T.C. SAKARYA ÜNİVERSİTESİ

BİLGİSAYAR VE BİLİŞİM BİLİMLERİ FAKÜLTESİ

BİLGİSAYAR MÜHENDİSLİĞİ

İŞLETİM SİSTEMLERİ - PROJE/TASARIM RAPORU

ÖĞRENCİLER VE NUMARALARI:

Nida KORKMAZ - G201210018

Muhammed Furkan ÖZGEN - G201210027

Sümeyye ALMAS- G201210063

Mustafa CEYLAN - G201210375

Abdulkadir Eren YURTASLAN- G201210081

GITHUB LİNKİ:

https://github.com/SumeyyeA/IsletimSistemleriProje

GİRİŞ

Programımız, FCFS (First Come First Serve) yöntemi kullanan bir görevlendirici şemasını simüle eden bir Java uygulamasıdır. Program, kullanıcı tarafından girilen prosesleri sırayla çalıştırır ve bu proseslerin çalışma sürelerini gösterir.

Programımızın genel yapısı aşağıdaki gibi olmuştur:

Arayüzler: Programın kullanıcı arayüzlerini tanımlayan modüldür. Bu modül, kullanıcının programa girdiği verileri alıp işleyen ve çıktılarını gösteren arayüzleri içerir.

Görevlendirici: Bu modül, sistem üzerinde çalışan proseslerin simülasyonunu gerçekleştirir. Görevlendirici, her sevk edilen iş için yeni bir proses oluşturur ve bu proseslerin fonksiyonlarını yönetir. Görevlendirici ayrıca, proseslerin askıya alınıp devam ettirilmesi veya sonlandırılması gibi işlemleri de gerçekleştirir.

Prosesler: Görevlendirici tarafından oluşturulan prosesleri temsil eden modüldür. Bu modül, proseslerin fonksiyonlarını tanımlar ve bu fonksiyonları gerçekleştirir. Prosesler ayrıca, kendilerine atanan rastgele oluşturulmuş renk şemasını kullanarak yazdırır.

Bu şekilde, programımızın genel yapısı oluşmuştur. Arayüzler modülü sayesinde, giris.txt dosyasından programa veri girişi yapılabilir ve bu verilerin işlenmesi sonucu oluşan çıktılar görüntülenebilir. Görevlendirici modülü ise, sistem üzerinde çalışan proseslerin simülasyonunu gerçekleştirir ve bu proseslerin fonksiyonlarını yönetir. Prosesler modülü ise, proseslerin fonksiyonlarını tanımlar ve bu fonksiyonları gerçekleştirir.

Gerçek işletim sistemlerinde, çok düzeyli görevlendirici şemaları kullanılır. Bu şemalar, çeşitli işlemlerin yürütülmesi için gereken kaynakları tahsis etmek ve bu kaynakları yönetmek amacıyla kullanılır.

Programımızın çeşitli modülleri şunlardır:

Queue sınıfı: Bu sınıf, proseslerin sırayla saklandığı kuyruk yapısını oluşturur. Bu sınıf, proseslerin eklenmesi, çıkarılması ve kuyruğun boş olup olmadığı gibi işlevleri içerir.

Process sınıfı: Bu sınıf, her bir prosesi temsil eden nesneleri oluşturur. Bu sınıf, prosesin kimliği, çalışma süresi ve renk şeması gibi özellikleri içerir.

ProcessBuilder sınıfı: Bu sınıf, işletim sistemi işlemleri oluşturmak için kullanılır. Her ProcessBuilder örneği, bir süreç nitelikleri koleksiyonunu yönetir.

Main sınıfı: Bu sınıf, programın çalıştırıldığı ana sınıftır. Bu sınıf, kullanıcıdan proseslerin girilmesini ve proseslerin çalıştırılmasını yönetir.

Görevlendirici şemaları hakkında bilgiler ve eksiklikler:

Gerçek işletim sistemleri, genellikle çok düzeyli görevlendirici şemalarını kullanırlar. Bu şemalar, prosesleri çalıştırmak için kullanılan kaynakları (örneğin, CPU, bellek gibi) daha verimli bir şekilde yönetirler. Bu sayede proseslerin çalışmaları daha hızlı ve verimli hale getirilir.

Ayrıca çok düzeyli görevlendirici şemaları, prosesleri önceliklerine göre çalıştırır. Bu sayede, öncelikli prosesler daha önce çalıştırılır ve diğer proseslerden daha fazla kaynak alır. Bu sayede, öncelikli proseslerin çalışmaları daha hızlı ve verimli hale getirilir.

Bu şekilde, çok düzeyli görevlendirici şemaları gerçek işletim sistemleri tarafından kullanılır çünkü proseslerin çalışmalarını daha verimli hale getirirler. Bu şemalar, prosesleri birbirinden ayrı çalıştırarak prosesler arasında bir etkileşim olmamasını sağlar ve prosesleri önceliklerine göre çalıştırarak öncelikli proseslerin çalışmalarını hızlandırır.

Programımızda kullandığımız çok düzeyli görevlendirici şeması da bu şemaların bir örneğidir. Bu şema, prosesleri FCFS yöntemi kullanarak sırayla çalıştırır ve bu sayede prosesler arasında bir etkileşim olmaz. Ayrıca, bu şema prosesleri önceliklerine göre çalıştırır ve öncelikli prosesler daha önce bitirilir.

