ExepCode Assignment – Specim 2021
Koodaaja: Juha Kyllönen
p. 045-6330577
e-mail. kyllosen.juha@gmail.com

1. Aluksi luin tehtävänannon huolella, jonka jälkeen etsin Googlesta tietoa, mitä tarkoittaa ”a producer – consumer” -algoritmi. Aika nopeasti löysin netistä hyvän Wikipedia - sivun, jossa oli tästä aiheesta tietoa. Myös koodi-esimerkkejä Googletin lukuisia. Ensimmäinen päivä meni tutustuessa tähän aiheeseen ja tämän kooditehtävän sisäistämiseen.

2. Työalustana minulla oli Linux Raspberry Pi 4 B 4Gt. Käytin Code::Blocks 17.12 työkalua koodin kirjoittamiseen, kääntämiseen ja testaamiseen. Minulla on asennettuna GCC 9.1 ja sillä käänsin koodin C++17 kääntäjällä. Aluksi minun piti säätää muutamia asetuksia Code::Blocks :ssa.

3. Itse tehtävän toteuksen päätin tehdä luokkia ja perintää hyväksi käyttäen. Minulla tuli main.cpp tiedosto, yksi .h tiedosto, jossa on vakioita määritettynä (kuvan maksimi leveys MAX_WIDTH ja maksimi korkeus MAX_HEIGHT sekä muistipuskurien maksimi syvyydet MAX_MEMORYBUFFER_DEPTH sekä viallisten width-height parien maksimi talletusmäärä MAX_ERRORBUFFER_DEPTH, jolla voit määrittää, että kuinka monta viimeisintä viallista width lenght paria talletetaan puskuriin. Lisäksi toteutin neljä eri luokkaa (jokainen näistä luokista sisältää sekä .cpp että .h tiedostot).

4. Aloitin tekemällä MemoryBuffer luokan. Tästä luokasta periytyvät kaikki muut toteuttamani luokat. Tässä luokassa olen määrittänyt useita staattisia muuttujia, joita tästä luokasta periytyvät luokat käyttävät sitten yhteisesti. Laitoin nämä staattiset muuttujat Private: jäseniksi, mutta sallin ainoastaan ProCon luokan muuttaa suoraan näitä muuttujia. Tein tämän kirjoittamalla rivin ”friend class ProCon;” MemoryBuffer .h tiedoston loppuun. Tein näin, koska ainoastaan toteuttamani ProCon luokka saa muuttaa näitä staattisia muuttujia suoraan. Muistipuskureihin (toisessa vektorissa on data ja toisessa vektorissa on aina edellämainitun datan width ja height) voidaan kirjoittaa monesta eri säikeestä yksi säie aina kerrallaan näihin staattisiin 2D-vektoreihin (joka tapahtuu ProCon luokasta). Näissä 2D vektoreissa on maksimissaan MAX_MEMORYBUFFER_DEPTH -riviä. Tuon edellä olevan MAX_MEMORYBUFFER_DEPTH vakion arvo voidaan itse määrittää header tiedostossa ennen kääntämistä. Jokainen 1D-kuvavektorin alkio on unsigned short -tyyppiä eli jokainen pikseli esitetään 16-bitillä. Yksi unsigned short vektori sisältää siis yhden kokonaisen kuvan bitit yksiulotteisessa vector containerissa. Todellisen kuvan pikselit kirjoitetaan tuosta 1D -vektorista yksi kerrallaan, aina järjestyksessä vasemmalta oikealle ja ylhäältä alas, käyttämällä hyväksi talletettuja kuvan widht ja height arvoja. Säikeitä ja mutex:ia käyttämällä toteutin tämän tehtävän niin, että vain yksi säie voi kirjoittaa tai lukea aina vuorollaan puskureihin.

