

Jakso 10

Ohjelman suoritus järjestelmässä

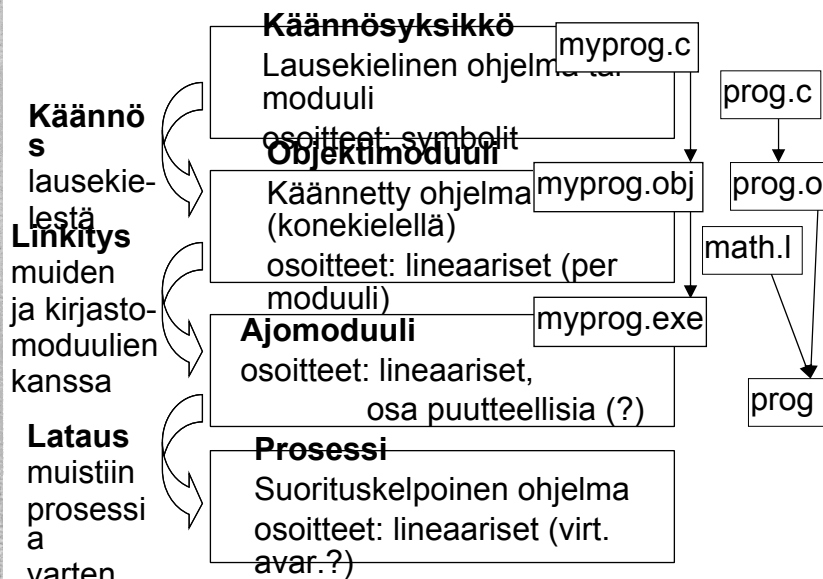
Käännös
Linkitys
Dynaaminen linkitys
Lataus

31/05/2004

Copyright Teemu Kerola, K2003

1

Lausekielestä suoritukseen



31/05/2004

Copyright Teemu Kerola, K2003

2

Käännösyksikkö ⁽⁴⁾

- Jollain ohjelmointikielellä kuvattu eheä kokonaisuus, joka halutaan aina kääntää yhdessä
 - kaikki yhteen liittyvät aliohjelmat
 - olioperustainen luokka
- Liian suuri kokonaisuus?
 - turhaa aikaa kääntämiseen joka muutoksen jälkeen
- Liian pieni kokonaisuus?
 - turhaa aikaa murehtia ja toteuttaa liitoksia muiden moduulien kanssa
- Käännösyksikön ohjelmointikieli ei ole tärkeä
 - niiden sitominen yhteen tapahtuu objektimoduulien tasolla

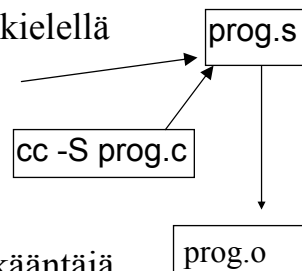
31/05/2004

Copyright Teemu Kerola, K2003

3

Assembler-kielinen käännösyksikkö

- Käännösyksikkö voi olla myös suoraan k.o. koneen symbolisella konekielellä kirjoitettu
 - suoraan käsin
 - kääntäjän generoimana korkean tason kielestä
- Käännöksen tekee assembler-kääntäjä tavallisen kääntäjän asemesta
 - yleensä osa tavallista kääntäjää



31/05/2004

Copyright Teemu Kerola, K2003

4

Objektimoduuli (8)

- Konekielinen koodi
 - moduulin sisäiset viitteet paikallaan (linearisessa muistiavaruudessa)
 - moduulin ulkopuoliset viitteet merkitty
- Linkitystä varten tiedot
 - tiedot niiden osoitteiden sijainnista, jotka täytyy päivittää, kun moduulin osoiteavaruus yhdistetään jonkin toisen moduulin osoiteavaruuden kanssa linkityksessä
 - tiedot viittauksista moduulin ulkopuolelle **IMPORT**
 - tiedot kohdista joista tähän moduuliin saa viitata ulkopuolelta **EXPORT**
 - symbolitaulu

