# SAKARYA ÜNİVERSİTESİ

# BİLGİSAYAR VE BİLİŞİM BİLİMLERİ FAKÜLTESİ BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ

**İŞLETİM SİSTEMLERİ DERSİ PROJE RAPORU**

# Grup No:12

**Hazırlayan Öğrenciler:**

## Bedirhan Yıldırım – B211210387 / 1B Emre Tümer – B221210303 / 1B

Roham Rasouli Kerahroudi – B211210561 / 1B

Yasin Enes Yılmaz – B221210309 / 1B

## Burak Osman Çelik – B211210051 / 1B

**Github Linkleri:**

Bedirhan Yıldırım = <https://github.com/bdrhnyldrm/IsletimSistemleriProjeOdevi> (Ödev dosyasının son hali bu linkte)

Emre Tümer = <https://github.com/emreetumer/isletimsistemleri>

Roham Rasouli Kerahroudi = <https://github.com/RohamRasouli/isletim-sistemleri-projesi>

Yasin Enes Yılmaz = <https://github.com/ysn52/isletimsistemleri_yasin>

Burak Osman Çelik = <https://github.com/BurakOsmanClk/Isletim_Sistemleri_Proje>

**Proje Açıklaması:**

Bu proje, temel işletim sistemi konseptlerini simüle etmek amacıyla Java programlama dili kullanılarak geliştirilmiştir. İşlemlerin yönetimi, kuyruk sistemleri, zaman asımı kontrolü ve renkli çıktılar gibi temel işletim sistemi özellikleri, bu projede ele alınmıştır.

**Proje Geliştirme Adımları:**

* İşlemleri temsil eden **Proses** sınıfının oluşturulması.
* İşlemlerin yönetimini sağlayan **ProsesYonetimi** sınıfının tasarlanması.
* Dış dosyadan işlemlerin okunup kuyruklara eklenmesi için **ReadWrite** sınıfının yazılması.
* Çalışan işlemleri, bekleyen işlemleri ve geri bildirim kuyruklarını yöneten **Runtime** sınıfının tasarlanması.
* Renkli çıktılar için **RenkAyar** sınıfının oluşturulması.

**Raporda İstenenler:**

**A)**

Bellek tahsis algoritmaları, işletim sistemlerinde programların çalışması için gerekli olan bellek alanını yönetmek için kullanılır. Bu algoritmalar, mevcut bellek içinde hangi bölgelerin bir programa tahsis edileceğini belirler. İşte yaygın olarak kullanılan bellek tahsis algoritmaları ve her birinin avantajları ve dezavantajları:

* **First Fit (İlk Uygun):**

**Çalışma Prensibi:** Bellekteki ilk uygun bölgelerden birine programı yerleştirir.

**Avantajlar:**

Basit ve hızlı bir algoritmadır.

İlk uygun boşluğu hızlı bir şekilde bulur.

**Dezavantajlar:**

Fragmentasyon sorunlarına yol açabilir.

İlk uygun boşluğu seçmek, daha sonraki programlar için uygun alan bırakmada etkili olmayabilir.

* **Best Fit (En Uygun):**

**Çalışma Prensibi:** Programı yerleştirmek için en küçük uygun boşluğu seçer.

**Avantajlar:**

Fragmentasyonu azaltabilir.

**Dezavantajlar:**

İşlem süresi biraz daha uzun olabilir, çünkü en uygun boşluğu bulmak için tarama yapılması gerekir.

* **Worst Fit (En Kötü):**

**Çalışma Prensibi:** Programı yerleştirmek için en büyük uygun boşluğu seçer.

**Avantajlar:**

Fragmentasyonu bir miktar azaltabilir.

**Dezavantajlar:**

Genellikle diğer algoritmalar kadar etkili değildir.

* **Next Fit (Sıradaki Uygun):**

**Çalışma Prensibi:** En son yerleştirilen programın hemen sonrasındaki ilk uygun boşluğa programı yerleştirir.

**Avantajlar:**

First Fit'e benzer, ancak bir önceki işlemi daha etkili takip eder.

**Dezavantajlar:**

Fragmentasyon sorunlarına neden olabilir.

* **Buddy Allocation (Arkadaş Tahsisi):**

**Çalışma Prensibi:** Belleği iki katına çıkarılabilir bloklara böler.

**Avantajlar:**

Fragmentasyonu minimize eder.

Hafif bir hesaplama yükü vardır.

