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

SAKARYA
ÜNİVERSİTESİ

Ders	**:**	**İşletim Sistemleri**	
Dönem	**:**	**2023-2024 Güz Dönemi**	
Grup No	**:**	**28**	
Konu	**:**	**Görevlendirici Kabuğu (Dispatcher Shell)**	
Grup Üyeleri	**:**	**İsmail Mete Uçar**	**G201210051**
		Fatih Özdemir	**G201210047**
		Melih Tufan Atlı	**G211210065**
		Ekrem Selçuk Çelik	**G211210065**
		Barış Cebeci	**G201210045**

Github :

https://github.com/Metecode/OShomework

Giriş

Bu rapor, Görevlendirici (Dispatcher) kapsamında ele alınan bellek tahsis algoritmaları, görevlendirme yapıları ve programın genel yapısını detaylandırmaktadır. İşletim sistemleri alanında kritik öneme sahip olan bellek yönetimi ve kaynak tahsisi, bu projede öncelikli odak noktalarımızdır. Raporumuz, bellek tahsis algoritmalarının seçimini ve uygulanmasını, görevlendiricinin bellek ve diğer kaynakları yönetme yöntemlerini ve programın modüler yapısını kapsamlı bir şekilde ele almakta ve gerçek dünya işletim sistemleriyle karşılaştırmalar yapmaktadır.

Görevlendirici Kabuğu ve Dört Seviyeli Öncelikli Görevlendirici

Projemizde kullanılan görevlendirici kabuğu, dört seviyeli öncelikli bir görevlendirme sistemini temel alır. Bu sistem, işlemleri öncelik düzeylerine göre sıralar ve kaynak tahsisinde bulunur. Her bir seviye, özel bir görev veya işlem grubuna atanmıştır. Öncelik düzeyi yüksek olan işlemler, daha düşük öncelikli işlemlere göre daha hızlı ve etkin bir şekilde kaynaklara erişim sağlar.

Bu yaklaşım, işletim sistemi tasarımında yaygın olarak kullanılan bir yöntemdir ve etkin kaynak yönetimi sağlar. Öncelikli görevlendirme, sistem kaynaklarının en verimli şekilde kullanılmasını sağlayarak, yüksek öncelikli işlemlerin hızlı bir şekilde tamamlanmasına olanak tanır. Ayrıca, bu yöntem, düşük öncelikli ancak önemli arka plan işlemlerinin de zamanında işlenmesine imkan verir, böylece sistemin genel verimliliği ve yanıt süresi iyileştirilir.

Bu dört seviyeli sistem, "gerçek" işletim sistemlerinde de benzer şekillerde görülür. Örneğin, çeşitli işletim sistemlerinde farklı öncelik seviyeleri, işlemci zamanını ve diğer kaynakları verimli bir şekilde tahsis etmek için kullanılır. Bu raporda, görevlendirici kabuğumuzun bu gerçek dünya sistemleriyle karşılaştırılmasını ve neden bu tür bir yapıyı tercih ettiğimizi daha ayrıntılı bir şekilde ele alacağız.

Hangi bellek tahsis algoritmalarını kullanabileceğimiz hakkın tartışma ve son gerçekleme kısmı:

Bellek yönetimi, her işletim sisteminin temel bir bileşenidir. Etkin bir bellek yönetimi, sistem kaynaklarının optimal kullanımını sağlar ve böylece performansı ve kararlılığı artırır. Bellek tahsis algoritmaları, bu yönetimin merkezinde yer alır. Genel olarak, bellek tahsis algoritmaları statik ve dinamik olmak üzere iki ana kategoriye ayrılabilir.

Statik tahsis yöntemleri, programların çalışma zamanından önce bellek ihtiyaçlarını belirlerken, dinamik tahsis yöntemleri, programlar çalışırken bellek ihtiyaçlarını karşılar. Dinamik tahsis yöntemleri arasında First Fit, Best Fit, Worst Fit ve Next Fit gibi algoritmalar bulunur. Her bir algoritmanın kendine has avantajları ve dezavantajları vardır.

Proje İçin Algoritma Seçimi

Projemizde, işlemleri yönetmek için basit bir FIFO (First-In, First-Out) kuyruk yapısı kullanan bir bellek tahsis algoritması uyguladık. Bu yaklaşım, işlemleri eklenme sırasına göre sıralar ve işler. Bu yöntemin seçilmesinin başlıca nedenleri şunlardır:

- Basitlik ve Anlaşılabilirlik: FIFO, uygulaması basit ve anlaşılması kolay bir algoritmadır. Bu, özellikle eğitim amaçlı projelerde veya karmaşık olmayan sistemlerde tercih edilen bir özelliktir.
- Adil Kaynak Dağılımı: FIFO, tüm işlemlerin eşit bir şekilde işlenmesini sağlar. Bu, her işlemin adil bir şekilde kaynaklara erişimini garanti eder.
- Düşük Geliştirme Maliyeti: Karmaşık bir bellek yönetimi sistemi geliştirmek zaman alıcı ve maliyetli olabilir. FIFO, bu maliyetleri minimize eder.

