T.C.

SAKARYA ÜNİVERSİTESİ

BİLGİSAYAR VE BİLİŞİM BİLİMLERİ FAKÜLTESİ

BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ

Ders	**:**	**İşletim Sistemleri**	
Dönem	**:**	**2022-2023 Güz Dönemi**	
Grup No	**:**	**7**	
Konu	**:**	**Dört Seviyeli Öncelikli Bir Proses Görevlendiricisinin (Dispatcher) Tasarımı ve Uygulaması**	
Grup Üyeleri	**:**	**Mert Eser Meral**	**G211210047**
		Derya Çahan	**G211210017**
		Zehra Bak	**G211210011**
		Bahri Batuhan Güneren	**G201210059**
		Furkan Tataroğlu	**G201210089**

Github : https://github.com/OSProjectGroup7

GÖREVLENDİRİCİ(DİSPATCHER)

Bir işletim sistemindeki görevlendirici (dispatcher), CPU'nun kontrolünü işletim sistemi çekirdeğinden hazır kuyruğundaki bir sonraki sürece aktaran bir bileşendir. Görevlendiricinin ana işlevi, işlemci zamanını verimli bir şekilde yönetmektir. Bu, hazır kuyruğunda bekleyen işlemler arasında hızlı bir geçiş yaparak, her bir işlemin işlemciyi adil ve etkili bir şekilde kullanmasını sağlar. Görevlendirici, işlemci kaynaklarının dağıtımında kritik bir rol oynar ve işletim sisteminin genel performansını önemli ölçüde etkileyebilir. Öncelik sırasına, işlem süresine veya diğer faktörlere göre işlemleri seçebilir ve işlemciye tahsis eder. Bu, sistem kaynaklarının en verimli şekilde kullanılmasını ve işlemlerin hızlı bir şekilde tamamlanmasını sağlar.

SIRALAMA ALGORİTMALARI

İlk Gelen İlk Çalışır (FCFS)

Bu algoritma, gelen işlemleri sırasına göre işler. İşlemler, geldikleri sırayla bir kuyrukta tutulur ve işletim sistemi, kuyruğun önündeki ilk işlemi işler. FCFS, basitliğiyle bilinir ve minimum kaynak kullanımı ile uygulanabilir. Ancak, kısa işlemlerin uzun işlemlerin tamamlanmasını beklemesi gerekebilir, bu da bazen verimsizliklere ve daha uzun bekleme sürelerine yol açabilir. Bu, özellikle uzun süreli işlemlerin sık olduğu durumlarda bir sorun olabilir.

Round Robin (RR)

Bu algoritma, her işleme eşit uzunlukta zaman dilimleri atar ve bu zaman dilimleri içinde işlemleri döngüsel olarak işler. Round Robin, işlemci zamanının adil bir şekilde paylaşılmasını sağlar ve uzun işlemlerin kısa işlemleri bloke etmesini önler. Her işlem, atanan zaman dilimi içinde işlenir ve zaman dilimi sona erdiğinde, işlem kuyruğun sonuna geri döner. Bu süreç, tüm işlemler tamamlanana kadar devam eder. Round Robin, özellikle zaman paylaşımlı sistemlerde etkili bir yöntemdir.

BELLEK TAHSİS ALGORİTMALARI

First Fit

Bu algoritma, bellek taleplerini karşılamak için kullanılabilir ilk uygun bloğu seçer. Belleğin başından itibaren arama yapar ve talep edilen büyüklüğe yeterli olan ilk boş bloğu atar. First Fit'in en büyük avantajı, basitliği ve hızlı uygulanabilirliğidir. Ancak, zamanla bellekte küçük ve kullanılmayan boşluklar (parçalanma) oluşabilir, bu da belleğin daha az verimli kullanılmasına yol açabilir. Bu yüzden, First Fit, belleğin kullanımını ve sistem performansını dikkatlice dengelemek için kullanılır.

Best Fit

Bu algoritma, bellek taleplerini karşılamak için kullanılabilir bellek blokları arasında talep edilen boyuta en uygun olanı seçer. Amacı, bellek parçalanmasını en aza indirmektir. Best Fit, sistemdeki tüm uygun boşlukları inceleyerek en iyi eşleşmeyi bulur. Bu yöntem, belleğin daha verimli kullanılmasını sağlayabilir, ancak uygun bloğu bulmak için gereken zaman ve hesaplama kaynakları diğer yöntemlere göre daha fazla olabilir. Özellikle bellek kaynaklarının kısıtlı olduğu durumlarda etkili bir çözüm sunar.

