TIE-02500 rinnakkaisuus harjoitustyö

Kevät 2015

Dokumentin historia		
5.3.2015	jyke	työohje julkaistu
24.4.2015	jyke	lisätty maininta vegeta testityökalusta

1. Harjoitus

Työssä on tarkoitus harjoitella seuraavia asioita:

- Suunnitella rinnakkaisuutta hyödyntävä ohjelma.
- Toteuttaa toimiva, selkeä ja hyvin dokumentoitu toteutus rinnakkain toimivista säikeistä.
- Harjoitella POSIX pthreads -rajapinnan käyttöä linuxissa.

2. HTTP-palvelin

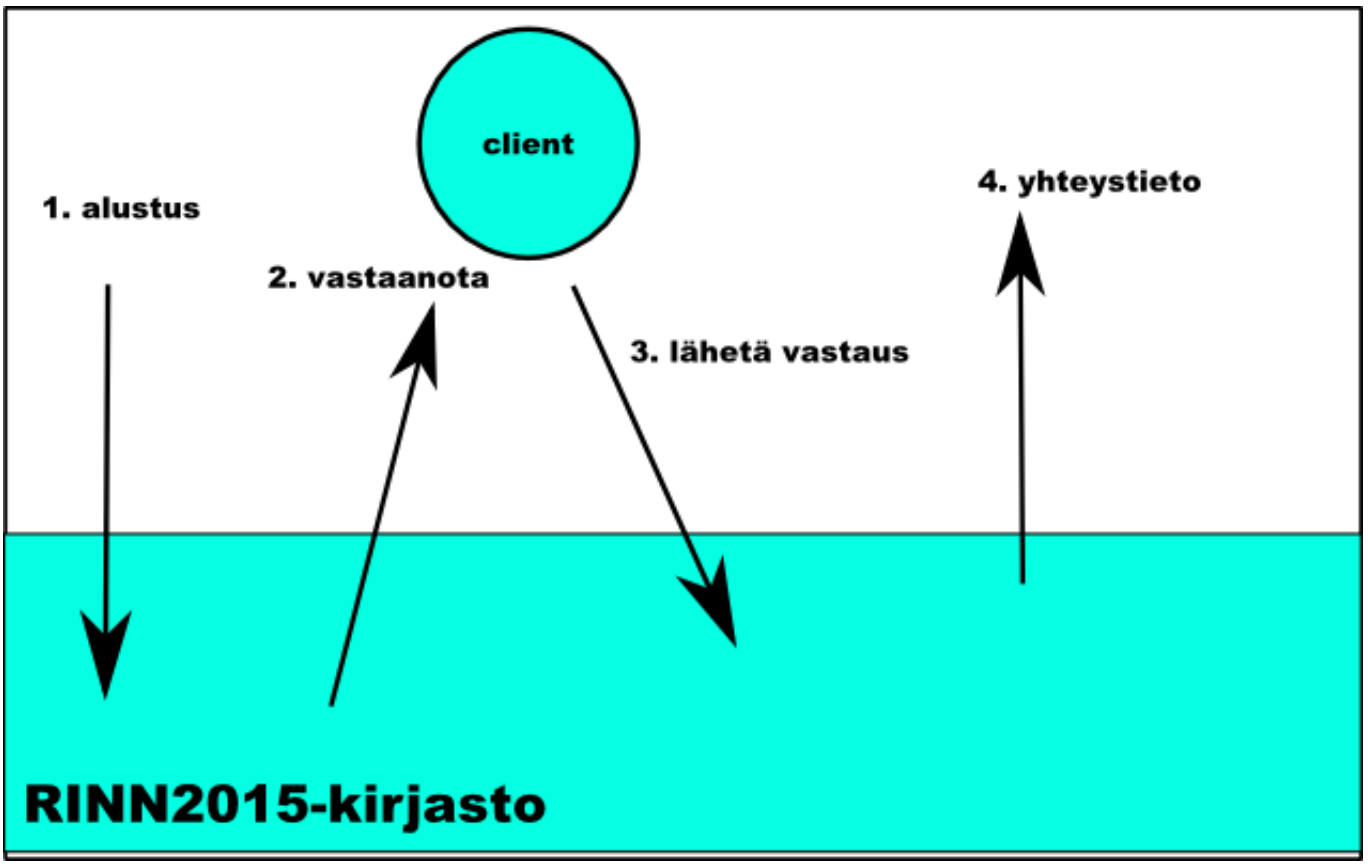
Kurssin puolelta tarjotaan C++-ohjelmointikirjasto, joka sijaitsee Lintulassa:

```
/home/rinn/pub/2015
```

, ja jonka avulla voi toteuttaa yksinkertaisen WWW palvelimen (HTTP protokolla). Kirjasto on tehty sarjalliseen toimintaan ja on sisäiseltä toiminnaltaan melko hidas.

