SAKARYA ÜNİVERSİTESİ

SAKARYA
ÜNİVERSİTESİ

BİLGİSAYAR MÜHENDİSLİĞİ

İŞLETİM SİSTEMLERİ

BSM 309 PROJE ÖDEVİ

AHMET ARİF KARAKULLUKÇU	B211210078
DURSUN GÜVEN	B211210096
HASAN HÜSEYİN GEZER	B211210062
ABDÜLKADİR USTA	B231210387
TUNCER ALEVLİ	B211210074

Programın Genel Yapısı

Program 11 sınıftan oluşur:

(Main,FileService,Color,MemoryService,TarayiciService,YaziciService,ModemService,CdDriverService,Process,Dispatcher,ProcessService)

Main Sınıfı

- Program için uygun sınıfları oluşturduğumuz okunacak dosyayı fileservise tanıttığımız ve processleri başlattığımız sınıf.

File Service Sınıfı

- Giris.txt doyasını okuyup uygun parçalama işlerini yaptıktan sonra her satır için bir process oluşturan ve bu satırdaki bilgileri ilgili processin içine yerleştiren, değişkenleri oluşturulan processleri processlist isimli bir listeye ekleyen sınıfımız.

Color Sınıfı

- Processlere benzersiz bir renk vermek için oluşturduğumuz her processe benzersiz bir renk tahsis eden sınıfımız.

Tarayıcı Service Sınıfı

- Sistemdeki toplam tarayıcı adetlerini tanımladığımız ve içinde tarayıcının müsait mi olduğunu , ilgili processin bu tarayıcıyı kullanmasını ve işi biten processin tarayıcıyı serbest bırakmasını sağlayan kodların bulunduğu sınıf.

Modem Service Sınıfı

- Sistemdeki toplam modem adetlerini tanımladığımız ve içinde modemin müsait mi olduğunu , ilgili processin bu modemi kullanmasını ve işi biten processin modemi serbest bırakmasını sağlayan kodların bulunduğu sınıf.

Yazıcı Service Sınıfı

- Sistemdeki toplam yazıcı adetlerini tanımladığımız ve içinde yazıcının müsait mi olduğunu , ilgili processin bu yazıcıyı kullanmasını ve işi biten processin yazıcıyı serbest bırakmasını sağlayan kodların bulunduğu sınıf.

CdDriver Service Sınıfı

- Sistemdeki toplam driver adetlerini tanımladığımız ve içinde driverın müsait mi olduğunu , ilgili processin bu driverı kullanmasını ve işi biten processin driverı serbest bırakmasını sağlayan kodların bulunduğu sınıf.

Memory Service Sınıfı

- Sistemdeki hafızayı simüle etmek için içerisinde 1024 elemanlı bir array bulunduran (arrayde 0 boş 1 dolu anlamına gelmektedir) user job ve realtime processler için maximum hafıza kapasitelerinin tanımlandığı içerisinde bestfit algoritmasını gerçekleyen, memory allocation ve memory release kodları bulunan sınıfımız.

Process Sınıfı

- Processe ait değerlerin (id, varış zamanı, öncelik, color vb) değişkenlerin tutulduğu, bu değişkenlere ait getter ve setter metodlarının oluşturulduğu sınıf.

Process Service Sınıfı

- Processi Çalıştırmaya, bitti mi diye kontrolunu sağlamaya, process hakkında durum ve hata mesajları yazdırmayı sağlayan kodların bulunduğu sınıf.

