Sovellusohjelmointi

ELEC-C7310

Assignment #1

Koodin siistiminen

Oppimispäiväkirja

Paavo Pärssinen
356644

Yleistä

Ensimmäisessä harjoitustyössä tärkeitä tavoitteita on muun muassa kirjoittaminen tiedostoon, signaalinkäsittely sekä lapsiprosessien luominen. Kirjoittaminen tiedostoon on jokseenkin tuttua aikaisemmilta ohjelmointikursseilta, mutta signaalinkäsittely ja lapsiprosessit on uutta. Työssä on myös muutamia lisätavoitteita, jotka pitää ottaa huomioon.

Luennoilla on melko hyvin esitelty kaikki tarvittava harjoitustyön tekemiseen ja luentomateriaalissa on muutamia esimerkkejä jotka tulevat helpottamaan harkkatyötä. Ongelmia kuitenkin voi aiheuttaa se, että kurssin aikana ei ole vielä käytännössä itse tehty mitään pienempiä tehtäviä aiheisiin liittyen. Ilman itse tehtyjä esimerkkitehtäviä tuntuu siltä, että hieman isomman tehtävän tekeminen kylmiltään voi olla haastavaa.

Tehtävän kuvaus

Valitsin ensimmäisen harjoitustyön aiheeksi ohjelmakoodin siistimisen; tavoitteena on kirjoittaa ohjelma, jolle syötetään komentorivillä tiedostojen nimiä. Jokaisen nimen saadessaan ohjelma avaa lapsiprosessin käsittelemään kyseistä tiedostoa. Tiedoston käsittelyn tavoitteena on poistaa kaikki kommentit koodista, eli käytännössä sen täytyy etsiä tekstistä merkkejä ”//” sekä ”/*” ja ”*/”. Merkkien ”//” jälkeen pitää luonnollisesti poistaa sitä seuraava rivi, ja ”/*” jälkeen taas poistetaan kaikki teksti ennen seuraavaa ”*/”-merkkiä. Muokattu teksti tallennetaan tämän jälkeen uuteen tiedostoon. Muuten tehtävä kuulostaa mielestäni melko helpolta, ainakin tekstin käsittelyn ei pitäisi olla erityisen vaikeaa.

Luulen että eniten ongelmia syntyy lapsiprosessien luomisessa, sekä C-ohjelmointikielen käytön uudelleenoppimisessa. C-ohjelmoinnin peruskurssista on jo jonkun aikaa, mutta käynnissä oleva C++ -kurssi sentään hieman helpottaa koodin kirjoittamisessa.

Suunnittelua

Työn tekeminen on varmaankin järkevää aloittaa kertaamalla luentojen aiheita ja esimerkkejä, sekä lukemalla netistä lisää juuri signaaleista ja lapsiprosesseista. Myös muita minulle tuntemattomia aiheita olisi hyvä tutkia jo heti alussa, mutta todennäköisesti alan tutkimaan näitä vasta kun ne tulevat työssä vastaan. Esimerkiksi oman kirjaston tekeminen on melko tuntematon ajatus, mutta luotan siihen että se onnistuu vähintään työn loppupuolella.

Kun olen saanut käsityksen lapsiprosessien käytöstä ja saanut sen toimimaan haluamallani tavalla on aika miettiä itse tekstinkäsittelyä. Käytännössä mieleeni tulee kaksi tapaa toteuttaa koodin siistiminen. Joko käsitellään tekstiä merkki merkiltä ja etsitään kommentteja, jätetään ne huomiotta ja kopioidaan muut. Toinen tapa voisi olla tekstin lukeminen rivi riviltä, ja yhden rivin käsittely kerrallaan. Rivien lukeminen voisi olla helpompi ainakin ”//”-tapauksessa, kun koko rivi on helppo jättää silloin käsittelemättä. Molemmissa tapauksissa siistittyä koodia lienee järkevää alkaa kirjoittamaan uuteen tiedostoon samaan aikaan kun vanhaa tiedostoa luetaan.

Kun edellämainitut tehtävän tärkeimmät asiat on tehty, alan miettimään muita vaatimuksia; esimerkiksi lokitiedoston kirjoittaminen, CTRL-C poistuminen, oma kirjasto jne.

Harjoittelua

Olen alkanut harjoittelemaan lapsiprosessien käyttöä; aluksi kirjoitin vain yksinkertaisen koodin joka käyttää fork() -komentoa ja tulostaa eri asiat lapsi-ja isäntäprosessin tapauksissa. Tästä hieman lisäämällä koodia päästiin melko helposti tilanteeseen, jossa isäntäprosessi pyytää käyttäjältä inputin stringinä, luo jokaisen inputin kohdalla lapsiprosessi ja lapsiprosessi tulostaa tekstin. Käytännössähän tämä on jo jokseenkin lähellä tehtävänannon suoritusta; lapsiprosessia pitää vain muokata avaamaan saamansa stringin niminen tiedosto ja käsittelemään sen!

Huomaa kyllä että ei ole hetkeen käyttänyt C-kieltä, Stringin käsitteleminen tuntuu kummalliselta ja vaikealta Pythonin ja C++:n jälkeen. Seuraavassa vaiheessa tehtävää tuleekin olemaan paljon tekstin käsittelemistä, kun aletaan siivoomaan kooditiedostoja, joten Stringit tulevat varmasti jälleen tutuiksi.

En ole vielä ehtinyt miettiä muita tehtävän vaatimuksia, esim. CTRL-C -signaalin käsittelyä ohjelmassa. Tätä tulee varmaan pohdittua vasta kun lapsiprosessi osaa jokseenkin käsitellä tekstitiedostoja.

