Programovací jazyk C(++)

5. Preprocesor, kompilace

Miroslav Spousta, 2005

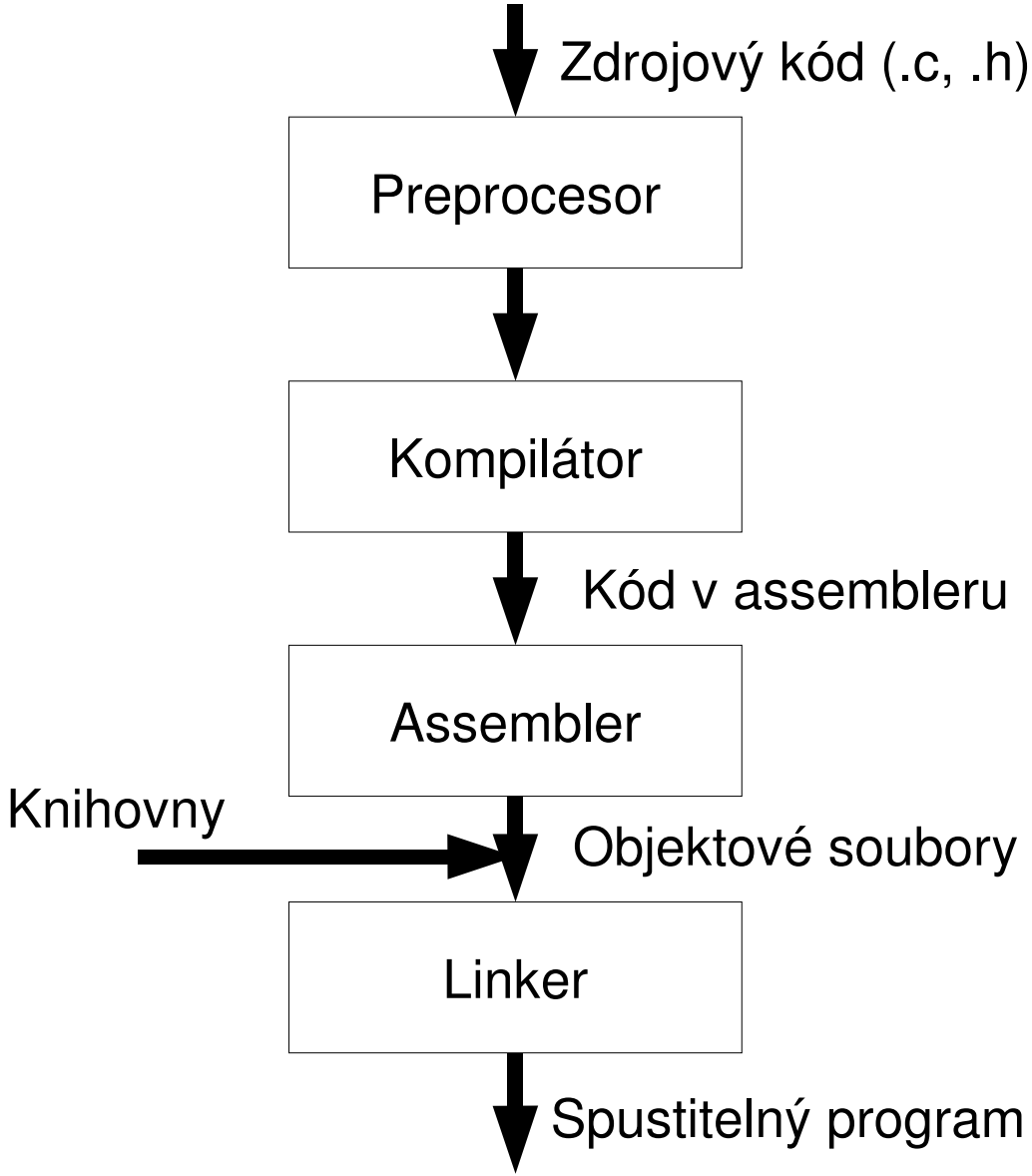
```
static struct vm_area_struct * unmap_fixup(struct mm_struct *mm,
        struct vm_area_struct *area, unsigned long addr, size_t len,
        struct vm_area_struct *extra)
{
        struct vm_area_struct *mpnt;
        unsigned long end = addr + len;

        area->vm_mm->total_vm -= len >> PAGE_SHIFT;
        if (area->vm_flags & VM_LOCKED)
                area->vm_mm->locked_vm -= len >> PAGE_SHIFT;

        /* Unmapping the whole area. */
        if (addr == area->vm_start && end == area->vm_end) {
                if (area->vm_ops && area->vm_ops->close)
                        area->vm_ops->close(area);
                if (area->vm_file)
                        fput(area->vm_file);
                kmem_cache_free(vm_area_cachep, area);
                return extra;
        }
```

