T.C.

SAKARYA ÜNİVERSİTESİ

BİLGİSAYAR VE BİLİŞİM BİLİMLERİ FAKÜLTESİ

BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ

İŞLETİM SİSTEMLERİ DERSİ PROJE ÖDEVİ

4 SEVİYELİ ÖNCELİKLİ GÖREVLENDİRİCİ

https://github.com/BerkaySY/52.Grup_OS

PROJE EKİBİ

B211210040 - Berkay SARAY

B211210079 - Mustafa YAKIN

B211210307 - Aziz ASLAN

G221210379 - Fırat ÇAĞLAR

SAKARYA

31 Aralık 2023

İçindekiler

İçindekiler **2**
Hangi Bellek Tahsis Uygulamasını Kullandık? **3**
1.First-Fit (İlk Uygun) 3
2.Best-Fit (En Uygun) 3
3.Worst-Fit (En Kötü) 4
4.Next-Fit (Sonraki Uygun) 4
5.Contiguous Memory Allocation (Ardışık Bellek Tahsisi) 4
6.Buddy System (Arkadaş Sistemi) 4
Görevlendirici Tarafından Kullanılan Yapılar **5**
A. FCFS 5
B. Geri Beslemeli Sıralayıcı 6
C. Round-Robin 6
D. Karışık Sıralayıcı 7
Programımızın Genel Yapısı **8**
A. Color 8
B. Dispatcher 8
C. Process 9
D. ProcessList 9
E. ResourceManager 10
F. MemoryManager 10
G. RealTime Queue 11
H. User Queue 11
Tartışmalar ve Olası İyileştirmeler **13**
1. Çok Düzeyli Görevlendirme Şemasının Avantajları: **13**
a. Paralel İşlemler ve Performans: 13
b. Öncelikli Görev Yönetimi: 13
c. Yüksek Kullanılabilirlik: 13
2. Gerçek İşletim Sistemleri ile Karşılaştırma: **13**
a. Linux İşletim Sistemi: 13
b. Windows İşletim Sistemi: 13
3. Çok Düzeyli Görevlendirme Şemasının Kullanım Nedenleri: **14**
a. Hız ve Etkinlik: 14
b. Kaynak Verimliliği: 14
c. Kullanılabilirlik ve Tepki Süresi: 14
4. Olası İyileştirmeler ve Eksiklikler: **14**
a. İyileştirmeler: 14
b. Eksiklikler: 14
5. Bellek ve Kaynak Ayırma Şemaları: 15
Bellek Ayırma Şemaları: 15
1. Statik Bellek Ayırma: 15
2. Dinamik Bellek Ayırma: 15
3. Önbellek Yönetimi: 15
Kaynak Ayırma Şemaları: 15
1. Öncelikli Kaynak Tahsisi: 15

2.Dinamik Kaynak Tahsisi: 15
3.İzinsiz Kaynak Erişimi Kontrolü: 16
KAYNAKÇA 17

Hangi Bellek Tahsis Uygulamasını Kullandık?

Projemiz sınırlı kullanılabilir kaynakların kısıtlamaları içinde çalışan bir çoklu programlama sistemini ele almaktadır. Sistem, dört seviyeli öncelikli proses görevlendiricisine sahiptir, bu da görevlerin önceliklerine göre sıralanmasını ve yönetilmesini sağlar.

Bellek tahsis algoritmaları, projemizde önemli bir rol oynamaktadır çünkü çoklu programlama ortamında bellek yönetimi kritik bir faktördür.

1.First-Fit (İlk Uygun)

Bellek blokları arasında ilk uygun olanı seçer. Basit ve hızlıdır ancak parçalanmaya eğimlidir. Projemizdeki öncelikli proses görevlendiricisi, hızlı bellek tahsisine ihtiyaç duyar ve bu nedenle basitliği ve hızı nedeniyle First-Fit tercih edilmiştir.

Neden First-Fit?;

Hız: First-Fit, bellek blokları arasında ilk uygun olanı seçer ve bu nedenle tahsis işlemlerini hızlı bir şekilde gerçekleştirir.Özellikle çoklu programlama ortamında, hızlı görev değişimleri ve bellek tahsisleri önemlidir.

Basitlik: First-Fit basit bir algoritmadır ve uygulanması kolaydır. Projenin karmaşıklığını arttırmadan bellek yönetimini sağlamak önemlidir, çünkü bu, genel sistem performansını olumlu yönde etkiler.

