Laboration 1: Sockets och Peer-to-Peer Chat

Syftet med den laborationen är att ge dig en introduktion till kommunikation med hjälp av TCP-sockets och även med RPC, Remote Procedure Call. Dessutom ska vi bygga ett enkelt meddelandesystem (chat) men istället för en arkitektur med central server ska vi bygga ett peer-to-peer (p2p) system.

Rekommenderat programspråk är Python 3 och den färdiga exempelkoden är skriven i Python 3. Ungefär samma socket-anrop återfinns på samma sätt i de flesta språk (t ex C eller Java) så egentligen är det fritt fram att prova andra språk också.

Uppgifter och redovisning

Följande uppgifter ska göras:

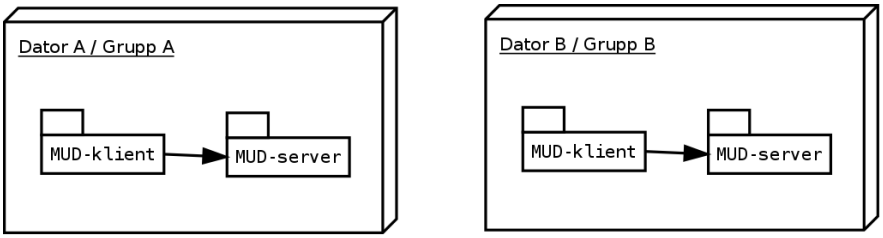
1. Enkelt MUD-spel med sockets. Klient-server på samma maskin.
2. Enkelt MUD-spel på olika maskiner. Kör klient mot annan grupps server.
3. Enkelt MUD-spel med RPC. Klient-server på samma (eller olika) maskiner.
4. Chat-system med p2p-arkitektur. Varje klient skickar meddelanden till alla andras servrar (alla peers).

Arbeta i grupper om två personer. Uppgift 2 och 4 kräver minst två respektive tre gruppers deltagande så för att kunna redovisa och lösa uppgift 4 måste minst tre grupper samarbeta.

Redovisa muntligt på något av labbpassen. För att bli godkänd på labben ska varje person kunna förklara programmen och hur sockets samt RPC fungerar.

Uppgift 1: Enkelt MUD-spel med sockets

Du håller på att bygga ett klassiskt MUD, Multi User Dungeon-spel. Du har precis klarat av singelanvändarversionen som inte innehåller någon nätverkskommunikation. Nästa steg är att dela upp koden i två delar, en klientdel samt en serverdel och fixa till kommunikationen mellan delarna. Slutversionen ska alltså kunna köras på två datorer, en klient och en server, men under första testfasen (uppgift 1) kan man enklast provköra både klient och server på samma maskin.



- Ladda ner exempelkoden: **small_mud.py** från canvas.
- Provkör spelet och se till att förstå programmet.
- Lägg till något extra, t ex, ett extra rum eller byt ut texten för något rum.

Uppgift:

- Dela upp programmet i två delar, en klientdel och en serverdel. Implementera kommunikationen mellan klient och server med hjälp av sockets.
- Serverdelen behöver endast klara av en klient.

Tips

TCP (och UDP) sockets är ett vanligt programmeringsgränsnitt (API) för att skapa kommunikationskanaler mellan två program. Dessa kanaler kan sedan användas för att skicka och ta emot data. I Linux kallas det helt enkelt för “sockets”. I Windows används ofta API-namnet “winsock”.

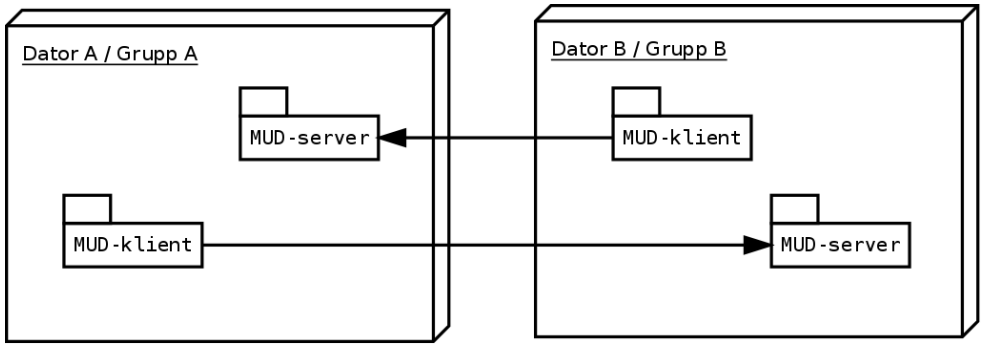
Läs gärna på lite om sockets innan du fortsätter:

- Kursboken kapitel 4, “Simple transient messaging with sockets”
- En bra översikt med exempel i C och Python: http://gnosis.cx/publish/programming/sockets.html. OBS: exempel som använder Python 2. Vi vill köra Python 3 men själva socketanropen är likadana.