RELOCATION TABLE

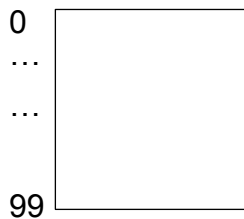
SYMBOL TABLE

31/05/2004

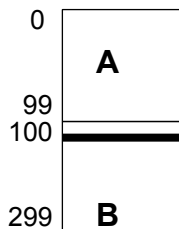
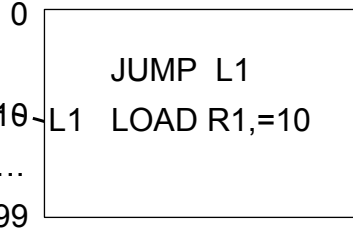
Copyright Teemu Kerola, K2003

5

Objektimoduuli A



Objektimoduuli B



Muisti

~~'jump' 0 0 11010~~

'load' 1 0
10

31/05/2004

Copyright Teemu Kerola, K2003

6

Symbolitaulu

- Kääntäjä generoi
- Ylläpidetään linkityksen aikana
- Joskus ylläpidetään myös latauksen jälkeen virheilmoitusten tekemistä varten
 - ohjelmien kehitysympäristöt ylläpitävät symbolitaulua koko ajan
- Jätetään pois valmiista ohjelmasta
 - vie turhaa tilaa

31/05/2004

Copyright Teemu Kerola, K2003

7

Lähdekieli vs. konekieli ⁽⁵⁾

- Pascal-lauseke: $N := I+J;$
- C-lauseke: $N = I+J$
- Java-lauseke: $N = I+J;$

TTK-91
symbolinen
konekieli:

I	DC	3
J	DC	4
N	DC	0
FORMULA: LOAD R1, I		
ADD R1, J		
STORE R1, N		

Pentium II, Motorola 680x0
ja

SPARC symbolinen
konekieli:

ks. Fig. 7-2
[Tane99]

31/05/2004

Copyright Teemu Kerola, K2003

8

(Assembler) kääntäjän ohjauskäskyt ⁽⁴⁾

- Eivät varsinaista koodia
 - Niistä ei tule konekäskyjä
- Ohjaavat käännoästä
 - TTK-91: DC
 - DS
 - EQU

Pentium II:

ks. Fig. 7-3
[Tane99]

Makrot ⁽⁶⁾

- Helpottavat ohjelmointia
- Usein toistuville koodisarjoille annetaan nimi => **makro**
- Makroilla voi olla parametreja
 - useimmiten nimiparametreja (call-by-name)
- Makrot käsitellään ennen kääntämistä
 - Eivät kuulu konekieleen
 - Makron 'kutsu' (käyttö) korvataan makron rungolla
- Esimerkkejä
 - swap
 - aliohjelmien prologi ja epilogi
 - itse tehdyt, kääntäjän käyttämät
- Erot aliohjelmiin

ks. Fig. 7-4 ja 7-6
[Tane99]

ks. Fig. 7-5
[Tane99]

Literaalit ⁽⁵⁾

- Vakioita

- Joko niin suuria, että eivät mahdu konekäskyn vakio-osaan

```
ttk-91: käskyn vakiot 2-  
tavuisia,
```

```
arvoalue: -32767...32767
```

- tai muuten vain halutaan pitää datan joukossa eikä käskyjen

```
Pi DC 3.14159265 ; (!!??)
```

yhteyteen

```
One DC 1 vrt. One EQU 1
```

talletettuna

```
OneMeg DC 1024576
```

- Niitä ei saisi muuttaa

```
LOAD R1, One
```

```
ADD R1,=1
```

```
STORE R1, One ; ask for
```

```
trouble
```

31/05/2004

Copyright Teemu Kerola, K2003

11

Literaalit

- Korkean tason kielissä kaikki isot vakiot ovat literaaleja

```
N:=35000;
```

```
Var myStr ="literal"
```

- kääntäjän pitäisi estää literaalien muuttamisen

```
FortranX: 5 = 6;
```

```
LOAD R1,six  
STORE R1, five
```

```
???????
```

```
?
```

- literaalia ei saisi välittää viiteparametrina

- aliohjelma voisi muuttaa sen arvoa?

