SAKARYA
ÜNİVERSİTESİ

BİLGİSAYAR VE BİLİŞİM BİLİMLERİ FAKÜLTESİ

BİLGİSAYAR MÜHENDİSLİĞİ

İŞLETİM SİSTEMLERİ DERSİ PROJE ÖDEVİ

GRUP 13

HAZIRLAYANLAR

CEREN YILDIRIM - G201210062

SEDA MENCİK - G201210083

REFİK BURAK AKBAŞ - G201210050

KAMİL ŞİMŞEK - G201210377

ÇAĞLAR YILDIZ - G201210053

PROGRAMIN GENEL YAPISI

Program komut isteminden çalıştırılarak başlıyor. Bu program package içerisindeki Main.java' yı çalıştırıyor. Program çalıştığı anda .jar dosyası oluşturulan ve main.java içerisinde gerçek bir process olarak başlatılan Görevlendirici package' ındaki Main.java çalışıyor.

Bu processin başlamasıyla beraber asıl işlemlerin yapılacak olduğu kısıma geçiş yapılmış oluyor. İlk önce içerisinde kuyrukla ilgili tüm işlemlerin yapılacak olduğu Görevlendirici class'ından bir nesne oluşturuluyor. Daha sonra processlerle ilgili bilgileri tutan txt dosyasını okuması ve gerekli şekilde bölmesi için oku fonksiyonu ve bütün processleri Process listesine özellikleriyle beraber yerleştirdikten sonra artık gerekli kuyruk ve sıralama işlemleri yapılmak üzere Calis fonksiyonu çağırılıyor.

Process classında txt'den okunacak olan, processin durumunun kontrolü için gerekli olan property'ler ve manuel olarak atanan pid property'si tutuluyor. Dosyadan okunan processler için parametrelerin alındığı ve kontroller için kullanılan processleri kopyalayabilmek için kullanılması gereken olmak üzere iki adet kurucu fonksiyon mevcut. Ayrıca Process nesnesini kullanarak process çalıştığında gerçekleşmesi gereken olaylar için Calistir fonksiyonu mevcut.

Gorevlendirici.java projede bahsedilen 4 seviyeli görevlendirici olarak çalışmakta. Projede dosya yolu program dosyasıyla aynı olan "ornek.txt" dosyasından beslenen Okunan Processler List yapısı mevcut. Dosya sonu gelene kadar Oku fonksiyonunun içerisinde her satır için processe ait varış zamanı, öncelik ve patlama zamanı değerleri alınır, bu değerler Process classından türemiş process nesnesine ait özelliklere atandıktan sonra bu nesne Okunan Processler listesine eklenir.

Bütün Processler listeye atıldıktan sonra, diğer bir deyişle ornek.txt dosyasının sonuna gelindiğinde main.java dosyasında görevlendirici classına ait Calis fonksiyonu çağırılmıştı.

Calis fonksiyonu, her bir turu temel zamanlama kuantumu (q) olan 1 saniyeye karşılık gelen sonsuz bir döngüyle çalışıyor.

Bu fonksiyon içerisinde her turda öncelikle eğer okunan processler boş değilse Kuyruğa ekle fonksiyonu çağırılıyor.

KuyrugaEkle(): Process listesi dolaşılıyor. Eğer varış zamanı programın mevcut anına eşit olan bir process varsa öncelik özelliğine göre ilgili kuyruğa yerleştiriliyor ve okunan processler listesinden siliniyor.

Process'lerin çalıştırılacağı her algoritma fonksiyonunun içerisinde, process durumu ile ilgili ekrana bilgi verilebilmesi adına gerekli atamalar (islemOncesiProses, islemSonrasiProses,oncekiProses) gerçekleştirildi.

Calis fonksiyonunun devamında önceliği 0 olan yani gerçek zamanlı processlerin kontrolü yapılıyor. Eğer gerçek zamanlı kuyruğu boş değilse, yani sırada önceliği 0 olan bir process varsa her koşulda o anda o process FCFS sıralayıcısı çağırılarak çalıştırılıyor.

FCFS(): Sıradaki process için process classından Calistir fonksiyonu çağırılıyor. Daha sonra process'in çalışma süresinin (patlama zamanı) bitip bitmediği kontrol edilerek duruma göre process sonlandırılıyor (siliniyor).

Eğer gerçek zamanlı process kuyruğunda process mevcut değilse önceliği 1 olan processlerin bulunduğu kuyruk kontrol ediliyor. Eğer boş değilse yani sırada önceliği 1 olan bir kullanıcı processi varsa Geri Besleme fonksiyonu çağırılıyor.