- Python’s socket programming howto: https://docs.python.org/3/howto/sockets.html (bra översikt)
- Python’s library documentation: https://docs.python.org/3/library/socket.html (väldigt mycket detaljer, referensmanual)

Uppgift 2: Enkelt MUD-spel med sockets på olika maskiner

Provkör nu uppgift 1 ovan ihop med en annan grupp. Er klient ska kunna prata med en annan grupps server.



- Servern bör skicka ett fint välkomstmeddelande först (med en unik rubrik) så att man säkert ser vilken server man pratar med när man startar klienten.

Uppgift 3: Enkelt MUD-spel med XML-RPC

Uppgift:

- Modifiera din ursprungliga MUD-klient och server så att de använder XML-RPC istället för sockets för kommunikationen.
- Provkör och se till att det fungerar, gärna mot en annan grupp (valfritt)
- Spela in trafiken med hjälp av Wireshark och titta närmare på trafiken som skickas. Hur stora är frågorna och svaren (i bytes)?

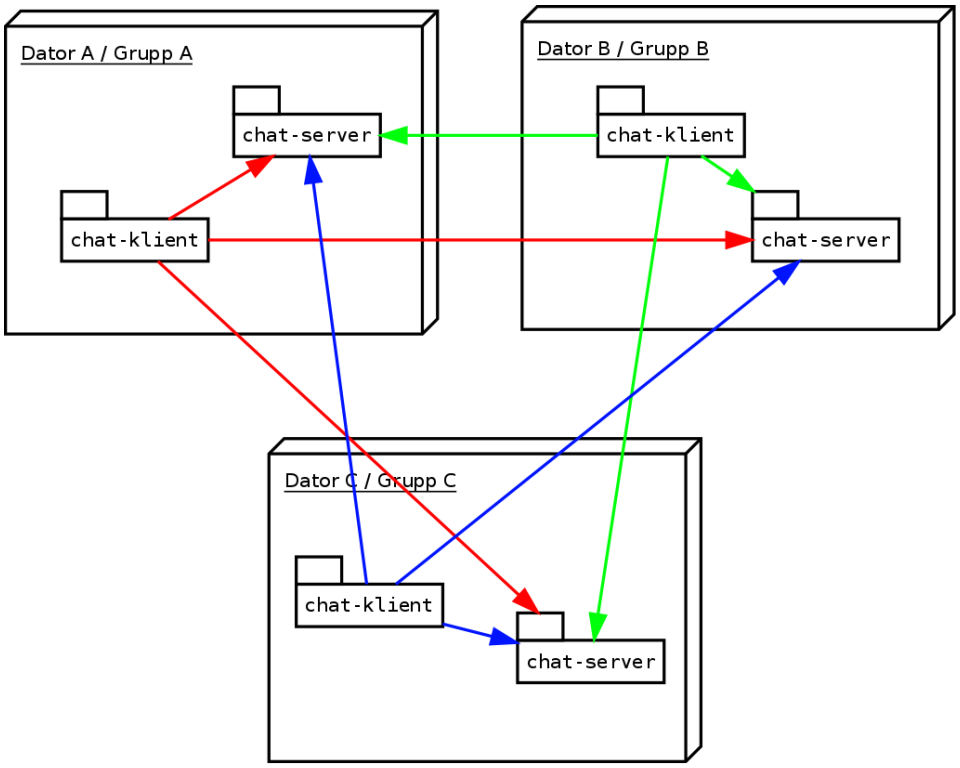
Använd exempelkoden från Pythons hjälpsidor: Standard Library, Internet protocols, simpleXMLRPCserver:
https://docs.python.org/3/library/xmlrpc.server.html

Uppgift 4: Chat-system med p2p-arkitektur

Slutuppgiften är att bygga ett enkelt chat-system enligt en p2p-arkitektur.

Detta innebär:

- Varje grupp ska bygga och köra en server som tar emot och skriver ut chat-meddelanden från alla andra klienter.
- Varje grupp ska bygga en klient som låter användaren “chatta”, dvs skriva in meddelanden (textrader). Dessa ska skickas till alla andra gruppers servrar (alla peers).
- Varje klient ska skicka med ett “id”, t ex ett namn, som kan skrivas ut först på textraderna.



Diskutera mellan grupperna innan vi sätter igång:

- Vad är enklast XML-RPC eller sockets?
- Hur ska gränssnittet/protokollet se ut? API?
- Ska chat-klienten prata med sin egen grupps server?
- Hur hittar vi varandra? Peers?

Diskussionsfrågor vid redovisning

1. Vad gör alla socket-funktionerna?

2. Hur kan en server hantera flera klienter (med sockets) samtidigt?
3. Vad är skillnaden mellan sockets och RPC?
4. Hur mycket trafik skickas vid ett XML-RPC-anrop jämfört med ett socket-anrop?
5. När man chattar, kan meddelanden från olika klienter komma fram i olika ordning till olika peers?