Tampereen yliopisto

Tero Mustakoski

KATSAUS LINKKEREIHIN JA LATAAJIIN

Kandidaatintyö
Informaatioteknologian ja viestinnän tiedekunta
Toukokuu 2025

TIIVISTELMÄ

Tero Mustakoski: Katsaus linkkereihin ja lataajiin
Kandidaatintyö
Tampereen yliopisto
Tieto- ja sähkötekniikan kandidaattiohjelma
Toukokuu 2025

Mitä tarkoittavat termit linkkeri, lataaja, objektitiedosto tai kirjasto? Linkkeri on ohjelma, joka kokoaa objektitiedostoista ja/tai kirjastoista suorituskelpoisen tiedoston. Lataaja taas on käyttöjärjestelmän osana oleva ohjelma, joka lataa suorituskelpoisen tiedoston eli ohjelman keskusmuistiin. Objektitiedostot ovat kääntäjän tai assemblerin tuottamia tiedostoja tietyssä tiedostoformaatissa. Ne sisältävät mm. binäärisiä konekäskyjä suorittimelle. Kirjastot taas ovat joukko objektitiedostoja yhdessä.

Työssä käydään läpi termejä vuosikymmenten varrella esiintyneiden lähteiden avulla. Aihepiiriä on pyritty käsittelemään mahdollisimman yleisellä tasolla, ilman liikoja toteutusyksityiskohtia. Työn avulla esimerkiksi aloitteleva ohjelmoija voi saada selkeämmän kuvan käännettävän ohjelman vaiheista ennen suoritukseen päätymistä.

Avainsanat: linkkeri, lataaja, objektitiedosto

Tämän julkaisun alkuperäisyys on tarkastettu Turnitin OriginalityCheck -ohjelmalla.

TEKOÄLYN KÄYTTÖ TÄSSÄ TYÖSSÄ

Opinnäytteessäni on käytetty tekoälysovelluksia:

- ☒ Ei
- ☐ Kyllä

Riskien tiedostaminen

Olen tietoinen siitä, että olen täysin vastuussa koko opinnäytteeni sisällöstä, mukaan lukien osat, joiden tuottamisessa on hyödynnetty tekoälyä, ja hyväksyn vastuun mahdollisista tästä seuranneista eettisten ohjeiden rikkomuksista.

SISÄLLYSLUETTELO

1. Johdanto 1
2. Koontiversion muodostamisesta 3
2.1 Yleiskuva 3
2.2 GCC 4
3. Linkkereistä 6
3.1 Yleistä linkkereistä 6
3.2 Joitakin käytössä olevia linkkereitä 7
4. Objektitiedostoista ja kirjastoista 9
4.1 Objektitiedostot 9
4.2 Kirjastot 10
5. Lataajista 12
6. Lähteiden tyylistä 13
7. Yhteenveto 14
Lähteet 16

1. JOHDANTO

Tietokoneohjelmien tuottaminen ja ajaminen on moderneissa tietokonejärjestelmissä tehty varsin vaivattomaksi ja automatisoiduksi. Ohjelmistokehitystyökalujen toiminnasta voidaan kuitenkin löytää loogisia askeleita, joilla suoritettava ohjelma rakennetaan vaihe kerrallaan. Vastaavasti käyttöjärjestelmissä täytyy olla jokin mekanismi, jolla ohjelmia ajetaan. Ihmisten ymmärtämät käännettävät ohjelmointikielet (esim. C, C++ tai erilaiset assembly-kielet) käännetään konekieliseksi objektikoodiksi. Näistä objektitiedostoista linkataan eli kootaan suorituskelpoinen ohjelma. Käyttöjärjestelmässä on puolestaan ohjelma nimeltään lataaja, joka ymmärtää suoritettavan ohjelman tiedostomuodon ja lataa ohjelman keskusmuistiin. Tietokoneen suoritin osaa suorittaa ohjelman konekieltä, joka on sarja käskyjä ja dataa loogisina ykkösinä ja nollina. [1] Tulkattavilla ohjelmointikielillä tehtyjen ohjelmien käsittely poikkeaa edellämainitusta, eikä sitä käsitellä tässä työssä.

Kääntäjäohjelmien toimintojen periaatteita käsittelevät tekstit keskittyvät usein käännösaikaiseen toimintaan. Linkkerit käsitellään varsin kompaktisti. Käyttöjärjestelmien opetuksessa lataaminen on melko niukasti käsitelty. [2][3] Linkitys ja lataus on nähty melko triviaaleina prosesseina, mutta asiat ovat nyttemmin monimutkaistuneet mm. olio-ohjelmoinnin, muistinhallintatekniikoiden ja jaettujen kirjastojen seurauksena [2]. Levine suositteleekin linkkereiden ja lataajien periaatteiden opettelua ohjelmoijille, ohjelmointikielten kehittäjille, käyttöjärjestelmäsuunnittelijoille, kääntäjien tekijöille ja tietenkin linkkereiden kehittäjille [2]. Lakos menee vielä pidemmälle ja sanoo, että ilman perustavanlaatuista ymmärrystä lähdekoodin muuntautumisesta ajettavaksi ohjelmaksi on mahdotonta tehdä isoja, skaalautuvia järjestelmiä [4]. Ainakin linkkausta ymmärretään huonosti vaikka linkkereiden ominaisuuksia käytetään oikeiden ohjelmistojen koodikannoissa [5][6].

