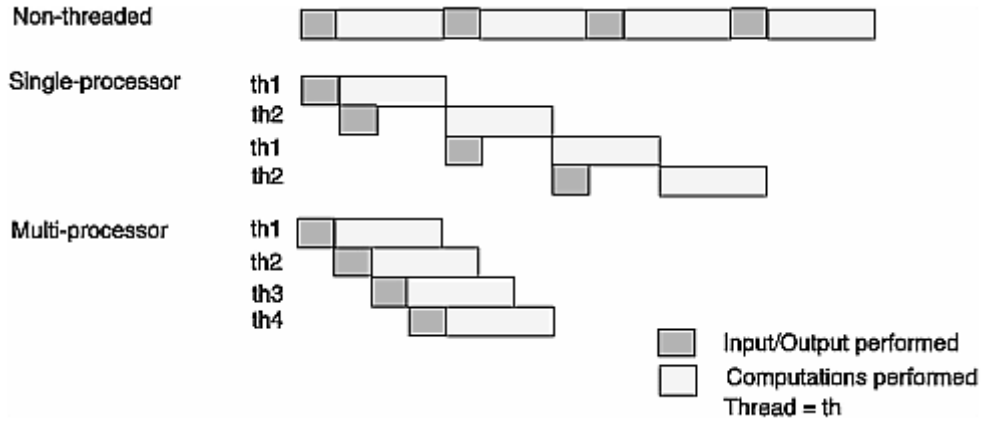
Programlama Dili Prensipleri

Lab Notları – 11

Java'da Thread Mantığı

Bu ders kapsamında thread konusu olarak sadece Java'da thread yapısından bahsedilecektir. C dilinde thread konusu bu ders kapsamında dâhil değildir.

Java'da thread yardımıyla işlemlerin paralel hesaplanması sağlanabilir. Her bir thread bir işlemi yüklenir yürütür ve bitirir. Aşağıdaki resimde eğer thread kullanılmaz ise işlemler arka arkaya dizilir ve işlemci sırayla hepsini işletir. Tek çekirdekli bir işlemcide threadler sahip oldukları işin bir kısmı yapılır daha sonra diğer thread'e geçilir. Çok çekirdekli bir işlemcide threadler aynı anda başlayıp aynı zamanda işlem görebilirler.



Java'da işlemler birer nesnedirler. Java'da işlem oluşturmak için öncelikle o işlemin sınıfını yazmak gerekir. Bu sınıf **Runnable** ara yüzünden kalıtım almak zorundadır. Örneğin ekrana karakter yazan bir sınıf tasarımı aşağıda görülmektedir. Runnable ara yüzünden kalıtlayan her sınıf run metodunu override etmek zorundadır. Bu metot sisteme neyin çalıştırılacağını söyler.

```
package arayuz;

/**
 *
 * @author M.Fatih
 */
public class KarakterIslem implements Runnable {
   private char yazilanKarakter;
   private int kacKere;

   public KarakterIslem(char c,int x){
      yazilanKarakter=c;
      kacKere=x;
   }

   @Override
   public void run() {
```

```
      for(int i=0;i<kacKere;i++)
      {
         System.out.print(yazilanKarakter);
      }
      System.out.println();
   }

}
```

```
package arayuz;

/**
 *
 * @author M.Fatih
 */
public class RakamIslem implements Runnable{
   private int sonSayi;

   public RakamIslem(int sayi){
      sonSayi=sayi;
   }

   @Override
   public void run() {
      for(int i=1;i<=sonSayi;i++){
         System.out.print(" "+i);
      }
      System.out.println();
   }

}
```

```
public static void main(String[] args) {
      // TODO code application logic here

      // İşlemler tanımlanıyor
      Runnable aYaz = new KarakterIslem('a',100);
      Runnable bYaz = new KarakterIslem('b',100);
      Runnable Yaz100 = new RakamIslem(100);

      // Thread oluşturuluyor
      Thread thread1 = new Thread(aYaz);
      Thread thread2 = new Thread(bYaz);
      Thread thread3 = new Thread(Yaz100);

      // Threadler başlatılıyor
      thread1.start();
      thread2.start();
      thread3.start();
   }
```

