T.C.

SAKARYA ÜNİVERSİTESİ

BİLGİSAYAR VE BİLİŞİM BİLİMLERİ FAKÜLTESİ

BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ

İŞLETİM SİSTEMLERİ ÖDEV RAPORU

2C - G201210076 - MUHAMMET EFDAL ŞAHİN

2B - G211210066 - MUSTAFA KARADENİZ

2B - G211210042 - MUSTAFA HACI BARAN

2C - G221210382 - YETGİN AKCAN

1B - B231210501 - NECMİ ERTÜRK

CPU Dispatcher

Github Link: https://github.com/karadnz/CPU_Dispatcher

Dispatcher Nedir?

Dispatcher, işletim sisteminin bir parçası olarak, CPU'ya hangi işlem veya iş parçacığının ne zaman ve ne kadar süreyle erişeceğini kontrol eden bir sistemdir. İşletim sisteminin CPU kullanımını yöneten temel bir programdır. Dispatcher'ın ana görevleri şunlardır:

Ana Görevler

1. **CPU Zamanlaması:**
   - Hazır kuyruğundaki işlemler arasından bir sonraki çalıştırılacak işlemi seçer.
   - Hazır kuyruğundaki işlemler arasından bir sonraki çalıştırılacak işlemi seçer.
2. **Kontekst Değişimi:**
   - Bir işlemin CPU kullanımını sonlandırdığında veya kesintiye uğradığında, dispatcher işlemin durumunu kaydeder ve yeni seçilen işlemin durumunu yükler.
   - Bu süreç "kontekst değişimi" olarak bilinir ve CPU'nun farklı işlemler arasında hızlı geçiş yapmasını sağlar.
3. **Kaynak Yönetimi:**
   - İşlemci, bellek ve giriş/çıkış gibi sistem kaynaklarının verimli kullanımını sağlar.

Dispatcher, işlemcinin verimli ve etkin bir şekilde kullanılmasını sağlayarak işletim sisteminin çoklu görev kabiliyetinin temelini oluşturur. Bu, hem sistem performansını artırır hem de kullanıcılara birden fazla uygulamanın eş zamanlı olarak çalıştığını hissettirir.

İşletim Sistemlerinde Bellek Tahsis Algoritmaları

İşletim sistemlerinde bellek tahsis algoritmaları, işlem ve programlara bellek kaynaklarının tahsis edilmesi ve boşaltılmasını yönetmek için kullanılan yöntemlerdir. Bu algoritmalar, uygulamaların işlev görmesi için gerekli belleğe sahip olmalarını sağlarken mevcut bellek kaynaklarının kullanımını en üst düzeye çıkarmak için hayati öneme sahiptir. İşte yaygın bellek tahsis algoritmalarından bazıları:

1. **İlk Uygun (First-Fit) Algoritması:**

- **Açıklama:** İstek için yeterince büyük olan ilk bellek bloğunu tahsis eder.
- **Artıları:** Basit ve hızlıdır.
- **Eksileri:** Zamanla, kullanılamayan küçük boşluklar belleğin her tarafına dağıldıkça parçalanmaya yol açabilir.

2. En İyi Uygun (Best-Fit) Algoritması:

- **Açıklama:** İsteği karşılayacak en küçük bloğu bulmak için tüm belleği arar.
- **Artıları:** En küçük artık bellek parçalarını üretme eğilimindedir, böylece parçalanmayı azaltır.
- **Eksileri:** Tüm blokları araştırması gerektiği için daha fazla hesaplama yoğunluğuna sahiptir ve daha küçük kullanılamaz alanlara yol açabilir.

3. En Kötü Uygun (Worst-Fit) Algoritması:

- **Açıklama:** Tahsis için mevcut en büyük bellek bloğunu seçer.
- **Artıları:** Küçük parçalar üretme olasılığı daha düşüktür, bu nedenle boyutları çok değişken olan tahsisler için uygun olabilir.
- **Eksileri:** Belleğin büyük kısımlarının kullanılmadan kalmasına yol açabilir.

4. Sonraki Uygun (Next-Fit) Algoritması:

- **Açıklama:** İlk uygun algoritmaya benzer, ancak aramaya belleğin başından değil, önceki tahsisin yapıldığı yerden başlar.
- **Artıları:** Belirli senaryolarda baştan başlamadığı için ilk uygun algoritmadan daha hızlı olabilir.
- **Eksileri:** İlk uygun gibi, parçalanmaya yol açabilir.

