T.C.
SAKARYA ÜNİVERSİTESİ
BİLGİSAYAR VE BİLİŞİM BİLİMLERİ FAKÜLTESİ
BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ

Ders	**:**	**İşletim Sistemleri**	
Dönem	**:**	**2022-2023 Güz Dönemi**	
Grup No	**:**	**51**	
Konu	**:**	**Görevlendirici Kabuğu (Dispatcher Shell)**	
Grup Üyeleri	**:**	**Muhammet Kemal Güvenç**	**B181210076**
		Hakan Kırık	**B201210370**
		Yasin Emin Esen	**B211210386**
		Alptekin Ocakdan	**G181210385**
Github	**:**	**https://github.com/kemalguvenc/DispatcherShell_Grup51**	

Proje Aşamaları

Tasarım

Projenin tasarımını yapmaya başlarken derste anlatılanlar ile yabancı kaynaklardaki bilgileri karşılaştırarak oluşturmaya başladık. İlk aşama olarak bilgi topladık. Görevlendirici nedir? İlk Gelen İlk Çalışır Algoritması nedir? Çevrimsel Sıralı algoritma nedir?

Görevlendirici (Dispatcher)

İşletim sistemleri bağlamında, görevlendirici, işletim sisteminin çekirdeğinin, işlemciyi (veya işlemcileri) yürütülmeyi bekleyen farklı işlemlere veya iş parçacıklarına tahsis etmekten sorumlu olan bir parçasıdır. Görevlendirici, bu süreçlerin veya iş parçacıklarının yürütülmesini programlamaktan ve herhangi bir zamanda işlemcinin kontrolünün hangisine verilmesi gerektiğini belirlemekten sorumludur.

Görevlendirici, hangi işlemin veya iş parçacığının daha sonra yürütülmesi gerektiğine karar vermek için çeşitli algoritmalar kullanır. Bu algoritmalar, sürecin önemi, yürütülmesi için beklediği süre veya ihtiyaç duyduğu kaynak miktarı gibi farklı faktörlere öncelik verecek şekilde tasarlanabilir.

Görevlendirici, tüm süreçlerin ve iş parçacıklarının zamanında ilerleme kaydetmesini sağlamada kritik bir rol oynadığı için işletim sisteminin önemli bir parçasıdır. Aynı zamanda, işletim sisteminin kaynaklarının verimli bir şekilde kullanılmasını sağlamaktan da sorumludur, böylece birden çok işlem ve iş parçacığı, performans sorunlarına neden olmadan aynı anda çalışabilir.

İlk Gelen İlk Çalışır (FCFS)

İlk gelen ilk çalışır alır (FCFS) algoritması, hazır kuyruğuna ulaşma sırasına göre her işleme bir öncelik atayan bir zamanlama algoritmasıdır. Bu algoritma altında, en uzun süre bekleyen işleme en yüksek öncelik verilir ve bir sonraki yürütülecek işlemdir.

FCFS algoritmasını uygulamak için, işletim sistemi yürütülmeye hazır bir işlem sırası tutar. Yeni bir işlem geldiğinde kuyruğun sonuna eklenir. Gönderici daha sonra kuyruğun önündeki işlemi seçer ve işlemciyi buna tahsis eder. İşlem yürütmeyi tamamladığında, sıradan kaldırılır ve sıradaki bir sonraki işlem, çalışması programlanan işlem olur.

FCFS algoritması basit ve uygulanması kolaydır, ancak özellikle sıranın önündeki işlemler çok fazla kaynak gerektiriyorsa veya yüksek bir CPU patlama süresine sahipse, kuyruğa sonradan gelen işlemler için uzun bekleme sürelerine yol açabilir. Sonuç olarak, FCFS algoritması, gerçek zamanlı sistemler veya yüksek derecede öngörülebilirlik gerektiren sistemler için her zaman en iyi seçim değildir.

