**ERCİYES ÜNİVERSİTESİ MÜHENDİSLİK FAKÜLTESİ**
**BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ**
**PROGRAMMING LABORATORY NOTLARI**

# 7. HAFTA NOTU
# (Soket Programlama)

*Bu not dersin 7. haftasında yapacağınız uygulamaya hazırlanmanızı sağlayacak olup temel kavramları anlatan bilgileri içermektedir.*

**DERSİN ÖĞRETİM ÜYESİ**

Prof. Dr. Bahriye AKAY

**DENEYİ YAPTIRAN ÖĞRETİM ELEMANI**

Arş. Gör. Gökhan AZİZOĞLU

**2026**
**KAYSERİ**

# 1. SOKET PROGRAMLAMA HAKKINDA TEMEL BİLGİLER

## 1.1. Soket Programlama Nedir?

Soket programlama, ağ üzerinden iki programın birbiriyle doğrudan iletişim kurmasını sağlayan bir yöntemdir. Bu iletişim, "soket" adı verilen yazılımsal uç noktalar üzerinden gerçekleştirilmektedir. Soket programlamada sunucu ve istemci olmak üzere genelde iki taraf vardır. Sunucu belli bir portu dinleyerek bağlantı beklemektedir. İstemci ise sunucuya bağlanarak veri göndermekte ya da veri talep etmektedir. Bu yapı internet üzerindeki pek çok sistemin temelini oluşturmaktadır. Web siteleri, mesajlaşma uygulamaları, çevrim içi oyunlar, dosya paylaşım sistemleri ve canlı veri akışı kullanan uygulamalar soket mantığıyla çalışmaktadır. Kısacası soket programlama, ağ iletişiminin temel taşlarından biridir.

## 1.2. IP Adresi ve Port Kavramı

Ağ üzerinde haberleşen her cihazın bir adresi vardır. Bu adres IP adresi olarak adlandırılmaktadır. Ancak yalnızca IP adresi, hangi uygulama ile iletişim kurulacağını belirtmek için yeterli değildir. Aynı cihaz üzerinde birden fazla uygulama aynı anda ağ erişimi kullanabileceği için, iletişimin hangi uygulama üzerinden yapılacağını göstermek amacıyla port numarası da kullanılmaktadır. Soket programlamada bağlantı kurulurken istemci, sunucunun IP adresini ve port numarasını bilmek zorundadır. Bu nedenle IP ve port bilgisi, ağ iletişiminin en temel bileşenlerinden biri olarak kullanılmaktadır.

## 1.3. TCP Protokolü

TCP protokolü, istemci ile sunucu arasında güvenilir bir bağlantı kurulmasını sağlamaktadır. Gönderilen verilerin sıralı biçimde karşı tarafa ulaşması, eksik ya da bozuk paketlerin yeniden gönderilmesi gibi işlemler bu protokol tarafından kontrol edilmektedir. Bu yönüyle TCP, özellikle mesajlaşma ve düzenli veri aktarımı gereken uygulamalarda sıkça tercih edilmektedir.

## 1.4. Bağlantı Kurma Süreci

TCP tabanlı bir uygulamada haberleşme başlamadan önce belirli adımlar izlenmektedir. Öncelikle sunucu tarafı, belirli bir port üzerinden dinleme yapacak şekilde hazırlanmaktadır. Daha sonra istemci, bu sunucuya bağlanma isteği göndermektedir. Sunucu bu isteği kabul ettiğinde iki taraf arasında bağlantı kurulmuş olmaktadır. Bağlantı kurulduktan sonra veri gönderme ve alma işlemleri başlamaktadır.

## 1.5. Veri Gönderme ve Veri Alma Mantığı

Soket programlamada gönderilen veriler ağ üzerinden byte dizileri şeklinde iletilmektedir. Bu nedenle bir mesaj gönderilmeden önce uygun biçimde byte dizisine dönüştürülmektedir. Karşı tarafta alınan byte verisi ise tekrar metne çevrilerek okunabilir hâle getirilmektedir. Veri gönderme ve alma işlemleri haberleşmenin temelini oluşturmaktadır. Bu yapı sayesinde istemci sunucuya mesaj gönderebilmekte, sunucu da gelen veriyi işleyip cevap verebilmektedir.

