Socket Programlama

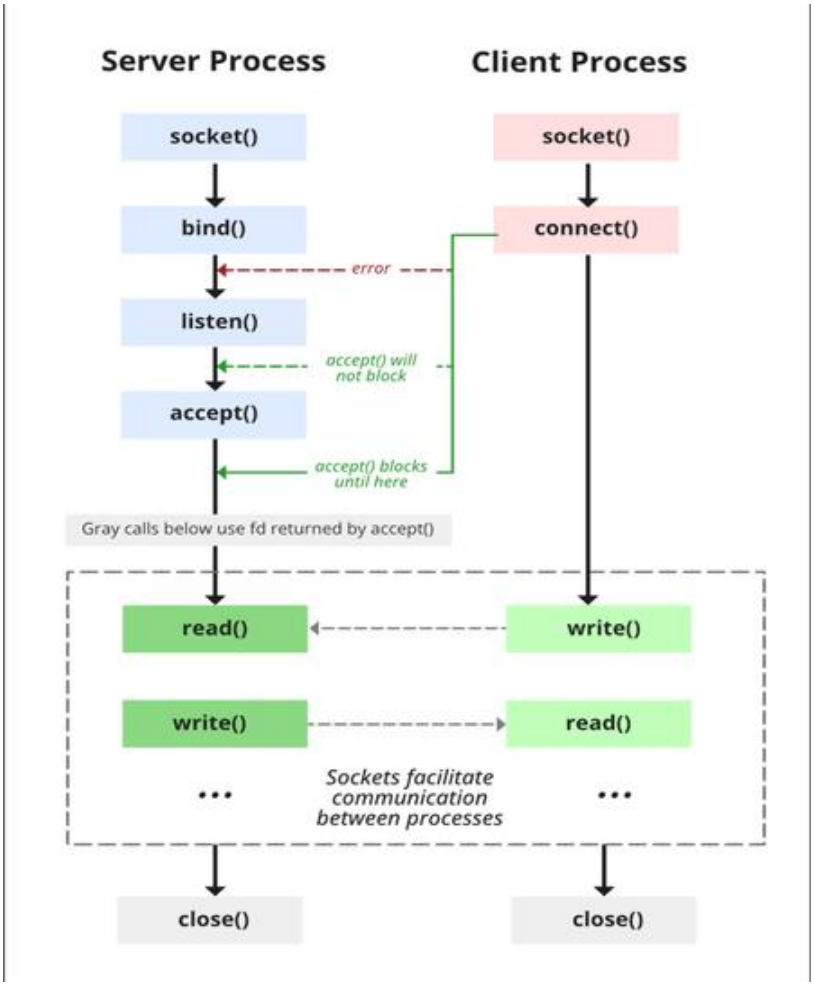
Soket programlama nedir?

Soket , bir ağda adlandırabileceğiniz ve adresleyebileceğiniz bir iletişim bağlantı noktasıdır. Soket programlama, uzak ve yerel süreçler arasında iletişim bağlantıları kurmak için soket API'lerinin nasıl kullanılacağını gösterir.

Bir soket kullanan işlemler, aynı sistemde veya farklı ağlardaki farklı sistemlerde bulunabilir. Soketler hem bağımsız hem de ağ uygulamaları için kullanışlıdır. Soketler, aynı makinedeki veya bir ağdaki işlemler arasında bilgi alışverişinde bulunmanıza, işi en verimli makineye dağıtmanıza ve merkezi verilere kolayca erişmenize olanak tanır. Soket uygulama programı arabirimleri (API'ler), TCP/IP için ağ standardıdır. Çok çeşitli işletim sistemleri soket API'lerini destekler.

Soketler genelde sunucu ve istemci etkileşimi için kullanılır. İstemciler sunucuya bağlanır, bilgi alışverişi yapılır daha sonra da sunucudan ayrılırlar. Sunucu istemciden gelen istekleri bekler. Bunu için sunucuya ait adres ve yine sunucuya atanmış port numarası üzerinden bağlantı gerçekleşir. Bağlantı sonrası veri alışverişi gerçekleşebilir.

Tipik bir soket tasarımını inceleyelim.



