

Assembly tutorial

Fordítás, linkelés

Smidla József

2016. március 2.

Hogyan készül a futtatható program?

Hogyan készül a futtatható program?

- 1 Megírjuk a forráskódot, például a main.c-t

Hogyan készül a futtatható program?

- 1 Megírjuk a forráskódot, például a `main.c`-t
- 2 Ezt lefordítjuk tárgykóddá, ebből lesz a `main.o`

Hogyan készül a futtatható program?

- ❶ Megírjuk a forráskódot, például a `main.c`-t
- ❷ Ezt lefordítjuk tárgykóddá, ebből lesz a `main.o`
- ❸ A `main.o`-hoz hozzácsatolunk (linkelünk) más tárgykódokat → futtatható program

Mi van a tárgykódban?

Hogyan készül a futtatható program?

- ❶ Megírjuk a forráskódot, például a `main.c`-t
- ❷ Ezt lefordítjuk tárgykóddá, ebből lesz a `main.o`
- ❸ A `main.o`-hoz hozzácsatolunk (linkelünk) más tárgykódokat → futtatható program

Mi van a tárgykódban?

- Futtatható gépkód

Hogyan készül a futtatható program?

- ❶ Megírjuk a forráskódot, például a `main.c`-t
- ❷ Ezt lefordítjuk tárgykóddá, ebből lesz a `main.o`
- ❸ A `main.o`-hoz hozzácsatolunk (linkelünk) más tárgykódokat → futtatható program

Mi van a tárgykódban?

- Futtatható gépi kód
- Hivatkozás olyan függvényekre, változókra, amik ebben a tárgykódban nincsenek meg, de használjuk őket

Hogyan készül a futtatható program?

- ➊ Megírjuk a forráskódot, például a `main.c`-t
- ➋ Ezt lefordítjuk tárgykóddá, ebből lesz a `main.o`
- ➌ A `main.o`-hoz hozzácsatolunk (linkelünk) más tárgykódokat → futtatható program

Mi van a tárgykódban?

- Futtatható gépkód
- Hivatkozás olyan függvényekre, változókra, amik ebben a tárgykódban nincsenek meg, de használjuk őket

Példa: A `printf` függvény egy speciális tárgykód gyűjteményben található (library)

main.c:

main.c:

```
int fooFuggveny();
```

```
int main() {  
    return fooFuggveny();  
}
```

main.c:

```
int fooFuggveny();
```

```
int main() {  
    return fooFuggveny();  
}
```

foo.c:

main.c:

```
int fooFuggveny();
```

```
int main() {  
    return fooFuggveny();  
}
```

foo.c:

```
int fooFuggveny() {  
    return 5;  
}
```

A buildelés lépései:

A buildelés lépései:

❶ `gcc -Wall -c main.c`
`main.c → main.o`

A buildelés lépései:

- 1 `gcc -Wall -c main.c`
`main.c → main.o`
- 2 `gcc -Wall -c foo.c`
`foo.c → foo.o`

A buildelés lépései:

- 1 `gcc -Wall -c main.c`
`main.c → main.o`
- 2 `gcc -Wall -c foo.c`
`foo.c → foo.o`
- 3 `gcc -Wall -o main main.o foo.o`
`main.o + foo.o → futtatható program`

Linuxos nm parancs: Kilistázza a tárgykódban / binárisban lévő szimbólumokat

```
> nm main.o
```

```
                U fooFuggveny
```

```
000000000000000000 T main
```

Linuxos nm parancs: Kilistázza a tárgykódban / binárisban lévő szimbólumokat

```
> nm main.o
```

```
                U fooFuggveny
```

```
0000000000000000 T main
```

Kiolvashatóak a külső hivatkozások is ...

```
mov.asm:
```

```
section .text
```

```
global az_elet_ertelme
```

```
az_elet_ertelme:
```

```
push    ebp
```

```
mov     ebp, esp
```

```
mov     eax, 42
```

```
mov     esp, ebp
```

```
pop     ebp
```

```
ret
```

A mov.asm-hez írjuk meg a következő C kódot
(mainC.c):

```
int az_elet_ertelme();
```

```
int foo() {  
    return 0;  
}
```

```
int main() {  
    return az_elet_ertelme();  
}
```