Työssä on tarkoitus suunnitella ja toteuttaa säikeitä hyödyntävä nopeammin toimiva palvelinohjelma, joka silti käyttää samaa valmiina annettua kirjastoa.

Kirjaston toiminta



Kuva 1: kirjaston toiminta

Kuvassa 1 on esitetty toimintajärjestys, jolla http-kirjastoa on tarkoitus käyttäää. Sarjallisesti toimiva http-vastauksia lähettävä koodi kirjastolla toteutettuna olisi (kokonainen mallikoodi löytyy ryhmien svn-tietovarastosta):

```
// 1. alustetaan kirjasto omalla opiskelijanumerolla
rinn2015::init_server( 123456 );

while( true ) {
  // 2. haetaan yksi http-yhteys käsittelyyn
  std::unique_ptr client( rinn2015::get_connection() );

  // 3. rakennetaan vastaustiedot
  client->reply_status = "200";
  std::stringstream vastaus;
  vastaus << "Hello World! ";
  client->reply_data = vastaus.str();

  // 4. lähetetään vastaus kirjaston avulla:
  uint32_t id = rinn2015::send_reply( std::move(client) );

  // 5. haetaan ja tulostetaan yhteyden tiedot:
  std::cout << "Palveltiin yhteys: " << rinn2015::peer_info( id )
            << std::endl;

}
```

Reunaehtoja

- Vain kirjaston otsikkotiedostoissa dokumentoituja asioita saa olettaa kirjaston toiminnasta.

- Jokaisen verkkoyhteyden (www-selaimen) kannalta ohjelma suorittaa seuraavat vaiheet:

  1. Yhteys vastaanotetaan (get_connection)
  2. Yhteys käsitellään (client-olio)
  3. Lähetetään vastaus (send_reply)
  4. Kirjataan **lokitiedostoon** mitä tehtiin ja ja mikä asiakas palveltiin (peer_info). Tämän lokitiedoston nimi on "HTTP.LOKI" ja se luodaan samaan hakemistoon, josta ohjelma käynnistettiin.

  Huomaa, että nämä operaatiot ovat peräkkäisiä **yksittäisen** yhteyden käsittelyn kannalta. Kun yhteyksiä käsitellään useampia rinnakkain, niin mikään ei pakota saman säikeen tekevän kaikkia yhden yhteyden käsittelyn toimenpiteitä...
- Pthread-säikeitä pitää toteutuksessa olla käytössä vähintään kolme.
- Jokaisessa valmiissa työssä tulee olla mukana seuraavat osat:

  1. Suunnitteludokumentti: PDF-dokumentti, josta käy ilmi miten rinnakkaisuutta on käytetty (minkälaiseen rinnakkaisuusasteeseen pyritään ja miten rinnakkaisuuden tuomat ongelmat on ratkaistu)
  2. Ohjelmakoodi: selkeä ja hyvin kommentoitu toteutus C++-ohjelmointikielellä (kirjasto vaatii C++:n, mutta oma koodi saa olla "pelkkää" C:tä)
  3. (KOLMEN HENGEN RYHMÄT) Testausdokumentti: PDF-dokumentti, josta käy ilmi miten ohjelmaa on testattu (testikoodit koodipalautuksessa mukana) ja minkälaisella varmuudella uskotte ohjelmanne toimivan oikein kaikissa tilanteissa rinnakkaisuuden kannalta. (Tämä osa on kolmen hengen ryhmillä pakollinen, pienemmillä ryhmillä tämä osa lasketaan pisteytettäväksi lisäosaksi.)
  4. Työn jakautuminen: PDF-dokumentti, josta käy ilmi miten kukin ryhmän jäsen on osallistunut työn tekemiseen. Tämä pitää olla jokaisen jänen "allekirjoittama" eli kaikkien pitää olla jaosta samaa mieltä. Kaikkien osallistumista koodaamiseen ei vaadita, mutta selkeä osallistuminen johonkin em. osioon pitää olla, jotta työstä voi saada itselleen hyväksymisen/pisteitä.

