implementace aplikace

Viditelná úprava pro uživatele proběhla přidáním zatrhávajícího políčka do nastavení, které povoluje nebo zakazuje tato rozšíření, druhá viditelná změna je volba cesty k těmto rozšířením. V rámci aplikace se tento adresář projde a pokusí se připojit všechny soubory s příponou `.obb`.

Před připojováním je nutné požádat o systémovou službu `STORAGE_SERVICE` [40]. Při získání tohoto objektu můžeme použít funkci pro připojení těchto souborů - `mountObb(cesta k souboru, heslo, posluchač)`. Parametr `posluchač` slouží k odposlechu událostí, protože předchozí funkce pouze požádá operační systém o danou službu a ten vrátí odpověď právě tomuto objektu, který se na základě předaných údajů (cesta k souboru a status) musí rozhodnout, co dělat. Jestliže dojde k připojení souboru `.obb`, který obsahuje soubor `environment.sh` v kořenovém adresáři, je z něj spuštěn příkazem:

```
. environment.sh parametr
```

Parametrem pro `environment.sh` (proměnná s názvem `parametr`) je cesta k adresáři, v němž se nachází (v OS Android je to `/mnt/obb/unikátní-id`). Cestu k připoje-

ném `.obb` souboru získáme opět pomocí služby `STORAGE_SERVICE`, která obsahuje metodu `getMountedObbPath` s jedním parametrem (cestou k původnímu `.obb` souboru). Tento soubor upravuje některé proměnné v prostředí terminálu, jde především o nastavení proměnných `PATH`, `LD_LIBRARY_PATH`, `LIBRARY_PATH`, ale může obsahovat i další proměnné.

Vytvoření instalačního souboru

Aplikace pro OS Android jsou zabaleny do instalovatelných APK souborů [12], které obsahují přeložené moduly a důležité informace pro spuštění aplikace (soubory s příponou `.dex` [12], `AndroidManifest.xml`, `resources.arsc` a další soubory). Aplikace používá k překladu systém **Gradle** [57], který lze importovat do IDE (například Android Studio nebo Eclipse) a lze jednoduše vygenerovat výsledný APK soubor. Tento překlad je možný i v konzoli, který popisují webové stránky pro Android vývojáře [11]. Původní aplikace si zachovává originální ID (`jackpal.androidterm`) a proto je nutné originální Android Terminál odinstalovat.

3.2 Vytvoření rozšiřujících balíčků

V této práci soubory OBB nebudou šifrovány heslem. O zabalení souborů do OBB se stará nástroj `jobb`, který se nachází ve složce `Android SDK/tools/`. Použití může vypadat následovně:

```
jobb -d libexec/ -pn jackpal.androidterm -pv 1 -o libexec.obb
```

`-d` ... složka, jejíž obsah zabalit.

`-pn` ... ID aplikace, která má přístup do tohoto souboru.

`-pv` ... verze souboru.

`-o` ... název výstupního souboru.

Pro zjednodušení je v projektu soubor `make-package.sh`, který zabalí složku `install` do tohoto formátu na základě nastavení uvedených v `configure.sh`.

3.3 Demonstrační aplikace

Tato sekce se bude zabývat návrhem aplikace, která bude sloužit jak pro účely testování vývojového prostředí, které je přeložené a k dispozici v mobilním zařízení (pomocí upravené aplikace emulující terminál), tak pro prozkoumání základních funkcí operačního systému Android.

V aplikaci se zaměříme na otevření OpenGL okna, ve kterém zobrazíme primitivní 2D a 3D objekty. Ze standardních knihoven použijeme `libEGL.so`, `libGLESv1_CM.so`, `libui.so`, `libgnustl_shared.so` a `libm_hard.a`. První problém nastane při snaze o otevření nativního okna, které není možné otevřít přímo ze spustitelného binárního souboru. Operační systém Android používá pro správce oken `SurfaceFlinger`, který se stará o kompozici a následný zápis do *frame bufferu* (`/dev/graphics/fb0`). Standardně o otevření okna požádá virtuální stroj, který v sytému Android vytvoří prostředí pro běh aplikace napsané

v jazyce Java, a je tedy nutné tuto činnost obejít. Jednou z možností je, stáhnout si hlavičkové soubory, které se používají při vývoji OS Android a pokusit se s jejich pomocí a s pomocí knihovny `libgui.so` požádat o nativní okno.

