VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

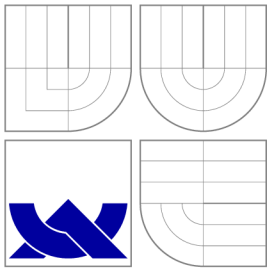
PARALELIZACE PROCESU PŘEKLADU A SESTAVOVÁNÍ PROGRAMOVÝCH MODULŮ

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

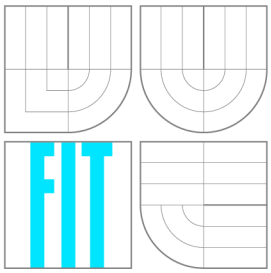
AUTOR PRÁCE
AUTHOR

VOJTĚCH KOLÁČEK

BRNO 2009



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

PARALELIZACE PROCESU PŘEKLADU A SESTAVOVÁNÍ PROGRAMOVÝCH MODULŮ

THE PARALLELIZATION OF A COMPILATION AND LINKING PROCESS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — VOJTĚCH KOLÁČEK
AUTHOR

VEDOUCÍ PRÁCE — Ing. KAREL SLANÝ
SUPERVISOR

BRNO 2009

Abstrakt

Tato bakalářská práce se zabývá problematikou paralelizace procesu překladu a sestavování programových modulů. Jsou zde probrány nástroje pro automatizaci překladu a sestavování programových modulů se zaměřením na možnosti paralelizace překladu. Cílem práce bylo implementovat tři systémy paralelizující překlad na více počítačích. Tyto systémy byly implementovány v programovacím jazyce C++ s využitím Windows API a spolupracují s překladači Microsoft C/C++ Compiler a Intel C++ Compiler.

Abstract

This bachelor's thesis deals with the parallelization of a compilation and linking process. Various tools for the automatization of the compilation process are discussed. The main criteria is focused on the ability of parallelization of the compilation process. Three systems which are able to parallelize the compilation process have been implemented. The implemented systems cooperate with Microsoft C/C++ Compiler and Intel C++ Compiler.

Klíčová slova

paralelizace, překlad, sestavení, C++, Microsoft Visual C++

Keywords

parallelization, compilation, linking process, C++, Microsoft Visual C++

Citace

Vojtěch Koláček: Paralelizace procesu překladu a sestavování programových modulů, bakalářská práce, Brno, FIT VUT v Brně, 2009

Paralelizace procesu překladu a sestavování programových modulů

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Ing. Karla Slaného. Další informace mi poskytl zaměstnanec firmy AVG Technologies CZ, s.r.o. Ing. Petr Hlávka. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Vojtěch Koláček
18. května 2009

Poděkování

Děkuji vedoucímu bakalářské práce Ing. Karlovi Slanému za poskytnutí konzultací a rad pří řešení problémů. Dále děkuji společnosti AVG Technologies CZ, s.r.o. a zvláště jejich zaměstnanci Ing. Petrovi Hlávkovi za to, že mi umožnili testovat výstupy mé práce na rozsáhlých zdrojových souborech aplikace AVG.

Obsah

Kapitola 1

Úvod

S rostoucími velikostmi aplikací a prodlužujícími se časy sestavení vyvstává potřeba sestavovací proces urychlit. Rychlejší hardware a optimalizace sestavovacího procesu tohoto cíle dosáhnout mohou, ale jen do určité míry; většího urychlení lze dosáhnout pomocí paralelizace. Sestavovací proces může mít mnoho částí: stažení zdrojových souborů z verzovacího systému, překlad, linkování, automatické testování, tvorba dokumentace a poznámek k vydání. V této práci se věnuji paralelizaci dvou základních úkonů, které se nachází ve všech sestavovacích procesech - překladu a linkování.

Tato práce má následující strukturu:

V druhé kapitole popisuji vývoj nástrojů pro automatizaci sestavení, jejich principy, výhody a nevýhody, a možnosti paralelizace. Dále zmiňuji existující nástroje pro paralelizaci sestavování na více počítačích. Ve třetí kapitole se zamýšlím nad možnými přístupy k paralelizaci na jednom a více počítačích, a navrhuji nástroje, které implementuji. Ve čtvrté kapitole popisuji experimenty provedené na implementovaných nástrojích, nad jejich složitostí, rozšiřitelností a efektivitou. V závěru jsou shrnuty výsledky a přínosy této bakalářské práce.

Kapitola 2

Překlad a sestavování programových modulů na platformě MS Windows

2.1 Překlad, sestavování a jejich automatizace

Překladem zdrojového kódu se obecně rozumí transformace zdrojového kódu napsaného v programovacím jazyce do výstupní reprezentace. Výstupem bývá často objektový kód, který je linkerem spojován do spustitelného souboru nebo knihovny. Tento způsob překladu do strojového kódu je doménou především jazyků překládaných do binárního kódu jako jsou C, C++, Fortran, Haskell atd.

Druhou kategorií jsou interpretované jazyky, kde je zdrojový kód typicky kompilován do vnitřního kódu (intermediate code), který je za běhu interpretován pomocí interpretu (interpreter). Mezi interpretované jazyky se počítají např. JavaScript, PHP, Smalltalk, Ruby aj.

Moderní interpretované jazyky však často přichází s technikou just-in-time compilation, která rozdíly mezi kompilovaným a interpretovaným kódem dále stírá. Jazyky uplatňující techniku just-in-time compilation překládají zdrojový kód do bytecode, který je virtuálním strojem (např. Java virtual machine, .NET Common Language Runtime) před spuštěním přeložen do strojového kódu. Mezi tyto jazyky patří Java a Microsoft .NET Common Intermediate Language (CIL).

Sestavování znamená spojení jednoho nebo více objektových souborů generovaných překladačem do výsledného programu. Hlavní částí objektového kódu generovaného překladačem je strojový kód; ten však nelze přímo spustit, protože odkazy do jiných částí programu nejsou vyhodnoceny a odkazuje se na ně pouze jménem. Objektový kód proto kromě strojového kódu typicky obsahuje také symboly (jména a adresy proměnných a funkcí), které linker použije k sestavení spustitelného programu.

Mezi nejpoužívanější formáty objektových a spustitelných souborů patří Executable and Linkable Format (ELF) a Common Object File Format (COFF) používané hlavně na UNIXových systémech, a Portable Executable (PE) používané na systémech Microsoft Windows. Jedním z formátů dříve používaných v UNIXových systémech je a.out, který byl později nahrazen formáty ELF a COFF.

Automatizované sestavení je obvykle chápáno jako proces automatizace překladu zdrojového kódu do strojového kódu nebo bytecode. K této základní funkci však u modernějších nástrojů přibyly další, jako například kontrola závislostí, paralelizace sestavení, možnost sestavení na více platformách, tvorba instalačních a jiných balíčků, spouštění testů, tvorba dokumentace, poznámek k vydání atd.

Automatizované sestavení bylo zavedeno jako alternativa k manuálnímu procesu, kdy musí člověk pokaždé provádět podobné úkony, které se opakují a jsou tedy náchylné k chybám. Dalšími výhodami automatizovaného sestavení je větší rychlost, menší chybovost, generování zpráv o výsledku sestavení (status report) a nové možnosti jako centralizovaná distribuce programu na více míst, automatický překlad kódu z verzovacích systémů (jako je např. CVS) aj.

2.2 Nástroje pro překlad a sestavování programových modulů

2.2.1 make

Jedním z prvních a na UNIXových systémech dosud populárních nástrojů pro automatizaci sestavení je nástroj *make*. Make byl vytvořen Stuartem Feldmanem v roce 1977 a přinesl integraci kompilace a instalace různých cílů (targets) v jediném souboru, což byl důležitý krok ve vývoji moderních sestavovacích nástrojů. Mezi nástroje používající stejný formát souboru, ale mající jinou kódovou základnu, patří například velmi rozšířené *GNU make* přinášející mnohé rozšíření k původnímu formátu a používané hlavně v systému GNU/Linux. V systémech FreeBSD, NetBSD a OpenBSD se často používá *BSD make* a v systému Windows v menší míře Microsoft *nmake*, které však není s původním *make* plně kompatibilní.

Závislosti jsou explicitně specifikovány v build souborech obvykle pojmenovaných `Makefile` nebo `makefile`, používajících vlastní formát specifický pro *make*. Program *make* udržuje statické definice závislostí definovaných v `makefile` a nedetekuje závislosti mezi soubory. Soubory se považují za změněné jen když se změní jejich čas modifikace, nebo když jsou smazány nebo přesunuty.

Syntaxe *make* [1] vypadá následovně:

```
myproduct : file1.c file2.c
    gcc -o myproduct file1.c file2.c
    echo "Finished building"
```

V této ukázce závisí cíl `myproduct` na souborech `file1.c` a `file2.c`. Pro vytvoření tohoto cíle vykoná *make* příkazy na následujících 2 řádcích.

Uvažujme, že `file1.c` je zdrojový soubor v jazyce C a že obsahuje řádek `#include "file1.h";`, tedy že `file1.c` závisí na `file1.h`. Pokud je ale po sestavení cíle `myproduct` `file1.h` změněn nebo smazán, *make* `myproduct` znovu nesestaví, protože závislost na `file1.h` je neznámá. Toto lze vyřešit přidáním `file1.h` do seznamu závislostí cíle `myproduct`, což je ovšem nepraktické, protože při každé změně závislostí ve zdrojových souborech by bylo potřeba manuálně změnit závislosti i v `makefile`.

GNU make obsahuje přepínač -j N, který paralelně spustí N procesů překladače. Používá graf závislostí mezi cíli k tomu, aby byly paralelně spuštěny jen cíle, které jsou na sobě nezávislé.

Nástroj *GNU make* podporuje paralelní překlad pouze na jednom stroji.

2.2.2 Jam

Jam je modernější nástroj pro automatizaci sestavení, určen jako náhrada či alternativa *make*. Pro popis závislostí používá tzv. `Jamfile`, které mohou mohou hierarchicky vkládat další `Jamfile`. *Jam* je multiplatformní a primárně je určen pro sestavování C a C++ projektů.

Jam obsahuje vestavěnou množinu pravidel zvanou *Jambase*, která definuje různé předpisy, které se používají k sestavení programu. *Jambase* dokáže např. při změně hlavičkového souboru znovu zkompilovat všechny soubory, které tento hlavičkový soubor inkludují. Na rozdíl od *make* není potřeba explicitně hlídat závislosti na hlavičkových souborech, jen samotné zdrojové soubory.

`Jamfile` mají vlastní, od make poněkud odlišný jazyk, jak lze vidět na ukázce z [1]:

```
SubDir TOP src ;

SubInclude TOP src subdir1 ;
SubInclude TOP src subdir2 ;

Main hello : hello.c foo.c bar.c baz.c ;
LinkLibraries hello : libother libcommon ;
```

První řádek specifikuje pozici `Jamfile` v adresářové hierarchii; tento se nachází v adresáři nazvaném `src`. Další dva řádky vkládají dva další `Jamfile` umístěné v podadresářích. Další řádek definuje spustitelný soubor `hello`, který obsahuje čtyři uvedené `.c` soubory, a poslední řádek knihovny, se kterými musí být `hello` slinkován. Tento Jamfile vytvoří spustitelný soubor `hello` na všech podporovaných platformách; potřebné prefixy a suffixy jsou pro každou platformu *Jamem* automaticky doplněny.

Běh *Jamu* se dělí na tři etapy:

- V první etapě jsou načteny všechny `Jamfile` a pro všechny cíle vyjmenované v argumentech programu (např. hello z výše uvedeného příkladu) jsou vygenerovány stromy závislostí (včetně vyhledání hlavičkových souborů ve zdrojových souborech). Pro detekci změny souboru je stejně jako u make použit čas poslední modifikace.
- Ve druhé etapě *Jam* zjistí, které soubory je potřeba znovu sestavit pro všechny specifikované cíle.
- Ve třetí etapě jsou použita konkrétní pravidla (jako je Main v `Jamfile` uvedeném výše) k sestavení cílů.

