KATEDRA
INFORMATIKY
UNIVERZITA PALACKÉHO V OLOMOUCI

Jazyk C: Proces překladu

KMI/OS1 Operační systémy I

Mgr. Markéta Trnečková, Ph.D.
www.marketa-trneckova.cz

Úlohy z předchozího cvičení

- `int2bits()`
- `bits2int()`
- `encode_date()`
- `decode_date()`

Příklad

V jakém pořadí jsou vyhodnocovány argumenty v jazyce C? Jak to můžeme například zjistit?

Jazyk C: Proces překladu

Učební text ke cvičení:
`https://phoenix.inf.upol.cz/~krajcap/courses/2025LS/OS1/tutorial02.pdf`

Organizační věci

- Proces překladu programu v jazyce C
- Princip znáte už z dřívějších kurzů
- Budeme využívat překladač gcc (GNU/Linux)
- Nainstalovat/zajistit si přístup k Linux

Základní příkazy v Linux

- `pwd` aktuální pracovní adresář
- `ls` výpis souborů a adresářů v aktuálním adresáři
- `cd` změna aktuálního adresáře
  - `cd ..` posun o adresář výše
  - `cd relativni_adresa`
- `mkdir nazev` vytvoření adresáře nazev
- `rm nazev` odstranění souboru nazev
- `cp odkud kam` kopie souboru (adresáře `-r`)
- `mv odkud kam` přesun/přejmenování souboru (adresáře)
- `cat nazev` zobrazit obsah souboru
- `less nazev` zobrazit obsah souboru (ukončení `q`)
- `nano nazev` upravit obsah souboru
- `vi nazev` upravit obsah souboru

Příklad `hello.c`

Příklad

```c
// hello.c
#include <stdio.h>

int main(int argc, char **argv)
{
    /* program vypise Hello World!*/
    printf("Hello World!\n");
    return 0;
}
```

Překlad pomocí `gcc`

```
gcc hello.c-o hello
```

Spuštění

```
./hello
```

Fáze překladu

1. **Předzpracování preprocesorem** – odstranění komentářů, expanze maker, vkládání souborů, ...
   příkaz `cpp`
2. **Překlad do jazyka symbolických adres** – zápis v podobě jednotlivých instrukcí procesoru
3. **Překlad do objektového souboru** – pomocí *assembleru*, obsahuje program přeložený do strojového kódu + další informace (konstanty, názvy funkcí a proměnných, ladící informace, ...)
   příkaz `as`
4. **Sloučení objektových souborů** a spojení s knihovnami, výsledkem je binární soubor
   příkaz `ld`

Využijeme přepínače `gcc`, abychom se podívali na jednotlivé výstupy.

Fáze překladu

Preprocesor

```
gcc -E hello.c
```

Příklad

Podívejte se na výstup preprocesoru. Všimněte si, že byly odstraněny komentáře. Zkuste najít definici funkce `printf()`.

Fáze překladu

Překlad do jazyka symbolických adres

```
gcc -S hello.c
```

- Vznikne textový soubor `hello.s`

Příklad

Pomocí editoru, nebo příkazu `cat` se podívejte na jeho obsah.

Fáze překladu

Překlad do objektového souboru

```
gcc -c hello.c
```

- Vznikne soubor `hello.o`
- soubor je binární – `hexdump -C hello.o`

Příklad

Prohlédněte si soubor `hello.o` pomocí nástroje `objdump`, který zobrazí logické části souboru (sekce). Zajímavé sekce jsou `.text` nebo `.rodata`.

```
objdump -s hello.o
```

Fáze překladu

Překlad do objektového souboru

- `objdump` nabízí také dissasembling (přepínač `-d`)
- dissasembling – překlad strojového kódu do jazyka symbolických adres
- `objdump -d -M intel hello.o`
- přepínač `-M intel` zajistí použití syntaxe z přednášek

Příklad

Vyzkoušejte `objdump -d -M intel hello.o`.
Objektový soubor neobsahuje kód fce `printf()`.