Pro jednoduchost je v tomto řešení použita starší knihovna `libui.so`, která obsahuje funkci `android_createDisplaySurface()` pro otevření nativního okna. Problém nastane tehdy, pokud je `SurfaceFlinger` stále aktivní (aplikace se pozastaví a čeká). Pro ukončení služby SurfaceFlinger je nutné se do zařízení připojit vzdáleně a se zvýšenými právy (uživatel `root`) spustit příkaz `stop`. Pro komunikaci se zařízením se používá nástroj `ADB` (Android Debug Bridge) a příkaz `adb shell`.

Jak je z návrhu patrné, nejedná se o aplikaci pro standardního uživatele, ale o aplikaci, která se snaží ukázat, že programování čistě v jazyce C a C++ v OS Android není jednoduché a autoři tohoto systému s ním nepočítají, přesto je to možné.

Implementace aplikace

O zobrazení se bude starat knihovna `libEGL.so` s doprovodem `libGLESv1_CM.so`. Nativní okno získáme z knihovny `libui.so`, aby nebylo nutné vkládat hlavičkové soubory pro tuto knihovnu (tyto soubory nejsou dostupné standardně v NDK), použijeme jen deklaraci funkce, kterou budeme potřebovat:

```
extern "C" EGLNativeWindowType android_createDisplaySurface();
```

Samotná implementace je rozdělena do více souborů. Soubor `main.cpp` slouží jako vstupní bod aplikace, `models.h` obsahuje definice pro modely (barvy a pozice trojúhelníků), soubory `renderer.cpp` a `renderer.h` inicializují okno a scénu, do které zobrazí modely. Projekt přeložíme pomocí vytvořené terminálové aplikace s pomocí programu `make` a souboru `Makefile`.

`Makefile`
: Použijeme standard C++11 (`-std=c++11`) a ABI s VFP (`-mhard-float`). Při sestavení aplikace připojíme knihovny pomocí parametrů `-lGLESv1_CM -lEGL -lui -lgnustl_shared`.

`main.cpp`
: Vytvoří instanci třídy `Renderer` a spustí ji.

`models.h`
: Obsahuje konstantní pole vrcholů těles pro scénu a jejich barvy.

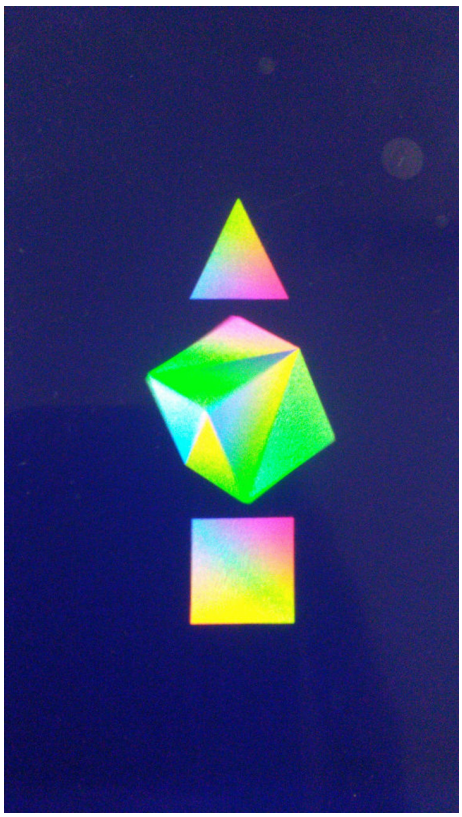
`renderer.h`, `renderer.cpp`
: Zde se nachází nejdůležitější část programu, ve funkci `init()` provedeme vytvoření scény do které budeme vykreslovat, dále funkce `run()` obsahuje kód pro nastavení scény a nekonečnou smyčku pro vykreslování. Funkce `destroy()` uvolní prostředky.

  Získání plochy pro vykreslování nám umožní funkce deklarované v souboru `EGL.h` `eglGetDisplay`, `eglChooseConfig`, `eglCreateWindowSurface`, `eglCreateContext` a její aktivaci provedeme pomocí funkce `eglMakeCurrent`. Pokusíme se vybrat nejdříve variantu, která zobrazuje 32 bitů (červenou, zelenou, modrou a alfa kanál) na 8 bitech s hloubkovým bufferem o 24 bitech, pokud tato varianta není k dispozici, vynecháme alfa kanál a bude se zobrazovat 24 bitů, poslední pokus zahrnuje variantu se 16 bity.

Zároveň vypíšeme informace na standardní výstup, jako jsou podporované nastavení zobrazení a počet snímků za sekundu (u testovaných zařízení se číslo pohybovalo na 60 snímcích za sekundu).

