Laborationsrapport

D0036D Nätverksprogrammering

Laboration: 3

Pontus Stenlund

Ponste-5@student.ltu.se

2017-05-30

Innehåll

Problemspecifikation

Vår uppgift var att implementera ett simpelt nätverksbaserat spel (både klient och server) i java, som stöder att flera klienter kan vara uppkopplade mot servern samtidigt. Vi skulle även skriva vårat eget applikationsprotokoll.

Uppgiften var uppdelad i två delar.

Deluppgift A

Vi skulle implementera så att vår klient och server hade fungerande sockets så att de kunde kommunicera med varandra. Vi skulle även implementera ett grafiskt gränssnitt för vår klient.

Deluppgift B

I denna deluppgift skulle vi utöka funktionaliteten på vår klient och server. Från att all kommunikation sker via TCP så skulle vi börja använda multicast för kommunikation från server till klienterna.

Vi skulle även implementera stöd för IPv6. Om det finns en IPv6 adress så ska den användas, annars så koppar klienterna upp sig via IPv4.

Sedan skulle vi implementera så att startade spelservrar på nätverket skulle dynamiskt upptäckas.

Användarhandledning

För att starta servern via Eclipse så öppnar man först projektet Java_Server och sedan högerklickar man på Server.java och sedan Run as -> Java Application.

För att starta klienten så upprepar man samma sak som ovan med Java_Client och högerklickar på Application->Run as -> Java Application.'

När man startar klienten så kommer det upp ett fönster med två knappar, find servers och connect to server. Till höger om knapparna finns det en drop-down meny där alla servrar som man hittar kommer att hamna.

När man har tryckt på find servers och valt en från listan så trycker man på connect to server så kommer spelet att starta.

För kontrollera spelaren så används dessa tangenter:

- W - För att förflytta spelaren uppåt.
- A - För att förflytta spelaren åt vänster.
- S - För att för förflytta spelaren nedåt.
- D – För att förflytta spelaren åt höger.
- Escape – För att avsluta spelet.

Systembeskrivning

Server

Servern är uppbyggd av dessa klasser och funktioner på följande sätt.

IPSender

IPSender(), är klassens kontruktor. Här skapas en ny tråd och startas.

Run(), lyssnar hela tiden efter IP förfrågningar på multicast adressen 239.255.255.250 och port 1900. När en förfrågan kommer in så svarar servern med sitt IP och sin port.

Player

Denna klass kommer att motsvara en spelare och all dess data som behövs för att spela. Så varje gång en spelare ansluter till servern så skapas en ny instans av klassen.

Dessa funktioner finns i Player.

Player(int clientID, Socket socket, DatagramSocket datagramSocket, Server server), Denna konstruktor tilldelar spelaren ett id, socketen som klienten anslöt på, samma datagram socket sin server, och en referens till servern. Här skapas och startar även en ny tråd för varje spelare.

Run(), i denna funktion så sker kommunikation med klienten. När en instans av Player har skapats så kommer ett meddelande att skickas till alla klienter som är anslutna till multicast adressen 229.15.15.15, port 7778 om den nya spelarens position.

Sedan skickas också en uppdatering via multicast om alla anslutna spelares id och position så att den nya spelaren får information om andra spelares position så att denna kan rita ut dem.

När denna tråd får ett meddelande från klienten så kommer den att kolla vilket event som spelaren skickade. Om det är ett ”Move” event så kommer den att kolla nästa del av meddelandet vilket är vilket håll som spelaren vill röra sig åt. Och när det är kollat så kommer den att sätta spelarens delta X eller delta Y till +-10. Sedan kommer server.MovePlayers() anropas.

Om det är ett ”Stop” event så kommer delta X och delta Y sättas till 0.

Om det är ett ”Leave” event så kommer det att skickas ett meddelande till alla klienter att en spelare har lämnat spelet och sedan kommer RemovePlayer() anropas.

Server

Det är i denna klass som finns dessa funktioner.

Server(), skapar en server socket och en datagram socket.

Run(), skapar en instans av IPSender och lägger sig sedan och lyssnar efter anslutningar till servern. När en ny klient ansluter sig till servern så skapas en ny instans av Player. Sedan lägger sig funktionen och lyssnar på nya anslutningar igen.

Move(), Går igenom alla anslutna spelare och kollar ifall dess förflyttning är godkänd. Om den blir godkänd så förändras spelarens position.

RemovePlayer(int id), när en spelare lämnar spelet så anropas denna funktion och tar bort spelaren från listan.

Client

Klienten innehåller dessa klasser och funktioner.