5. Arkadaş Sistemi (Buddy System):

- **Açıklama:** Parçalanmayı azaltmaya çalışarak belleği bölümlere ayırır. Her blok, bir boyut isteğine uyacak şekilde yarıya bölünür.
- **Artıları:** Daha az parçalanma ile belleği verimli bir şekilde yönetir.
- **Eksileri:** Uygulamak ve sürdürmek karmaşık olabilir.

6. Yığın Tahsis (Slab Allocation):

- **Açıklama:** Genellikle çekirdek bellek tahsisi için sabit boyutlu nesneler için kullanılır. Aynı boyuttaki nesneleri bir yığın içinde gruplar.
- **Artıları:** Aynı boyuttaki nesnelerin sık tahsis ve boşaltmaları ile ilişkili gider ve parçalanmayı azaltır.
- **Eksileri:** Daha az esnektir, esas olarak sabit boyutlu tahsisler için uygundur.

7. Segmentasyon:

- **Açıklama:** Belleği, kod, yığın ve veri segmentleri gibi bir işlemin mantıksal bölümlerine göre segmentlere ayırır.
- **Artıları:** İşlemin yapısına özgü bellek kullanımını verimli ve esnek bir şekilde sunar.
- **Eksileri:** Dış parçalanmaya yol açabilir.

8. Sayfalama:

- **Açıklama:** Belleği sabit boyutlu sayfalara böler. Fiziksel bellek de aynı boyutta sayfa çerçevelerine bölünür ve sayfalar bu çerçevelere yüklenir.
- **Artıları:** Dış parçalanmayı azaltır ve bellek tahsisini basitleştirir.
- **Eksileri:** İç parçalanmaya yol açabilir ve sayfa eşleme için ek donanım desteği gerektirir.

Bizim projemizde ise zamani gelen prosesler once onceliklerine gore siralanirlar. Daha sonra ise ayni oncelige sahip olanlar arasinda az bellek isteyenden cok bellek isteyene gore siralanirlar. Boylelikle kuyruga daha Fazle proses alinabilir.

Görevlendirici tarafında kullanılan yapılar ve gerekçeleri

- **Liste ve dizi Listesi:** proseslerin bilgilerini tutmak için ve zamanı gelen prosesleri tutmak için kullanıldı.
- **Kuyruk ve bağlı liste:** gerçek zamanlı ve kullanıcı zamanlı proseslerinin "hazır" olanlarını tutmak için kullanıldı. Gerçek zamanlı kuyruktan 1 tane ve kullanıcı zamanlı proseslerden 3 tane oluşturulup kullanıldı.
- **İterator:** 3 farklı yerde kullanıldı. Kullanılmasının sebebi doğrudan liste ve kuyruklardan üzerinden eleman kaldırma işleminde eşzamanlılık hatası veriyordu.

Program yapısının ve bireysel modüllerin tanımı ve gerekçesi

- **Proses.java:** Prosesin bilgilerini ve durumlarini yazdiran fonksiyonlari bulunduran sınıf.
- **BeklemeListesi.java:** Dosyadan prosesleri okuyan ve listede tutan sınıf.
- **Gorevlendirici.java:** Kaynakları/bellekleri güncelleyen, kaynak/bellek kontrolü yapan, proseslere gereken kaynakları/bellekleri atayan, zamanı gelen prosesleri bulan ve uygun kuyruklara atayan sınıf.
- **FCFS.java:** Gerçek zamanlı prosesleri çalıştıran kuyrukların üzerinde işlem yapan sınıf.
- **GBG.java:** Kullanıcı proseslerini çalıştıran kuyrukların üzerinde işlem yapan sınıf.
- **RR.java:** En düşük öncelikli kuyruğu RoundRobin çalıştıran sınıf.

Görevlendiricinin eksiklikleri ve olası iyileştirmeler

- Belleğin yanı sıra kaynaklara ve çalışma zamanına göre de sıralama eklenebilir.
- 20 saniye kuralı prosesin gelme süresinden ziyade cpu da geçirdiği süreye göre uygulanabilir.

Kuyruk özellikleri

- **FCFS.java:**
  - FCFS Gerçek zamanlı prosesler için kullanıldı.
  - Kuyruğun başındaki proses işleme sokuldu.
  - İşlem bitene kadar bekledi ve işlem bittikten sonra belleği geri iade etti.