Spuštění aplikace

Obrázek 3.1: Překlad demonstrační aplikace v aplikaci emulující terminál



Obrázek 3.2: Snímek displeje telefonu po spuštění demonstrační aplikace

Překlad výsledné demonstrační aplikace je zobrazen na obrázku 3.1. Překlad je spuštěn příkazem `make` v adresáři obsahujícím zdrojové soubory. na obrázku 3.2 je zobrazen displej mobilu, na kterém je aplikace spuštěna, vidíme trojúhelník, čtverec a krychli. Při spuštění se tyto objekty pohybují v 3D prostoru směrem od kamery a po cestě rotují.

Tato aplikace měla demonstrovat, jak je složité otevřít v zařízení s OS Android nativní okno. Grafické okno se podařilo otevřít nestandardním způsobem, ale například zpracování vstupů od uživatele by vyžadovalo mnoho práce navíc, protože to OS Android nepodporuje při spuštění aplikace přímo z binárního souboru. Vývoj demonstrační aplikace přímo v zařízení trval déle než na PC a to z důvodu chybějícího IDE a ladícího programu.

Kapitola 4

Testování vytvořeného prostředí

Testování probíhalo na zařízení *Samsung Galaxy Nexus* s neoficiálním systémem Android (verze 5.1.1) od CyanogenMod. Jde o zařízení s procesorem TI OMAP 4460 (2x Cortex-A9, 1 200 MHz, 45nm) a 1 GB operační paměti (RAM). Nejdříve se testovací sada překládala pomocí nástrojů gcc verze 5.3.0, binutils, make, sed a poté pomocí sady nástrojů Clang 3.8.0, binutils, make, sed. Testovací sada obsahovala mnou vypracovanou verzi školních domácích úloh z předmětů Jazyk C, Algoritmy a Formální jazyky a překladače s upravenými `Makefile` soubory pro OS Android a procesory ARM. Časové údaje byly získány z programu `time` (složka `real`).

Překlad byl proveden nejdříve pro ABI bez VFP (`-mfloat-abi=soft`), kdy se program sestavuje s matematickou knihovnou `libm.so` v zařízení a následně i s ABI s VFP (`-mfloat-abi=hard`), kdy se program sestavuje s matematickou knihovnou `libm_hard.a`, která byla zkopírována ze sady NDK. Při překladu s parametrem `-mfloat-abi=hard` je nutné přidat definici `-D_NDK_MATH_NO_SOFTFP=1` k překladu C a C++ zdrojových souborů. Spustitelné soubory musí být přeloženy s parametry `-fPIE` a `-pie` pro vytvoření binárních souborů ve formátu *Position Independent Executable* a sdílené knihovny s parametrem `-fPIC` (*Position Independent Code*).

4.1 Výsledky testů

Příklady z předmětu *Jazyk C*

Tato sada školních příkladů se sestává ze dvou domácích úloh. První část zahrnuje projekt se jménem `prvocisla`, `prvocisla-inline` a `steg-decode`, druhá `fold`, `fold2`, `wordcount` a `wordcount-dynamic`. Program `wordcount` byl odebrán z testů protože se sestavuje staticky, pro programy `prvocisla` a `prvocisla-inline` byla navýšena velikost zásobníku. Pro srovnání s PC (Intel® Core™ i7-4510U Processor, 8GB RAM, Linux Mint 17.3 64 bitů, GCC 5.3.0) jsou uvedeny časy překladu a spuštění i velikosti binárních souborů pro první část těchto úloh. Srovnání s již existující aplikací (Termux) je uvedeno v tabulkách 4.3 a 4.5. Překlad probíhal na stejném mobilním zařízení, na kterém se překládaly předchozí projekty.

GCC	Doba překladu [s]	Velikost [B]	Spuštění [s]
prvocisla	22,61	11 056	6,82
	29,53	9 992	6,67
prvocisla-inline	29,43	11 064	7,53
	30,02	10 000	7,49
steg-decode	32,93	14 252	0,02
	33,88	14 644	0,02
Clang			
prvocisla	13,52	44 928	7,07
	14,60	42 908	6,99
prvocisla-inline	14,46	45 008	8,57
	14,75	42 988	8,52
steg-decode	21,14	48 356	0,02
	21,04	47 484	0,02

Tabulka 4.1: Překlad první části domácích úloh z předmětu Jazyk C v zařízení s OS Android. První řádek je přeložený s parametrem `-mfloat-abi=soft`, druhý `-mfloat-abi=hard`.

PC (GCC)	Doba překladu [s]	Velikost [B]	Spuštění [s]
prvocisla	0,24	13 105	0,95
prvocisla-inline	0,24	13 105	0,91
steg-decode	0,31	13 837	0,02

Tabulka 4.2: Překlad první části domácích úloh z předmětu Jazyk C v PC.