Worst Fit

Bu algoritma, mevcut boş bellek blokları içinde en büyük olanını tahsis etmeye çalışır. Amacı, büyük bellek parçalarını korumak ve daha küçük bellek talepleri için yer bırakmaktır. Worst Fit, belleğin daha düzensiz kullanılmasına yol açabilir ve zamanla büyük boşlukların azalmasına neden olabilir. Bu yöntem, büyük bellek taleplerinin sık olduğu durumlarda etkili olabilir ancak parçalanmayı artırabilir ve bellek yönetimini zorlaştırabilir.

Next Fit

Bu algoritma, First Fit'e benzer şekilde çalışır, ancak belleğin başından değil, son bellek tahsis işleminin gerçekleştiği noktadan aramaya başlar. Next Fit, bulduğu ilk yeterli boşluğu kullanır. Bu yöntem, bellek aramalarını hızlandırabilir, ancak zamanla bellekte parçalanma sorunlarına yol açabilir. Özellikle, bellekte sürekli olarak ilerleyen bir arama yapısı olduğundan, belleğin bazı bölümleri daha az kullanılırken, diğer bölümleri daha yoğun kullanılabilir.

Projede "Best Fit" algoritması tercih edilmiştir. Bunun sebebi bellek yönetimi ve parçalanma önleme ile ilgilidir. Best Fit, bellek tahsisinde talep edilen boyuta en uygun olan boş bloğu seçer, bu da bellek kullanımını optimize eder ve parçalanmayı azaltır. Bu yöntem, özellikle bellek kaynaklarının sınırlı olduğu veya verimli kullanılmasının önemli olduğu durumlarda etkilidir.

Bu projede, görevlendirici sınıfı bellek ve diğer kaynakları yönetmek için bir dizi yapı kullanır. Bu yapılar arasında proseslerin sıralanması için kuyruklar (RealTimeQueue, FirstPriorityList, SecondPriorityList, RoundRobinList, UserJobQueue) ve kaynak tahsisi için ayrı sınıflar (MemoryAllocator, ResourcesAllocator) bulunur. Prosesler, kaynak ihtiyaçlarına göre değerlendirilir ve uygun kaynaklar tahsis edilerek ilgili kuyruklara yerleştirilir. Bu süreç, sistemin etkin bir şekilde çalışmasını sağlayarak kaynakların verimli kullanımını ve proseslerin düzenli işlenmesini destekler.

KULLANILAN SINIFLAR

Dispatcher:

"Dispatcher.java" sınıfı, işletim sistemindeki prosesleri yönetir. Prosesler için bir LinkedList oluşturur ve çeşitli kuyruklar (RealTimeQueue, FirstPriorityList, vs.) ve kaynak yöneticileri (MemoryAllocator, ResourcesAllocator) ile entegre çalışır. Prosesler, belirli kriterlere (öncelik, kaynak ihtiyacı) göre sınıflandırılır ve uygun kuyruklara yerleştirilir. Sınıf, proseslerin bellek ve diğer kaynak ihtiyaçlarını yönetir ve zaman aşımını kontrol eder. Ayrıca, proseslerin sisteme giriş zamanını ve kaynak kullanımını izler, uygun işlem sırasına göre yönetir.

FirstPriorityList:

"FirstPriorityList" sınıfı, birinci öncelik seviyesindeki prosesleri yönetir. Prosesler, LinkedList yapısında saklanır ve sırayla işlenir. Sınıf, proseslerin işlenmesi ve önceliklerinin güncellenmesi için metodlar içerir. Bir proses işlendikten sonra, eğer hala işlem gerektiriyorsa, önceliği artırılır ve ikinci öncelik listesine (SecondPriorityList) eklenir. Eğer proses tamamlanmışsa, işlem sonlandırılır.

Mab:

"Mab" sınıfı, bellek bloklarını temsil eden bir yapıdır. Her "Mab" nesnesi, bir bellek bloğunun boyutunu, tahsis edilip edilmediğini ve diğer "Mab" nesneleriyle olan bağlantılarını (önceki ve sonraki) saklar. Bu sınıf, bellek tahsis algoritmalarının yönetiminde kullanılır. Her bellek bloğu, boyutu ve tahsis durumuyla birlikte oluşturulur. Bellek tahsis ve geri kazanım sürecinde, bu blokların özellikleri güncellenir ve birbirleriyle bağlantıları değişebilir. Bu, bellek yönetiminde verimlilik ve düzenin sağlanmasına yardımcı olur.