C++

Kompilace programu



Preprocesor

- na vstupu dostane zdojový kód (.c nebo .h)
- vypustí komentáře
- interpretuje speciální direktivy preprocesoru

  příkazy, které jsou uvozeny **#**, např. **`#include <stdio.h>`**

  na výstupu preprocesoru už se tyto direktivy nevyskytují
- za direktivou preprocesoru se nepíše středník (!)
- preprocesor nekontroluje syntax, pouze vypouští nebo nahrazuje text
- **`#define PI 3.14159265`**

Direktiva `include`

- vkládá do aktuálního souboru jiný soubor

  obecně libovolný, ale používá se hlavně pro tzv. header soubory *.h

- **`#include <soubor>`** nebo **`#include "soubor"`**

  verze s **<>** znamená vlož systémový soubor (součást překladače/systému)

  verze s **""**znamená vlož soubor z aktuálního adresáře (součást překládaného programu)

- některé standardní include soubory (C):

  stdio.h – standardní vstup a výstup

  time.h – práce s časem, math.h – matematické operace

  wchar.h – práce s unicode znaky

  stdlib.h – různé funkce (převod řetězců na čísla, náhodná čísla, ...)

Makra bez argumentů

- neboli symbolické konstanty
- **`#define JMENO hodnota`**

  `#define PI 3.24159265`

  nadefinuje symbolickou knostantu jménem **`JMENO`**

  každý další výskyt dané konstanty (**`JMENO`**) v kódu nahradí řetězcem **`hodnota`**

  až na výskyty, které jsou v řetězcových konstantách
- hodnotu konstanty je možné testovat (porovnávat), případně oddefinovat
- často je vhodnější používat výčtový typ (enum)
- opět: za hodnotou není středník – (až na výjimky :-))
- hodnota může být cokoliv, třeba příkaz, nebo část příkazu

Makra

- podobně jako konstantu lze nadefinovat makro

  oproti konstantě má jeden nebo více argumentů

- **`#define sq(arg)(arg * arg)`**

  pokud je obsahem makra matematický výraz, uzavírá se do uvozovek

  správně by se měl uzavírat také každý výskyt argumentu:

  `sq(1+2) /* rozvine se na 1+2*1+2 */`

  správně má být: **`#define sq(arg)((arg) * (arg))`**

- některé standardní funkce mohou být také definovány jako makro

  např.: **`#define getchar() getc(stdin)`**

Podmíněný překlad

- **`#ifdef, #ifndef, #if, #else, #elsif, #endif`**
- direktivy umožňují existenci více variant kódu v jednom zdrojovém souboru

  neboli některé části kódu a jiné vynechat

  bloky jsou vyznačeny pomocí direktiv preprocesoru
- **`#ifdef, #ifndef`**: testují, zda je daná konstanta (ne)definována
- **`#if`**: testuje, zda je daná (konstantní) podmínka splněná

  musí být vyhodnotitelná před překladem :-)
- **`#else, #elsif`**: else, resp. else-if větve
- **`#endif`**: ukončení bloku

Podmíněný překlad

- často se používá, pokud je část kódu jen pro specifický případ

  např. jen pro některou platformu, OS, normu, …

  často může znepřehledňovat kód (nesmí se přehánět)

- nebo pro zamezení několikanásobnému vložení souboru (viz dále)

- případně pro vypuštění ladicích informací ve finální verzi programu

```
#ifdef LINUX
#define DEBUG(a) syslog(LOG_DEBUG, a)
#else
#define DEBUG(a) fprintf(stderr, a)
#endif

DEBUG("chyba, nemuzu otevrit soubor.txt\n");
```