Termux (GCC)	Doba překladu [s]	Velikost [B]	Spuštění [s]
prvocisla	0,85	50 528	7.07
prvocisla-inline	0,89	50 528	7.63
steg-decode	1.21	51 096	0,02

Tabulka 4.3: Překlad první části domácích úloh z předmětu Jazyk C v aplikaci Termux pomocí GCC 6.1.0 (`-mfloat-abi=softfp`).

GCC	Doba překladu [s]	Velikost [B]
fold	15,09	8 664
	13,15	8 620
fold2	13,68	12 204
	10,90	12 208
wordcount-dynamic	84,42	6 652 / 11 804
	87,86	6 608 / 7 812
Clang		
fold	7,80	9 104
	8,13	9 108
fold2	9,70	58 672
	10,07	58 680
wordcount-dynamic	69,26	9 028 / 12 956
	71,45	9 036 / 8 664

Tabulka 4.4: Překlad druhé části domácích úloh z předmětu Jazyk C. První řádek je přeložený s parametrem `-mfloat-abi=soft`, druhý `-mfloat-abi=hard`.

Termux (GCC)	Doba překladu [s]	Velikost [B]
fold	0,70	11 856
fold2	3,60	21 216
wordcount-dynamic	2,70	7 500 / 49 044

Tabulka 4.5: Překlad druhé části domácích úloh z předmětu Jazyk C v programu Termux pomocí GCC 6.1.0 (`-mfloat-abi=softfp`).

Příklady z předmětu *Algoritmy*

Pro předmět Algoritmy se vypracovávaly také dvě sady úloh, první sada obsahovala úlohy `c201` (jednosměrně vázaný lineární seznam), `c203` (fronta znaků v poli), `c206` (dvousměrně vázaný lineární seznam) a druhá `c016` (tabulka s rozptýlenými položkami), `c401` (rekurzívní implementace operaci nad binárními vyhledávacími stromy), `c402` (nerekurzívní implementace operací nad binárními vyhledávacími stromy).

GCC	Doba překladu [s]	Velikost binárního souboru [B]
c201	5,77	19 820
	5,35	14 156
c203	4,18	14 464
	5,76	14 468
c206	3,66	26 912
	5,44	26 912
c016	4,55	14 816
	4,46	12 776
c401	3,90	13 512
	5,61	13 512
c402	4,69	16 160
	6,32	16 164
Clang		
c201	5,44	14 256
	5,35	14 260
c203	4,51	15 468
	4,92	15 476
c206	2,41	27 244
	4,94	27 248
c016	4,81	12 888
	3,93	15 504
c401	3,97	13 640
	3,73	13 644
c402	3,34	16 896
	3,27	16 900

Tabulka 4.6: Překlad domácích úloh z předmětu Algoritmy. První řádek je přeložený s parametrem `-mfloat-abi=soft`, druhý `-mfloat-abi=hard`.

Příklady z předmětu *Formální jazyky a překladače*

Tento předmět obsahoval jeden, zato rozsáhlejší projekt `IFJ14`, který sloužil jako interpret jazyka *ifj14* založeném na jazyce Pascal.

	Doba překladu [s]	Velikost binárního souboru [B]
GCC	58.17	105 240
	60.41	100 692
Clang	41.72	107 436
	37.71	94 992

Tabulka 4.7: Překlad domácí úlohy z předmětu Formální jazyky a překladače. První řádek je přeložený s parametrem `-mfloat-abi=soft`, druhý `-mfloat-abi=hard`.

4.2 Shrnutí testování

Z výsledků testů je patrné, že překlad přímo v zařízení trval mnohem delší dobu, než je tomu u osobního počítače. Běh aplikací byl u první sady testovacích příkladů z předmětu Jazyk C asi 7x pomalejší v mobilním zařízení. V aplikaci Termux byl překlad mnohem rychlejší než při překladu nástroji z této práce, ale výsledná velikost těchto souborů byla větší. Běh programů přeložených aplikací Termux byl nepatrně pomalejší. Všechny přeložené programy fungovaly správně s parametry `soft`, `softfp` (s matematickou knihovnou `libm.so`) i `hard` (s matematickou knihovnou `libm_hard.a` a definicí `-D_NDK_MATH_NO_SOFTFP=1`) pro argument `-mfloat-abi`.

Z výsledků testování nelze určit, který překladač je vhodnější při programování v OS Android. GCC zpravidla vytvoří menší velikost binárního souboru a také je běh aplikace o něco rychlejší, Clang naproti tomu provádí překlad o něco rychleji. Oba překladače vytvořily vždy validní spustitelné soubory.

