SAKARYA
ÜNİVERSİTESİ

BİLGİSAYAR MÜHENDİSLİĞİ
İŞLETİM SİSTEMLERİ
BSM 309 PROJE ÖDEVİ

Serdar Arıcı G191210020
Adem Yılmaz G191210305
Burak Ortakuz G191210041
Mehmet Ali Ateş G171210500
Mehmet Asım Özlen G211210300

05.01.2023

1 Programın Genel Yapısı

Program 5 sınıftan oluşur. Color, Görevlendirici, Kuyruk, Main ve Proses. Color sınıfında her yeni proses için tanımlanır ve rastgele bir renk ataması yapılır. Program çıktısında atanan renk ile yazdırılır. Proses sınıfı ile her yeni proses tanımlanır ve prosesin genel özelliklerini tutar. *Proses ID*, *Proses Durumu*, *Proses Varis Zamanı*, *Proses Oncelik Degeri* gibi değerler bu sınıfın içindedir. Ayrıca *getProsesId()*, *getProsesDurumu()*, *setProsesDurumu(String prosesDurumu)*, *getOncelikDegeri()* ve *setOncelikDegeri(int oncelikDegeri)* gibi fonksiyonlar ile prosesin bilgilerini okuyabilir ya da güncelleyebiliriz.

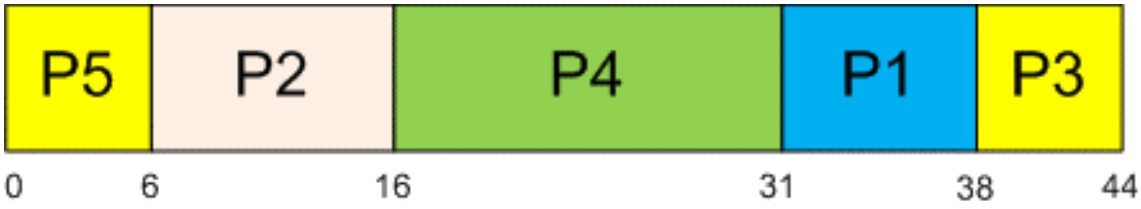
Kuyruk Sınıfı proses sınıflarından oluşan listenin belirli sıraya dizildiği sınıftır. Kuyruk sınıfını kullanarak listeye bir proses ekleme veya listeden bir proses silme yapılabildiği gibi listenin boş olup olmadığını en yüksek öncelikli prosesin hangisi olduğu gibi bilgileri almamızı sağlar. Kuyruk sınıfı sayesinde prosesler arasında bir sıra oluşur ve işlenecek prosese kolayca erişim sağlanır.

Görevlendirici Sınıfı birden fazla Kuyruk oluşturur. *Gerçek Zamanlı Prosesler Kuyruğu*, *Kullanıcı Prosesleri Kuyruğu*, *Yüksek Öncelik Prosesler Kuyruğu* gibi kuyrukları içerir. Böylelikle gerçek zamanlı işlenecek prosesler gibi aynı grupta olan prosesleri kendi aralarında sıraya alıp, FCFS sıralamasına kolaylıkla alabiliriz. Görevlendirici ayrıca *Basla()* fonksiyonu ile prosesleri uygun kuyruğa yerleştirdiği gibi kuyrukların çalışma sırasını ve önceliğini de belirler ve prosesleri çalıştırıp çıktı olarak yazdırmayı sağlar. Programın genel işleyişi bu sınıf içerisinde gerçekleşmektedir.