Çevrimsel Sıralı (Round Robin)

Çevrimsel Sıralı (RR) algoritma, bir işletim sisteminde işlemlerin yürütülmesini programlamak için kullanılan bir zamanlama algoritmasıdır. Önleyici bir algoritmadır, yani

işletim sistemi o anda çalışmakta olan bir işlemi kesebilir ve bunun yerine çalışacak farklı bir işlemi programlayabilir.

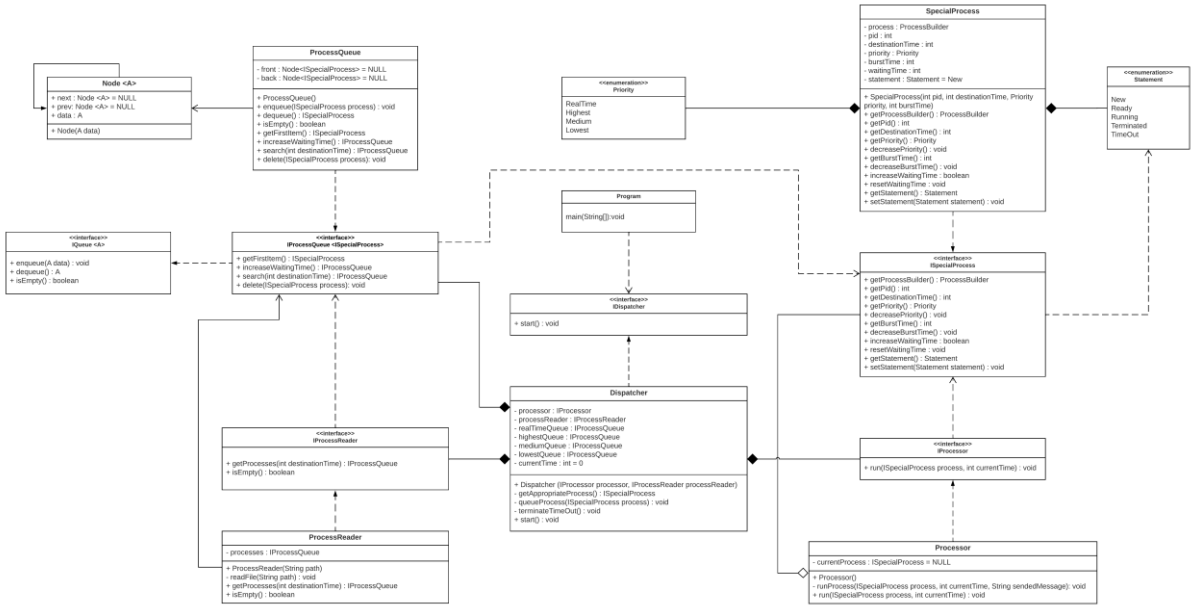
RR algoritması altında, işletim sistemi yürütülmeye hazır bir işlem sırası tutar. Yeni bir işlem geldiğinde kuyruğun sonuna eklenir. Görevlendirici daha sonra kuyruğun önündeki işlemi seçer ve işlemciyi buna tahsis eder. Sürecin kesintiye uğramadan ve kuyruğun sonuna geri eklenmeden önce, zaman kuantumu adı verilen sabit bir süre boyunca çalışmasına izin verilir. Sıradaki bir sonraki işleme daha sonra çalıştırma fırsatı verilir.

Bu işlem kuyruktaki tüm işlemler tamamlanana kadar devam eder. Zaman kuantumu, süreçlerin programlanması için çok uzun süre beklemek zorunda kalmamasını sağlamak için tipik olarak birkaç milisaniye gibi küçük bir değere ayarlanır.

RR algoritması, tek bir işlemin işlemciyi tekelleştirmesini önlemede etkilidir, çünkü her işlemin kesintiye uğramadan önce yalnızca kısa bir süre çalışmasına izin verilir. Bu, yürütülmesi gereken birçok işlemin olduğu, ancak tam olarak yürütüldükleri sıranın önemli olmadığı ortamlar için iyi bir seçim olabilir. Ancak, sürekli kesintiye uğrayan ve devam eden süreçlerin ek yükü, bazı durumlarda RR algoritmasını diğer algoritmalardan daha az verimli hale getirebilir.