İlk Uygun İlgisi: Projemizdeki öncelikli proses görevlendiricisi, en hızlı ve en kolay uygun bellek bloğunu seçmeyi tercih eder. First-Fit , bu ihtiyacı karşılamak için uygun bir seçenektir.

2.Best-Fit (En Uygun)

Bellek boşlukları arasında en küçük olanı seçer ancak ondan büyük olan en küçük boşluğu seçer. Parçalanmayı azaltabilir, ancak işlemci zamanı daha fazla kullanılabilir çünkü en küçük uygun boşluğu bulmak için tarama yapmak gerekir.

Neden kullanılmadı?: Best Fit, bellek tahsisinde verimli bir kullanım sağlayabilir, ancak genellikle daha yavaş çalışır çünkü uygun bloğu bulmak için tüm boşlukları tarar. Projemizdeki çoklu programlama sistemi, hızlı bellek tahsisi ve serbest bırakma ihtiyacından dolayı, performansı artırmak amacıyla tercih edilmemiştir.

3.Worst-Fit (En Kötü)

Worst Fit, talep edilen bellek bloğu boyutuna uygun olmayan en büyük bloğu seçer.

Neden kullanılmadı?: Worst Fit, dış parçalanmayı azaltabilir, ancak genellikle büyük blokların seçilmesi nedeniyle bellek boşlukları daha hızlı dolabilir. Bu, projemizdeki sık ve hızlı bellek tahsisi ihtiyacına uygun değildir. Ayrıca, blokları aramak için ek işlem maliyeti getirebilir.

4.Next-Fit (Sonraki Uygun)

Next Fit, en son kullanılan bellek bloğunun hemen sonrasındaki ilk uygun bloğu seçer.

Neden kullanılmadı?: Next Fit, First Fit'e benzer, ancak daha hızlıdır çünkü en son kullanılan bloktan devam eder. Ancak, projemizdeki çoklu programlama sistemi genellikle First Fit ile benzer performans elde edebileceği için Next Fit tercih edilmemiştir.

5.Contiguous Memory Allocation (Ardışık Bellek Tahsisi)

Programa veya veriye tek bir bellek bloğu atar. Kolay erişim sağlar ancak dış parçalanma sorunlarına duyarlıdır.

Neden kullanılmadı?: Projemizdeki çoklu programlama sistemine, sık ve hızlı bellek tahsis ve serbest bırakma ihtiyacı olduğundan dolayı ardışık bellek tahsisi tercih edilmedi. Dış parçalanma, sistem performansını olumsuz etkileyebilirdi.

6.Buddy System (Arkadaş Sistemi)

Bellek alanını ikişerli bloklara böler ve talep edilen boyuta en yakın olanı tahsis eder. Verimli ve hızlıdır, ancak iç parçalanma sorunlarına neden olabilir.

Neden kullanılmadı?: Buddy System, hızlı ve etkili bellek yönetimi sağlamak açısından faydalı olabilir ancak iç parçalanma sorunlarına duyarlıdır. Projemizdeki öncelikli proses görevlendiricisi, bu iç parçalanma sorunlarından kaçınmak için bu yöntemi tercih etmedik.

