Oppimispäiväkirja.

5.11.2023, Santeri Kokkonen, 907679

Tässä dokumentissa käsittelen sovellusohjelmoinnin assignment 1:sen suorittamis prosessia. Kirjoitan ensimmäiseksi kronologisesti tekemisistäni, testaamisesta ja lopuksi hieman pohdintaa tehtävästä yms. Kirjoitan tämän, kun olen jo viimeistellyt koodini ja olen läpäissyt testipenkin.

Progressio

Aloitin projektin lauantaina 28.10. Kun aloin lukemaan tehtävänantoa, ei se vaikuttanut mitenkään ylitse pääsemättömän vaikealta päässäni. Vain yksinkertainen lähetin-vastaanotin-pari. Kuitenkin kun aloin sitten tekemään tuli ilmi hyvin äkkiä, että tässä voi mennä hetki aikaa. Olin oppinut hyvin c-peruskurssilla c-koodausta, mutta tästä on jo puolitoista vuotta niin paljon on päässyt unohtumaan. Kanssa muistaakseni sillä kurssilla ei tehty kovinkaan isoja projekteja, ja täten kokonaisen c projektin suorittaminen monine source- ja header-kooditiedostoineen tuntui hankalalta.

Olin osallistunut kurssin lähes kaikille luennoille, mutta koska en ollut ennen projektin aloittamista ollut tehnyt vielä itse mitään koodia niin ei kyllä luennoistakaan ollut oikein muuta apua, kuin hento muistijälki. Toisaalta luentojen suurin hyöty oli se, että koska olin osallistunut melkein kaikille luennoille ja muistin, että ei niiden aikana käsitelty mitään erityisen vaikeaa niin pysyin rauhallisena tietäen, että ei tehtäväkään tällöin varmasti ole mikään mahdottomuus.

Itse koodaamisen aloitin tutkimalla annettua mockup tiedostoa. Ensimmäinen varsinainen koodauspäivä nyt lähinnä menikin siihen, että muistelin miten c koodi toimii. Myös aikaa vei paljon se, että opin compilemaan c-source koodi tiedostoja ja ylipäätänsä käyttämään linux terminaalia oikein. Nämä kuitenkin lopulta sain haltuun, ja ymmärsin miten ja miksi mockup koodi toimii, jolloin vihdoin pääsin ajattelemaan itse projektin aloittamista.

Aluksi paljon tutkin miten signaalit oikeastaan toimivat, ja miten esimerkkeinä annetut signal handlerit toimii. Sitten sen opittuani tapasin kurssikolleegoitani, ja kyselin heiltä paljon minkälaisia ideoita heillä oli esimerkiksi morse koodaus/dekoodaukseen, signal handlereihin yms. Samalla myös sain tärkeän vinkin kurssin opettajalta, että ei kannata luottaa ajastuksiin, vaan tehdä esimerkiksi jonkinlainen lähettäjän ja vastaanottajan välinen kuittaus systeemi.

Sitten lähdin työstämään. Ensin tein morse koodaus/dekoodaus funktiot erilliseen source tiedostoon. Otin tässä yksinkertaisen toimintamallin. Eli loin kaksikokoelmaa, ASCII- ja morsemerkit, joissa aina ASCII merkkiä vastaava morsetus oli toisessa kokoelmassa samalla indeksillä. Tällöin on helppoa vain etsiä ASCII-merkin/morsetuksen indeksi ja ottaa toisesta kokoelmasta arvo samalta indeksiltä. Ymmärrän, että tämä ei todellakaan ole tehokkain/elegantein tapa tehdä koodaus/dekoodaus, mutta ajattelen, että se on riittävä tähän tehtävään.

Signaloinnissa päätin, että vastaanotto- sekä lähetysprosessilla on omat signal handlerit, tällöin kummatkin kykenevät lähettämään ja vastaanottamaan. Tämä oli tärkeää, jotta vastaanottaja pystyy aina kuittaamaan saapuneet signaalit. Käytin signal handleri pohjana luento 4 koodia signal betterer. Päätin, että lähetin lähettää aina yhden morsemerkin kerrallaan, ja jää sen jälkeen odottamaan kuittausta. Taas vastaanottaja ottaa vastaan yhden merkin, ottaa sen talteen tarvittaessa ja lähettää kuittauksen. Tämä toimi alusta asti hyvin ja luotettavasti.

