BM309 – İŞLETİM SİSTEMLERİ

Aralık 2019

ÖDEV – 1

161180067

Müzeyyennur YILGIN

GAZİ ÜNİVERSİTESİ

BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ

1. Wait() ve signal() işlemleri atomik olmazsa karşılıklı dışlama (mutual exclusion) şartının ihlal edilebileceğini gösteriniz.

Wait operation semaforla bağlantılı olan değeri azaltır. Bu azaltma atomik olarak yapılmaktadır. Bu değer bir olduğunda iki wait operation bir semafor üzerinde gerçekleşmiş olursa ve bu iki wait operation işlemi atomik olarak gerçekleştirilemez ise her iki operation da semafor değerini düşürmeye devam eder. Böylelikle karşılıklı dışlanma (mutual exclusion) mümkün olmaktadır. Eğer wait ve signal operation'ları atomik olmaz ise de kilitli olan yerin kilidinin kalkması gibi istenilmeyen durumlar ortaya çıkmaktadır. Çünkü atomik olmayan yapılar kesilebilir yani interrupt'a uğrayabilirler. Bundan dolayı wait ve signal operation'ları eğer atomik olmazsa wait operation'u döngü şeklinde bir bölgeyi kilitlerken bir kesilme yaşayabilir. Bu kesilme sonucunda döngü ortadan kalkar ve kilitli olması gereken kısmın istenilmeyen zamanda kilidi kalkabilir.
Semafor S=1 değerinin ve P1, P2 süreçlerinin aynı anda wait(S) komutunu varsayalım:

T0: P1, S=1'in değerini belirlemektedir

T1: P2, S=1'in değerini belirlemektedir

T2: P1, S'yi 1 azaltıp kritik bölüme girmektedir

T3: P2, S'yi 1 azaltıp kritik bölüme girmektedir

2. Çalışmaya hazır süreç listesinde bir süreç birden çok kez yer alırsa ne olur? Buna izin vermek ne işe yarayabilir? Yorumlayınız.

Hazır süreç listesine bir süreci birden fazla kez eklemek sürecin birden fazla kez sıraya alıp çalışmasına olanak sağlamaktadır. Bu durumda süreç birden fazla kez sırayla çalışır. Böylece daha kısa sürede süreç bitmiş olurken bu durum diğer süreçlerin bekleme sürelerini de etkilemektedir.

3. Çok seviyeli kuyruk kullanan bir sistemde farklı seviyelerde farklı zaman quantumu kullanmak neden avantajlı olabilir? Açıklayınız.