Görevlendirici Tarafından Kullanılan Yapılar

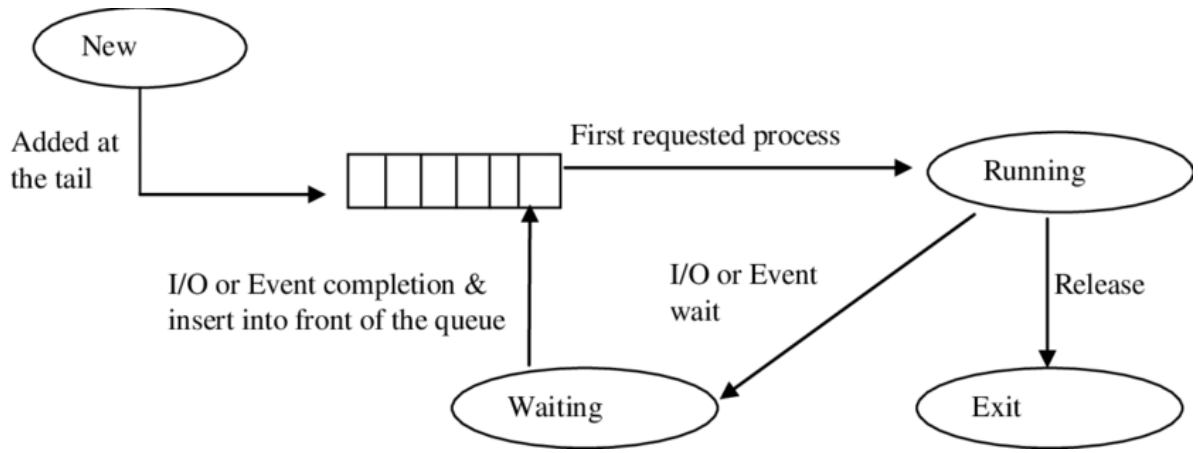
A. FCFS

İşlemciye gelen görevleri bir kuyrukta sıralı şekilde işleme alma mantığına dayanır. Yani, gelen işler sırayla kuyruğa eklenir ve işlemci boş olduğunda kuyruktaki ilk işlemciye verilir. Öncelik veya öncelik sırası gibi faktörler olmadığı için gelen işlerin sırasına göre işlenirler.

Bu algoritma, basitliği nedeniyle sıkça tercih edilir. Ancak bazı durumlarda bekleme süreleri uzayabilir. Örneğin, bir işlem işlemciyi uzun süre meşgul ederse, sıradaki işlerin başlama zamanı gecikebilir ve bu da bekleme sürelerini artırabilir.

FCFS'nin avantajlarından biri, basit uygulamasıdır. Karmaşık hesaplamalara veya öncelik kurallarına ihtiyaç duymaz. Ancak, işlem süreleri değişken olduğunda performans düşebilir. Kısa işlemler uzun işlemlerden sonra gelirse, bekleme süreleri artabilir ve verimlilik azalabilir.

Genel olarak, FCFS basit işlerde veya işlem sürelerinin çok değişken olmadığı durumlarda tercih edilebilir. Ancak, daha karmaşık sistemlerde veya iş yükü değişkenlik gösterdiğinde daha etkili planlama algoritmaları tercih edilebilir.



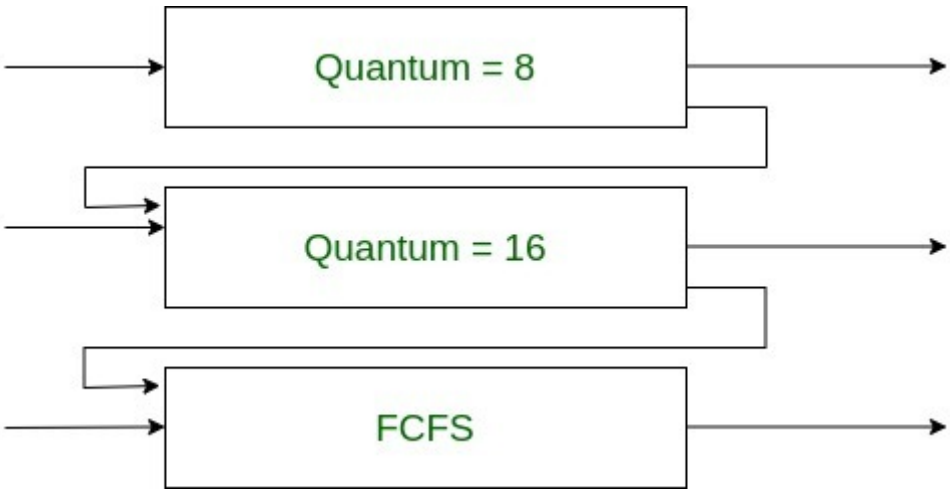
B. Geri Beslemeli Sıralayıcı

Geri Beslemeli Sıralayıcı, işlerin önceliklerini değiştirebilme yeteneğine sahip. Yani, bir işlem başladığında bir öncelikle başlar, ama eğer bir süre içinde bitmezse, önceliği düşer. Tam tersi, düşük öncelikli işlemler belirli bir süre sonra önceliklerini yükseltebilir. Bu sayede, işletim sistemi işleri dengeler ve adil bir şekilde işlemciler arasında dağıtır.

Yani, çoklu öncelik seviyeleriyle işlemciyi yönetmeye çalışıyor. Özellikle, acil işlerin hemen bitirilmesi gerektiğinde önceliklerini arttırıp hızlıca halletmeye çalışıyoruz. Böylece, işlemciyi en iyi şekilde kullanabiliriz ve farklı işler arasında daha verimli geçişler yapabiliriz

Ama tabii bu yöntemin bazı zorlukları da var. Mesela, işlerin sürekli olarak öncelik değiştirmesi adil olmayabilir. Bazı işler de uzun süre düşük öncelikte kalabilir. Ayrıca, yönetim bazen karmaşık olabilir ve sürekli öncelik değişimi sistem performansını da etkileyebilir.

