**F.Ü. Mühendislik Fakültesi**
**Bilgisayar Mühendisliği Bölümü**
**Bilgisayar Sistemleri Laboratuvarı**

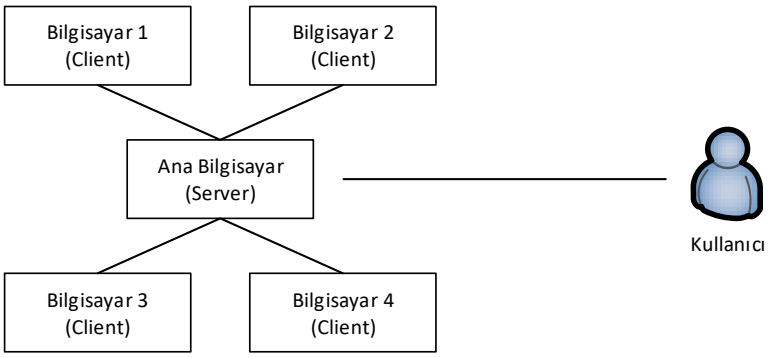
# DENEY NO: 6
# JAVA İLE DAĞITIK PROGRAMLAMA

**Deneyin Amacı:**

Bu deneyde nesne tabanlı bir programlama dili olan JAVA'nın genel yapısı ve ağ ortamında dağıtık programlama (client/server) için Java'nın nasıl kullanıldığını görmek amaçlanmıştır. Bunun için deneyde önce dağıtıklık kavramı anlatılmış daha sonra uygulamalara geçilmiştir.

## 1. Dağıtık Sistem Nedir

Katmanların birbirinden uzak yerlerde bulunduğu sistemlere **dağıtık sistemler** (distributed systems) adı verilir. Dağıtık sistemler, farklı bilgisayarlara ait donanım ve yazılım bileşenleri arasında yalnızca mesajlaşma yöntemleri kullanılarak haberleşme ve koordinasyonunun sağlanabildiği, ortak bir amaca hizmet eden network sistemleridir. Başka bir deyişle, fiziksel olarak bağımsız bilgisayarların kullanıcıya tek bir bilgisayar gibi görünerek, iş yükünün paylaşıldığı sistemlerdir. Bir dağıtık sistem, kurumsal bir şirketteki yapılanmaya benzetilebilir. Şirket kendi içerisinde alt birimlere ayrılmış, iş paylaşımı yapmış olsa da, müşteri şirket ile muhatap olmaktadır. Şekil 1'de kullanıcı ana bilgisayar ile muhatap olmakta, ana bilgisayar işi alt bilgisayarlara bölüştürmektedir. Sonucu kullanıcıya yine ana bilgisayar gösterir.




*Şekil 1. Dağıtık Sistem Mimarisi*

Dağıtık sistemlerdeki en önemli nokta süreçler arası haberleşmedir. Süreçler arasında mesaj iletimi önemli ve zor bir konudur. Sistem içindeki bilgisayarlar arası mesafe, networkun durumu (hatların yoğun olması) ve diğer fiziksel şartlar mesaj iletimini etkileyebilmektedir. Bu nedenle mesaj iletimini düzenlemek amacıyla çeşitli orta katman yazılımları ve mesaj gönderme protokolleri oluşturulmuştur. RPC (Remote Procedure Call) ve RMI (Remote Object Invocation) bu orta katman yazılımlardan önemli iki tanesidir. TCP ve UDP de en çok kullanılan mesaj transfer protokolleridir. Bilgisayarı bu protokoller yardımıyla birbiriyle haberleştirmek için öncelikle aynı ağa bağlı olmaları gerekmektedir.

## 2. Niçin Dağıtık Programlama?

**Kaynak Paylaşımı:** Eğer farklı kapasitelerdeki farklı siteler, diğer bir siteye bağlı ise, bir sitedeki bir kullanıcının, diğer sitede mevcut kaynakları kullanması mümkün olabilir. Örneğin; A sitesindeki bir kullanıcı, B sitesinde bulunan bir lazer yazıcıyı kullanıyor olabilir. Bu sırada, B'deki kullanıcı da A'da bulunan bir dosyaya erişebilir.