Diyagram

Bu bilgiler ışığında yapacağımız projenin ilk önce diyagramını oluşturmaya karar verdik.



Program

Priority

Bu Priority enum, Dispatcher sınıfındaki bir işlemin önceliğini belirlemek için kullanılıyor. Dört olası değeri vardır: RealTime, Highest, Medium ve Lowest. Önceliğine göre hangi kuyruğa işlem ekleneceğini belirlemek için queueProcess yönteminde kullanılır.

Statement

Statement enum, bir işlemin içinde olabileceği durumları temsil eder. Aşağıdaki değerlere sahiptir:

New: Bu, sürecin yeni oluşturulduğunu gösterir.

Ready: Bu, işlemin yürütülmeye hazır olduğunu gösterir.

Running: Bu, işlemin şu anda yürütülmekte olduğunu gösterir.

Terminated: Bu, işlemin yürütmeyi sonlandırdığını veya tamamladığını gösterir.

TimeOut: Bu, işlemin zaman aşımına uğradığını, yani yürütme için izin verilen süre sınırını aştığını gösterir.

Statement enum, bir işlemin mevcut durumunu takip etmek için SpecialProcess sınıfında kullanılır. SpecialProcess sınıfının getStatement () ve setStatement (Statement statement) yöntemleri, sırasıyla bir işlemin Statement almanıza ve güncellemenize olanak tanır.

Node <A>

Node çift bağlantılı listedeki düğümü temsil eden bir sınıftır. Node sınıfının üç üye değişkeni vardır:

next: Bu değişken, listedeki sonraki düğüme bir referans tutar. Bu, listedeki son düğümse, sonraki boş olacaktır.

prev: Bu değişken, listedeki önceki düğüme bir referans tutar. Bu, listedeki ilk düğümse, önceki boş olacaktır.

data: Bu değişken, düğümle ilişkili verileri tutar. Verilerin tipi tip parametresi A ile belirtilir.

Node sınıfı, A türünde tek bir bağımsız değişken alan ve onu veri üyesi değişkenine atayan tek bir oluşturucuya sahiptir. Sonraki ve önceki üye değişkenlerinin her ikisi de null olarak başlatılır.

Bir Node nesnesi oluşturulduktan sonra, sonraki ve önceki üye değişkenlerini diğer Node nesnelerine referans verecek şekilde ayarlayarak çift bağlantılı bir listenin parçası olarak kullanılabilir.

IQueue<A>

Bu kod, kuyruk veri yapısını temsil eden IQueue adlı bir interfacei tanımlar. IQueue interfaceinde, kuyrukta depolanabilecek nesnelerin türünü belirten tek bir tür parametresi A vardır. İnterface üç yöntemi tanımlar:

enqueue(A data): Bu metot kuyruğun sonuna bir öğe ekler.

dequeue(): Bu yöntem, kuyruktaki ilk öğeyi kaldırır ve döndürür.

isEmpty(): Bu yöntem, kuyruğun boş olup olmadığını gösteren bir boolean değeri döndürür.

ISpecialProcess

Bu kod, özel bir işlemi temsil eden ISpecialProcess adlı bir interfacei tanımlar. ISpecialProcess interfacei, süreç hakkında bilgi sağlayan ve çeşitli şekillerde manipüle edilmesine izin veren birkaç yöntem tanımlar. Bu yöntemler şunları içerir:

getProcessBuilder(): Bu yöntem, işlemi başlatmak ve yapılandırmak için kullanılabilecek bir ProcessBuilder nesnesi döndürür.

getPid(): Bu yöntem, işlemin işlem kimliğini döndürür.