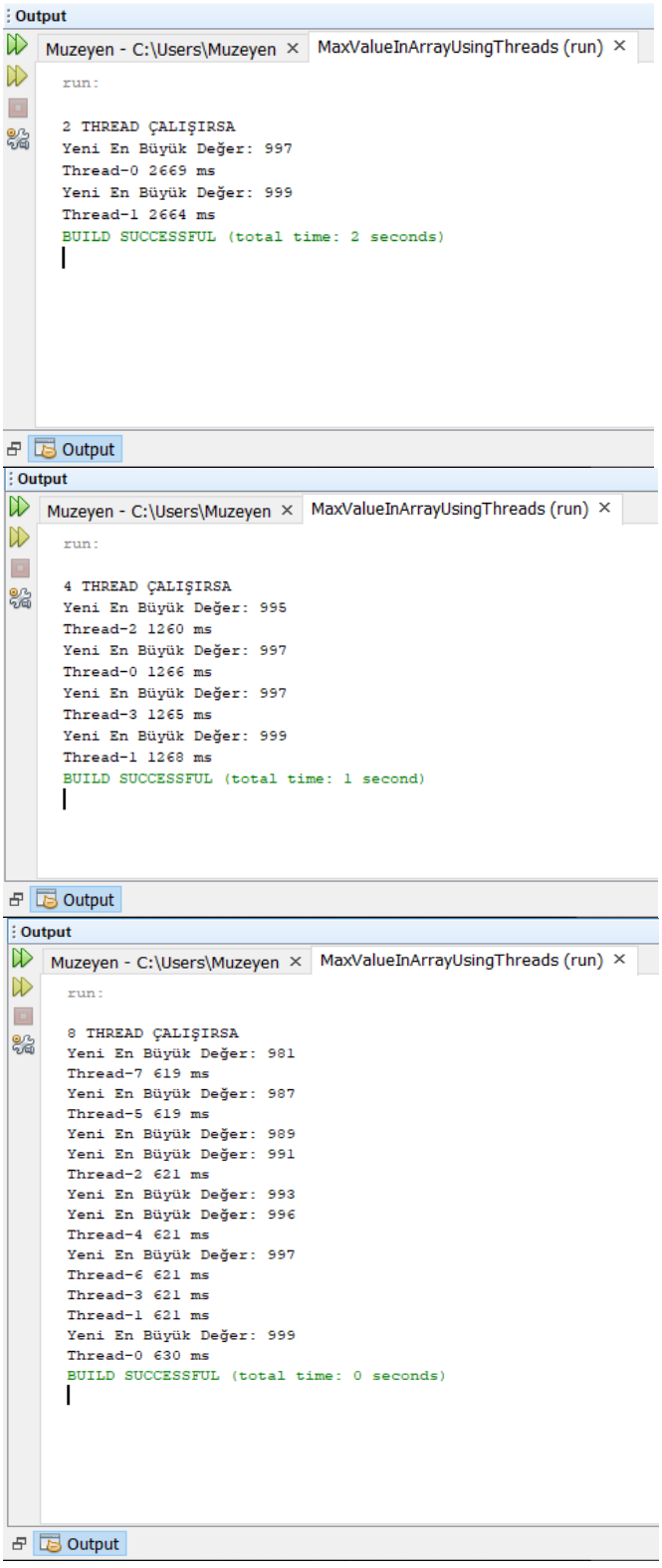
Bazı süreçler sık servis yapma ihtiyacı duyarken bazı servisler daha az servis yapmaya ihtiyaç duymaktadır. Bu sebeple farklı seviyelerde zaman quantumu kullanmak avantajlı olabilmektedir. Sık servis gerektiren process'ler küçük zaman quantumu ile sıraya girebilirken sık servis kullanmayan process'ler daha büyük quantum ile sıraya girebilir. Bu şekilde sık servis yapmaya gerek duymayan process'ler daha az bağlam anahtarı gerektirirler böylece bilgisayar daha verimli kullanılır.

4. Aşağıdaki iki problemi (a ve b) çözen çok iş parçalı (multithreaded) Java programı geliştiriniz. Programınız thread sayısını kullanıcıdan alabilir veya kod içerisinde bir sabitte tutabilir.

a. Verilen bir tamsayı dizisindeki en büyük değeri bulmak

```
1
2   package maxvalueinarrayusingthreads;
3   import java.util.Random;
4
5   class MaxThread extends Thread
6   {
7           int ilkIndex, sonIndex;
8           int[] dizi;
9           public MaxThread(int[] array,int startIndex, int finishIndex)
10          {
11              dizi = array;
12              ilkIndex = startIndex;
13              sonIndex = finishIndex;
14          }
15
16          @Override
17          public void run()
18          {
19              long start  = System.currentTimeMillis();
20              enBuyukDeger();
21              System.out.println("Yeni En Büyük Değer: "+ enBuyukDeger());
22              long finish = System.currentTimeMillis();
23              System.out.println(this.getName() + " " + (finish-start) + " ms"); //thread ismi ve ne kadar sürede işini yaptığı yazılır
24          }
25          public int enBuyukDeger()
26          {
27              int max = dizi[0];
28              for (int i = ilkIndex; i < sonIndex; i++) {
29                  if(dizi[i]>max)
30                      max = dizi[i];
31                  try {
32                      MaxThread.sleep(2);
33                  }
34                  catch (InterruptedException e) {
35                  }
36              }
37              return max;
38          }
39  }
40
```

```
41  public class MaxValueInArrayUsingThreads {
42
43      public static void main(String[] args)
44      {
45          int[] randomDizi = new int[1000];
46           Random random = new Random();
47           for (int i = 0; i < randomDizi.length; i++)
48           {
49               randomDizi[i] = random.nextInt(1000); //dizinin içerisinde maksimum 1000 olabilecek rastgele sayılar ürettik
50           }
51           //En büyük sayıyı bulmak için diziyi parçalara bölüp thread'lere aratarsak
52
53           //4 thread çalışırsa
54           MaxThread fourThread1 = new MaxThread(randomDizi, 0, 250);
55           MaxThread fourThread2 = new MaxThread(randomDizi, 250, 500);
56           MaxThread fourThread3 = new MaxThread(randomDizi, 500, 750);
57           MaxThread fourThread4 = new MaxThread(randomDizi, 750, 1000);
58           System.out.println("");
59           System.out.println("4 THREAD ÇALIŞIRSA");
60           fourThread1.start();
61           fourThread2.start();
62           fourThread3.start();
63           fourThread4.start();
64
65      }
66  }
```



Burada en büyük değeri kendi parçalarında aramaktadırlar. En büyük değeri bulduklarında yeni en büyük değer ile değiştirmektedirler. Her bir threadin ne kadar çalıştığı görünmektedir.