Ainoa ongelma tuli testipenkin pitkissä tiedostoissa. Olin ensimmäisessä iteraatiossa luonut lähettäjän signal handlerin niin, että se loi sen jokaiselle lähetettävälle ASCII merkille aina uudestaan. Tämä toimi virheettä

lyhyille testeille, mutta jostain syystä, mitä en tiedä, se ei toiminut pitkille. Kuitenkin ymmärsin, että ylipäätänsä se ei ollut tehokkain/elegantein tapa koodata lähettäjä-prosessin signal handleri, joten muutin sitä niin, että signal handleri luodaan main-source koodissa vain kerran, ja se pysyy samana kaikille lähetettäville merkeille. Tämän muutoksen jälkeen pitkätkin testit toimivat mainiosti.

Signaaleista vielä sen verran, että päätin, että lähetin käyttää 4 eri signaalia: SIGUSR1 ja SIGUSR2 morsemerkkien (- ja .) lähettämiseen, SIGALRM signaloimaan, että aloitetaan lähettämään uusi ASCII merkki, ja vielä SIGINT merkitsemään, että lähetys päättyy. Vastaanottaja vain lähetti SIGUSR1:stä kuittauksina.

Toinen ongelma mikä tuli vastaan, oli se, että olin debugannut ohjelmaa lähinnä käyttäen vain terminaalisyötettä, jolloin inputtiin sisältyi \n merkki. Koska tätä merkkiä ei ollut morsekokoelmissa, niin periaatteessa koko ohjelma meni tästä syötteestä sekaisin. No ensin ajattelin, että jos input = STDIN niin silloin luen vain yhden tavun vähemmän, mutta tästä aiheutui ongelma pipe-testeissä. Ohjelma jätti aina viimeisen tavun lukematta/kirjoittamatta. Lopulta päädyin ratkaisuun, että morsefunktiossa tarkistetaan, että löytyyhän syötetty merkki kokoelmista, jos ei niin ilmoitetaan siitä log tiedostoon, ja jätetään merkki huomiotta. Näin ollen ohjelmani pystyy käsittelemään välilyöntejä, rivin vaihtoja ja ääkkösiä, mutta kuitenkin niin, että se ei lähetä niitä.

Myös pieni ongelma oli se, että aluksi vastaanottaja prosessi sammutti koko ohjelman ja testipenkin saatuaan SIGINT signaalin, jolloin testipenkki suoritti ainoastaan ensimmäisen testin. Tämä korjaantui sillä, että määrittelin prosessin reaktion uudestaan SIGINT:ttiin.

Testaus

Kun kehitin koodia yleensä aina tein ohjelman yhden komponentin kerrallaan. Esimerkiksi tein aluksi lähettimen ja vastaanottimen erillisiksi ohjelmiksi. Sitten testasin, että ne toimii niin, että avasin 4 eri terminaalia. Yhdessä pyöri vastaanotin, toisessa lähetin, kolmannessa signal dump ohjelma, joka vain havaitsi tietyt sille lähetyt signaalit, eli tässä tapauksessa vastaanottimen kuittaussignaalit, ja neljännessä lähetin manuaalisesti kuittaus signaaleita lähettimelle. Ohjelmat aina printtasivat mitä signaaleita ne ottivat vastaan ja mitä ne tekivät. Näin pystyin saamaan varmistuksen, että lähetin ja vastaanotin toimivat halutun laisesti.

Muuten testaukseksi oli lähinnä syötteen lukemista terminaalista, ja paljon printf-käskyjä, jotta näin mitä ohjelma aina teki. Myös loin kaksi tekstitiedostoa, joilla testasin tiedonsiirtoa kahden tiedoston välillä. Näillä konsteilla homma toimi. Lopuksi toki aloin käyttämään testipenkkiä varmistamaan, että kaikki on mallillaan. Lisäksi katsoin, että ”-Wall -pedantic” parametreillä ei tullut compile varoituksia yms., ja tarkistin valgrind työkalulla, että ohjelma ei vuoda muistia.