- Myös joissakin assemblerkielissä on literaalinen implisiittinen (automaattinen) määrittely

- helpommin luettavaa koodia

```
Load R14,  
=F'234567'
```

- literaalin 234567 tilanvaraus automaattisesti

31/05/2004

Copyright Teemu Kerola, K2003

12

Assembler-käännös (10)

- 1. vaihe:
 - laske käskyjen tilanvaraukset
 - ttk-91 helppoa, koska kaikki käskyt 4 tavua!
 - generoi symbolitaulu ks. Kuva 6.2
[Häk98]
 - arvot, arvon vaatima tavumäärä
 - uudelleensijoitustiedot (omana tauluna?)
 - generoi tai käytä muita tauluja
 - literaalitaulu (tilanvaraus lopuksi)
 - kääntäjän ohjauskäskytaulu
 - operaatiokooditaulu

31/05/2004

Copyright Teemu Kerola, K2003

13

Assembler-käännös (2)

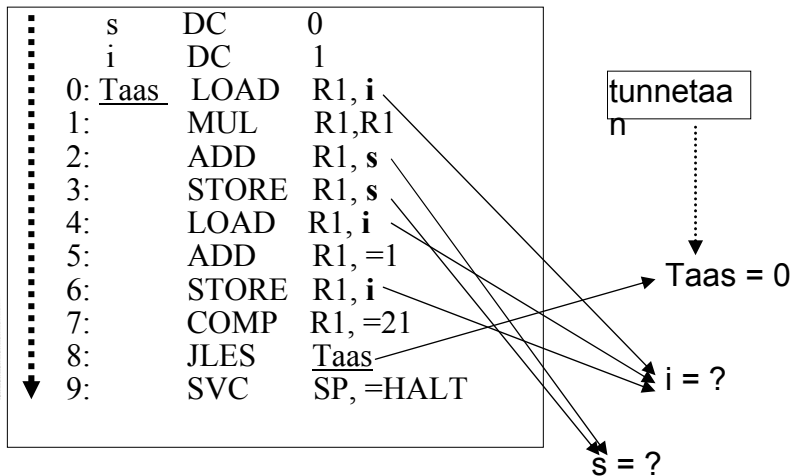
- 2. Vaihe ks. Kuva 6.3
[Häk98]
 - generoi lopullinen objektimoduuli ks. Fig. 7-16
[Tane99]
 - tulosta symbolinen assembler -listaus
 - generoi taulut linkitystä varten
 - osana objektimoduulia
 - anna virheilmoitukset
- 3. Vaihe
 - koodin optimointi
 - voi olla jo ennen 2. vaihetta tai sen yhteydessä

31/05/2004

Copyright Teemu Kerola, K2003

14

TTK-91: Assembler-käännös



31/05/2004

Copyright Teemu Kerola, K2003

15

Symbolitaulu 1. vaiheen aikana

ks. kalvo 15

Symboli	tyyppi	arvo	uud. sij.tietoa
s	data	?	2, 3
i	data	?	0, 4, 6
Taas	viite	0	8
HALT	vakio	11	

Konekäsyt 2 ja 8

	OPER	Rj	M	Ri	ADDR
2:	ADD	1	1	0	?
....					
8:	JLES	0	0	0	0

31/05/2004

Copyright Teemu Kerola, K2003

16

Koodi & data 1. vaiheen jälkeen

```

s   DC   0
i   DC   1

0: Taas  LOAD  R1, i
1:      MUL   R1, R1
2:      ADD   R1, s
3:      STORE R1, s
4:      LOAD  R1, i
5:      ADD   R1, =1
6:      STORE R1, i
7:      COMP  R1, =21
8:      JLES  Taas
9:      SVC   SP, =HALT
10:     0      ; siis s = 10
11:     1      ; siis i = 11
    
```

