Eindopdracht CPPLS2
versie 1.0, najaar 2020
AvanSync

Bob Polis, Avans Hogeschool, 's-Hertogenbosch

11 december 2020

Inhoudsopgave

1 Wat ga je maken?

Je gaat twee C++ command line tools (*console apps*) maken, waarmee je een eenvoudige "Dropbox"-functionaliteit implementeert. Het gaat om een *client* en een *server*, die over een TCP/IP-netwerk communiceren. Beide mogen op dezelfde computer draaien, wat voor het ontwikkelen en testen handig is. De client maakt in dat geval verbinding met *localhost*.

1.1 Gebruiker — client — server

De server beheert een vaste directory, die synchroon moet blijven met een andere, vaste directory bij de client. De client kan over het netwerk allerlei commando's geven om de situatie bij de server op te vragen en te veranderen. Denk daarbij aan het opvragen van directory listings, het downloaden of uploaden van files, het maken van directories, en hernoemen of verwijderen van files of directories.

De gebruiker heeft een interactieve sessie met de client, door op de command line commando's te geven. De client communiceert vervolgens met de server volgens een strak gedefinieerd protocol (zie §2.2, p. 4).

Je mag zelf bepalen op wat voor manier de gebruiker commando's geeft aan de client. Dat mag door de gebruiker commando's in te laten typen die door de client geïnterpreteerd worden. Het mag ook door in de client een menu-structuur te implementeren, waarbij je bijvoorbeeld steeds een genummerd lijstje met mogelijkheden presenteert.

Je moet echter wel de functionaliteit implementeren die hieronder in §2 wordt beschreven.

1.2 Networking over TCP/IP sockets

De code voor networking krijg je cadeau. Er zijn bij de opdracht source files gegeven voor een client en een server, die berichten kunnen uitwisselen. Deze code maakt gebruik van de header-only library Asio[1], en is getest en werkend bevonden op Windows, macOS en Linux. Voor de documentatie[2] gebruik je de standaloneversie, niet de versie die bij Boost zit.

Het mooie van deze library is onder andere dat je communicatie over networking sockets kunt doen volgens het iostream-model van C++. In de voorbeeldcode kun je dat ook gedemonstreerd zien.

2 Functionele eisen

Hieronder staan de functionele eisen uitgesplitst voor de client en de server. Hoe de client precies commando's naar de server stuurt, en hoe die antwoordt, is te vinden in §2.2, p 4.

2.1 Client

De client moet voor de gebruiker de volgende use cases implementeren:

[1] https://think-async.com/Asio/

[2] https://think-async.com/Asio/asio-1.18.0/doc/

1. *Informatie over de server opvragen.* De server stuurt een regel tekst terug die de client kan tonen.

2. *Een directory listing opvragen.* De gebruiker specificeert een lokale directory, en de client vraagt aan de server om de gegevens van de daarmee overeenstemmende directory aan de serverzijde. De server stuurt de gegevens terug, één item per regel, volgens het protocol hieronder.

3. *Een file downloaden.* De gebruiker specificeert een pad naar een lokale file, en de client communiceert met de server om de data op te halen van de daarmee overeenstemmende file aan de serverzijde en de lokale file daarmee te overschrijven.

   Eventueel kun je een optie inbouwen om de lokale file niet te overschrijven maar in plaats daarvan de data weg te schrijven in een nieuwe file met een andere (gegenereerde) naam.

4. *Een file uploaden.* De gebruiker specificeert een lokale file. De client overlegt dan met de server en stuurt de file data, die op de overeenkomstige plek aan de serverzijde de daar aanwezige file overschrijft, of, als die daar nog niet bestaat, creëert.

5. *Een file of directory hernoemen.* De gebruiker specificeert een lokale file of directory, en geeft voor dat item een nieuwe naam op. De client geeft dat door aan de server, die laat weten of het gelukt is. De client kan dat melden aan de gebruiker.

6. *Een file of directory verwijderen.* De gebruiker specificeert een lokale file of directory, en de client vraagt aan de server om het daarmee overeenstemmende item te verwijderen. De server laat weten of dat gelukt is. De client kan dat melden aan de gebruiker.

7. *Een nieuwe directory maken.* De gebruiker specificeert een directory waarin een nieuwe directory gemaakt moet worden, en geeft daarvoor ook een naam op. De client overlegt weer met de server, die dat aan de serverzijde op de overeenkomstige locatie voor elkaar brengt.

8. *Directory synchronisatie.* De bedoeling is hier om de directory aan de serverzijde gelijk te krijgen aan de lokale directory (*mirroring*). Het is de *client* die de synchronisatie uitvoert, en die heeft daarvoor de informatie nodig over de complete remote directory om die te kunnen vergelijken met de lokale directory.

   De client kan de synchronisatie realiseren met behulp van de beschikbare commando's voor de server. Door vergelijken van modificatiedatums kan de

client bepalen of een lokale file nieuwer is dan die aan de serverzijde, en die dan uploaden als dat het geval is.