Tässä työssä selvitetään, mikä on linkkeri, lataaja, objektitiedosto ja kirjasto. Työ tehdään kirjallisuuskatsauksena. Andorissa on käytetty hakusanoja ”linker AND loader”, ”linker AND compiler” sekä ”linker AND embedded”. Koska lähteiden ikä vaihtelee paljon, on myös tavoitteena selvittää, onko kerrontatavoissa tapahtunut muutoksia vuosien varrella.

Luvussa 2 esitellään ohjelman muodostuminen koontiversioksi askeleittain yleisellä ta-

solla. Luvut 3-5 keskittyvät tarkemmin kuvaamaan linkkereiden sekä lataajien toimintaa ja niihin liittyviä tiedostomuotoja. Luvussa 6 esitetään, minkälaista em. asioiden esittäminen on ollut eri lähteissä. Yhteenvedossa puolestaan kerrataan työn keskeiset havainnot.

2. KOONTIVERSION MUODOSTAMISESTA

Yksittäinen käännettävällä ohjelmointikielellä kirjoitettu tietokoneohjelma täytyy muuttaa lähdekoodista suorituskelpoiseksi tiedostoksi, mikäli se halutaan ladata keskusmuistiin ja laittaa suoritukseen. Tällaista yksittäistä suorituskelpoista tiedostoa kutsutaan koontiversioksi.

2.1 Yleiskuva

Korkean tason ohjelmointikielillä on kätevä kirjoittaa tietokoneohjelmia, sillä tietokoneen yksityiskohtaista toimintaa abstrahoidaan piiloon. Suoritin voi suorittaa kuitenkin vain käskymuistissa olevaa konekieltä. Väliin tarvitaan käytännössä muita ohjelmia, jotka auttavat tässä muutosprosessissa (kuva 2.1) [1][7]. Nämä ohjelmat toimivat yleensä integroidun ohjelmointiympäristön taustalla.

Kääntäjällä lähdekoodi käännetään jonkin suoritinarkkitehtuurin assembly-kieliseksi koodiksi. Tämän jälkeen assembleri tekee oman käännöksensä konekieliseksi binääritiedostoksi. Näitä tiedostoja kutsutaan objektitiedostoiksi, ja ne sisältävät konekäskyinä samat asiat kuin lähdekoodimoduuli. Ohjelmoijat jakavat yleensä lähdekoodinsa moduuleihin, jotta kokonaisuus olisi hallittavissa. Lisäksi usein käytetään muiden tekemiä ohjelmointikirjastoja, jotka ovat myös konekielisiä objektitiedostoja. Linkkeri kokoaa objektitiedostot yhdeksi suorituskelpoiseksi ohjelmaksi. Ohjelmaa tässä yhteydessä sanotaan buildiksi eli koontiversioksi.

Käyttöjärjestelmän osana on lataaja, joka siirtää binäärisen ohjelman pysyväismuistista keskusmuistiin. Käyttöjärjestelmä huolehtii myös ohjelman suorituksesta. Lataajan pitää pystyä käsittelemään linkkerin sille tekemiä tiedostoja. Tätä varten on kehitelty erilaisia tiedostoformaatteja. Objektitiedostojen muoto muistuttaa usein myös suorituskelpoisen ohjelman formaattia, mutta modulaarisuudesta johtuen esimerkiksi eri tiedostossa olevien aliohjelmien muistiosoitteita ei tiedetä.

Kuvan 2.1 loogisiin askeleisiin liitetään myös ajalliset termit kirjoitusaikainen (tai editointiaikainen), käännösaikainen, linkitysaikainen, latausaikainen sekä suoritusaikainen. Ter-

Kuva 2.1. *Käännettävällä ohjelmointikielellä kirjoitettu yksittäinen tietokoneohjelma kulkee tyypillisesti kuvan vaiheiden läpi, ennenkuin se päätyy suorittimelle. Kuva on referoitu lähteestä [1].*

millä kirjoitusaikainen viitataan lähdekoodin muokkaamiseen, termillä käännösaikainen kääntäjän työskentelyyn, termillä linkitysaikainen prosessin linkkerivaiheeseen, termillä latausaikainen vastaavasti lataushetkeen ja termillä suoritusaikainen varsinaiseen ohjelman suorittamiseen. Jotkut kääntäjät tekevät myös suoraan objektitiedostoja, mutta niidenkin sisältä voidaan erottaa konekieleen kääntäminen omana osuutenaan.

2.2 GCC

Käytännön esimerkkinä kuvan 2.1 alkuvaiheista esitellään usein C-kielisen ohjelman rakentaminen GCC:llä Linux-ympäristössä [2][4][7][8]. GNU Compiler Collection eli GCC on kokoelma kääntäjiä [9]. Siihen läheisesti liittyy myös ohjelmointityökalujen kokoelma GNU Binary Utilities [10]. Tarkalleen ottaen ohjelmana GCC on *compiler driver*, eli eräänlainen käännösohjaaja, joka käskyttää muita ohjelmia toteuttamaan käännöksen eri vaiheita.

Seuraavaksi lukijalle esitelläänkin hieman tätä esimerkkiä, joka teknisyydestään huolimatta sisältää toimintoja, jolla koontiversion muodostumista voi tarkkailla käytännössä.