5. Toteutin tarvittavat producer ja consumer -funktiot samaan ProCon luokkaan. Tämä ProCon luokka perii MemoryBuffer luokan. Sekä producer että consumer funktioissa on staattinen (yhteinen) lock_guard ja mutex, jonka avulla näitä staattisia producer ja consumer funktioita voidaan käyttää ainoastaan yhdestä säikeestä aina vuorollaan. Itse muistipuskurit toteutin FIFO tyyppisesti, eli First-In – First-Out. Sekä producer että consumer -funktiot pitävät koko ajan kirjaa siitä, mikä on molemman muistipuskurin täyttöaste. Tyhjistä puskureista ei voida lukea ja täysiin puskureihin ei voida enää kirjoittaa. Producer funktio kirjoittaa aina pinojen päälle ja consumer funktio lukee aina pinojen alimmaisen vektorin. Sen jälkeen, kun consumer funktio on lukenut muistipuskureista alimmaiset vektorit niin se

tuhoaa alimmaiset muistipuskurien vektorit ja shiftaa molemmat pinot yhden rivin (indeksin) verran alaspäin. Tällöin muistipuskurien koko (syvyys) pienenee yhden rivin verran ja vastaavasti kun producer funktiota kutsutaan niin muistipuskurien koko kasvaa yhden rivin (syvyys) verran.

6. Sitten tein imaginarySDK luokan, jossa on mm. tuo vaadittu ”onDataCallback” funktio. Ennen kuin ”onDataCallback” -funktiota kutsutaan niin tarkistetaan erillisellä funktiolla, onko sille välitetyt width ja height muuttujat sallitut. Jos width ja height olivat sallitut niin tämän jälkeen ”onDataCallback” säie ohjataan ProCon::producer säikeeseen. Lisäksi tässä luokassa on oma ”readDataFromBuffer” -funktio puskurista lukemiseen. ImaginarySDK luokka perii ProCon luokan.

7. Viimeinen luokka on testOperation, joka perii ProCon luokan. Tässä testOperation luokassa on testScript funktio, jolla testasin tämän projektin toimivuutta. TestScript:llä tein aluksi testidataa, jolla täytetään testauksessa käytettävän 1D-vektorin elementit. Jos kaikki toimii niinkuin pitääkin niin aluksi tulostan konsoliin, että mikä säie on menossa (joko producer tai consumer). Sitten tulostan muistipuskurin täyttöastetta kuvaavat muuttujat eli fillCount ja emptyCount. Lopuksi tulostan jokaisessa sallitussa säikeessä olleet width ja height arvot. Seuraavaksi tulostan konsoliin viimeisimmän muistipuskurista luetun vektorin kaikki pikselit (row_index, column_index) ja aina kyseisen pikselin arvon. Tämän jälkeen tulostan, että kuinka monta kertaa producer ja consumer funktioita kutsuttiin eri säikeistä. Jos width ja height arvot eivät olleet sallitut niin tulostan siitä ilmoituksen, eikä näillä väärillä width ja height arvoilla kutsuta ”onDataCallback” -funktiota. Viimeisimpänä tulostan konsoliin kaikki (width, height) parien arvot, joissa oli jotain vikaa.

8. Main -funktiossa alkaa tämän ohjelman suoritus kutsumalla tuota 7. kohdan testScript funktiota.

9. Olen käyttänyt koodissani myös muutamia uusimpia C++11/14/17 kielen moderneja ominaisuuksia, mm. muuttujien alustus ”initializer list”:llä, ”auto” sekä ”ranged for-loop” vektorien käsittelyssä. Käytin tehtävän toteutuksessa ihan perus STL kirjastoja, kuten ”iostream”, ”vector”, ”thread” ja ”mutex” kirjastoja.

KOMMENTTEJA:
Tehtävä oli sopivan haastava. Mutta aika nopeasti sain lopullisen koodin toimimaan sekä kääntäjällä että ajettavana ohjelmana. Minulla ei tule enää yhtään error(s) eikä warning(s). Se on sellaista iterointia kääntäjällä, kun koodi ei heti käänny ilman virheitä. Tietysti koodin rakennetta kannattaa aina parantaa ja sitä että koodi olisi mahdollisimman puhdasta ja helppolukuista kaikille, jotka koodia tulevat joskus lukemaan.