Ancak bu şema bazı eksikliklere de sahiptir. Örneğin, bu şemalar yüksek bir bellek ve kaynak yönetim overhead'i gerektirir ve bu nedenle daha yavaş çalışabilir. Ayrıca, bu şemalar proseslerin bellek ve kaynak ihtiyaçları değiştiğinde otomatik olarak bellek ve kaynak miktarını ayarlar, ancak bu da proseslerin verimini azaltabilir.

Başka bir örnek vermek gerekirse, prosesler arasında öncelikler farklı olabilir ve bu durumda öncelikli prosesler daha önce bitirilebilir. Ayrıca, proseslerin çalışma süreleri farklı olabilir ve bu da işletim sistemini yavaşlatabilir.

Olası iyileştirmeler:

Prosesler arasında daha adil bir kaynak dağılımı sağlanması:

Bazen bazı prosesler daha fazla kaynak tüketebilirler ve bu nedenle diğer proseslerin çalışmalarını olumsuz etkileyebilir. Bu nedenle prosesler arasında daha adil bir kaynak dağılımı sağlanması, olası bir iyileştirme olabilir.

Prosesler arasında daha verimli bir haberleşme sağlanması:

Prosesler arasında bilgi paylaşımı yapılması gerekebilir. Ancak bu haberleşme verimli değilse proseslerin çalışmaları olumsuz etkilenebilir. Bu nedenle, prosesler arasında daha verimli bir haberleşme sağlanması, olası bir iyileştirme ihtimalidir.

Dinamik Şemalar:

Dinamik şemalar, proseslerin bellek ve kaynak ihtiyaçları değiştiğinde de verimli bir şekilde çalışabilir ancak yüksek bir bellek ve kaynak yönetim overhead'i gerektirmezler. Bu sayede, proseslerin çalışmaları daha hızlı ve verimli hale getirilebilir.

Bellek ve kaynak ayırma şemalarının daha verimli hale getirilmesi:

Gerçek işletim sistemlerinde, bellek ve kaynakların doğru bir şekilde yönetilmesi önemlidir. Bu nedenle, bellek ve kaynak ayırma şemalarının daha verimli hale getirilmesi muhtemel bir iyileştirme sayılabilir.

Şimdi biraz bellek ve kaynak ayırma şemalarından bahsedelim.

Bellek ve kaynak ayırma şemaları, işletim sistemlerinde proseslerin bellek ve diğer kaynakları nasıl kullanacağını belirleyen yöntemlerdir. Bu şemalar sayesinde, prosesler arasında bellek ve kaynakların daha adil bir şekilde dağıtılması sağlanır ve proseslerin çalışmaları daha verimli hale getirilir.

Bellek Ayırma şemaları:

Statik Bellek Ayırma: Bu şema, her prosesin ihtiyaç duyduğu bellek miktarını önceden belirler ve bu miktarı proses için ayırır. Proseslerin bellek ihtiyaçları değişmediği sürece verimli bir şekilde çalışır. Ancak proseslerin bellek ihtiyaçları değiştiğinde bu şema verimsiz hale gelebilir.

Dinamik Bellek Ayırma: Bu şema, proseslerin bellek ihtiyaçları değiştiğinde bellek miktarını otomatik olarak ayarlar. Bu sayede proseslerin bellek ihtiyaçları değiştiğinde de verimli bir şekilde çalışabilir. Ancak bu şema, yüksek bir bellek yönetim overhead'i gerektirir ve bu nedenle daha yavaş çalışabilir.

Kaynak Ayırma şemaları:

Statik Kaynak Ayırma: Bu şema, her prosesin ihtiyaç duyduğu kaynak miktarını önceden belirler ve bu miktarı proses için ayırır. Proseslerin kaynak ihtiyaçları değişmediği sürece verimli bir şekilde çalışır. Ancak proseslerin kaynak ihtiyaçları değiştiğinde bu şema verimsiz hale gelebilir.

Dinamik Kaynak Ayırma: Bu şema, proseslerin kaynak ihtiyaçları değiştiğinde kaynak miktarını otomatik olarak ayarlar. Bu sayede, proseslerin kaynak ihtiyaçları değiştiğinde de verimli bir şekilde çalışabilir. Ancak, bu şema yüksek bir kaynak yönetim overhead'i gerektirir ve bu nedenle daha yavaş çalışabilir.

Özetle, bellek ve kaynak ayırma şemalarının statik ve dinamik olmak üzere iki çeşidi vardır. Statik şemalar proseslerin bellek ve kaynak ihtiyaçları değişmediği sürece verimli bir şekilde çalışırken, dinamik şemalar ise proseslerin bellek ve kaynak ihtiyaçları değiştiğinde de verimli bir şekilde çalışabilir ancak yüksek bir bellek ve kaynak yönetim overhead'i gerektirir ve bu nedenle daha yavaş çalışabilir. Bu şemalar, proseslerin bellek ve kaynak ihtiyaçları değiştiğinde otomatik olarak bellek ve kaynak miktarını ayarlar. Yukarıda da olası iyileştirmelerde bellek ve kaynak ayırma şemalarının rolünden bahsetmeye çalıştık.

MAIN.JAVA ÇIKTISI