getDestinationTime(): Bu metot işlemin varış zamanını döndürür. Bu koddan varış zamanının neyi temsil ettiği açık değildir.

getPriority(): Bu metot işlemin önceliğini döndürür.

decreasePriority(): Bu yöntem, işlemin önceliğini azaltır.

getBurstTime(): Bu metot işlemin patlama zamanını döndürür. Patlama süresi, işlemin işini tamamlaması için gereken süredir.

reduceBurstTime(): Bu yöntem, işlemin patlama süresini 1 azaltır.

increaseWaitingTime(): Bu metot işlemin bekleme süresini 1 arttırır. Eğer bekleme süresi 20 olursa bu metot true değerini döndürür.

resetWaitingTime(): Bu metot işlemin bekleme süresini 0 olarak resetler.

getStatement(): Bu yöntem, işlemle ilişkili bir Deyim nesnesi döndürür.

setStatement(Statement statement): Bu yöntem, işlemle ilişkili Beyan nesnesini ayarlar.

SpecialProcess

SpecialProcess sınıfı, ISpecialProcess interfaceinin bir uygulamasıdır. Hedef zamanı, öncelik ve patlama zamanı gibi özel özelliklere sahip bir süreci temsil eder.

Sınıfın, bir SpecialProcess nesnesinin üye değişkenlerini verilen parametrelerle başlatan bir yapıcısı vardır. Ayrıca, işlemi başlatmak için “kullanılacak” belirtilen komutla yeni bir ProcessBuilder nesnesi oluşturur. SpecialProcess sınıfı, ISpecialProcess interfaceinde tanımlanan yöntemleri şu şekilde uygular:

getProcessBuilder(): Bu yöntem, SpecialProcess nesnesiyle ilişkili ProcessBuilder nesnesini döndürür.
getPid(): Bu yöntem, SpecialProcess nesnesinin işlem kimliğini döndürür.

getDestinationTime(): Bu metot, SpecialProcess nesnesinin varış zamanını döndürür.

getPriority(): Bu yöntem, SpecialProcess nesnesinin önceliğini döndürür.

decreasePriority(): Bu yöntem, SpecialProcess nesnesinin önceliğini bir düzey azaltır. Geçerli öncelik Highest ise, Medium olarak değiştirilir. Geçerli öncelik Medium ise, Lowest olarak değiştirilir.

getBurstTime(): Bu yöntem, SpecialProcess nesnesinin patlama zamanını döndürür.

reduceBurstTime(): Bu metod SpecialProcess nesnesinin burst süresini 1 azaltır. Burst time zaten 0 ise 0 olarak kalır.

increaseWaitingTime(): Bu metot, SpecialProcess nesnesinin bekleme süresini 1 arttırır. Bekleme süresi 20'den büyük veya ona eşit olursa metot true, aksi takdirde false döndürür.

resetWaitingTime(): Bu metod, SpecialProcess nesnesinin bekleme süresini 0 olarak ayarlar.

getStatement(): Bu yöntem, işlemin geçerli durumunu gösteren SpecialProcess nesnesinin Statement'ını döndürür.

setStatement(Statement statement): Bu yöntem, SpecialProcess nesnesinin Bildirimini verilen ifade parametresine ayarlar.

IProcessQueue <ISpecialProcess>

Bu kod, IQueue interfaceini genişleten ve birkaç ek yöntem ekleyen IProcessQueue adlı bir interfacei tanımlar. Java'daki bir interface, bir sınıfın uygulayabileceği bir dizi ilgili yöntemi tanımlayan bir türdür. Bir interface uygulayarak, bir sınıf interfacede tanımlanan tüm yöntemleri uygulamayı kabul eder.