Geri Beslemeli Sıralama, dinamik ve esnek bir işlemci planlama yöntemi olmakla birlikte, kullanım alanına ve sistemin gereksinimlerine bağlı olarak avantajları ve dezavantajları bulunur.



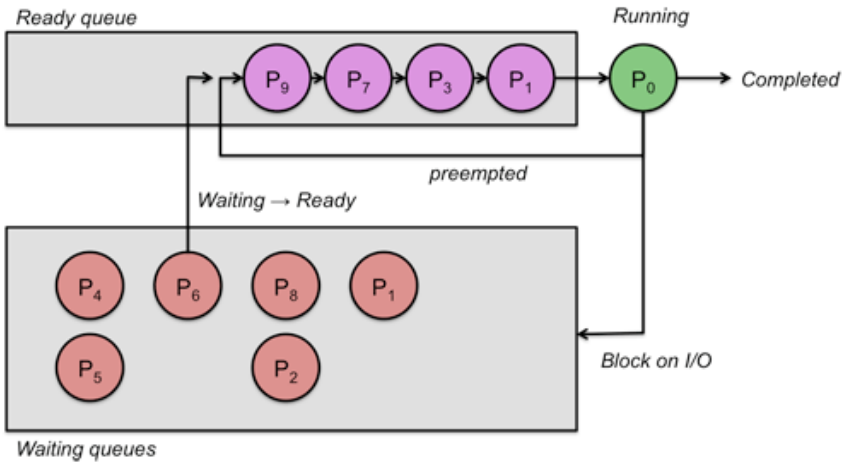
C. Round-Robin

Bu yöntem, işlemciyi paylaşan birden fazla iş arasında adil bir dağılım sağlamak için tasarlanmıştır.Bu algoritma, işleri sırayla ele alır ve her bir işe eşit bir zaman dilimi ayırır.

Örneğin, bir grup öğrencinin ders çalışma sırasında adil bir şekilde zaman alması gibi düşünebiliriz. İlk ödevi başlatırsınız, belirli bir süre çalışırsınız ve sonra diğer ödevlere geçersiniz.

Round Robin'in güzel yanı, işler arasında eşit bir paylaşım yapmasıdır. Ancak, bazı durumlarda işler uzun sürebilir ve bekleme süreleri artabilir. Özellikle bir işlem çok uzun sürerse, diğer işlerin başlama zamanı gecikebilir

Genel olarak, Round Robin adil bir sistemdir ve işlemci zamanını işler arasında dengeli bir şekilde dağıtır. Ancak, büyük iş yükleri veya uzun işler söz konusu olduğunda daha verimli planlama yöntemleri tercih edilebilir.



Ç. Karışık Sıralayıcı

Karışık Sıralayıcı terimi, birden fazla işin aynı anda çalıştırıldığı durumları ifade eder. Bu sıralayıcı, işlerin farklı önceliklere sahip olmasını yönetir ve onları belirli bir düzene göre sırayla çalıştırır.

Projede bu terim, gerçek zamanlı ve kullanıcı işlemlerinin birlikte yönetildiği bir sistemi anlatabilir. Gerçek zamanlı işlemler daha yüksek öncelikli kabul edilirken, kullanıcı işlemleri daha düşük öncelikli olabilir. Bu sıralama tarzı, farklı önceliklerdeki işlerin birbiri ardına çalıştırılmasını sağlar.

Böylece, sistem öncelikli işlere hızlı cevap verirken, diğer işleri de sırayla gerçekleştirir. Bu durum, hem hızlı tepki verme ihtiyacı olan gerçek zamanlı işlerin önemini korurken, kullanıcı işlemlerine de adil davranmayı amaçlar.

Programımızın Genel Yapısı

A. Color

Bu sınıf, rastgele renkler oluşturmak için bazı işlevler barındırıyor. Kodun genel amacı, onaltılık sembollerle renk kodları oluşturmak ve bu kodları RGB formatına çevirmek.

İlk olarak, **getRandomHex()** fonksiyonu rastgele onaltılık semboller üretiyor. Bu semboller daha sonra birleştirilerek **getHexColor()** fonksiyonu içinde altı sembol uzunluğunda bir renk kodu oluşturuyor. Örneğin, *#FFFFFF* gibi.

Ardından, **hex2RGB()** fonksiyonu bu oluşturulan renk kodunu alıp, onu RGB (Red, Green, Blue) formatına dönüştürüyor. Yani, o uzun renk kodunu kırmızı, yeşil ve mavi bileşenlere ayırıp, her birini ondalık sistemdeki bir sayıya dönüştürüyor.