Algoritmanın Sınırlamaları ve Potansiyel İyileştirmeler

FIFO algoritmasının bazı sınırlamaları bulunmaktadır. Örneğin, hafıza fragmentasyonu ve düşük hafıza kullanım verimliliği gibi sorunlar ortaya çıkabilir. Bu sınırlamaları aşmak için, gelecekte Best Fit veya Next Fit gibi daha gelişmiş algoritmaları entegre etmeyi düşünebiliriz. Bu algoritmalar, bellek kullanımını optimize ederek sistem performansını artırabilir.

Sonuç

Sonuç olarak, projemiz için FIFO algoritmasını seçtik. Bu seçim, projenin amaçları, kaynakları ve gereksinimleri doğrultusunda yapılmıştır. Gelecekteki geliştirmelerle, daha gelişmiş bellek tahsis algoritmalarını entegre etmeyi hedefliyoruz. Bu sayede, projemizin performansını ve etkinliğini artırabiliriz.

Görevlendirici Tarafından Bellek ve Diğer Kaynakları Yönetimi

Genel Yapı

Projemizdeki görevlendirici, bellek ve diğer kaynakları yönetmek için bir dizi kuyruk yapısı ve kaynak yönetim sınıfı kullanır. Bu yapı, işlemlerin etkin bir şekilde sıralanmasını ve kaynakların adil bir şekilde tahsis edilmesini sağlar.

Kuyruk Yapısı

Kuyruk yapısı, işlemleri yönetmek için kullanılır. Bu yapı, `Queue` sınıfı tarafından temsil edilir ve aşağıdaki özelliklere sahiptir:

- Node: Her bir `Node` nesnesi, bir `MyProcess` nesnesini içerir ve bir sonraki düğüme işaret eder. Bu, işlemlerin sıralı bir şekilde saklanmasını sağlar.
- Push Metodu: Yeni bir işlemi kuyruğa ekler. Bu, işlemlerin FIFO (First-In, First-Out) mantığına göre işlenmesini sağlar.
- Pop Metodu: Kuyruğun başındaki işlemi kaldırır ve döndürür. Bu, işlemin işlenmesi veya kaynak tahsisi için hazır olduğunu gösterir.

Kaynak Yönetimi

Kaynak yönetimi, `Resource` sınıfı tarafından sağlanır. Bu sınıf, sistem kaynaklarını (örneğin, yazıcılar, tarayıcılar) temsil eder ve her kaynağın mevcut olup olmadığını takip eder. Kaynakların yönetimi şu şekilde gerçekleşir:

- Kaynak ID ve Durumu: Her `Resource` nesnesi, bir ID ve mevcut durumu ile tanımlanır. Bu, hangi kaynağın kullanılabilir olduğunu ve hangisinin meşgul olduğunu belirlemeye yardımcı olur.

- Kaynak Tahsisi ve Serbest Bırakma: İşlemler, gerekli kaynakları talep edebilir ve kullanım sonrası serbest bırakabilir. Bu, kaynakların etkin kullanımını ve çakışmaların önlenmesini sağlar.

Sistemdeki Etkileşim

Bu yapılar, sistemin bellek ve kaynak yönetimini etkin bir şekilde gerçekleştirmesine olanak tanır. İşlemler, kuyruklar aracılığıyla sıralanır ve kaynaklar `Resource` sınıfı üzerinden yönetilir. Bu, kaynakların verimli kullanımını ve işlemlerin düzgün bir şekilde işlenmesini sağlar.

Programın Genel Yapısı ve Modülleri

Genel Bakış

Projemiz, işletim sistemlerinin temel işlevlerini simüle etmek üzere tasarlanmış bir yapıya sahiptir. Bu yapı, çeşitli modüllerden oluşur ve her modül, belirli bir işlevi yerine getirir. Programın modüler yapısı, sistemin esnekliğini, genişletilebilirliğini ve bakımının kolaylığını sağlar.

