T.C.

SAKARYA ÜNİVERSİTESİ

BİLGİSAYAR VE BİLİŞİM BİLİMLERİ FAKÜLTESİ

BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ

İŞLETİM SİSTEMLERİ PROJE RAPORU

B201210033- Yusuf ÖZASLAN

B201210351- Eray BAYSAL

B201210097- Mazhar AKYOLDAŞ

B191210555- Tea SHKURTİ

B201210089- Yusuf GÜNCE

GitHub: https://github.com/YusufOzaslan/Operating-Systems-Project

SAKARYA

Ocak, 2023

İşletim Sistemleri Dersi

Özet

Java programlama dili kullanılarak sınırlı kullanılabilir kaynakların kısıtlamaları içinde çalışan dört seviyeli öncelikli proses görevlendiricisine sahip bir çoklu programlama sistemi simüle edilmiştir.

Not: Ödevde istenilen jar dosyası projenin **dist** klasörü içerisindedir. **Github:**https://github.com/YusufOzaslan/Operating-Systems-Project

Anahtar Kelimeler: Dispatcher, FCFS, proses, round robin, kuantum, geri besleme, java, zaman aşımı

GELİŞTİRİLEN YAZILIM

Program çalıştırıldığında giris dosyasından okunan prosesler Dispatcher nesnesinde bulunun ve prosesleri tutan kuyruğa yerleştirilir. Proseslerin varış zamanı, öncelik değeri, işlenme zamanı gibi özellikleri vardır. Kuyruğa yerleştirilen prosesler özelliklerine göre sıralanarak uygun öncelikli kuyruklara yerleştirilir.

Gelen Procces'lerin Sıralanması ve Kuyruklara Alınması

Giris dosyasından gelen process'leri öncelik değerlerine göre öncelikli kuyruklara yerleştiriyoruz. Dört öncelik değeri için dört adet kuyruğumuz bulunmakta.

Ardından process'ler karışık sırayla geldiği için ArrivalTime değişkenlerine göre sıraladık. Bu bizi process'leri her seferinde aramaktan kurtardı. Time değişkenine göre ilk elemana bakılıyor eğer ilk elemanın ArrivalTime'ı, Time değerimizden büyükse devamına bakmaya gerek kalmamaktadır. Sıralama fonksiyonunun içine bakacak olursak; Gelen kuyruğu SelectionShort algoritmasıyla beraber ArrivalTime'a göre sıraladık.

Görevlendiricinin Çalışması

Görevlendirici, sıfır öncelik değerine sahip prosesleri bulunduran kuyruğu gezmekle başlar. Görevendiriciye gelen proses varsa bu proses FCFS algoritmasıyla işletilir. Sonlanana kadar devam eder. Bu kuyruk gezilirken aynı zamanda diğer kuyruklar da kontrol edilir. Kontrol için feedback() ve timeOutCheck fonksiyonları çağrılır. Eğer sıfır öncelikli proseslerin bulunduğu kuyrukta proses yoksa feedback fonksiyonunda öncekik sırasına göre prosesler

çalıştırılır. Kuantum değeri bir olduğu için feedback'de her proses bir saniye çalışır ve prosesin önceliği düşürülerek askıya alınır. Askıya alınan prosesler farklı bir kuyrukta tutulur ve tekrar sıra kendine gelene kadar o kuyrukta bekler.

Tüm bu işlemler timer değişkeniyle senkronize edilir. timeOutCheck fonksiyonuyla da yirmi saniye çalışmayan prosesler tespit edilerek kuyruktan çıkarılır ve prosesin zaman aşımına uğradığını gösteren mesajlar konsola yazılır.

Olası İyileştirmeler

Görevlendiricide kullanılan sıralama algoritmasında daha efektif bir seçim yapabilirdik. Onun haricinde görevlendirici şuanda sadece 4 seviye için çalışmaktadır daha yüksek seviyelerde ek olarak kod yazmamız gerekli ve öncelik değiştirmek istediğimizde maliyeti yüksek olabilir.