Bu sayede, program rastgele renkler oluşturabiliyor ve bu renklerin RGB bileşenlerini veriyor. Örneğin, '255;255;255' gibi. Daha sonra bu renkleri konsola yazdırma işlemi yaparken her bir satırda, proseste farklı bir renk gelecek şekilde atama yapıyoruz.

B. Dispatcher

Öncelikle, Dispatcher sınıfı, işlemlerin kaynakları, belleği, kullanıcı işlemlerini ve gerçek zamanlı işlemleri yönetmek için farklı sınıfları kullanıyor. Bu sınıflar, işlem listelerini, sıra sistemlerini ve çeşitli işlem planlama algoritmalarını içeriyor.

Dispatcher sınıfının Execute() adlı ana metodu, önce bir dosyadan işlemleri okuyor gibi gözüküyor. Daha sonra, bir döngü içinde işlemlerin durumuna göre onları farklı kuyruklara yerleştiriyor.

Bir döngü süresince, işlemler belirli şartlara göre farklı kuyruklara atanıyor. Ardından, işlem planlama algoritmaları (GBG, FCFS ve RR gibi) bu kuyruklardan işlemleri çekerek işlemleri yürütüyor.

C. Process

İşlemler bir kimlik (ID), varış zamanı, öncelik, işlem süresi, bellek boyutu, yazıcı, tarayıcı, modem ve CD sayısı gibi özelliklere sahip olabilir.

Sınıfın yapıcı metodu, işlemin bu özelliklerine başlangıç değerleri atıyor. Her bir işlem oluşturulduğunda, o işleme özel bir renk oluşturuyor.**(Color.java)**

İşlemin durumunu izlemek için bir değişken var. İşlem, başlatıldığında, çalıştırıldığında, askıya alındığında, devam edildiğinde veya tamamlandığında farklı durumlar alabilir.

Bu sınıf, işlemin durumuna göre belirli mesajlar oluşturmak için iki farklı metoda sahip. Bir tanesi, işlemle ilgili genel mesajlar üretiyor ve işlemin durumunu ekrana yazdırıyor. Diğeri ise, işlem sırasında oluşabilecek hataları ele alıp, bunları ekrana yazdırıyor.

Kısacası, 'Process' sınıfı, bir işlemi temsil etmek için gerekli olan bilgileri saklıyor ve bu işlemlerin durumlarına göre belirli mesajlar üretebiliyor.

Ç. ProcessList

Sınıfın içerisinde _process_list adında bir liste tutuyoruz. Bu liste, Process sınıflarının bir dizisini tutuyor. Ayrıca, gerçek zamanlı işlem sayısını (*rt_process_count*) takip eden sayısal değişken var.

Sınıfın yapıcı metodu, bu işlemleri tutmak için bir dizi oluşturuyor ve sayma işlevi görecek olan değişkenlere başlangıç değerleri atıyor.

ReadFile() metodu, verilen dosyadan -**giris.txt**- işlemleri okuyor. Dosya satır satır okunuyor ve her satır, virgülle ayrılmış farklı bilgiler içeriyor. Bu bilgiler sırasıyla varış zamanı, öncelik, işlem süresi, bellek boyutu, yazıcı, tarayıcı, modem ve CD sayılarını içeriyor. Bu bilgiler kullanılarak yeni bir Process nesnesi oluşturuluyor ve bu işlem listesine ekleniyor. Aynı zamanda gerçek zamanlı işlem sayısı güncelleniyor.

getProcessList() metodu, işlem listesini döndürüyor. Diğer metodlar ise işlemlerin boş olup olmadığını kontrol etmek, listenin ilk işlemini almak, gerçek zamanlı ve öncelik 3 olmayan işlem sayılarını almak gibi çeşitli işlevler sunuyor.

D. ResourceManager

Sistemdeki kaynakları kontrol etmek ve işlemlere bu kaynakları tahsis etmek için kullandığımız bir class. Yani, bir işlem ne kadar yazıcı, tarayıcı, modem veya CD sürücüsü gibi kaynağa ihtiyaç duyarsa, bu sınıf bu kaynakları sağlamaya çalışıyor.

Ayrıca ProcessList ile işlemlere kaynak tahsis etme ve bu kaynakları geri alma gibi işlemleri yapabiliyoruz. Örneğin, bir işlem bir yazıcı talep ettiğinde, bu sınıf mevcut yazıcıları kontrol ederek bu talebi karşılamaya çalışıyor.