Mezi hlavní výhody *Jamu* patří rychlost, automatická detekce závislostí a tvorba stromů závislostí; sestaveny jsou právě ty cíle, jejichž zdroje byly změněny, přidány či odebrány. Jam je také portován na velké množství platforem (UNIX, Linux, Windows, Mac OS a jiné) a podporuje jejich zápisy cest k souborům. Cílen je převážně na jazyky C a C++.

Mezi nevýhody Jamu patří hůře dohledatelná dokumentace a příliš velké množství výstupního textu během překladu, ve kterém se těžce orientuje a který nelze jednoduše omezit. Syntaxe `Jamfile` je velmi striktní, řada příkazů musí končit mezerou a středníkem; vynechání mezery přitom vede k nesrozumitelným chybovým hlášením nebo těžce odhalitelným chybám v překladu. Automatická detekce závislostí také není dokonalá, například hlavičkové soubory inkludované bez jména podadresáře (např. `#include "myproject.h"`)

jsou kompilátorem nalezeny, protože aktuální adresář bývá součástí cesty, kterou kompilátor prohledává, ale *Jam* závislost na těchto hlavičkových souborech nemusí odhalit. Řešením je inkludovat soubory včetně relativní cesty (např. `#include "subdir/myproject.h"`), což je ale nepraktické.

Jam podporuje stejný mechanismus paralelizace jako *GNU make* (přepínač -j).

2.2.3 Ant

Mezi modernější sestavovací nástroje patří *Ant*, který používá k popisu akcí XML soubor. Zde se nachází seznam cílů (targets), přičemž každý cíl obsahuje seznam akcí, které musí Ant provést ke sestavení konkrétního cíle (tasks). *Ant* obsahuje velké množství vestavěných akcí pro různé části sestavovacího procesu, je však možné další akce jednoduše přidat, což vede k velké flexibilitě.

Formát konfiguračního souboru `build.xml` obsahuje několik základních tagů, jak lze vidět na následující ukázce z [1]:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<project name="My Project" basedir="." default="dist">

  <target name="dist"
          description="Jar up project ready for distribution"
          depends="run_tests">
    <jar destfile="dist/project.jar" basedir="build/classes"/>
  </target>

  <target name="compile"
          description="Compile the source files">
    <javac srcdir="src" destdir="build/classes"/>
  </target>

  <target name="clean"
          description="Remove all generated files">
    <delete dir="build/classes/org"/>
    <delete dir="build/test_classes/org"/>
    <delete dir="dist/*"/>
  </target>

</project>
```

Základní XML element `<project>` se musí v build souboru vyskytovat právě jednou a obsahuje mimo jiné elementy `<target>`, což jsou jména cílů k provedení, které se předávají *Antu* jako parametry příkazové řádky, a elementy `<property>`, které zastávají roli konstanty. Elementy `<target>` mohou specifikovat, že závisí na dalších elementech `<target>`. Každý `<target>` obsahuje jeden nebo více elementů `<akce>`, což jsou elementy popisující akce, které *Ant* provede při sestavování cíle. V ukázce jsou zmíněné např. `<delete>`, `<javac>` a `<jar>`, *Ant* však obsahuje přes 80 základních (core) akcí a přes 60 volitelných akcí. Základní akce slouží ke kompilaci Javy, práci se souborovým systémem, archivací, e-mailem a CVS. Volitelné akce rozšiřují podporu na další verzovací systémy, unit testy a další.

Mezi slabé stránky *Antu* patří komplexní řetězec závislostí (pokud cíl 3 závisí na cílech 1 a 2 a cíl 2 také závisí na cíli 1, a *Ant* se spustí s parametry „cíl 2 cíl 3", cíl 1 se vykoná 2x, jednou pro cíl 2 a jednou pro cíl 3). Další slabou stránkou je, že build soubory používají k uložení hodnot elementy `<property>`, které ale nejsou tak mocné, jako proměnné v programovacích jazycích. Pokud je property jednou definována, nelze ji již dále v build souboru změnit, property nemůže odkazovat na jméno jiné property atd.

Ant je psaný v jazyku Java, potřebuje ke běhu Java platformu a nejlépe se hodí k sestavování projektů v jazyce Java; jde pomocí něho však překládat i projekty v jiných jazycích, např. C/C++, i když mnohem větší možnosti má v tomto směru *NAnt*.

Paralelizace je v *Antu* implementována pomocí akce `<parallel>`. Všechny další akce do ní vnořené jsou spuštěné v novém procesu.

2.2.4 NAnt

NAnt je sestavovací nástroj postavený na *Antu*, ale cílený na prostředí .NET.

2.2.5 SCons

SCons je moderní nástroj implementovaný v Pythonu a používá jako své build soubory skripty v Pythonu typicky pojmenované `SConscript`.

Mezi vlastnosti *Sconsu* patří přenositelné build soubory (`SConscript` používají takový zápis specifikace programů, který je nezávislý na platformě), automatické zjišťování závislostí pro množství jazyků (C, C++, D, Java a Fortran), podepisování všech souborů MD5 hashem pro přesnější detekci, zda soubor musí či nemusí být znovu zkompilován a integrace s verzovacími systémy (SCCS, RCS, CVS, BitKeeper, Perforce a jiné).

Tím, že se pro build soubory používá plnohodnotný programovací jazyk, získává *SCons* mnohé možnosti, jako je použití debuggeru, profileru a dalších nástrojů, které jsou tradičně součástí programovacích jazyků, ale u nástrojů pro automatické sestavení typicky chybí. Další velkou výhodou je rozšiřitelnost a modularita; ta část *SConsu*, která provádí zjišťování závislostí a spouštění příkazů (tzv. "build engine") je oddělena od té části *SConsu*, která specifikuje, které soubory se mají kompilovat.

Tuto vlastnost lze demonstrovat na následujícím `SConscript` souboru převzatém z [1]:

```
# Explicitně umožní tomuto souboru použít prostředí definované
# v~jiném souboru
Import('env')

# Definuje spustitelný soubor pojmenovaný 'hello'
env.Program(target = 'hello',
            source = ['hello.c', 'foo.c', 'bar.c', 'baz.c'],
            LIBS = ['other', 'common'])
```

Řádek obsahující `env.Program` definuje spustitelný soubor 'hello' obsahující čtyři zdrojové soubory v jazyce C, které je potřeba slinkovat s knihovnami v seznamu LIBS. Příslušné prefixy a suffixy *SCons* doplní pro každou platformu zvlášť.

Mezi slabé stránky *SConsu* patří například časová náročnost inicializace sestavení, což je ovšem daň za automatickou detekci závislostí.

Scons podporuje stejný mechanismus paralelizace jako *Jam* a *GNU make* (přepínač -j).

2.2.6 MSBuild

MSBuild je build platforma společnosti Microsoft a jako své build soubory používá XML soubory s podobnou syntaxí, jako má *Ant*. *MSBuild* byl poprvé použit v .NET Framework 2.0 a Visual Studio 2005 (8.0) a novější ho používají k sestavování svých projektových souborů. Projekty (.csproj, .vcproj aj.) vytvořené ve Visual Studio 2005 a novějších jsou tedy vlastně build soubory pro *MSBuild* (podle MSBuild XML Schema).

Jako u *Antu* jsou základními elementy MSBuild souborů cíle (target) a úkoly (tasks). Cíle (elementy `<target>`) umožňují seskupování úkolů do do sekvencí. Každý cíl by měl být identifikován atributem `Name` a může být přímo volán parametrem MSBuildu. Úkol definuje akce, které *MSBuild* provede; může jít buď o vestavěné akce jako je např. `Copy`, nebo akce definované uživatelem.

Následuje ukázka akce `Copy` převzata z [4]:

```
<Target Name="CopyFiles">
  <Copy
    SourceFiles="@(MySourceFiles)"
    DestinationFolder="\\PDSERVER01\SourceBackup\"
  />
</Target>
```

Items jsou elementy, které definují vstupní soubory projektu. Jsou definované jako potomci elementu `<ItemGroup>` a nejběžnějším elementem je `Compile`, který označuje souboru určené k překladu.

Následuje ukázka převzata z [4]:

```
<ItemGroup>
  <Compile Include="Form1.Designer.vb">
    <DependentUpon>Form1.vb</DependentUpon>
<SubType>Form</SubType>
  </Compile>
</ItemGroup>
```

Vlastnosti (potomci elementu `PropertyGroup`) se používají k ukládání vlastností projektu (konstant). Je možné je definovat jen za určitých podmínek; například následující XML z [4] definuje `PropertyGroup`, která bude vložena jen pokud je projekt sestavován jako `Release` pro platformu `AnyCPU`:

```
<PropertyGroup Condition=" '$(Configuration)|$(Platform)' ==
                          'Release|AnyCPU' ">
  <DebugType>pdbonly</DebugType>
  <DefineDebug>false</DefineDebug>
  <DefineTrace>true</DefineTrace>
  <Optimize>true</Optimize>
  <OutputPath>bin\Release\</OutputPath>
</PropertyGroup>
```

Zatímco jednotlivé projekty jsou uloženy ve výše uvedeném XML formátu, Solution (.sln) soubory jsou uložené v textovém formátu podobném Visual Basic syntaxi. Solution soubor obsahuje informace o solutionu, jako je seznam projektů, konfigurace sestavení a ostatní nastavení nezávislé na projektu.

Následuje ukázka Solution souboru z [4]:

```
Microsoft Visual Studio Solution File, Format Version 9.00
# Visual Studio 2005
Project("{F184B08F-C81C-45F6-A57F-5ABD9991F28F}") = "FirstProject",
  "FirstProject\FirstProject.vbproj",
  "{FD041258-F0D6-4A69-9426-1A40B52126D4}"
EndProject
Global
  GlobalSection(SolutionConfigurationPlatforms) = preSolution
    Debug|Any CPU = Debug|Any CPU
  EndGlobalSection
  GlobalSection(ProjectConfigurationPlatforms) = postSolution
    {FD041258-F0D6-4A69-9426-1A40B52126D4}.Debug|Any CPU.ActiveCfg =
       Debug|Any CPU
    {FD041258-F0D6-4A69-9426-1A40B52126D4}.Debug|Any CPU.Build.0 =
       Debug|Any CPU
  EndGlobalSection
  GlobalSection(SolutionProperties) = preSolution
    HideSolutionNode = FALSE
  EndGlobalSection
EndGlobal
```

Solution v této ukázce obsahuje jeden projekt (`FirstProject`) s výčtem závislostí a globální sekci obsahující nastavení vztahující se k solutionu, jako konfigurace platforem aj.

MSBuild přináší velkou změnu oproti původním formátům projektových souborů a build procesům Visual Studia před verzí 2005, které byly pro každý jazyk jiné; a to jednotné a transparentní sestavovací prostředí s jednotným formátem build souborů pro různé programovací jazyky. Nevýhodou zůstává manuální správa závislostí mezi projekty, která se u velkých projektů stává problematickou. Úprava solution souborů, které jako jediné nepoužívají jako svůj formát XML, je také složitější [3].

Paralelizace je v MSBuildu realizována od verze .NET Framework 3.5, a to ve formě parametru příkazové řádky `/maxcpucount` (`/m`) a parametru *MSBuild* akce `BuildInParallel` [5].

Parametr příkazové řádky `/maxcpucount:N` může spustit až N procesů *MSBuildu*, které se starají o sestavení jednotlivých projektů. Parametr akce `BuildInParallel` určuje, jestli se konkrétní cíl smí paralelně sestavovat; pokud tento parametr u akce chybí, jeho hodnota je automaticky `true`.

Následující ukázka sestaví paralelně sestaví projekty `b.proj`, `c.proj` a `d.proj`:

```
a.proj:
  <Target Name="default">
    <MSBuild Projects="b.proj;c.proj;d.proj"
      BuildInParallel="true" />
  </Target>
Příkazový řádek: msbuild a.proj /m:4
```

2.3 Nástroje pro paralelizaci sestavení na více počítačích

2.3.1 distcc

Nástroj *distcc* slouží k distribuci překladu na více počítačů propojených sítí. Je navržen ke spolupráci s překladačem GCC, a to jazyky C, C++ a Objective-C; do jisté míry je však schopen spolupracovat i s Intel C++ Compiler a Sun Microsystems' Sun Studio Compiler Suite. Program *distcc* je používán převážně na systémech GNU/Linux [2].

Program *distcc* funguje jako agent pro překladač *gcc*, musí být tedy schopen porozumět parametrům příkazové řádky, které *gcc* přijímá. Poté *distcc* spustí preprocesor jazyka C, který načte hlavičkové soubory a provede direktivy preprocesoru (jako je `#ifdef`). Výstup preprocesoru už není dále závislý na hostitelském systému, protože do něj byly hlavičkové soubory, makra atd. vloženy. Tyto prekompilované soubory se pomocí TCP pošlou ostatním počítačům, kde jsou zkompilovány a výstupní objektové soubory jsou poslány zpět hostitelskému počítači.

Tato technika využívá možnosti oddělení preprocesoru jazyků C od překladače a zásadním způsobem šetří kapacitu sítě, protože není třeba po síti přenášet všechny hlavičkové soubory, ale jen relevantní části uložené ve výstupu preprocesoru.

Výhodou *distcc* je velké zrychlení, kdy přidání dalšího počítače urychlí proces sestavení až o 90% (oproti teoretickým 100%). Mezi nevýhody patří omezení na jazyky z rodiny C a omezení na platformu GNU/Linux.

2.3.2 IncrediBuild

IncrediBuild od firmy Xoreax je proprietární nástroj pro distribuci sestavení *MSBuild* projektů, projektů založených na make, obecných skriptů a aplikací pro více počítačů propojených sítí [7]. Na rozdíl od distcc řeší IncrediBuild problém rozdílného prostředí, operačního systému, DLL, COM objektů atd. cestou "virtuálního prostředí".

IncrediBuild používá systém agentů a koordinátora. Agenti jsou jednotlivé počítače, které mohou jednak iniciovat úkol, jednak přijímají a provádějí úkoly od koordinátora. Koordinátor je počítač, který přijímá požadavky a rozděluje je jednotlivým agentům.

Když agent vyšle požadavek na akci (např. na kompilaci projektu), na agenty, na kterých tato akce bude provedena, je zkopírován obraz prostředí iniciátora žádosti včetně zdrojových kódů a překladače. To má mnoho výhod; pokud překladač odkazuje na některý zdrojový soubor, knihovnu nebo záznam v registru, výsledek bude stejný, jako by se akce provedla na iniciátorovi žádosti. Odpadají také problémy s různými verzemi překladače, kterými někdy trpí distcc.

Mezi nevýhody naopak zvláště u větších projektů patří menší rychlost a větší zátěž sítě, kdy je po síti třeba všem spolupracujícím agentům odeslat velké množství dat.

Kapitola 3

Návrh řešení a implementace

3.1 Návrh

Jistou formu paralelizace v rámci jednoho počítače podporují všechny předem zmíněné nástroje pro automatické sestavení, limitem je však v tomto případě počet jader procesoru (typicky 1-4), a případně i rychlost diskové paměti. Dalším problémem při paralelizaci sestavení na jednom počítači je linker, který je již z principu neparalelizovatelný, protože spojuje objektové soubory vygenerované překladačem do spustitelného souboru.

Kvůli těmto omezením jsem se rozhodl pro návrh systému, který bude urychlovat sestavovací proces paralelizací na více počítačích. Tento přístup má své výhody, ale i nevýhody. Mezi výhody patří více využitelného strojového času díky použití procesorů na dalších počítačích. Mezi nevýhody patří zátěž sítě, po které se musí dopravovat sestavovaná a sestavená data. Potenciálně je však takovýto distribuovaný paralelní systém schopen vyššího urychlení sestavovacího procesu.

Protože možnosti nástrojů pro překlad a sestavování na platformě MS Windows diskutovaných v podkapitole 2.2 jsou z hlediska paralelizace velmi podobné, rozhodl jsem se pro implementaci systému nezávislého na nástroji pro překlad a sestavování.

Konkrétně jsem zvolil cestu nahrazení příkazu překladače agentní aplikací, která bude distribuovat překlad na více počítačů. Podobným způsobem je možné distribuovat i proces linkování, zde je však vzhledem ke kratšímu času linkování oproti překladu přínos paralelního zpracování na více počítačích sporný. Tento přístup má velkou výhodu v tom, že není závislý na nástroji pro automatizaci sestavení. Jedinou podmínkou je, aby použitý nástroj pro automatické sestavení měl dobrou podporu paralelního zpracování překladu a linkování. Dobře vygenerovaný strom závislostí může zvláště u velkých projektů vést k velké paralelizaci. Naopak špatně vygenerovaný strom či závislost velkého množství projektů na několika společných může vést k nutnosti sekvenčního zpracování.

Vzhledem k cílení na platformu MS Windows jsem vyvíjel systém pro spolupráci s nástroji dominantní platformy .NET, jmenovitě Microsoft Visual Studio, Microsoft C/C++ Compiler (`cl.exe`) a Microsoft Incremental Linker (`link.exe`).

Jako perspektivní se jevilo několik variant distribuce zdrojových souborů:

1. Uložení zdrojových souborů do sdíleného adresáře a překlad na vzdálených počítačích přímo ze síťového disku. Tento přístup má výhodu hlavně ve své jednoduchosti. Všechny vstupní i výstupní soubory se nachází na jednom místě, takže by výsledné binární soubory měly být stejné, jako kdyby sestavení probíhalo na jednom počítači. Mezi nevýhody bude pravděpodobně patřit nižší rychlost, protože je po síti

třeba sekvenčně přenášet velké množství hlavičkových souborů, což bude i na rychlých sítích pravděpodobně časově náročné.

2. Uložení zdrojových souborů na každý počítač, který se bude sestavení účastnit, a ukládání výstupních souborů do sdíleného adresáře na síťovém disku počítače iniciujícího překlad. Tento přístup sliboval větší zrychlení oproti prvnímu způsobu, protože jsou zdrojové soubory přeneseny zároveň a pouze jednou. Na druhou stranu vyvstávají problémy s konzistencí. Pokud při sestavení dojde ke změně zdrojových souborů (jako je generování hlavičkových souborů z XML souborů), pak se tato změna projeví jen na iniciujícím počítači a překlad na dalších počítačích následně selže. Tomuto lze předejít spuštěním těchto akcí na všech počítačích účastnících se sestavení ještě před spuštěním sestavení, nebo spuštěním těchto projektů na jednom počítači a distribucí již kompletních zdrojových souborů na ostatní počítače.

3. Spuštění preprocesoru na překládané soubory a distribuce výstupu preprocesoru na ostatní počítače. Tato varianta se jevila jako nejperspektivnější, protože jsou na ostatní počítače odeslány už prekompilované zdrojové soubory, které nejsou závislé na dalších hlavičkových souborech. Tuto metodu s úspěchem používá nástroj distcc.

Výše popsané varianty systémů paralelizace překladu jsem implementoval v jazyce C++ s využitím Windows API.

3.2 Parametry překladače

Všechny varianty systémů navrhované v podkapitole 3.1 musí dokázat zpracovat parametry příkazové řádky překladače, aby je mohly upravit a předat dále požadavek na překlad na vzdálené počítače.

Parametry překladače Microsoft C/C++ Compiler se dělí na několik skupin [8]:

1. **Optimalizace.** Tyto příkazy určují nastavení optimalizací, tedy optimalizaci na velikost nebo rychlost kódu, stupeň expanze inline funkcí atd. Tyto parametry není třeba brát v úvahu, protože v sobě nenesou informace o umístění a lze je tedy beze změny předat dále. Mezi tyto parametry patří například `/Os` (upřednostnit malý kód) nebo `/Od` (vypnout optimalizace).

2. **Generování kódu.** Tyto parametry nastavují volby jako jsou podpora instrukčních sad (SSE nebo SSE2), zacílení na některou z rodin procesorů (Pentium - Pentium III, Pentium 4, Athlon atd.), tvorba vícevláknových aplikací nebo dynamických knihoven. Také tyto parametry v sobě nenesou informace o umístění a lze je tedy beze změny předat dále. Příkladem může být přepínač `/EH` určující způsob ošetřování výjimek, nebo `/G7` optimalizující kód pro architekturu Pentium 4 nebo Athlon.

3. **Výstupní soubory.** Tyto příkazy nastavují cestu výstupních souborů jako jsou PDB soubory, objektové soubory (`.obj`), spustitelné soubory nebo prekompilované hlavičkové soubory. Tyto cesty je potřeba změnit na UNC cesty ke sdílenému adresáři, aby k nim mohly přistupovat vzdálené počítače.

4. **Ladění, preprocesor, nastavení jazyka, linkování, prekompilované hlavičkové soubory, a ostatní.** Všechny tyto příkazy o sobě nenesou informaci o umístění a není tedy třeba na nich nic měnit. Jedinou výjimkou jsou tzv. *response files* začínající znakem `@`. Tyto soubory obsahují další příkazy překladače ve stejném formátu, jako na příkazové řádce. Response files je potřeba přečíst a expandovat, protože se mohou nacházet kdekoliv v lokálním souborovém systému, kde by k nim vzdálené počítače neměly přístup.

   Response files mohou být ve formátu UTF-16, UTF-8 nebo ASCII [10], je proto potřeba počítat se všemi těmito kódováními. Z toho důvodu je pro načítání response files použita sada rutin EZUTF od Paula Sanderse [6] dostupná pod licencí GNU LGPL.

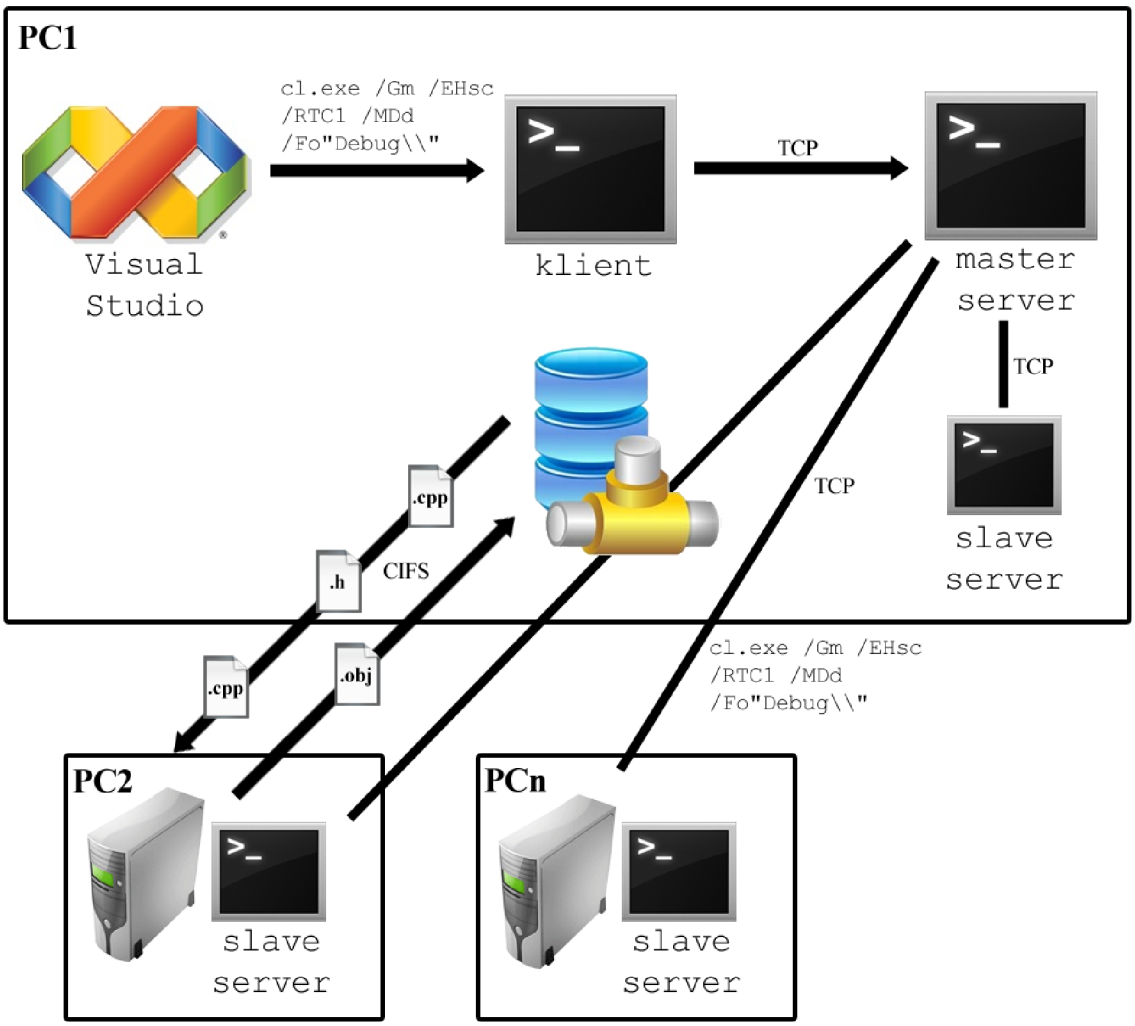
5. **Zdrojové soubory.** Jedná se o seznam zdrojových souborů pro překlad.

3.3 Systém 1 - sestavení ze síťového disku

3.3.1 Architektura

Tato varianta vyžaduje, aby byly všechny zdrojové soubory umístěné v adresáři sdíleném přes CIFS a přístupném ze všech počítačů v systému. Samotný překlad je rozdělen na více počítačů, které ke zdrojovým souborům přistupují ze síťového disku, a také tam ukládají výstupy překladu.

Architekturu tohoto systému ilustruje diagram 3.1.



Obrázek 3.1: Architektura systému 1

Samotný systém se dělí na 3 hlavní části - *klient*, který přijímá požadavky na překlad, *master server*, který přijímá požadavky od klienta a přeposílá je slave serverům, a *slave server*, který přijímá požadavky, provede překlad a vrátí výsledky překladu.

3.3.2 Klient

Klient je aplikace, která nahrazuje soubor překladače (`cl.exe`), který je volán nástrojem pro překlad a sestavování (tady Microsoft Visual Studio) s parametry pro překlad zdrojových souborů. Klient tedy musí dokázat zpracovat parametry příkazové řádky překladače, aby je mohl upravit a předat dále.

Ze skupin parametrů uvedených v podkapitole 3.2 je v tomto systému potřeba nahradit cesty výstupních a zdrojových souborů, a expandovat response files. Vzhledem k tomu, že stejná UNC cesta ke sdílenému adresáři by se musela přidat ke zdrojovým i výstupním souborům, je výhodnější na vzdálených počítačích spouštět kompilaci s pracovním adresářem nastaveným na cestu ke sdílenému adresáři, kde se budou soubory jak číst, tak zapisovat. Relativní cesty k souborům tedy zůstanou beze změny.

Jak lze vidět, v tomto systému téměř nedochází k úpravě parametrů určených překladači. Jedinou výjimkou je expandování *response files* na parametry příkazové řádky. Všechny potřebné úpravy provádí další prvky systému.

Soubory předané v parametrech příkazové řádky typicky patří do jednoho projektu. Bylo by tedy teoreticky možné rozdělit tyto soubory na několik skupin obsahujících méně souborů, kde každá skupina by se překládala paralelně na jiném počítači. Toto rozdělení s sebou nese jisté snížení výkonu (hlavičkové soubory se musí přenést přes síť několikrát), ale umožňuje to lepší paralelizaci i u menšího množství projektů, nebo v případech, kdy projekty jsou na sobě závislé. V případě závislostí projektů je třeba nejdříve přeložit tyto závislosti, což tvoří neparalelizovatelnou část, kterou je třeba co nejvíce zmenšit.

Tento způsob jsem implementoval a otestoval. Došlo při něm ale k závažnému problému. Překladač ukládá do souboru *program database* (VCx0.PDB, kde x je verze Visual C++) ladící informace a informace o projektu, a při paralelním překladu používá službu `mspdbsrv.exe`. Tato služba slouží jako arbitr přístupu k `PDB` souborům, aby se více procesů ve stejnou chvíli nepokoušelo zapisovat do stejného souboru.

Při přístupu k jednomu souboru z více počítačů se ovšem tato služba nepoužije, a často proto dochází k chybám v překladu způsobeným nemožností zapisovat do `PDB` souboru. Tento problém by bylo možno vyřešit vypnutím generování `PDB` souborů odstraněním parametru `/Z`, což je ale velmi nepraktické. Rozhodl jsem se proto vstupní soubory nedělit na části a překládat je jako celek vždy na jednom počítači, a to i za cenu potenciálně pomalejšího překladu způsobeného čekáním na překlad závislých projektů.

Po zpracování příkazové řádky a expandování response files je příkaz spolu s pracovním adresářem pomocí protokolu popsaného v podkapitole 3.3.5 poslána master serveru. Master server jako odpověď vrátí texty vypsané překladačem, které jsou generovány na standardní výstup a chybový výstup, a klient ukončí činnost s návratovou hodnotou taktéž vrácenou master serverem.

3.3.3 Master server

Master server je aplikace, která přijímá požadavky na překlad od klientů a přeposílá je slave serverům. Stará se také o nahrazování lokálních cest za UNC cesty, má nastaven maximální počet připojení na jeden slave server aj.

Master server je spuštěn na počítači, který iniciuje překlad.

Při spuštění načte master server ze souboru `master_server.conf` konfiguraci. Konfigu-

rační soubor je textový soubor, kde jsou záznamy uloženy na samostatném řádku ve tvaru

