Föreläsning 12

- Client/Server - Serversidan
- TCP/IP (och UDP)

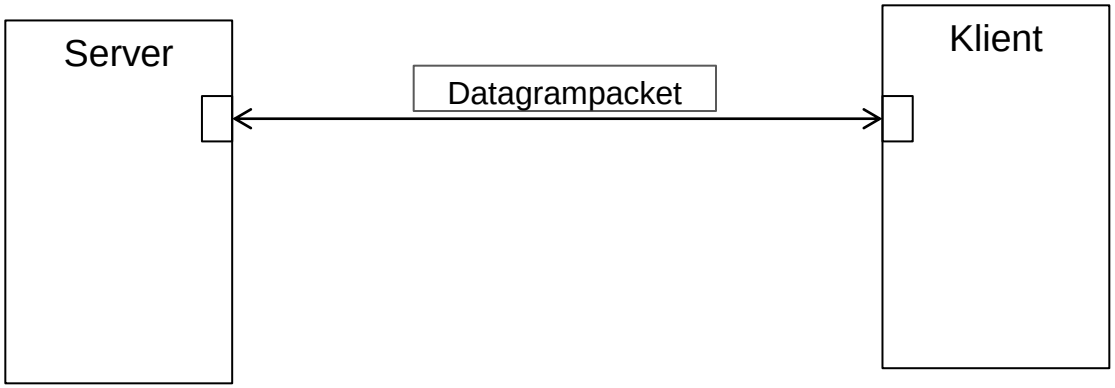
JNP: s 283 - 309

Client/Server, UDP

Klassen ***DatagramSocket*** används på både serversidan och klientsidan.

Servern lyssnar efter *request* på en speciell port och ger *response* på klientens port.

All kommunikation förpackas i ***DatagramPacket***-objekt.



Client/Server, TCP

I java används klassen ***ServerSocket*** för anslutning av klienter.

Vid anslutning av en klient skapas en Socket för kommunikation med klienten.

Kommunikationen sker via en ***InputStream*** och en ***OutputStream*** i vardera enhet