Ayrıca, bu sınıf, bir işlemin talep ettiği kaynakların mevcut kaynaklardan daha fazla olup olmadığını kontrol edebiliyor. Yani, bir işlem için yeterli kaynak varsa, bu sınıf bu kaynakları tahsis edebilir; yoksa reddedebilir.

E. MemoryManager

Bu sınıf, gerçek zamanlı ve kullanıcı işlemleri için ayrı ayrı bellek blokları tutar. Bu sınıfta yaptığımız işlemleri şöyle sıralayabiliriz:

- ➔ **Bellek Blokları:** İki farklı liste kullanarak gerçek zamanlı (_rt_memory_blocks) ve kullanıcı (_user_memory_blocks) işlemleri için bellek bloklarını saklar.

- ➔ **Başlangıç Bellek Boyutları:** Gerçek zamanlı işlemler için bellek boyutu 64 birim, kullanıcı işlemleri için ise 960 birim olarak belirlenir.

- ➔ **Bellek Tahsisi:** *AllocateMemory() fonksiyonu*, işleme uygun belleğin olup olmadığını kontrol eder. Uygunsa, işleme bellek tahsis eder. Eğer gerçek zamanlı işlemse _rt_memory_blocks listesine, kullanıcı işlemiyse _user_memory_blocks listesine ekler.

- ➔ **Bellekten Çıkarma:** *DeallocateMemory() fonksiyonu*, işlemin bellekten çıkarılmasını sağlar. İşlemin gerçek zamanlı veya kullanıcı işlemi olmasına göre ilgili listeden işlemi çıkarır ve belleği geri kazanır.

- ➔ **İlk Uygun Yer Eşleme (First Fit):** *FirstFit() fonksiyonu*, işlem için ilk uygun bellek bloğunu bulur ve bu bloğa işlemi yerleştirir.

- ➔ **Bellek Durumu Kontrolü**: *isReady() fonksiyonu*, bir işlemin belirli bir bellek boyutuna sahip olup olamayacağını kontrol eder.

- ➔ **Geçerli Bellek Boyutu Kontrolü**: *isValid() fonksiyonu*, bir işlemin belirli bir bellek boyutundan daha büyük olup olamayacağını kontrol eder.

F. RealTime Queue

Gerçek zamanlı prosesleri tutmak için bu kuyruğu kullanıyoruz. İçerisinde yazdığımız fonksiyonların açıklamaları ise şöyle,
Sınıfın yapıcı metodu, gerçek zamanlı işlemleri saklamak için bir liste oluşturuyor. Ardından:

- ➔ **Enqueue() metodu**, bir işlemi gerçek zamanlı kuyruğa ekliyor. Yani, bir işlem gerçek zamanlı olarak çalıştırılacaksa, bu kuyruğa ekleniyor.

- ➔ **Dequeue() metodu**, gerçek zamanlı kuyruktan bir işlemi çıkarıyor. Eğer kuyruk boş değilse, en öndeki işlem kuyruktan çıkarılıyor.

- ➔ **isEmpty() metodu**, kuyruğun boş olup olmadığını kontrol ediyor. Eğer gerçek zamanlı kuyruk boşsa, true değerini döndürüyor; doluysa false döndürüyor.

- ➔ **getRTQueue() metodu**, gerçek zamanlı kuyruktaki işlemleri döndürüyor. Bu şekilde, bu kuyruk dışında da işlemlerin listesine erişilebiliyor.

- ➔ **getFirst() metodu**, kuyruğun en başındaki işlemi döndürüyor. Bu şekilde, gerçek zamanlı kuyruktaki ilk işlem diğer işlemlerle karşılaştırılabilir veya işlemlerin durumu incelenebilir.

G. User Queue

Bu sınıf, farklı öncelik seviyelerine sahip işlemleri bir arada tutar. Yani, önceliklerine göre ayrılmış kuyruklarda işlemleri saklar.

Bu sınıfın içerisinde yaptıklarımız ise şöyle:

- ➔ **Öncelikli Kuyruklar**: Üç ayrı kuyruk oluşturur. Her kuyruk, farklı işlem öncelik seviyelerini temsil eder. Mesela, 1'den 3'e kadar olan öncelikler için ayrı ayrı kuyruklar bulunur.

- ➔ **İşlem Ekleme**: *Enqueue() fonksiyonu*, işlem önceliğine bağlı olarak işlemi ilgili öncelik kuyruğuna ekler. Örneğin, 1. öncelik kuyruğuna 1 önceliğine sahip bir işlem ekler.

