

**TEKNOLOJİ FAKÜLTESİ**

# BİLGİSAYAR MÜHENDİSLİĞİ

**Dr. Öğr. Üyesi SELMAN HIZAL**

**İşletim Sistemleri Dersi**

**1.Proje Ödevi**

Hazırlayanlar:

Seyfullah Kurt 23080103219

Beytullah Aziz Yapar 23080103217

**Giriş**

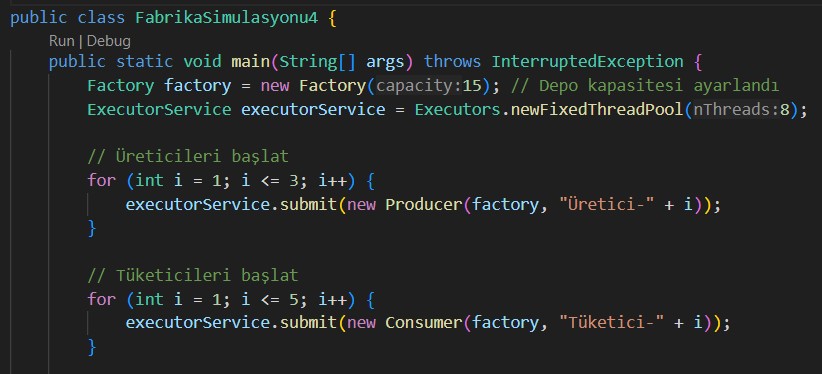
Bu projenin amacı, işletim sistemlerinin ileri düzey kavramlarını uygulamalı olarak öğrenmek ve bu kavramlar üzerine kapsamlı projeler geliştirmektir. İşletim sistemleri, modern bilgisayarların temel yapı taşlarından biridir ve bu sistemlerin verimli ve güvenli çalışması, bilgisayar performansının ve kullanıcı deneyiminin temelini oluşturur. Bu bağlamda, gelişmiş thread yönetimi ve senkronizasyon, process yönetimi ve iletişimi, dosya sistemi yönetimi ve gelişmiş girdi/çıktı işlemleri, ve sistem kaynak yönetimi ve planlama gibi konular üzerine kapsamlı projeler gerçekleştirilmiştir.

1. **Gelişmiş Thread Yönetimi ve Senkronizasyon:**

Üretici-tüketici probleminin gelişmiş bir versiyonunu içeren bu bölümde java dili kullanılarak bir simulasyon geliştirilmiştir. Simulasyonda farklı hızlarda çalışan birden fazla üretici ve tüketici bulunmaktadır. Aşırı yüklemenin önüne geçmek için buffer mekanizması kullanılmıştır ve buffer üzerindeki değişikliklerin simulasyonda nasıl bir etki yarattığı gözlemlenmiştir.

**Kod:**

Projede 3 producer 5 consumer olmak üzere 8 threat oluşturuldu ve değişmleri gözlemleyebilmek için buffer değişkeni tanımlandı.



Consumer ve producer arasında dengesizlik oluşturmak için üretim ve tüketim süreleri ayarlandı.

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

Açıklama otomatik olarak oluşturuldu

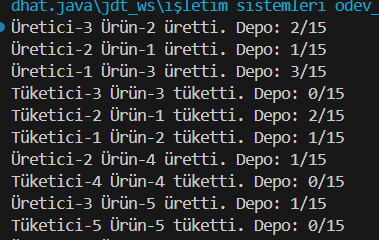
metin, yazı tipi, ekran görüntüsü, grafik içeren bir resim