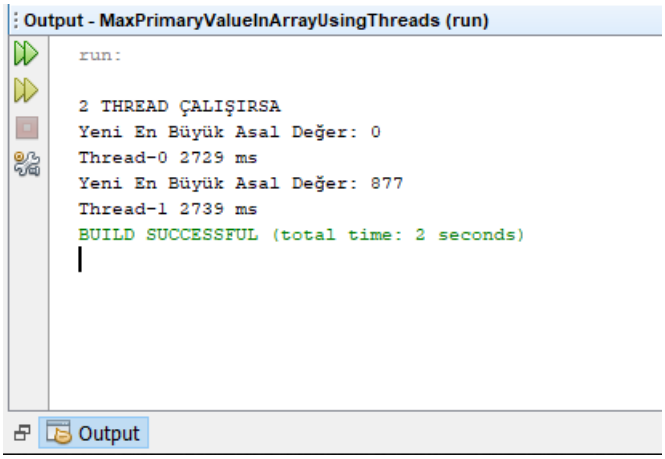
b. Verilen bir tamsayı dizisindeki en büyük asal sayıyı bulmak

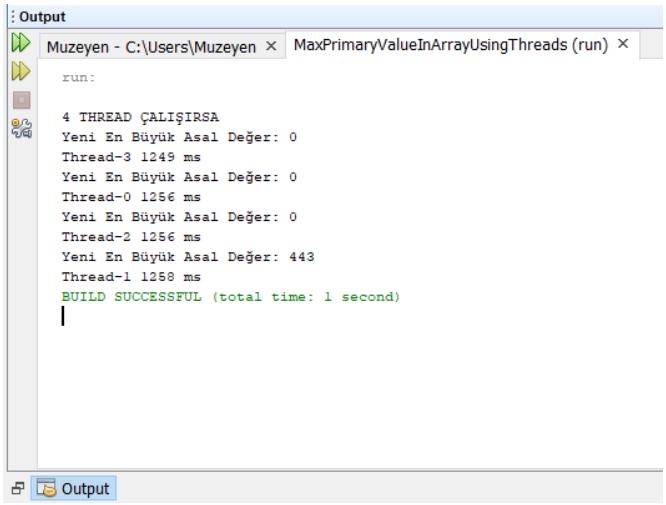
```
class MaxThread extends Thread {
        int ilkIndex, sonIndex;
        int[] dizi;
        public MaxThread(int[] array,int startIndex, int finishIndex)
        {
            dizi = array;
            ilkIndex = startIndex;
            sonIndex = finishIndex;
        }
        @Override
        public void run()
        {
            long start  = System.currentTimeMillis();
            enBuyukAsalDeger();
            System.out.println("Yeni En Büyük Asal Değer: "+ enBuyukAsalDeger());
            long finish = System.currentTimeMillis();
            System.out.println(this.getName() + " " + (finish-start) + " ms"); //thread ismi ve ne kadar sürede işini yaptığı yazılır
        }
        public int enBuyukAsalDeger()
        {
            int max = 0;
            boolean asal=true;
            for (int i = ilkIndex; i < sonIndex; i++) {
                if (dizi[i] <= 1)
                    i++;
                else{
                    for (int j = 2; j < dizi[i]; j++) {
                        int kalan = dizi[i] % j;
                        if (kalan == 0){
                            asal=false;
                            break;
                        }
                    }
                    if(asal==true){
                       if(max<dizi[i])
                          max=dizi[i];
                    }try {
                    MaxThread.sleep(2);
                    }
                    catch (InterruptedException e) {
                    }
                }
            }
            return max;
        }
    }
```

