Kompilace a Makefile.
OSD

O. Fišer

April 18, 2011

Obsah

Compiler - překladač

- program, který vezme zdrojový text programu a přeloží jej do jazyka stroje, což jsou obvykle kódy instrukcí pro daný procesor.
- Výsledkem tohoto překladu obvykle není spustitelný program, protože v místech volání podprogramů z jiných modulů a knihoven nejsou konkrétní adresy těchto podprogramů, ale pouze značky, které se zpracují později při linkování.
- Souborům, které produkuje překladač se říká objektové soubory (nemá to nic společného s objektově orientovaným programováním).
- Důležitou funkcí překladače je syntaktická kontrola.

Linker - sestavení programu

- slouží k sestavení samostatně přeložených modulů a knihoven do funkčního celku.
- Linker do kódu doplní konkrétní adresy podprogramů (statický překlad) nebo kód pro jejich zavolání (při použití dynamických knihoven).
- Pokud vytváříme spustitelný program (u knihoven to je trochu jinak), linker nakonec doplní do programu kód, který se používá při spuštění programu. Tato část, stejně jako formát použitých statických nebo dynamických knihoven se liší v závislosti na použitém operačním systému a hardwarové platformě. Pokud proto chceme vyrobit spustitelný program pro jinou platformu, musíme u jazyků typu C, C++ provést překlad na zvolené platformě znovu.

Debugger

- program pro ladění a hledání chyb v programu.
- Aby mělo ladění s debuggerem smysl, je obvykle nutné říci překladači, aby do výsledného kódu přidal speciální značky, díky nimž si debugger bude umět spojit konkrétní část binárního kódu programu s konkrétními řádky zdrojového textu.

Preprocessor

- speciální program používaný v jazycích C a C++ pro zpracovávání maker a symbolických konstant.
- Tento program v podstatě provádí pouze textové náhrady definovaných symbolů v textu a umožnuje podmíněný překlad. Příkazy pro preprocessor začínají vždy znakem mřížky (#), například #define nebo #ifndef. Příkazy preprocessoru byste měli používat co nejméně.

Assembler

- program pro překlad jazyka symbolických instrukcí do binárního kódu. Někdy bývá nesprávně zaměňován se samotným jazykem symbolických instrukcí, což je vemi nízkoúrovňový jazyk, kde zdrojový kód tvoří posloupnost zkratek instrukcí procesoru.
- Při velmi speciálních příležitostech lze jazyk symbolických instrukcí kombinovat s kódem v jazyce C. Velmi speciální situace zahrnují přístup k nejnižším úrovním hardwaru a brutální optimalizace. Při běžném programování se s tím nesetkáte (a ani by to většinou nebylo prospěšné).

Syntaktická chyba

- prohřešek proti gramatice programovacího jazyka. Počítač je matematický stroj a vyžaduje naprosto jednoznačný předpis své činnosti.
- Ve zdrojovém kódu nesmí být žádné syntaktické chyby, aby bylo jednoznačné, co má překladač dělat.
- Překladač neví, co chcete naprogramovat, proto se nemůže ani pokoušet o opravu vašich chyb, to musíte udělat sami.
- Mezi syntaktické chyby patří především překlepy, ale může to být také použití nekompatibilních datových typů nebo třeba chybné použití operátorů (např. nemůžete zkoušet násobit textové řetězce).

Sémantická chyba

- chyba v logice vašeho programu. Váš program může být syntakticky správný, překladač jej přeloží, linker slinkuje, ale program přesto může fungovat chybně.
- Za tyto chyby je vždy zodpovědný programátor a je na něm, aby se jich zbavil.
- Tyto chyby překladač ani linker neodhalí. Pokud máte štěstí, může na ně překladač nepřímo upozornit pomocí varovných hlášení při překladu.

Chybové hlášení - error

- zpráva, kterou vám překladač nebo linker sdělují, proč nelze z vašeho kódu vytvořit spustitelný program.
- Tato zpráva obvykle obsahuje jméno souboru a číslo řádku, kde se chyba nachází. Pozor! V jazyce C může někdy překladač odhalit chybu na jiném řádku, než je skutečná příčina.
- Pokud se vám označené číslo řádku nezdá, zkuste hledat chybu i několik řádků okolo.
- Abyste mohli co nejlépe lokalizovat své chyby, pište na každý řádek zdrojového textu maximálně jeden příkaz. Usnadníte si tím nejen lokalizaci chyb, ale i ladění.
- Chybové hlášení obsahuje také vysvětlení, co je podle překladače špatně.

Varovné hlášení - warning

- upozornění, kterým vám překladač nebo linker dávají najevo, že váš program sice možná jde přeložit, ale že obsahuje podezřelé konstrukce, které mohou naznačovat závažnější problém.
- Moudrý programátor věnuje varovným hlášením překladače náležitou pozornost.
- Vaší ambicí by měl být překlad bez varovných hlášení.
- Dobré porozumění chybovým hlášením vyžaduje znalost angličtiny a především dobrou znalost fungování počítače a logiky programovacího jazyka. Pokud zatím tyto znalosti nemáte, nezoufejte. Není tak těžké se to naučit. Čím více budete programovat, tím to pro vás bude snazší.