Závěr

V této práci bylo cílem vytvořit jednoduché vývojové prostředí pro překlad aplikací psaných v jazycích C a C++ přímo v mobilním zařízení s operačním systémem Android a zároveň toto prostředí otestovat pomocí demonstrační aplikace a sadou školních příkladů. Vývojové prostředí se skládá z aplikace emulující terminál a z rozšiřujících balíčků, které obsahují nástroje nutné pro překlad nebo rozšiřují funkcionalitu. Aplikace emulující terminál je založena na již existující a ověřené aplikaci s otevřeným zdrojovým kódem a je doplněna o vlastní kód umožňující propojit tyto balíčky s terminálovým oknem. K práci jsou přiloženy skripty, které automatizují překlad nástrojů a vytváření balíčků pro tuto aplikaci. I přesto, že je práce zaměřená na 32 bitové procesory firmy ARM, je možné spouštět výsledné aplikace i na 64 bitovém prostředí. Při vytváření nástrojů se vyskytlo několik chyb, které se podařilo opravit opravnými soubory a správnou konfigurací před překladem. Podařilo se přeložit jak GCC, tak i Clang s pomocí křížového překladače vygenerovaného z projektu CrystaX NDK.

Nástroje s překladačem GCC a Clang byly otestovány na reálných zařízeních, kdy se projevily jejich rozdíly, jako je velikost výsledných binárních souborů, doba překladu a rychlost výsledné aplikace při spuštění. Vytvořené rozšiřující balíčky pro aplikaci emulující terminál obsahující GCC nebo Clang s pomocnými nástroji zabírají méně než 100 MB a lze je umístit do libovolného adresáře v mobilním zařízení. Toto řešení umožňuje použití těchto nástrojů pro zařízení s malou vnitřní pamětí, které jsou rozšiřitelné o externí paměťové médium. Vytvořené vývojové prostředí podporuje spouštění pouze konzolových aplikací, protože operační systém Android spouští aplikace ve virtuálním stroji. Tento virtuální stroj vytvoří pro každou aplikaci rozhraní pro její běh, jako je například otevření nativního okna a umožňuje zpracování vstupů od uživatele. Pokud by se toto prostředí mělo stát použitelným pro aplikace používající grafické rozhraní, bude nutné přidat možnost exportu standardního instalačního souboru pro systém Android (APK), který by umožnil spuštění aplikace v tomto virtuálním stroji a nativní kód by byl propojen formou JNI s Java kódem. Překlad přímo v mobilním zařízení s operačním systémem Android není aktuálně oficiálně podporován a trvá mnohem déle než na osobních počítačích. Jelikož se výkon mobilních zařízení zvyšuje, je možné, že v blízké době bude doba překladu srovnatelná s dobou překladu na osobních počítačích. Operační systém Android není navržen pro aplikace přeložené přímo do binární podoby, ale je možné na něm vyvíjet a překládat menší konzolové aplikace psané v jazycích C a C++, které mají blíž k hardwaru mobilního zařízení.

Pro pohodlné programování v zařízení s operačním systémem Android by bylo nutné přeložit více pomocných nástrojů (některé v zařízení chybí, jiné mají často omezenější možnosti než na PC). Bohužel standardní knihovna jazyka C (Bionic) používaná v tomto systému a odlišná adresářová struktura mnohdy vynucuje opravy existujících zdrojových souborů a některé nástroje je obtížné přeložit. Jedním z velice důležitých nástrojů, který v této práci nebyl přeložen, je GDB pro ladění aplikací. Vývoj aplikací přímo v zařízení by byl jednodušší, pokud by vytvořené vývojové prostředí bylo více než jen terminálové okno. Tato nová

aplikace by mohla umožňovat přepínání mezi otevřenými soubory, jejich editaci se zvýrazněním syntaxe a spouštět překlad a ladění. I přes uvedené nedostatky je toto prostředí použitelné a může sloužit například pro základní výuku programování, kdy není k dispozici osobní počítač.