**Dezavantajlar:**

Wasteful (israf) olabilir, çünkü bellek blokları iki katına çıkarılır ve bir program ihtiyacından fazla alan kaplayabilir.

**B)**

Görevlendirici (scheduler), bir işletim sistemi içinde çalışan görevlerin (process) zamanını ve kaynakları yöneten bir bileşendir. Görevlendirici, bellek ve diğer kaynakları kuyruğa almak, göndermek ve tahsis etmek için belirli veri yapılarını kullanır. İşte bu görevleri gerçekleştirmek üzere kullanılan temel veri yapıları:

* **Bellek Yönetimi Kuyrukları:**

**Runtime Sınıfında Tanımlı (Runtime.java):**

* **ArrayList<Queue<Proses>> feedbackQueues**: Öncelik bazlı kuyrukları içeren bir liste. Her bir kuyruk, belirli bir önceliğe sahip işlemleri içerir.
* **Queue<Proses> runtimeProcesses**: Çalışan işlemlerin tutulduğu kuyruk.
* **List<Proses> pausedProcesses**: Bekleyen ve durdurulmuş işlemlerin tutulduğu liste.
* **Proses Nesnesi ve Özellikleri:**

**Proses Sınıfında Tanımlı (Proses.java):**

* **int hafiza, yazici, tarayici, modem, cdDrivers**: İşlem tarafından talep edilen kaynaklar.
* **boolean isError**: İşlemde hata olup olmadığını belirten bir bayrak.
* **int time, sure, oncelik**: İşlemin oluşturulma zamanı, süresi ve önceliği.
* **boolean isStarted, isPaused**: İşlemin başlatılıp başlatılmadığı ve duraklatılıp duraklatılmadığına dair bayraklar.
* **ReadWrite Sınıfı (readWrite.java):**
* Dış bir dosyadan okunan işlemlerin özelliklerini içeren **Proses** nesnelerini oluşturur ve **Runtime** sınıfındaki kuyruklara ekler.
* **RenkAyar Sınıfı (RenkAyar.java):**
* Konsol çıktılarını renklendirmek için kullanılan bir yardımcı sınıf.

Bu veri yapıları, işlemlerin bellek ve diğer kaynak taleplerini organize etmek ve yönetmek için kullanılır. **Runtime** sınıfı, bu kuyrukları kullanarak işlemleri bellek tahsis etmeye, çalıştırmaya, bekletmeye ve sonlandırmaya yönlendirir. İşte bu veri yapıları, işletim sistemi içindeki kaynak yönetimi görevlerini etkili bir şekilde gerçekleştirmek için kullanılır.

**C)**

Projemizin genel yapısını anlamak için modüllerin ve ana işlevlerin açıklamalarını inceleyelim:

**Main Modül (Main.java):**

**Ana İşlev:** Programın başlangıcını yönetir. Kuyrukları oluşturur, bir dosya yolu alır (opsiyonel) ve **ProsesYonetimi** sınıfını başlatır.

**ProsesYonetimi Modülü (ProsesYonetimi.java):**

**Ana İşlev:** İşlemleri yönetir, zamanı kontrol eder ve işlemleri başlatır, bekletir veya sonlandırır.

**Metodlar:**

* **run()**: Thread'ın ana çalışma metodu. İşlemleri zaman kontrolü ve yönetimi için sürekli olarak takip eder.

**Proses Modülü (Proses.java):**

**Ana İşlev:** Bir işlemi temsil eder ve ilgili özelliklere sahiptir. İşlem başlatma, bekletme, sonlandırma gibi işlevleri içerir.

**Metodlar:**

* **ProsesiBaslat()**: İşlemi başlatır.
* **ProsesiYurut()**: İşlemi yürütür.
* **ProsesiBeklet()**: İşlemi bekletir.
* **ProsesiBitir()**: İşlemi sonlandırır.
* **ProsesZamanAsımı()**: İşlemin zaman aşımını kontrol eder.

**ProsesYonetimi Modülü (Runtime.java):**

**Ana İşlev:** Bellek, işlem yönetimi ve zaman kontrolü için temel metotları içerir.

**Metodlar:**

* **ProsesCalismaZamani()**: İşlem çalışma zamanını kontrol eder.
* **ProsesGeriBildirim()**: İşlem geri bildirimini yönetir.
* **ProsesBeklet()**: İşlemi bekletir.
* **ProsesGeriBildirim()**: İşlem geri bildirimini kontrol eder.
* **hataProsesKontrol()**: Hata durumlarını kontrol eder.
* **zamanAsimiProsesKontrol()**: İşlem zaman aşımını kontrol eder.