Make

- Program make je utilita pro automatizaci překladu zdrojových kódů do binárních souborů (spustitelné soubory ELF, EXE, knihovny a podobně).
- Soubor nazvaný **Makefile** určuje postup utility make při překladu a definuje závislosti mezi zdrojovými soubory.
- Při sestavování cíle sleduje make topologické seřazení v Makefile.
- Přestože jsou dnes v oblibě různá integrovaná vývojová prostředí (IDE) a jazykově specifické kompilátory, je make a tedy i Makefile stále široce vyžíván, zvláště pak na unixových platformách.

Ruční kompilace

- Napište si jednoduchý 'Hello word' program

.

```
#include <stdio.h>
int main(){
printf("Hello World!\n");
return 0; }
```

- A přeložte jej:

.

```
g++ main.c -o hello
```

- Spuštěním programu se na stdout vypíše 'Hello World!'

Základní Makefile

- Základní struktura je:

.

```
target: dependencies
[tab] system command
```

- Pro náš případ tedy:

.

```
all:
    g++ main.c -o hello
```

- Pro spuštění vytvořte soubor Makefile, vyplňte strukturu a spusťtě program 'make'

Použití závislostí

- Někdy je vhodné použít více cílů. Pokud změníte jen jeden soubor v projektu, nebudete muset rekompilovat vše, ale pouze ten upravený.

.

```
all: hello
hello: main.o factorial.o hello.o
	g++ main.o factorial.o hello.o -o hello
main.o: main.cpp
	g++ -c main.cpp
factorial.o: factorial.cpp
	g++ -c factorial.cpp
hello.o: hello.cpp
	g++ -c hello.cpp
clean:
	rm -rf *.o hello
```

- Všimněte si cíle 'clean', ten slouží k odstranění všech generovaných souborů.

Proměnné a komentáře

.

```
# Promenna CC bude jmeno kompileru, ktery se pouzije
CC=g++
# CFLAGS budou parametry, pouzite pri kompilaci
CFLAGS=-c -Wall
all: hello
hello: main.o factorial.o hello.o
	$(CC) main.o factorial.o hello.o -o hello
main.o: main.cpp
	$(CC) $(CFLAGS) main.cpp
factorial.o: factorial.cpp
	$(CC) $(CFLAGS) factorial.cpp
hello.o: hello.cpp
	$(CC) $(CFLAGS) hello.cpp
clean:
	rm -rf *.o hello
```

- Proměnné mohou celý kód zpřehlednit a zjednodušit.

Parametry kompilace

- man gcc
- -Wall = warnings on
- -pedantic = přísná kontrola syntaxe kódu
- -c = c file
- -o = object file

Kompilace dynamické knihovny

- Pro zkompilování dynamické knihovny použijte následující konstrukci:

.

```
libneco.so: neco1.o neco2.o ... necon.o
    gcc -shared -o libneco.so neco1.o ... necon.o -lm
```

- -shared = sdílená dynamická knihovna
- -lm = pokud knihovna volá funkce jiné knihovny, toto potlačí chybovou hlášku
- -fPIC = position-independent code - přidat k podmínkám kompilace *.o souborů
- -L. -lneco = při kompilaci celého programu přidejte mezi objekty
  - -L. = vloží knihovnu v adresáři '.'
  - -lname = jméno knuhovny (libneco.so)

Funkce main

- Main je první funkce, která je spuštěna.

.

```
int main(int argc, char **argv) {
TELO
}
```

- Proměnná argc obsahuje počet vstupních argumentů
- Pole argv obsahuje hodnoty argumentů

#include

Vložení knihovny do kódu:

.

```
#include "nd.h"
#include "nsd.h"
#include <stdio.h>
```

- “” - vyhledává soubor od aktuálního umístění souboru se zdrojovým kódem
- <> - vkládaný soubor vyhledává podle proměnné PATH anebo cest zadaných parametrem -I

For cyklus, if

- for cyklus

.

```
for (int i = 0; i < 20; i++){
TELO
}
```

- If
  - můžete použít: ==, !=, <= >=, <, >
  - shlukovat podmínky lze pomocí && a ||

i

```
f (i<0){ TELO }
```

fprintf

- Pro vypsání výstupu na stdout

.

```
fprintf("Hodnota promenne x je %d a y=%d",x,y);
```

- %i - integer
- %d - double
- %c - char

sscanf

- Slouží k převedení hodnot ze vstupu - parsing

.

```
int a;
sscanf(argv[1], "%i",&a);
```

- Z pole argv, polohy 1, načte integer do promenne a

Zadání

- Napište program 'main.c'
  - vstupem jsou dva argumenty (dvě celá čísla)
  - do programu vložte (include) poskytnuté dva dva zdrojové kódy (nd.c, nsd.c)
  - výstupem je buď podíl dvou čísel anebo nalezený společný dělitel
- Napište makefile
  - z poskytnutých souborů sestavte knihovnu (libnsd.so)
  - Přeložte Váš program, knihovnu a vytvořte finální program
  - přidání knihovny do cesty - export LD_LIBRARY_PATH=.

Reference


http://www.fit.vutbr.cz/~martinek/clang/gcc.html


http://mrbook.org/tutorials/make/