Fáze překladu

Vytvoření spustitelného souboru

Překlad pomocí `gcc`

```
gcc -o hello hello.o
```

Překlad pomocí linkeru

```
ld -o hello hello.o /lib64/crt1.o -dynamic-linker
/lib64/ld-linux-x86-64.so.2 -lc
```

Linker spojí vstupní objektový soubor se souborem crt1.o (C runtime) a standardní knihovnou jazyka C (přepínač `-lc`). Přepínač `-dynamic-linker` zajišťuje, že knihovna bude připojena dynamicky.

Rozdělení kódu do více souborů

- lepší přehlednost kódu
- vkládání souborů do jednoho `#include`
- oddělený překlad
- **Výhody odděleného překladu**:
  - není nutné vždy překládat celý zdrojový kód
  - jednotlivé části mohou být psány v jiném jazyce
- pro pohodlný překlad – nástroj `make`
- `make` sestaví program podle pravidel, která zapisujeme do `Makefile`

Makefile

- Pravidla

```
cil: zavislosti
<TAB!>prikaz pro sestaveni cile
<TAB!>prikaz pro sestaveni cile
```

Příklad

```
hello: hello.o
	gcc -o hello hello.o
hello.o: hello.c
	gcc -c hello.c
```

2 pravidla.
1. pravidlo – pro sestavení spustitelného souboru `hello` potřebujeme `hello.o`. Pokud ho máme, zavoláme příkaz `gcc -o hello hello.o`
2. pravidlo **Vysvětlete**.

Makefile

Příklad

Vytvořte Makefile obsahující tato pravidla.

```
hello: hello.o
	gcc -o hello hello.o
hello.o: hello.c
	gcc -c hello.c
```

Pomocí `make` přeložte.

Makefile

`make` zjistí závislosti a spustí příkazy ve správném pořadí.

Příklad

Odstraňte `hello.o` (`rm hello.o`) a spusťte `make` znovu.

`make` zajistí, aby se překládaly jen změněné části.

Příklad

Spusťte znovu `make`.

Sestavení většího programu

- některé části kódu (funkce) chceme mít oddělené od hlavního programu
- vytvoříme hlavičkový soubor a soubor, kde tyto funkce budeme definovat

Sestavení většího programu

Hlavičkový soubor

- `soubor.h`
- obsahuje pouze deklarace funkcí
- je vhodné používat direktivy preprocesoru, které zajistí, že se deklarace funkcí vloží pouze jednou

Příklad

```
// hello.h
#ifndef MYFUNCS_H
#define MYFUNCS_H

/* funkce pro vypocet faktorialu */
unsigned int fact(unsigned int n);

#endif // MYFUNCS_H
```

Sestavení většího programu

Zdrojový soubor

- `soubor.c`
- obsahuje pouze definice funkcí

Příklad

```c
// myfuncs.c
#include "myfuncs.h"

unsigned int fact(unsigned int n){
    // funkce pro vypocet faktorialu
    if (n == 0) return 1;
    return n * fact(n - 1);
}
```

Sestavení většího programu

Hlavní zdrojový soubor

- `hlavni.c`
- obsahuje funkci `main()`
- vkládáme do něj hlavičkový soubor
- **Proč nevkládáme zdrojový?**

Příklad

```c
// hello.c
#include <stdio.h>
#include "myfuncs.h"

int main(int argc, char* argv[]){
    printf("5! = %i\n", fact(5));
    return 0;
}
```

Sestavení většího programu

```
gcc -o hello hello.o myfuncs.o
```

Příklad

Jak bude vypadat Makefile? Kolik bude mít pravidel?

Sestavení většího programu

Příklad

Jak bude vypadat Makefile? Kolik bude mít pravidel?