```
klíč=hodnota
```

Konfigurovatelné položky jsou:

1. `replace=original*replacement`. Tento řádek říká, že se má v řetězci pracovního adresáře a parametrů překladače nahradit text `original` za `replacement`. Řádků tohto typu může být libovolný počet.
2. `server=servername`. Tento řádek přidává do seznamu slave serverů, na kterých se bude provádět překlad, IP adresu nebo hostname `hostname`. Řádků tohto typu může být libovolný počet.
3. `max_processes=N`. Tento řádek určuje, kolik požadavků na překlad může být v určité chvíli odesláno na jeden slave server.
4. Řádky začínající středníkem slouží jako komentáře.

Následuje příklad konfiguračního souboru, který nahradí cestu `"D:\svn\"` za `"\\shiina\svn\"`, `"D:\projects\distributed\"` za `"\\shiina\distributed\"`, přidá slave servery `localhost` a `pbuild3` a omezí počet požadavků na překlad na jednom slave serveru na 4.

```
replace=D:\svn\*\\shiina\svn\
replace=D:\projects\distributed\*\\shiina\distributed\
server=localhost
server=pbuild3
max_processes=4
```

Po obdržení požadavku na překlad od klienta nahradí master server texty specifikované v konfiguračním souboru (typicky je to nahrazení absolutní cesty za UNC cestu sdíleného adresáře), a pokusí se od vlastní třídy `CShared` získat adresu volného slave serveru. Pokud ji nedostane, zařadí se do fronty a periodicky se dotazuje, zda již dostal adresu serveru. Až ji dostane, zašle požadavek na překlad slave serveru, jehož adresu obdržel. Odpověď potom přepošle zpět klientu a ukončí běh vlákna.

Každý požadavek se vyřizuje ve vlastním vlákně, proto je potřeba třída `CShared`, která pomocí kritické sekce garantuje korektnost přístupu ke sdíleným datům. Sdílená data tvoří seznam slave serverů, jejich vytíženost (tedy počet aktivních požadavků na překlad) a frontu požadavků.

3.3.4 Slave server

Slave server je aplikace, která přijímá požadavky na překlad od master serveru, spouští proces překladače (`cl.exe`), ukládá standardní výstup a chybový výstup překladače a posílá ho zpět master serveru. Má také nastaven maximální počet současně běžících procesů překladu.

Slave server může být spuštěn na počítači, který inicioval překlad, i na ostatních počítačích, které mají přístup ke sdílenému adresáři se zdrojovými soubory.

Při spuštění načte slave server ze souboru `slave_server.conf` konfiguraci. Konfigurační soubor je textový soubor, kde jsou záznamy uloženy ve stejném tvaru jako u master serveru, tedy

```
klíč=hodnota
```

Na rozdíl od master serveru lze u slave serveru konfigurovat jen hodnotu `max_processes=N`, kde číslo N určuje, kolik procesů překladu může v jednu chvíli běžet. Pokud pošle master server více požadavků na překlad než N, spustí se jen N procesů a ostatní požadavky se zařadí do fronty.

Po obdržení požadavku na překlad od master serveru se vytvoří nové vlákno obsluhující tento požadavek. Potom se slave server dotáže třídy `CShared` (stejné jako u master serveru), zda není počet procesů překladu na maximu. Pokud ano, zařadí se do fronty a periodicky se dotazuje, zda se místo již neuvolnilo. Až se místo uvolní, vytvoří pomocí funkce Windows API `CreateProcess` novou instanci překladače, které předá parametry obdržené od master serveru a nastaví pracovní adresář na hodnotu taktéž obdrženou od master serveru. Po dokončení překladu slave server pošle návratový kód, standardní výstup a chybový výstup překladače zpět master serveru, a ukončí se vlákno.

3.3.5 Komunikační protokol

Klient, master server a slave server spolu komunikují přes protokol TCP jednoduchým textovým protokolem. První zprávu (požadavek na překlad) vždy iniciuje klient a posílá ji master serveru, který ji (po případných náhradách textu) přepošle slave serveru. Slave server odpoví zprávou obsahující výsledek překladu (úspěch nebo neúspěch), návratový kód překladače, text vypsaný překladačem na standardní výstup a chybový výstup. Master server tuto zprávu bez úprav přepošle klientu, který vypíše text vrácený překladačem na standardní výstup, respektive na chybový výstup, a ukončí se se stejnou návratovou hodnotou jako překladač.

Požadavek na překlad má následující formát:

```
CL PAYLOAD_LENGTH
WORKING_DIRECTORY
PAYLOAD
```

`"CL"` je řetězec označující požadavek na překlad.

`PAYLOAD_LENGTH` je číslo označující délku obsahu zprávy v bajtech (např. 1432).

`WORKING_DIRECTORY` obsahuje řetězec určující pracovní adresář, ve kterém má být spuštěn překlad.

`PAYLOAD` je samotný obsah požadavku, tedy parametry překladače.

Pokud proběhl překlad v pořádku, odpověď má následující formát:

```
OK STDOUT_LENGTH STDERR_LENGTH
STDOUT
STDERR
```

`"OK"` je řetězec označující úspěch překladu.

`STDOUT_LENGTH` je číslo označující délku textu (v bajtech), který překladač vypsal na standardní výstup (např. 2954).

`STDERR_LENGTH` je číslo označující délku textu (v bajtech), který překladač vypsal na standardní chybový výstup (např. 4221).

`STDOUT` obsahuje text, který překladač vypsal na standardní výstup.

`STDERR` obsahuje text, který překladač vypsal na standardní chybový výstup.

Pokud překlad selhal, odpověď má následující formát:

```
ER STDOUT_LENGTH STDERR_LENGTH
RETURN_CODE
STDOUT
STDERR
```

Části `STDOUT_LENGTH`, `STDERR_LENGTH`, `STDOUT` a `STDERR` jsou stejné jako v předchozí odpovědi.

`"ER"` je řetězec označující neúspěch překladu.

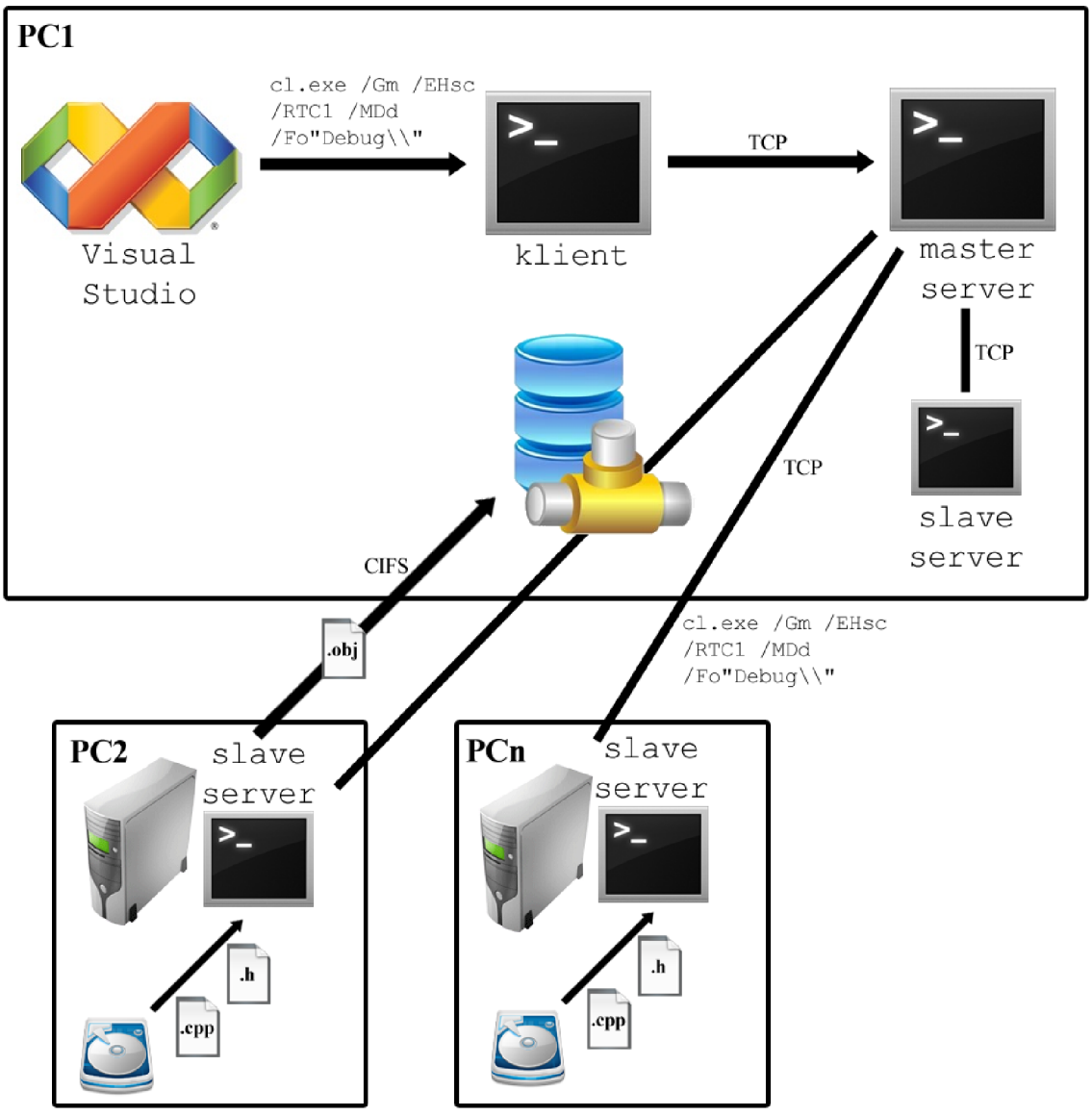
`RETURN_CODE` je číslo určující návratovou hodnotu, kterou vrátil překladač (např. 127).

3.4 Systém 2 - sestavení na lokálních discích

3.4.1 Architektura

Systém 2 vyžaduje, aby měly všechny počítače v systému umístěnu na lokálním disku identickou verzi zdrojových souborů. Překlad jednotlivých projektů je potom rozdělen na více počítačů, které zdrojové soubory čtou z lokálního disku a výstupy překladu ukládají do adresáře na hostitelském počítači sdíleném přes CIFS.

Architekturu tohoto systému ilustruje diagram 3.2.



Obrázek 3.2: Architektura systému 2

Tento systém se dělí na tři části, podobně jako systém v podkapitole 3.3 - klient, master server a slave server. Jejich úloha je také obdobná, dochází jen ke změnám při přepisování

cest ke zdrojovým souborům a výstupním adresářům.

Aby tento systém korektně fungoval, je třeba, aby při překladu nedocházelo ke změnám zdrojových souborů. Pokud jsou tyto změny nutné, je třeba je provést před začátkem překladu, a to na všech počítačích v systému. Mezi typické změny zdrojových souborů patří např. generování hlavičkových a jazykových souborů z XML souborů.

3.4.2 Klient

Jako u předchozího systému nahrazuje klient soubor překladače (`cl.exe`), zpracuje parametry příkazové řádky, expanduje response files a výsledná data pošle master serveru. Master server jako odpověď vrátí texty vypsané překladačem, které jsou vypsány na standardní výstup a chybový výstup, a klient ukončí činnost s návratovou hodnotou taktéž vrácenou master serverem.

3.4.3 Master server

Master server je, stejně jako v předchozím systému, aplikace, která přijímá požadavky na překlad od klientů a přeposílá je slave serverům. Stará se také o nahrazování lokálních cest, nastavuje maximální počet připojení na jeden slave server, přidává k výstupním cestám UNC cestu sdíleného adresáře ap.

Na rozdíl od předchozího systému je však potřeba nahradit větší množství cest v pracovním adresáři a parametrech překladače. Protože chceme, aby překlad probíhal z lokálního disku, ale výsledky se ukládaly do sdíleného adresáře na hostitelském počítači, je potřeba:

1. Nahradit cestu k pracovnímu adresáři na lokálním souborovém systému hostitelského systému za cestu na souborovém systému počítače provádějícího překlad.
2. Upravit parametry příkazové řádky tak, aby se výsledky překladu ukládaly do sdíleného adresáře na hostitelském počítači.

Prvního bodu je možno dosáhnout pomocí už existujícího konfiguračního příkazu `replace`, kvůli druhé volbě je potřeba v konfiguračním souboru přidat volbu

```
replace=?original*replacement
```

Tento řádek říká, že se má v proměnné `netpath`, která obsahuje cestu k pracovnímu adresáři poslanou klientem, nahradit text `original` za `replacement`. Tato proměnná je potom přidána před cestu v některých parametrech příkazové řádky:

1. `/Fa"..."` Tento parametr určuje cestu, kam budou uloženy soubory s asemblerem, pokud je použit přepínač `/FA`
2. `/Fd"..."` Tento parametr určuje cestu, kam budou uloženy soubor s programovou databází (`.pdb`)
3. `/Fo"..."` Tento parametr určuje cestu, kam budou uloženy objektové soubory (`.obj`)
4. `/Fp"..."` Tento parametr určuje cestu, kam budou uloženy prekompilované hlavičkové soubory (`.pch`)
5. `/FR"..."` Tento parametr určuje cestu, kam budou uloženy `.sbr` soubory potřebné pro utilitu BSCMAKE

Příkladem konfiguračního souboru je následující text:

```
replace=?D:\svn\*\\shiina\svn\
replace=D:\svn\*?LOCAL?
server=localhost
server=b02
max_processes=4
```

Tento soubor způsobí při přijmutí požadavku na překlad v proměnné `netpath` nahrazení textu `"D:\svn\"` za `"\\shiina\svn\"`, a tato proměnná je poté přidána jako prefix k výše uvedeným parametrům příkazové řádky. Poté je v cestě k pracovnímu adresáři a v parametrech nahrazeno `"D:\svn\"` za `"?LOCAL?"`. Jsou přidány slave servery `localhost` a `b02` a je omezen počet požadavků na překlad na jednom slave serveru na 4.

Požadavek od klienta obsahující následující data:

```
WORKING_DIRECTORY: D:\svn\system1\client
PAYLOAD: /Od /D "WIN32" (...) /Fo"Debug\" /Fd"Debug\vc80.pdb" (...)
".\client.cpp" /nologo /errorReport:prompt
```

je tedy zaměněn za:

```
WORKING_DIRECTORY: ?LOCAL?system1\client
PAYLOAD: /Od /D "WIN32" (...) /Fo"\\shiina\svn\system1\client\Debug\"
/Fd"\\shiina\svn\system1\client\Debug\vc80.pdb" (...) ".\client.cpp"
/nologo /errorReport:prompt
```

Nahrazení za zástupný řetězec `"?LOCAL?"` je zvoleno kvůli unikátnosti řetězce (cesta nesmí obsahovat znak `"?"`. Slave server tento řetězec později nahradí za cestu ke zdrojovým souborům na lokálním souborovém systému.

3.4.4 Slave server

Jako v předchozím systému, slave server přijímá požadavky na překlad od master serveru, spouští proces překladače (`cl.exe`), ukládá standardní výstup a posílá ho zpět master serveru a má také nastaven maximální počet současně běžících procesů překladu.

Oproti předchozímu systému však musí slave server nastavit cestu k pracovnímu adresáři na cestu ke zdrojovým souborům na lokálním disku. Toho lze dosáhnout pomocí záznamu v konfiguračním souboru `replace=original*replacement` (obdobná funkčnost jako u master serveru), který nahradí zástupný řetězec vložený master serverem za lokální cestu.

Pro náš příklad by konfigurační soubor vypadal takto:

```
replace=?LOCAL?*C:\build\
max_processes=4
```

Požadavek na překlad by tedy po změně obsahoval:

```
WORKING_DIRECTORY: C:\build\system1\client
PAYLOAD: /Od /D "WIN32" (...) /Fo"\\shiina\svn\system1\client\Debug\"
/Fd"\\shiina\svn\system1\client\Debug\vc80.pdb" (...) ".\client.cpp"
/nologo /errorReport:prompt
```

Další postup je již stejný jako u systému v podkapitole 3.3 - slave server přeloží požadované soubory a výsledek odešle zpět master serveru.

3.4.5 Komunikační protokol

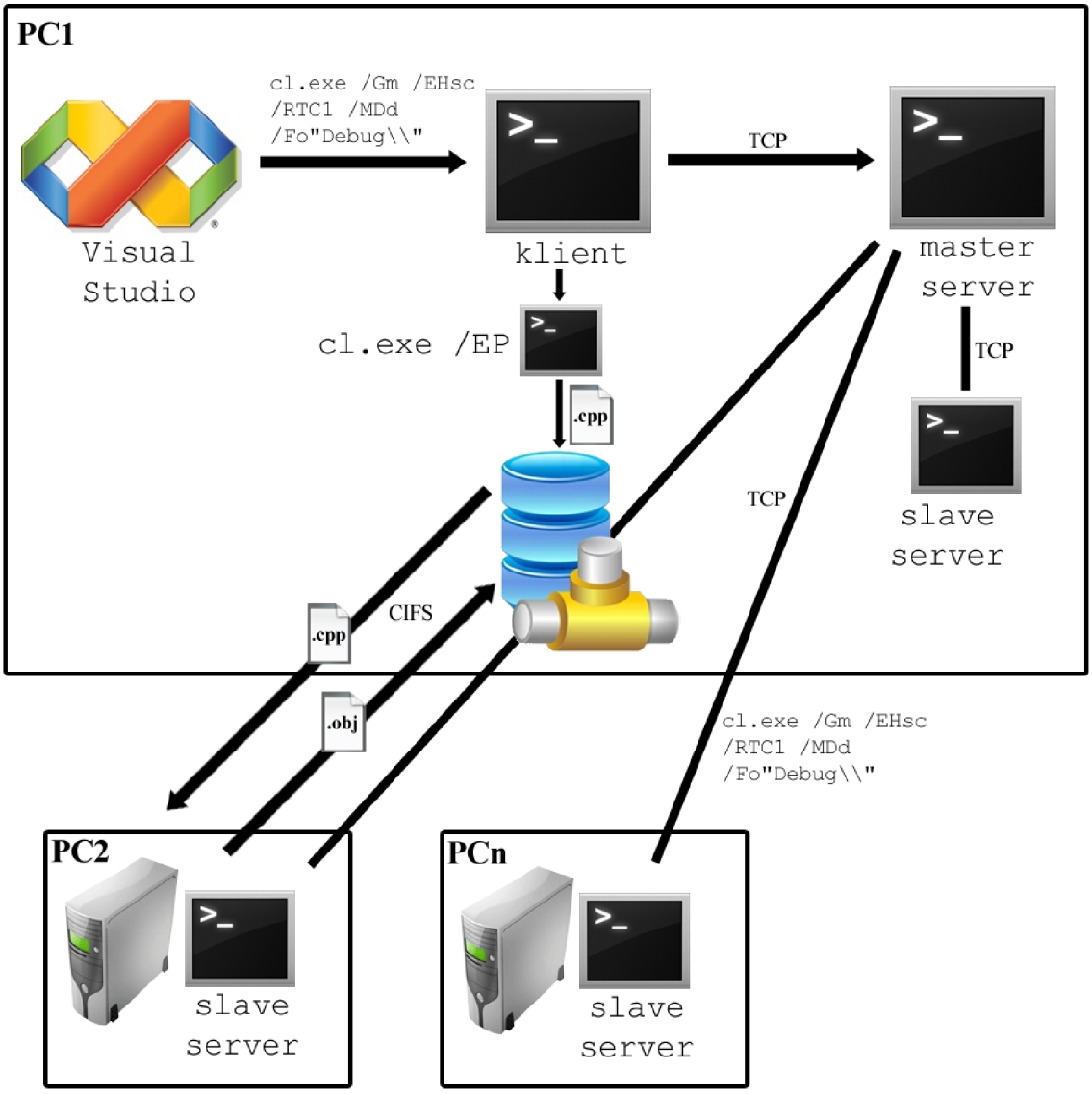
Komunikační protokol je stejný, jako u systému v podkapitole 3.3.

3.5 Systém 3 - sestavení z preprocesovaných zdrojových souborů

3.5.1 Architektura

Tento vyžaduje, aby byl adresář se zdrojovými kódy sdílen přes CIFS a přístupný ze všech počítačů v systému. Na rozdíl od systému 1 v podkapitole 3.3 však vzdálené počítače ze sdíleného adresáře čtou soubory už zpracované preprocesorem, u kterých už není potřeba přenášet potřebné hlavičkové soubory, expandovat direktivy podmíněného překladu, makra ap. Výsledky překladu jsou opět ukládány do sdíleného adresáře.

Architekturu tohoto systému ilustruje následující diagram:



Obrázek 3.3: Architektura systému 3

Tento systém se dělí na stejné části, jako předchozí systémy - klient, master server a

slave server. Jejich úloha je také obdobná, jen klient před odesláním požadavku na překlad master serveru zpracuje zdrojové soubory preprocesorem.

3.5.2 Klient

Také v tomto systému klient nejdříve zpracuje parametry příkazové řádky a expanduje response files, potom ale každý soubor, který se má přeložit, zpracuje preprocesorem. Toho se docílí spuštěním překladače `cl.exe` s parametrem `/EP` navíc. Upravený zdrojový kód se poté uloží do dočasného souboru a jméno původního souboru v parametrech překladu odesílaných master serveru se nahradí jménem dočasného souboru.

Pro zpracování všech zdrojových souborů preprocesorem se požadavek na překlad odešle master serveru.

3.5.3 Master server

Zde master server koná stejnou činnost jako v podkapitole 3.3 - nahrazuje lokální cestu k pracovnímu adresáři za UNC cestu a přeposílá požadavky na překlad slave serverům.

3.5.4 Slave server

Jako v podkapitole 3.3 slave server už nic nenahrazuje, pouze přeloží požadavky a vrátí master serveru výsledek.

3.5.5 Komunikační protokol

Komunikační protokol je stejný, jako u předchozích systémů.

3.6 Shrnutí

Všechny tři systémy využívají společný síťový protokol a stejný koncept klient - master server - slave servery. Všechny tři tyto části jsou navrženy tak, aby byly co nejobecnější a nejvíce rozšiřitelné. Síťový protokol je také schopný přenášet obecné požadavky a odpovědi na ně.

V ranné fázi implementace se pomocí síťového protokolu přenášely i požadavky na linkování výstupních souborů. Implementoval jsem aplikaci *linker*, která podobně jako aplikace klient nahrazuje `cl.exe` nahrazovala linker (`link.exe`) a distribuovala linkování na vzdálené počítače. Formát požadavku na linkování byl podobný, jako požadavky na překlad, tyto pokusy však byly zastaveny, protože přenesení výstupních souborů po síti, jejich slinkování a přenesení zpět trvalo několikanásobně déle, než linkování na lokálním počítači.

Systémy jsou navrhnuty ke spolupráci s Microsoft C/C++ Compiler, lze je však téměř beze změn použít i spolu s překladačem Intel C++ Compiler. Podpora dalších překladačů je možná, bylo by však třeba analyzovat parametry příkazové řádky, které jsou závislé na souborovém systému a je potřeba je nahradit. Příkladem je např. přepínač `-o` u překladače GCC, který je ekvivalentem přepínače `/Fo` u Microsoft C/C++ Compileru.

Kvůli použití Windows API pro síťovou komunikaci (Windows Sockets 2), tvorbu procesů (funkce CreateProcess) a vláken (funkce CreateThread) jsou systémy limitovány na platformu MS Windows.

Systémy nejsou náročné na výpočetní výkon, ale mohou způsobovat velké datové přenosy. Více o nárocích na zdroje v kapitole 4.

Kapitola 4

Experimenty

4.1 Postup při experimentech

V kapitole 3 jsem popsal tři systémy, které ilustrují různé přístupy k paralelizaci sestavení. Zde popíši testy, které zhodnotí silné a slabé stránky jednotlivých systémů a jejich efekt na zrychlení procesu sestavení.

Díky laskavému souhlasu firmy AVG Technologies CZ, s.r.o. jsem měl možnost provést testy systémů na paralelizaci sestavení na velmi velkém souboru dat (antiviru AVG). Tato aplikace zahrnuje 600 MB dat, 200 projektů a 30 000 zdrojových souborů.

Sestavení probíhalo na platformě Microsoft Visual Studio 2005 a Windows XP.

Experimenty byly prováděny na počítačích s následující konfigurací:

CPU Intel Core 2 Duo E6750 @ 2.66 GHz
2-3GB DDR RAM
7200RPM HDD

Počítače byly propojeny sítí o rychlosti 1Gb/s přes switch od firmy SMC.

Nejdříve byla provedena 3 sestavení s paralelizací pouze v rámci jednoho počítače. Počet paralelně sestavovaných projektů byl nastaven na 4. Vzorec pro výpočet efektivních procesorů je [9]

$$E = F \cdot J \cdot H \tag{4.1}$$

kde E je počet efektivních procesorů, F počet fyzických procesorů, J počet jader na fyzickém procesoru a H počet efektivních procesorů za jádro kvůli hyperthreadingu. To pro použité počítače dává výsledek

$$E = 1 \cdot 2 \cdot 1 = 2 \tag{4.2}$$

Pokusy ukázaly, že nejrychlejší sestavení proběhlo, pokud běžely 2 procesy na každý efektivní procesor.

Pokus	Doba sestavení	Průměrné vytížení CPU
Pokus 1	18:20	80%
Pokus 2	17:54	83%
Pokus 3	19:46	71%
Průměr	18:40	78%

Tabulka 4.1: Doba sestavení a využití CPU na jednom počítači

Průměrná doba sestavení na jednom počítači je tedy asi 18 minut a 40 sekund. Experimenty na systémech z kapitoly 3 prováděné v této kapitole byly opakovány 3x a porovnány s průměrnou dobou sestavení a zatížení CPU uvedenými výše.

4.2 Systém 1

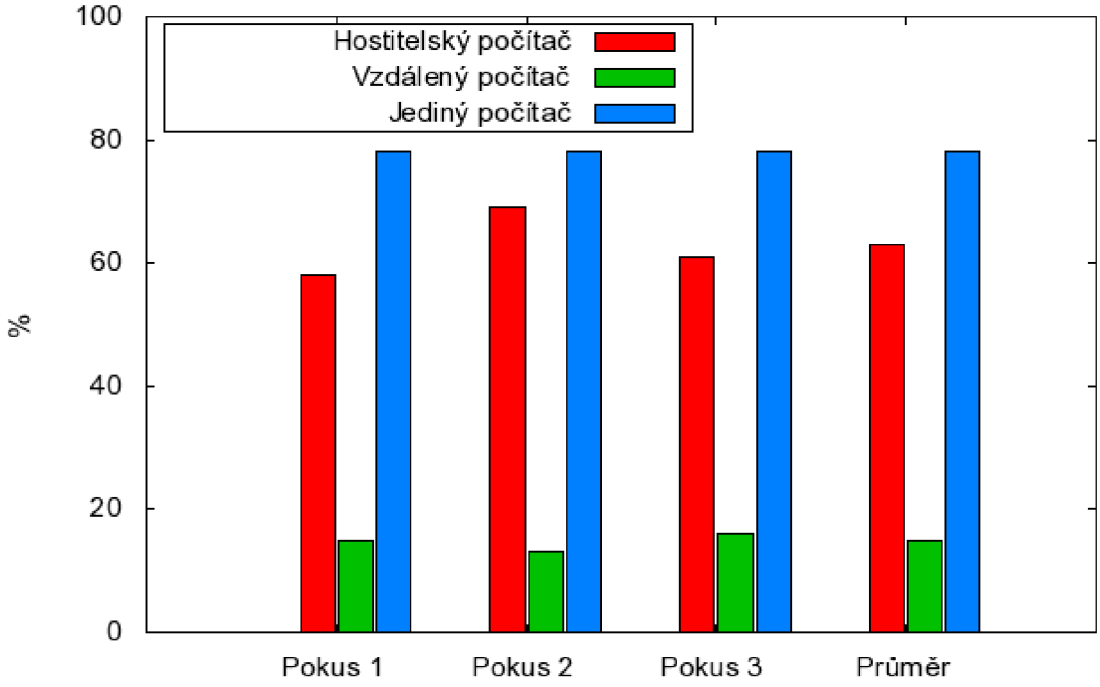
Systém 1 představený v podkapitole 3.3 byl testován na dvou počítačích: hostitelském, kde běžel klient, master server a slave server, a vzdáleném, kde běžel slave server. Překlad probíhal na obou počítačích, bylo monitorováno vytížení CPU a byl změřen celkový čas sestavení. Proběhly 3 experimenty.

Jak lze vidět z tabulky 4.2, čas sestavení s využitím tohoto systému je pomalejší, než při sestavení na jediném počítači, a to asi o 32%. Průměrný čas sestavení je tu asi 24 minut a 45 sekund. Graf 4.1 ukazuje zatížení CPU obou počítačů ve srovnání s průměrným zatížením CPU při sestavení na jediném počítači. Zvláště u vzdáleného počítače je vytížení velmi nízké.

Nízké využití CPU u vzdáleného počítač bylo zapříčiněno hlavně dlouhým čekáním na přenesení zdrojových souborů z hostitelského počítače; průměrný přenos dat byl více než 150Mb/s. Další zpomalení na obou počítačích bylo způsobeno čekáním na sestavení projektů, na kterých závisely další projekty. Při sestavení těchto projektů na vzdáleném počítači trvalo čekání mnohem delší dobu, než při sestavení na hostitelském počítači, což vysvětluje relativně velké rozdíly v době sestavení (při překladu těchto projektů na vzdáleném počítači se zvětšovala neparalelizovatelná část).

Pokus	Doba sestavení
Pokus 1	26:21
Pokus 2	22:47
Pokus 3	25:08
Průměr	24:45

Tabulka 4.2: Doba sestavení při sestavení na 2 počítačích s využitím systému 1



Obrázek 4.1: Využití CPU při sestavení na 2 počítačích s využitím systému 1

4.3 Systém 2

Systém 2 představený v podkapitole 3.4 byl opět testován na dvou počítačích: hostitelském, kde běžel klient, master server a slave server, a vzdáleném, kde běžel slave server. Překlad probíhal na obou počítačích, bylo monitorováno vytížení CPU a byl změřen celkový čas sestavení. Proběhly 3 experimenty.

Z tabulky 4.3 lze vidět, že s využitím tohoto systému bylo dosaženo průměrné doby sestavení asi 13 minut, což je oproti sestavení na jednom počítači o 30% kratší doba.

Podle rovnice Amdahlova zákona

$$S = \frac{1}{(1-P) + \frac{P}{N}} \tag{4.3}$$

kde P je paralelizovatelná část, N je počet počítačů a S je zrychlení, je maximální zrychlení pro $N = 2$ počítačů a paralelizovatelnou část $p = 1$

$$S = \frac{1}{(1-1) + \frac{1}{2}} = 2 \tag{4.4}$$

U tohoto systému došlo k zrychlení asi 1.3, což ze stejného vzorce odpovídá paralelizovatelné části

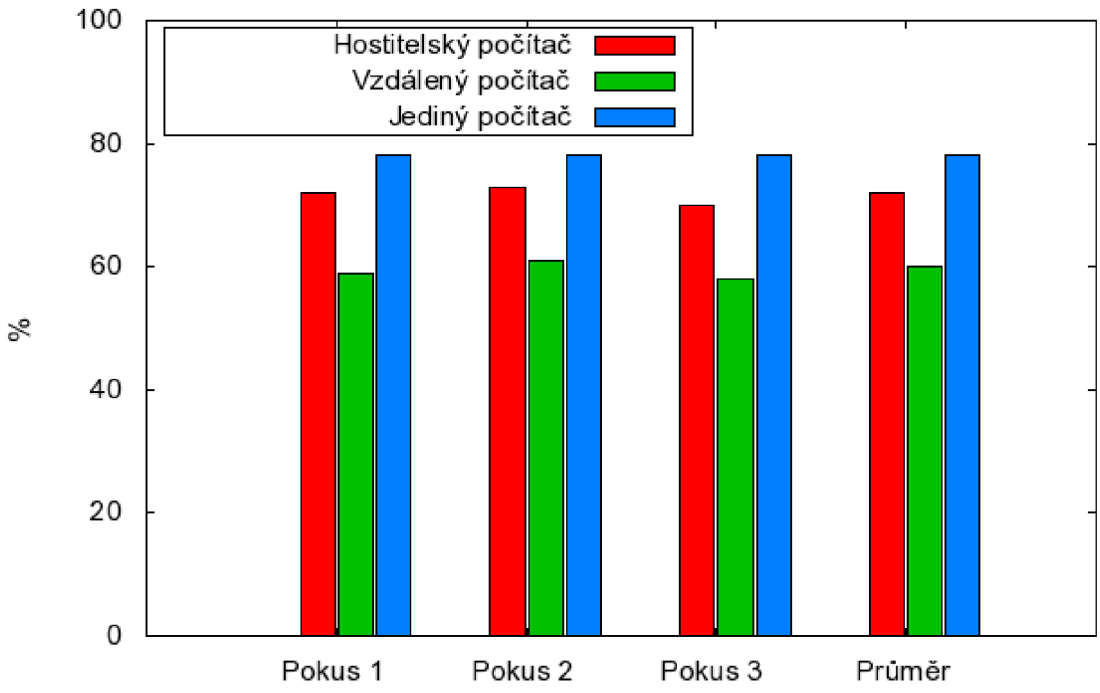
$$P = \frac{\frac{1}{S} - 1}{\frac{1}{N} - 1} = 46\% \tag{4.5}$$

Neparalelizovatelnou část tvoří zřejmě:

1. Před spuštěním paralelního sestavení bylo potřeba na obou počítačích spustit projekty, které vygenerovaly hlavičkové a resource soubory z XML souborů.
2. Projekty, na kterých jsou závislé další projekty, tvoří neparalelizovatelnou část, protože je třeba je sestavit dříve, než je možné začít sestavovat na nich závislé projekty.
3. Paralelizován je jen překlad, linkování probíhá pouze na hostitelském počítači.

Pokus	Doba sestavení
Pokus 1	12:59
Pokus 2	12:23
Pokus 3	13:37
Průměr	13:00

Tabulka 4.3: Doba sestavení při sestavení na 2 počítačích s využitím systému 2



Obrázek 4.2: Využití CPU při sestavení na 2 počítačích s využitím systému 2

4.4 Systém 3

Systém 3 představený v podkapitole 3.4 byl také testován na dvou počítačích, tentokrát však proběhly dvě skupiny experimentů. Ve skupině 1 probíhal překlad na obou počítačích, ve skupině 2 probíhal překlad pouze na vzdáleném počítači a hostitelský počítač se věnoval pouze prekompilaci a linkování. U obou počítačů bylo monitorováno vytížení CPU a byl změřen celkový čas sestavení. U každé skupiny testů proběhly 3 experimenty.

Jak lze vidět z tabulky 4.4, u první skupiny bylo dosaženo průměrné doby sestavení 25 minut a 33 sekund, což je oproti sestavení na jednom počítači zpomalení o 38%. U druhé skupiny (graf 4.5) bylo dosaženo průměrné doby sestavení 20 minut a 35 sekund, což je oproti sestavení na jednom počítači zpomalení o 10%. Na grafu 4.3 lze vidět, že prekompilace dokáže vytížit celé CPU hostitelského počítače.