Ana Modüller ve İşlevleri

- MyProcess Modülü:
  - İşlev: Bu modül, sistemdeki her bir işlemi temsil eder. İşlemlerin özellikleri (örneğin, varış zamanı, öncelik, çalışma süresi) ve durumları bu sınıf tarafından yönetilir.
  - Gerekçe: Her işlemi ayrı bir nesne olarak modellemek, işlemlerin yönetimini ve izlenmesini kolaylaştırır.
- Queue Modülü:
  - İşlev: İşlemleri FIFO (First-In, First-Out) mantığına göre sıraya alır ve yönetir.
  - Gerekçe: Bu modül, işlemlerin adil ve sıralı bir şekilde işlenmesini sağlar, böylece her işleme eşit şans verilir.
- Resource Modülü:
  - İşlev: Sistem kaynaklarını (yazıcılar, tarayıcılar vb.) temsil eder ve yönetir.
  - Gerekçe: Kaynakların etkin yönetimi, çakışmaların önlenmesi ve sistem kaynaklarının optimal kullanımı için kritik öneme sahiptir.
- FileRead Modülü:
  - İşlev: Gerekli konfigürasyonları ve işlem bilgilerini dosyalardan okur.
  - Gerekçe: Bu modül, sistem yapılandırmasını esnek ve değiştirilebilir kılar.
- Colors Modülü:
  - İşlev: Arayüzde ve çıktılarda kullanılmak üzere renk kodları sağlar.
  - Gerekçe: Kullanıcı arayüzünün görsel açıdan çekici ve anlaşılır olmasını sağlar.

Arayüzler ve İşlevleri

Projemizde, modüller arasındaki etkileşimi yönetmek için çeşitli arayüzler kullanılmaktadır. Bu arayüzler, modüllerin birbiriyle uyumlu ve esnek bir şekilde çalışmasını sağlar. Örneğin, `Queue` modülü, `MyProcess` nesnelerini sıraya alırken,

`Resource` modülü, bu işlemlerin kaynak taleplerini yönetir. Arayüzler, bu modüllerin birbiriyle etkileşimini düzenler ve sistemin bütünsel olarak çalışmasını sağlar.

Çok Düzeyli Görevlendirme Şeması ve Gerçek İşletim Sistemleriyle Karşılaştırma

Gerçek İşletim Sistemlerinde Görevlendirme Şemaları

Gerçek işletim sistemlerinde, görevlendirme ve kaynak yönetimi karmaşık ve çok katmanlıdır. Bu sistemler, genellikle öncelik tabanlı, zaman paylaşımlı veya karma bir yaklaşım kullanır. Öncelik tabanlı sistemler, işlemleri önemlerine göre sıralar; zaman paylaşımlı sistemler, işlemlere belirli zaman dilimleri atayarak adil bir kaynak dağılımı sağlar.

Projede Kullanılan Çok Düzeyli Görevlendirme Şeması

Projemizdeki görevlendirici, basit bir FIFO kuyruk yapısı ve öncelikli sıralama mekanizması kullanır. Bu yapı, işlemleri öncelik düzeylerine göre sıralar ve böylece kaynak tahsisinde adil bir yönetim sağlar.

Neden Çok Düzeyli Bir Yaklaşım?

Çok düzeyli bir yaklaşımın kullanılmasının ana nedeni, kaynakların etkin kullanımını sağlamak ve işlemlerin gereksinimlerine göre önceliklendirilmesidir. Bu yaklaşım, özellikle karmaşık ve çeşitli iş yüklerinin olduğu durumlarda, kaynakların verimli kullanımını ve sistem performansının optimizasyonunu sağlar.

Eksiklikler ve İyileştirme Önerileri

Projemizin görevlendirme şeması, basitliği ve anlaşılabilirliği ile avantaj sağlasa da, gerçek dünya senaryolarının karmaşıklığını tam olarak yansıtmayabilir. İyileştirme önerileri şunlardır:

- Dinamik Önceliklendirme: İşlemlerin önceliklerinin, çalışma zamanında değişebilmesi, sistem yanıt süresini iyileştirebilir.
- Gelişmiş Bellek Yönetimi: Bellek fragmentasyonunu azaltan ve bellek kullanım verimliliğini artıran algoritmalar (örneğin, Best Fit veya Buddy System) entegre edilebilir.
- Kaynak Kısıtlamalarını Daha İyi Yönetme: Kaynak kısıtlamalarını ve çakışmalarını yönetmek için daha gelişmiş algoritmaların kullanımı, sistem stabilitesini artırabilir.

Bellek ve Kaynak Ayırma Şemaları

Bellek ve kaynak yönetimi, işletim sistemlerinin en kritik bileşenlerindendir. Projemizdeki basit FIFO ve öncelikli kuyruk yaklaşımı, bu alanlarda da benzer prensiplere dayanmaktadır. Ancak, gerçek işletim sistemlerindeki gibi, daha dinamik ve esnek bellek yönetimi tekniklerinin entegrasyonu, sistem performansını ve kararlılığını önemli ölçüde artırabilir.