Thread Sınıfı yield Metodu

yield metodu geçici olarak diğer threadlere zaman verir. Örneğin yukarıdaki sayı yazan sınıf yield ile tekrar düzenlenirse her sayı yazdıktan sonra birkaç karakter yazacaktır.

```
@Override
   public void run() {
      for(int i=1;i<=sonSayi;i++){
         System.out.print(" "+i);
         Thread.yield();
      }
      System.out.println();
   }
```

Thread Sınıfı sleep Metodu:

Belirtilen süre zarfında thread'i uykuya alır. Böylelikle diğer threadler onun yerine çalışmaya başlar. Örneğin yine sayı sınıfı aşağıdaki gibi düzenlenirse

```
@Override
   public void run() {
      try{

         for(int i=1;i<=sonSayi;i++){
            System.out.print(" "+i);
            if(i >= 50)Thread.sleep(1); //1 milisaniye uyu
         }
         System.out.println();
      }
      catch(InterruptedException ex){

      }
   }
```

Thread Havuzu

Bir önceki başlıkta her bir işlem için bir thread oluşturmuştuk fakat devasa bir programda çok fazla sayıda işlemler olacaktır. Her bir işlem için bir thread tanımlamak verimi oldukça düşürecektir. Bunun yerine sayısını bizim belirtebileceğimiz bir thread havuzu oluşturmak daha mantıklıdır.

```
public static void main(String[] args) {

      // Havuzda 3 adet thread oluşturuluyor
      ExecutorService havuz = Executors.newFixedThreadPool(3);

      // işlemler havuza gönderiliyor
      havuz.execute(new KarakterIslem('a',100));
      havuz.execute(new KarakterIslem('b',100));
      havuz.execute(new RakamIslem(100));

      // Havuzu kapat
      havuz.shutdown();
   }
```

Yukarıdaki kod bloğunda havuzdaki thread sayısını 1 yapsaydık o zaman sıralı bir şekilde işlemleri çalıştırmış olacaktık.

Thread Senkronizasyonu

Paylaşılan bir kaynağa farklı thread'ler tarafından aynı anda erişim olursa sistem çökmeleri yaşanabilir. Senkronizasyon içermeyen aşağıdaki örneği inceleyelim.

```
public class Hesap {
   private int toplamPara = 0;
   public int ToplamPara(){
         return toplamPara;
   }
    public void Arttir(int miktar){
         int yeniToplam = toplamPara + miktar;

         try{
            // Veri çöküşünü daha iyi görebilmek için bekleme verildi.
            Thread.sleep(1);
         }
         catch(InterruptedException ex){
         }
         toplamPara = yeniToplam;
   }
}
```

```
public class ParaEkle implements Runnable{
   private Hesap hesap;
   public ParaEkle(Hesap hesap){
      this.hesap = hesap;
   }
   @Override
   public void run() {
      hesap.Arttir(1);
   }
}
```

```
import java.util.concurrent.*;

public class Deneme {
   public static void main(String[] args) {
      Hesap hesap = new Hesap();
      ExecutorService havuz = Executors.newFixedThreadPool(3);

       // İşlemleri oluştur
      for(int i=0;i<100;i++){
         havuz.execute(new ParaEkle(hesap));
      }
      havuz.shutdown();
      // Bütün işlemler bitene kadar bekle
      while(!havuz.isTerminated()){
```

```
      }
      System.out.println("Toplam Miktar:"+hesap.ToplamPara());

   }
}
```

Yukarıdaki programda senkronizasyon ayarlanmadığı için beklenen çıktı elde edilemeyecektir. Program çalıştığında 3 thread oluşturulan 100 işlemi yapmaya çalışıyor ve hepsi aynı işi yani parayı 1 arttırma işini yapıyorlar ve beklenen değer 100 iken ekran çıktısına bakıldığında 100'den farklı ve tahmin edilemeyen değerler üretecektir.

