A red text on a white background

Description automatically generated

MÜHENDİSLİK VE DOĞA BİLİMLERİ FAKÜLTESİ

İŞLETİM SİSTEMLERİ ÖDEV-2

HAZIRLAYAN

MERT TOSUN

210601027

BİLGİSAYAR MÜHENDİSLİĞİ

TESLİM TARİHİ

21.04.2024

DERSİN EĞİTMENİ

Assist. Prof. Dr. Kemal Çağrı Serdaroğlu

Ödevin Amacı

Bu ödevin amacı, thread kavramını anlamak ve uygulamaktır. Aynı zamanda, birden fazla iş parçacığının senkronizasyonunu sağlayarak paylaşılan bir kaynağa erişimdeki sorunları çözmek için senkronizasyon tekniklerini öğrenmektir. Ödevde thread kullanarak bir matris üzerinde çeşitli işlemler yapılmaktadır. Bu işlemler arasında negatif sayı tespiti, negatif ve pozitif sayıların sayısını bulma, en büyük ve en küçük sayıyı bulma gibi işlemler bulunmaktadır. Ayrıca, tek bir thread kullanılarak aynı işlemlerin nasıl gerçekleştirileceği de incelenmektedir. Bu şekilde, threadler ve senkronizasyon konuları üzerinde denemeler yaparak daha iyi anlamaya ve deneyim kazanmaya çalışmaktayız.

RACE CONDİTİON NEDİR ?

Race condition, thread veya süreçlerin aynı anda paylaşılan bir kaynağa erişmeye çalıştığında ortaya çıkan bir senkronizasyon hatasıdır. Threadler arasında race durumu oluştuğunda, programın davranışı öngörülemez hale gelir ve istenmeyen sonuçlar meydana gelebilir.

Yazdığımız kodlardan örnek vermek gerekirse SayiSaymaThread sınıfındaki negatif sayı sayısı, pozitif sayı sayısı ve sıfır sayısı değişkenleri aynı dizideki elemanları işleyen farklı threadler tarafından eş zamanlı olarak güncelleniyor. Bu durumda, bir race condition oluşabilir. Örneğin, iki iş parçacığı aynı anda negatif sayı sayısını artırmaya çalışırsa, sonuçlar hatalı olabilir.

CRİTİCAL SECTİON NEDİR ?

Critical section'lar genellikle senkronizasyon mekanizmaları (örneğin semafor veya lock) kullanılarak korunur. Bu mekanizmalar, yalnızca bir threadin critical section'a girip işi bitirip çıkmasını sağlayarak, diğer threadlerin sırayla kaynağa erişimini sağlar. Bu şekilde, veri bütünlüğü ve programın tutarlılığı korunmuş olur.

LOCK NEDİR ?

Lock kavramı, thread ortamlarında paylaşılan kaynaklara erişimi kontrol etmek için kullanılan bir senkronizasyon mekanizmasıdır. Özellikle aynı anda birden fazla thread’in aynı kaynağa erişimini engellemek veya düzenlemek için kullanılır. Kodlardaki senaryoları ele alarak lock kavramını açıklamam gerekirse:

1-) Negatif Sayı Bulma (threadA): Örneğin, bu senaryoda her bir satır için bir thread oluşturulur ve bu iş threadler matrisin her bir satırını tarar, negatif bir sayı bulduğunda, diğer threadlerin aynı satıra erişimini engellemek için bir lock(kilit) mekanizması kullanılabilir. Böylece, bir thread negatif bir sayıyı bulduğunda, diğer threadler o satıra erişim yapamazlar, bu da veri bütünlüğünü ve doğruluğunu sağlar.