Je zult dus recursief door de lokale directory moeten wandelen en daarbij steeds bepalen wat de benodigde acties zijn om de directory aan de serverzijde daaraan gelijk te maken.

9. *Verbinding verbreken.* De gebruiker laat weten te willen stoppen, en de client verbreekt de verbinding met de server volgens het protocol.

2.2 Het netwerkprotocol tussen client en server

De client en server communiceren via TCP/IP volgens een strak gedefinieerd protocol. Hieronder zie je de commando's die de client naar de server kan sturen, welke vorm ze moeten hebben, en hoe de server antwoordt.

Bij het verzenden en ontvangen van text data moet elke regelovergang gecodeerd worden als "`\r\n`", dus niet slechts "`\n`".

In de beschrijvingen hieronder wordt elke regelovergang aangegeven als "⁋".

`INFO`⁋ *Algemene info over de server.* De server antwoordt met een regel tekst, met daarin de naam en versie. Er mag ook een auteur en copyright genoemd worden. Bijvoorbeeld:
`AvanSync server 1.0, copyright (c) 2020 Bob Polis.`⁋

`DIR`⁋***path***⁋ *Directory listing.* De server stuurt eerst het aantal items, in de vorm van een getal in tekstvorm, afgesloten met een regelovergang (⁋). Daarna alle items in de directory, één per regel. Voor elk item worden enkele gegevens achter elkaar gezet, gescheiden door een verticale streep ("`|`").

Een item bestaat uit:

- een letter die het type aangeeft (`F` = file, `D` = directory, `*` = anders),
- de naam van het item,
- de modificatiedatum van het item in de vorm *yyyy-mm-dd hh:mm:ss*,
- de grootte van het item in bytes als het een file is; anders: 0.

Een geldige respons is bijvoorbeeld:

```
F|file1.txt|2020-03-05 23:45:18|34107⁋
F|file2.txt|2018-10-23 08:05:30|218⁋
D|Stuff|2020-11-11 16:25:00|0⁋
F|trainingdata.dat|2016-04-01 21:15:53|200⁋
*|my_symlink|2020-12-05 12:34:56|0⁋
```

Als de gevraagde directory niet bestaat is de respons:
`Error: no such directory⁋`

De server mag de verschillende items in willekeurige volgorde aanleveren.

GET⁋***path***⁋ *Download een file.* Het *path* moet naar een bestaande file verwijzen. De server stuurt eerst de file size in bytes, als een getal in tekstvorm. Daarna een regelovergang (⁋), gevolgd door het beloofde aantal bytes aan binary data. (Zorg er voor dat je precies dat aantal bytes ook echt leest, anders loop je kans dat de zaak gaat "hangen". Zie ook de tips op pagina 6.)
Als het *path* niet bestaat of geen file is, is de respons:
`Error: no such file⁋`
Als je geen leesrechten hebt is de melding:
`Error: no permission⁋`

PUT⁋***path***⁋***file size in bytes***⁋ *Upload een file.* De gebruiker heeft een lokale file aangewezen voor de client (die natuurlijk wel binnen de voor de client gebruikte vaste directory ligt). De client bepaalt de file size daarvan, in bytes, en stuurt bovenstaand commando naar de server. Het *path* is de eindbestemming van de file, en bevat dus de gewenste filename die hij op de server moet krijgen. De respons van de server is:
`OK⁋`
als het *path* geschikt is om de file te plaatsen (bestaande directory, schrijfrechten), en er voldoende diskruimte is. Anders is de response:
`Error: invalid path⁋`
of:
`Error: no permission⁋`
of:
`Error: not enough disk space⁋`
afhankelijk van wat er mis ging.

REN⁋***path***⁋***new name***⁋ *Hernoem een file of directory.* Het *path* moet naar een geldige file of directory verwijzen, die je een nieuwe naam wil geven. Als dit in orde is, en het hernoemen lukt, is de respons:
`OK⁋`
Anders krijg je een foutmelding, bijvoorbeeld:
`Error: no such file or directory⁋`
of:
`Error: no permission⁋`

DEL⁋***path***⁋ *Verwijder een file of directory.* Het *path* moet naar een geldige file of directory verwijzen en je moet permissie hebben om die te verwijderen. Als alles in orde is en het verwijderen lukt, is de respons:

`OK`¶

Anders krijg je een foutmelding, bijvoorbeeld:

`Error: no such file or directory`¶

of:

`Error: no permission`¶

`MKDIR`¶***parent dir***¶***dir name***¶ *Creëer een nieuwe directory.* De *parent dir* moet een geldige directory zijn waarin je schrijfrechten hebt. Als alles goed gaat en de nieuwe directory is gecreëerd met de opgegeven naam, is de respons:

`OK`¶

Anders krijg je een foutmelding, bijvoorbeeld:

`Error: no such directory`¶

of:

`Error: no permission`¶

`QUIT`¶ *Sluit de verbinding met de server af.* De verbinding wordt direct verbroken. De server stuurt geen respons meer.

