Opgave project “Shttpd”

Dirk Vermeir

13 juli 2006

Inhoudsopgave

Samenvatting

Dit is de opgave voor het project van het vak “Structuur van Computerprogramma’s II” voor de tweede zittijd van het academiejaar 2005-2006.

Deze tekst is ook beschikbaar in postscript (stru2-2005.ps) en pdf (stru2-2005.pdf) formaat.

1 Achtergrond

1.1 Client-server

Het programma maakt gebruik van de normale internet protocollen en standaarden: TCP/IP, DNS. Zeer summier:

- Een machine op het net heeft een numeriek adres, meestal voorgesteld als een rij van `4` bytes. Bvb. is `134.184.65.2` het internet adres van `tinf2`.
- D.m.v. de DNS service kunnen een of meerdere namen geassocieerd worden met een internet adres. Bvb. is `tinf2.vub.ac.be` de officiële naam van de machine met adres `134.184.65.2`. Zulke namen zijn hiërarchisch gestructureerd (de machine “`tinf2`” behoort tot het domein “`vub.ac.be`” dat zelf weer een deeldomein is van “`ac.be`”, hetwelk een deel is van het “top” domein “`be`”. Men kan bvb. d.m.v. het “**nslookup**” programma nagaan wat het adres is dat hoort bij een naam (of omgekeerd).

- Een machine op het net heeft een aantal poorten waarlangs een verbinding kan gemaakt worden met een poort van een andere machine. Sommige poorten worden, bij conventie, gebruikt voor bepaalde diensten. Bvb. wordt de poort nummer `80` veelal gebruikt door web servers. Wanneer bvb. **mozilla** (of **konqueror**, of **netscape**) vanop de machine `134.184.65.2` (`tinf2`) een verbinding wenst te maken naar de webserver op `elvas.vub.ac.be` (adres `134.18.65.31`), dan wordt er een verbinding geopend tussen een poort, bvb `8026`, op `134.184.65.2` en poort `80` op `134.18.65.31`. D.m.v. het tcp protocol kunnen dan data uitgewisseld worden tussen de toepassingen aan beide kanten van de verbinding.

Voor het netwerk aspect kan gebruik gemaakt worden van het dvnet pakket dat o.a. een klasse `Socket` (voor clients) en `SocketServer` (voor servers) definieert. Omdat `Socket` afgeleid is van `iostream` is het sturen en ontvangen van gegevens via een netwerkverbinding heel makkelijk.

De map demo bevat een volledig uitgewerkt voorbeeld van een “echo server”, dit is een server die elke lijn gestuurd door een client terugstuurt, na de tekst te hebben omgezet naar hoofdletters.

1.2 Multi-threading

Omdat een typische server toepassing verschillende sessies/clients tegelijkertijd moet kunnen behandelen is het aangewezen om deze parallelle activiteiten te implementeren via verschillende “threads”.

Voor deze toepassing volstaat het een thread te beschouwen als een proces dat zijn address space deelt met een ander proces (dit is trouwens min of meer de huidige implementatie van threads in linux). De uitvoering van een thread wordt verder geregeld zoals voor een proces; een multi-threaded programma kan dus verschillende activiteiten gelijktijdig uitvoeren. Omdat threads dezelfde address space delen is er natuurlijk een probleem om gemeenschappelijke data structuren te beschermen tegen gelijktijdige toegang. Een simpele oplossing gebruikt daarvoor zgn. “monitors” , dat zijn stukken code die slechts door één thread tegelijk kunnen uitgevoerd worden.

Voor “threads” en “monitors” zijn klassen beschikbaar in de dvthread bibliotheek (Java ondersteunt gelijkaardige klassen). De documentatie voor `dvthread` bevat ook een programma dat het gebruik van threads en monitors illustreert om concurrente toegang tot een gedeelde buffer te regelen voor verschillende “reader” en “writer” threads.

Meer info over threads en monitors is te vinden in deze nota’s

Indien een programma een onbekend aantal threads dient op te starten is het nuttig om de nieuwe (sinds versie `0.4.5`) constructor functie van `Dv::Thread::Thread` te gebruiken: indien met de constructor de bool parameter `true` wordt meegegeven, dan zorgt de `dvthread` bibliotheek ervoor dat het nieuw aangemaakte `Dv::Thread::Thread` object automatisch wordt deleted wanneer de corresponderende thread klaar is. Indien geen verdere informatie over de exit status van een thread moet opgevraagd worden, dan spaart het gebruik van deze optie een extra “manager” thread klasse uit die anders nodig zou zijn om andere threads te vernietigen, eens ze klaar zijn.