2-) Negatif ve Pozitif Sayı Sayısı Bulma (threadB): Bu senaryoda da her bir satır için thread oluşturulur ve bu threadler aynı anda matrisin farklı satırlarında çalışır. Ancak, her bir thread negatif veya pozitif sayı bulduğunda, bu bilgiyi paylaşılan bir değişkene yazarken lock mekanizması kullanılabilir. Böylece, aynı anda bu değişkene erişmeye çalışan threadlerin arasında çakışmalar önlenir ve doğru sayım yapılır.

3-) En Büyük ve En Küçük Sayıyı Bulma (threadC): Bu senaryoda da her bir satır için threadler oluşturulur ve her bir thread matrisin farklı satırlarını tarar. En büyük ve en küçük sayıları bulmak için her thread kendi satırındaki değerleri karşılaştırır. Bu işlemlerde, en büyük ve en küçük sayıları güncellemek için bir kilit mekanizması kullanılabilir, böylece aynı anda bu değerleri değiştirmeye çalışan threadler arasında çakışmalar engellenir.

Lock kavramı, paylaşılan kaynaklara eş zamanlı erişimi kontrol etmek için kullanılan bir araç olarak önemlidir. Lock mekanizması, veri bütünlüğünü sağlar ve race condition gibi sorunların önüne geçer, böylece paralel işlemler daha güvenli hale gelir.

System.nanoTime() Fonksiyonu :

Java'da sistem saati çözünürlüğüne bağlı olarak nanosaniye cinsinden bir zaman döndüren bir metottur. Ölçüm yapmak istediğimiz kısımda başlangıç ve bitiş olarak tanımlayarak işlemin süresini ölçeriz.

A MADDESİNDE İSTENEN YAPI : Matriste negatif sayı bulunursa ekrana yazdırılıp program sonlandırılacak.

Lock Kullanarak :

metin, ekran görüntüsü, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

Lock Kullanmadan :

metin, ekran görüntüsü, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

A maddesinde istenen yapıları oluşturduğumuz zaman Lock kullanarak oluşturduğumuz yapıda java.util.concurrent.locks.Lock, java.util.concurrent.locks.ReentrantLock kütüphanelerini import ederek kullandığımız lock yapısı ile kritik bölgede işlemler sürerken erişim engellenir. İşlemler sona erdiğinde unlock yapısı kullanılarak lock serbest bırakılır ve erişim açılır. Bu sayede race condition durumunun oluşması engellenmiş olur.

Lock kullanmadan bu işlem yapıldığında ise synchronized anahtar kelimesiyle belirli bir kod bloğu veya metot aynı anda sadece bir thread tarafından çalıştırılabilir. Bu sayede race conditionu lock kullanmadan engellemiş oluruz.

LOCK KULLANARAK VE LOCK KULLANMADAN KARŞILAŞTIRMASI:

10X10 matriste LOCK kullanarak test: 10X10 matriste LOCK kullanmadan test:

metin, yazı tipi, ekran görüntüsü, çizgi içeren bir resim

Açıklama otomatik olarak oluşturuldu 

3X3 matriste LOCK kullanarak test: 3X3 matriste LOCK kullanmadan test:

1000X1000 matriste LOCK kullanarak test: 1000X1000 matriste LOCK kullanmadan test:

 metin, yazı tipi, ekran görüntüsü, çizgi içeren bir resim

Açıklama otomatik olarak oluşturuldu

500X500 matriste LOCK kullanarak test: 500X500 matriste LOCK kullanmadan test:

Sonuçları yorumlamak istersek, hangi senkronizasyon mekanizmasının kullanılacağı, uygulamanın gereksinimlerine, kod karmaşıklığına ve performans beklentilerine bağlıdır. Her iki mekanizma da doğru senaryoda etkili olabilir ve performans açısından belirgin bir fark olmayabilir. Ancak, özellikle karmaşık senaryolar ve yüksek performans gereksinimleri söz konusu olduğunda, Lock daha fazla esneklik ve kontrol sağlayabilir. Karşılaştırmaları dikkatlice incelersek genel olarak Lock kullanılanların performans bakımından daha iyi olduğunu görebiliriz.