IQueue interfaceinde, kuyrukta depolanabilecek nesnelerin türünü belirten ISpecialProcess adlı tek bir tür parametresi vardır. IProcessQueue interfacei, IQueue'yi genişletir ve dört ek yöntem ekler:

getFirstItem(): Bu metot kuyruktaki ilk öğeyi döndürür.

increaseWaitingTime(): Bu metot tüm işlemlerin bekleme süresini arttırır.
sıradaki işlemleri 1 ile sıralar ve bekleme süresi 20 saniyeden uzun olan tüm işlemleri içeren bir sıra döndürür.

search(int destinationTime): Bu yöntem, belirli bir varış saatine sahip bir işlem için kuyruğu arar ve bu tür tüm işlemleri içeren bir sıra döndürür.

delete(ISpecialProcess process): Bu yöntem, belirtilen bir işlemi sıradan kaldırır.

ProcessQueue

ProcessQueue sınıfı, ISpecialProcess nesnelerini depolayan bir kuyruk veri yapısıdır. Node sınıfı kullanılarak çift bağlantılı bir liste olarak uygulanır. Sınıfın, front ve back üye değişkenlerini null olarak başlatan ve listenin boş olduğunu belirten bir varsayılan oluşturucusu vardır. ProcessQueue sınıfı, IProcessQueue interfaceinde tanımlanan yöntemleri şu şekilde uygular:

enqueue(ISpecialProcess process): Bu metot, yeni bir Node nesnesi oluşturup onu listeye ekleyerek kuyruğun sonuna yeni bir ISpecialProcess nesnesi ekler. Liste boşsa, yeni düğüm listedeki hem ilk hem de son düğüm olur. Aksi takdirde, yeni düğüm mevcut son düğümden sonra eklenir ve yeni son düğüm olur.

dequeue(): Bu yöntem, listedeki ilk Node silerek kuyruktaki ilk ISpecialProcess nesnesini kaldırır ve döndürür. İlk düğüm silindikten sonra liste boşalırsa, front ve back üye değişkenlerinin her ikisi de null olarak ayarlanır. Aksi takdirde, listedeki ikinci düğüm yeni ilk düğüm olur ve bir sonraki üyesi null olarak ayarlanır.

isEmpty(): Bu yöntem, kuyruğun boş olup olmadığını gösteren bir boolean değeri döndürür. Front üye null ise kuyruk boş kabul edilir.

getFirstItem(): Bu yöntem, kuyruktaki ilk ISpecialProcess nesnesini döndürür.

increaseWaitingTime(): Bu metot, kuyruktaki tüm süreçlerin bekleme süresini 1 arttırır ve bekleme süresi 20 saniyeden fazla olan tüm süreçleri içeren yeni bir ProcessQueue nesnesi döndürür. Bunu, ISpecialProcess nesneleri listesinde gezinerek ve her biri için increaseWaitingTime() yöntemini çağırarak yapar. Bir işlemin bekleme süresi 20 saniyeyi geçerse yeni kuyruğa eklenir.

search(int destinationTime): Bu yöntem, kuyrukta verilen destinationTime parametresine eşit bir hedef zamana sahip ISpecialProcess nesneleri arar. Arama ölçütleriyle eşleşen tüm işlemleri içeren yeni bir ProcessQueue nesnesi döndürür. Yöntem bunu, ISpecialProcess nesnelerinin listesini geçerek ve her birinin varış zamanını kontrol ederek yapar. Bir işlemin hedef zamanı, verilen destinationTime eşitse, yeni kuyruğa eklenir.

delete(ISpecialProcess process): Bu yöntem, belirli bir ISpecialProcess nesnesini sıradan kaldırır. Bunu, ISpecialProcess nesneleri listesinde gezinerek ve verilen işlem parametresiyle eşleşeni arayarak yapar. Eğer işlem bulunursa çevredeki düğümlerin prev ve next üye değişkenleri güncellenerek listeden silinir. Silinen işlem listedeki ilk veya son düğüm ise, front veya back üye değişkeni buna göre güncellenir.