Dispatcher Sınıfı

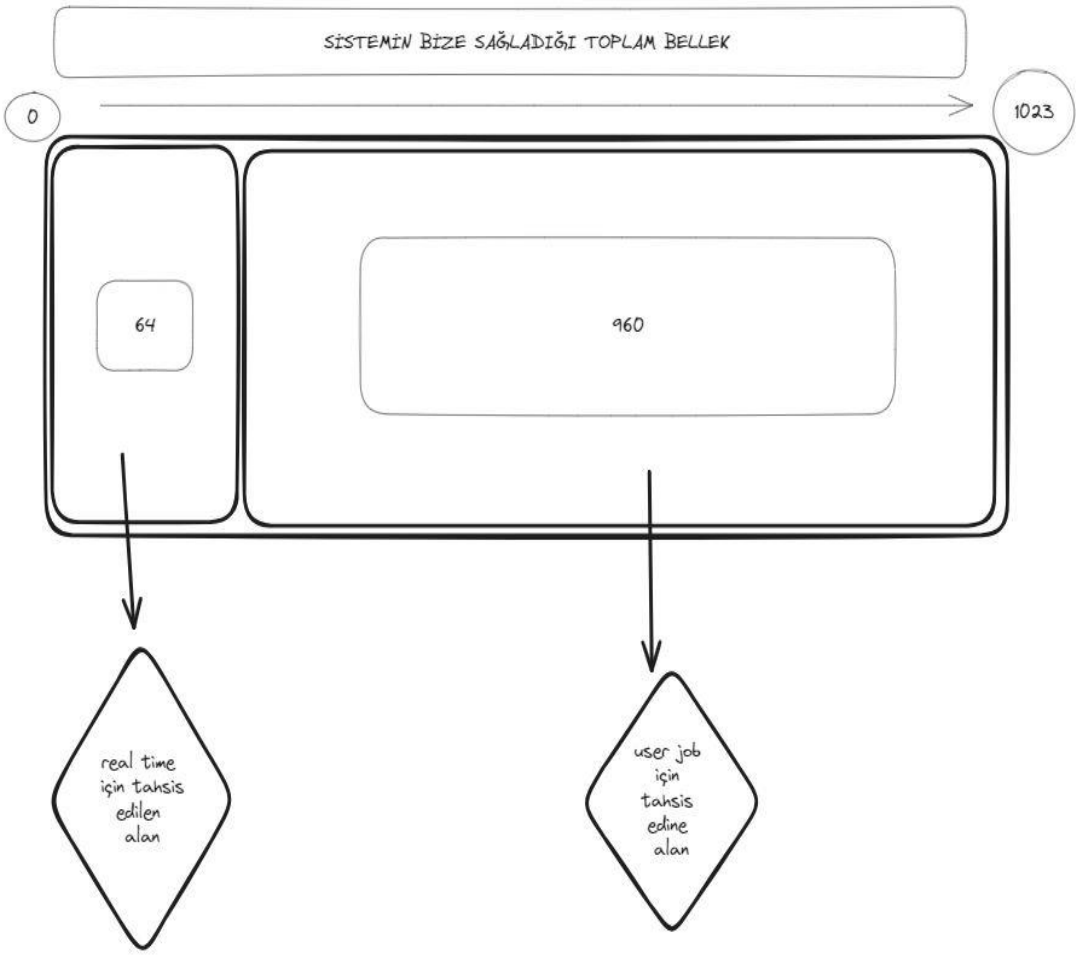
- Processlerin uygun kuyruklara alındığı bu kuyruklarda işlendiği bittiğinde kuyruktan atılıp gerektiğinde askıya alındığı kısacası kuyrukları yöneten projedeki diğer servisleri uygun yerlerde kullanıp çıktı üreten sınıf.

Bellek Tahsis Algoritmaları

- Sistemimizde her bir processe ait bellek alanı tanımlanmış ve process çalışırken processin bu bellek alanını işgal etmesi, process çalışmayı bitirdikten veya zaman aşımına uğradıktan sonra da işgal ettiği bu bellek alanını iade etmesi istenmiştir. Ödev dökümanında belirtildiği üzere realtime processler maximum 64mb user job processler ise maximum 960 mb yer kaplamaktadır. Bu nedenle hafızayı 64 ve 960 üzere 2 parçaya bölmeyi 64 lük kısmı realtime processler için 960 lık kısmı ise user job processler için ayırmayı uygun gördük.

- Bazen bellekteki toplam boş yer processi çalıştırmaya yetmesine rağmen bu alan bitişik olmadığı için process çalışamayabilmektedir buna bellek fragmantasyonu denir. Bu fragmantasyounu en aza indirmek için processi , processin kullanacağı mb boyutuna en yakın olan, processin sığabileceği bellek boşluğuna yerleştirmeyi karar kıldık. Bu işlemi Best Fit algoritması ile MemoryService sınıfımızda gerçekleştirdik.

