T.C.
SAKARYA ÜNİVERSİTESİ

BİLGİSAYAR VE BİLİŞİM BİLİMLERİ FAKÜLTESİ BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ
İŞLETİM SİSTEMLERİ ÖDEV RAPORU

2A - G201210357 - Ahmet Samet Şahin
1A- B191210075 - Nur Sena Aytar
1A - B191210012 - Ömer Tufan Ayhan
1A - B191210054 - Yasemin Muzıroğlu

PROF DR. AHMET ZENGİN

Github Linki : **https://github.com/omertayhan/TAYS**

SAKARYA

Ocak, 2023

İçindekiler

ÇOKLU PROGRAMLAMA SİSTEMİ

ÖZET

Ödev java diliyle gerçekleştirilmiştir. Program sınıfıyla beraber toplamda 9 adet sınıf bulunmaktadır.

Proses tiplerinin tutulduğu ProcessType sınıfı, proses durumlarının tutulduğu ProcessSituation sınıfı, proses'lerin tutulacağı yapı olan kuyruk yapısı için Queue sınıfı, parametre olarak verilen txt dosyasının okunması ve kuyruk içindeki proseslerin sıralanması için yazılmış metotların olduğu GeneralFunctions sınıfı, genel kuyruğa atılan proseslerin önceliklerine göre farklı kuyruklara ayrıştırılması ve ayrıştırılan kuyrukların kendi içerisinde sıralama algoritmalarını çağırdığı Interlayer sınıfı, yeni proses oluşturulması, prosesin çalıştırılması ve okunması gibi işlemlerin yapıldığı Process sınıfı, kesintiye uğramış proseslerin oluşturulması için InterruptedProcess sınıfı, dispatcher işlemlerinin yapıldığı ProcessDispatcher sınıfı ve son olarak parametre olarak verilen txt dosyasını okuyan ve gerekli işlemlerin başlatılmasını sağlayan Program sınıfı, projedeki sınıflardır. Geliştirilen Yazılım bölümünde tüm bu sınıflar ve işlevleri detaylı bir şekilde açıklanacaktır. Aynı zamanda kod içerisinde kodun anlaşılmasını kolaylaştırmak için bol sayıda açıklama satırı bulunmaktadır.

FCFS Nedir?

Bilgisayar bilimlerinin çeşitli alanlarında kullanılan bir yaklaşımdır. Bu yaklaşıma göre bir kaynak veya bir sıraya ilk gelenin ilk önce işini bitirerek çıkması hedeflenir. Örneğin CPU Scheduling (İşlemci zamanlama) problemi sırasında işlemciye gelen işlemlerin hangi sıra ile çalışacağı bu algoritmaya göre belirlenirse ilk gelen iş bitmeden ikinci iş başlayamaz. Bir sıra (queue) için benzeri durum düşünülürse sıraya ilk giren ilk çıkar (First In First Out, FIFO Fifo)

ROUND ROBİN ALGORİTMASI NEDİR, NASIL ÇALIŞIR ?

Bir zamanlama (scheduling) algoritmasıdır. Özellikle işletim sistemi tasarımında işlemcinin (CPU) zamanlamasında kullanılan meşhur algoritmalardan birisidir. Bu algoritmaya göre sırası gelen işlem, işlemcide işi bitmese bile belirli bir zaman biriminden sonra (time quadrant) işlemciyi terk etmek zorundadır.

Bu sayede işletim sisteminde kıtlık (Starvation) olma ihtimali kalmaz. Çünkü hiç bir zaman bir işlemin CPU'yu alıp diğer işlemlere sıra gelmesini engellemesi mümkün olmaz.

Örneğin aşağıda süreleri verilen işlemlerin aynı anda bekleme sırasına (Ready queue) geldiğini düşünelim ve round robin algoritmasına göre nasıl bir sıra ile çalıştırılacağına bakalım.

İşlem	CPU Zamanı
A	10
B	15
C	7

Yukarıda verilen bu işlemlerin Round Robin algoritmasına göre CPU'da çalışma süreleri ve sırasıyla CPU'da yer değiştirmeleri (context switch) aşağıda verilmiştir:

Zaman	İşlem	Çalışma	Kalan
0	A	3	7
3	B	3	12
6	C	3	4
9	A	3	4
12	B	3	9
15	C	3	1
18	A	3	1
21	B	3	6
24	C	1	0
25	A	1	0
26	B	3	3
29	B	3	0

Anahtar Kelimeler: dispatcher,process,queue

GELİŞTİRİLEN YAZILIM

ProcessType sınıfı, projede gerçekleştirilmesi istenen proses'lerin öncelik tiplerinin enum olarak tutulduğu sınıftır. REALTIME(Öncelik=0), USERBASED1(Öncelik=1), USERBASED2(Öncelik=2) ve USERBASED3(Öncelik=3) olmak üzere dört farklı proses tipi tanımlanmıştır.