IProcessReader

IProcessReader interfacei, belirli bir varış saatine ulaşan işlemleri okumaktan ve döndürmekten sorumlu bir nesne olan bir işlem okuyucusunu temsil eder. Aşağıdaki yöntemlere sahiptir:

getProcesses(int destinationTime): Bu yöntem, verilen destinationTime'a ulaşan işlemlerin bir sırasını döndürür.

isEmpty(): Bu metot, gelecekte okunacak başka proses olup olmadığını gösteren bir boolean değeri döndürür.

ProcessReader

Bu, IProcessReader interfaceini uygulayan ve bir dosyadan özel işlemlerde okumak ve belirli bir zamana eşit bir hedef zamanı olan herhangi bir özel işlemi döndürmek için bir yöntem sağlayan bir sınıftır. ProcessReader sınıfı, dosyadan okunan özel işlemleri depolamak için kullanılan bir IProcessQueue nesnesi olan processes özel alanına sahiptir. readFile yöntemi, belirtilen yoldan bir dosyada okur ve dosyadaki her satır için, satırın içerdiği bilgilerle yeni bir ISpecialProcess nesnesi oluşturur ve onu processes kuyruğuna ekler. getProcesses yöntemi, processes kuyruğunda verilen zamana eşit bir hedef zamana sahip özel işlemler arar, bunları processes kuyruğundan kaldırır ve yeni bir IProcessQueue nesnesinde döndürür. isEmpty yöntemi, processes kuyruğunda herhangi bir özel işlem kalıp kalmadığını belirten bir boolean değeri döndürür.

IProcessor

IProcessor interfacei, bağımsız değişken olarak bir ISpecialProcess nesnesi ve bir currentTime alan run adlı tek bir yöntem tanımlar ve verilen işlemi çalıştırır.

Processor

Bu, IProcessor interfaceinin bir uygulamasıdır. "İşlemci" sınıfı, şu anda çalışan işlemi depolayan currentProcess adlı bir örnek değişkene sahiptir.

runProcess yöntemi, bir SpecialProcess nesnesi, geçerli zamanı temsil eden bir integer ve konsolda görüntülenecek bir mesajı temsil eden bir string alır. Verilen bilgilerle biçimlendirilmiş bir dize oluşturur ve ardından SpecialProcess nesnesinin ProcessBuilder nesnesini kullanarak yeni bir işlem başlatır. İşlem, process.jar adlı bir jar dosyasını çalıştırır ve biçimlendirilmiş dizgiyi bir bağımsız değişken olarak iletir.

Çalıştır yöntemi, belirli bir SpecialProcess nesnesini çalıştırmak için kullanılır. Verilen process null ise ve o anki process Terminated deyimine sahipse, kuyruktaki tüm processler tamamlanmış demektir ve bu bilgiyi konsolda görüntülemek için runProcess methodu çağrılır. Verilen process'in TimeOut ifadesi varsa, bu process'in zaman sınırını aştığı anlamına gelir ve bu bilgiyi konsolda görüntülemek için runProcess methodu çağrılır. currentProcess örnek değişkeni null ise, Processor nesnesinin henüz herhangi bir işlemi çalıştırmadığı ve verilen işlemin currentProcess olarak ayarlandığı ve patlama süresini azaltmak için decreaseBurstTime yöntemi çağrıldığı anlamına gelir. Verilen işlem currentProcess ile aynı değilse Processor nesnesi yeni bir işleme geçiyor demektir. Bu durumda runProcess methodu

currentProcess ile çağrılarak sonlandırıldığını veya hazır olduğunu gösterir ve currentProcess verilen process'e set edilerek runProcess methodu yeni currentProcess ile çağrılır. Çalışmaya başladığını göstermek için. Verilen process currentProcess ile aynı ise Processor nesnesi aynı process'i çalıştırmaya devam ediyor demektir ve runProcess methodu currentProcess ile çağrılarak hala çalışmakta olduğunu gösterir. Her durumda, patlama süresini azaltmak için currentProcess'in decreaseBurstTime yöntemi çağrılır.