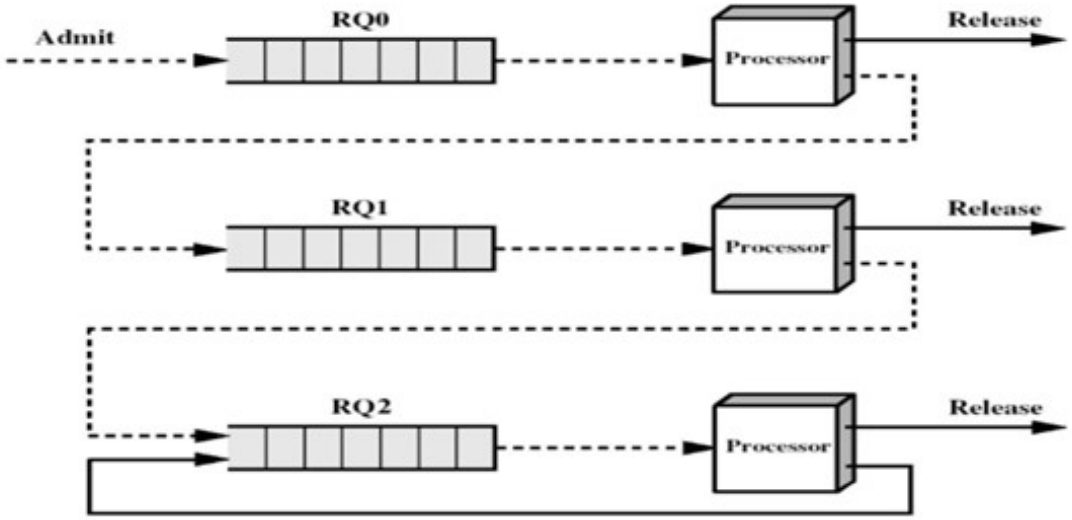
Main Sınıfı tüm sınıfları işleten sınıftır. Öncelikle *giris.txt* dosyasından prosesleri okur ve bu prosesleri bir kuyruğa alır. Bu kuyruğu *Görevlendirici Sınıfı*na göndererek önceliklerine göre sınıflandırıp yine önceliklerine göre işlenmesini sağlar.