Procces Sınıfı

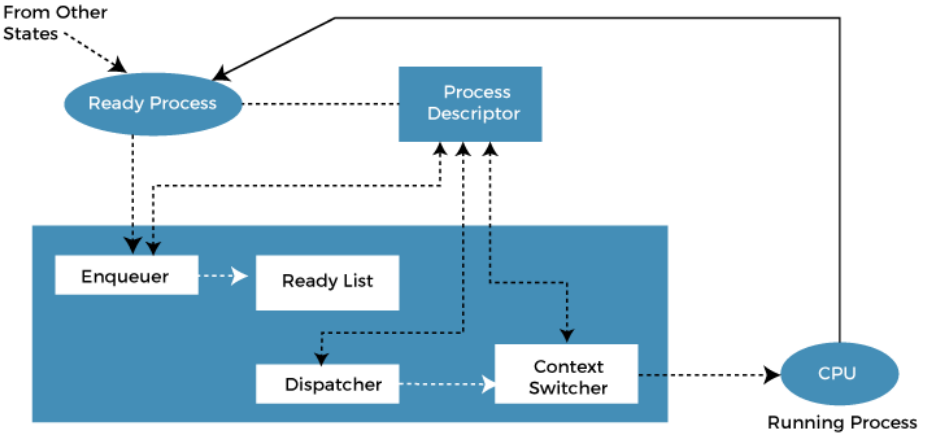
Dosyadan okunan her proses için kendi yazdığımız proses sınıfından nesneler oluştururuz ve gerekli değişkenlein atamasını yaparız.

Görevlendirici tarafından prosesin çalışması için proses sınıfımızın execute fonksiyonu çağrılmalıdır. Bu fonksiyon Java proseslerini kullanarak daha önce oluşturduğumuz jar dosyasını çalıştırır ve konsola prosesin bilgileri yazdırılır. Java prosesi Runtime.exec() kullanılarak çağrılır.

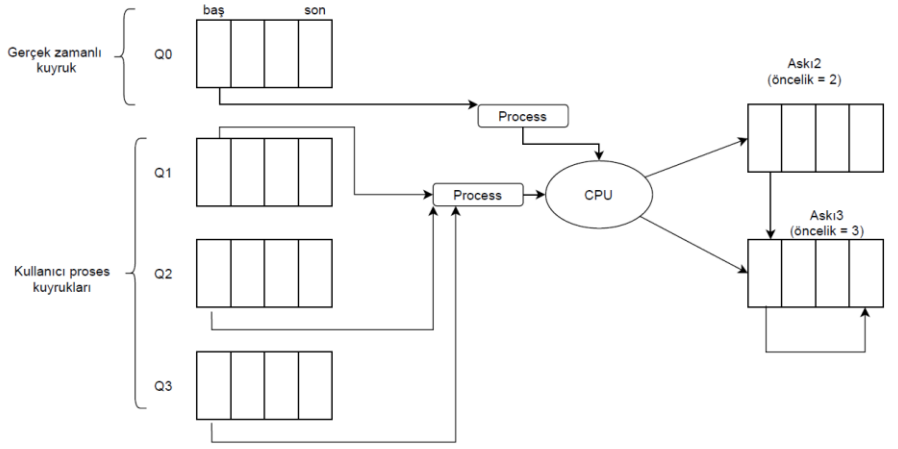
Proses mesajlarının renkli çıkabilmesi için ColoredSystemOutPrintLn sınıfındaki ANSI escape sequence'ları kullanılmıştır. Her yeni proses oluştuğunda COLORS array'indeki ANSI kodlarından biri o prosese özel bir colorId property'si şeklinde atanır.Bu colorId'ler, proses mesajlarının stringlerinde başa gelir. Proses mesajları ekrana renkli bir şekilde çıkar.

Görevlendirici Karşılaştırması

Görevlendirici modülünün işletim sistemi üzerindeki görevi gelen işlermlerin CPU'yu kullanımını düzenler, process'leri uygun pozisyonlara koyar ve kullanıcı modunu değiştirmeyi sağlar. Görevlendirici modülleri tek bir amaca hizmet etmediği için birden fazla çeşiti vardır. Bu yüzden kendi ihtiyaçlarımıza özel oalrak görevlendirici algoritması yazabiliriz. İşletim sisteminin kullandığı görevlendirici şemasına bakacak olursak (bkz Görsel:1), gelen process'ler hazır kuyruğuna alınıyor ve sırası gelen process çalıştırılmak üzere CPU'ya yönlendiriliyor.