Literatura

[1] ABI Management — Android Developers. [Online; navštíveno 30.4.2016].
URL http://developer.android.com/ndk/guides/abis.html

[2] About Android TV — Android Developers. [Online; navštíveno 30.4.2016].
URL http://developer.android.com/tv/index.html

[3] Android 6.0 Marshmallow — Android Developers. [Online; navštíveno 30.4.2016].
URL http://developer.android.com/about/versions/marshmallow/index.html

[4] Android Apps on Google Play. [Online; navštíveno 30.4.2016].
URL https://play.google.com/store/apps

[5] Android Debug Bridge — Android Developers. [Online; navštíveno 1.5.2016].
URL http://developer.android.com/tools/help/adb.html

[6] Android NDK — Android Developer. [Online; navštíveno 22.4.2016].
URL http://developer.android.com/tools/sdk/ndk/index.html

[7] Android Open Source Project. [Online; navštíveno 22.4.2016].
URL https://source.android.com/

[8] Android Wear — Android Developers. [Online; navštíveno 30.4.2016].
URL http://developer.android.com/wear/index.html

[9] ArmHardFloatPort — Debian Wiki. [Online; navštíveno 30.4.2016].
URL https://wiki.debian.org/ArmHardFloatPort

[10] ARMv8-A Architecture — ARM. [Online; navštíveno 30.4.2016].
URL https://www.arm.com/products/processors/
instruction-set-architectures/armv8-architecture.php

[11] Building and Running from the Command Line — Android Developers. [Online; navštíveno 5.5.2016].
URL http://developer.android.com/tools/building/building-cmdline.html

[12] Building and Running Overview — Android Developers. [Online; navštíveno 5.5.2016].
URL http://developer.android.com/tools/building/index.html

[13] C++ Library Support — Android Developers. [Online; navštíveno 22.4.2016].
URL http://developer.android.com/ndk/guides/cpp-support.html

[14] clang: a C language family frontend for LLVM. [Online; navštíveno 1.5.2016].
URL http://clang.llvm.org/

[15] Clang vs Other Open Source Compilers. [Online; navštíveno 22.4.2016].
URL http://clang.llvm.org/comparison.html#gcc

[16] CMake. [Online; navštíveno 22.4.2016].
URL https://cmake.org/

[17] Code Gen Options — Using the GNU Compiler Collection (GCC). [Online; navštíveno 2.5.2016].
URL https://gcc.gnu.org/onlinedocs/gcc/Code-Gen-Options.html

[18] Cortex-A Series — ARM. [Online; navštíveno 30.4.2016].
URL https://www.arm.com/products/processors/cortex-a/index.php

[19] Floating Point — ARM. [Online; navštíveno 30.4.2016].
URL http://www.arm.com/products/processors/technologies/vector-floating-point.php

[20] GCC 6 Release Series — GNU Project. [Online; navštíveno 11.5.2016].
URL https://gcc.gnu.org/gcc-6/

[21] GCC Releases — GNU Project. [Online; navštíveno 30.4.2016].
URL https://gcc.gnu.org/frontends.html

[22] Getting Started: Building and Running Clang. [Online; navštíveno 2.5.2016].
URL http://clang.llvm.org/get_started.html

[23] GIMPLE — GNU Compiler Collection (GCC) Internals. [Online; navštíveno 22.4.2016].
URL https://gcc.gnu.org/onlinedocs/gccint/GIMPLE.html

[24] GNU Make — GNU Project - Free Software Foundation. [Online; navštíveno 1.5.2016].
URL https://www.gnu.org/software/make/

[25] The GNU MPFR Library. [Online; navštíveno 1.5.2016].
URL http://www.mpfr.org/

[26] Installing GCC: Building — GNU Project. [Online; navštíveno 22.4.2016].
URL https://gcc.gnu.org/install/build.html

[27] Introduction to cross-compiling for Linux. [Online; navštíveno 22.4.2016].
URL http://landley.net/writing/docs/cross-compiling.html

[28] jackpal/Android-Terminal-Emulator — GitHub. [Online; navštíveno 5.5.2016].
URL https://github.com/jackpal/Android-Terminal-Emulator

[29] JNI Tips — Android Developer. [Online; navštíveno 22.4.2016].
URL http://developer.android.com/training/articles/perf-jni.html

[30] JOBB — Android Developers. [Online; navštíveno 1.5.2016].
URL http://developer.android.com/tools/help/jobb.html

[31] Linux From Scratch. [Online; navštíveno 1.5.2016].
URL
http://www.linuxfromscratch.org/lfs/view/6.5/chapter06/binutils.html

[32] LLVM 2.6 Release Notes. [Online; navštíveno 1.5.2016].
URL http://llvm.org/releases/2.6/docs/ReleaseNotes.html

[33] The LLVM Compiler Infrastructure Project. [Online; navštíveno 1.5.2016].
URL http://llvm.org/

[34] MPC. [Online; navštíveno 1.5.2016].
URL http://www.multiprecision.org/index.php?prog=mpc