Bellek Tahsisi Diyagramı



Görevlendiricinin Gerçekleştirilmesi

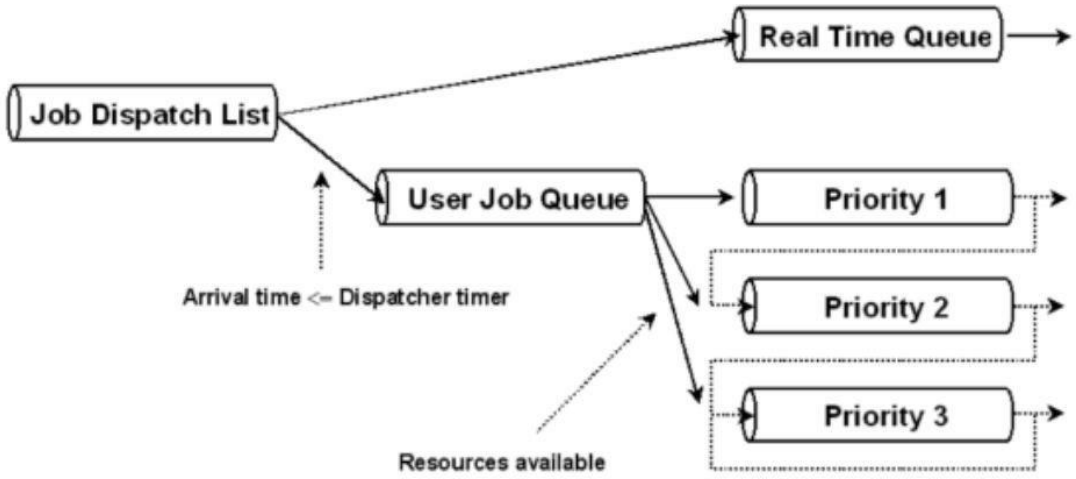
- Görevlendirici bünyesinde realtime queue , priority1 queue, priority2 queue, priority3 queue, userjob queue olmak üzere 5 kuyruk ve processlerin tutulduğu processlist isimli bir listeyi barındırır.

- Dispatcher sınıfımızda programıBaslat isimli bir fonksiyon tanımladık bu fonksiyon processlisten kuyruklara processlerin giriş işlemini başlatmaktadır. Programı hiçbir process kalmayıncaya kadar devam ettirdik. Bunun kontrolünü tümProcesslerTamamlandiMi isimli fonksiyonumuzla yaptık. Zamanı tutmak için counter isimli bir değişken tanımladık. Processlistten okuduğumuz processin varış zamanı eğer sistemin zamanından küçük veya eşit ise processi priority değişkenine göre uygun kuyruğa aldık. Önceliği 0 olan processleri realtime kuyruğuna diğer değişkenleri ise userjob kuyruğuna aldık. RealTime kuyruğuna bir process aldığımızda o an çalışan bir userjob process varsa askıya alınmasını sağladık. Bu processi 3.öncelikteyse 2. Önceliğe, 2.öncelikteyse 1.öncelik kuyruğuna aldık. User job processleri kuyruktan okuyarak sistemde anlık olarak çalışıp çalışamadıklarını bize gösteren sistemdeCalisabilirMi fonksiyonunu çağırdık. Eğer process sistemde çalışabiliyor ise processin isterlerine göre sistemdeki kaynakları işgal etmesini sağladık ve processi uygun öncelik kuyruğuna ekledik. Eğer bu process real time ise 64 mbdan fazla olam durumunda, userjob ise 960 mbdan fazla olma durumunda sistemden atılmasını ve hata mesajının yazdırılmasını sağladık.

- Processler kuyruklara girdikten sonra realtime queue dolu olması halinde oradaki processi çalıştırdık. Eğer dolu değilse 1. Öncelik doluysa 1.öncelik kuyruğunun başındaki processi çalıştırıp 2. Önceliğe gönderdik. Real time ve 1. Öncelik kuyruğunda process yok ise ve 2. Öncelik kuyruğu dolu ise 2.öncelik kuyruğunun başındaki processi işletip 3. Öncelik kuyruğuna gönderdik. Real time 1. Ve 2 . öncelik boş olduğu durumlarda ise 3. Öncelik kuyruğunda processlerin round robin prensibine uygun bir şekilde çalışmasını sağladık. Bunu 3. Öncelik kuyruğunun başındaki processi işletip kuyruktan çıkartıp tekrar kuruğa sokarak (kuyruk veri yapısında frontdan backe geçmesini sağlayarak) yaptık.