## 2. DENEY İÇERİSİNDE KULLANILACAK KÜTÜPHANELER VE FONKSİYONLAR HAKKINDA KISA BİLGİ

### 2.1. System.Net

Adresleme ve uç nokta tanımlama işlemlerinin temelini oluşturmaktadır. Soket tabanlı haberleşmede bir cihazın ağ üzerindeki konumunu belirlemek için IP adresi, ilgili uygulamanın hangi iletişim kapısı üzerinden veri alışverişi yapacağını tanımlamak için ise port numarası kullanılmaktadır. Bu kütüphane içerisinde yer alan "IPAddress" yapısı IPv4 ve IPv6 adreslerini temsil ederken, "IPEndPoint" sınıfı bir IP adresini port bilgisiyle birlikte ele alarak istemci ve sunucu tarafındaki bağlantı hedeflerinin tanımlanmasını sağlamaktadır. Sunucu uygulamasının belirli bir adrese bağlanması, istemcinin ise doğru hedefe yönlendirilmesi bu yapıların doğru kullanılmasına bağlıdır. Ağ iletişiminde bağlantının mantıksal olarak kurulabilmesi için önce bu adresleme katmanının doğru şekilde modellenmesi gerekmektedir. Kısacası, bu kütüphane haberleşmenin nereye yapılacağını belirleyen temel yapıyı sağlamaktadır.

### 2.2. System.Net.Sockets

İstemci ile sunucu arasındaki veri gönderme ve alma işlemleri bu kütüphane ile gerçekleştirilmektedir. Sunucu tarafında bağlantı beklemek, istemci tarafında sunucuya bağlanmak ve iki taraf arasında mesaj alışverişi yapmak bu yapı sayesinde mümkün olmaktadır. Deneyde kullanılacak temel haberleşme işlemleri doğrudan bu kütüphane üzerinden yürütülecektir.

### 2.3. System.Threading

Bu kütüphane, aynı anda birden fazla işlemin yürütülmesini sağlamak için kullanılmaktadır. Soket programlamada sunucu tarafı bir istemci ile haberleşirken aynı anda başka istemcilerden de bağlantı alabilmektedir. Böyle durumlarda her istemcinin ayrı bir iş parçacığında çalıştırılması programın daha düzenli ve verimli çalışmasını sağlayacaktır. Ayrıca kullanıcı arayüzü olan uygulamalarda donmaları önlemek için de thread yapısı önemli bir yere sahiptir. Bu nedenle çoklu bağlantı mantığını kurarken bu kütüphaneden yararlanılmaktadır.

### 2.4. StartServer(int port)

Bu fonksiyon, sunucunun çalışmasını başlatmak için kullanılacaktır. Fonksiyon çağrıldığında, program verilen port üzerinden bağlantıları dinlemeye başlayacaktır. Yani sunucu artık istemcilerden gelecek bağlantı isteklerini kabul edebilecek duruma gelecektir. Soket programlamada haberleşmenin başlayabilmesi için ilk olarak sunucu tarafının hazır hâle getirilmesi gerekmektedir. Bu hazırlık işlemi bu fonksiyon ile gerçekleştirilecektir.

### 2.5. ConnectToServer(string ip, int port)

Bu fonksiyon, istemcinin sunucuya bağlanmasını sağlayacaktır. Fonksiyon içerisinde sunucunun IP adresi ve port bilgisi kullanılarak bağlantı oluşturulacaktır. Bağlantı başarılı olursa istemci sunucu ile veri alışverişi yapabilecek hale gelecektir. Bu fonksiyon, istemci tarafındaki haberleşmenin başlangıç adımıdır.

### 2.6. SendMessage(Socket client, string message)

Hazırlanmış olan bir veriyi karşı tarafa göndermek için kullanılmaktadır. Gönderilecek veri önce uygun biçimde byte dizisine çevrilmekte, ardından soket üzerinden iletilmektedir. Bu fonksiyon hem istemci hem de sunucu tarafında kullanılabilmektedir. Böylece iki taraf arasında veri iletimi sağlanmış olmaktadır.