Main:

"Main" sınıfı, programın giriş noktasıdır ve uygulamanın çalışmasını başlatır. Bu sınıf içinde, "Dispatcher" ve "ProcessReader" nesneleri oluşturulur. "Dispatcher" nesnesi, proses yönetimini ve kaynak tahsisini gerçekleştirir. "ProcessReader" nesnesi, proseslerle ilgili verileri okur ve işler. "processread" metodunu çağırarak, bu verileri okur ve ilgili prosesleri yönetmek üzere "Dispatcher" nesnesine aktarır. Bu şekilde, "Main" sınıfı, uygulamanın başlaması ve proseslerin yönetilmesi sürecini başlatır.

MemoryAllocateAlgorithms:

"MemoryAllocateAlgorithms" sınıfı, bellek tahsis algoritmalarını yönetir. Bellek bloklarını "Mab" sınıfı nesneleri olarak saklar ve bu blokları bir bağlı liste yapısında tutar. Sınıf, bellek bloklarını sona ekleme ve bellek tahsis algoritmaları (First Fit, Best Fit, Worst Fit, Next Fit) kullanarak uygun blokları bulma fonksiyonları içerir. Bu fonksiyonlar, talep edilen boyuta uygun müsait bellek bloklarını bulur ve geri döndürür. Sınıf ayrıca, bellek bloklarının durumunu gösteren bir gösterim fonksiyonu içerir. Bu yapı, bellek yönetiminde esneklik ve verimlilik sağlar.

MemoryAllocator:

"MemoryAllocator" sınıfı, işletim sistemlerinde bellek yönetimi için kullanılır. Bu sınıf, toplam bellek boyutu ve gerçek zamanlı işlemler için ayrılan belleği tanımlar. Bellek blokları, "MemoryAllocateAlgorithms" sınıfı ile yönetilir. "allocateMemory" metodu, prosesler için bellek tahsis eder ve farklı bellek tahsis algoritmalarını (First Fit, Best Fit, Worst Fit, Next Fit) destekler. Bellek bloğu bulunursa, bu bloğun tahsis edildiği işaretlenir ve son tahsis edilen blok güncellenir. "deallocateMemory" metodu, tahsis edilen belleği serbest bırakır. "displayMemoryStatus" ise belleğin mevcut durumunu gösterir.

ProcessItems:

"ProcessItems" sınıfı, işletim sistemlerindeki proseslerin özelliklerini ve durumlarını temsil eder. Her proses için benzersiz bir ID, prosesin askıya alındığı süre (suspend), çalışma süresi (burstTime), sistemdeki varış zamanı (arrival), öncelik seviyesi (priority), ve kaynak talepleri (yazıcı, tarayıcı, modem, CD sürücüsü, ve bellek boyutu) gibi bilgileri içerir. Sınıf, bu özelliklerin get ve set metodlarını sağlayarak, proseslerin yönetimi ve takibi için gerekli bilgilerin saklanmasını ve güncellenmesini sağlar.

ProcessReader:

"ProcessReader" sınıfı, proses bilgilerini bir dosyadan okur ve bu bilgileri işler. "Dispatcher" sınıfına bir referans alır ve dosyadan okunan proses bilgilerini bu dispatcher'a ekler. Sınıf, "giris.txt" dosyasını açar, her satırı okur, ve her satırdaki verileri ayrıştırarak yeni bir "ProcessItems" nesnesi oluşturur. Her prosesin varış zamanı, önceliği, çalışma süresi, bellek ihtiyacı ve diğer kaynak ihtiyaçları gibi bilgiler dosyadan alınır ve ilgili proses özelliklerine atanır. Daha sonra bu prosesler, "Dispatcher" sınıfının kuyruğuna eklenir ve "Dispatcher" tarafından işlenmek üzere yönlendirilir. Bu süreç, proseslerin sistem tarafından verimli bir şekilde yönetilmesini sağlar.