ProcessSituation sınıfı, proseslerin durumlarının enum olarak tutulduğu sınıftır. READY, RUNNING, INTERRUPTED, TERMINATED olmak üzere dört farklı proses durumu tanımlanmıştır.

Queue sınıfı, oluşturulacak proseslerin atılacağı kuyruk yapısını oluşturmak için kullanılan sınıftır. Kuyruğun ilk ve son elemanlarını, kapasitesini ve proses sayısını tutan değişkenlere sahiptir. Aynı zamanda getir(int index), getSize(), ekle(Process nesne), genişlet(int boyut), degistir(int index, Process nesne) ve cikar() adlarında 6 adet fonksiyona sahiptir. Getir fonksiyonu, kuyruğun içindeki elemanlar içerisinden parametre olarak aldığı index değerine sahip değeri döndüren fonksiyondur. getSize fonksiyonu, kuyruğun mevcut eleman sayısını geri döndüren fonksiyondur. ekle fonksiyonu , kendisine parametre olarak verilen proses nesnesini, kuyruk boşsa ekleyen, değilse kuyruğu genişletip sonra ekleyen fonksiyondur. genislet fonksiyonu, kendisine parametre olarak verilen değer kadar kuyruğun kapasitesini arttıran fonksiyondur. degistir fonksiyonu, kendisine parametre olarak verilen index değeri ve proses tipindeki nesneyi kullanarak, mevcut durumdaki kuyruğun index değerinde bulunan proses'i , parametre olarak verilen proses ile değiştiren fonksiyondur. cikar fonksiyonu, kuyruğun başındaki proses'i kuyruktan çıkaran fonksiyondur.

Process sınıfı, process özelliklerini (benzersiz id, varrış zamanı, öncelik numarası, toplam çalışma süresi, durumu vs.) içeren sınıftır. İçerisinde bulunan fonksiyonlar aşağıdaki gibidir:

- **public void ExecuteProcess(Process p, String message) :**Proses çalıştığında ve Windows işletim sisteminde koonsol ekranına prosese dair bilgiyi yazdırır.

- **private static void readProcess(Process p,java.lang.Process proces :** Proses çalıştırıldığında gerçekleştirilen komutları okur ve bilgi ekrana yazdırır.

Program sınıfı, main fomksiyonunu içeren class'tır. Başlangıçta txt üzerinden okunacak process'lerin ilk etapta atılacağı genek kuyruk yapısını tanımlar. Txt dosyasından okuduğu process'leri kuyruk yapısına ekler. Process'lerin öncelik durumuna göre processleri kuyruklara ayrıştırıp dispacther'a bağlar.

GeneralFunctions sınıfı, yapılacak genel işlemler için oluşturulmuş sınıftır. İçerisinde yer alan fonksiyonlar aşağıdaki gibidir:

- **Queue ReadFile(string FilePath):** Txt dosyasından okunan processleri genel kuyruğa atar. Her bir satır için bilgiler parse edilir ve process sınıfının içi doldurulur. Her bir satır virgüllere uygun olarak parçalanır. Her bir satır içerisindeki ilk elde edilen değer varış zamanı, ikinci elde edilen değer öncelik, üçüncü elde edilen değer ise işlemcinin bir processi işlemek için i/o wait gelene kadar ihtiyacı olan zaman aralığı olarak kabul edilir.

- **bubbleSortWithArrivalTime (Queue queue):** Kuyruğun içerisindeki processlerin zamanlarına göre sıralar.

- **bubbleSortWithPriority (Queue queue):** Optimizasyon alanıdır. Varış zamanları aynı olup önceliğe göre sıralama yapılarak işlem süresi kısaltılır.

Interlayer sınıfı, txt üzerinden okunup genel kuyruğa atılan processlerin önceliklerine göre ayrı kuyruklara ayrıştırıldığı classtır. Her bir kuyruk sort algoritmalarına tabi tutularak kuyruk içindeki proseslerin, varış zamanına ve önceliklerine göre kuyruğu teşkil etmeleri sağllanır. İşlem sırasında dispatcher başlatılır ve kuyruklar parametre olarak getirilir.

InterruptedProcess sınıfı, parçalanmış processlere ait özelliklerin tutulduğu sınıftır.

