SAKARYA ÜNİVERSİTESİ

BİLGİSAYAR VE BİLİŞİM BİLİMLERİ FAKÜLTESİ

BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ

İşletim Sistemleri Proje Ödevi Raporu

AD	Numara	Grup
YAZAN ALHASAN	B211210580	1-C
MHD OMAR SHAMIEH	B211210553	1-C
LİNA ELSEVADİ	G201210570	1-B
MUHAMMED KEREK	B211210582	1-B
ASEM HAGİ HASAN	B201210567	1-B

Github linki: https://github.com/Yzn-al-hsn/Isletim-Sistemleri.git

Proje Ana Başlığı

Varsayımsal İşletim Sistemi Test Ortamı (HOST), sınırlı kullanılabilir kaynakların kısıtlamaları dahilinde çalışan, dört seviyeli öncelikli bir süreç dağıtıcısına sahip, çok programlı bir sistemdir.

Problem Tanıtımı

Bu proje, işletim sistemlerinin temel kavramlarını modelleyen bir simülasyon geliştirmektedir. Simülasyon, gerçek zamanlı ve kullanıcı işlemlerini yöneten farklı öncelik seviyelerine sahip bir çoklu programlama sistemini içermekte. İşlemler, varış zamanlarına, önceliklerine, gereken işlemci zamanına, hafıza boyutlarına ve talep ettikleri diğer kaynaklara göre sıralanıyor. İşlemler hazır duruma geldiğinde, İlk Gelen İlk Hizmet Alır (FCFS) mantığı ile yüksek öncelikli gerçek zamanlı işlemler işletiliyor. Eğer yeterli kaynak ve hafıza mevcutsa, daha düşük öncelikli kullanıcı işlemleri uygun öncelik kuyruğuna aktarılıyor. Sistem, zaman dilimi (quantum) esasına dayalı bir Round Robin algoritması kullanarak kullanıcı işlemlerini çalıştırıyor ve gerektiğinde önceliklerini değiştiriyor. Bir işlem tamamlandığında veya zamanı dolduğunda, kullandığı kaynaklar tekrar dağıtım için sistem tarafından serbest bırakılıyor. Bütün işlemler tamamlandığında veya kuyruklarda işlem kalmadığında sistem sonlandırılıyor.

Bellek Tahsisi Algoritmaları

Projemizde, bellek tahsis algoritmalarına dair spesifik bir uygulamadan bahsetmedik. Bu nedenle, simülasyonumuzda First-Fit, Best-Fit veya Worst-Fit gibi klasik bellek tahsis algoritmalarının herhangi birini açıkça kullanmadık. Ancak, simülasyonumuzun anlatımında süreçlerin bellek ihtiyaçları ve diğer kaynak taleplerinin nasıl yönetileceğini genel hatlarıyla ele aldık.

Simülasyonumuzda süreçler, varış zamanlarına ve önceliklerine göre sıralanıyor ve çalıştırılıyor. Gerçek zamanlı süreçler ve kullanıcı süreçleri, kendi içlerinde öncelik düzeylerine ve varış zamanlarına göre yönetiliyor. Her süreç için CPU zamanı, bellek ihtiyacı ve diğer kaynak istekleri göz önünde bulunduruluyor ve bir süreç çalıştırıldığında, bu kaynakların tahsis edildiği varsayılıyor.

Özetle, simülasyonumuzda bellek tahsisinin doğrudan bir algoritması uygulanmamıştır; bunun yerine, süreçlerin ve kaynakların yönetimi için daha soyut bir yaklaşım benimsemişizdir. Gerçek bir işletim sistemi simülasyonu geliştirmekte olsaydınız, bellek tahsis algoritmasının seçimi ve uygulanması, simülasyonun gerçekçiliği ve doğruluğu için önemli bir faktör olacaktı.

Projede Kullanılan Yapılar

Projemizde kullandığımız yapı sınıflarını aşağıdaki şekilde açıklayabiliriz:

ProcessQueue.Java

- **ProcessQueue** sınıfı, işletim sistemi simülasyonunda süreçleri (process) yönetmek için kullanılır. Bu sınıf, süreçlerin sıraya alınması, saklanması ve sırayla işlenmesi için bir kuyruk yapısını temsil eder.
- İşlemcinin çalıştıracağı süreçleri belirleyen planlayıcı (scheduler) tarafından kullanılır.
- Her bir süreç, varış zamanına, önceliğine veya diğer kriterlere göre **ProcessQueue** içinde uygun bir şekilde sıralanır.
- Süreçler, planlayıcı tarafından **ProcessQueue**'dan çekilip işlemciye gönderilirken, öncelik veya varış zamanına göre seçilir.
- Bu sınıf, özellikle İlk Gelen İlk Hizmet Alır (First-Come, First-Served - FCFS) veya öncelik tabanlı planlama algoritmalarında önemli bir rol oynar.