**Hesaplamayı Hızlandırma:** Belli bir hesaplama, eş zamanlı olarak çalışabilen alt hesaplamalar şekilde bölümlenebilirse, dağıtık bir sistem farklı siteler arasında alt hesaplamaları dağıtmaya izin verir, alt hesaplamalar eş zamanlı olarak çalıştırılabilir ve böylece hesaplamalarda hızlanma sağlanır. Ek olarak, eğer belli bir site işlerle aşırı yüklenmişse, işlerin bir kısmı hafif yüklü diğer sitelere taşınabilir. Buna da **yük paylaşımı** denir.

**Güvenilirlik:** Dağıtık bir sistemde bir site başarısız olursa, kalan siteler çalışmaya ve daha güvenilir sistemler vermeye devam ederler. Eğer bir sistem, birden çok, büyük ve özerk sistemlerden oluşuyorsa (yani genel amaçlı bilgisayarlarsa), onların birinin başarısızlığı geri kalanını etkilemez. Fakat eğer, sistem küçük makinelerden oluşuyorsa, onların her biri de bazı

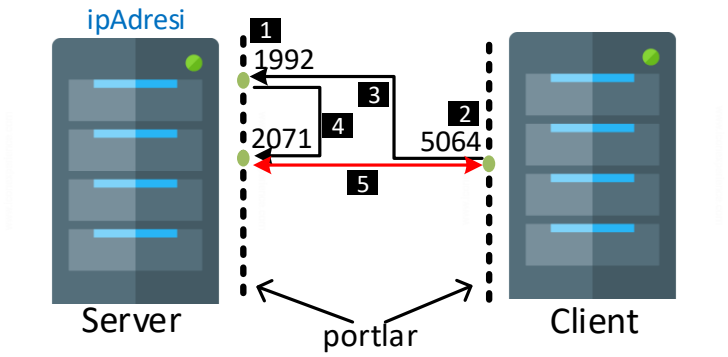
önemli sistem fonksiyonlarından sorumlu ise (örneğin; dosya sisteminin ya da giriş/çıkışın terminal özelliği), tek bir hata böyle sistemlerin çalışmasını durdurabilir. Genel olarak, sitelerin bazılarında başarısızlıklar olsa bile, (hem donanım hem de verinin) yeteri kadar yedeklenmesiyle sistem çalışmasına devam edebilir.

Dağıtık programlamanın getirdiği avantajlar şu şekilde özetlenebilir:

- Ölçeklendirilebilirliği (sisteme ekleme-çıkarma yapma kolaylığını) artırır.
- Sadece belli donanımlara sahip bilgisayarlarda çalışan yazılımlardan bir sistem oluşturabilme imkanı sağlar.
- Servisleri ayrı bilgisayarlarda çalıştığından sistem daha güvenli olur ve erişim kontrolleri daha kolay yapılır.
- Bilgisayarların işlem güçlerini birleştirerek süper sanal bilgisayarlar oluşturulabilir. (https://boinc.berkeley.edu/ - BOINC isimli süper bilgisayar projesinde, kullanıcılar yükledikleri uygulamalar ile CPU, ram gibi kaynaklarını proje kapsamında çözülmesi amaçlanan problemler için kullandırtabilmektedir.)

## 3. Java ile Dağıtık Programlama Nasıl Gerçekleştirilir?

Dağıtık programlama bir işin birden fazla bilgisayar tarafından yapılmasını hedefler. Ortada birden fazla eleman varsa, bunların birlikte çalışabilmesi için aralarında haberleşmesi gerekir. Dağıtık programlama temel olarak sistemdeki bilgisayarların haberleşmesine dayalıdır. Bu haberleşme ağ bağlantıları sayesinde gerçekleştirilir. Dağıtık programlama deneyini gerçekleştirmek için öncelikle kullanacağımız bilgisayarların aynı ağa bağlı olması gereklidir. Aralarında ağ bağlantısı bulunan bilgisayarlardan biri server biri client olarak ayarlanarak bağlantı test edilir. Bağlantının gerçekleştirilme aşamaları şekil 2'de gösterildiği gibi olur.




*Şekil 2. Server-Client Arasında Bağlantının Sağlanması*

1- Server bilgisayarda bir adet ServerSocket nesnesi oluşturularak müsait bir port dinlenir (şekilde port numarası 1992 seçilmiş).
2- Client bilgisayarda bir adet Socket nesnesi oluşturularak server bilgisayarın açılan portuna bağlantı denemesinde bulunulur (ipAdresi,1992 parametreleri ile).
3- Porta ulaşılınca 4. Adıma geçilir.
4- ServerSocket nesnesi accept() fonksiyonu aracılığıyla boş bir port açarak gelen isteği bu port ile eşleştirir.
5- İki bilgisayar açılan portlar aracılığıyla iletişim sağlar (2071 ve 5064 portları).