Kaikilla
symboleilla
tunnettu
arvo

Symbolitaulu 1. vaiheen jälkeen

ks. kalvo 17

Symboli	tyyppi	arvo	uud. sij.tietoa
s	data	10	2, 3
i	data	11	0, 4, 6
Taas	viite	0	8
HALT	vakio	11	

	OPER	Rj	M	Ri	ADDR
2:	ADD	1	1	0	10
....					
8:	JLES	0	0	0	0

TTK-91-objektimoduuli

Moduulin otsake
EXPORT-hakemisto
IMPORT-hakemisto
Uudelleensijoitushakemisto
Koodi ja alustettu data
Moduulin lopuke

(Kuva 6.3
[Häkk98])

TTK-91-objektimoduuli

- Moduulin otsakeosa
 - moduulin nimi
 - linkittäjän tarvitsemia tietoja
 - objektimoduulin osien pituudet
 - käänköspäivämäärä
 - kääntäjän nimi ja versio
 - ensimmäisen suoritettavan käskyn osoite
 - ellei aina 0

TTK-91-objektimoduuli

- **EXPORT-hakemisto**
 - tunnukset, joihin voidaan viitata muista moduuleista
 - rutiinit, aliohjelmat, (oliot, metodit)
 - yhteiskäyttöinen data
 - tunnuksen osoite (= symbolin arvo)
 - mahdollinen käyttöoikeus
 - R/W/E/RW

31/05/2004

Copyright Teemu Kerola, K2003

21

TTK-91-objektimoduuli

- **IMPORT-hakemisto**
 - muissa moduuleissa määritellyt tunnukset
 - tunnus
 - niiden käskyjen osoitteet, jossa tunnus esiintyy
- **Koodi ja alustettu data**
 - alustamattomille muuttujille ei tarvitse varata (vielä) tilaa, mutta ne on otettava huomioon data-alueen koossa

31/05/2004

Copyright Teemu Kerola, K2003

22

TTK-91-objektimoduuli

• Uudelleensijoitushakemisto

- niiden käskyjen osoitteet, joiden osoiteosaa on korjattava, kun siirrytään moduulien yhteiseen osoiteavaruuteen
 - suoraviivainen lisäys (joka käskyyn) ei toimi, sillä käskyn osoiteosa voi olla vakio, jota ei saa muuttaa
 - erikseen
 - (a) **paikalliseen dataan viittaavat** ja
 - (b) **hyppykäskyt**,
- sillä linkittäessä yhdistetään erikseen data- ja koodialueet

31/05/2004

Copyright Teemu Kerola, K2003

23

Korkean tason kielen käänös (7)

• Enemmän vaiheita

- Syntaktisten alkioiden etsintä (front end)
 - Syntaksipuun generointi ja jäsenitys
- Lauseiden tunnistaminen syntaksipuun avulla
- Välikielen (välikoodin) generointi (ei aina)
- Koodin generointi (back end)
 - ei (yleensä) Java-ohjelmille

BEGIN 123.45 IF (

Välikieliesitys ja symbolitaulut

Lisää tietoa?  Kääntäjien ja ohj. Kielten kurssit

31/05/2004

Copyright Teemu Kerola, K2003

24

Linkitys

- Uudelleensijoitusongelma (relocation problem)
 - jokaisen objektimoduulin osoitteet alkavat 0:sta
 - tulosmoduulissa kaikki yhdessä lineaarisessa osoiteavaruudessa
 - useimpien moduulien kaikkia osoitteita täytyy muuttaa
 - käskyjen osoitteet
 - datan osoitteet