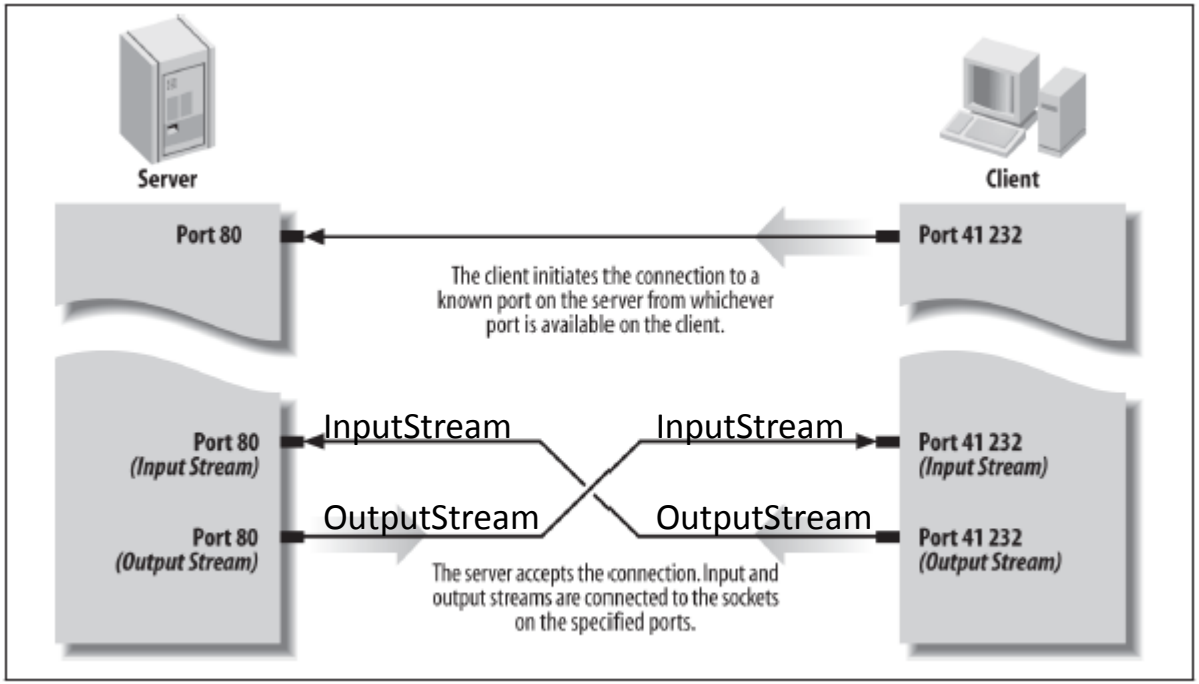


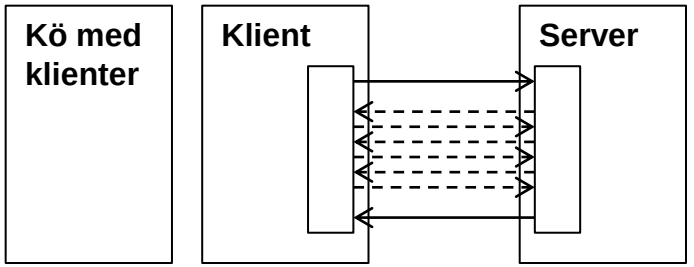
Figure 1-5. A client/server connection

Olika typer av servrar

Iterativ server

En klient i taget hanteras av servern

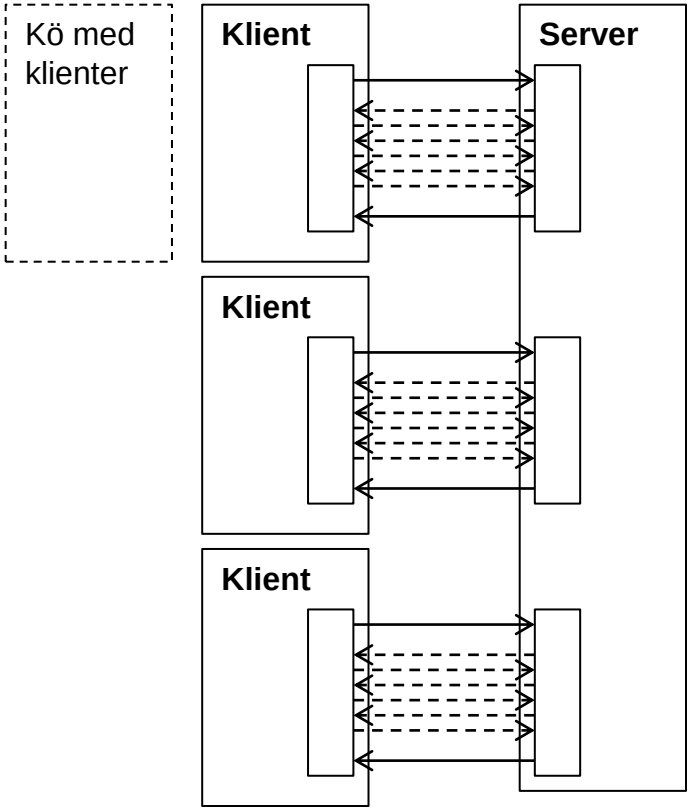
- Då en request exekveras snabbt
- Enklare att konstruera servern



Flertrådad server

Flera klienter hanteras av servern samtidigt

- Då en request tar tid att exekvera
- Mer komplex konstruktion av servern



Iterativ server

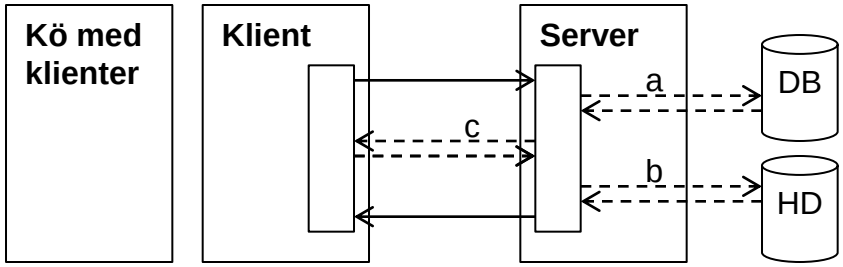
En iterativ server hanterar en klient i taget.