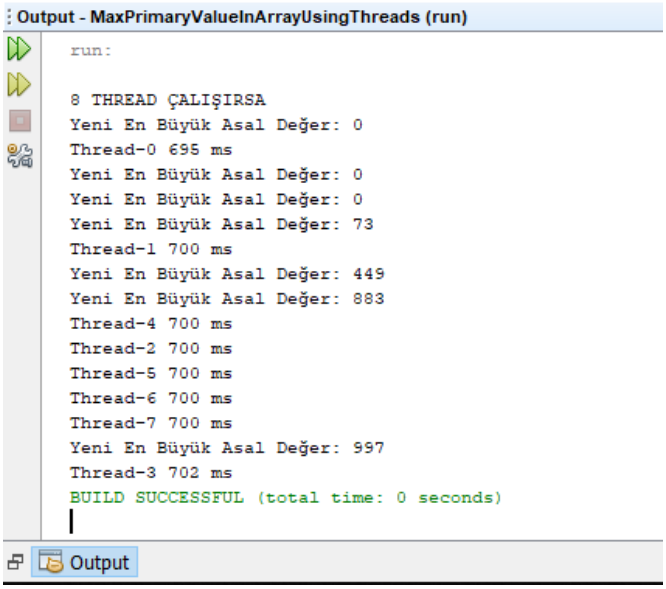
```
}
public class MaxPrimaryValueInArrayUsingThreads {

    public static void main(String[] args) {
        int[] randomDizi = new int[1000];
        Random random = new Random();
        for (int i = 0; i < randomDizi.length; i++)
        {
            randomDizi[i] = random.nextInt(1000); //dizinin içerisinde maksimum 1000 olabilecek rastgele sayılar ürettik
        }
        //En büyük sayıyı bulmak için diziyi parçalara bölüp thread'lere aratarsak

         //4 thread çalışırsa
        MaxThread fourThread1 = new MaxThread(randomDizi, 0, 250);
        MaxThread fourThread2 = new MaxThread(randomDizi, 250, 500);
        MaxThread fourThread3 = new MaxThread(randomDizi, 500, 750);
        MaxThread fourThread4 = new MaxThread(randomDizi, 750, 1000);
        System.out.println("");
        System.out.println("4 THREAD ÇALIŞIRSA");
        fourThread1.start();
        fourThread2.start();
        fourThread3.start();
        fourThread4.start();
    }
}
```



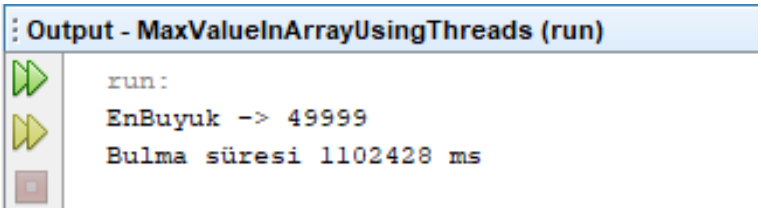




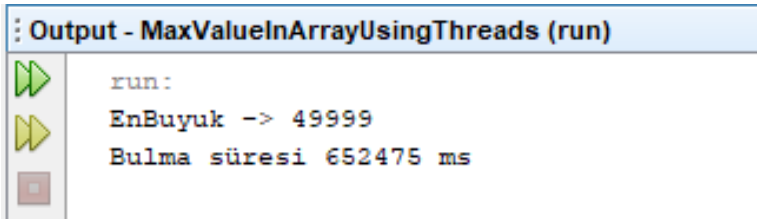
Burada thread'ler kendi kısımlarındaki en büyük asal sayıyı aramaktadırlar. Dizi içerisinde büyük asal sayı bulunduğunda max değerine atanmaktadır.

c. **En az 1 milyon sayı içeren bir dizi için programınızın 2, 4 ve 8 thread ile a ve b işlerini ne kadar sürede tamamladığını ekran görüntüsüyle gösteriniz.**

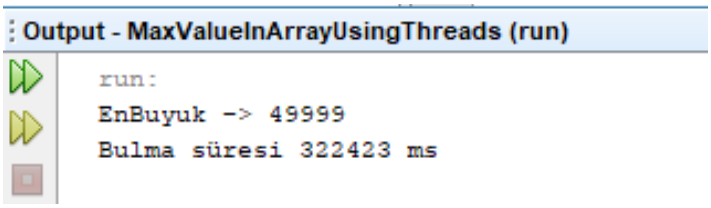
Thread sayısı ne kadar fazla olursa dizi o kadar fazla parçacığa bölünüp arandığı için o kadar kısa sürede bulunmaktadır. Thread sayısının artması performansı arttırır. Thread sayısı arttıkça bilgisayar daha verimli çalışır.



2 thread ile en büyük değer bulma



4 thread ile en büyük değer bulma



8 thread ile en değer bulma