Sunucu ve istemci modeli için durum diyagramı

1- Soket, iletişim için bir nokta oluşturur.
2- Soketin istemciler tarafından bulunabilmesi için sokete benzersiz bir ad bağlanır.
3- Dinleme, sunucuya bağlanmaya çalışan istemcileri yakalayıp kabul kısmına gönderir.
4- İstemci sunucuya bağlanmak için bu kısımdan geçmelidir.
5- İstemcinin bağlantı isteğini kabul etmek için kullanılır.
6- Sunucu ve istemci arasında bağlantı kurulduğunda aralarındaki veri aktarımını temsil eder.
7- Sunucu veya istemci işlemi durdurmak istediğinde kullanılır.

Soket Bağlantısı için gereken Protokoller

Soketin adres ailesi tanımlanması gerekir. Belirlenen aileye uygun soket tipi belirtilir. Soket tipi iletişim biçimini belirtir. Soket protokolü ise iletişim protokolünü temsil eder. (N/A, TCP, UDP IP, ICMP)

Address family	Socket type	Socket protocol
AF_UNIX	SOCK_STREAM	N/A
	SOCK_DGRAM	N/A
AF_INET	SOCK_STREAM	TCP
	SOCK_DGRAM	UDP
	SOCK_RAW	IP, ICMP
AF_INET6	SOCK_STREAM	TCP
	SOCK_DGRAM	UDP
	SOCK_RAW	IP6, ICMP6
AF_UNIX_CCSID	SOCK_STREAM	N/A
	SOCK_DGRAM	N/A

1. Server oluşturma

Sunucu uygulamasını oluşturmak için ServerSocket sınıfının örneğini oluşturmamız gerekir. Burada client ve server arasındaki iletişim için 5000 port numarasını kullanıyoruz. Başka herhangi bir bağlantı noktası numarası da seçebilirsiniz. accept() yöntemi istemciyi bekler. İstemciler verilen bağlantı noktası numarasıyla bağlanırsa, bir Socket örneği döndürür.

```
ServerSocket serverSocket=new ServerSocket(5000);
Server server=new Server(serverSocket);
```

Client Oluşturma:

İstemci uygulamasını oluşturmak için Socket sınıfının örneğini oluşturmamız gerekir. Burada, Sunucunun IP adresini veya ana bilgisayar adını ve bir port numarasını geçmemiz gerekiyor. Burada sunucumuz aynı sistem üzerinde çalıştığı için "localhost" kullanıyoruz.

```
Socket socket = new Socket("localhost", 5000);
```

Server'a gelen isteklerin kabulü ve Thread Yardımı ile iş parçacığı oluşturulması

```
Socket socket=serverSocket.accept();
Thread thread=new Thread(clientHandler);
thread.start();
```

Thread sınıfı kullanılmadığı zaman serverımız sadece bir client bağlantısını kabul edip diğer gelen istekleri blockluyor bundan dolayı Thread sınıfı kullanarak iş parçacığı oluşturup birden fazla clientin serverSocket.accept fonksiyonu ile serverımaza bağlanmamızı sağlıyoruz.

ClientHandler ile client'lerin tutulması

```
ClientHandler clientHandler=new ClientHandler(socket);
```

ClientHandler her gelen yeni isteğin clienthandler'ın constructor'ın socket parametresi ile yeni bir client olarak tutulmasını sağlar.

Okuma-Yazma Kodları:

```
public void listenForMessage() {
           new Thread(new Runnable() {
               @Override
               public void run() {
                   String msgFromGroupChat;
                   msgFromGroupChat=bufferedReader.readLine();

                   while (socket.isConnected()) {
                           try {
                                System.out.println(msgFromGroupChat);
               } catch (Exception e) {
                                   e.printStackTrace();
               }
           }).start();
       }
```

Yukaradaki fonksiyon ile server'dan gelen mesajları dinliyoruz. Override edilen run fonksiyonu ile client çalıştığı müddetçe mesajların okunması ve ekrana çıktı vermesi sağlanıyor.

```
public void sendMessage() throws Exception {
        try {
            Scanner scanner = new Scanner(System.in);
            while (socket.isConnected()) {
                String messageToSend = scanner.nextLine();
                    bufferedWriter.write(messageToSend);
                    bufferedWriter.newLine();
                    bufferedWriter.flush();
                }

            }
        } catch (IOException e) {
            closeEverything(socket, bufferedReader, bufferedWriter);
        }
    }
```