GeriBesleme(): Öncelikle sıradaki ilgili process Calistir fonksiyonu ile çalıştırılıyor. Eğer process süresi (patlama zamanı) 0 değilse yani process bitmediyse; process önceliğine göre process öncelik 1 kuyruğundan silinip, öncelik değeri 1 artırılıp öncelik 2 kuyruğuna ekleniyor ya da process öncelik 2 kuyruğundan silinip, öncelik değeri 1 artırılıp öncelik 3 kuyruğuna ekleniyor. Böylece geri besleme mekanizması sağlanmış oluyor. Processin süresi dolmuşsa da process sonlandırılıyor (siliniyor).

Eğer process öncelik 1 kuyruğunda process mevcut değilse process öncelik 2 kuyruğu kontrol ediliyor. Boş değilse yani sırada önceliği 2 olan bir process varsa yine Geri Besleme fonksiyonu çağırılıyor ve yukarda açıklanmış olan işlemler gerçekleşiyor.

Process öncelik 2 kuyruğu da boş ise sıradaki process öncelik 3 kuyruğu kontrol ediliyor. Boş değilse Round Robin fonksiyonu çağırılıyor.

RoundRobin(): Baştaki process Calistir fonksiyonu çağırılarak çalıştırılıyor. Eğer process süresi (patlama zamanı) bitmemişse ilgili process kuyruğun başından silinip sonuna ekleniyor. Bu şekilde round robin mekanizması sağlanmış oluyor. Her processe ayrılan zaman dilimi, programın temel kuantalama zamanı olan 1 saniyeye eşit. Eğer process süresi bitmişse process sonlandırılıyor.

Bütün bu işlemlerden sonra eğer process öncelik 3 kuyruğu da boşsa tüm kuyruklar kontrol edilmiş olduğu için hiçbir kuyrukta process kalmamış oluyor. Bu koşulda da programın sonlandırıldığına dair bilgi ekrana yazıldıktan sonra sonsuz döngüden çıkılıp program sonlandırılıyor.

Program sonlanmamışsa her döngünün sonunda, processin çalıştığı andan itibaren 20 saniye geçtikten sonra zaman aşımına uğraması ile ilgili kontrol yapacak olan ZamanAsimiKontrol fonksiyonu çağırılıyor.

ZamanAsimiKontrol():Öncelik kuyruklarındaki bütün processler tek tek kontrol edilip zaman aşımına uğramış bir process mevcutsa, kuyruktan ilgili process siliniyor ve ekrana processin zaman aşımına uğradığına dair bilgi veriliyor.

Daha sonra her tur için hangi processle ilgili ne işlem yapıldıysa ekrana ilgili process bilgilerini ve yapılan işlemi belli koşulları kontrol ederek yazdıran Yazdır fonksiyonu çağırılıyor.

Yazdir(): Onceki ve islemOncekiProsesler kullanılarak gerekli koşullar saglanıyor ve process'lerin mevcut durumları (baslama, yürütülüyor, askida sonlandi), pid degerleri, öncelikleri ve kalan süreleri ekrana yazdiriliyor.

Her turdan önce program sayacı yani programın zaman değeri 1 artırılıyor.

ÇOK DÜZEYLİ GÖREVLENDİRİCİ ŞEMASININ KULLANIM NEDENLERİ VE EKSİKLİKLERİ

Çok düzeyli bir görevlendirici işletim sistemlerinde kritik veya daha önemli işlerin daha önce yapılmasını sağlar ve bu da sistem etkinliğini artırır. Ayrıca her işleme sıra gelmesine olanak tanır. Ancak işler arasında sık geçiş yaşandığında bilgisayar sürekli iş değiştirmeye de zaman harcar ve sistem performansı düşer. Ayrıca görevlere öncelik atanırken hata yapılması işlerin yanlış önceliklere sahip olmasına neden olabilir.

Gerçek işletim sistemlerinde kullanılan şemalarla karşılaştıracak olursak, örneğin:

-Tek düzeyli görevlendirici şeması: İşler önceliklerine göre ayrılmaz ve FCFS algoritması ile çalışır. Bu yüzden bazı kritik veya daha önemli işler daha yavaş işlenir ve sistem etkinliği düşer.

-CFS(Completely Fair Scheduler):CFS görecleri CPU kullanımına göre öncelikli olarak sıralayarak işleri eşit şekilde paylaştırır. İhtiyaca göre çok düzeyli görevlendirici şeması veya CFS tercih edilebilir.

-FCFS(First-Come First-Served):Bu görevlendirici şema, işlemleri geliş sırasına göre işler. İlk gelen ilk çalışır mantığıyla çalışır. Çok düzeyli görevlendirici

BELLEK VE KAYNAK AYIRMA ŞEMALARI

Program esnasında process çalışırken daha yüksek öncelikli bir process geldiyse, mevcut process bellek ve kaynağını daha yüksek öncelikli process için serbest bırakır.

Genel manada Görevlendirici class'ında kullanılan yapının kompleks olması eksikliklerden biridir. Bu konuda yapılan temiz kod standartları ile anlaşılır bir hale getirilmiş ve bağımlılıklar azaltılarak yapı kurulmaya çalışılmıştır.