Bu problemi çözmek için önemli olan bölgeyi kritik bölge olarak belirleyip bu bölgeye aynı anda sadece 1 thread'in girmesine izin vermek. Bunun için yukarıdaki kodu aşağıdaki şekle sokuyoruz.

```
public class Hesap {
   private int toplamPara = 0;
   private final Lock bolge = new ReentrantLock();
   public int ToplamPara(){
         return toplamPara;
   }
    public void Arttir(int miktar){
         bolge.lock();  //Kritik bölge başlangıç
         int yeniToplam = toplamPara + miktar;

         try{
            // Veri çöküşünü daha iyi görebilmek için bekleme verildi.
            Thread.sleep(1);
         }
         catch(InterruptedException ex){
         }
         toplamPara = yeniToplam;
         bolge.unlock(); //Kritik bölgeyi kapat.
   }
}
```

Bu düzenleme yapıldıktan sonra çalıştırıldığında her thread o bölgeye tek başına girecek ve değeri bir arttıracak ve sonuçta ekrana yazılan değer 100 olacaktır.

Basit Matris Çarpımının Thread ile Gerçekleştirimi

Aşağıdaki program değerleri rastgele atanmış iki matrisi çarpıp sonucu ekrana basmaktadır. Burada boyut küçük bir değer girildiğinde thread sayısı arttıkça hesaplama süresi artacaktır. Fakat çok büyük boyutlu 1000x1000 gibi matrislerin çarpımlarında belli sayıya kadar threadi arttırmak hesaplama süresini azaltır.

```
public class Matris {
   public int dizi[][];

   public Matris(Random rnd,int boyut){
```

```
        dizi = new int[boyut][boyut];

        for(int i=0;i<boyut;i++){
            for(int j=0;j<boyut;j++){
                dizi[i][j] = rnd.nextInt(10);
            }
        }
    }
    public Matris(int boyut){
        dizi = new int[boyut][boyut];
    }

    @Override
    public String toString() {
        String ekran="";
        for(int satir = 0 ; satir < dizi.length; satir++)
        {
            for (int sutun = 0 ; sutun < dizi.length; sutun++ )
            {
              ekran += "\t" + dizi[satir][sutun];
            }
            ekran += "\n";
         }
        return ekran;
    }

}
```

```
public class Carpma implements Runnable {
    private int satir;
    private int sutun;
    private Matris A;
    private Matris B;
    private Matris Sonuc;

    public Carpma(int satir, int sutun, Matris A, Matris B, Matris sonuc)    {
        this.satir = satir;
        this.sutun = sutun;
        this.A = A;
        this.B = B;
        this.Sonuc = sonuc;
    }

    @Override
    public void run()    {
        for(int i = 0; i < B.dizi.length; i++)       {
          Sonuc.dizi[satir][sutun] += A.dizi[satir][i] * B.dizi[i][sutun];
        }
    }
}
```

```
public class JavaApplication17 {
```

```
    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        int boyut=3;
        Random rand = new Random();

        Matris sonuc = new Matris(boyut);

        Matris A = new Matris(rand, boyut);
        Matris B = new Matris(rand, boyut);


        ExecutorService havuz = Executors.newFixedThreadPool(8);
        long baslangic = System.nanoTime(); //hesaplama başlıyor

        for(int satir = 0 ; satir < boyut; satir++)
        {
          for (int sutun = 0 ; sutun < boyut; sutun++ )         {
              havuz.execute(new Carpma(satir, sutun, A, B, sonuc));
          }
        }
        havuz.shutdown();

        while(!havuz.isTerminated()){  }

        long bitis = System.nanoTime();//hesaplama bitiyor
        double sure = (bitis-baslangic)/1000000.0;

        // A Matrisi yazdırılıyor
        System.out.println(" A Matrix : ");
        System.out.println(A);
         // B Matrisi yazdırılıyor
        System.out.println(" B Matrix : ");
        System.out.println(B);
          // Sonuç Matrisi yazdırılıyor
        System.out.println(" Sonuç : ");
        System.out.println(sonuc);

        System.out.println("\nHesaplanma Süresi " + String.format("%.2f", sure) + " milisaniye.");

    }

}
```

Hazırlayan

Arş. Gör. Dr. M. Fatih ADAK