➔ **İşlem Çıkarma**: *Dequeue() fonksiyonu*, öncelik sırasına göre kuyruklardan işlemi çıkarır. Öncelik yüksekse, ona öncelik verir.

➔ **Öncelikli İlk İşlem**: *getFirst() fonksiyonu*, önceliklerine göre kuyruklardan ilk işlemi döndürür.

➔ **Boşluk Kontrolü**: *isEmpty() fonksiyonu*, tüm öncelik kuyruklarının boş olup olmadığını kontrol eder.

➔ **Belirli Öncelik Kuyruğu Alma**: *getPriQueue() fonksiyonu*, belirli bir öncelik seviyesine ait kuyruğu döndürür.

➔ **Round Robin İçin İşlem Ekleme**: *EnqueueRR() fonksiyonu*, Round Robin algoritması için özel olarak öncelik 3 kuyruğuna işlem ekler.

➔ **Round Robin Çalıştırma Kontrolü**: *check4RunRR() fonksiyonu*, Round Robin için işlem olup olmadığını kontrol eder.

Tartışmalar ve Olası İyileştirmeler

1. Çok Düzeyli Görevlendirme Şemasının Avantajları:

a. Paralel İşlemler ve Performans:

Çok düzeyli görevlendirme, birçok görevin aynı anda çalışmasına izin verir. Bu, işlemci kaynaklarının daha etkili bir şekilde kullanılmasını sağlar ve sistem performansını artırabilir.

b. Öncelikli Görev Yönetimi:

Dört seviyeli öncelikli proses görevlendirici, görevlere öncelikler atayarak kritik işlerin öncelikli olarak işlenmesini sağlar. Bu, kritik işlerin hızlı bir şekilde tamamlanmasına olanak tanır.

c. Yüksek Kullanılabilirlik:.

Çok düzeyli görevlendirme, sistemdeki görevlerin birbirlerinden bağımsız olarak çalışmasına olanak tanır. Bu, bir görevin hataya neden olması durumunda diğer görevlerin etkilenmesini önleyebilir ve sistemde yüksek kullanılabilirlik sağlar.

2. Gerçek İşletim Sistemleri ile Karşılaştırma:

a. Linux İşletim Sistemi:

Linux, çok düzeyli görevlendirme şemalarını kullanarak yüksek performanslı ve güvenilir bir işletim sistemidir. Çekirdek seviyesinde çoklu görevlendirme, farklı önceliklere sahip görevlerin birlikte çalışmasını yönetir.

b. Windows İşletim Sistemi:

Windows, özellikle çoklu çekirdekli sistemlerde paralel işlemler için çok düzeyli görevlendirme stratejilerini benimser. Görev öncelikleri, kullanıcı arayüzü ve arka plandaki işlemler arasında dengeli bir şekilde yönetilir.

3. Çok Düzeyli Görevlendirme Şemasının Kullanım Nedenleri:

a. Hız ve Etkinlik:

Çok düzeyli görevlendirme, sistemdeki görevlerin hızlı bir şekilde işlenmesini sağlar. Özellikle paralel işlemler ve öncelikli görev yönetimi ile sistem performansını artırır.

b. Kaynak Verimliliği:

Görevlendirme şeması, işlemci ve diğer kaynakları daha verimli bir şekilde yönetmeye yardımcı olur. Bu, sistemdeki kaynakların optimal kullanımını sağlar.

c. Kullanılabilirlik ve Tepki Süresi:

Çok düzeyli görevlendirme, kullanılabilirliği artırır ve kullanıcı etkileşimlerine daha hızlı tepki süreleri sağlar. Öncelikli görev yönetimi, kritik işlerin öncelikli olarak ele alınmasını sağlar.

4. Olası İyileştirmeler ve Eksiklikler:

a. İyileştirmeler:

Görev öncelikleri ve kaynak yönetimi stratejilerinin daha dinamik olması, sistem performansını daha da artırabilir.

b. Eksiklikler:

Eğer öncelikli görev yönetimi düzgün yapılmazsa, düşük önceliğe sahip görevlerin ihmal edilmesi gibi sorunlar ortaya çıkabilir.

Sistemin karmaşıklığı arttıkça yönetim zorlukları yaşanabilir. Bu durum, yönetim araçları ve stratejilerin daha iyi geliştirilmesini gerektirebilir.