Lisäys suunnitelmaan

Aloin pohtimaan lokitiedoston kirjoittamista; jos haluaa kirjoittaa yhteen lokitiedostoon kaiken mitä lapsiprosessit tekee, tulee vastaan ongelmia. Mikäli tämä toteutettaisiin siten, että jokainen lapsiprosessi avaa, kirjoittaa ja sulkee tiedoston, kävisi jossain vaiheessa niin, että kaksi prosessia kirjoittaa samaan aikaan tiedostoon.

Mahdollinen ratkaisu: siirretään putkilla lokikirjoitukset isäntäprosessille, joka on ainoa joka kirjoittaa lokitiedostoa. Ongelmia: missä vaiheessa ohjelman suoritusta isäntäprosessi kirjoittaa lokitiedoston? Kenties aina kun käyttäjä syöttää ohjelmalle jotain tai lopettaa sen? Tuleeko buffer vastaan nopeasti, minkä kokoinen sen pitää olla?

Olen nyt tehnyt nimetyn putken isäntäprosessin ja aliprosessien välille, testaukset jatkuvat.

Kokeilujen tulos

Päädyin totta tosiaan tekemään lokitiedoston kirjoittamisen käyttämällä putkia. Tällä hetkellä avaan heti ohjelman auetessa lapsiprosessin jonka tarkoituksen on vain ottaa aina kaikki putkesta ja kirjoittaa lokitiedostoon. Muut prosessit eivät koske lokitiedostoon. Kaikki muut prosessit käyttävä writeToPipe() -metodiani, joka työntää merkkijonon, sekä ajanhetken putkeen. Tämän jälkeen putkea vahtaava lapsiprosessi ottaa putkesta merkkijonon ja kirjoittaa sen lokin perälle.

Olen nyt myös saanut suurin piirtein toteutettua itse koodin siistijäprosessin. Luen prosessissa aina yhden rivin kerrallaan, sekä siitä rivistä kaksi kirjainta kerrallaan. Vertaan näitä ”//”, ”/*” ja ”*/” -merkkeihin ja päätän siten minkä kopioin uuteen tiedostoon. Lisäksi huomasin tiedostoon jäävän tyhjiä rivejä, joten päätin tehdä vielä lisämetodin joka poistaa tyhjät rivit. Lopputulos vaikuttaa lupaavalta.

Seuraavana vuorossa on CTRL-C -signaalin käsittely, sekä kaikenlainen virhemahdollisuuksien käsittely.

Viimeistelyä

Signaalien käsittely on mielestäni nyt toimivalla mallilla. Käytän main-funktiossa sigactionia, joka toimii mainin sisällä olevan fgetsin kanssa. Aluksi tuli hieman ongelmia kun koitin käyttää vain signal-funktiota sigactionin sijaan, jolloin ohjelma vain kaatui fgetsin kohdalla mutta jätti taustaprosessit jylläämään. Nyt ctrl-c toimii samalla tavalla kuin ohjelman oma exit-komento.

Toinen pieni ongelma joka aiheutti hieman pohdintaa oli mystinen lokitiedoston toimimattomuus. Ongelma olikin vain se, että ajoin nyt ensimmäistä kertaa koodin komentoriviltä saadakseni ctrl –c :n toimimaan, jolloin lokitiedosto tallentui eri paikkaan kuin yleensä. Yleensä käyttämäni Eclipse kun taas on tallentanut lokitiedoston yhtä kansiota ylemmäs.

Tein myös ensimmäistä kertaa makefile-tiedoston, en alkanut tekemään sen kummempaa kuin tiedoston joka kääntää kaksi .c -tiedostoa -Wall ja -pedantic lisäyksillä.

Aloin lisäämään ohjelmaan file locking -ominaisuuksia, tämä olisi ollut hyvä keino kirjoittaa lokitiedosto useissa prosesseissa (päällekkäisen avaamisen estäminen). Tällä hetkellä ohjelmassani on vain yksi aliprosessi joka kirjoittaa lokia, mutta toisaalta file locking estää tietenkin sen, että joku muu ohjelma ei voi käyttää tiedostoa kun siihen lisätään jotain. Eli ei pitäisi olla ihan turhaa hommaa. Implementoiminen vaikuttaa melko yksinkertaiselta.

Lopputulos

Koodi toimii kuten pitääkin, itse olen melko tyytyväinen tulokseen. Lokiin kirjoittaminen putkien avulla ei välttämättä ollut helpoin tai järkevin keino, mutta se toimii ainakin tässä mittakaavassa. Lisäksi opin jo hieman putkien käytöstä seuraavaa työtä varten.

Parannusehdotuksia löytyy muutamia. Tällä hetkellä koodin puhdistaja jättää aluksi tyhjiä rivejä koodiin, ja sen jälkeen poistaa kaikki tyhjät rivit. Tämä tietenkin saataa poistaa myös koodaajaan tarkoituksellisia tyhjiä rivejä jotka yleensä selkiyttäisivät koodia. Ratkaisu ei ole ihanteellinen, mutta ajanpuutteen vuoksi en kerennyt parantamaan mekaniikkaa. Ongelmiaa voi myös ilmetä, mikäli tiedoston nimi on huiman pitkä tai jotain vastaavaa. Virheenkäsittelyä olisi voinut tehdä lisää ellei aika olisi loppunut kesken.

Tärkeimmät opit tästä työstä itselleni on fork() -komennon käyttäminen, eli aliprosessit. Olenkin aikaisemmin miettinyt miten voidaan tehdä useita asioita kerralla, ja tämä selkiyttää asiaa. Putkien käyttö taas helpottaa jos halutaan siirtää tietoa ohjelmien tai aliprosessien kesken. Työn aikana myös C-koodin kirjoittaminen palautui melko hyvin mieleen.