In het onderstaand voorbeeld wordt een klasse `Session` afgeleid van `Dv::Thread::Thread` zodanig dat elk `Session` object op de hierboven beschreven manier vernietigd wordt.

Voorbeeld 1.1 Een thread klasse afgeleid van Dv::Thread::Thread

```
class Session: public Dv::Thread::Thread {
 public:
   /** Een Session kan alleen met deze "factory" functie gemaakt
    * worden. Op die manier wordt vermeden dat per abuus
    * een Session op de stack wordt gemaakt, wat desastreuze
    * gevolgen zou hebben.
    */
   static Session* make(..) { return new Session(..); }
   /** Destructor. */
   virtual ~Session(..) { .. }
   ..
 private:
   /** De echte constructor is private, zie hoger.
    * Het meegeven van de optionele parameter "true" aan de
    * Dv::Thread::Thread constructor zorgt ervoor dat het
    * dvthread pakket zelf een delete zal uitvoeren eens
    * de aangemaakte Thread klaar is.
    */
   Session(..): Dv::Thread::Thread(true) { .. }
   ..
};
```

Een typische server *main loop* ziet er dan uit zoals in de onderstaande code.

```
while (true) {
  .. // Accept a new connection.
  ..
  (Session::make(..))->start(); // Launch new session thread.
  ..
  }
```

2 Eisen

2.1 Functionele eisen

2.1.1 Inleiding

Schrijf[1] een eenvoudig webserver programma shttpd.

Het starten van de server gebeurt d.m.v. het volgende bevel. `shttpd` *`configuration-file`* waarbij de inhoud van het configuratiebestand een aantal parameters vastlegt, zoals getoond in het voorbeeld .

[1]Functionele eisen leggen vast wat het programma moet doen. Niet-functionele eisen leggen vast aan welke andere voorwaarden het programma moet voldoen, bvb. onder welk operating system het moet werken, welke programmeertalen of bibliotheken mogen gebruikt worden, etc.

Voorbeeld 2.1 Voorbeeld shttpd configuratiebestand

```
# sample shttpd configuration file
# documentroot: map waaronder bestanden opgezocht worden
documentroot=/home/anne/htdocs
# portnr: poort waarop de server luistert
portnr=9999
# logfile: where log messages are written to
logfile=/home/anne/shttpd.log
# typesfile
typesfile=/home/anne/shttpd.types
```

Een lijn in het configuratiebestand is dus ofwel een commentaarlijn, die begint met het teken '#', ofwel een definitie van de vorm `naam=waarde` waarbij *`naam`* en *`waarde`* één van de betekenissen uit de onderstaande tabel hebben.

Naam	Waarde
port	nummer van de poort gebruikt door de server
logfile	name van het bestand voor logboodschappen
documentroot	absoluut pad naar een map waaronder de documenten die kunnen aangeboden worden zich bevinden
typesfile	absoluut pad naar een bestand dat een suffix van een bestandsnaam associeert met een "Content-Type".

Tabel 1: Configuratie-elementen

Het opstarten resulteert in een daemon proces dat als webserver functioneert. Het programma kan enkel gestopt worden dmv het "kill" bevel.

2.1.2 Het HTTP protocol

- Het HTTP/1.0 protocol wordt volledig beschreven in het bijgevoegde "RFC[2]" document (nummer 1945).
- Merk op dat shttpd `Full-Request` en `Full-Response` dient te implementeren. Echter, enkel `GET` en `HEAD` requests moeten minimaal geïmplementeerd worden.
- De volgende headers moeten niet ondersteund worden: `Allow`, `Authorization`, `Content-Type` van de client, `Content-Encoding`, `WWW-Authenticate`.
- Voor het bepalen van het type van de inhoud van een bestand mag uitgegaan worden van de suffix van de naam van dat bestand. De mapping van suffixes naar types moet opgeslagen worden in een "types" bestand waar elke lijn van de vorm `suffix type` is. Minstens de types `text/html`, `text/plain`, `text/xml`, `image/gif`, `image/jpeg`, `image/tiff`, `application/pdf`, en `application/msword` moeten ondersteund worden.
- Enkel het aanbieden van "gewone" files en mappen ("directories") moet ondersteund worden, waarbij voor een map `text/html` wordt doorgestuurd die de inhoud van de map zo weergeeft dat de gebruiker kan klikken op een link om een bepaald bestand uit de map te bekomen.