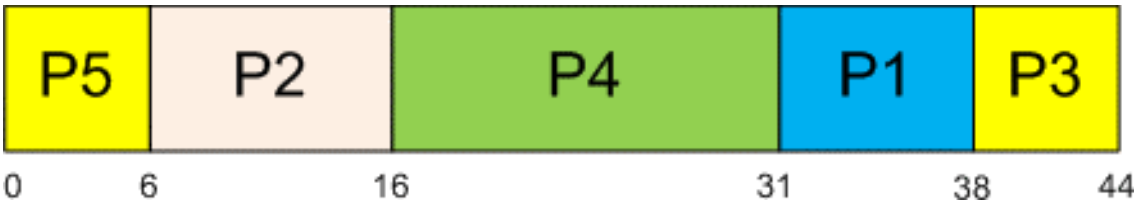
- Procesleri işlettikten sonra bitti mi diye kontrolunu yapıp bittiyse kaynakları geri iade etmelerini sağladık. Zaman aşımı yapan processleri sistemden atmak için her saniye processZamanAşımı fonksiyonunu çağırdık.

Görevlendirici Mantık Akışı

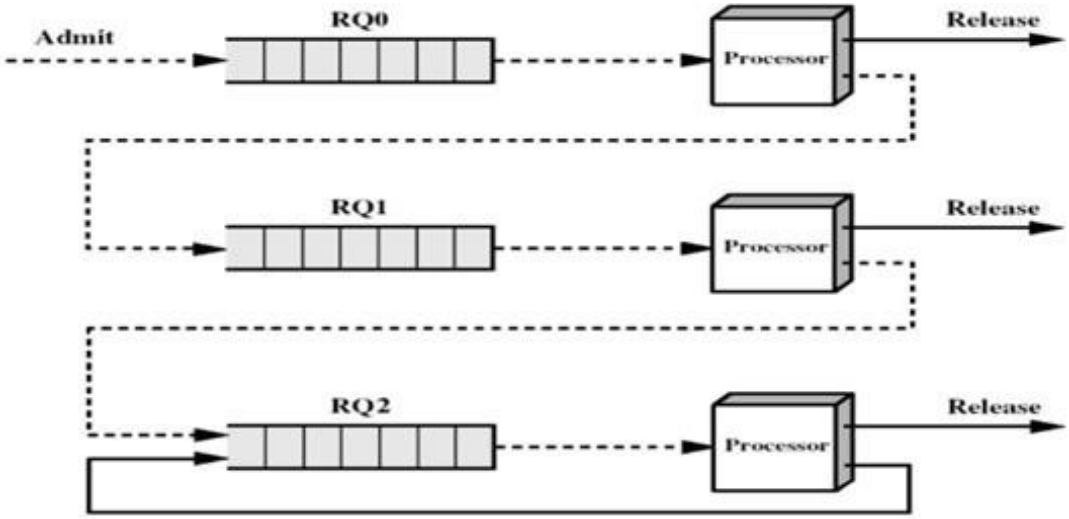


Görevlendirici Tarafından Kullanılan Yapılar

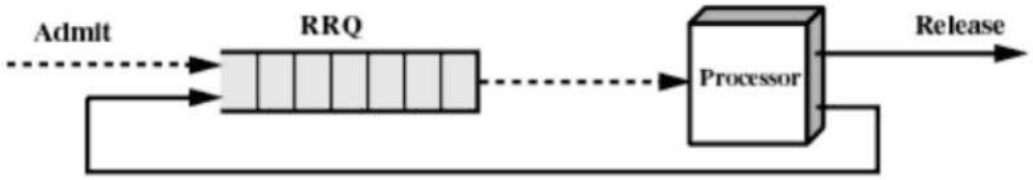
1. ***FCFS***



2.Üç Seviyeli Geri Beslemeli Görevlendirici

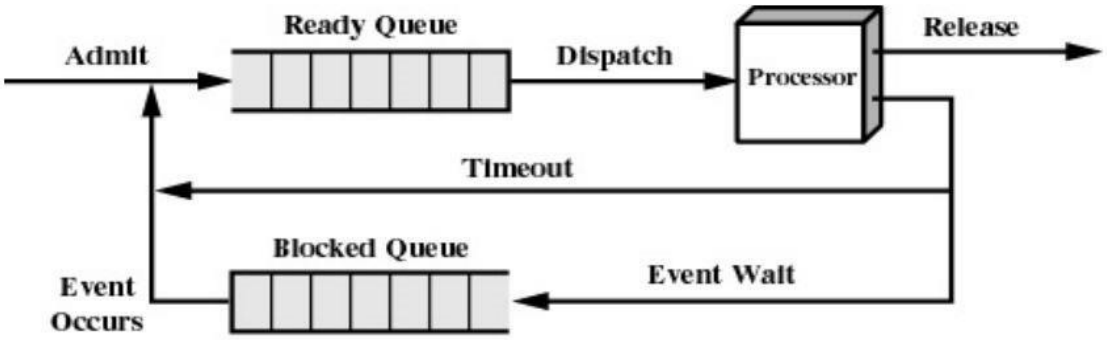


3.Raund Robin Görevlendiricisi

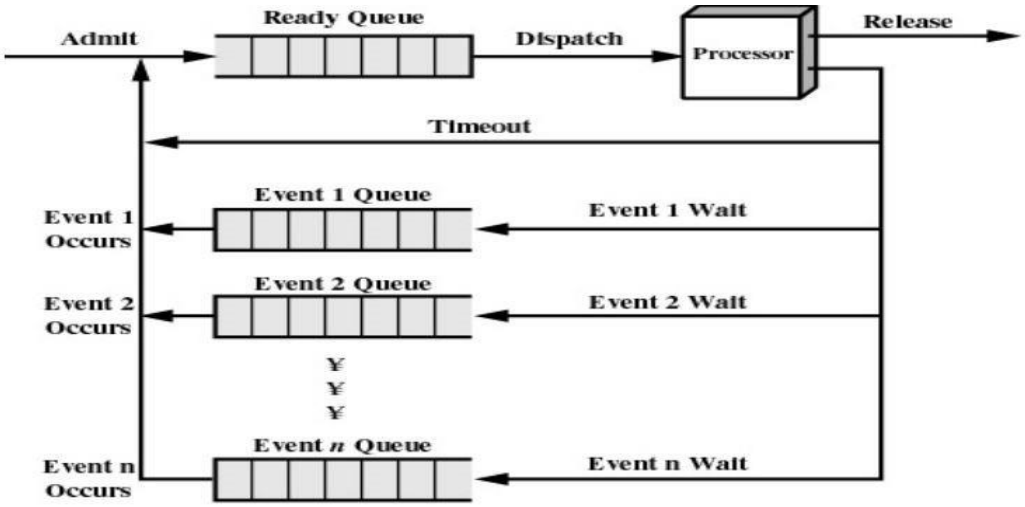


İşletim Sistemi Örnek Algoritmaları

Tekli Kuyruk



Çoklu Kuyruk



İşletim Sistemlerinin Çok Düzeyli Görevlendirici Kullanma Nedenleri

- ***Kaynak Kullanımının Etkinleştirilmesi*** : İşletim sistemleri, CPU, bellek, disk ve diğer donanım kaynaklarını etkili bir şekilde kullanmak için çok düzeyli görevlendirici kullanır. Bu sayede, birden fazla uygulama aynı anda çalışabilir ve kullanıcılar arasında kaynakların adil bir şekilde paylaşılması sağlanır.

- ***Verimlilik Artışı***: Çok düzeyli görevlendirme sayesinde, işletim sistemi, kullanıcının aynı anda birden fazla işi hızla tamamlamasına olanak tanır. Bu da üretkenliği artırır ve zaman kaybını en aza indirir.

