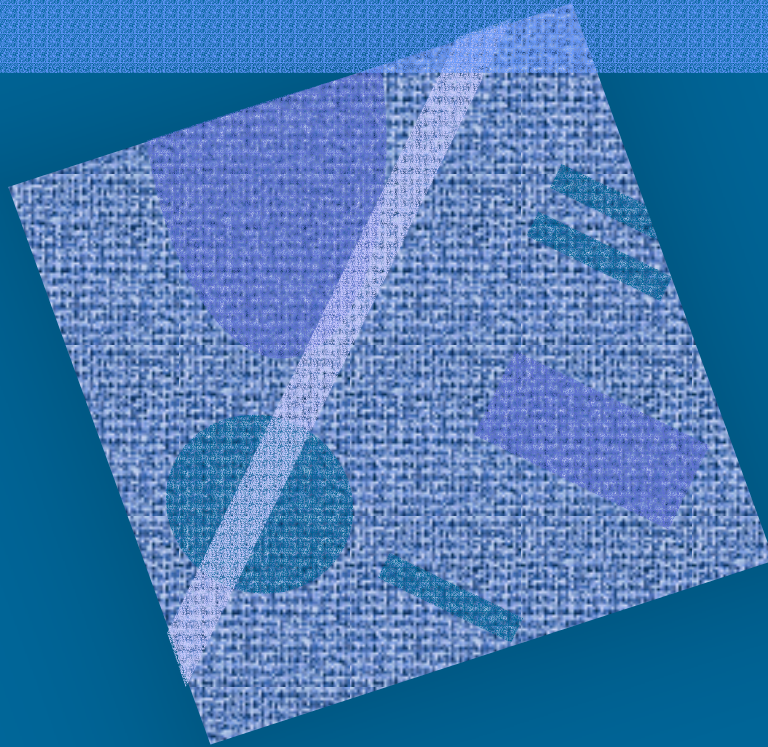


Luento 10

Käännös, linkitys ja lataus



Käännös

Linkitys

Dynaaminen linkitys

Lataus

Lausekielestä suoritukseen (3)

Käännös
lausekie-
lestä

Linkitys
muiden
ja kirjasto-
moduulien
kanssa

Lataus
muistiin
prosessia
varten

Käännösyksikkö

myprog.c

Lausekielinen ohjelma tai moduuli
osoitteet: symbolit

Objektimoduuli

myprog.obj

Käännetty ohjelma (konekielellä)
osoitteet: lineaariset (per moduuli)

Ajomoduuli

myprog.exe

osoitteet: lineaariset,
osa puutteellisia (?)

Prosessi

Suorituskelpoinen ohjelma
osoitteet: lineaariset (virt. avar.?)

prog.c

prog.

math.l

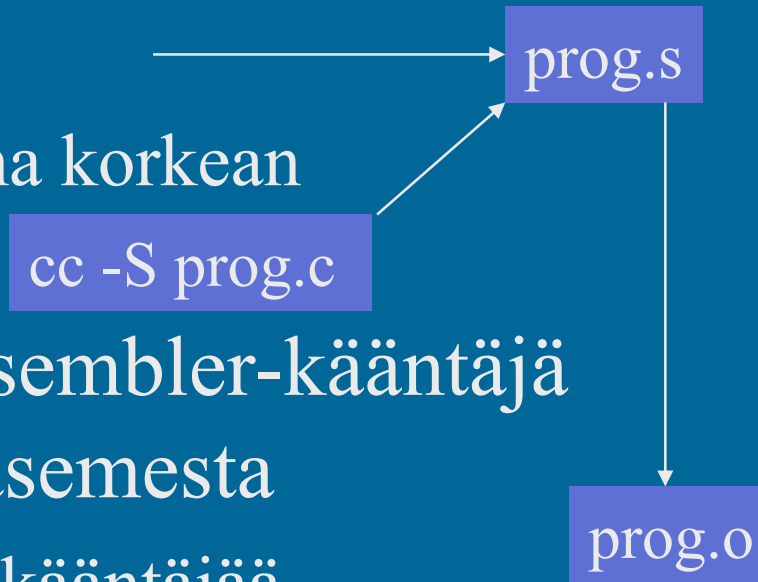
prog

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 liitoksien suunnitteluun ja toteutukseen muiden moduulien kanssa
- Käännösyksikön ohjelmointikieli ei ole tärkeä
 - niiden sitominen yhteen tapahtuu objektimoduulien tasolla