Ensimmäiseksi .c-päätteinen lähdekoodi menee esikääntäjään CPP (engl. C preprocessor) [9]. Esikääntäjä valmistelee lähdekoodin varsinaiselle kääntäjälle (engl. compiler proper). Se poistaa kommentit ja käsittelee #-alkuiset esikääntäjäkomennot eli direktiivit. Niitä ovat esimerkiksi otsikkotiedostojen sisällytykset, määrittelyt sekä makrot. GCC:stä saa halutessaan vivulla -E ulostuloksi .i-päätteisen esikäännetyn tiedoston. Esikäännöksessä GCC kutsuu CPP:tä.

Varsinaisessa käännösvaiheessa kokoelman C-kielen kääntäjä CC1 kääntää C:n assemblyksi monivaiheisessa prosessissa, joka on liian pitkä tässä työssä käsiteltäväksi [9]. Assemblykielisen .s-tiedoston saa ulos GCC:stä vivuilla -S ja -fverbose-asm. GCC kykenee myös ristiinkääntämiseen, mikä tarkoittaa kääntämistä jollekin muulle tietokonearkkitehtuurille kuin sille, jolla työskennellään koodin parissa.

Assembleri AS puolestaan tekee käännöksen konekieleen [10]. Ulostulona on objektitiedosto halutussa formaatissa. GCC:n vipu -c antaa ulostuloksi objektitiedoston .o-

tiedostopäätteellä.

Ilman erikoisia valintavipuja ohjelma rakentuu suoraan suorituskelpoiseksi ja linkityskin tapahtuu automaattisesti. GCC kutsuu linkkeriä LD [10]. Jos suorituskelpoiselle tiedostolle ei ole ilmoittanut erikseen nimeä, tulee sille nimeksi a.out.

3. LINKKEREISTÄ

Koontiversiota muodostettaessa tarvitaan apuohjelmia tekemään lähdekoodista askel kerrallaan muutos kohti suorituskelpoista ohjelmaa. Linkkeri on ohjelma, joka toteuttaa viimeisen askeleen tässä muutosprosessissa.

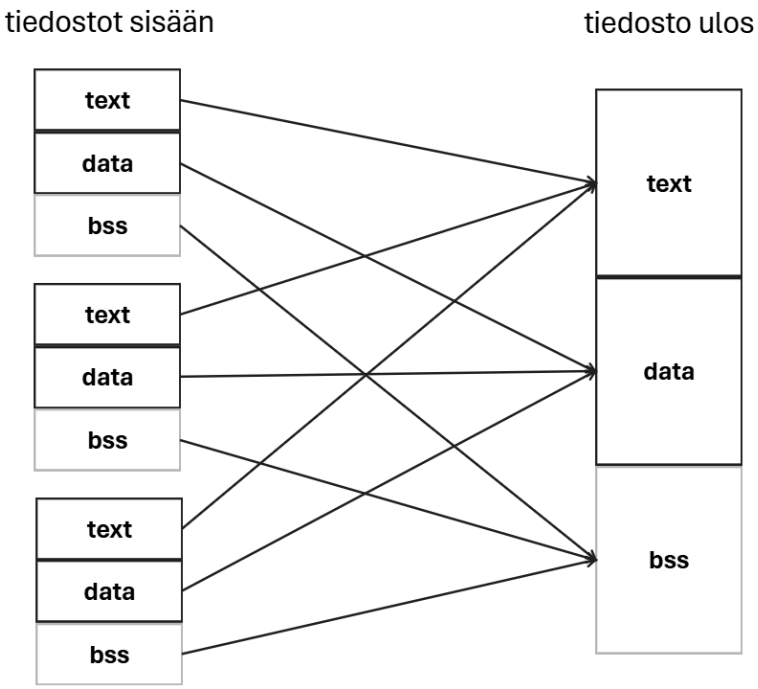
3.1 Yleistä linkkereistä

Linkkeri on kääntäjän tai assemblerin kanssa käytettävä ohjelma, joka yhdistää objektitiedostojen erilliset loogiset avaruudet yhdistettyyn loogiseen avaruuteen [11]. Linkkeri on ensimmäinen työkalu koontiversion muodostamisessa, joka käsittelee ohjelmoijan tuottamaa ohjelmaa yhtenä kokonaisuutena [2]. Linkkeri tekee suorituskelpoisen ohjelman kuvan, missä on otsikkotietojen lisäksi tyypillisesti aseteltu lohko konekäskyille (text), alustetuille muuttujille (data) ja alustamattomille muuttujille (bss) (kuva 3.1). Koska jo objektitiedostoissa on näitä samoja lohkoja, on selvää että ohjelmien sisäiset viittaukset pitää uudelleenjärjestellä. [1][2][4][7][8]

Linkkerin pitääkin muuttaa (engl. relocate) konekäskyjen muistisoitteet vastaamaan muuttunutta tilannetta. Tätä varten objektitiedostoissa on yleensä lista paikoista, missä on tarve osoitteiden uudelleentarkasteluun eli relokaatioinformaatio tai relokaatiotaulukko.

Objektitiedostoissa on tyypillisesti myös symbolista tietoa (esim. funktiot, globaalit muuttujat, lohkojen nimet) ja/tai viittauksia ulkoisiin symboleihin, joita objektitiedostosta itsestään ei löydy. Linkkerin täytyy etsiä kohteet kaikille symboliviittauksille, ja tehdä niistäkin taulukko. Symboleita etsitään myös kirjastoista. Linkkereistä löytyy yleensä tuki kirjastojen käytölle ja kirjastojen luomiseen. [2][7]

Linkkereitä voi ohjastaa komentoriviargumenteilla, linkkeriskriptitiedostoilla tai objektitiedostojen sisällä olevilla ohjeistuksilla [2]. Kuva 3.2 havainnollistaa linkkereiden syötteitä ja tulosteita.