31/05/2004

Copyright Teemu Kerola, K2003

25

Linkitysesimerkki

- Neljä moduulia: A, B, C ja D
- Laske joka moduulille *uudelleensijoitusvakio* (moduulin alkuosoite)
- Lisää k.o. vakio kunkin moduulin sisäisiin viitteisiin
- Etsi kaikki moduulien väliset viitteet, ja aseta kyseisten viitteiden osoitteet oikein

ks. Fig. 7-14
[Tane99]

(relocation
constant)

ks. Fig. 7-15 (a)
[Tane99]

ks. Fig. 7-15 (b)
[Tane99]

31/05/2004

Copyright Teemu Kerola, K2003

26

Muuttujan X viittausten päivitys ⁽³⁾

- Miten löytää linkityksen aikana kaikki kohdat, joissa muuttujaan X viitataan?
 - Vastaus 1: taulukko, jossa kaikki kohdat listattu
 - taulukko voi olla hyvin iso
 - kaikille muuttujille tilaa maksimi tarpeen verran?
 - Vastaus 2: Muuttujan X viittaukset on linkitetty keskenään linkitetyksi listaksi objektimoduulissa
 - vain linkitetyn listan alkuosoite taulukossa (tarvitaan vain yksi osoite per muuttuja)
 - X:n osoitteen paikalla aluksi linkki seuraavaan käskyyn, missä X:ään viitataan
 - listan voi käyttää vain yhden kerran

31/05/2004

Copyright Teemu Kerola, K2003

27

Muuttujan X viittaukset linkitettynä listana

symbolitaulu, moduuli

ABC	Symb	sij	viittaus
X	700	23	

lähdekoodi

```
23:  Load R1, X
...
34  Store R3, X(R1)
...
555 Add  R4, X

700 DC  0 ; X
```

objektimoduuli

```
23:  Load 1 0 34
...
34:  Store 3 1
555
...
555: Add 4 0 -1
```

31/05/2004

Copyright Teemu Kerola, K2003

28

Staattinen linkitys ⁽⁵⁾

- Tavallinen (staattinen) linkitys vaatii, että kaikki ohjelmakoodissa viitatus moduulit ja kirjastorutiinit on linkitetty ennen suoritusta
- Ajomoduulista tulee hyvin iso
 - mukana myös paljon moduuleja, joihin ei yhdellä suorituskerralla tule lainkaan viittauksia
 - Esim: kääntäjässä koodin optimointikoodi, vaikka optimointia ei suoriteta
 - Esim: pelissä tasojen 8-22 moduulit, vaikka aloittelija ei pääse tasoa 3 ylemmäksi vielä kuukausiin

Dynaaminen linkitys ⁽⁴⁾

- Jätetään linkityksessä kutsukohdat muihin moduuleihin auki
- Pienempi ajomoduuli, mutta hitaampi suorittaa
- Viittaus ”ratkaisemattomaan” (eli ei-linkitettyyn) moduuliin ratkotaan suoritusaikana
 - Suoritus keskeytyy ja puuttuva moduuli linkitetään paikalleen (kaikki viittaukset siihen korjataan kuntoon)

Windows DLL

- DLL - Dynamic Link Library
 - koodia, dataa tai molempia
- Säästää tilaa myös yhteiskäytön vuoksi
- Helpompi korjata virheitä
 - ei tarvita uutta käännöstä eikä lähdekielistä koodia!
 - riittää kun DLL vaihdetaan uuteen
 - seuraavassa suorituksessa uusi versio käyttöön
- Ajomoduli kootaan kuten tavallinen objektimoduuli
 - DLL-moduulit ja DLL-viittaukset merkitty erikoislipukkein (huomioidaan linkityksen yhteydessä)

.dll	yleinen tapaus
.drv	driver
.fon	font

ks. Fig. 7.19 [Tane99]

