KARADENİZ TEKNİK ÜNİVERSİTESİ
BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ
BİLGİSAYAR AĞLARI LABORATUARI

Soket Programlama

1. Giriş

JAVA dili süreçlerarası iletişim için TCP ve UDP olmak üzere iki farklı socket yapısı kullanır. Her iki socket yapısı da haberleşmede Client-Server modelini kullanır. Bu deneyde, Server-Client soket setlemeleri ve Multithreading konularından bahsedildikten sonra TCP yapısına göre bir soket uygulaması üzerinde Producer-Consumer probleminin çok threadli çözümü üretilen kaynak stacke itilerek, tüketilen de stackten çekilerek gerçeklenecektir.

2. Server Socket Setlemeleri

Server Socket setleme adımları aşağıda verilmiştir:

- **ServerSocket Nesnesi Türetme:** `servSock` isimli `ServerSocket` nesnesi özel amaçlar için ayrılmış numaraların dışında, yani 1024-65535 arası bir port numarası seçerek aşağıdaki gibi oluşturulur:

```
ServerSocket servSock = new ServerSocket(1234);
```

- **Server'ı Bekleme Konumuna Getirme:** `servSoc`, `ServerSocket` sınıfına ait `accept()` methodunu kullanarak herhangi bir Client'ın bağlanmasını bekler. Bağlantı kurulduğunda `link` socket nesnesi şu şekilde türetilir:

```
Socket link = servSock.accept();
```

- **Yollanacak Veriler için Input/Output Stream Setleme:** Client bilgisayarlardan gelecek mesajları almak ve onlara mesaj göndermek için Socket sınıfına ait `getInputStream` ve `getOutputStream` methodları kullanılarak `input` ve `output` isimli stream nesneleri aşağıdaki gibi türetilir:

```
Scanner input = new Scanner(link.getInputStream());
PrintWriter output = new PrintWriter(link.getOutputStream(),true);
```

- **Veri Gönderme ve Alma:** Veri gönderme ve alma işlemleri için sırasıyla `println()` ve `nextLine()` methodları kullanılır:

```
output.println(".....");
String message = input.nextLine();
```

- **Bağlantıyı Sonlandırma:** Bağlantı Socket sınıfının `close()` methodu ile sonlandırılır:

```
link.close();
```

3. Client Socket Setlemeleri

Server ile bağlantı kurma dışında Client socket setlemeleri yukarıda Server için anlatılanlarla aynıdır. Server ile bağlantı kurmak için `link` socket nesnesi Server'ın ismi (veya IP adresi) ve port numarası parametreleri yardımıyla aşağıdaki gibi türetilir:

```
Socket link = new Socket( "ServerName" , 1234);
```

4. Client-Server Uygulaması

Kaynak kodlardan **Socket** klasöründe basit bir Client-Server uygulaması verilmiştir. Uygulama test edilirken öncelikle `TCPServer.java` programı koşularak Server başlatılır. Sonra `TCPClient.java` koşularak Server ile bağlantı kurulur. Bu uygulamada Client tarafından gönderilen mesajlar Server tarafından sayılır ve "BYE BYE" mesajı geldiğinde bağlantı sonlandırılırken toplam mesaj sayısı yazılır. Örnek bir program çıktısı aşağıdaki gibidir:

SERVER

```
[0] Waiting for connection...
[3] Received Message : MERHABA
[5] Received Message : NASILSIN
[7] Because of Received Message: BYE BYE
 * Closing connection... *
```

CLIENT

```
[1] Connected to Server
[2] Enter message : MERHABA
[4] Enter message : NASILSIN
[6] Enter message : BYE BYE

[8] SERVER > 3 messages received.
* Closing connection... *
```

5. Multithreading

Server bilgisayara 1'den fazla Client bilgisayarın bağlantı kurması durumunda Server'da, her bir Client için yeni bir thread başlatılmalıdır. Bunun için öncelikle `Thread` sınıfı miraslanarak bir thread nesnesi türetilir. `Start()` methodu ile bu threadin yapacağı işin kodunu barındıran `run()` methodu çağrılır. Çok sayıda thread aynı anda koşarken birbirlerine zaman ayırmaları için `sleep()` methodu kullanılır ve parametre olarak aldığı milisaniye kadar askıda durur.