Yukardaki adımları gerçekleştirmek için server tarafında yazılması gereken Java kodu aşağıdaki gibidir:

```
// Belirtilen PORT_NUMARASI'nı yöneten yeni bir socket nesnesi ouşturulur (Adım 1).
ServerSocket soket = new ServerSocket(PORT_NUMARASI);

System.out.println("Baglanti Bekleniyor");
// Belirtilen PORT_NUMARASI'nı dinleyerek yeni bağlantı bekler  (Adım 4).
Socket baglanti = soket.accept();
System.out.println("Baglanti Gerceklesti");
```

Bilgisayarlarda 16 bitle ifade edilebilecek sayıda ($2^{16}$ = 65536 adet) port bulunmaktadır. Bu portlardan ilk 1024 ü özel amaçla ayrılmış portlardır. Bilgisayar sistemlerinde indisler 0'dan başladığı için 1024. port bizim kullanabileceğimiz portların birincisi, 65535. port ise sonuncusu olmaktadır. Açılacak olan `PORT_NUMARASI` bu aralıklarda bir değer alabilir.

Client tarafına ise aşağıdaki kod yazılarak bağlantı sağlanabilir.

```
Socket baglanti = new Socket(IP_ADRESI, PORT_NUMARASI);
```

Burada yer alan `IP_ADRESI` serverin ip adresi ile değiştirilmelidir. Deneme için server ve client aynı bilgisayarda çalıştırılabilir. Kendi bilgisayarında açılan porta erişmek için `IP_ADRESI` kısmına "localhost" yazılmalıdır. Burdaki port adresi serverSocket oluştururken

belirten PORT_NUMARASI değeridir. Şekil 2'deki örnek için PORT_NUMARASI değeri 1992'dir (Adım 2,3,5).

Yukarda yer alan kodlar client.java ve server.java isimli iki ayrı main içerisine atılıp, önce server sonra client çalıştırıldığında bağlantının başarılı bir şekilde gerçekleşmesi beklenmektedir. Ancak bağlantı gerçekleştikten sonra bir işlem yapılmadığından programlar sonlanır.

Server ile Client arasındaki bağlantı kurulduktan sonra her iki taraf da sahip oldukları soket nesneleri (baglanti) yardımıyla birbiriyle veri alışverişi yapabilirler.

**Veri Gönderimi:**

```
PrintWriter output = new PrintWriter(
     baglanti.getOutputStream(),true
);
output.println("Gönderilecek Mesaj");
```

**Veri Alımı:**

```
BufferedReader in = new BufferedReader(
     new InputStreamReader(baglanti.getInputStream())
);
in.readLine();
```

Burada in.readLine satır gelmesi için bekler, bir sonraki adıma geçmez. Bunun için in.readLine komutunda beklemesini istemiyorsak bundan önce in.ready() fonksiyonuyla okunacak bir şey olup olmadığı kontrol edilir, varsa okuma komutu çağrılır. Sürekli veri okunması yapılacak ise bu işlem döngüye sokulur.
İki cihazın birbiriyle haberleşmeleri için öncelikle mesaj yapılarını bilmeleri gerekmektedir. A ile B arasındaki sayılar arasında C'yi arayacağımız bir uygulama olduğunu düşünerek, "A-B-C" gibi bir mesaj geldiğini varsayalım. Bu mesajı ayrıştırmak ve değerleri uygun değişkenlere atamak için string işlemlerinden yararlanılır. Birinci parametre olarak A'yı ikinci parametre olarak B'yi üçüncü parametre olarak ise C'yi almak için yazmamız gereken kod aşağıdaki gibi olur. Burada A,B,C'nin tipleri iki bilgisayar tarafından da bilinmeli (haberleşme kuralı).

```
veri = “2000-200000-deneme”;
String [] x = gelen.split("-"); // veriyi - işaretlerine göre ayır
long A = Long.parseLong(x[0]); // 1. parametre long değer olmalı
long B = Long.parseLong(x[1]); // 2. parametre long değer olmalı
String C = x[2]; // 3. Parametre string değer olmalı
```