Assembler-kielinen käännösyksikkö (2)

- 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



Objektimoduuli (8)

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

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

Lähdekieli vs. konekieli (3)

- 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]

(Assembler) kääntäjän ohjauskäskyt (4)

- Eivät varsinaista koodia
 - niistä ei tule konekäskyjä
- Ohjaavat käännöstä

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 ks. Fig. 7-4 ja 7-6 [Tane99]
 - aliohjelmien prologi ja epilogi
 - itse tehdyt, kääntäjän käyttämät
- Erot aliohjelmiin ks. Fig. 7-5 [Tane99]

Literaalit (5)

- Vakioita
- 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 yhteyteen talletettuna
Pi DC 3.14159265 ; (!!??)
One DC 1 vrt. One EQU 1
OneMeg DC 1024576
- Niitä ei saisi muuttaa
LOAD R1, One
ADD R1,=1
STORE R1, One ; ask for trouble

Literaalit (2)

- 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? `Java string?`
- Myös joissakin assemblerkielissä literaalien implisiittinen (automaattinen) määrittely
 - helpommin luettavaa koodia `Load R14, =F'234567'`
 - literaalin 234567 tilanvaraus automaattisesti

Assembler käännös (10)

- 1. vaihe:
 - laske käskyjen tilanvaraukset
 - ttk-91 helppoa, koska kaikki käskyt 4 tavua!
 - generoi symbolitaulu ks. Kuva 6.2 [Häkk98]
 - arvot, arvon vaatima tavumäärä
 - uudelleensijoitustiedot (omana tauluna?)
 - generoi tai käytä muita tauluja
 - literaalitaulu (tilanvaraus lopuksi)
 - kääntäjän ohjauskäskytaulu
 - operaatiokooditaulu

Assembler käännös (8)

- 2. vaihe

- generoi lopullinen objektimoduuli

ks. Kuva 6.3 [Häkk98]

- tulosta symbolinen konekielinen listaus
- generoi taulut linkitystä varten

ks. Fig. 7-16 [Tane99]

- osana objektimoduulia

- anna virheilmoitukset

- 3. vaihe

- koodin optimointi

- voi olla oikeasti ennen 2. vaihetta tai sen yhteydessä

TTK-91 Assembler käänös, 1. vaihe



```
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
```

tunnetaan

Taas = 0

i = ?

s = ?

Symbolitaulu 1. vaiheen aikana

ks. kalvo 14

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

Kone-
käskyt
2 ja 8

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

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	; i = 11

Kaikilla
symboleilla
tunnettu arvo

Symbolitaulu 1. vaiheen jälkeen:

ks. kalvo 16

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

Koodi 2. vaiheen jälkeen:

	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
 - käänнос päivämäärä
 - 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

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

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

Korkean tason kielen käännös (7)

- Enemmän vaihteita

- Syntaktisten alkioiden etsintä

- Syntaksipuun generointi ja jäsenys

- Lauseiden tunnistaminen syntaksipuun avulla

- Välikielen (välikoodin) generointi (ei aina)

Välikieliesitys ja symbolitaulut

- Koodin generointi

- ei (yleensä) Java-ohjelmille

Lisää
tietoa?



Kääntäjien
ja ohj. kielten
kurssit

ks. syntaksipuun,
jäsenyspuun
esimerkit

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

Linkitys esimerkki (4)

- Neljä moduulia: A, B, C ja D ks. Fig. 7-14 [Tane99]
- Laske joka moduulille uudelleen-
sijoitusvakio (moduulin alkuosoite) (relocation
constant)
- Lisää k.o. vakio kunkin moduulin sisäisiin viitteisiin
- Etsi kaikki moduulien väliset viitteet, ks. Fig. 7-15 (a) [Tane99]
ja aseta kyseisten viitteiden osoitteet oikein ks. Fig. 7-15 (b) [Tane99]

Muuttujan X viittausten päivitys ⁽³⁾

- Miten löytää linkityksen aikana kaikki kohdat, jossa muuttujaan X viitataan?
- Vastaus 1: taulukko, jossa kaikki kohdat listattu
 - taulukko voi olla hyvin iso
 - kaikille muuttujille tilaa maksimitarpeen 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?