2 Görevlendirici Tarafından Kullanılan Yapılar

2.1 FCFS

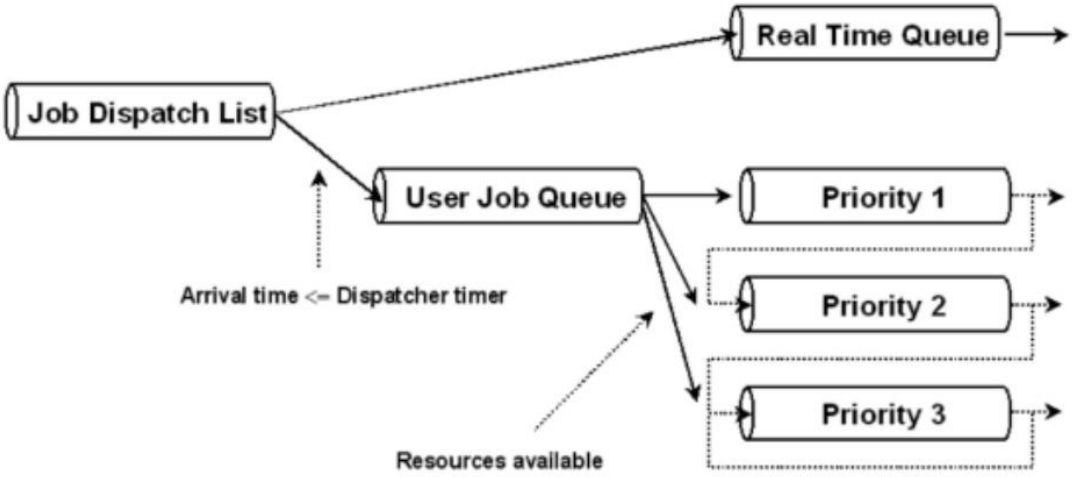


2.2 Üç Seviyeli Geri Beslemeli Görevlendirici



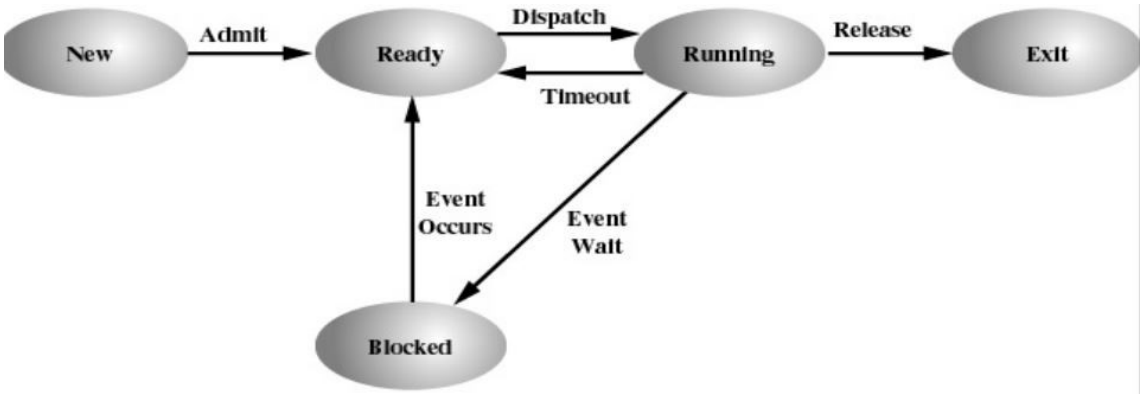
2.3 Round Robin Görevlendiricisi

2.4 Görevlendirici Mantık Akışı

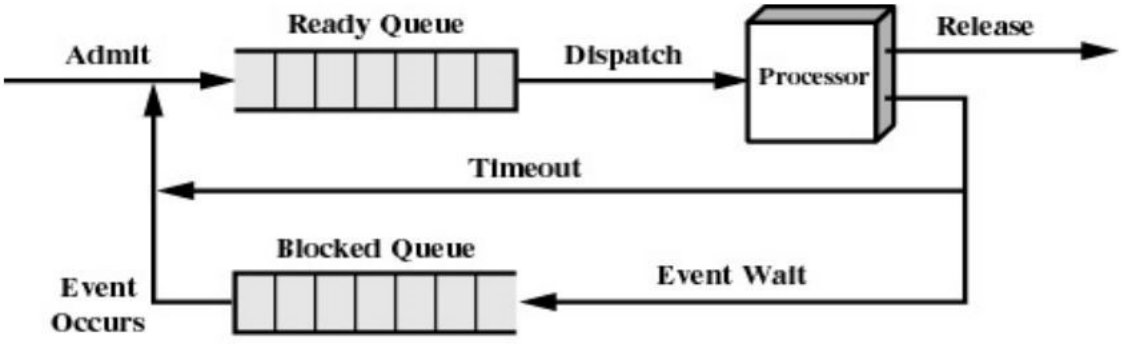


3 İşletim Sistemi Örnek Algoritmaları

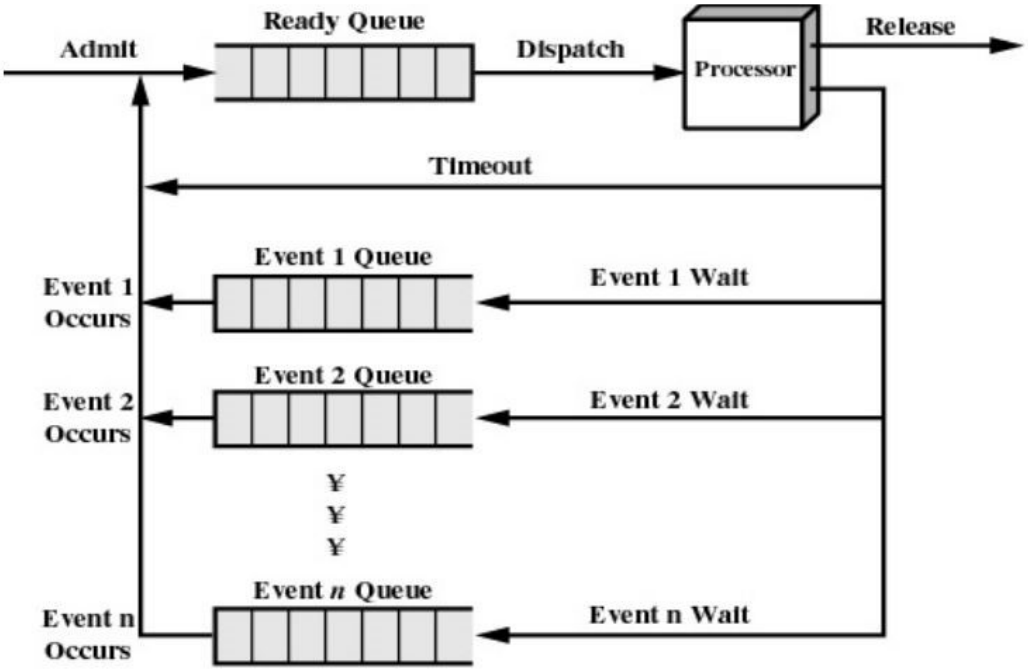
3.1 Genel Proses



3.2 Tekli Kuyruk



3.3 Çoklu Kuyruk



4 Çoklu Algoritmalar Kullanımı

Çoklu algoritma kullanımı, işletim sistemlerinin performansını ve verimliliğini artırmak için yapılır. Çoklu algoritma, bir işletim sisteminde birden fazla prosesin aynı anda yürütülmesini ve paylaşılan kaynakların düzenli bir şekilde yönetilmesini sağlar. Bu sayede, işletim sistemi daha hızlı ve etkin bir şekilde çalışır ve kullanıcılar için daha iyi bir deneyim sunar.

Çoklu algoritma kullanımı, prosesler arasında eşitlik ve dengeyi sağlar. Örneğin, bir işletim sisteminde bir prosesin diğerlerine göre daha fazla kaynak tüketmesi önlenir ve tüm prosesler eşit şekilde işlem görebilir. Bu sayede, işletim sistemi daha stabil bir hale gelir ve kullanıcılar için daha güvenli bir ortam yaratılır. Çoklu algoritma kullanımı, her farklı gruptaki prosesler arasında farklı sıralamalar yaparak esnek bir ortam oluşturmaktadır. Örneğin, gerçek zamanlı çalışmak isteyen prosesler kendi aralarında FCFS algoritması ile sıralanırken, kullanıcı prosesleri üç seviyeli geri beslemeli görevlendirici algoritması ile çalışabilir.

5 Olası İyileştirmeler

İlk olarak gerçek zamanlı prosesleri ele aldığımızda, FCFS algoritmasına göre çalıştıklarından sistemi timeout süresi kadar meşgul edebilirler. Bu durumda periyodik yapılabilecek çağrılar sistemin akışını bozup sistemi kullanılamayacak hale getirebilir. Bu durumu engellemek için yüksek öncelikli prosesleri de proses geri besleme görevlendirici kuyruğuna alıp daha fazla zaman tanımak ve kuyrukta direkt ilk sıraya alınması gibi yöntemler denenebilir.