Command syntax	**Response**
`INFO`¶	`AvanSync server 1.0`¶
`DIR`¶*path*¶	`F|file`¶`D|directory`¶`*|other`¶
`GET`¶*path*¶	*file size in bytes*¶*file data*
`PUT`¶*path*¶*file size in bytes*¶	`OK`¶
file data	`OK`¶
`REN`¶*path*¶*new name*¶	`OK`¶
`DEL`¶*path*¶	`OK`¶
`MKDIR`¶*parent dir*¶*dir name*¶	`OK`¶
`QUIT`¶	*n.v.t.*

Tabel 1: *Samenvatting van het protocol* (¶ = `\r\n`).

3 Tips

- Om files en directories te manipuleren gebruik je de classes en functies in `std::filesystem`[3].
- Houd je client- en server-projecten gescheiden, zodat je de executables gemakkelijk tegelijkertijd kunt laten draaien.

[3] https://en.cppreference.com/w/cpp/filesystem

- De Asio library is header-only; er is dus geen DLL.

  Bouw je met Visual Studio onder Windows? Zet dan de Asio-directory op een plek waar beide projecten makkelijk bij kunnen. Stel de projecten zo in dat ze ook headers in de Asio-directory vinden.

  Werk je in macOS of Linux? Installeer hem dan met je favoriete package manager, of gebruik eveneens de Asio-directory rechtstreeks.

- Data lezen over het netwerk: denk eraan dat je niet meer leest dan er aan de andere kant geschreven wordt, anders blijft je programma wachten tot er meer komt (*blocking read*).

- Data schrijven over het netwerk: denk eraan dat je niet meer schrijft dan dat er aan de andere kant verwacht wordt, anders blijft je programma wachten tot de andere kant meer data gaat lezen (*blocking write*).

  Dit is de reden dat er in het netwerkprotocol goed op is gelet dat er wordt gecommuniceerd hoeveel regels tekst, of hoeveel bytes er gaan komen, wanneer de hoeveelheid data niet tevoren bekend is.

4 Beoordeling

Als je aan de basiseisen voldoet heb je een 6,0. De plus- en minpunten bepalen het uiteindelijke cijfer waar je op uit komt.

4.1 Basiseisen

1. Zowel de client als de server zijn een C++ command line tool (*console app*).
2. De client implementeert de use cases voor de gebruiker zoals beschreven in §2.1, p. 2, en vertaalt de gebruikerscommando's naar commando's voor de server volgens het protocol, beschreven in §2.2, p. 4.
3. De server implementeert zijn kant van datzelfde protocol.
4. File namen bestaande uit ASCII-tekens[4] gaan goed. Letters met accenten of vreemde talen hoeven niet correct verwerkt te worden.
5. Je past RAII toe waar nodig.
6. Je past de Rule of Five toe waar nodig.

[4] https://en.wikipedia.org/wiki/ASCII

7. Alle pointers worden gemanaged met smart pointers, bij voorkeur `std::unique_ptr`.
8. Je gebruikt exception handling.
9. Je mag alleen de C++ Standard Library, de Asio library, en eventueeel een unit test framework gebruiken. Er zijn geen andere libraries toegestaan.

4.2 Pluspunten

- Als je vreemde tekens (accenten e.d.) in filenamen correct overbrengt: $+1$ punt.
- Als je cross-platform ontwikkelt, dus wanneer client en server op verschillende platforms kunnen draaien: $+1$ punt.
- Als bij cross-platform gebruik ook nog de vreemde tekens correct worden vertaald: $+1$ punt. Dit betekent concreet dat je rekening moet houden met UTF-16 geëncodeerde filenamen op Windows en UTF-8 geëncodeerde filenamen onder macOS en Linux.
- Als je unit tests maakt, die zich als client gedragen, om het netwerkprotocol en gedrag van de server te testen: $+1$ punt.

Als je bovenstaande pluspunten probeert te verwerken, maar het is slechts ten dele gelukt, dan worden de extra punten naar verhouding toegekend.

4.3 Minpunten

- Als je bij de implementatie van de basiseisen zaken niet helemaal compleet of correct gebouwd hebt, worden er punten afgetrokken: -1 punt per gemist item.
- C++-idioom: we veronderstellen dat je C++ correct gebruikt. Er zal puntenaftrek plaatsvinden wanneer dat niet het geval is. Bijvoorbeeld:
  - Parameter passing: *primitives* en `std::shared_ptr` horen by value doorgegeven te worden; *objects* moeten by (const) reference: $-0{,}5$ punt per geval wanneer niet.
  - Const correctness: methods zijn `const` wanneer ze een instantie niet wijzigen, en objecten worden als `const` reference doorgegeven wanneer ze niet gewijzigd worden door die method: $-0{,}5$ punt per geval wanneer niet op die manier.

– We willen geen *naked* `delete` zien, behalve in zelf geschreven RAII classes: −1 punt per *naked* `delete`.

De assessor kan naar eigen inzicht op dit gebied punten aftrekken voor zaken die hier niet expliciet genoemd staan, maar die je redelijkerwijs zou moeten weten.