Programın Çalışma Süreci

- Başlangıç ve Yapılandırma:
  - Program başlatıldığında, ilk olarak `FileRead` modülü devreye girer. Bu modül, gerekli konfigürasyon bilgilerini ve işlem detaylarını dosyalardan okur.
  - Okunan veriler, sistemdeki işlemleri ve kaynakları temsil eden `MyProcess` ve `Resource` nesnelerinin oluşturulmasında kullanılır.
- İşlem Kuyruğunun Oluşturulması:
  - `MyProcess` nesneleri, `Queue` modülü tarafından yönetilen kuyruklara yerleştirilir. Bu kuyruklar, işlemleri FIFO (First-In, First-Out) veya öncelik düzeyine göre sıralar.
  - Bu sıralama, işlemlerin işletim sistemindeki sırayla işlenmesini temsil eder.
- Kaynak Yönetimi:
  - `Resource` modülü, sistem kaynaklarını yönetir. Her `Resource` nesnesi, bir sistem kaynağını (örneğin, yazıcı, tarayıcı) temsil eder ve bu kaynakların kullanım durumunu takip eder.
  - İşlemler, gerekli kaynakları talep eder ve kaynaklar müsait olduğunda bu kaynaklar tahsis edilir.
- İşlem Yürütme ve Kaynak Tahsisi:
  - Program, kuyruktaki işlemleri sırasıyla işler. Her bir işlem, gerekli kaynakları kullanarak belirli bir işlemi tamamlar.
  - İşlem tamamlandığında veya belirli bir zaman dilimi sona erdiğinde, işlem kaynakları serbest bırakır ve kuyruktan çıkarılır.
- İşlemlerin Tamamlanması:
  - Tüm işlemler kuyruktan geçirildikten ve gerekli işlemleri tamamladıktan sonra, program sonlanır.
  - Bu süreç, bir işletim sistemindeki işlem yönetimi ve kaynak tahsisinin temel bir simülasyonunu oluşturur.

Genel Tartışma ve Analiz

Projedeki Mevcut Yaklaşımın Analizi

Projemiz, işletim sistemlerinin temel fonksiyonlarını simüle etmek için basit ve modüler bir yapı kullanıyor. Bu yaklaşım, öğrenme ve eğitim amaçlı olduğu için, gerçek dünya işletim sistemlerinin karmaşıklığını ve verimliliğini tam olarak yansıtmayabilir. Ana odak, işlemlerin yönetimi ve kaynak tahsisinin basitleştirilmiş bir modeli üzerine kurulmuştur.

- Görevlendirme ve Kaynak Yönetimi: Kullanılan FIFO ve öncelikli kuyruk yapısı, işlemlerin sıralanmasında basit ve anlaşılır bir metod sağlar. Ancak, bu yaklaşım, gerçek zamanlı veya karmaşık senaryolarda yetersiz kalabilir.
- Bellek Yönetimi: Bellek yönetimi, projede yeterince detaylandırılmamış olabilir. Gerçek işletim sistemlerinde bellek yönetimi, sistem performansı üzerinde önemli bir etkiye sahiptir.

Geliştirilebilecek Alanlar

- Gelişmiş Görevlendirme Algoritmaları: Öncelik tabanlı veya zaman dilimli (time-slice) yaklaşımların entegrasyonu, işlemlerin daha etkin bir şekilde yönetilmesini

sağlayabilir. Özellikle, farklı önceliklere ve gereksinimlere sahip işlemlerin daha adil ve verimli bir şekilde işlenmesine olanak tanır.
- Dinamik Bellek Yönetimi: Bellek tahsisinde kullanılan algoritmaları geliştirmek (örneğin, Best Fit, Worst Fit veya Buddy System), bellek kullanım verimliliğini ve sistem performansını artırabilir.
- Kaynak Kısıtlamalarının İyileştirilmesi: Kaynak kısıtlamalarını ve çakışmalarını daha iyi yönetmek için gelişmiş algoritmaların kullanımı, özellikle çoklu işlem ve çok kullanıcılı sistemlerde önemlidir.

Potansiyel Hatalar ve Zorluklar

- Kaynak Yönetimi: Kaynakların etkin yönetilmemesi, özellikle çoklu kullanıcı ve çoklu işlem ortamlarında ciddi performans sorunlarına yol açabilir.
- Ölümcül Kilitlenmeler (Deadlocks): Eğer kaynak tahsis mekanizmaları dikkatli bir şekilde tasarlanmazsa, ölümcül kilitlenmeler meydana gelebilir.
- Bellek Sızıntıları ve Fragmentasyon: Etkin olmayan bellek yönetimi, bellek sızıntılarına ve fragmentasyona yol açabilir, bu da zamanla sistem performansını düşürebilir.