### 2.7. ReceiveMessage(Socket client)

Karşı taraftan gelen veriyi okumak için kullanılmaktadır. Gelen veri önce byte olarak alınmakta, daha sonra okunabilir metne dönüştürülmektedir. Bu fonksiyon sayesinde gönderilen veri ekranda gösterilebilmekte veya program içerisinde işlenebilmektedir. Veri alma işlemi, soket haberleşmesinin temel parçalarından biridir.

### 2.8. HandleClient(Socket client)

Sunucuya bağlanan bir istemci ile yapılacak işlemleri yönetmek için kullanılmaktadır. Sunucu bir bağlantı kabul ettikten sonra, o istemciden gelen verileri dinleme ve gerekli cevapları verme işlemlerini bu fonksiyon içinde gerçekleştirmektedir. Özellikle birden fazla istemci ile çalışılan uygulamalarda her istemci için ayrı bir işlem yürütülmesi gerektiğinden, bu fonksiyon önemli bir görev üstlenmektedir.

### 2.9. CloseConnection(Socket client)

Haberleşme bittikten sonra bağlantıyı doğru şekilde kapatmak için kullanılmaktadır. Soket bağlantılarının açık bırakılması sistem kaynaklarının gereksiz kullanılmasına neden olabilmektedir.

## 3. ÖRNEK UYGULAMA VE TEMEL KOD PARÇALARI

Bu bölümde, soket programlamanın temel mantığını göstermek amacıyla basit bir istemci-sunucu uygulamasına ait örnek kod parçaları verilmiştir. Uygulamada sunucu belirli bir port üzerinden bağlantı beklemekte, istemci ise bu sunucuya bağlanarak metin tabanlı mesaj göndermektedir. Gönderilen mesaj sunucu tarafında alınmakta ve ekrana yazdırılmaktadır. Bu örnek, bağlantı kurma, veri gönderme ve veri alma adımlarını birlikte görülmesini amaçlamaktadır.

### 3.1.Sunucunun Başlatılması

Aşağıdaki kod parçasında sunucu tarafında bir `TcpListener` nesnesi oluşturulmakta, belirli bir port üzerinden dinleme başlatılarak istemci bağlantısı beklenmektedir.

```
using System;
using System.Net;
using System.Net.Sockets;
using System.Text;

class ServerProgram
{
    static void Main()
    {
        StartServer(5000);
    }

    static void StartServer(int port)
    {
        TcpListener server = new TcpListener(IPAddress.Any, port);
        server.Start();

        Console.WriteLine("Sunucu başlatıldı. Bağlantı bekleniyor...");

        TcpClient client = server.AcceptTcpClient();
        Console.WriteLine("Bir istemci bağlandı.");

        ReceiveMessage(client);

        client.Close();
        server.Stop();

        Console.ReadLine();
    }
}
```

### 3.2. İstemcinin Sunucuya Bağlanması

Aşağıdaki kod parçasında istemci tarafı sunucuya bağlanmakta ve bağlantı başarılı olduktan sonra mesaj gönderme işlemine geçmektedir.

```
using System;
using System.Net.Sockets;
using System.Text;

class ClientProgram
{
    static void Main()
    {
         try
         {
            ConnectToServer("127.0.0.1", 5000);
         }
         catch (Exception ex)
         {
            Console.WriteLine($"Hata: {ex.Message}");
         }
    }

    static void ConnectToServer(string ip, int port)
    {
        using TcpClient client = new TcpClient();

        try
        {
            client.Connect(ip, port);
            Console.WriteLine("Sunucuya bağlanıldı.");
            Console.WriteLine("Çıkmak için 'exit' yazınız.");

            using NetworkStream stream = client.GetStream();

            while (true)
            {
                Console.Write("Mesaj: ");
                string? message = Console.ReadLine();

                if (string.IsNullOrWhiteSpace(message))
                { continue; }

                if (message.Equals("exit", StringComparison.CurrentCultureIgnoreCase))
                { break; }

                SendMessage(stream, message);
            }
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Hata: {ex.Message}");
        }
    }
}
```