Fördelar

- Effektiv då overhead med trådhantering försvinner
- Enkel att skriva

Nackdelar

- Om en request tar tid att exekvera så får klienter i kö vänta på sin service. Om en operation medför att servertråden får vänta/blockeras viss tid så drabbas klienterna i kö. Detta kan ske t.ex. vi kommunikation med en databas (a), en hårddisk (b) eller klienten (c).
- Om servertråden inte får en resurs den väntar på kan servern helt blockeras eller bli betydligt långsammare.



Iterativ server, UDP

En iterativ server som använder UDP är enkel att skriva.

1. Skapa DatagramSocket och lite till.
2. Lyssna efter request
3. Hantera request
4. Skicka response
5. Tillbaka till steg 2

```
public Server(port) throws SocketException {
    socket = new DatagramSocket(port);
    thread.start();
}

public void run() {
    DatagramPacket packet;
    String response, ticket;
    byte[] buffer = new byte[256];
    byte[] outData;
    while(true) {
        try {
            packet = new DatagramPacket(buffer,buffer.length);
            socket.receive(packet);

            ticket = new String(packet.getData(),0,packet.getLength());
            response = getResponse(ticket);

            outData = response.getBytes();
            packet = new DatagramPacket(outData,outData.length,
packet.getAddress(),packet.getPort());
            socket.send(packet);
        } catch(Exception e) {
            System.err.println(e);
        }
    }
}
```

I LotteryServerA är getResponse långsam (0-4 sek).

LotteryServerA.java

LotteryClientA.java

Iterativ server, TCP

En iterativ server som använder UDP är enkel att skriva.

1. Skapa ServerSocket och lite till.
2. Lyssna efter anslutande klient
3. Skapa strömmar
4. Läs request
5. Hantera request
6. Skicka response
7. Stäng anslutning
8. Tillbaka till steg 2

```
public Server(int port) throws IOException {
    serverSocket = new ServerSocket(port);
    thread.start();
}

public void run() {
    String ticket,response;
    System.out.println("Server running");
    while(true) {
        try (Socket socket = serverSocket.accept();
             DataInputStream dis = new DataInputStream(
socket.getInputStream());
             DataOutputStream dos = new DataOutputStream(
socket.getOutputStream())) {
            ticket = dis.readUTF();

            response = getResponse(ticket);

            dos.writeUTF(response);
            dos.flush();
        } catch(IOException e) {
            System.err.println(e);
        }
    }
}
```

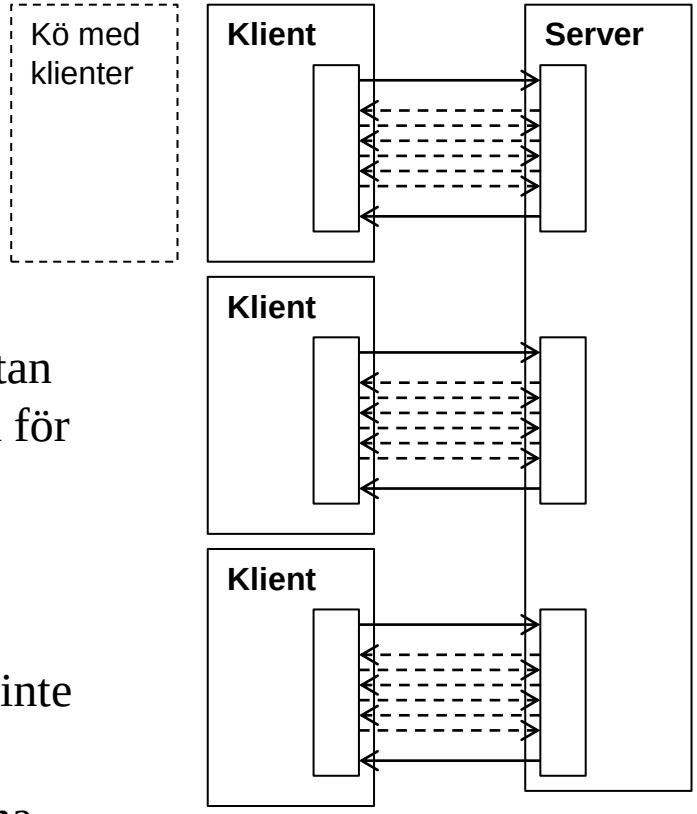
I LotteryServerB är getResponse långsam (0-4 sek).