Kuva 3.1. *Yksinkertaistettuna, näin linkkeri saattaisi linkata objektitiedostot ohjelmakuvaksi. bss-osuuksien suoritusaikainen koko ilmoitetaan, mutta dataa ei välttämättä tallenneta [4][12].*

3.2 Joitakin käytössä olevia linkkereitä

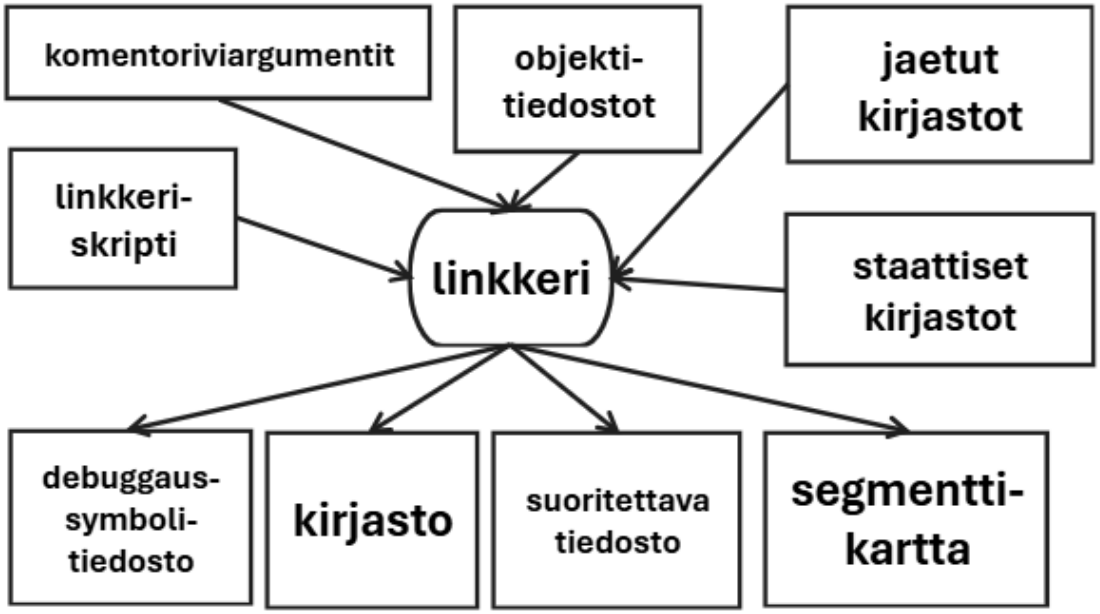
Mainittakoon joitakin nykyään käytössä olevia linkkereitä: LD, GOLD, LLD ja MOLD. LD eli GNU linker on luvusta 2.2 tuttu linkkeri, joka tulee GNU Binary Utilities -kokoelman mukana [10]. Se kehitettiin Cygnus Solutionsin toimesta GNU-projektiin [13].

GOLD on Googlen julkaisema ELF-linkkeri [14]. Se on GNU:n linkkeriä nopeampi, pelkästään ELF-tiedostoformaattia tukeva linkkeri. Se on suunniteltu olemaan LD:n korvike varsinkin moderneissa moniydinjärjestelmissä.

LLD puolestaan on LLVM-projektin linkkeri [15]. LLVM on kokoelma kääntäjätyökaluja, jotka saivat alkunsa Illinoisin yliopistossa. Sittemmin ne ovat levinneet laajempaan käyttöön. LLD:n kehittäjät väittävät sen olevan nopeampi, kuin aikaisemmat tässä mainitut linkkerit.

MOLD on uudempi tulokas linkkerimaailmassa. Se on tarkoitettu korvaamaan LD, kun tarkoituksena on linkittää käyttäjäavaruuden ohjelmia. Pääkehittäjä Ueyaman mukaan se on nopeampi, kuin tässä aikaisemmin mainitut linkkerit. [16][17]

On tärkeää, että linkkeri tuottaa oikeellisen ohjelman. Toiseksi olennaiseksi ominaisuu-



Kuva 3.2. *Linkkeriin voi syöttää monenlaista tietoa, ja linkkeri tuottaa erilaisia tiedostoja. Linkkeriskriptit ovat linkkerikohtaista komentoskriptikieltä linkkerin käytöksen ohjaamiseen. Komentoriviargumenteilla viitataan parametreihin, joita komentokehotteeseen voi kirjoittaa linkkeriohjelman perään. Objektitiedostot ovat modulaarisia binäärikoodia ja muuta tietoa sisältäviä ohjelmatiedostoja. Kirjastot sisältävät myös joukon yleishyödyllisiä ohjelmanpätkiä binäärikoodina. Debugattaessa eli virheiden etsinnässä ja poistossa voi olla hyötyä debuggaussymboleista. Segmenttikarttatiedostossa luetellaan ohjelman lohkojen muistipaikat ja koot sekä symbolien osoitteet. Aloittelevalle ohjelmoijalle kuitenkin tutuin tuotos lienee suoritettava ohjelmatiedosto. Kuva on referoitu lähteestä [2].*