### 3.3.Mesaj Gönderme İşlemi

Aşağıdaki metot, istemci ya da sunucu tarafında hazırlanmış bir mesajın karşı tarafa gönderilmesini sağlamaktadır. Mesaj önce byte dizisine dönüştürülmekte, ardından ağ akışı üzerinden iletilmektedir.

```
static void SendMessage(TcpClient client, string message)
{
    NetworkStream stream = client.GetStream();
    byte[] data = Encoding.UTF8.GetBytes(message);
    stream.Write(data, 0, data.Length);

    Console.WriteLine("Mesaj gönderildi.");
}
```

### 3.4.Mesaj Alma İşlemi

Aşağıdaki metot, karşı taraftan gelen veriyi okumakta ve byte dizisini tekrar metne dönüştürmektedir. Bu metotta gelen veri önce `buffer` adı verilen geçici bellek alanına alınmaktadır. `Read()` metodu ile okunan byte sayısı belirlendikten sonra yalnızca alınan veri kadar bölüm metne çevrilmektedir. Bu sayede gereksiz bellek alanı işleme katılmamış olur.

```
static void ReceiveMessage(TcpClient client)
{
    NetworkStream stream = client.GetStream();
    byte[] buffer = new byte[1024];

    int byteCount = stream.Read(buffer, 0, buffer.Length);
    string message = Encoding.UTF8.GetString(buffer, 0, byteCount);

    Console.WriteLine("Alınan mesaj: " + message);
}
```

### 3.5. Sürekli Mesaj Dinleme Yapısı

Bazı uygulamalarda yalnızca tek bir mesaj almak yeterli değildir. Bağlantı açık kaldığı sürece mesajların sürekli dinlenmesi gerekebilir. Böyle durumlarda veri alma işlemi bir döngü içerisinde yürütülür.

```
static void HandleClient(TcpClient client)
{
    NetworkStream stream = client.GetStream();
    byte[] buffer = new byte[1024];

    while (true)
    {
        int byteCount = stream.Read(buffer, 0, buffer.Length);

        if (byteCount == 0)
        {
            break;
        }

        string message = Encoding.UTF8.GetString(buffer, 0, byteCount);
        Console.WriteLine("Alınan mesaj: " + message);
    }

    client.Close();
}
```

### 3.6. Çoklu İstemci İçin Thread Kullanımı

Sunucuya aynı anda birden fazla istemci bağlanacaksa, her istemcinin ayrı bir iş parçacığında ele alınması gerekmektedir. Aşağıdaki örnekte her bağlantı için yeni bir thread başlatılmaktadır. Bu yapıda sunucu yeni bağlantıları kabul etmeye devam ederken, her istemci ile ilgili işlemler ayrı thread içinde yürütülmektedir. Böylece bir istemci ile iletişim sürerken başka istemciler de sunucuya bağlanabilmektedir.

```
using System.Threading;

static void StartServerLoop(int port)
{
    TcpListener server = new TcpListener(IPAddress.Any, port);
    server.Start();

    Console.WriteLine("Sunucu başlatıldı. Bağlantı bekleniyor...");

    while (true)
    {
        TcpClient client = server.AcceptTcpClient();
        Console.WriteLine("Bir istemci bağlandı.");

        Thread clientThread = new Thread(() => HandleClient(client));
        clientThread.Start();
    }
}
```

## 4. DERSİN ÖĞRENME ÇIKTILARI

Bu deney çalışmasının sonunda aşağıdaki kazanımları elde edilmesi beklenmektedir:

- İstemci-sunucu haberleşmesinin temel mantığının kavraması,
- Ağ üzerinden iletişim kuran iki farklı uygulamanın nasıl çalıştığının anlanması,
- TCP tabanlı iletişim yapısının temel özelliklerinin tanınması,
- IP adresi, port ve soket kavramlarının ağ iletişimindeki yerinin açıklanabilmesi,
- Ağ üzerinden veri gönderme ve alma işlemlerinin temel çalışma mantığının kavranması,
- Temel düzeyde bir istemci-sunucu uygulamasının geliştirilebilmesi,
- Eşzamanlı çalışmanın ağ uygulamalarındaki öneminin fark edilmesi.