LotteryServerB.java

LotteryClientB.java

Flertrådad server

En flertrådad server hanterar ett antal klienter samtidigt.



Fördelar

- Om tråden, vilken hanterar request från en klient, tillfälligt avbryts i väntan på en resurs påverkas inte prestandan för andra klienter

Nackdelar

- Vid exekvering på en processor ökar inte den totala prestandan i servern.
- Mer komplex att skriva. Gemensamma resurser måste synkroniseras.

Flertrådad server

En tråd per klient

- Server startar en ny tråd för varje ny klient.
- Om klienterna är många samtidigt kan systemet tyngas ner.
- En del resurser går åt till trådhanteringen

Tråd-pool

- Servern startar ett visst antal trådar vilka servar klienter som kopplar upp
- Om klienterna är många så hinner kanske inte trådarna med. Det innebär att klienten får vänta på sin tur eller kanske t.o.m. inte får någon anslutning

Flertrådad server, UDP

En design för en flertrådad server är att låta varje klient behandlas av en tråd som startas.

1. Skapa DatagramSocket och lite till.
2. Lyssna efter request
3. Skapa tråd som hanterar request
4. Tillbaka till steg 2

```
public Server(int port) throws SocketException {
    socket = new DatagramSocket(port);
    thread.start();
}

public void run() {
    DatagramPacket packet;
    byte[] buffer = new byte[256];
    String ticket;
    System.out.println("Server running");
    while(true) {
        try {
            packet = new DatagramPacket(buffer,buffer.length);
            socket.receive(packet);

            ticket = new String(packet.getData(),0,packet.getLength());
            new ClientHandler(packet.getAddress(), packet.getPort(),
ticket);

        } catch(Exception e) {
            System.err.println(e);
        }
    }
}
```

LotteryServerC.java

LotteryClientA.java

Flertrådad server, UDP

En design för en flertrådad server är att låta varje klient behandlas av en tråd som startas.

1. Tråd hanterar request, skickar response och avslutas

```
private class ClientHandler extends Thread {
        private InetAddress address;
        private int port;
        private String ticket;

        public ClientHandler(InetAddress address, int port, String ticket) {
            this.address = address;
            this.port = port;
            this.ticket = ticket;
            start();
        }

        public void run() {
            try {
                String response = getResponse(ticket);
                byte[] outData = response.getBytes();
                DatagramPacket packet = new DatagramPacket(outData,
outData.length, address, port);
                socket.send(packet);
            } catch (IOException e) {}
        }
    }
```

I LotteryServerC är getResponse långsam (0-4 sek).

LotteryServerC.java

LotteryClientA.java

Flertrådad server, TCP

En design för en flertrådad server är att låta varje klient behandlas av en tråd som startas.

1. Skapa ServerSocket och lite till.
2. Lyssna efter anslutande klient
3. Skapa tråd som hanterar request
4. Tillbaka till steg 2

```
public LotteryServerD(int port) throws IOException {
    serverSocket = new ServerSocket(port);
    server.start();
}

public void run() {
    System.out.println("Server running");
    while(true) {
        try  {
            Socket socket = serverSocket.accept();
            new ClientHandler(socket).start();
        } catch(IOException e) {
            System.err.println(e);
        }
    }
}
```

LotteryServerD.java

LotteryClientB.java

Flertrådad server, TCP

En design för en flertrådad server är att låta varje klient behandlas av en tråd som startas.