Serverden cliente göndereceğimiz mesaj, cliente yaptırmak istediğimiz komut olacaktır. String olarak okunan komut ayrıştırılarak ilgili işlemler cliente yaptırılır. Ürettiği sonuç tekrar servere cevap olarak döndürülür. 1 client 1 server için işlem bu şekilde gerçekleşir. Ancak soket dinleme işlemi, porta bir bağlantı gerçekleşene kadar alt satıra geçmez. Yani bir serverde birden fazla client bağlantısı dinlemek istiyorsak, serverde oluşturduğumuz socket nesnelerini Threadler yardımı ile oluşturup dinlemek gerekmektedir. Thread yardımıyla birbirinden bağımsız iş parçacıkları kendilerine ait soketleri dinler ve bir veri geldiğinde bunu algılayabilir. Javada thread oluşturmak için sınıfı Thread sınıfından türetmek ve ardından run fonksiyonunu override etmek gerekir.

**Örnek thread oluşturma :**

```
public class Ornek extends Thread {

    public void run() {
        int i=0;
        while(true){
          System.out.print(i++ + " ");
       }
    }
    public static void main(String args[]) {
        (new Ornek()).start();
        (new Ornek()).start();

    }
}
```

Thread sınıfından türetilen bir nesnenin start metodu çağrıldığında, run içerisine yazdığımız kodları tetiklenir. Yukardaki kod ile Ornek adında bir thread oluşturduk ve main fonksiyonu içerisinden bu threadi tetikleyerek çalıştırdık. Hemen ardından aynı threadi bir kez daha çalıştırdık. Sonuçta oluşturduğumuz sonsuz döngü sistemin tüm kaynaklarını tüketmemiş oldu ve iki sonsuz döngü aynı anda çalıştırılmış oldu. Thread iki kez tetiklendiğinden dolayı yukardaki kodun aynı sayıdan ikişer kez yazdırması beklenmektedir. Ancak işlemler eş zamanlı çalıştığından ekranda gözüken çıktılar sıralı olarak görülmez.

## 4. Deney Senaryosu

ABC sitesinin kullanıcılarına ait bilgiler veritabanında tutulmaktadır. ABC sitesi kayıt olurken istediği kullanıcı adını olduğu gibi; şifreyi ise md5 fonksiyonundan geçirerek kaydediyor. Kullanıcı giriş yapmaya çalıştığı zaman girişte kullanılan kullanıcı adı olduğu gibi, şifre ise md5 fonksiyonundan geçirildikten sonra veritabanında bu değerlere karşılık bir kayıtın olup olmadığı kontrol ediyor ve giriş işlemi bu şekilde yapılıyor. Şifre veritabanında hash hali ile tutulduğundan, veritabanında kayıtlı şifre bilgisi başkaları tarafından ele geçirilse dahi şifrenin öğrenilmesinin önüne geçilmiş olunuyor.

ABC sitesindeki veritabanı bilgilerinin bir şekilde elimize ulaştığını varsayarak; Md5 fonksiyonundan geçirilmiş şifresini bildiğimiz X kullanıcısına ait şifreyi bulmamız gerekmektedir. Ancak md5 tek yönlü bir algoritma olduğu için hash halindeki veriden orjinal şifreyi elde etmemiz mümkün değildir. Bunun için sitede istenen şifrenin 8 basamaklı bir sayı olduğu bilgisinden de yola çıkarak tek tek tüm olasılıkları md5 fonksiyonundan geçirmemiz ve sonucu elimizdeki md5li veri ile karşılaştırmamız gerekmektedir (kaba kuvvet algoritması). Ancak bunu bir bilgisayardan yapmaya çalıştığımızda T süre kadar zaman alır. Bu süreyi kısaltmak için belirli aralıktaki şifreleri farklı bilgisayarlara denettirerek (dağıtık programlama) sonuca ulaşmak planlanmaktadır. Deneyimizde bir adet iş dağıtan bilgisayar ve iş yapan en az iki bilgisayar olacaktır.