**ReadWrite Modülü (readWrite.java):**

**Ana İşlev:** Dosyadan okuma işlemlerini yönetir ve işlemleri kuyruklara ekler.

**Metodlar:**

* **kuyrugaEkle(int time)**: Belirli bir zamandaki işlemleri kuyruklara ekler.

**RenkAyar Modülü (RenkAyar.java):**

**Ana İşlev:** Konsol çıktılarını renklendirmek için kullanılan sabit renkleri içerir.

**Metodlar:**

* **rastgeleRenk(int id)**: Belirli bir ID için rastgele bir renk döndürür.

**D)**

Bu çok düzeyli görevlendirme şemasının gerçek işletim sistemleri tarafından kullanılan şemalara karşı avantajları ve olası iyileştirmelerle birlikte bazı eksiklikleri aşağıda tartışılmıştır:

**Avantajlar:**

**Öncelikli İşlemler:**

* + Çok düzeyli görevlendirme, işlemlere öncelik düzeyleri tanıyarak kritik görevlerin daha hızlı yanıt vermesini sağlar.

**İş Parçacığı Desteği:**

* + Çok düzeyli görevlendirme, iş parçacığı (thread) desteği ile paralel işlemleri daha etkili bir şekilde yönetebilir.

**Dinamik Ayarlamalar:**

* + Önceliklerin dinamik olarak değiştirilebilmesi, sistemin performansını uygulama gereksinimlerine göre optimize etmeyi sağlar.

**Adil Kaynak Paylaşımı:**

* + Farklı öncelik seviyelerindeki işlemler arasında adil bir kaynak paylaşımı sağlar.

**Hata Koruması:**

* + Düşük öncelikli işlemler, yüksek öncelikli işlemlerin hata durumlarına karşı daha dirençli olabilir, bu da sistem stabilitesini artırabilir.

**Olası İyileştirmeler:**

**Dinamik Öncelik Ayarlamaları**:

* + Sistem, iş yüküne ve kullanıcı taleplerine göre öncelikleri otomatik olarak ayarlayabilirdi. Bu, dinamik ortamlarda daha etkili bir performans sağlayabilir.

**Genişletilmiş İşlem Yönetimi:**

* + İş parçacıkları, çekirdek sayısına göre daha efektif bir şekilde yönetilebilir. Çoklu çekirdekli sistemlerde paralel işlemlerin daha iyi desteklenmesi sağlanabilir.

**Dinamik Bellek Yönetimi:**

* + Bellek tahsisinde daha sofistike algoritmaların kullanılması, sistem performansını artırabilir ve bellek kullanımını optimize edebilir.

**Eksiklikler:**

**Güvenlik Sorunları:**

* + Öncelik bazlı görevlendirme, yüksek öncelikli işlemlerin diğerlerini engelleyebileceği bir güvenlik sorunu yaratabilir.

**Küçük Ölçekli Sistemler:**

* + Çok düzeyli görevlendirme, küçük ölçekli sistemlerde gereksiz karmaşıklığa neden olabilir ve bu durum performansı düşürebilir.

**Bellek Kullanımı:**

* + Bellek yönetimi algoritmalarının geliştirilmesi, bellek kullanımını daha etkili hale getirebilir. Özellikle büyük ölçekli uygulamalarda bu önemli bir faktördür.

**Küresel Kaynak Yönetimi:**

* + Çok düzeyli görevlendirme, küresel kaynak yönetimi için daha sofistike bir mekanizma gerektirebilir. Bu, sistem genelinde daha etkili kaynak kullanımını sağlayabilir.

**Sonuç ve Değerlendirme:**

Bu proje, işletim sistemi simülasyonunu basit bir düzeyde gerçekleştirmek amacıyla geliştirilmiştir. İşlemlerin yönetimi, kuyruk sistemleri ve renkli çıktılar gibi temel konular üzerine odaklanarak, bir işletim sistemi davranışının temelini atmaktadır. Bu projenin geliştirilmesi, işletim sistemleri konularına ilgi duyanların temel anlayışlarını güçlendirebilir ve işletim sistemi davranışlarını daha iyi anlamalarına yardımcı olabilir.