Sonuç olarak, çok düzeyli görevlendirme şemaları, işletim sistemlerinin paralel işleme yeteneklerini artırmak ve kaynakları etkili bir şekilde kullanmak için kullanılır. Gerçek işletim sistemleri de bu tür şemalardan faydalanarak yüksek performans ve güvenilirlik sağlar. Ancak, bu şemalardaki karmaşıklıkları yönetmek ve performansı daha da arttırmak adına sürekli iyileştirmeler gereklidir.

5. Bellek ve Kaynak Ayırma Şemaları:

Bellek Ayırma Şemaları:

Çok düzeyli görevlendirme şemalarında bellek ayırma stratejileri, sistemin bellek yönetimini optimize etmeye yöneliktir.

1. Statik Bellek Ayırma:

   Açıklama: Sistem başlangıcında belirli bellek blokları sabit bir şekilde ayrılır.

   Tartışma: Statik bellek ayırma, bellek alanının önceden tahsis edilmesini sağlar, ancak değişen bellek ihtiyaçlarına uyum sağlamakta sınırlıdır.

2. Dinamik Bellek Ayırma:

   Açıklama: Bellek, çalışan görevlere ihtiyaçlarına göre dinamik olarak tahsis edilir.

   Tartışma: Dinamik bellek ayırma, daha esnek bir bellek yönetimi sağlar ancak yönetim maliyeti ve bellek parçalanması gibi zorlukları beraberinde getirebilir.

3. Önbellek Yönetimi:

   Açıklama: Sık kullanılan verileri önbellekte tutma stratejileri kullanılır.

   Tartışma: Önbellek yönetimi, işlemci hızını artırabilir ancak etkili bir önbellek yönetimi için sürekli izleme ve ayarlama gereklidir.

Kaynak Ayırma Şemaları:

Çok düzeyli görevlendirme şemalarındaki kaynak ayırma stratejileri, işletim sisteminin kaynakları etkili bir şekilde kullanmasını sağlamak için kritiktir.

1. Öncelikli Kaynak Tahsisi:

   Açıklama: Yüksek öncelikli görevlere öncelikli olarak kaynak tahsis edilir.

   Tartışma: Bu strateji, kritik görevlerin kaynaklara hızlı bir şekilde erişimini sağlar ancak düşük öncelikli görevlerin kaynak taleplerini erteleyebilir.

2.Dinamik Kaynak Tahsisi:

   Açıklama: Kaynaklar, ihtiyaca göre dinamik olarak tahsis edilir.

   Tartışma: Dinamik kaynak tahsisi, kaynakların daha etkili bir şekilde kullanılmasını sağlar ancak sistem karmaşıklığını artırabilir.

3.İzinsiz Kaynak Erişimi Kontrolü:

Açıklama: Her görevin sadece belirli kaynaklara izinli erişim sağlaması.

Tartışma: Bu strateji, sistemdeki kaynak kullanımını daha güvenli ve kontrol edilebilir hale getirebilir ancak izin ayarlarının yönetimi önemlidir.

KAYNAKÇA

“First Fit Best Fit Worst Fit in OS (Example).” PrepInsta, 1 April 2022, https://prepinsta.com/operating-systems/first-fit-best-fit-worst-fit-in-os-example/. Accessed 31 December 2023.

Jain, Sandeep. “Partition Allocation Methods in Memory Management.” GeeksforGeeks, 26 February 2023, https://www.geeksforgeeks.org/partition-allocation-methods-in-memory-management/. Accessed 31 December 2023.

Jain, Sandeep. “Partition Allocation Methods in Memory Management.” GeeksforGeeks, 26 February 2023, https://www.geeksforgeeks.org/partition-allocation-methods-in-memory-management/. Accessed 31 December 2023

“Who Multi-Tasks and Why? Multi-Tasking Ability, Perceived Multi-Tasking Ability, Impulsivity, and Sensation Seeking.” NCBI, 23 January 2013, https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3553130/. Accessed 31 December 2023.

"Operating System Concepts" by Abraham Silberschatz, Peter Baer Galvin, Greg Gagne
https://drive.uqu.edu.sa/_/mskhayat/files/MySubjects/2017SS%20Operating%20Systems/Abraham%20Silberschatz-Operating%20System%20Concepts%20(9th,2012_12).pdf

"Modern Operating Systems" by Andrew S. Tanenbaum
https://drive.uqu.edu.sa/_/mskhayat/files/MySubjects/2017SS%20Operating%20Systems/Abraham%20Silberschatz-Operating%20System%20Concepts%20(9th,2012_12).pdf

"Operating Systems: Three Easy Pieces" by Remzi H. Arpaci-Dusseau, Andrea C. Arpaci-Dusseau
https://pages.cs.wisc.edu/~remzi/OSTEP/