## 5. MD5 (Message Digest 5) Algoritması Nedir?

Tek yönlü bir özütleme (hashing) algoritmasıdır. Girilen veri ister bir karakterden oluşsun isterse terabaytlar büyüklüğünde bir dosya olsun çıkışta 128 bit (her biri 16'lık tabanda olan 32 rakam uzunluğunda) bir çıktı üretir. Sonsuz giriş kümesine karşılık $16^{32}$ kadar çıktı olacağından aynı sonucu verecek birden fazla girdi bulunacaktır. Ancak şifre gibi küçük veriler için böyle bir çakışma söz konusu değildir. Bundan dolayı şifreler genellikle veritabanlarında bu şekliyle tutulur. Girilen şifreler bu fonksiyondan geçirildikten sonra karşılaştırma yapılır. Tabloda farklı girişler için oluşturulan md5 değeleri verilmiştir.

| | |
|---|---|
| 1 | c4ca4238a0b923820dcc509a6f75849b |

| | |
|---|---|
| fırat üniversitesi | 44d4177ff9be0f3e45f27847719d1107 |
| Fırat Üniversitesi | a9ccc25be1bd49a199a69814d33e7b0d |

Md5 fonksiyonu aşağıdaki şekilde verilmiştir. Sizden yapmanız istenilen bir döngü ile dışardan belirlenen aralıktaki tüm sayıların md5 fonksiyonundan geçirilmiş hallerini, yine dışardan girilen özütlenmiş değer ile kıyaslamanızdır.

**Java Md5 fonksiyonu:**

```
private static String getMd5(long sayi) throws Exception{
      String yourString = Long.toString(sayi);
      byte[] bytesOfMessage = yourString.getBytes("UTF-8");

      MessageDigest md = MessageDigest.getInstance("MD5");
      byte[] thedigest = md.digest(bytesOfMessage);
      BigInteger bigInt = new BigInteger(1,thedigest);
      String hashText = bigInt.toString(16);
      while(hashText.length() < 32 ){
         hashText = "0"+hashText;
      }
      return hashText;
}
```

## 6. Deneyin Adım Adım Gerçekleştirilme Aşamaları

1. Dışardan başlangıç, bitiş ve özütlenmiş veri bilgilerini alan bir fonksiyon oluşturulur. Bu fonksiyon başlangıç ve bitiş değerleri arasındaki tüm değerleri md5 fonksiyonundan geçirerek özütlenmiş halleri bulunur. Bulunan değer, dışardan girilen özütlenmiş veri bilgisi ile aynı ise fonksiyon o anda aktif olarak işlem gören sayısı geri döndürür.
2. Main metodu içerisinde bir string tanımlanır. Bu string "Başlangıç-Bitiş-a12cdd…" gibi üç parametreyi aralarına boşluk, tire, virgül gibi bir karakter ekleyerek ifade eder. Bu stringin çözümlenmesi için bir split metodu yazılarak, ilk değer başlangıç, ikinci değer bitiş ve üçüncü değer ise özütlenmiş veri yerine konulur ve 1. Adımda oluşturulan fonksiyon çağrılır.
3. Bilgisayarlar aynı ağa bağlanır.
4. Bir bilgisayar başka bilgisayarların bağlanması için Java Socket sınıfında yararlanarak port açar ve diğer bilgisayar buna bağlanır. İki bilgisayar arasında mesajlama işlemi test edilir.
5. Diğer bilgisayarlarda istemciden gelen mesaj 2. Maddedeki formatta gönderilir. Alınan mesaj 2. Maddedeki string yerine yazılarak kod çalıştırılır.
6. Bir bilgisayar açtığı portu dinlerken tüm kaynakları onun için harcar ve başka bir işlem yapamaz. Birden fazla bilgisayarın bağlanabilmesi için ana bilgisayarda port dinleme işlemi thread kullanacak şekilde uyarlanır.

İlk 5 maddenin gerçekleştirilmesi 80 puan,

6. maddenin gerçekleştirilmesi 20 puan üzerinden hesaplanacaktır.

## 7. Hazırlık Soruları

1- Dağıtık programlama nedir? Dağıtık programlamaya niçin ihtiyaç duyulur?

2- Java'da socket sınıfı ne işe yarar? Dağıtık programlamada niçin socket sınıfı kullanılır?

3- Sunucuda 2010 nolu port açtıran ve istemcinin bağlanmasını bekleyen satırları yazınız (2 satır). İstemci bağlandığı zaman konsolda 'istemci bağlandı' yazdırın.

4- İstemciden 192.168.1.128 ip adresine sahip sunucunun 5040. Portuna bağlanmak için gerekli nesneyi oluşturunuz.

5- Soketten veri göndermek için kullanılan nesneyi oluşturun.

6- Soketten veri almak için kullanılan nesneyi oluşturun.

7- Hash algoritması nedir? Deneyde ne için kullanılmıştır?

8- Thread nedir? Ne işe yarar? Bu deneydeki kullanılma amacı nedir?

9- Java programlama dilinde Thread nasıl oluşturulur?