Bizim görevlendirici mekanizmamıza bakacak olursak :



Gelen processler önceliklerine göre cpu kullanımına izin veriliyor ve ardından önceliği 0 değilse bir alt seviyeye düşürülüp işleme devam ettiriliyor. Önceliği sıfır olmayan bir process işlem görürken sıfır öncelikli bir process gelirse işlem gören process askı kuyruğuna alınıyor ve sıfır öncelikli process'in işlemi bitene kadar bekliyor bekleme süresi 20 saniyeyi aşarsa process zaman aşımına uğrayıp siliniyor. Gerçek işletim sistemi process'leri alırken hazır kuyruğundan alıp işlettikten sonra aynı listenin sonuna tekrar ekleyerek devam etmekte biz bunu modellerken 2 adet askı kuyruğu kullandığımız için depolama performansımız düşük kaldı.

EKRAN ÇIKTISI

```
0 sn proses basladi                    (id:0  oncelik:1  kalan sure:2 sn)
1 sn proses askida                     (id:0  oncelik:2  kalan sure:1 sn)
1 sn proses basladi                    (id:1  oncelik:0  kalan sure:1 sn)
2 sn proses sonlandi            (id:1  oncelik:0  kalan sure:0 sn)
2 sn proses basladi                    (id:3  oncelik:0  kalan sure:3 sn)
3 sn proses yurutuluyor         (id:3  oncelik:0  kalan sure:2 sn)
4 sn proses yurutuluyor         (id:3  oncelik:0  kalan sure:1 sn)
5 sn proses sonlandi            (id:3  oncelik:0  kalan sure:0 sn)
5 sn proses basladi                    (id:6  oncelik:0  kalan sure:4 sn)
6 sn proses yurutuluyor         (id:6  oncelik:0  kalan sure:3 sn)
7 sn proses yurutuluyor         (id:6  oncelik:0  kalan sure:2 sn)
8 sn proses yurutuluyor         (id:6  oncelik:0  kalan sure:1 sn)
9 sn proses sonlandi            (id:6  oncelik:0  kalan sure:0 sn)
9 sn proses basladi                    (id:7  oncelik:0  kalan sure:4 sn)
10 sn proses yurutuluyor               (id:7  oncelik:0  kalan sure:3 sn)
11 sn proses yurutuluyor               (id:7  oncelik:0  kalan sure:2 sn)
12 sn proses yurutuluyor               (id:7  oncelik:0  kalan sure:1 sn)
13 sn proses sonlandi           (id:7  oncelik:0  kalan sure:0 sn)
13 sn proses basladi                   (id:8  oncelik:0  kalan sure:2 sn)
14 sn proses yurutuluyor               (id:8  oncelik:0  kalan sure:1 sn)
15 sn proses sonlandi           (id:8  oncelik:0  kalan sure:0 sn)
15 sn proses basladi                   (id:10  oncelik:0  kalan sure:3 sn)
16 sn proses yurutuluyor               (id:10  oncelik:0  kalan sure:2 sn)
17 sn proses yurutuluyor               (id:10  oncelik:0  kalan sure:1 sn)
18 sn proses sonlandi           (id:10  oncelik:0  kalan sure:0 sn)
18 sn proses basladi                   (id:16  oncelik:0  kalan sure:4 sn)
19 sn proses yurutuluyor               (id:16  oncelik:0  kalan sure:3 sn)
20 sn proses yurutuluyor               (id:16  oncelik:0  kalan sure:2 sn)
21 sn proses yurutuluyor               (id:16  oncelik:0  kalan sure:1 sn)
21 sn proses zaman asimi               (id:0  oncelik:2  kalan sure:1 sn)
21 sn proses zaman asimi               (id:4  oncelik:2  kalan sure:2 sn)
21 sn proses zaman asimi               (id:2  oncelik:3  kalan sure:2 sn)
22 sn proses sonlandi           (id:16  oncelik:0  kalan sure:0 sn)
22 sn proses basladi                   (id:17  oncelik:0  kalan sure:4 sn)
22 sn proses zaman asimi               (id:5  oncelik:2  kalan sure:3 sn)
23 sn proses yurutuluyor               (id:17  oncelik:0  kalan sure:3 sn)
24 sn proses yurutuluyor               (id:17  oncelik:0  kalan sure:2 sn)
24 sn proses zaman asimi               (id:9  oncelik:2  kalan sure:4 sn)
25 sn proses yurutuluyor               (id:17  oncelik:0  kalan sure:1 sn)
25 sn proses zaman asimi               (id:11  oncelik:3  kalan sure:2 sn)
26 sn proses sonlandi           (id:17  oncelik:0  kalan sure:0 sn)
```