- ***Öncelik Yönetimi***: İşletim sistemleri, farklı görevler arasında öncelik atama yeteneği sayesinde kritik işlevleri önceliklendirir. Örneğin, kullanıcının aniden bir dosya kaydetme işlemi yapması, diğer daha az önemli görevlerden öncelikli olarak işlenir.

Olası İyileştirmeler

Çok seviyeli görevlendiriciler ,modern işletim sistemlerinde sıkça kullanılan bir özelliktir ve birçok avantaja sahiptir. Ancak, bu özelliklerin bazı eksiklikleri ve zorlukları da vardır.

- ***Kaynak Kullanımının Artması***: Birden fazla görevi eş zamanlı olarak çalıştırmak, sistem kaynaklarına daha fazla yük bindirir. Özellikle bellek kullanımı ve CPU kapasitesi, çok düzeyli görevlendirmeyle birlikte artabilir. Bu, performansın düşmesine ve gecikmelerin yaşanmasına neden olabilir.

- **Öncelik Sorunları**: Çok sayıda görev eş zamanlı olarak çalıştırıldığında, önceliklendirme konusunda zorluklar ortaya çıkabilir. Kritik bir görevin, daha az kritik bir görev tarafından engellenmesi gibi durumlar yaşanabilir.

- Çok seviyeli görevlendiricilerde düşük öncelikli process real timeı veya ondan daha yüksek öncelikli bir processi beklediği için zaman aşımına uğrayabilir . Zaman aşımına uğramaması için zaman aşım limiti arttırılmalıdır.