Kaynak kodlardan `Multithreading` klasöründe, biri ekrana 5 kere **"Hello"** diğeri de `0-4` arası sayıları yazan iki threadi çağıran `ThreadHelloCount` adlı java programı için örnek çıktı aşağıda verilmiştir:

```
Hello!
0
1
Hello!
2
3
Hello!
Hello!
4
```

Yukarıdaki program çıktısına dikkat edilirse threadler farklı sıklıklarla çağrılmıştır. Bunun nedeni `sleep()` methodundaki `Math.random()*1000` ifadesidir. Böylece random fanksiyonu ile her bir thread `0-1` saniye arasında değişen farklı sürelerde askıda durmaktadır. Dolayısıyla `ThreadHelloCount` adlı program her koşuşunda farklı bir çıktı üretecektir.

Kaynak kodlardan **Multithreading** klasöründe daha önce anlatılan socket uygulamasının çok threadli uygulaması verilmiştir. `MultiServer.java` programında, `Thread` sınıfını miraslayan `ClientHandler` sınıfı herbir Client bağlantısında bir thread başlatmaktadır. `MultiClient.java` programı socket uygulamasındaki `TCPClient.java` ile hemen hemen aynıdır.

6. Senkronize Edilmiş Threadler

Farklı threadlerin ortak kullandıkları kaynaklara eşzamanlı erişimleri yanlış sonuçlar üretmeye neden olabilir. O yüzden ortak kaynaklara eşzamanlı erişimi engelleyecek bir mekanizmaya ihtiyaç vardır. Java dili bunu `synchronized` anahtar kelimesi ile gerçekleştirir. Örneğin aşağıdaki methodda ortak kullanılan `sum` adlı değişkenin aynı anda farklı threadler tarafından güncellenmemesi için `updateSum()` adlı method synchronized yapılmıştır.

```
public synchronized void updateSum(int amount)
{
      sum += amount;
}
```

Synchronized bir methodu koşan threade o anlık bir iş düşmüyorsa `wait()` methodunu çağırarak synchronized method üzerindeki kilidi kaldırır ve böylece diğer threadlerin de o methodu koşmasına izin verir. Eğer bir thread işini tamamlamışsa ve `wait()` konumundaki başka bir thread'in çalışmasını sağlamak istiyorsa `notify()` methodunu kullanır. Wait konumundaki bütün threadlerin çalıştırılması için de `notifyall()` methodu kullanılır. Bu durumda hangi thread'e öncelik verileceğine JVM karar verir.

6.1. Producer-Consumer Problemi

Bilindiği gibi producer-consumer probleminde producerın ürettiği ve consumerın tükettiği kaynak ortak kullanılmaktadır. Burada en önemli problem kaynağa eş zamanlı erişimi engelleyerek tutarlılığı sağlamaktır. Producer ve consumerın ortak çağırdıkları methodlar `synchronized` yapılarak tutarlı bir kaynak güncellemesi yapılabilir.

Producer-consumer uygulaması, kaynak kodlardan **Multi Producer-Consumer** adlı klasörün içindedir. Producer ve consumerın ortak erişeceği kaynağı (resourse) üretme ve tüketme işlerini yapan `addOne()` ve `takeOne()` methodları, `Resourse.java` programı içindedir.

`ResourseServer.java` programı öncelikle `item` isimli bir `Resourse` nesnesi türetir ve `Producer` içinde `item.addOne()` çağrısı ile üretmeye başlar.

Herhangi bir Client tarafından Servera bağlantı kurulduğunda `ResourseServer.java` programında `handler` isimli bir `ClientThread` threadi başlatılır ve Clientların isteklerine cevap verilir. Client, Server'a "`1`" karakteri yolladıkça kaynakları tutan `item` nesnesinin `takeOne()` methodu çağrılarak kaynak harcanır. "`0`" karakteri ile de bağlantı sonlandırılır.

`addOne()` methodunda kaynak üretimi belli bir `MAX` (5) değerle sınırlandırılmıştır. Bu değere ulaşıldığında `wait()` methodu çağrılarak Clientların kaynağı tüketmesi beklenir. Herhangi bir kaynak üretildiğinde tüketilebilmesi için `notifyall()` methodu ile Clientlara bilgi verilir. `takeOne()` methodunda da kaynak bittiğinde (0 olunca) `wait()` methodu çağrılarak Serverın kaynak üretmesi beklenir. Herhangi bir kaynak tüketildiğinde üretilebilmesi için `notify()` methodu ile Servera bilgi verilir.

7. Deney Hazırlığı

Bölüm 6.1'de anlatılan **Multi Producer-Consumer** uygulamasında Client bilgisayarda koşacak program **Multithreading** uygulamasındaki `MultiClient.java` programına çok benzediği için ayrıca anlatılmamıştır. `MultiClient.java` programını **Multi Producer-Consumer** klasörüne kopyalayıp ismini `ConsumerClient.java` olarak değiştiriniz ve gerekli değişiklikleri yaparak **Multi Producer-Consumer** uygulamasının doğru bir şekilde çalışmasını sağlayınız. **JAVA** derleyicisi olarak deneyin web sayfasında verilen **JCreator** programını kullanınız.