Yukarıdaki fonskiyon ile diğer clientlere gönderilmesi için Servera mesajla gönderiyoruz.Eğer herhangi bir gönderme hatası alınırsa okuma,yazma ve socket bağlantısını kesiyoruz.

```
    @Override
    public void run() {
        String messageFromClient;
        while (socket.isConnected()) {
            try{
                messageFromClient=bufferedReader.readLine();
                if(messageFromClient.charAt(0)=='|'){
                    String[] fullMessage=messageFromClient.split(" ");
                    for (String string : fullMessage) {
                        System.out.println(string);
                    }
                    System.out.println("message:"+fullMessage[3]+"\n");
                    broadcastMessagePrivate(fullMessage[3],fullMessage[1]);
                }else
                    broadcastMessageEveryBody(messageFromClient);
            }catch(IOException e){
                closeEverything(socket,bufferedReader,bufferedWriter);
                System.out.println("hata 19");
                break;
            }   }
    }
```

ClientHandlerde bulunan Run() fonksiyonu Runnable sınıfının implement edilen bir sınıftır. Bulunduğu server çalıştığı sürece gelen mesajı okur ve önünde “|” ifadesi varsa mesajı private olarak atıyor yoksa public olarak atıyor.

```
public void broadcastMessageEveryBody(String messageToSend){
        for(ClientHandler clientHandler:clientHandlers){
            try {
                if(!clientHandler.clientUsername.equals(clientUsername)){
                    clientHandler.bufferedWriter.write(clientUsername+":"+messageToSend);
                    clientHandler.bufferedWriter.newLine();

                    clientHandler.bufferedWriter.flush();
                }
            } catch (IOException e) {
                closeEverything(socket,bufferedReader,bufferedWriter);
            }
        }

    }
```

Yukaridaki fonksiyonumuzu tüm çevrimiçi kullanıcılara mesaj göndermek için kullanıyoruz.

```
 public void sendPrivate(String messageToSend,String receiverName){
        for(ClientHandler clientHandler:clientHandlers){
            try {
                if(clientHandler.clientUsername.equals(receiverName)){
                    clientHandler.bufferedWriter.write("| "+receiverName+"
"+messageToSend);
                    clientHandler.bufferedWriter.newLine();
                    clientHandler.bufferedWriter.flush();
                }
            } catch (IOException e) {
                closeEverything(socket,bufferedReader,bufferedWriter);
            }
        }

    }
```

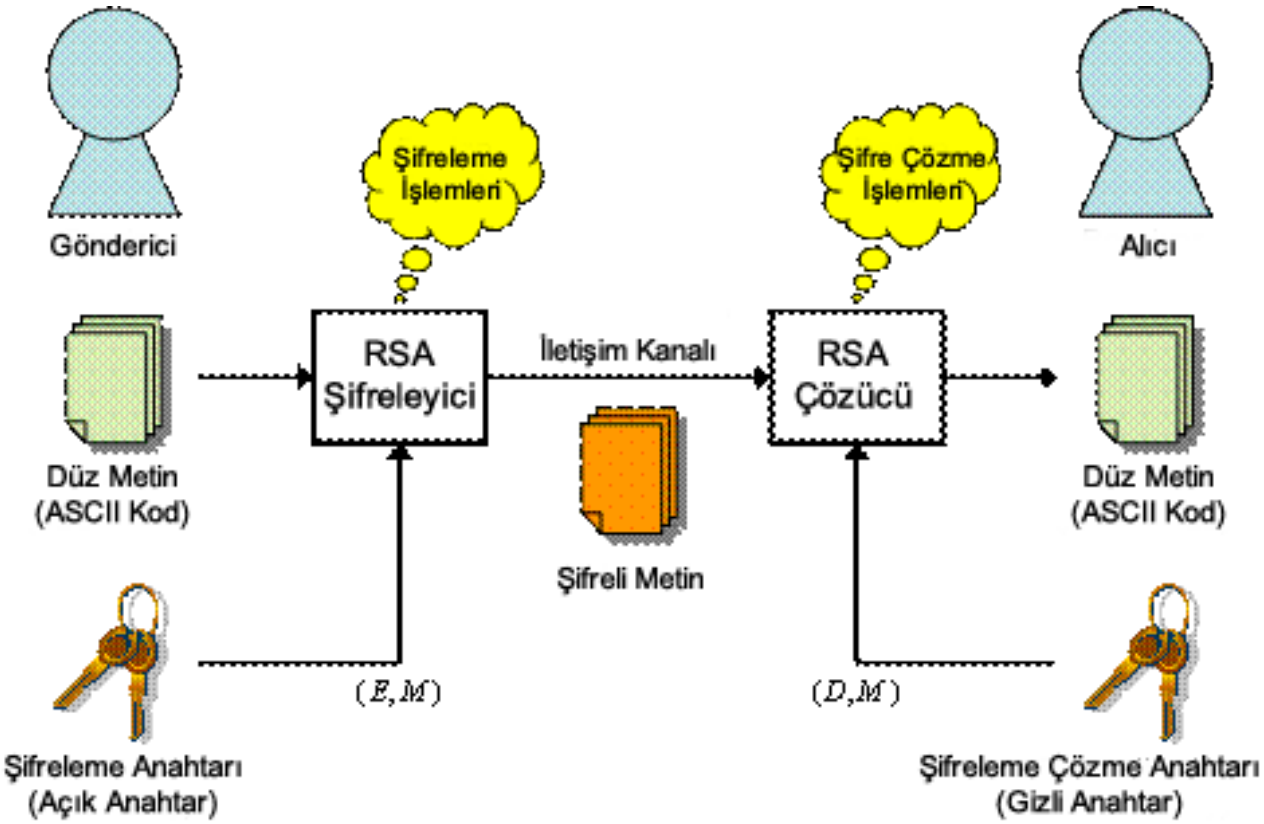
Bu fonksiyonumuzu terminalde @username ile gönderilen sadece username isimli kullanıcımıza gönderiyoruz.”|” eklememizin private mesajdan geldiğini göstermek içindir.