Pohdintaa

Koodistani tuli lopulta mielestäni aivan kelpo ohjelma. Ehkä suurin itseä hieman häiritsevä tekijä koodissa on se, että sen olisi voinut tehdä siistimmän näköiseksi. Esimerkiksi main-source koodi on ehkä turhan pitkä ja useat fprintf&fflush käskyt vaikeuttavat sen luettavuutta. Kuitenkin tyydyin siihen, että tähän tarkoitukseen koodi on tarpeeksi hyvää, ja koska c source/header tiedosto rakenteet tulivat uusina asioina, en halunnut turhaan lähteä sekoittamaan lisää päätä. Jälkeen päin ajateltuna koodissa on aika paljon toisteisuutta, jota olisi voinut karsia paljon luomalle spesifejä funktioita.

Toinen parannuskohde on log tiedosto. Tyydyin tällä kertaa hyvinkin yksinkertaiseen log tiedostoon, johon tulee vain sanallisesti hieman ohjelman toiminnasta. Ensinnäkin siihen voisi lisätä esimerkiksi timestampit, tieto siitä kuinka paljon dataa on siirrettävänä ja kuinka paljon sitä onnistuneesti saadaan siirrettyä, ja vielä tarkat virhekuvaukset, jos sellaisia tulee. Kuitenkin koska tämä oli ensimmäinen kerta kun tein log tiedoston

johonkin koodiprojektiini, minulla ei ollut kovinkaan paljon aikaa/jaksamista ja näitä spesifejä ominaisuuksia ei suoraan vaadittu, niin päätin olla toteuttamatta ne. Nämä ovat sellaisia asioita mitä voisi lisätä ohjelmaan.

Muutenkin koodin jäi muutamia asioita, joita kuvailisin epäpuhtauksiksi. Sellaisia asioita, jotka voisi toteuttaa fiksummin. Esimerkkinä vastaanottimessa luon 9 tavua pituisen stringin aina yhden ASCII merkin morsetusta varten. Aina ASCII-merkkien välillä nollaan tätä stringiä manuaalisesti syöttämällä sen indekseille merkkejä '\0'. Koska käyttökokemusta ei hirveästi ole c koodin kanssa niin, päädyin tällaisiin erikoisiin ratkaisuihin. Myös aiemmin mainitut morse funktiot kuuluvat samaan kategoriaan. Löysin netistä, että esim morsedekoodaukseen on selvä puumalli, jonka avulla se olisi varmasti paljon tehokkaampaa. Nyt kanssa, jos kokoelmat menisivät jotenkin sekaisin, niin koko ohjelman toiminta vaarantuisi. Eli morse koodaus/dekoodaus on yksi parannuskohde.

Vielä mainitakseni, yleisestikin ohjelman tehokkuudessa on parannettavaa. Nyt testipenkin pitkiin testiin menee yli 10 s, eli tiedonsiirtonopeus on hyvin alhainen, ja olen varma, että sitä saisi kehitettyä paljon. Esimerkiksi tarkastelemalla erilaisia signaali vaihtoehtoja. Olen kuullut jostain jono ratkaisuista, mitkä voisivat olla nopeampia.

Viimeisenä huomiona se, että ymmärrän, että ohjelman kommentit, muuttujanimet ja log tiedosto on kirjoitettu sekaisin suomeksi ja englanniksi. Jos tämä ohjelma pitäisi viedä tuotantoon niin, nämä asiat pitäisi yhdenmukaistaa. Mutta ajattelen, että tällainen finglish ei varmaan kenenkään maata kaada, koska se on ollut jo käytäntönä kurssilla (diat englanniksi, mutta puhetaan suomea yms.), ja eikä sitä spesifioitu kummalla kielellä pitää tehdä.

Kuitenkin olen pääasiassa tyytyväinen ohjelmaani. Se toimii erittäin luotettavasti, en ole saanut kertaakaan virheellistä merkkiä, ja se on tarpeeksi nopea. Se on myös modulaarinen, ja siinä hyödynnetään globaaleja muuttujia. Siinä on myös extra, ei pyydettyjä, ominaisuuksia, kuten se, että syötteessä voi olla ei ASCII+-merkkejä ilman, että ohjelma lakkaa toimimasta.