deksi on noussut linkkerin nopeus käytössä. Varsinainen kääntäjä vaatii kyllä eniten aikaa, jos ensimmäistä kertaa muodostetaan koontiversio. Usein kuitenkin ohjelmoijan työ on syklistä: muokataan lähdekoodia — käännetään ohjelma — ajetaan virheenjäljittimellä ohjelma — muokataan uudestaan lähdekoodia ja niin edelleen. Jos kerralla muokataan vain yhtä lähdekoodimoduulia, on varsinaisen kääntäjän työ on pienempi. Linkkeri joutuu kuitenkin tällöinkin käsittelemään koko ohjelmaa yhtenä kokonaisuutena. Tämäntyyppisessä työskentelyssä linkkerin käyttämä aika nousee isompaan rooliin. [16][17]

4. OBJEKTITIEDOSTOISTA JA KIRJASTOISTA

4.1 Objektitiedostot

Kääntäjät ja assemblerit tuottavat lähdekooditiedostoista objektitiedostoja, joissa on binäärimuotoista koodia ja muuta relevanttia tietoa [2]. Syntyneiden objektitiedostojen määrä vastaa lähdekooditiedostojen moduulien määrää [7].

Objektitiedostoissa voi olla ainakin seuraavaa sisältöä:

- Otsikkotietoa: tekopäivämäärä, vastaavan lähdekooditiedoston nimi, taikanumero, lohkojen ja taulukoiden koot yms.,
- objektikoodia eli konekäskyjä ja dataa suorittimelle,
- relokaatiotietoa korjattavista osoitteista, mikäli esimerkiksi linkkeri siirtää objektikoodilohkoja,
- symbolit eli kyseisen objektitiedoston globaalit symbolit ja toisaalta muualta tarvittavat symbolit vaikkapa taulukkomouodossa, ja
- debuggaamiseen tarvittavaa tietoa.

Tämä lista ei ole tyhjentävä, mutta toisaalta objektikoodin lisäksi ei välttämättä tarvita muuta sisältöä. [2]

Objektiedostoille on kehitelty monia tiedostoformaatteja eli määritelmiä tallennettavan tiedon rakenteelle. Tietokonejärjestelmän toimintatavasta riippuen objektitiedostoformaatin tulee tukea linkitystä, suorittamista ja lataamista muistiin (esimerkiksi kirjastona). Formaatista riippuen tuetaan mitä tahansa kombinaatiota näistä kolmesta käyttötarkoituksesta. Saman järjestelmän sisällä siis myös suoritettavien tiedostojen ja kirjastojen tiedostoformaatit voivat noudattaa samankaltaista rakennetta objektitiedostojen kanssa. [2]

Unix-yhteensopivissa järjestelmissä on ennen ollut käytössä a.out-formaatti (engl. assembler output). Sitä ei tule sekoittaa luvun 2.2 suorituskelpoisen tiedoston oletusnimeen. Formaatti on ollut käytössä objektitiedostoihin, kirjastoihin ja suoritettaviin tiedostoihin liityen. Nykyään on käytössä yleistynyt ELF-tiedostomuoto (engl. Executable and Linkable

Format). [2][7]

Microsoft-ympäristöstä mainittakoon yksinkertaiset COM-tiedostot DOS-aikakaudelta ja nykyään käytössä oleva PE-formaatti (joskus PE/COFF, engl. Portable Executable/Common Object File Format), joka on jatkokehitelty Unixin COFF-formaatista. MacOS puolestaan käyttää formaattia nimeltään Mach-O (engl. Mach object file format). COM-formaatti on esimerkki tiedostoformaatista, joka sisältää pelkkää objektikoodia ja se ladataan ja suoritetaan sellaisenaan. Muut tässä mainitut formaatit ovat monimutkaisempia ja näitä formaatteja sovelletaan objektitiedostoihin, kirjastoihin ja suoritettaviin tiedostoihin. [2][7]

4.2 Kirjastot

Jo varhain tietokoneita ohjelmoitaessa havaittiin, että tiettyjä ohjelmarutiineja tarvitaan usein. Syntyikin ajatus näiden rutiinien uudelleenkäytöstä, jolloin saadaan uuteen ohjelmaan jo jonkun verran valmista, toimivaksi todettua ja käytössä testattua koodia. Samalla myös ohjelmistojen koon kasvaessa kompleksisuuden hallintaan tarvittiin modulaarisuutta, jota kirjastojen käyttö tukee. Näin ollen, nykyään useissa ohjelmointikielissä on saatavilla ohjelmointikirjastoja eli tässä tekstissä kirjastoja. [2]

Kirjastot ovat oikeastaan kokoelma objektitiedostoja. Yksinkertaisimmillaan kirjastot ovat vain sekvenssi objektitiedostoja. Nykyään, kun etsintää ei tarvitse suorittaa välttämättä sekventiaalisesti, kirjastoissa on hakemista helpottavia tietoja moduulien (eli objektitiedostojen) ja symbolien paikoista. [2][6]