B MADDESİNDE İSTENEN YAPI:

Matristeki toplam pozitif, negatif ve sıfır sayıları belirlenecek.

Lock kullanarak : Lock kullanmadan :

metin, ekran görüntüsü, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu metin, ekran görüntüsü, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

2X2 LOCK kullanarak test: 2X2 LOCK kullanmadan test:

metin, yazı tipi, ekran görüntüsü, tipografi içeren bir resim

Açıklama otomatik olarak oluşturuldu metin, yazı tipi, ekran görüntüsü, tipografi içeren bir resim

Açıklama otomatik olarak oluşturuldu

50X50 LOCK kullanarak test: 50X50 LOCK kullanmadan test:

metin, yazı tipi, ekran görüntüsü, tipografi içeren bir resim

Açıklama otomatik olarak oluşturuldu metin, yazı tipi, ekran görüntüsü, tipografi içeren bir resim

Açıklama otomatik olarak oluşturuldu

1000X1000 LOCK kullanarak test: 1000X1000 LOCK kullanmadan test:

metin, yazı tipi, ekran görüntüsü, tipografi içeren bir resim

Açıklama otomatik olarak oluşturuldu metin, yazı tipi, ekran görüntüsü, tipografi içeren bir resim

Açıklama otomatik olarak oluşturuldu

15000X15000 LOCK kullanarak test: 1500X1500 LOCK kullanmadan test:

metin, yazı tipi, ekran görüntüsü, tipografi içeren bir resim

Açıklama otomatik olarak oluşturuldu metin, yazı tipi, ekran görüntüsü, tipografi içeren bir resim

Açıklama otomatik olarak oluşturuldu

Bu kısımdaki testleri incelediğimizde ise yine aynı şekilde farklı performans sonuçlarına ulaşmış olduk lock kullanarak yaptığımız testlerde genel olarak bakarsak az farkla da olsa daha iyi sonuçlar edindiğimizi görüyoruz. Lock kullanmadan senkronizasyon sağladığımızda genel olarak locktan biraz daha yüksek veya yakın değerlerde sonuçlarla karşılaştık.

LOCK YAPISI VEYA DİĞER SENKRONİZASYON METOTLARINI KULLANMAZSAK;

Matristeki her satırın içindeki sayıların pozitif, negatif ve sıfır sayılarını ayrı ayrı sayan bir programımız var.

Eğer race condition oluşmazsa:

İki veya daha fazla thread aynı anda aynı değişkenlere erişemezdi. Bu durumda, her bir thread sırayla işlem yapar ve bir thread'in işlemi bitmeden diğerine geçilmez.

Her thread, sırasıyla satırları işler ve kendi hesaplamalarını yapar. Örneğin, bir thread bir satırı işlemeye başlar, o satırdaki sayıları tarar ve pozitif, negatif ve sıfır sayıları için ilgili değişkenleri artırır. Diğer thread ise aynı anda başka bir satırı işler.

Eğer race condition olmazsa, her threadin hesapladığı sonuçlar doğru olurdu, çünkü her bir thread sırasıyla işlem yapar ve birbirlerinin hesaplamalarını etkilemezler.

Race condition oluşursa ne olur :

Race condition, aynı anda birden fazla thread'in aynı değişkenlere erişmeye çalışması durumunda karmaşık sonuçlar elde edilmesine neden olabilir.

Örneğin, iki thread aynı anda aynı değişkeni artırmaya çalışırsa, sonuçlar beklenmedik olabilir. Bir thread diğerinin artırma işlemini göz ardı edebilir veya ikisinin de işlemi aynı anda gerçekleşirse, beklenmedik sonuçlar ortaya çıkabilir.

Bu durum, verilerin güncellenmesi sırasında tutarlılık sağlanmasını engelleyebilir ve programın istenmeyen şekilde davranmasına neden olabilir.