Windows DLL:n linkityksen kaksi tapaa ⁽³⁾

- Epäsuora dynaaminen linkitys
- Suora dynaaminen linkitys
- DLL suoritetaan osana kutsuvaa prosessia käyttäen sen omaa aktivointitietuepinoa

DLL:n epäsuora dynaaminen linkitys

(implicit linking)

- kaikki viitatus moduulit ladataan (lataus aloitetaan) virtuaalimuistiin ja niihin viitataan staattisesti linkitetyn pienemmän liitospalikan (import library) avulla

```
...  
call stubS  
...  
stubS: "Wait until load S done"  
call S  
exit
```

31/05/2004

Copyright Teemu Kerola, K2003

33

DLL:n suora dynaaminen linkitys

(explicit linking)

- koodiin generoidaan suoraan viitepaikalle käskyt, joiden avulla linkitys tapahtuu tarvittaessa
- DLL ladataan vain jos siihen tulee viittaus

```
....  
"link S"  
"load S"  
call S  
.....
```

31/05/2004

Copyright Teemu Kerola, K2003

34

Nimien sidonta ⁽³⁾

(binding time)

- Milloin symbolin L suoritusaikainen muistiosoite sidotaan (lasketaan valmiiksi)?
 - ohjelman kirjoitusaikana?
 - käännösaikana?
 - linkityksessä?
 - latauksessa?
 - kantarekisterin asetuksen aikana?
 - osoitteen sisältämän konekäskyn suoritusaikana
- Jos muuttujan sijaintipaikkaa siirretään sitomisen jälkeen, mennään metsään ...!



virtuaaliosoite

Sijainnista riippumaton koodi

(position independent)

- Jos koodi siirretään toiseen paikkaan, niin mitään osoitetta ei tarvitse päivittää
- Kaikki muistiviitteet ovat
 - absoluuttisia (esim. keskeytyskäsitelijän osoite)
 - suhteessa PC:hen
 - tai pinossa
- Siellä ei ole viittauksia mihinkään koodiin tai tietorakenteeseen suorien (fyysisten muistiosoitteiden) avulla (tähän koodisegmenttiin)

Lataus (4)

- Ajomoduurista luodaan suorituskelpoinen prosessi (rakennetaan PCB ja sen viitteet kuntoon)
- Prosessin koodialueet (tai ainakin sen pääohjelma) ja tarvittava ladataan muistiin, prosessi siirretään R-to-R jonoon
- Sitten kun prosessi saa suoritustuoron suorittimella, MMU ja laiterekisterit ladataan PCB:n avulla tämän prosessin tiedoilla
 - virtuaalimuistia käytettäessä nimien sidonta tehdään viime hetkellä (konekäskyn suoritusajana) MMU:n avulla

31/05/2004

Copyright Teemu Kerola, K2003

37

-- Jakson 10 loppu --

Andy	14025	0
Anton	31253	4
Cathy	65254	6
Dick	54185	0
Erik	47357	6
Francois	56445	3
Frank	14332	3
Gerrit	32334	4
Hans	44546	4
Henri	75544	2
Jan	17097	5
Jaco	64533	6
Maarten	23267	0
Reind	63453	1
Roel	76764	7
Willem	34544	6
Wibren	34344	1

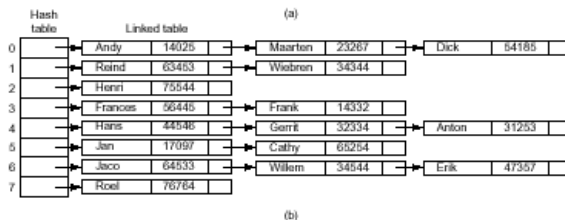


Figure 7-12. Hash coding. (a) Symbols, values, and the hash codes derived from the symbols. (b) Eight-entry hash table with linked lists of symbols and values.

[Tane99]

31/05/2004

Copyright Teemu Kerola, K2003

38