Muuttujan X viittaukset linkitettyinä listana ⁽¹⁾

symbolitaulu, moduuli ABC

Symb	sij	1. viittaus
X	700	23

lähdekoodi, moduuli ABC

```
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
```

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 koodin optimointia ei suoriteta joka kerta
 - esim: pelissä tasojen 8-22 moduulit, kun aloittelija ei pääse tasoa 3 ylemmäksi vielä kuukausiin

Dynaaminen linkitys (3)

- 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 (4)

- DLL - Dynamically Linked Library

- koodia, dataa tai molempia

.dll	yleinen tapaus
.drv	driver
.fon	font

- Säästää tilaa myös yhteiskäytön vuoksi

- Helpompi korjata virheitä

ks. Fig. 7.19 [Tane99]

- ei tarvita uutta käännöstä eikä lähdekielistä koodia!

- riittää kun DLL vaihdetaan uuteen

- seuraavassa suorituksessa uusi versio käyttöön

- Ajomoduuli kootaan kuten tavallinen objektimoduuli

- DLL moduulit ja DLL viittaukset merkitty erikoislipukkeella (huomioidaan linkityksen yhteydessä)

Windows DLL:n linkityksen kaksi tapaa (3)

- Epäsuora dynaaminen linkitys
- Suora dynaaminen linkitys
- DLL:ssä oleva koodi 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

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  
...
```

Nimien sidonta (2)

(name binding)

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

≠
virtuaaliosoite

Sijainnista riippumaton koodi ⁽³⁾

(position independent code)

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

Lataus (4)

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

-- Luennon 10 loppu --

Andy	14025	0
Anton	31253	4
Cathy	65254	5
Dick	54185	0
Erk	47357	6
Frances	56445	3
Frank	14332	3
Gerrit	32334	4
Hans	44546	4
Henri	75544	2
Jan	17097	5
Jaco	64533	6
Maarten	23287	0
Reind	63453	1
Roel	76784	7
Willem	34544	6
Wiebren	34344	1

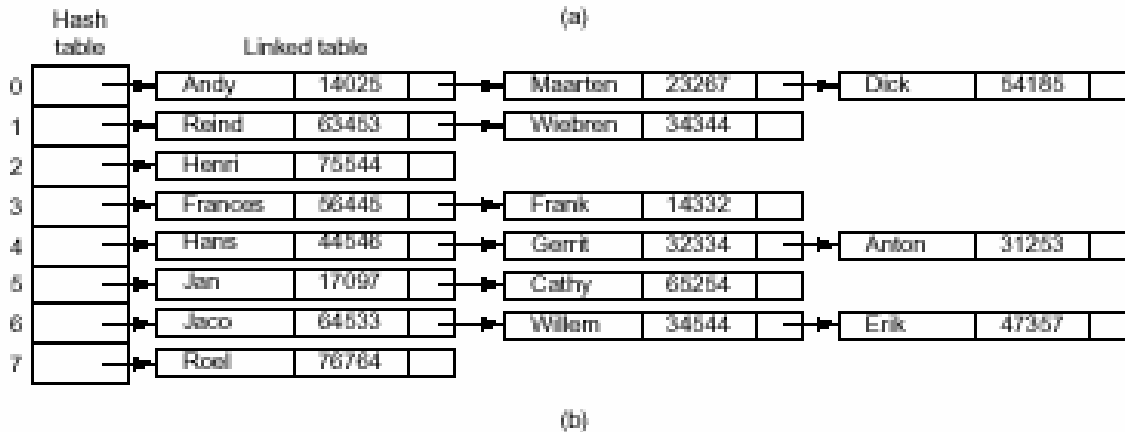


Figure 7-12. Hash coding. (a) Symbols, values, and the hash codes derived from the symbols. (b) Eight-entry hash table with linked lists of symbols and values.

[Tane99]