ResourceManager.Java

- **ResourceManager** sınıfı, simülasyonda bulunan kaynakların (bellek, yazıcı, tarayıcı gibi I/O cihazları) yönetiminden sorumludur.
- Her bir süreç belirli kaynakları talep edebilir ve **ResourceManager**, bu kaynakların süreçler arasında adil ve etkin bir şekilde tahsis edilmesini sağlar.
- Kaynak çakışmalarını önlemek, bellek gibi kaynakların maksimum kullanımını sağlamak ve süreçlerin taleplerini karşılamak için bu sınıf içinde algoritmalar ve yönetim stratejileri bulunur.
- Ayrıca, süreçler tamamlandığında veya sonlandırıldığında, **ResourceManager** tahsis edilen kaynakların serbest bırakılmasını ve tekrar kullanılabilir hale gelmesini sağlar.

Process .Java

- **Process** sınıfı, simüle edilen işletim sistemindeki bir sürecin kendisini temsil eder. Bu sınıf, sürecin PID (Process Identifier), önceliği, CPU zamanı, gereken bellek miktarı ve talep ettiği diğer kaynaklar gibi bilgilerini içerir.
- Ayrıca sürecin durumunu (hazır, çalışıyor, tamamlandı gibi) da takip eder.
- **Process** nesneleri, planlayıcının işleyeceği süreçlerin bilgilerini taşır ve **ProcessQueue**'dan ile **ResourceManager** tarafından kullanılır.
- Bu sınıf, süreçlerin oluşturulması, çalıştırılması ve sonlandırılması sırasında gerekli tüm bilgileri ve işlemleri kapsar.

Projede Kullanınlan Çeşitli Mödüller Ve Ana İşlevler

Process.Java

- Process sınıfının kurucu fonksiyonu procese ait özelliklerine atama yapar.
- **RunProcess** fonksiyonunun görevi de Processi Tetikleyerek çalıştırır.
- **LowerPriority** fonksiyonunun görevi processin bir sonraki tur için önceliğini düşürmesidir.

ProcessQueue.Java

- Kurucu fonksiyonu PriorityQueue sınıfından bir nesne oluşturur.
- **Enqueue** fonksiyonu bir prosessi parametre olarak alır ve Pocess kuyruğuna koyar.
- **Dequeue** fonksiyonu en yüksek öncelikli processi kuyruktan çıkarır ve bu Processi döndürür.
- **Peek** fonksiyonu kuyrukta bulunan ilk processi kuyruktan çıkarmadan döndürür.

ResourceManageer.Java

- İçerisinde kullandığımız kaynakları değişkenler olarak bulunduruyor.
- İçerisinde iki fonksiyoun bulumaktadır.
- **AllocateResources fonksiyonu** bellek atama işleminden sorumlu olan fonksiyondur.
- **DeAllocateResources** fonksiyonu da tahsis edilen kaynakları serbest bırakır.

Dispatcher.Java

- Processi, bulunduğu kuyruğu, ve kaynakları birbirine bağlayarak kaynakları yönetmek için kullandığımız Görevlendirici sınıfıdır.
- **DispatchNextProcess** fonksyionu kuyruktaki ilk processi çıkarır ve değerini saklar, processin durumunu Çalışıyor olarak değiştirir, Processi çalıştırır ve kaynakları tahsis eder, processin durumunu Tamamlandı'ya çeevirir ve en son kaynakları serbest bırakır.
- **PrintProcess** fonksiyonu bir Processi parametre olarak alır, kontrol işlemlerinden sorumludur ve kontrol sağlanıyorsa processin bilgilerini ekrana yazdırır.

FileIO.Java

- Dosyaları okuma işlemlerinden sorumludur.
- Bir doyayı alır, okur ve okuduğu her satırı bir String listesine ekler.

SimulationController.Java

- ProcessQueue, Dispatcher, FileIO sınıflarını kullanır.
- İki fonksiyon bulundurur:
- **Inıtialize** fonksiyonu FileIOdan bir String listesi alır ve bu listeyi bölümlere ayırır.
- Bu bölümlerin her değişkeni processe ait bir değişken olarak tanımlanır ve bu değişkeni ProcessQueue sınıfına ekler

- **Start** fonksiyonunun görevi dispatcher'ı çağırır ve Initialize fonksiyonunda oluşturduğumuz ProcessQueue'yu yönetir.

FCFSScheduler.Java

- İlk giren ilk işlem görür algoritmasının mantığını çalıştırır.

RoundRobinFeedBackScheduler

- Çevrimsel sıralı algoritmasının mantığını çalıştırır.

UserFeedBackScheduler

- İçerisideki **Schedule** foknsiyonunun görevi Processleri öncelik sıralamalarına kuyruğa koyar.
- **Dispatcher** fonksiyonunun görevi Kuyrukta bulunan processleri sıra ile almaktır.

Main.Java

- Simülasyonun başlatıldığı nokta olarak kabul edebiliriz.
- **StratDispatcher** fonksiyonunun görevi SimulationController sınıfından bir nesne oluşturup tetiklemektir.

Yapılabilen Ek İyileştirmeler

Processleri yönetmek tarafı ideal olduğunu kabul edebiliriz, yalnız yaptığımız simülasyonun gerçek bir işletim sistemi olabilmesi için arayüze, dosya yönetimi ve klasör yönetimine ihtiyacımız vardır.

SON OLARAK BU PROJE İÇİN BÜTÜN HOCALARA TEŞEKKÜR EDERİZ.