3. Tulostukset

Ohjelman tulostuksia ei ole mitenkään määritelty, joten ne saa päättää itse. Arvostelussa keskitytään työn rinnakkaisuuden oikeaan toimintaan, jota debug-tulostuksilla ei pystytä varmistamaan...

4. Lisäosia

Mallikoodin mukaisia "200" vastauksia rinnakkain lähettävä palvelin on minimitoteutus, jolla harjoitustyöstä saa arvosanan "hyväksytty". Tentissä laskettavia lisäpisteitä varten tulee työssä olla enemmän toiminnallisuutta, kuten esimerkiksi:

- Palvelin käsittelee path-osiota ja lähettää vastauksen vain niihin "dokumentteihin" jotka palvelin on määritelty tuntemaan (vähintään 10). Muihin vastataan virhetiedolla. Sekä virheet että onnistuneet yhteydet kirjataan lokitiedostoon. "Dokumentti" EI tarvitse

(mutta saa) olla tiedostosta luettu vaan se voidaan tuottaa ohjelman sisällä...

- Käsittelyssä on prioriteetteja. Jos esimerkiksi polun ensimmäinen osa (client->path[0]) on numero väliltä 0-9, niin isommalla numerolla olevat kyselyt käsitellään nopeammin.
- Ohjelma toteuttaa pyynnön tyypin (client->type) perusteella yksinkertaisen CRUD-palvelun , jossa ohjelman muistiin voi tallettaa ja muokata informaatiota (esim. "avain:arvo" merkkijonopareja). Tarvittavia erityyppisiä http-yhteyksiä voi lähettää curl komentorivityökalulla (liitä palautukseesi myös dokumentaatio miten CRUD-palveluasi voi tarkastaja testata).
- Oma lisäosa. Jos haluat toteuttaa jonkin itse keksimäsi toiminnallisuuden, niin hyväksytä se ensin kurssihenkilökunnalla (sähköposti: rinn@cs.tut.fi). Lisäosan tulee tuoda jotain lisämutkikkuutta ohjelman **rinnakkaisuuteen** - tällä kurssilla ei viilata hienoa webbipalvelinta...

5. Testaus

Kurssikirjasto laskee käynnistyvälle http-palvelulle porttinumeron init-rutiinille annetun opiskelijanumeron perusteella, joten joudut ottamaan yhtettä testauksessa tähän porttinumeroon. Huomaa myös, että nimen "rinn.cs.tut.fi" takana on useampia virtuaalikoneita, joista vain yhdelle käynnistyy testipalvelimesi. Tietoturvan takia porttisuodatus estää liikennöintiä koneilta toiselle ja julkiseen internettiin, joten käytännössä joudut testaamaan palvelintasi aina saman koneen sisällä. Eli esimerkkinä:

```
rinn1 % ./testipalvelin &
HTTP server listening in port: 36456
rinn1 % curl -s 'localhost:36456/testi?[1-100]'
```

Curl-komento lähettää sata kappaletta http-pyyntöjä käynnistetylle palvelimelle kun sekä palvelin että curl on ajettu samalla koneella (rinn1.cs.tut.fi).

(Lisäys 24.4.2015) Edellä mainittu curl-työkalu jää odottamaan vastausta yhdestä http-kyselystä ennen kuin se lähettää seuraavan. Paremmin rinnakkaista käsittelyä testaavan työkalun pitäisi lähettää myös kyselyitä rinnakkain. Tätä varten on nyt asennettuna lintulassa vegeta-työkalu jonka saa rinn.cs.tut.fi koneilla ajettua:

```
rinn1 % VEGETA=/usr/local/lang/vegeta/vegeta
rinn1 % echo "GET http://localhost:36456/" | $VEGETA attack -workers 42 -timeout 2s -durati
```

Tuloksista saa halutessaan tehtyä myös kauniin käppyrän:

```
cat result.bin | $VEGETA report -reporter=plot > plot.html
```

Lisätietoja vegetan kotisivuilta

6. Ongelmia?

Selvitä ongelmatilanteet ajoissa joko sähköpostilla: rinn@cs.tut.fi tai poikkea luennoijan työhuoneella: TE211.