Açıklama otomatik olarak oluşturuldu

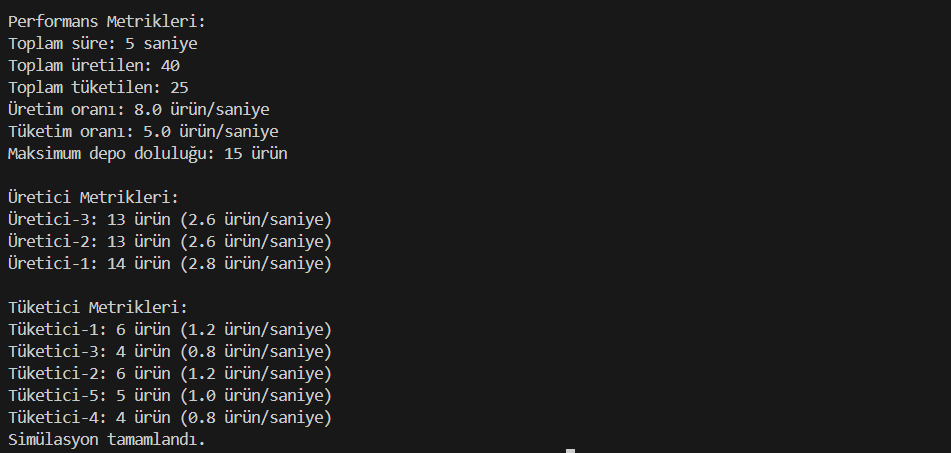
**Karşılaşılan Problemler ve Çözümleri:**

İnteeger tipinin tek seferde sadece 1 thread tarafından değiştirilebilmesi yüzünden üretilen, tüketilen ürünlerin adeti ve performans metrikleri yanlış bilgi vermekteydi. Bu yüzden Atomicİnteeger veri tipi kullanılarak threadlerin oluşturduğu veri karmaşından kurtulduk. Semaforların doğru tanımlanmaması sebebi ile senkonizasyon problemleri ile karşılaştık. Bunları da günümüz yapay zeka modellerinden yardım alarak semafor tanımlamalarını hallederek çözdük bu sayede üretilen bir ürün iki farklı thread tarafınfan tüketilmeye çalışılmadı.

**Çıktı:**

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

Açıklama otomatik olarak oluşturuldu**

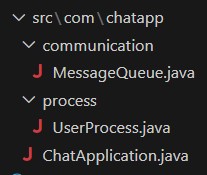
****

1. **Process Yönetimi ve İletişimi (Inter-Process Communication, IPC):**

Bu bölümde, birden fazla process'in birlikte çalıştığı ve veri alışverişi yaptığı bir sistem geliştirilmiştir. Process'ler arasında mesaj kuyruğu (message queue) kullanılarak iletişim sağlanmıştır. Örnek senaryo olarak yerine getirilmeye çalışılmıştır ve bir chat uygulaması simülasyonu gerçekleştirilmiştir.

**Kod:**

Proje 1. Bölümde olduğu gibi java üzerinde 1 paket altında 3 ayrı class olarak gerçekleştirlmiştir.



3 adet kullanıcı için 3 adet thread oluşturulmuştur.metin, ekran görüntüsü, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

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

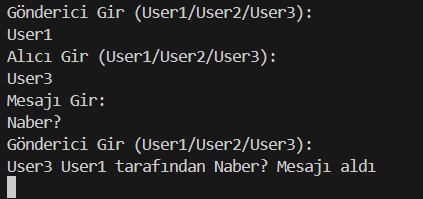
Açıklama otomatik olarak oluşturuldu

Sistem kullanıcıdan Gönderici ve Alıcı bilgilerini aldıktan sonra gönderilecek olan mesajı istemektedireğer q ya basılırsa program tamamen sonlandırılmaktadır. Mesaj alındıktan sonra alıcıya gönderici tarafından mesaj geldiğinin bilgisi gösterilmektedir.