Application

Application(string ip, int port), konstruktor för klassen. Skapar ramen för spel fönstret och spelbrädet som är en instans av Board(). Lägger även till window listener så att när fönstret stängs ner så anropas board.OnExit().

Main(String[] args), skapar en instans av SelectServer och en instans av Application när serverns IP och port har sats av SelectServer.

Board

Board(String ip, int port), skapar spelbrädet och skapar en instans av Client.

OnExit(), anropar client.Exit().

PaintComponent(Graphics g), ritar ut komponenter på spelbrädet.

ActionPerformed(ActionEvent e), lyssnar efter ett event när man trycker på någon av de två knapparna när man letar och väljer server.

TAdapter

Är en privat klass i Board.

KeyReleased(KeyEvent e), lyssnar på när en tangent har släppts upp. Skickar eventet e vidare till Client's keyReleased()

KeyPressed(KeyEvent e), lyssnar på när en tangent har blivit intryckt. Skickar eventet e vidare till Client's keyPressed()

Client

Client(String ip, int s_port), konstruktor för Client. Anropar först CheckForIPv6 för att kolla ifall det finns en IPv6 adress till servern. Sedan ansluter den sig till servern, skapar en DataOutPutStream och skickar ett Connect meddelande till servern. Om den får tillbaka ett NewPlayer meddelande från servern så skapas en instans av Player med id från servern och sätter sedan positionen på spelaren med resten av informationen.

Sedan skapas en instans av ClientListener som hädanefter kommer att lyssna på inkommande data från servern.

KeyPressed(KeyEvent e), hanterar vilken tangent som har blivit nedtryckt. Beroende på vilken knapp det är så skickas ett meddelande till servern som innehåller spelarens id, vilken händelse och sedan vilket håll spelaren vill röra sig åt.

KeyReleased(KeyEvent e), hanterar vilken tangent som har blivit släppt. Skickar sedan ett meddelande till servern med händelsen "Stop", och sedan vilket håll som spelaren slutat röra sig åt.

Exit(), skickar ett meddelande till servern om att den kommer att lämna spelet och anropar sedan socket.close().

CheckForIPv6(String ip), kollar ifall det finns en IPv6 till adressen som skickas in. Returnerar IPv6 adressen eller IPv4 adressen.

DoDrawing(Graphics g, Board b), ritar ut spelarens ikon på spelbrädet.

ClientListener

Är en privat subklass till Client.

Run(), lyssnar hela tiden på data från servern och beroende på vilket meddelande som servern har skickat så skapas en ny spelare, ändras den någon spelares position eller så tas någon spelare bort från spelbrädet.

Player

Klassen representerar alla spelare på spelbrädet.

Player(String playerIcon, int playerID), konstruktor för Player. Hämtar ikonen från sökvägen playerIcon och sätter spelarens id till playerID.

GetX(), returnerar spelarens x position

GetY(), returnerar spelarens y position.

GetPlayerIcon(), returnerar spelarens ikon.

SetPosition(int xPos, int yPos), sätter spelarens position till den angivna och byter spelarens ikon beroende på vad delta positionen är.

SetEnemyPosition(int xPos, int yPos), sätter fiendespelarens position till den angivna och byter spelarens ikon beroende på vad delta positionen är.

SelectServer

SelectServer(), konstruktor för SelectServer. Skapar fönstret för server selection med knappar och lista över servrar, skapar en ny multicast socket och ansluter den till gruppen. Skapar även en instans av ServerListener.

ActionPerformed(ActionEvent e), hanterar vad som ska hända beroende på vilken knapp på fönstret som har blivit nedtryckt.

Om ”Find Server” knappen har blivit nedtryckt så skickar den ett *”SERVICE QUERY JavaGameServer”* till multicast gruppen.

Om ”Connect to Server” knappen har blivit nedtryckt så hämtar den server adressen och port från den server som har blivit vald i listan och sätter sedan sina variabler ip och port.

ServerListerner

ServerListener(), skapar en ny tråd och startar den.

Run(), lyssnar på svar från *”SERVICE QUERY JavaGameServer”* meddelandet som man har skickat och servrarna som svarar lagras i en lista med dess namn, IP adress och port.

Applikationsprotokoll

Klienten skickar *”SERVICE QUERY JavaGameServer”* meddelande via multicast till adressen 239.255.255.250 och port 1900 som ett DatagramPacket.

När servern får detta paket så kommer den att konvertera det till en sträng. Sedan så delar den upp strängen vid mellanslag, kolla att alla element i arrayen stämmer överens med ”SERVICE QUERY JavaGameServer”. Om det stämmer så bygger servern upp en byte array med strängen ”SERVICE REPLY JavaGameServer BattleOfCanons 130.240.54.16 7777” och skickar det till adressen 239.255.255.250, port 1900.