IDispatcher

IDispatcher interfacei, işlemci tarafından yürütülecek işlemleri gönderme sürecini başlatmak için kullanılan "start" adlı bir yöntemi belirtir.

Dispatcher

Bu kod, IDispatcher interfaceini uygulayan Dispatcher sınıfını tanımlar. Bu sınıf, IProcessor interfaceini kullanarak süreçlerin planlanmasından ve çalıştırılmasından sorumludur.

Dispatcher sınıfı, işlemleri önceliklerine göre depolamak için dört kuyruğa ve geçerli zamanı takip etmek için bir currentTime değişkenine sahiptir. Ayrıca, işlemleri çalıştırmak için kullanılacak IProcessor nesnesine bir referansı tutmak için bir işlemci alanına ve bir dosyadan süreçleri okumak için kullanılacak IProcessReader nesnesine bir referansı tutmak için bir processReader alanına sahiptir.

Dağıtıcıyı başlatmak için start() yöntemi kullanılır. Sürekli olarak aşağıdakileri yapan sonsuz bir döngüye sahiptir:

1. processReader nesnesinden geçerli zamanda gelen tüm işlemleri alın.
2. Alınan işlemleri uygun öncelik sırasına göre sıralayın.
3. Öncelik sıralarında 20 saniyeden uzun süredir bekleyen işlemleri kontrol edin ve sonlandırın.
4. Önceliğe göre çalıştırmak için uygun işlemi alın.
5. Çalıştırılacak başka işlem yoksa, döngüden çıkın.
6. İşlemci nesnesini kullanarak işlemi çalıştırın.
7. İşlem çalışmayı bitirdiyse durumunu "sonlandırıldı" olarak ayarlayın.
8. CurrentTime'ı 1 saniye artırın.

queueProcess, bir ISpecialProcess nesnesini bağımsız değişken olarak alır ve önceliğine göre uygun kuyruğa yerleştirir. İşlemin önceliği RealTime ise realTimeQueue içine alınır. Öncelik Highest ise, highestQueue yerleştirilir ve böyle devam eder. Öncelik RealTime , Highest veya Medium değilse, Lowest olduğu varsayılır ve lowestQueue yerleştirilir.

terminatedTime en yüksek, orta ve en düşük öncelikli kuyruklardaki işlemlerin bekleme sürelerini kontrol etmek ve bekleme süresi 20 saniyeyi geçen işlemleri sonlandırmakla görevlidir. Bunu, öncelik sıralarının her birinde increaseWaitingTime() yöntemini çağırarak ve ardından döndürülen tüm işlemleri kuyruktan çıkarıp sonlandırarak yapar."increaseWaitingTime() yönteminin, bekleme süresi 20 saniyeyi geçen işlemlerden oluşan bir sıra döndürmesi beklenir.

getAppropriateProcess, öncelik sıralarından en uygun işlemi alır. Gerçek zamanlı kuyruk boş değilse, gerçek zamanlı kuyruğun ilk öğesini döndürür. En yüksek öncelikli sıra boş değilse, en yüksek öncelikli sıranın ilk öğesini sıradan çıkarır ve geri döndürür. Orta öncelikli sıra boş değilse, orta öncelikli sıranın ilk öğesini sıradan çıkarır ve geri döndürür. En düşük öncelikli sıra boş değilse, en düşük öncelikli sıranın ilk öğesini sıradan çıkarır ve geri döndürür. Tüm öncelik sıraları boşsa, null değerini döndürür.

Program

Bu kod, bir işletim sisteminde bir işlem zamanlayıcıyı simüle eden bir programın ana yöntemidir. Program, işlemler hakkında bilgi içeren bir dosyayı okur ve ardından bu işlemleri yönetmek için "Dispatcher" sınıfının bir örneğini kullanır.

Ana yöntem önce "processReader" adlı bir "ProcessReader" örneği oluşturur ve yapıcıya ilk komut satırı bağımsız değişkenini (args[0]) iletir. Bu sınıf, süreçler hakkında bilgi içeren dosyadaki okumadan sorumludur.