8. Deney Tasarımı ve Uygulaması

- `Resourse.java` programı içindeki `addOne()` ve `takeOne()` methodlarında gerekli değişikleri yaparak `0..99` arası random bir sayı olarak üretilen kaynak stacke itilirken **"`PUSHED ITEM = ##`"**; tüketilen kaynak stackten çekilirken **"`POPED ITEM = ##`"** şeklinde mesaj yazılmasını sağlayınız (Burada `##`, `0..99` arası bir sayıdır).
- **Stack** adlı klasördeki `StackImplement.java` isimli uygulamada Java dilinde stack veri yapısının nasıl kullanılacağı gösterilmiştir. Stack tanımlayabilmek için programının başında `java.util.*` adlı package import edilmelidir.
- Serverın, Clienta stackten çektiği değeri **"`YOU POPED = ##`"** şeklinde yollaması için programda gerekli değişiklikleri yapınız.
- Stackin dolu veya boş olduğu bilgisini **"`STACK IS FULL/EMPTY`"** şeklinde ekrana yazınız.
- Clienttan gelen isteklere ("`1`" veya "`0`") bağlı olarak `ResourseServer.java` ve `ConsumerClient.java` programlarının ekran çıktısı aşağıdakine benzer olmalıdır:
-

ResourseServer	ConsumerClient
PUSHED ITEM = 42	Enter message ('0' to exit): 1
PUSHED ITEM = 57	SERVER> YOU POPED = 69
PUSHED ITEM = 85	
PUSHED ITEM = 97	Enter message ('0' to exit): 1
PUSHED ITEM = 69	SERVER> YOU POPED = 74
STACK IS FULL	
	Enter message ('0' to exit): 1
New client accepted.	SERVER> YOU POPED = 41
POPED ITEM = 69	Enter message ('0' to exit): 1
PUSHED ITEM = 74	SERVER> YOU POPED = 16
POPED ITEM = 74	
PUSHED ITEM = 41	Enter message ('0' to exit): 1
STACK IS FULL	SERVER> YOU POPED = 97
POPED ITEM = 41	
PUSHED ITEM = 16	Enter message ('0' to exit): 1
POPED ITEM = 16	SERVER> YOU POPED = 85
POPED ITEM = 97	
POPED ITEM = 85	Enter message ('0' to exit): 1
POPED ITEM = 57	SERVER> YOU POPED = 57
PUSHED ITEM = 70	
POPED ITEM = 70	Enter message ('0' to exit): 1
POPED ITEM = 42	SERVER> YOU POPED = 70
STACK IS EMPTY	
PUSHED ITEM = 12	Enter message ('0' to exit): 1
POPED ITEM = 12	SERVER> YOU POPED = 42
PUSHED ITEM = 34	
PUSHED ITEM = 1	Enter message ('0' to exit): 1
PUSHED ITEM = 85	SERVER> YOU POPED = 12
PUSHED ITEM = 34	
PUSHED ITEM = 55	Enter message ('0' to exit): 0
STACK IS FULL	SERVER> Connection closed...
Closing down connection...	Closing connection...

9. Deney Soruları

- `addOne()` methodundaki `notifyall();` satırı kapatılırsa nasıl bir problemle karşılaşılır? `Producer` ve `Consumer` hangi sırada wait durumuna düşer?
- `takeOne()` methodundaki `notify();` satırı kapatılırsa nasıl bir problemle karşılaşılır? `Producer` ve `Consumer` hangi sırada wait durumuna düşer?
- `addOne()` methodundaki `notifyall();` satırı kapatılırsa `Stack`'in hangi durumunda (`EMPTY/FULL`) bile `Producer` ve `Consumer` sorunsuz çalışmaya devam eder?
- `takeOne()` methodundaki `notify();` satırı kapatılırsa `Stack`'in hangi durumunda (`EMPTY/FULL`) bile `Producer` ve `Consumer` sorunsuz çalışmaya devam eder?

10. Deney Raporu

Deney raporunu, kaynak kodların içindeki **`Rapor.docx`** adlı şablon belgeyi kullanarak grup adına yazıp printer çıktısı olarak deneyden sonra bir hafta içerisinde teslim ediniz. Teslim tarihini 2 gün aşan raporlara mevcut rapor notunun %50 'si verilecek, iki günden fazla geç gelen raporlar kabul edilmeyecektir.