RealTimeQueue:

"RealTimeQueue" sınıfı, gerçek zamanlı prosesleri yönetmek için bir LinkedList kullanır. Prosesler, FCFS (First-Come, First-Served) algoritması kullanılarak kuyruğa eklenir. "FCFS_execute" metodu, kuyruktaki ilk prosesi işler. Bu prosesin çalışma süresi (burstTime) bitene kadar her döngüde azaltılır ve prosesin durumu güncellenir. Proses tamamlandığında, kalan çalışma süresi ve diğer bilgileri konsola yazdırılır.

ResourcesAllocator:

"ResourcesAllocator" sınıfı, proseslerin yazıcı, tarayıcı, modem ve CD sürücüsü gibi kaynak taleplerini yönetir. rsrcChk metodu, bir prosesin kaynak taleplerinin mevcut kaynaklarla uyumlu olup olmadığını kontrol eder. rsrcAlloc metodu, uygun kaynakları tahsis eder ve rsrcFree metodu, kullanılan kaynakları serbest bırakır.

RoundRobinList

"RoundRobinList" sınıfı, Round Robin sıralama algoritmasını uygular. Bu sınıf, bir LinkedList yapısı kullanarak prosesleri saklar. "RR_add" metodu, yeni prosesleri kuyruğun sonuna ekler. "RR_execute" metodu, kuyruktaki prosesleri sırayla işler. Her bir proses, belirli bir zaman dilimi için işlenir (timer arttırılır, burstTime azaltılır). Eğer bir prosesin işlem süresi bitmişse, kuyruktan çıkarılır ve sonlandırılır. Aksi takdirde, proses askıya alınır ve sıradaki prosese geçilir.

Rsrc

"Rsrc" sınıfı, işletim sistemi içinde kullanılan kaynakları (yazıcılar, tarayıcılar, modemler, CD sürücüleri) temsil eder. Her bir kaynak türü için belirli bir miktarı saklar ve yönetir. Sınıf, bu kaynakların mevcut sayısını tutar ve kaynakların tahsis edilmesi veya serbest bırakılması için gereken işlemleri sağlar. Sınıf, kaynakların kullanımını ve mevcudiyetini izlemek için getter ve setter metodları içerir.

SecondPriorityList

"SecondPriorityList" sınıfı, ikinci öncelik seviyesindeki prosesleri yönetir. Bu sınıf, bir LinkedList ve RoundRobinList içerir. Prosesler, "SPL_add" metodu ile bu listeye eklenir. "SPL_execute" metodu, listeye eklenen prosesleri işler. Her proses için, işlem süresi, kalan süresi ve diğer bilgiler güncellenir. Eğer bir prosesin işlem süresi bitmemişse, önceliği artırılır ve Round Robin listesine eklenir. Proses tamamlandığında, sonlandırılır.

UserJobQueue

"UserJobQueue" sınıfı, kullanıcı işlerini (proseslerini) yönetir. Bu sınıf, bir LinkedList yapısında prosesleri saklar ve "UJQ_add" metodu ile yeni prosesler ekler. "UJ_Dispatch" metodu, kuyruktaki her prosesi işler. Proseslerin öncelik seviyelerine göre, bellek ve kaynak ihtiyaçlarını kontrol eder ve uygunsa bellek ve kaynakları tahsis eder. Eğer prosesin bellek veya kaynak talepleri uygun değilse, kuyruktan çıkarılır ve hata mesajı yazdırılır. Aksi durumda, proses uygun öncelik listesine (FirstPriorityList veya SecondPriorityList) veya Round Robin listesine eklenir.

OLASI İYİLEŞTİRMELER

Bu projede kullanılan çok düzeyli görevlendirme şeması, gerçek işletim sistemlerindeki karmaşık senaryolara bir yaklaşım sunar. Gerçek sistemler, çeşitli proses önceliklerini ve kaynak ihtiyaçlarını yönetirken, etkili zaman ve kaynak dağıtımı yaparlar. Bu şema, farklı öncelik seviyelerine sahip prosesleri ayrı kuyruklarda yöneterek, her proses tipine uygun kaynak tahsisi ve işlem zamanı sunar. Ancak, bu yapı karmaşıklığı artırabilir ve bazı durumlarda kaynakların etkin kullanımını zorlaştırabilir. Gerçek sistemlerde gözlemlenen adaptif bellek yönetimi ve dinamik kaynak tahsis mekanizmalarının entegrasyonu, performans ve etkinlik açısından iyileştirmeler sağlayabilir. Bu proje, basit bir işletim sistemi modeli sunsa da, gerçek sistemlerdeki gibi önceliklere dayalı kaynak ve zaman yönetimini simüle eder.