SUNUCU İSTEK YOĞUNLUĞUNUN MULTITHREAD İLE KONTROLÜ

\*Yaşar Can Çilingir, \*Sergen Azizoğlu

\*Bilgisayar Mühendisliği Bölümü, Kocaeli Üniversitesi

Kocaeli, Türkiye

yasarcanccd@gmail.com

sergenazizoglu@gmail.com

**Anahtar Kelimeler – “Thread”, “Sunucu”, “Yoğunluk”, ”İstek”, ”Senkronizasyon”**

***Özet* – “Sunucu İstek Yoğunluğunun Multithread İle Kontrolü” Yazılım Laboratuvarı I dersi kapsamında verilen, Java dilinde yazılmış,Thread kütüphanesiyle senkronize edilmiş ve Swing grafik arayüz kütüphanesiyle tasarlanmış bir sunucu istek kontrol uygulamasıdır. Rastgele üretilen değerlerle sunucuda istek ve cevap simüle edilmiş, arayüz üzerinde progress barlar ile anlık çalışan threadler ve kapasite yüzdeleri gösterilmiştir.**

1.GİRİŞ

Program bir sunucu sistemini simüle etmektedir. Sistem başlangıçta 1 ana sunucu ve 2 alt sunucudan oluşmaktadır. Gelen isteğin ana sunucu ve alt sunucularda senkronize olarak paylaştırılması ve yine senkronize olarak isteklerin yanıtlanması amaçlanmıştır. Alt sunucuların yoğunluğu %70’i aştığı durumlarda, dinamik olarak yeni alt sunucuların üretilmesi ve isteklerin yeni sunuculara da aktarılması sağlanmıştır.

Program başlangıcında istek değeri, random olarak üretilmektedir. Bu üretilen istek programın başlangıcında çalıştırılmış olan 1 ana ve 2 alt sunucuya senkronize olarak paylaştırılmaktadır. Ana ve alt sunucular paralel olarak çalışmaktadır. Ana sunucunun kapasitesi 10000, alt sunucuların kapasiteleri 500 olarak atanmıştır. Ana sunucu 500 ms aralıklarla 1 - 400 arasında istek kabul etmektedir. 200 ms aralıklarla da 1-50 arasında isteğe karşılık vermektedir. Alt sunucular 500 ms aralıklarla 1 – 150 arasında istek kabul etmektedir. 300 ms aralıklarla 1 - 40 arasında isteğe karşılık vermektedir. Gerektiği durumlarda istek yoğunluğuna bağlı olarak yeni alt sunucular üretilmektedir.

Gerekli şartlar sağlandığında bu programda kullanılan çözüm yöntemleri rahatlıkla gerçek sunucu sistemlerinde uygulanabilir ve yoğunluk problemi çözümünü sağlar.

2.TEMEL BİLGİLER

Programımız Java dilinde yazılmıştır. Geliştirme ortamı olarak “NetBeans 8.2” kullanılmıştır. Windows 10 işletim sistemlerinde sorunsuz olarak çalıştırılmıştır.

Java programlama dilinin thread kütüphanesi ve swing grafik arayüz kütüphanesi kullanılmıştır.

3.YÖNTEM

Thread kütüphanesi kullanabilmek için Runnable interface’i implement edilmiştir. Her sunucu için 2 thread kullanılmıştır. Java ile senkronizasyon için synchronized, wait ve notify yapıları kullanılmıştır.

Threadlerin senkronizasyonu ve aynı değişkene aynı anda ulaşmalarını engellemek için Threadler ile değişkenler arasında anahtar kilit ilişkisi oluşturulmuştur. İlgili thread değişkene erişebilmek için anahtara sahip olmak zorundadır. Anahtarlandırma işlemi JAVA’nın synchronized yapısı ile sağlanmıştır. Anahtarları threadler arasında düzgün bir şekilde paylaştırma işlemi ise Wait() ve Notify() yapıları ile sağlanmıştır.

4.YAPILAN ARAŞTIRMALAR

JAVA programlama dilinde thread yazımı hakkında, runnable interface’i hakkında bilgiler araştırılmıştır. Threadlerin aynı değişkeni senkronize olarak kullanması ve gerektiğinde threadlerin kill edilmesi hakkında bilgiler araştırılmıştır.

Swing arayüz kütüphanesinde progress bar listesi oluşturma ve değerlerini yönetme konusunda bilgiler araştırılmıştır.

5.KULLANILAN FONKSİYONLAR

*void produce():* Main server için yeni istek üretilmesini, sub serverler için main serverin isteklerini almayı sağlayan fonksiyondur.

*void consume*(): Main server ve sub serverlerin isteklere belirli saniyeler aralıklar ile yanıt vermesini sağlayan fonksiyondur.

*void run():* Threadlerin çalışmasını sağlayan fonksiyondur.

*void control():* Sub serverlerin yoğunluğunu anlık olarak kontrol eden ve %70’i geçtiğinde yeni subserver oluşturulmasını sağlayan fonksiyondur.

void baslangic(): Serverleri arayüzde başlangıç konumlarına yerleştiren fonksiyondur.

6.SONUÇ

Threadler ile sunucu yapısı simüle edilmiştir. Threadlerin senkronizasyonu kullanılarak sunucuların paralel, senkronize ve eş zamanlı çalışması doğru olarak sağlanmıştır. Yoğunluğa bağlı olarak alt sunucuların dinamik olarak üretilebilmesi ve sonlandırılabilmesi başarıyla sağlanmıştır.

7. KABA KOD

1. Başla
2. MainProducer, MainConsumer threadlerini oluştur.
3. Sabit SubProducer ve sabit SubConsumer threadlerini oluştur.
4. Control threadi ve sunucu takip (arayüz) threadlerini oluştur.
5. MainProducer threadi ile random sayıda istek oluştur.
6. Aynı anda SubProducer ile alt serverlara oluşturulan isteği random sayıda aktar.
7. MainConsumer ve SubConsumerlar’ın eş zamanlı olarak isteklere random sayıda yanıt vermesini sağlar.
8. Sunucu takip (arayüz) threadi, thread sayılarını arayüze yansımasını sağlar.
9. Control threadi sub serverlerin doluluk oranlarını kontrol eder.
10. if(subserver doluluk oranı >= %70)
11. Subserverin yoğunluğunu bölerek yeni subserver oluşturur ve bölünen yoğunluğu yeni subservere aktarır.
12. if(subserver doluluk oranı ==0 && subserver sayısı >=2)
13. Subserveri silme ve threadlerini öldürme işlemini gerçekleştir.
14. Arayüzde progress barlar ile server yoğunluklarını ve yüzdelerini anlık olarak göster.
15. Mainserver ve ilk iki sabit subserver dışındakilerin kapasiteleri sıfırlandığında arayüzden ilgili progress bar silinir.

6.KAYNAKÇA

<https://www.udemy.com/course/sifirdan-ileri-seviyeye-komple-java-gelistirici-kursu/>

<https://stackoverflow.com/questions/2491588/how-a-thread-should-close-itself-in-java>

<https://www.geeksforgeeks.org/synchronized-in-java/>

<https://www.baeldung.com/java-wait-notify>

<https://www.geeksforgeeks.org/inter-thread-communication-java/>

<https://www.youtube.com/watch?v=7_oE2ybXEFs&t=2s>

<https://stackoverflow.com/questions/8916064/how-to-add-a-progress-bar>

<https://www.geeksforgeeks.org/producer-consumer-solution-using-threads-java/>

7.EKRAN ÇIKTILARI