Staattinen kirjasto linkitetään linkkerin toimesta kiinteästi suoritettavaan tiedostoon. Näin ollen jokaisella ohjelmalla on oma kopionsa kirjastosta. Jaetussa kirjastossa ajatuksena on puolestaan se, että käyttöjärjestelmä jakaa yhden kopion kirjastosta (tai ainakin osioista, jotka eivät muutu) keskusmuistiin. Tätä kopiota voi hyödyntää yksi tai useampi ohjelma samaan aikaan. Ainoastaan paikalliset muuttujat on oltava erikseen ohjelmien muistiavaruuksissa. [2][7]

Jaetut kirjastot pitää linkittää dynaamisesti latausaikaisesti tai ajonaikaisesti. Mikäli linkkeri ei löydä linkkausaikana symboleita syötetyistä objektitiedostoista, sen täytyy kuitenkin käydä verifioimassa jaetusta kirjastosta symbolin olemassaolo. Linkkeri voi tämän jälkeen esimerkiksi jättää objektitiedostoon lataajalle tiedon tarvittavista jaetuista kirjastoista sekä hyppytaulukon (engl. jump table), johon aikanaan täytetään tarvittavat referenssit kirjastokutsuille. [2] Taulukkkoon 4.1 on koottu tarkempaa vertailua staattisista ja jaetuista kirjastoista.

Taulukko 4.1. *Taulukossa on staattisten ja jaettujen kirjastojen keskeiset käyttöerot jossakin ohjelmassa. Jaettu kirjasto vähentää keskusmuistin tarvetta ja se tarvitsee päivittää vain kerran käyttöjärjestelmätasolla uuden version ilmentyessä. Järjestelmien välillä hankaluutta löytyy keskenään epäsopivien jaettujen kirjastojen versioiden käytöstä. Jaetun kirjaston käyttö voi olla ohjelman käynnistyksen tai suorituksen aikana hitaampi. Staattinen kirjasto on linkattava ohjelmaan mukaan joka kerta, kun kirjasto päivittyy ja jokaisessa ohjelman keskusmuistikuvassa on kertaalleen sama kirjasto. Staattisella kirjastolla varustettu ohjelma on kuitenkin yksinkertaisempi ladata keskusmuistiin ja siirtää toiseen järjestelmään. Taulukkoon on saatu inspiraatiota lähteestä [7].*

	Staattinen kirjasto	Jaettu kirjasto
Muistinkäyttö	Lisää	Vähentää
Päivitettävyys	Erikseen joka ohjelmaan	Kerralla käyttöjärjestelmään
Ohjelman Siirrettävyys	Mainio	Yhteensopiva kirjasto löydyttävä
Käynnistys, suoritus	Nopea	Dynaaminen, hidas

Vaihtelevaa määrää ohjelmia moniajava käyttöjärjestelmä ei voi taata jotain tiettyä muistiosoitealuetta jaetulle kirjastolle. Jaettu kirjasto ladataan keskusmuistiin (ja kuvataan sitä käyttävien ohjelmien mahdollisiin virtuaalisiin muistiavaruuksiin) siten, että siitä ei ole häiriötä muille muistiin talletetuille asioille. Tämän vuoksi jaettujen kirjastojen binäärikoodi on ns. paikkariippumatonta koodia (engl. position-independent code). Paikkariippumattoman koodin täytyy käyttää suhteellisia osoitteita absoluuttisten sijaan. [2][7]

Microsoft-ympäristössä dynaamiset jaetut kirjastot tunnistaa .dll-tiedostopäätteestä (engl. dynamic link library). Unix-puolella vastaavanlainen tiedostopääte on .so (engl. shared object). Staattiset kirjastot noissa ympäristöissä puolestaan ovat .lib- ja .a-päätteisiä. [1]

5. LATAAJISTA

Lataaja on käyttöjärjestelmän mukana tuleva ohjelma, joka sijoittelee suoritettavan ohjelman keskusmuistiin. Lataamisessa perustehtäviä ovat:

- Otsikkotietojen luku, jotta tiedetään paljonko keskusmuistia täytyy varata,
- muistin osoiteavaruuden allokointi ohjelman eri lohkoille,
- ohjelman lohkojen luku keskusmuistiin,
- bss-lohkon nollaus,
- pinomuistin luonti, jos arkkitehtuuri sellaisen tarvitsee,
- ajonaikaisen tiedon asettaminen (esim. ohjelman argumentit), sekä
- ohjelman käynnistäminen.

Muistinhallintaa sisältävissä arkkitehtuureissa on huomioitava sivutus ja mahdollinen tietojen jako sivuttain pelkästään luettavaan tietoon sekä tietoon, joka mahdollisesti muuttuu ohjelman suorituksessa. [1][2][6][7]

Lataaja voi olla absoluuttinen, jolloin absoluuttisessa binäärimuodossa oleva ohjelma ladataan aina samaan kohtaan muistiavaruutta. Absoluuttisella binäärimuotoisella ohjelmalla viitataan nimenomaan ohjelmaan, jossa muistiosoitteet on kirjoitettu absoluuttisina arvoina, ja suoritettavan ohjelman kuva on tarkoitus ladata sellaisenaan muistiin. [3][11]

Relokatoiva lataaja tulee kyseeseen, kun ei etukäteen tiedetä, mihin kohtaa muistiavaruutta ohjelma eri osineen ladataan (esimerkiksi jos on käyttöjärjestelmä, joka sallii useamman ohjelman ajamisen samaan aikaan). Tällaisen lataajan tulee käydä korjaamassa ohjelman konekäskyjen muistiosoitteita, jälleen kerran esimerkiksi relokaatiotaulukon avustuksella. [2][3][11]