RSA Algoritması Nedir ? Nasıl Çalışır ?

RSA algoritması asimetrik bir kriptoloji algoritmasıdır. Asimetrik çalıştığı için iki farklı anahtar kullanır. Bu anahtarlara **Public Key** ve **Private Key** denilir. Adından anlaşılacağı üzere public key herkes tarafından bilinirken, private key sadece mesajı yayınlayan kişi tarafından bilinir.

Biz mesajlaşma uygulamamızda rsa’yı kullanarak mesaj göndereceğimiz kişinin public key’ini alarak ve göndereceğimiz mesajı encrypt fonksiyonumuza parametre vererek

şifreliyoruz. Şifreli mesajı deşifre etmek için alıcının kendine has private key'i ve gelen şifrelenmiş mesajı ve şifreli metni decrype fonksiyonuna parametre vererek deşifre işlemenini gerçekleştiriyoruz



RSA mesajlaşma şeması

```
    private void generateKeys() {
            GeneratePublicPrivateKeys generatePublicPrivateKeys = new
GeneratePublicPrivateKeys();
            Keys keys = generatePublicPrivateKeys.generateKeys("RSA", 512);
            publicKey = keys.getPublicKey();
            System.out.println("public key:"+getHexString(publicKey.getEncoded()));
            privateKey = keys.getPrivateKey();
            System.out.println("Keys generated for you: " + userName);
        }
```

generateKeys() fonkisyonumuz RSA şifreleme yöntemini kullanarak bizim belirlediğimiz 512 baytlık public key ve private key oluşturuyor.Ve terminale oluşturduğuna dair bir yazı çıkartıyor.

```
private static String encrypt(PublicKey publicKey, String message) throws Exception {
            Cipher encryptCipher = Cipher.getInstance("RSA");
            encryptCipher.init(Cipher.ENCRYPT_MODE, publicKey);

            byte[] cipherText = encryptCipher.doFinal(message.getBytes("UTF8"));

            return Base64.getEncoder().encodeToString(cipherText);
        }
```

Encrypt() fonkisyonu göndermek mesaj göndermek istediğimiz kişinin public keyini ve mesajımızı parametre alarak bunu RSA yöntemi ile şifreliyor.

```
 private static String decrypt(PrivateKey privateKey, String encrypted) throws Exception {
            byte[] bytes = Base64.getDecoder().decode(encrypted);

            Cipher decriptCipher = Cipher.getInstance("RSA");
            decriptCipher.init(Cipher.DECRYPT_MODE, privateKey);

            return new String(decriptCipher.doFinal(bytes), "UTF8");
        }
```

Decyrpt() fonksiyonu mesaj alan clientin kendi private keyini kullanarak gelen şifreli mesajı RSA yöntemi ile asıl metne deşifre ediyor.

Kaynakça:

https://www.ibm.com/docs/en/zos/2.4.0?topic=certificates-size-considerations-public-private-keys
https://www.ibm.com/docs/en/i/7.1?topic=communications-socket-programming