C MADDESİNDE İSTENEN YAPI:

Matrisin içinde bulunan en büyük ve en küçük sayı tespit edilecek.

5X5lik LOCK kullanarak: 5X5lik LOCK kullanmadan:

metin, yazı tipi, ekran görüntüsü, tipografi içeren bir resim

Açıklama otomatik olarak oluşturuldu metin, yazı tipi, ekran görüntüsü, tipografi içeren bir resim

Açıklama otomatik olarak oluşturuldu

100X100lük LOCK kullanarak: 100X100lük LOCK kullanmadan:

metin, yazı tipi, ekran görüntüsü, tipografi içeren bir resim

Açıklama otomatik olarak oluşturuldu metin, yazı tipi, ekran görüntüsü, tipografi içeren bir resim

Açıklama otomatik olarak oluşturuldu

8000X8000 LOCK kullanarak: 8000X8000 LOCK kullanmadan:

metin, yazı tipi, ekran görüntüsü, tipografi içeren bir resim

Açıklama otomatik olarak oluşturuldu metin, yazı tipi, ekran görüntüsü, sayı, numara içeren bir resim

Açıklama otomatik olarak oluşturuldu

20000X20000 LOCK kullanarak: 20000X20000 LOCK kullanmadan:

metin, yazı tipi, ekran görüntüsü, tipografi içeren bir resim

Açıklama otomatik olarak oluşturuldu metin, yazı tipi, ekran görüntüsü, tipografi içeren bir resim

Açıklama otomatik olarak oluşturuldu

Bu maddeyi incelediğimiz zaman ise matriste sayı aralığımız -5 ile 5 arasında olduğu için NXN değeri yüksek olan matrislerin en büyük değeri 5 , en küçük değeri -5 olarak geliyor ancak N değerini düşük tuttuğumuzda rastgele gelen sayılarda en küçük ve en büyük -5 ve 5’den farklı çıkabiliyor. Süre karşılaştırması yaparsak lock kullandığımız programımızda daha iyi performans sağladık. Ayrıca iki programıda da race condition durumuna dikkat ederek yazdım.

TEK THREAD KULLANARAK A-B VE C MADDELERİNİN YAPILMASI VE ÖRNEKLERİ:

Tek bir thread kullanarak yapılan işlemler genellikle sıralı ve belirli bir düzen içinde gerçekleşir. Bu durumda, programın her adımı sırasıyla işlenir ve bir adım tamamlanmadan diğerine geçilmez. Bu durum, programın daha öngörülebilir ve kontrol edilebilir olmasını sağlar. Ancak, tek thread kullanımı genellikle çoklu işlemcili veya paralel işlem yapılmasını gerektiren durumlarda yetersiz kalabilir ve işlemci kaynaklarının etkin kullanımını engelleyebilir. Bizde tek thread kullanarak a, b ve c deki işlemleri yaptık şimdi bu işlemlerin performans sonuçlarına bakalım ve diğer lock kullanarak yaptığımız işlemlerle karşılaştıralım.

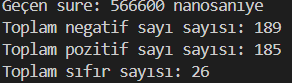
4X4 TEK THREAD KULLANARAK: 4X4 LOCK KULLANARAK: (A İŞLEMİ)

4X4 LOCK KULLANMADAN:

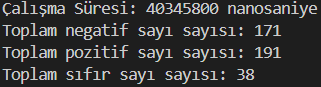


20X20 TEK THREAD KULLANARAK: 20X20 LOCK KULLANARAK: (B İŞLEMİ)

 metin, yazı tipi, ekran görüntüsü, tipografi içeren bir resim

Açıklama otomatik olarak oluşturuldu

20X20 LOCK KULLANMADAN:



8000X8000 TEK THREAD KULLANARAK: 8000X8000 LOCK KULLANARAK: (C İŞLEMİ

metin, yazı tipi, ekran görüntüsü, tipografi içeren bir resim

Açıklama otomatik olarak oluşturuldu metin, yazı tipi, ekran görüntüsü, tipografi içeren bir resim