Dynaamisesti linkkaavat lataajat lataavat esimerkiksi dynaamisesti jaettujen kirjastojen binäärikoodit keskusmuistiin, ja huolehtivat kirjastoista löytyvät symbolit ohjelman saataville. [2]

6. LÄHTEIDEN TYYLISTÄ

Lukemissani lähteissä ovat perusasiat säilyneet jokseenkin samoina vuosikymmeniä, sillä tietokoneohjelmien tuottamisen fundamentit eivät ole muuttuneet. Vaihtelua esiintyy jonkun verran sanastossa. Esimerkiksi termit kuten *binding*, *building*, *combining*, *collecting*, ja *linking* voivat tarkoittaa jokseenkin samaa asiaa [11]. Linkkeristä käytetään joskus nimitystä *linkage editor* tai *link editor*. Lataaja voi olla *loader* tai *program fetch* [11].

Oma vaikeutensa on keksiä suomennoksia termeille. Esimerkiksi sanan *section* (joita ovat mm. luvun 3 data, text ja bss) olen suomentanut lohkoksi, mutta en ole varma onko se paras suomennos. Tämä on taas eri asia kuin lähdekoodissa oleva koodilohko.

Joissakin lähteissä on käsitelty lähinnä sen aikakauden tietokonejärjestelmistä käsin asiaa. Esimerkiksi Presser ja White kirjoittavat paljon IBM System/360:n toiminnasta. Käytössä ovat luonnollisesti suunnilleen ne sanat, mitkä IBM on keksinyt asioille. [11] Toisaalta esim. Levine osaa monipuolisesti tuoda esiin erilaisten järjestelmien ratkaisuja [2].

Linkkerin ja lataajan toiminta riippuu paljon tietokonejärjestelmän arkkitehtuurista. Tästä on Levinen kirjassa kokonainen luku. [2] Tämä vaikeuttaa jonkun verran asian käsittelyä yleisellä tasolla.

Linkkerit ja lataajat eivät ole saaneet kovin paljoa huomiota Levinen ja Barronin kirjojen esipuheiden mukaan. [2][3] Tämän voi huomata vielä nykyisinkin. Esimerkiksi kirjassa *Compilers: Principles and Practice* on linkkerit ja lataajat mahdutettu liitteeseen C! [18] Poikkeuksena tästä on lähinnä Levinen kirja, joka omasta mielestäni on edelleenkin merkittävin lähde tämän työn aihepiiristä, joskin jo ikääntynyt sekin [2].

Perusasiat (varsinkin linkkereistä) löytyvät yleensä hyvin sulautettuja järjestelmiä käsittelevistä teksteistä, kuten Ouallinen kirjasta [8]. Syy tähän on ilmeinen: sulautettujen järjestelmien ohjelmia kehitetään yleensä työpöytäympäristössä, mutta ohjelmat päätyvät esimerkiksi mikrokontrollereihin, joissa on aivan eri prosessoriarkkitehtuurit ja omalaatuiset muistikartoitukset. Käsiteltäväksi tulee mm. luvussa 2.2 mainittu ristiinkääntäminen.

7. YHTEENVETO

Ihmisten on hankala tulkita isoa määrää ykkösiä ja nollia, joita tietokoneen suoritin käsittelee. Tätä varten on luotu ensin assembly-kieliä (ja assemblerit), ja vuosikymmenten mittaan on kehitelty myös erilaisia korkean tason ohjelmointikieliä ohjelmoijien työtä helpottamaan. Käännettävät ohjelmointikielet tarvitsevat kääntäjän, ja kääntäjiä onkin tutkittu paljon.

Kääntäjä on tietysti vain osa prosessia, jota oli kuvattu luvussa 2. Ohjelma kulkee lähdekoodista kääntäjälle, siitä assemblerille ja lopulta linkkerille. Käyttöjärjestelmässä tarvitaan myös lataaja lataamaan ohjelma keskusmuistiin.

Ohjelmistotyökaluja käytettäessä ei tarvitse välttämättä miettiä syvällisemmin taustatapahtumia, kun ohjelman koontiversio muodostetaan. Tässä työssä kuitenkin pureuduttiin kuvan 2.1 loppupuolelle, koska linkkereistä (luku 3), lataajista (luku 5) ja niihin liittyvistä tiedostomuodoista (luku 4) on hyvä saada perusymmärrys.

Luvussa 3.1 käsiteltiin linkkerin kolme perustehtävää (ja pari muuta asiaa): ohjelmakuvan muodostaminen yhteen loogiseen avaruuteen, konekäskyjen muistiosoitteiden korjaaminen (relokaatio) sekä ulkoisten symbolien paikannus. Luvussa 3.2 esiteltiin joitakin linkkereitä: LD, GOLD, LLD ja MOLD. Lisäksi huomattiin, että linkkereiden ominaisuuksista tärkeitä ovat ainakin oikeellisuus ja nopeus.

Luku 4 alkoi objektitiedostojen käsittelyllä. Ainakin konekäskyjä binäärimuodossa saadaan kääntäjältä/assemblerilta objektitiedostoon. Yleensä objektitiedostoissa on myös lohkojen kokotietoja ja relokaatio- sekä symbolitietoa. Puhuttiin myös tiedostoformaateista, jotka ovat linkitettäviä, ladattavia ja/tai suoritettavia tai yhdistelmiä näistä.