**Karşılaşılan Problemler ve Çözümleri:**

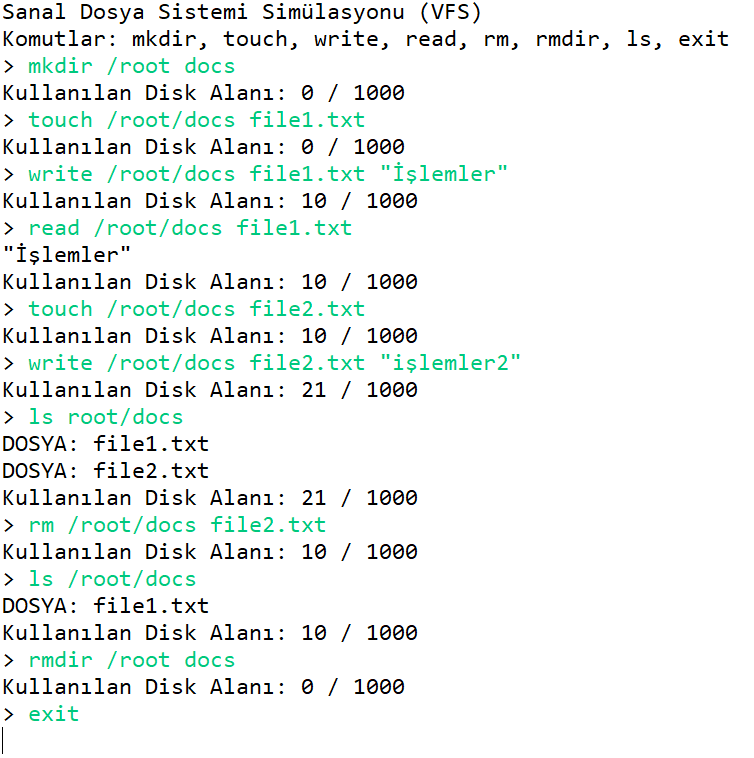
Çıktıda da görülebileceği üzere gönderici gir yazısı gönderilen metotdan önce yazılmaktadır bunun sebebi kullanılan ana kodun kendisinin de bir thread olmasından kaynaklı while döngüsü threadin çıktısını beklemeden tekrar başlıyor ve bu yüzden herzaman önce çalışıyor bunu çözmek için yazma işlemini ana threade alabilirdik fakat mesaj alındı bilgisini mesajı alan kişinin görmesi gerektiğinden değiştirmedik ve çözümünü bulamadık.

**Çıktı:**



1. **Dosya Sistemi Yönetimi ve Gelişmiş Girdi/Çıktı İşlemleri:**

Bir sanal dosya sistemi (Virtual File System, VFS) simüle edilmiş ve bu dosya sistemi, farklı türde dosya ve dizinleri desteklemiş, dosya işlemleri (okuma, yazma, silme, listeleme) yapılabilmiş ve aynı zamanda disk alanı yönetimi de içermiştir. Örnek senaryo olarak bir işletim sistemi simülasyonu yapılmıştır.



Sanal dosya sistemimizde mkdir, touch, write, read, ls, rm, rmdir ve exit olmak üzere 8 adet komutu desteklemektedir bu komutlar yaptığı işler ve kullanım yöntemleri şu şekildedir:

1 **mkdir <path> <dirname>**:

* **Amaç**: Belirtilen yolda yeni bir dizin (klasör) oluşturur.
* **Kullanım**: mkdir /root docs
* **Açıklama**: Bu komut, /root dizininde docs adlı bir alt dizin oluşturur.

2 **touch <path> <filename>**:

* **Amaç**: Belirtilen yolda boş bir dosya oluşturur.
* **Kullanım**: touch /root/docs file1.txt
* **Açıklama**: Bu komut, /root/docs dizininde file1.txt adlı boş bir dosya oluşturur.

3 **write <path> <filename> <content>**:

* **Amaç**: Belirtilen dosyaya içerik yazar. Eğer dosya mevcut değilse hata verir.
* **Kullanım**: write /root/docs file1.txt "Hello, world!"
* **Açıklama**: Bu komut, /root/docs/file1.txt dosyasına "Hello, world!" metnini yazar.

4 **read <path> <filename>**:

* **Amaç**: Belirtilen dosyanın içeriğini okur ve ekrana yazdırır.
* **Kullanım**: read /root/docs file1.txt
* **Açıklama**: Bu komut, /root/docs/file1.txt dosyasının içeriğini ekrana yazdırır.