Příklad (Řešení)

```
hello: hello.o myfuncs.o
	gcc -o hello hello.o myfuncs.o
hello.o: hello.c
	gcc -c hello.c
myfuncs.o: myfuncs.c
	gcc -c myfuncs.c
```

Vyzkoušejte!

Vyzkoušejte také provést změny pouze v souboru `hello.c`. Při opětovném volání `make` uvidíte, že se znovu nebude vytvářet myfuncs.o.

Sestavení většího programu

vylepšení

- bývá zvykem nastavit přepínače, pomocí kterých se udává chování překladače
- `gcc PREPINACE -o hello hello.o myfuncs.o`
- **Přepínače**:
  - míra optimalizace: `-O1`, `-O2`, `-O3`
  - standard jazyka: `-std`
  - zobrazení upozornění: `-W`
  - generování ladících informací: `-g`
- `make` podporuje proměnné (podobně jako unixový shell)

Sestavení většího programu

Proměnné v Makefile

Příklad

```
CFLAGS=-O1 -Wall -std=c99 -g
hello: hello.o myfuncs.o
	gcc $(CFLAGS) -o hello hello.o myfuncs.o
hello.o: hello.c
	gcc $(CFLAGS) -c hello.c
myfuncs.o: myfuncs.c
	gcc $(CFLAGS) -c myfuncs.c
```

Programování v assembleru

Překlad programu

- zdrojový kód `demo.asm`
- překlad pomocí `nasm` (přívětivější než `as`)
  `nasm -f elf64 demo.asm`
- vznikne soubor `demo.o`, ve formátu `elf64` (implicitně využívaný v `gcc`)

Programování v assembleru
Příklad

Příklad

```
; soubor demo.asm
global foo
section .text
foo:
    mov eax, 42
    ret
```

- `;` komentář
- `global` jaké proměnné a funkce daný zdrojový kód poskytuje
- `section .text` – vyznačení sekce `.text` (obsahuje kód programu v jazyce symbolických adres)
- `foo:` návěští uvozující kód funkce (`foo` je identifikátor funkce)

Programování v assembleru
Příklad

Příklad

```
; soubor demo.asm
global foo
section .text
foo:
    mov eax, 42
    ret
```

- `mov eax 42`
  uloží do registru `eax` hodnotu 42
  `eax` (resp. `rax`) slouží k předání celočíselné návratové hodnoty
- `ret`
  instrukce návratu z funkce

Programování v assembleru
Příklad

Příklad

Přeložte

```
; soubor demo.asm
global foo
section .text
foo:
    mov eax, 42
    ret
```

pomocí příkazu

```
nasm -f elf64 demo.asm
```

Programování v assembleru

Spojení s jazykem C

Příklad

```c
// hello.c
#include <stdio.h>

/* Prototyp funkce napsane v asembleru */
int foo();

int main(int argc, char* argv[]){
    printf("Answer to life, etc.: %i\n",foo());
    return 0;
}
```

Samotné provázání pomocí linkeru.

Příklad

Jak by vypadal Makefile?

Programování v assembleru

Spojení s jazykem C

Příklad (Řešení)

```
hello: hello.o demo.o
	gcc -o hello hello.o demo.o
hello.o: hello.c
	gcc -c hello.c
demo.o: demo.asm
	nasm -f elf64 demo.asm
```

Jazyk C
Úkoly k procvičení

1. Vezměte funkce `int2bits()` a `bits2int()` a umístěte je do samostatného souboru `bits.c` a vytvořte odpovídající hlavičkový soubor `bits.h`. Vytvořte program, který popsané funkce bude používat. Pro překlad programu vytvořte vhodný makefile a program přeložte.
2. Vezměte funkce `encode_date()` a `decode_date()` a umístěte je do samostatného souboru `dates.c` a vytvořte odpovídající hlavičkový soubor `dates.h`. Vytvořte program, který popsané funkce bude používat. Pro překlad programu vytvořte vhodný makefile a program přeložte.