A mov.asm-hez írjuk meg a következő C kódot
(mainC.c):

```
int az_elet_ertelme();

int foo() {
    return 0;
}

int main() {
    return az_elet_ertelme();
}
```

Fordítás gcc-vel, majd az nm kimenete:

```
                U az_elet_ertelme
0000000000000000 T foo
000000000000000b T main
```

Most ugyanez a mainCPP.cpp-ben:

```
int az_elet_ertelme();
```

```
int foo() {  
    return 0;  
}
```

```
int main() {  
    return az_elet_ertelme();  
}
```

Most ugyanez a mainCPP.cpp-ben:

```
int az_elet_ertelme();
```

```
int foo() {  
    return 0;  
}
```

```
int main() {  
    return az_elet_ertelme();  
}
```

Fordítás g++-al, majd az nm kimenete:

```
0000000000000000b T main  
                U _Z15az_elet_ertelmev  
0000000000000000 T _Z3foov
```

Probléma:

Probléma:

- A mainCPP.o-ban a `_Z15az_elet_ertelmev` hivatkozás szerepel

Probléma:

- A mainCPP.o-ban a `_Z15az__elet__ertelmev` hivatkozás szerepel
- A mov.o-ban viszont az `az__elet__ertelme` szimbólum van

Probléma:

- A mainCPP.o-ban a `_Z15az_elet_ertelmev` hivatkozás szerepel
- A mov.o-ban viszont az `az_elet_ertelme` szimbólum van
- Linkelési hiba, mert a kettőt nem tudjuk összelinkelni

```
mainCPP.o: In function 'main':
```

```
mainCPP.cpp:(.text+0x1c): undefined reference to  
'az_elet_ertelme()'
```

```
collect2: error: ld returned 1 exit status
```

C nyelv: minden függvényt a neve azonosít

C nyelv: minden függvényt a neve azonosít

C++: egy függvényt nem csak a neve azonosít, hanem:

C nyelv: minden függvényt a neve azonosít

C++: egy függvényt nem csak a neve azonosít, hanem:

- paraméterek típusa

C nyelv: minden függvényt a neve azonosít

C++: egy függvényt nem csak a neve azonosít, hanem:

- paraméterek típusa
- paraméterek száma

C nyelv: minden függvényt a neve azonosít

C++: egy függvényt nem csak a neve azonosít, hanem:

- paraméterek típusa
- paraméterek száma
- const / nem const

C nyelv: minden függvényt a neve azonosít

C++: egy függvényt nem csak a neve azonosít, hanem:

- paraméterek típusa
- paraméterek száma
- const / nem const
- melyik osztályhoz tartozik

C nyelv: minden függvényt a neve azonosít

C++: egy függvényt nem csak a neve azonosít, hanem:

- paraméterek típusa
- paraméterek száma
- const / nem const
- melyik osztályhoz tartozik
- templatezés

C nyelv: minden függvényt a neve azonosít

C++: egy függvényt nem csak a neve azonosít, hanem:

- paraméterek típusa
- paraméterek száma
- const / nem const
- melyik osztályhoz tartozik
- templatezés
- ...

C nyelv: minden függvényt a neve azonosít

C++: egy függvényt nem csak a neve azonosít, hanem:

- paraméterek típusa
- paraméterek száma
- const / nem const
- melyik osztályhoz tartozik
- templatezés
- ...

A tárgykódban meg kell különböztetni az azonos nevű, de különböző függvényeket → mangling: kidekoráljuk a függvények nevét

Így lesz az az_elet_ertelme függvényből _Z15az_elet_ertelmev

Mondjuk meg a fordítónak, hogy ez a függvény olyan, mintha C-ben írtuk volna:

```
extern "C" int az_elet_ertelme();
```

```
int foo() {  
    return 0;  
}
```

```
int main() {  
    return az_elet_ertelme();  
}
```

Mondjuk meg a fordítónak, hogy ez a függvény olyan, mintha C-ben írtuk volna:

```
extern "C" int az_elet_ertelme();
```

```
int foo() {  
    return 0;  
}
```

```
int main() {  
    return az_elet_ertelme();  
}
```

Az nm kimenete:

```
          U az_elet_ertelme  
0000000a T main  
00000000 T _Z3foov
```

- C kódnál nincs gond
- C++-ban ne felejtjük el az extern "C"-t