[35] NDK Programmer's Guide: ABI Management. [Online; navštíveno 1.5.2016].
URL http://brian.io/android-ndk-r10c-docs/Programmers_Guide/html/md_3_
_key__topics__c_p_u__support__chapter_1-section_8__a_b_is.html

[36] NEON — ARM. [Online; navštíveno 30.4.2016].
URL http://www.arm.com/products/processors/technologies/neon.php

[37] Processors — ARM. [Online; navštíveno 30.4.2016].
URL http://www.arm.com/products/processors/index.php

[38] RISC vs. CISC. [Online; navštíveno 30.4.2016].
URL http:
//cs.stanford.edu/people/eroberts/courses/soco/projects/risc/risccisc/

[39] Standalone Toolchain — Android Developers. [Online; navštíveno 1.5.2016].
URL http://developer.android.com/ndk/guides/standalone_toolchain.html

[40] StorageManager — Android Developers. [Online; navštíveno 5.5.2016].
URL http://developer.android.com/reference/android/os/storage/
StorageManager.html

[41] Submodel Options — Using the GNU Compiler Collection (GCC). [Online; navštíveno 22.4.2016].
URL https://gcc.gnu.org/onlinedocs/gcc-5.3.0/gcc/Submodel-Options.html

[42] Using Autotools. [Online; navštíveno 22.4.2016].
URL https://developer.gnome.org/anjuta-build-tutorial/stable/
create-autotools.html.en

[43] Using LD, the GNU linker - Options. [Online; navštíveno 1.5.2016].
URL https://ftp.gnu.org/old-gnu/Manuals/ld-2.9.1/html_node/ld_3.html

[44] The GNU General Public License v3.0 — GNU Project - Free Software Foundation. 2007, [Online; navštíveno 30.4.2016].
URL http://www.gnu.org/licenses/gpl-3.0.en.html

[45] A Brief History of GCC. 2008, [Online; navštíveno 30.4.2016].
URL https://gcc.gnu.org/wiki/History

[46] CrystaX. 2012, [Online; navštíveno 22.4.2016].
URL https://www.crystax.net/en/android/ndk

[47] CrossCompilingXorg. 2013, [Online; navštíveno 2.5.2016].
URL http://www.x.org/wiki/CrossCompilingXorg/

[48] Terminal IDE — Android Apps on Google Play. 2013, [Online; navštíveno 1.5.2016].
URL https:
//play.google.com/store/apps/details?id=com.spartacusrex.spartacuside

[49] Android. 2014, [Online; navštíveno 15.5.2016].
URL https://android.com/

[50] cmake-generators(7) — CMake 3.0.2 Documentation. 2014, [Online; navštíveno 2.5.2016].
URL https://cmake.org/cmake/help/v3.0/manual/cmake-generators.7.html

[51] GNU Binutils. 2014, [Online; navštíveno 1.5.2016].
URL https://www.gnu.org/software/binutils/

[52] Linux and Unix which command and examples. 2014, [Online; navštíveno 2.5.2016].
URL http://www.computerhope.com/unix/uwhich.htm

[53] Tar — GNU Project - Free Software Foundation. 2014, [Online; navštíveno 7.5.2016].
URL https://www.gnu.org/software/tar/

[54] Android Kernel Features. 2015, [Online; navštíveno 30.4.2016].
URL http://elinux.org/Android_Kernel_Features

[55] CppDroid - C/C++ IDE — Android Apps on Google Play. 2015, [Online; navštíveno 1.5.2016].
URL https://play.google.com/store/apps/details?id=name.antonsmirnov.
android.cppdroid

[56] GNU Libtool — The GNU Portable Library Tool. 2015, [Online; navštíveno 1.5.2016].
URL https://www.gnu.org/software/libtool/

[57] Gradle — Modern Open-Source Enterprise Build Automation. 2015, [Online; navštíveno 5.5.2016].
URL http://gradle.org/

[58] Ninja, a small build system with a focus on speed. 2015, [Online; navštíveno 22.4.2016].
URL https://ninja-build.org/

[59] Target Triplet — OSDev Wiki. 2015, [Online; navštíveno 22.4.2016].
URL http://wiki.osdev.org/Target_Triplet

[60] Terminal Emulator for Android — Android Apps on Google Play. 2015, [Online; navštíveno 1.5.2016].
URL https://play.google.com/store/apps/details?id=jackpal.androidterm

[61] Binder — ZDNet. 2016, [Online; navštíveno 30.4.2016].
URL http://www.zdnet.com/article/patrick-brady-dissects-android/

[62] C4droid - C/C++ compiler & IDE — Android Apps on Google Play. 2016, [Online; navštíveno 1.5.2016].
URL https://play.google.com/store/apps/details?id=com.n0n3m4.droidc