2.1.3 Formaat logbestand

Voor elk verzoek wordt precies één lijn aan de logfile toegevoegd. Zo'n lijn heeft een formaat zoals hierna geïllustreerd.

```
igwe.vub.ac.be - - [12/Feb/2000:13:18:49 +0100] "GET /images/openssl_ics.gif HTTP/1.0" 200 2063
134.184.65.2 - - [01/Nov/2000:15:42:44 +0100] "GET /ssl/kiesoos.cgi?rolnr=58769&stjcode=5L10021 HTTP/1.0"
```

[2] Request For Comments.

Een lijn bestaat dus uit een aantal velden gescheiden door spaties. De betekenis van de velden is als volgt:

1. De naam of het adres van de machine van waaruit het verzoek werd gestuurd. Sommige lijnen zullen hier een domeinadres bevatten.
2. Een niet ingevuld veld (`-`).
3. Een niet ingevuld veld (`-`).
4. Het tijdstip van het verzoek, tussen vierkante haken (`[]`
5. Het verzoek zelf, omringd door accenten (`"`). Dit verzoek bestaat zelf weer uit een aantal delen, gescheiden door spaties:
   (a) Het "werkwoord": `GET`, `PUT`, of `HEAD`
   (b) Een padnaam, evt. gevolgd door formuliergegevens na een vraagteken.
   (c) Een aanduiding van het protocol (typisch `HTTP/1.0` of `HTTP/1.1`)
6. Een code die het resultaat geeft. De mogelijke codes zijn beschreven in RFC1945.
7. Het aantal bytes dat werd teruggestuurd naar de aanvrager.

2.2 Niet-functionele eisen

- Het programma moet volledig en uitsluitend in C++ geschreven worden.
- Het programma moet gebaseerd zijn op een object-georiënteerd ontwerp, volgens de richtlijnen in Hoofdstuk 11 van het handboek. Functies die langer zijn dan 30 lijnen zullen met argwaan worden bekeken.
- Het programma moet correct werken op wilma .
- Enkel de C++ (of C) standaard library en de dvutil, dvnet en dvthread bibliotheken mogen gebruikt worden. In `dvutil` bevinden zich diverse klassen (bvb. `DateFile`, `Directory`, `logstream`) en functies (bvb. `make_daemon`) die vermoedelijk in dit project nuttig kunnen zijn.
- Eigen implementaties van containers zijn niet toegestaan, enkel standaard library containers zoals `map`, `set`, `vector`, mogen gebruikt worden.
- De source (C++) bestanden dienen de suffix "`.cc`", "`.C`" of "`.cpp`" te gebruiken. De "header files" met declaraties dienen de suffix "`.h`" te gebruiken.
- De opgestuurde tarbal moet een `Makefile` en alle nodige bronbestanden bevatten zodat de volgende reeks van bevelen zal resulteren in een correct uitvoerbaar programma "shttpd":

```
make clean
rm -f shttpd *.o
make all
```

- Het programma moet robuust zijn. Dit wil zeggen dat het ook een redelijke actie moet ondernemen indien de input extreem of foutief is.

3 Hints

- Gebruik één thread per verzoek, zie ook het voorbeeld .
- Om het configuratiebestand te parsen is `Dv::Util::Props` nuttig.
- Vergeet niet datastructuren die door verschillende threads kunnen gebruikt worden te beschermen dmv een (klasse afgeleid van) `Dv::Thread::Monitor`.

4 Indienen

De uiterste indiendatum is 27/8/2004, 23:59. Indienen gebeurt door me een e-mail bericht te sturen met een tarbal `shttp.tar` als aanhangsel. Een nette afdruk van een document dat uw ontwerp (hoe en waarom) kort beschrijft moet vóór die datum op het secretariaat van de vakgroep afgegeven worden. Later volgt dan een (individuele) mondeling verdediging. Zie ook de website.

5 Vragen

`<mailto:dvermeir@vub.ac.be>`