Hädanefter så kommer klienten att skicka sina meddelanden via DataInputStream och ta emot meddelanden från servern som byte arrays via DatagramPacket.
Servern tar emot meddelanden från klienten via DataInputStream och skicka meddelanden som byte arrays via DatagramPacket.

När klienten kopplar upp sig till servern så skickar klienten följande meddelande *”Connect:”* och när servern får det meddelandet så vet den att det är en ny klient och skickar följande sträng via DataOutputStream, ”NewPlayer:ID:ID:10:10”. ID ska bytas ut mot en siffra. ID ska ökas med ett vid varje ny anslutning så att nästa klient inte får samma ID. ”NewPlayer” säger åt klienten att anslutningen har blivit accepterad, att den har fått ID t.ex. 3 och att startpositionen är 10 i x och y led.

Efter det kommer den att skicka en byte array för varje ansluten spelare till den nya klienten så att denne kan rita ut de befintliga spelarna på deras respektive position.

Alla meddelanden som klienten skickar till servern är strängar som sedan konverteras till byte arrays och skickas via DatagramPacket. Strängarna byggs upp efter denna mall.
”Klientens ID:Event:Riktning:” där klientens id är en integer tilldelat av servern.
Servern kommer att dela strängen vid ”:”, så vi får en array av strängar. Sedan jämför servern strängen på andra platsen i arrayen för att att se vilket event det är.

Event är vad spelaren begär om att få göra. Eventen är kopplade till knapptryckningarna.
”Move” så begär klienten att få förflytta sig. Detta event skickas när klienten registrerar en knapptryckning.
”Stop” så begär klienten att få stanna. Detta event skickas när klienten registrerar att en knapptryckning avslutas.
”Leave” så säger klienten till servern att den kommer att avsluta anslutningen. Detta event skickas när klienten registrerar att användaren har tryckt på escape eller kryssar ner fönstret.
”Riktning”, så säger klienten till vilken riktning som den begär att förflytta sig åt.
Riktningarna är Up, Down, Left och Right.

Här följer ett par exempel på ett meddelande som skickas vid när en spelare vill förflytta sig.
”5:Move:Right:”, där 5 är klientens id och spelaren vill förflytta sig åt höger.
”2:Move:Down:”, där klientens id är 2 och spelaren vill förflytta sig nedåt.

När servern får ett meddelande av klienten så kommer den att dela strängen vid ” : ” . Om vi tar exemplet ”2:Move:Down:” så får vi en array med tre strängar. [”2”, ”Move”, ”Down”] så då kollar servern först ifall första elementet är ”Connect”.
Om första elementet inte är ”Connect” så går den vidare och kollar vad det andra elementet är. I detta fall så är det ”Move”. Då konverterar servern första elementet till en integer och jämför med trådens id. Om trådens id och meddelandets id är lika så kollar servern det tredje elemetet i arrayen för att se vilken riktning spelaren vill förflytta sig. I det här fallet så vill spelaren förflytta sig åt nedåt. Då sätter servern trådens deltaY till 10.
Sedan skickas ett multicast meddelande till alla klienter om spelarens nya position.

När servern svarar klienterna så skickas meddelanden via multicast adressen 229.15.15.15, port 7778.
Mallen för svar från servern är följande.

”Klientens id: X position: Y position:”, där Klientens id är det som har blivit tilldelat av servern, X och Y position är klientens position i x och y led i spelvärlden.
Detta meddelande kan se ut på följande vis ”5:110:70:”
Med undantag då en klient kopplar upp sig första gången för då skickas följande meddelande bara till den klienten: ”NewPlayer:Klientens id: X position: Y position:”

Sedan skickas ett meddelande till alla klienter, t.ex. ”3:120:80”. Detta innebär att klienten måste kolla genom sin lista av anslutna spelare för att uppdatera positionen. Men om klientens id inte finns med i listan så skapas en ny spelare med det id som fanns i meddelandet.

Klienterna gör då som servern och delar strängen i meddelandet och jämför

Lösningens begränsningar

En begränsning är att om spelaren inte avslutar spelet korrekt, dvs genom att trycka på escape eller kryssa ner spelrutan så kommer inte servern att veta att spelaren har avslutat spelet och därför inte ta bort spelaren från listan. Så då kommer den att skicka ut den spelarens position till alla klienter som ansluter hädanefter. Detta kan lösas genom att starta om servern.

En annan begränsning är att i vissa fall så anropar jag inte Thread.join() vilket kan medföra att tråden alltid är igång tills programmet har stängts ner. Detta medför att den tar upp onödiga resurser.