Aliluvussa 4.2 päädyttiin ohjelmanpätkien uudelleenkäytön ja modulaarisuuden myötä käsittelemään ohjelmointikirjastoja. Kirjastot ovat kuitenkin vain nippu objektitiedostoja. Aliluvun lopussa vertailtiin staattisia ja jaettuja kirjastoja.

Luvussa 5 käytiin lyhyesti läpi ohjelman keskusmuistiin lataamista. Lataajatyypeistä käydään läpi absoluuttinen lataaja, relokatoiva lataaja sekä dynaamisesti linkkaava lataaja.

Tämän työn lähteet ulottuvat 60-luvun lopusta nykypäivään. Linkkerit ja lataajat ovat hieman pienemmän (akateemisen) piirin mielenkiinnon kohteena, joten kirjallisuutta tulee vain harvakseltaan. Luvussa 6 mainittiin haasteina termien kirjavuus, arkkitehtuurisensitiivisyys ja suomentaminen. Toisaalta suppea perusasia on esitetty kohtalaisen hyvin useammassa lähteessä, eikä 90-luvun monimutkaistumisen jälkeen ole isoja muutoksia tullut perusteisiin. Levinen kirja *Linkers and Loaders* on kuitenkin mielestäni paras lähde, mikä tuli vastaan [2].

Kandidaatintyöhön on hankala mahduttaa kaikkea mitä lähteissä vastaan tuli. Tutkimuskysymykseen "mikä on linkkeri, lataaja, objektitiedosto ja kirjasto" saatiin kuitenkin vastaus. Joidenkin asioiden käsittelyä olen suorastaan yksinkertaistanut (esimerkiksi kirjastojen osalta), joten tässä työssä esitettyjä asioita ei tule ottaa lopullisina totuuksina.

Asiaa voisi jatkokäsitellä syventymällä melkein mihin tahansa osa-alueeseen tarkemmin. Pois jäi asiaa mm. debuggauksesta, linkitysaikaisesta optimoinnista, tiedostoformaattien tarkemmasta sisällöstä ja päällelatauksesta (engl. overlay). Eräs konkreettinen esiintullut asia on tarve linkkereiden nopeusvertailuun. Toivon kuitenkin, että lukija saa tästä työstä hyvän lähtökohdan linkkerien ja lataajien perusteisiin.

LÄHTEET

[1] D. A. Patterson ja J. L. Hennessy. *Computer organization and design RISC-V edition: the hardware software interface*. 2. painos. Elsevier, Cambridge, MA, 2021, 688 s.

[2] J. R. Levine. *Linkers and Loaders*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1999, 264 s.

[3] D. W. Barron. *Assemblers and Loaders*. Macdonald, London, 1969, 61 s.

[4] J. Lakos. *Large-scale C++. Volume I, Process and architecture*. Addison-Wesley, Boston, 2020, 1024 s.

[5] S. Kell, D. P. Mulligan ja P. Sewell. ”The missing link: explaining ELF static linking, semantically”. *SIGPLAN notices* 51.10 (2016), s. 607–623. ISSN: 0362-1340.

[6] C. W. Fraser ja D. R. Hanson. ”A machine-independent linker”. *Software, practice & experience* 12.4 (1982), s. 351–366. ISSN: 0038-0644.

[7] M. Stevanovic. *Advanced C and C++ Compiling*. Apress, Berkeley, CA, 2014, 326 s.

[8] S. Oualline. *Bare Metal C: Embedded Programming for the Real World*. No Starch Press, New York, 2022. ISBN: 9781718501621.

[9] *GCC online documentation*. https://gcc.gnu.org/onlinedocs/. Viitattu 13.11.2024. Free Software Foundation Inc., 2024.

[10] *Documentation for binutils 2.43*. https://sourceware.org/binutils/docs/. Viitattu 13.11.2024. Free Software Foundation Inc., 2024.

[11] L. Presser ja J. R. White. ”Linkers and Loaders”. *ACM computing surveys* 4.3 (1972), s. 149–167. ISSN: 0360-0300.

[12] P. Van der Linden. *Expert C programming : deep C secrets*. 1st edition. Englewood Cliffs, N.J: SunSoft Press, 1994. ISBN: 9780133522235.

[13] C. DiBona, C. DiBona, S. Ockman ja M. Stone. *Open sources : voices from the open source revolution*. First edition. Sebastopol, California: O’Reilly, 1999. ISBN: 9781306815765.

[14] I. L. Taylor. ”A new ELF linker”. Teoksessa: *Proceedings of the GCC Developers’ Summit 2008*. 2008, s. 129–136.

[15] *LLD - The LLVM Linker*. LLVM Developer Group. 2025. URL: https://lld.llvm.org/ (viitattu 13. 03. 2025).

[16] *mold: A Modern Linker*. GitHub. 2025. URL: https://github.com/rui314/mold (viitattu 13. 03. 2025).

[17] R. Ueyama. *Can the mold linker be /usr/bin/ld?* FOSDEM. 2024. URL: https://archive.fosdem.org/2024/schedule/event/fosdem-2024-2606-can-the-mold-linker-be-usr-bin-ld-/ (viitattu 13. 03. 2025).

[18] P. H. Dave ja H. B. Dave. ”Appendix C: Linkers and Loaders”. Teoksessa: *Compilers: Principles and Practice*. India: Pearson India, 2012. ISBN: 9788131776117.