Direktovy preprocesoru

- symbolické konstanty je možné měnit pouze tak, že je oddefinujeme

  `#undef JMENO`

  a znovu nadefinujeme: **`#define JMENO nova_hodnota`**

- **`#error message, #warn message`**

  během kompilace vytiskne hlášku a skončí (pouze **`#error`**)

  např. pokud pro danou situaci nelze pokračovat v překladu

Kompilátor

- na vstupu dostane kód bez komentářů a s rozvinutými direktivami preprocesoru (pokud chceme vidět, často si musíme explicitně říct)
- na výstupu bude kód v assembleru pro daný procesor
- kompilátor se (obecně) skládá z několika částí:

  lexikální analýza (rozdělení vstupu na tokeny)

  syntaktická analýza (vyhodnocení konstrukcí (např. if-then-else)

  sémantická analýza (kontrola typů)

  optimalizace (nezávislé či zavislé na cílovém assembleru)

  generování kódu

Assembler

- přeloží kód ze symbolické formy do řečí procesoru (CPU)
- vytváří tzv. objektový soubor (s příponou .o nebo .obj)

  téměř spustitelný soubor, ale s nevyřešenými odkazy na externí objekty

  např. knihovní funkce, funkce z ostatních object souborů, atd.

Assembler

```
.section        .rodata
.LC0:
        .string "Hello world!\n"
.text
        .align 4
.globl main
        .type    main,@function
main:
        pushl %ebp
        movl %esp,%ebp
        subl $8,%esp
        addl $-12,%esp
        pushl $.LC0
        call printf
        addl $16,%esp
        xorl %eax,%eax
        jmp .L2
        .p2align 4,,7
.L2:
        leave
        ret
.Lfe1:
        .size    main,.Lfe1-main
```

Linker

- spojí objektové soubory a knihovny, přibalí spouštěcí kód (který zavolá funkci main) a vytvoří spustitelný soubor
- knihovna je balík objektových souborů

  většinou k ní existuje tzv. hlavičkový soubor (header file, např stdio.h)

  knihovna může být statická (připojuje se při linkování spustitelného souboru)

  nebo dynamická (linkuje se při startu souboru)

Struktura zdrojového kódu

- tzv. oddělený překlad souborů
- původně hlavně kvůli omezeným prostředkům kompilátoru

  i dnes pro velké projekty, ale používá se hlavně kvůli přehlednosti
- program se skládá z více částí (zdrojových souborů .c)

  každý soubor obsahuje ucelenou část kódu

  soubory se překládají každý zvlášť do objektového souboru

  dohromady je sestavuje až linker (jinak by se např. mohly celé vkládat do jednoho C souboru)
- vkládají se pouze hlavičkové soubory (.h)

  které obsahují delkarace funkcí a externí deklarace proměnných

Paměťové třídy

- paměťová třída určuje vytvoření a zacházení s „objekty“ v C

  každý objekt má právě jednu paměťovou třídu

- třída auto

  tzv. automatické proměnné, používá se implicitně pro lokální proměnné

  jinde se použít nedá

  proměnné se vytvoří na zásobníku, po ukončení funkce se proměnná zruší

  pokud není inicializovaná, má pokaždé novou hodnotu

- třída register

  často používané automatické proměnné

  nápověda kompilátoru

Paměťová třída `extern`

- globální proměnné nebo funkce, mohou být použity kdekoliv v programu (i z jiných objektových souborů)
- implicitní třída pro globální proměnné
- pokud před globální proměnnou přidáme **`extern`**, dostaneme informativní deklaraci

  tedy nevyhrazuje se místo pro danou proměnnou
- informativních deklarací proměnné může být v jednom souboru několik, definice jen jedna

  pokud zapomeneme na definici, oznámí nám chybu až linker (nenajde proměnnou, jen odkazy na ni)

Paměťová třída `static`

- statické funkce a statické globální proměnné

  jsou definovány v daném souboru lokálně

  je možné je používat pouze v daném zdrojovém souboru (tzv. interní linkování)

  v jiném souboru se může vyskytovat jiná proměnná/funkce stejného jména

- lokální statické proměnné

  uvnitř funkcí, ale na rozdíl od automatických proměnných vznikají už při spuštění programu

  zachovávají si hodnotu při opětovném volání funkce (chovají se vlastně jako globální proměnné)

  ale oproti globálním proměnným jsou přístupné jen v dané funkci

Struktura zdrojového kódu

- zdrojové soubory (.c) obsahují:

  potřebné **`#include`** (vždy se vkládají pouze hlavičkové soubory!)

  definice lokálních typů

  deklarace statických globálních proměnných

  exportované funkce, lokální funkce (uvozené **`static`**)

- hlavičkový soubor (.h) obsahuje:

  ochránu před opětovným vložením do stejného zdrojového kódu

  definice globálních typů

  externí deklarace globálních proměnných

  hlavičky funkcí exportovaných vně souboru .c