1. Tråd hanterar request, skickar response och avslutas

```
private class ClientHandler extends Thread {
      private Socket socket;

      public ClientHandler(Socket socket) {
          this.socket = socket;
      }

      public void run() {
          String ticket,response;
          try (DataOutputStream dos = new DataOutputStream(
socket.getOutputStream());
               DataInputStream dis = new DataInputStream(
socket.getInputStream())    ) {
              ticket = dis.readUTF();
              response = getResponse(ticket);
              dos.writeUTF(response);
              dos.flush();
          } catch(IOException e) {}
          try {
              socket.close();
          } catch(Exception e) {}
      }
  }
```

I LotteryServerD är getResponse långsam (0-4 sek)

LotteryServerD.java

LotteryClientB.java

Flera request i samma uppkoppling

En server kan tillåta att en klient behåller uppkopplingen under flera request. Det är klienthanteraren som förändras.

I while-loopen läser och skriver tråden upprepade gånger.

Då klienten stänger sin socket kastas IOException (dis.readUTF, dos.writeUTF eller dos.flush). Undantaget fångas efter while-loopen varvid klienttråden avslutas.

```
private class ClientHandler extends Thread {
    private Socket socket;

    public ClientHandler(Socket socket) {
        this.socket = socket;
    }

    public void run() {
        String ticket,response;
        System.out.println("Klient uppkopplad");
        try (DataOutputStream dos = new DataOutputStream(
socket.getOutputStream());
             DataInputStream dis = new DataInputStream(
socket.getInputStream())    ) {
            while(true) {
                ticket = dis.readUTF();
                response = getResponse(ticket);
                dos.writeUTF(response);
                dos.flush();
            }
        } catch(IOException e) {}
        try {
            socket.close();
        } catch(Exception e) {}
        System.out.println("Klient nerkopplad");
    }
}
```

- LotteryServerE.java
- LotteryClientE.java
- UI.java

Trådpool hanterar klienter

En trådpool är ett antal trådar vilka utför uppgifter vilka lagras i en buffert.

Om varje ansluten klient placeras i bufferten kan trådarna successivt beta av klienterna.

RunOnThread (F7) går ganska enkelt att göra om till en trådpool vilken exekverar Runnable-implementeringar.

Det behövs en objektsamling för att hålla reda på trådarna:

```
private LinkedList<Worker> workers;
```

Trådarna måste instantieras och startas

```
workers = new LinkedList<Worker>();
for(int i=0; i<n; i++) {
    worker = new Worker();
    worker.start();
    workers.add(worker);
}
```

Trådarna bör kunna avslutas – se stop-metoden

LotteryServerF.java

LotteryClientBDF.

Trådpool hanterar klienter

Skillnaderna mot LotteryServerD är ganska små:

- En trådpool instansieras och startas i konstruktorn.

```
pool = new RunOnThreadN(nbrOfThreads);
serverSocket = new ServerSocket(port);
pool.start();
server.start();
```

- Den inre klassen *ClientHandler* är ingen tråd utan implementerar *Runnable*. En tråd i trådpoolen exekverar run-metoden.

```
private class ClientHandler implements Runnable {

}
```

- Efter att en klient anslutit placeras klienthanteraren i trådpoolens buffert för att exekveras när tid finns.

```
Socket socket = serverSocket.accept();
pool.execute(new ClientHandler(socket));
```

LotteryServerF.java

LotteryClientBDF.

Server notifierar klienter

En server kan notifiera en klient vid en speciell tidpunkt. För detta krävs:

UDP

Att servern lagrat klientinfo (InetAddress/ip-adress + port) i en objektsamling.

Vid notifieringen skickas DatagramPacket till klienten.

TCP

Att klienten har en ServerSocket vilken ligger och lyssnar på en speciell port.

Servern måste lagra klientinfo(InetAddress/ip-adress + port på vilken klienten lyssnar efter uppkoppling) i en objektsamling

Vid notifiering kopplar servern upp mot klienten och överför data.

AlarmServer.java

AlarmClient.java