Proses geri besleme görevlendiricisindeki kuantum (q) süresi sabit 1 sn olmamalıdır. Proseslerin kalan zamanlarına ve sistemin çalışma hızına göre dinamik olarak değişen bir değerde olmalıdır. Gerçek işletim sistemlerinde olan Olay *(Event)* bekleyicileri sisteme dahil edilmelidir. Örneğin, proseslerden biri, bir input bekliyor ise işlemciyi meşgul etmemeli olay gerçekleşene kadar bekletilmelidir. Ayrıca yedekleme gibi aciliyeti olmayan prosesler işlemcinin boş kaldığı zamanlarda devreye alınmalı ve yeni prosesler geldiğinde tekrar uyku moduna geçecek şekilde bir algoritma eklenmelidir.

6 Bellek ve Kaynak Kullanımı

İşletim sistemlerinde prosesler, bellek ve diğer kaynakları paylaşırlar. Bellek, proseslerin verilerinin saklandığı ve işlemlerin yapıldığı bir alandır. İşletim sistemleri, proseslerin bellek kullanımını düzenleyerek sistemin stabilitesini ve performansını artırır. Örneğin, bir prosesin bellek kullanımını sınırlamak veya bir prosesin bellek kullanımını izlemek gibi yöntemler kullanılabilir.

Kaynaklar, proseslerin çalışması için gerekli olan fiziksel veya dijital unsurlardır. Örneğin, bir prosesin çalışması için CPU, disket sürücüsü, veritabanı gibi kaynaklar gerekir. İşletim sistemleri, proseslerin kaynak kullanımını düzenleyerek sistemin verimliliğini artırır. Örneğin, bir prosesin CPU kullanımını sınırlamak veya bir prosesin veritabanı erişimini izlemek gibi yöntemler kullanılabilir. Kaynak ve bellek kullanımı her işletim sisteminin olmazsa olmazıdır. Bir işletim sistemi kaynaklarını ve belleğini verimli kullandığı ve dağılımını dengeli yaptığı ölçüde yüksek performansa sahip olacaktır.

7 Github Linki

Buraya tıklayınız.

Ayrıca; https://github.com/ademylmz34/IsletimSistemiOdev