I když se tento systém jevil jako potenciálně perspektivní, při testech s přepínači `/P` a `/EP` programu `cl.exe` (získání výstupu preprocesoru) bylo zjištěno, že získání výstupu prekompilovaných zdrojových kódů trvá u Microsoft C/C++ Compiler až 3x déle, než samotný překlad. Tím se tato varianta stává velmi nepraktickou; testoval jsem tedy překladač Intel C++ Compiler 11.0 (`icl.exe`), u kterého je běh preprocesoru asi 3x rychlejší, než samotný překlad. Nabízí se tedy možnost buď použít místo překladače firmy Microsoft překladač firmy Intel, nebo překladač firmy Intel použít jen na prekompilaci.

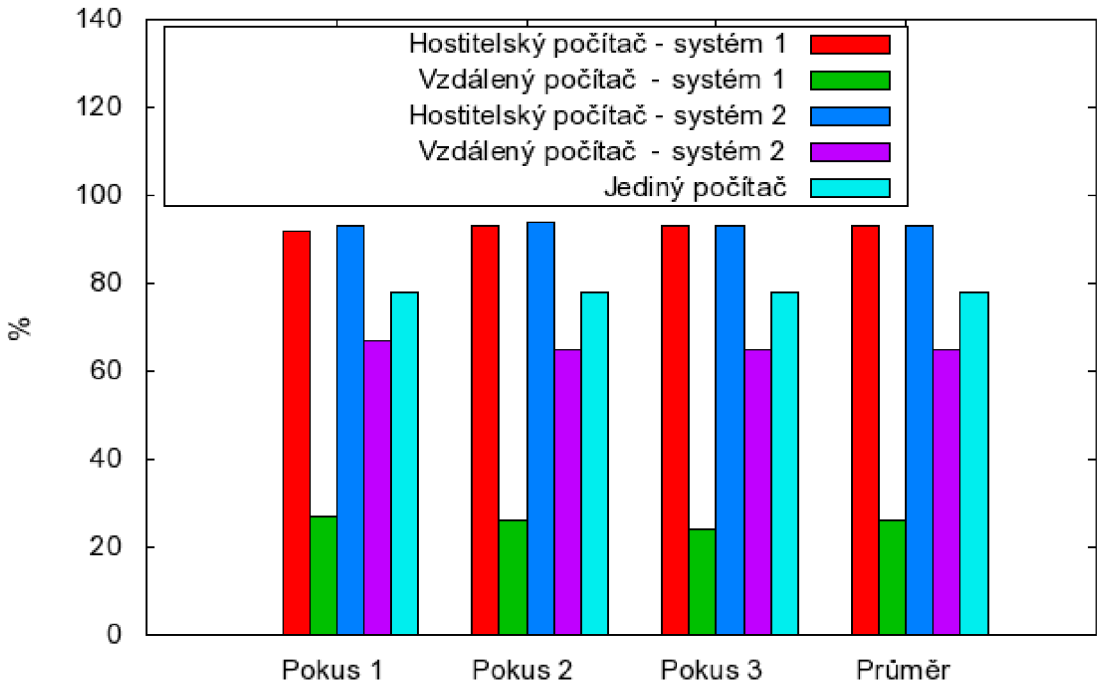
Kvůli jistým odlišnostem mezi Intel C++ Compiler a Microsoft C/C++ Compiler, které jsou jinak z velké části navzájem kompatibilní, se mi nepodařilo aplikaci sestavit, takže jsem nemohl otestovat chování systému 3 s překladačem Intel C++ Compiler.

Pokus	Doba sestavení
Pokus 1	25:24
Pokus 2	25:12
Pokus 3	26:03
Průměr	25:33

Tabulka 4.4: Doba sestavení při sestavení na 2 počítačích s využitím systému 3, skupina 1

Pokus	Doba sestavení
Pokus 1	20:27
Pokus 2	20:31
Pokus 3	20:48
Průměr	20:35

Tabulka 4.5: Doba sestavení při sestavení na 2 počítačích s využitím systému 3, skupina 2



Obrázek 4.3: Využití CPU při sestavení na 2 počítačích s využitím systému 3

4.5 Shrnutí

Jak lze vidět z tabulky 4.6, výsledky jednotlivých systémů se od sebe značně liší.

Systém 1 lze označit za neperspektivní - i když byly počítače propojeny 1Gb/s sítí, přenos dat z hostitelského počítače byl příliš pomalý a celková doba sestavení se prodloužila. Jistého zrychlení by bylo možno dosáhnout transparentním cachováním souborů (hlavně hlavičkových) na straně vzdáleného počítače, protože pak je třeba je přenášet pro každý zdrojový soubor, který je inkluduje. I přes toto opatření by však celkový čas překladu byl pravděpodobně větší, než na jediném počítači.

Systém 2 dosáhl celkem uspokojivého zrychlení, které je však vykoupeno větší složitostí přípravy na sestavení. Je třeba distribuovat zdrojové soubory na všechny počítače účastnící se překladu a v případě, že dochází ke změnám zdrojových souborů, je potřeba tyto změny provést před začátkem sestavení. To zahrnuje manuální změny, případně údržbu dalšího projektu, který se postará o vygenerování souborů.

Při použití Microsoft C/C++ Compiler a systému 3 se celkový čas sestavení prodloužil průměrně o 38%, za což může velmi pomalé získání výstupu preprocesoru. U Intel C++ Compiler trvá získání výstupu preprocesoru asi 3x rychleji, než samotný překlad, bylo by tedy teoreticky možné ho použít na prekompilaci. Vzniká zde ale problém, že Intel C++ Compiler a Microsoft C/C++ Compiler nejsou zcela kompatibilní, takže je v některých případech potřeba upravit zdrojové soubory, aby je bylo možné přeložit pomocí Intel C++ Compiler.

Systém	Průměrná doba sestavení
Systém 1	24:45
Systém 2	13:00
Systém 3	25:33
Jediný počítač	18:40

Tabulka 4.6: Srovnání průměrných časů sestavení

Kapitola 5

Závěr

V této bakalářské práci jsem prozkoumal možnosti paralelizace překladu a sestavování na jednom i více počítačích na platformě MS Windows. Seznámil jsem se s procesem překladu a sestavování a s nástroji pro jejich automatizaci, s historií a vývojem těchto nástrojů, s jejich principy a společnými znaky. Dále jsem prozkoumal možnosti těchto nástrojů v oblasti paralelizace překladu a seznámil se s existujícími nástroji pro paralelizaci překladu a sestavení na více počítačích.

V druhé části bakalářské práce jsem podle zadání rozebral vhodnost jednotlivých nástrojů s ohledem na paralelizaci a diskutoval jsem tři přístupy pro paralelizaci překladu na více počítačích s ohledem na efektivitu a náročnost na zdroje, hlavně síťové propojení. Tři systémy využívající tyto rozdílné přístupy jsem implementoval v jazyce C++.

Závěrečná část mé práce je věnována experimentům s jednotlivými systémy pro paralelizaci se zaměřením na rychlost překladu, stupeň paralelizace a vytížení systému. Porovnal jsem vhodnost jednotlivých systémů, využití zdrojů počítačů účastnících se sestavení, určil jsem stupeň paralelizace překladu sestavované aplikace a zdůvodnil jsem příčiny vzniku neparalelizovatelné části.

První systém se ukázal neperspektivním, protože se sestavení pomoci něho prodloužilo o 32% ve srovnání se sestavením na jediném počítači. Toto bylo způsobeno dlouhým časem přenosu dat na vzdálený počítač. Druhý systém dosáhl uspokojivého urychlení sestavení, a to o 30% při překladu na dvou počítačích. Třetí systém prodloužil čas sestavení až o 38%, což je zapříčíněno pomalým získáním výstupu preprocesoru u překladače Microsoft C/C++ Compiler. Toto omezení by bylo možné odstranit použitím Intel C++ Compiler, kde trvá zíslání výstupu preprocesoru několikanásobně kratší dobu.

V průběhu řešení bakalářské práce vznikly tři systémy, které demonstrují různé přístupy k paralelizaci překladu a sestavování programových modulů na více počítačích. Z těchto systémů by se dalo vycházet při návrhu a implementaci složitějších nástrojů, které by stavěly na silných stránkách jednotlivých systémů a vytvořily robustnější platformu pro paralelizaci sestavování s menší závislostí na překladači a operačním systému.

Literatura

[1] Doar, M.: *Practical Development Environments*. O'Reilly, 2005, ISBN 0-596-00796-5.

[2] von Hagen, W.; Jones, B.: *Linux Server Hacks, Volume Two*. O'Reilly, 2005, ISBN 0-596-10082-5.

[3] Hashimi, S. Y.; Hashimi, S. I.: *Deploying .NET Applications: Learning MSBuild and ClickOnce*. Apress, 2006, ISBN 1-59059-652-8.

[4] Parsons, A.; Randolph, N.: *Professional Visual Studio 2005*. Wiley, 2006, ISBN 0-7645-9846-5.

[5] WWW stránky: Building Multiple Projects in Parallel. `http://msdn.microsoft.com/en-us/library/bb651793.aspx`, [Online; navštíveno 12.5.2009].

[6] WWW stránky: High Performance Unicode Text File I/O Routines for C++. `http://www.codeproject.com/KB/files/EZUTF.aspx`, [Online; navštíveno 12.5.2009].

[7] WWW stránky: IncrediBuild by Xoreax Software - High-Performance Grid Engine for Distributed Builds, Scripts and Applications. `http://www.xoreax.com/`, [Online; navštíveno 12.5.2009].

[8] WWW stránky: Microsoft C and C++ Compiler Options. `http://msdn.microsoft.com/en-us/library/19z1t1wy(VS.71).aspx`, [Online; navštíveno 12.5.2009].

[9] WWW stránky: /MP (Build with Multiple Processes). `http://msdn.microsoft.com/en-us/library/bb385193.aspx`, [Online; navštíveno 12.5.2009].

[10] WWW stránky: Unicode Support in the Compiler and Linker. `http://msdn.microsoft.com/en-us/library/xwy0e8f2.aspx`, [Online; navštíveno 12.5.2009].

Seznam příloh

Příloha A

Slovník pojmů

CIFS Common Internet File System Protocol je protokol firmy Microsoft vycházející z protokolu SMB. Používá se ke sdílení souborů primárně na platformě Windows.

COFF Common Object File Format je standardní souborový formát pro uložení spustitelných souborů, objektového kódu a sdílených knihoven používaný na UNIXových systémech a jako varianta PE/COFF na systémech Windows. Na UNIXových systémech byl z velké části nahrazen formátem ELF.

CVS Concurrent Versions System je systém, který slouží ke správě verzí projektu. Monitoruje změny na souborech a umožňuje více vývojářům spolupracovat na jednom projektu.

ELF Executable and Linkable Format je standardní souborový formát pro uložení spustitelných souborů, linkovatelných objektů, sdílených knihoven a ladících výpisů. Je používán hlavně v systémech Linux, Solaris, IRIX nebo FreeBSD.

EZUTF High Performance Unicode Text File I/O Routines for C++, sada funkcí pro čtení souborů v kódování Unicode (UTF-16 i UTF-8) i ASCII.

GCC GNU Compiler Collection je kolekce překladačů pro jazyky C, C++, Java, Ada, Objective-C, Objective-C++ a Fortran

PDB Typ souboru PDB (program database) obsahuje ladící informace a informace o projektu. Soubor je vytvořen, pokud dojde k překladu pomocí Microsoft C/C++ Compiler s přepnačem /ZI nebo /Zi.

TCP Transmission Control Protocol je jedním ze základních protokolů sady protokolů Internetu a pracuje na transportní vrstvě. Protokol garantuje spolehlivé doručování zpráv a doručování ve správném pořadí.

UNC Uniform Naming Convention je formát specifikující zápis umístění síťového zdroje, jako je sdílený soubor, adresář nebo tiskárna. Používá se hlavně na platformě Windows.

XML Extensible Markup Language je obecný značkovací jazyk umožňující snadné vytváření konkrétních značkovacích jazyků pro různé účely a široké spektrum různých typů dat.

Příloha B

Obsah CD

CD obsahuje implementované systémy v adresářích `SYSTEM1`, `SYSTEM2` a `SYSTEM3`, a solution na testování těchto systémů v adresáři `TEST`. V kořenovém adresáři se také nachází soubor `README.txt`, který obsahuje následující instrukce k použití jednotlivých systémů:

Tento návod počíta s nainstalovaným Microsoft Visual Studiem 2005. Pro sestavení testovacího solutionu, který se nachází v adresáři TEST, je třeba provést několik úkonů závislých na systému, pomocí kterého budete sestavovat. Předtím je však třeba na všech počítačích účastnících se překladu zkopírovat soubor překladače `cl.exe` typicky se nacházející v `[DISK]:\Program Files\Microsoft Visual Studio 8\VC\bin\` do souboru `cl2.exe`. Původní soubor překladače bude na iniciujícím počítači přepsán agentní aplikací systému, slave server proto bude při překladu volat `cl2.exe`. Obsah CD je třeba zkopírovat na lokální disk do adresáře a tento adresář nasdílet pomocí CIFS s právy na zápis. Je nutné se ujistit, že k tomuto adresáři mohou přistupovat všechny počítače účastnící se překladu. Na všechny počítače účastnící se sestavení je třeba zkopírovat do jednoho adresáře soubory `SYSTEMx\debug\slave_server.exe` a `slave_server.conf`, kde znak x je číslo použitého systému.

Systém 1
Systém 1 se nachází v adresáři `SYSTEM1`. Nejdříve je třeba na počítači iniciujícím překlad editovat soubor `SYSTEM1\debug\master_server.conf`:

```
replace=D:\project\*\\shiina\project\
server=localhost
server=b01
max_processes=8
```

Cestu `D:\project\` je třeba změnit za cestu k adresáři s obsahem CD a cestu `\\shiina\project\` za UNC cestu k tomuto adresáři sdílenému přes CIFS. Pro každý počítač účastnící se sestavení (a na kterém běží slave server) je třeba přidat řádek `server=<adresa počítače>`.

Systém 2
Systém 2 se nachází v adresáři `SYSTEM2`. Nejdříve je třeba na počítači iniciujícím překlad editovat soubor `SYSTEM2\debug\master_server.conf`:

```
replace=?D:\project\*\\shiina\project\
replace=D:\project\*?LOCAL?
server=localhost
server=b01
max_processes=8
```

Cestu `D:\project\` je třeba změnit za cestu k adresáři s obsahem CD a cestu `\\shiina\project\` za UNC cestu k tomuto adresáři sdílenému přes CIFS. Pro každý počítač účastnící se sestavení (a na kterém běží slave server) je třeba přidat řádek `server=<adresa počítače>`. Na všech počítačích účastnících se sestavení je třeba editovat soubor `slave_server.conf`:

```
replace=?LOCAL?*D:\proj\
max_processes=4
```

Cestu `D:\proj\` je třeba nahradit za cestu k adresáři, kam bude zkopírován adresář `TEST` nacházející se na CD.

Systém 3

Systém 3 se nachází v adresáři `SYSTEM3`. Nejdříve je třeba na počítači iniciujícím překlad editovat soubor `SYSTEM3\debug\master_server.conf`:

```
replace=D:\project\*\\shiina\project\
server=localhost
server=b02
max_processes=8
```

Cestu `D:\project\` je třeba změnit za cestu k adresáři s obsahem CD a cestu `\\shiina\project\` za UNC cestu k tomuto adresáři sdílenému přes CIFS. Pro každý počítač účastnící se sestavení (a na kterém běží slave server) je třeba přidat řádek `server=<adresa počítače>`.

Pro všechny systémy

Dále je třeba na počítači iniciujícím překlad nahradit soubor překladače `cl.exe` (typicky se nacházející v `[DISK]:\Program Files\Microsoft Visual Studio 8\VC\bin\`) agentní aplikací `client.exe`, která se nachází v podadresáři `SYSTEMx\debug`. Pro všechny systémy je poté třeba na iniciujícím počítači spustit `SYSTEMx\debug\master_server.exe` a na všech počítačích účastnících se sestavení spustit dávkový soubor *START → All Programs → Microsoft Visual Studio 2005 → Visual Studio Tools → Visual Studio 2005 Command Prompt* a v něm spustit `slave_server.exe`. Znak x je číslo použitého systému.

Po nastavení systému stačí na iniciujícím počítači otevřít solution `TEST\test_solution.sln` ve Visual Studiu a sestavit ho stisknutím klávesy F7. Počet paralelně sestavovaných projektů lze nastavit v nabídce Visual Studia *Tools → Options → Projects and Solutions → Build and Run → maximum number of parallel project builds.*