Ardından, "processor" adı verilen bir "Processor" örneği oluşturulur. Bu sınıf, süreçlerin yürütülmesinden sorumlu olacaktır.

Son olarak, "processor" ve "processReader" örneklerini yapıcıya argüman olarak ileterek "dispatcher" adlı bir "Dispatcher" örneği oluşturulur.

Ardından, dosyadan okunan işlemleri yönetme ve planlama sürecini başlatan "dispatcher" örneğinin "start" yöntemi çağrılır.

Tartışma

Hem Linux hem de Windows'ta, işletim sisteminin çekirdeği, daha sonra hangi işlemin yürütüleceğini seçmekten sorumlu olan bir görevlendirici içerir. Bir işlem çalışmaya hazır olduğunda, hazır işlemler kuyruğuna alınır. Görevlendirici, kuyruğun önündeki işlemi seçer ve CPU'yu bu işleme tahsis eder. Bu işlem, yürütmeyi bitirene veya bazı kaynakları beklerken bloke edilene kadar devam eder. İşlem yürütmeyi bitirdiğinde veya bloke edildiğinde, dağıtıcı kuyruktaki bir sonraki işlemi seçer ve CPU'yu buna tahsis eder.

Linux ve Windows çekirdekleri arasındaki temel farklardan biri, Linux'un tamamen önleyici bir zamanlayıcı kullanmasıdır; bu, çekirdeğin, CPU'yu farklı bir işleme vermek için herhangi bir zamanda çalışan bir işlemi kesebileceği anlamına gelir. Buna karşılık, Windows çekirdeği yalnızca kısmen önleyicidir, yani bir işlemi yalnızca belirli koşullar altında, örneğin

işlemin bir sistem çağrısını engellemesi veya işlemin kendisine ayrılan zaman dilimini kullanması durumunda kesintiye uğratabileceği anlamına gelir.

Linux ve Windows tarafından kullanılan zamanlama algoritmalarında da farklılıklar vardır. Linux, Tamamen Adil Zamanlayıcı (CFS) ve Round Robin Zamanlayıcı dahil olmak üzere çeşitli zamanlama algoritmaları kullanırken Windows, önceliğe dayalı zamanlama ve zaman dilimlemenin bir kombinasyonunu kullanır.

Bununla beraber bizim yazdığımız Dispatcher sınıfı, simüle edilmiş bir işletim sisteminde işlemlerin yürütülmesini yönetmekten sorumlu görünüyor. Bunu, farklı önceliklere sahip (gerçek zamanlı, en yüksek, orta ve en düşük) işlemler için dört sıra tutarak yapar.

Start yöntemi sonsuz bir döngüde çalışır ve çalışacak yeni işlemleri sürekli kontrol eder, 20 saniyelik bekleme süresi sınırını aşan işlemleri sonlandırır ve çalışmaya hazır işlemleri yürütür. Bunu yapmak için, yeni işlemleri kontrol etmek için processReader nesnesini, bekleme süresi sınırını aşan işlemleri sonlandırmak için terminateTimeOut yöntemini ve çalışmaya hazır işlemleri yürütmek için processor nesnesini kullanır.

QueueProcess yöntemi, bir işlemi önceliğine göre uygun kuyruğa yerleştirir ve getAppropriateProcess yöntemi, kuyrukları belirli bir sırayla kontrol ederek çalıştırılacak bir sonraki işlemi alır: gerçek zamanlı, en yüksek, orta ve en düşük.

Bu şekilde Dispatcher sınıfı, simüle edilmiş işletim sisteminde süreçlerin yürütülmesini planlamaktan ve yönetmekten sorumludur. İşlemlerin yürütülmesini önceliklerine göre önceliklendirir ve işlemlerin çalıştırılmadan önce çok uzun süre beklememesini sağlar.