5 **rm <path> <filename>**:

* **Amaç**: Belirtilen dosyayı siler.
* **Kullanım**: rm /root/docs file1.txt
* **Açıklama**: Bu komut, /root/docs/file1.txt dosyasını siler.

6 **rmdir <path> <dirname>**:

* **Amaç**: Belirtilen dizini ve içeriğini siler.
* **Kullanım**: rmdir /root docs
* **Açıklama**: Bu komut, /root dizininde bulunan docs adlı alt dizini ve içeriğini siler.

7 **ls <path>**:

* **Amaç**: Belirtilen dizinin içeriğini listeleyerek ekrana yazar.
* **Kullanım**: ls /root/docs
* **Açıklama**: Bu komut, /root/docs dizinindeki dosya ve alt dizinleri listeler.

8 **exit**:

* **Amaç**: Programı sonlandırır.
* **Kullanım**: exit

**Karşılaşılan Problemler ve Çözümleri:**

Proje geliştirilirken karşılaşılan en önemli sorunlardan biri, dosya yazma işlemlerinde disk alanı kullanımının doğru bir şekilde güncellenmemesiydi. Örneğin, dosyalar yazıldığında, diskte kapladıkları alan yalnızca son eklenen dosyanın boyutu kadar görünüyordu. Bu durum, dosya silindiğinde hafızada kapladığı alanın güncellenmemesiyle daha da karmaşık hale geliyordu. Sorunu çözmek için, dosya yazma ve silme işlemleri sırasında, her dosyanın mevcut boyutunu ve toplam disk kullanımını dikkate alacak şekilde kodu yeniden yapılandırdık. Bu şekilde, bir dosya silindiğinde, hafızada kapladığı alan doğru bir şekilde güncellenerek kullanılabilir disk alanı yeniden hesaplandı.

1. **Sistem Kaynak Yönetimi ve Planlama (Scheduling):**

CPU zamanlayıcı algoritmalarını (Round Robin, Priority Scheduling, Multi-Level Queue Scheduling) kullanarak bir işletim sistemi simüle edilmiştir. Farklı önceliklerdeki işlerin etkin bir şekilde zamanlanması sağlanmış ve örnek senaryo olarak bir işletim sistemi simülasyonu gerçekleştirilmiştir.

metin, ekran görüntüsü, yazı tipi, sayı, numara içeren bir resim

Açıklama otomatik olarak oluşturuldu

metin, ekran görüntüsü, yazı tipi, sayı, numara içeren bir resim

Açıklama otomatik olarak oluşturuldu

Kullanıcıdan aldığımız proses sayısı, proses burst time, proses gelme zamanı ve quantum değerleri ile Round Robin, Priority Scheduling ve Multi-Level Queue Scheduling algoritmalarını çalıştırarak proses sıralama işlemi ve performans metriklerini hazırladık. Bu algoritmalar sayesinde her prosesin bekleme süresi, dönüş süresi gibi performans metriklerini hesaplayarak karşılaştırma yapabildik ve süreçlerin verimli bir şekilde yönetilmesini sağladık.

**Karşılaşılan Problemler ve Çözümleri:**

Performans metriklerini hazırlarken zorlandık. Özellikle performans metriklerinin hesaplanması ve güncellenmesi aşamalarında sorun yaşadık, bu da sonuçlarda hatalı veriler oluşmasına neden oldu. Ortalama bekleme süresi ve dönüş süresi gibi metrikleri doğru hesaplamak ve her algoritmanın verilerini ayrı ayrı tutmak karmaşık bir süreçti. Bu süreçte yaptığımız bazı hatalar, metriklerin tutarsız güncellenmesi ve doğru veri yönetimi yapamamamızdan kaynaklandı. Kodumuzu modüler hale getirerek ve daha fazla test yaparak bu sorunları büyük ölçüde çözmeyi başardık, ancak bu deneyim, algoritma performansını değerlendirmenin ne kadar hassas ve dikkat gerektiren bir iş olduğunu gösterdi