- **GBG.java:**
  - Eğer Gerçek Zamanlı kuyruk boş ise çalışır.
  - Boş olmayan en yuksek öncelikli sıralayıcı çalışır.
  - 1 saniye çalışır ve proses zamanını azaltır ve prosesin önceliğini düşürür.
- **RR.java:**
  - Eger Gerçek Zamanlı kuyruk ve diğer kullanıcı kuyrukları boş ise çalışır.
  - Diğer kuyruklara proses gelene kadar kuyruktaki prosesleri işler.

Diger İşletim sistemleriyle karsilastirma

Hem Linux hem de Windows'ta, işletim sisteminin çekirdeği, daha sonra hangi işlemin yürütüleceğini seçmekten sorumlu olan bir görevlendirici içerir. Bir işlem çalışmaya hazır olduğunda, hazır işlemler kuyruğuna alınır. Görevlendirici, kuyruğun önündeki işlemi seçer ve CPU'yu bu işleme tahsis eder. Bu işlem, yürütmeyi bitirene veya bazı kaynakları beklerken bloke edilene kadar devam eder. İşlem yürütmeyi bitirdiğinde veya bloke edildiğinde, dağıtıcı kuyruktaki bir sonraki işlemi seçer ve CPU'yu buna tahsis eder. Linux ve Windows çekirdekleri arasındaki temel farklardan biri, Linux'un tamamen önleyici bir zamanlayıcı kullanmasıdır; bu, çekirdeğin, CPU'yu farklı bir işleme vermek için herhangi bir zamanda çalışan bir işlemi kesebileceği anlamına gelir. Buna karşılık, Windows çekirdeği yalnızca kısmen önleyicidir, yani bir işlemi yalnızca belirli koşullar altında, örneğin işlemin bir sistem çağrısını engellemesi veya işlemin kendisine ayrılan zaman dilimini kullanması durumunda kesintiye uğratabileceği anlamına gelir. Linux ve Windows tarafından kullanılan zamanlama algoritmalarında da farklılıklar vardır. Linux, Tamamen Adil Zamanlayıcı (CFS) ve Round Robin Zamanlayıcı dahil olmak üzere çeşitli zamanlama algoritmaları kullanırken Windows, önceliğe dayalı zamanlama ve zaman dilimlemenin bir kombinasyonunu kullanır. Bununla beraber bizim yazdığımız Dispatcher sınıfı, varış zamanı geçen süreye eşit ya da daha az olan proseslerin kaynaklarına ve belleğine bakıyor eğer donanımın herhangi bir prosese kaynak ayırmamış halindeki kaynaktan daha büyükse prosesi iptal ediyor. Ama eğer MAX kaynak ve bellekten daha az ise ama mevcut kaynak ve bellek boş değilse o zaman kaynağın boşalmasını bekleyecek. Eğer gerçek zamanlı bir proses ise sadece bellek uygunluğuna bakılır ve herhangi bir kaynağa ihtiyaç duymaz. Kaynak ve bellek yeterliyse öncelliğine göre ilgili kuyruğa eklenir. Kuyrukların çalışmasında bir hiyerarşi mevcuttur. Eğer gerçek zamanlı proseslerin tutulduğu kuyruk boş değil ise her zaman burası çalışır, eğer mevcut çalışan kullanıcı zamanlı proses varsa kesintiye uğrarar ve gerçek zamanlı kuyruktaki prosesin bitmesini bekler. Gerçek zamanlı prosesler, proses bitene kadar çalışır. Kullanıcı zamanlı kuyrukda da önceliğe göre 3 kuyruğa ayrılmıştır. Kuyruklardaki prosesler 1 saniye çalışır ve eğer bitmişse kaynakları ve belleği iade edilir. Eğer bitmediyse önceliği ve çalışma süresi 1 azaltılır. Eğer kullanıcı zamanlı 1 ve 2 öncelikli kuyruk boş ise ve kullanıcı zamanlı 3 öncellikli kuyruk dolu ise o zaman burada basit bir çevrimsel RR çalışır. 1 Saniye prosesi işler ve kuyruğun sonuna atar. Eğer daha yüksek öncelikli bir kuyruğa proses gelirse o zaman öncelikli kuyruk çalışır. Varış zamanı ile birlikte aktif olan

bulunma zamanı 20 saniyeyi aşarsa proses iptal edilir ve eğer kaynakları ve bellekleri varsa iade edilir. Tüm kuyruklar boş ve kuyruğa yerleşmeyen prosesleri tutan liste boş olursa görevlendirici sonlanır.