[63] The GNU Multiple Precision Arithmetic Library. 2016, [Online; navštíveno 1.5.2016].
URL https://gmplib.org/

[64] GNU sed — GNU Project - Free Software Foundation. 2016, [Online; navštíveno 7.5.2016].
URL https://www.gnu.org/software/sed/

[65] Termux — Android Apps on Google Play. 2016, [Online; navštíveno 1.5.2016].
URL https://play.google.com/store/apps/details?id=com.termux

[66] Visual Studio — Microsoft Developer Tools. 2016, [Online; navštíveno 1.5.2016].
URL https://www.visualstudio.com/

[67] Hat, R.: Position Independent Executables (PIE). 2015, [Online; navštíveno 22.4.2016].
URL https://access.redhat.com/blogs/766093/posts/1975793

[68] McLean, P.: Apple's other open secret: the LLVM Compiler. 2008, [Online; navštíveno 1.5.2016].
URL http://appleinsider.com/articles/08/06/20/apples_other_open_secret_the_llvm_complier

[69] Rehman, A. U.: Android Architecture. 2013, [Online; navštíveno 30.4.2016].
URL http://atiqurrehman.com/android-architecture/

Přílohy

Seznam příloh

Příloha A

Obsah CD

K této bakalářské práci je přiložený obsah, skládající se ze 3 adresářů:

- androidterm
- demoapp
- tools

androidterm

Tato složka obsahuje zdrojové kódy aplikace pro emulaci terminálu. Před samotným překladem je nutné tuto složku zkopírovat do umístění s oprávněním zápisu, dále je nutné vytvořit soubor `local.properties` v kořeni tohoto adresáře a naplnit jej informacemi, kde se nachází rozbalené nástroje *Android SDK* a *Android NDK*. Android SDK musí obsahovat `SDK` verzi 22 a `build-tools` ve verzi 22.0.1. Na počítači musí být rovněž nainstalovaná Java verze 7 nebo novější. Přeložená aplikace se bude nacházet v `./term/build/outputs/apk` pod jménem `term-debug.apk`. Obsah souboru `local.properties` může vypadat následovně:

```
sdk.dir=/home/username/Android/Sdk
ndk.dir=/home/username/Android/Ndk
```

Překlad pak proběhne následujícím příkazem:

```
./gradlew assembleDebug
```

demoapp

V tomto adresáři se nachází demonstrační aplikace. Pro překlad je nutné ji umístit do zařízení do interní paměti, kde jsou nastavena páva pro zápis a spouštění. Překlad se provede jednoduše pomocí souboru `Makefile`, tento soubor využívá nástroje `g++`, ale není problém jej vyměnit za nástroj `clang++`. Je nutné mít v zařízení aktivní balík s GCC (nebo Clang) a balík obsahující nástroj `GNU Make`.

tools

Tento adresář obsahuje nejdůležitější skripty, které slouží ke stažení, překladu a instalaci modulů. Použití tohoto balíku bylo již zmíněno v kapitole *Překlad nástrojů pro systém Android*. Před využitím těchto skriptů je nutné upravit soubor `configure.sh`, kde je nutné definovat cesty k nainstalovaným nástrojům Android SDK a CrystaX NDK. Překlad a vytvoření balíku GCC s podporou pro Makefile soubory proběhne těmito příkazy (je nutné mít tuto složku v umístění s povolením zápisu):

```
./make-module.sh binutils
./make-module.sh gmp
./make-module.sh mpfr
./make-module.sh mpc
./make-module.sh gcc5
./make-module.sh make
./make-package.sh gcc
```

Uvedené příkazy lze zkrátit do dvou (moduly v parametru se provedou postupně):

```
./make-module.sh binutils gmp mpfr mpc gcc5 make
./make-package.sh gcc
```

Nyní je v kořenovém adresáři vytvořen soubor `gcc.obb`, který po nahrání do zařízení do složky `AndroidTerm` na externí paměti lze propojit s výše vytvořeným terminálem (v terminálu je možné tuto cestu upravit). Překlad modulu Clang s nástrojem GNU Make proběhne podobně (výsledkem je soubor `clang.obb`):

```
./make-module.sh binutils
./make-module.sh clang380
./make-module.sh make
./make-package.sh clang
```

Nebo:

```
./make-module.sh binutils clang380 make
./make-package.sh clang
```

Přeložené soubory se instalují do složky `install`, a proto je nutné ji vymazat před vytvářením nového balíčku. Nastavení lze změnit v souboru `configure.sh`.