Açıklama otomatik olarak oluşturuldu

8000X8000 LOCK KULLANMADAN:

metin, yazı tipi, ekran görüntüsü, tipografi içeren bir resim

Açıklama otomatik olarak oluşturuldu

Genel olarak, tek thread kullanılarak yapılan işlemlerde senkronizasyon yapmayla uğraşmaması, race condition riskinin olmaması ve daha az işlemci kaynağı kullanımı gibi faktörler, işlemlerin daha hızlı ve daha verimli bir şekilde gerçekleştirilmesini sağlayabilir. Bu nedenle, belirli senaryolarda tek thread kullanımı tercih edilebilir. Bizim yaptığımız işlemlerde de olduğu gibi tek thread kullanımı avantaj sağlayabilir.

A İŞLEMİ AKIŞ ŞEMASI :

satir[i] < 0

Evet

İ=0 İ<satir.length İ++

Negatif bulunur

HAYIR

Negatif bulunmaz

B İŞLEMİ AKIŞ ŞEMASI:

i = 0; i < satir.length; i++

toplam Negatif = 0; toplam Pozitif = 0; toplam Negatif = 0; toplamNegatif = 0;

        int toplamPozitif = 0;

        int sifirsayisi = 0;

int toplamPozitif = 0;

int sifirsayisi = 0;toplamNegatif = 0;

int toplamPozitif = 0;

int sifirsayisi = 0;

sifirsayisi = 0; int toplamNegatif = 0;

int toplamPozitif = 0;

int sifirsayisi = 0;

Toplam negatif , pozitif , sifir sayisi

Sifirsayisi++

Pozitifsayisi++

Negatifsayisi++

satir[i]=0

satir[i] < 0

satir[i] > 0

C İŞLEMİ AKIŞ ŞEMASI:

i = 0; i < satir.length; i++

Enbuyuk=maxvalue

Enkucuk=minvalue

En büyük sayı

En küçük sayı

enKucuk = satir[i]

enBuyuk = satir[i]

satir[i] < enKucuk

satir[i] > enBuyuk

Bu kısımda gerçekleştirdiğimiz işlemlerin akış şemalarına yer verdik.

Projemizde kısaca özet geçmek gerekirse bu projenin amacı, işletim sistemlerinin temel prensiplerini anlamak ve senkronizasyon tekniklerini öğrenmekti. Projede race condition gibi konuları ele aldım ve çözümler geliştirdim. Ayrıca, lock mekanizmaları ve tek thread kullanımının avantajlarını ve dezavantajlarını gözlemledim. Kodun yazım aşamasında yorum satırlarıyla sık sık ne yaptığım hakkında bilgiler verdim.

NOT: Ben kendi derleyicimde kodu çalıştırdığımda bazı yerlerde fail verdi fakat continue dediğim zaman sorunsuz çalıştı sanırım sorun benim kullandığım derleyici ile alakalı.

Ödevde ayrıca volatile olarak geçen sözcük java anahtar sözcüğüdür ve bir değişkeni işaret eder. Bu anahtar sözcüğü bir değişkenin değerinin birden fazla thread arasında tutarlı bir şekilde paylaşılmasını sağlar. Java'da her bir thread, kendi bellek önbelleğinde bir kopya değere sahiptir. Bu, bir thread'in değişkenin değerini güncellediğinde, bu güncelleme diğer thread'ler tarafından görülebilir olmayabilir. volatile anahtar sözcüğü, bir değişkenin değerini ana belleğe yazar ve tüm thread'lerin bu ana bellekten okumasını zorunlu kılar. Bu şekilde, bir thread'in bir değişkenin son güncel değerini görmesi sağlanır.

YARARLANDIĞIM KAYNAKLAR

<https://codingbytime.com/>

<https://www.borakasmer.com/>