ProcessDispatcher sınıfı içerisinde yer alan fonksiyonlar aşağıdaki gibidir:

- **private void run():** Çalışma işlemi gerçekleştirilir. Dispatcher bağlanır. Süre ekrana basılır. Askıya alınmış kuyruklar dinlenir ve 20sn aşımına uğrayıp uğramadıkları kontrol edilir. 20 saniyeyi aşan processler terminated durumuna çekilir. Askıda olması sebebiyle terminated durumuna çekilmiş fakat kuyruktan silinmemiş processler varsa silinir. Kuyruk dönülmeden önce optimizasyon için kuyruğun boş olup olmadığı kontrolleri yapılır. Gerçek zamanlı kuyuğun elemanları dönülür. Bir kuyruk boşalmadan diğer prosesler başlamamalıdır bu sebeple ilk olarak bu kuyruk dönülür. O saniye içerisinde varan bir proses varsa içerisi çalışmaz. Varan proses ready durumuna çekilerek çalışmaya hazır hale getirilir. O saniye içerisinde çalışmakta olan bir gerçek zamanlı proses olup olmadığı (!HaveRealTimeRunningProcess()), o saniye içerisinde çalışıp sonlanmış bir proses(dolayısıyla durumu running değil) olup olmadığı (tempTimer!=timer) kontrolleri yapılır.Eğer bu kontroller sonrası aksi durum çıkmazsa prosesin çalışmasına izin verilir. Bu işlem sırasında öncelik sırasına önem verilir.

Görevlendiricinin görevi, bütün kuyrukların boşalması ile sonlanır.Görevlendiricinin işelmlerinin bitmesi ile zamanlayıcı durdurulur ve sisteme bilgisi yazdırılır.

- **public static void RunDispatcher (Queue realTimeProcesses, Queue userBasedProcesses1, Queue userBasedProcesses2, Queue userBasedProcesses3):** Zamanlayıcı başlatılır. Kuruğa atamalar yapılır. Zamanlayıcı ayarlanır.

- **ProcessChangeSituationOfProcess (int index, ProcessSituation situation, ProcessType type):** Process durumunu düzenleyen fonksiyondur. Prcoesse ait dört durumun incelemesini yapar. Processin yeni durumunu ekrana basar.

- **private boolean HaveRealTimeRunningProcess(int index):** Parametre olarak index numarası verilen prosesin o anda çalışamkta olup olmadığını kontrol eden fonksiyondur.

- **private boolean HaveRealTimeRunningProcess():** Çalışmakta olan bir gerçek zamanlı proses olup olmadığını kontrol eden fonksiyondur.

- **private void PrintNewSituation(Process p):** Processin yeni durum bilgisini, idsini, kalan süresini string değişkenine atar ve ExecureProcess fonksiyonunu çalıştırır.

- **private boolean InterruptControl(String pid):** Kesme kontrolü sağlar.

ÇIKTILAR

Proses listesi olarak kullanılan txt. dosyası aşağıdaki komutlarla komut satırında parametre olarak programa verilerek proje çalıştırılmıştır.

```
Administrator: C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.19045.2364]
(c) Microsoft Corporation. All rights reserved.

C:\Users\x\Downloads\Proje2023\Proje2023\OS23>java -jar test.jar test.txt
```

Bu şekilde çalıştırılan programımızın çıktısı şu şekildedir:

```
Administrator: C:\Windows\System32\cmd.exe
Saniye  = 22
      proses su an yurutuluyor (id:7    oncelik: 3  kalan sure: 4 sn)
      prosess askiya alindi - (id:7    oncelik: 3  kalan sure: 3 sn)
Saniye  = 23
      proses su an yurutuluyor (id:8    oncelik: 3  kalan sure: 1 sn)
      proses sonlandirildi.(id:8    oncelik: 3  kalan sure: 0 sn)
Saniye  = 24
      proses su an yurutuluyor (id:3    oncelik: 3  kalan sure: 4 sn)
      prosess askiya alindi - (id:3    oncelik: 3  kalan sure: 3 sn)
Saniye  = 25
      proses su an yurutuluyor (id:4    oncelik: 3  kalan sure: 2 sn)
      prosess askiya alindi - (id:4    oncelik: 3  kalan sure: 1 sn)
Saniye  = 26
      proses su an yurutuluyor (id:0    oncelik: 3  kalan sure: 2 sn)
      prosess askiya alindi - (id:0    oncelik: 3  kalan sure: 1 sn)
Saniye  = 27
      proses su an yurutuluyor (id:2    oncelik: 3  kalan sure: 1 sn)
      proses sonlandirildi.(id:2    oncelik: 3  kalan sure: 0 sn)
Saniye  = 28
      proses su an yurutuluyor (id:6    oncelik: 3  kalan sure: 2 sn)
      prosess askiya alindi - (id:6    oncelik: 3  kalan sure: 1 sn)
Saniye  = 29
      proses su an yurutuluyor (id:7    oncelik: 3  kalan sure: 3 sn)
      prosess askiya alindi - (id:7    oncelik: 3  kalan sure: 2 sn)
Saniye  = 30
      proses su an yurutuluyor (id:3    oncelik: 3  kalan sure: 3 sn)
      prosess askiya alindi - (id:3    oncelik: 3  kalan sure: 2 sn)
Saniye  = 31
      proses su an yurutuluyor (id:4    oncelik: 3  kalan sure: 1 sn)
      proses sonlandirildi.(id:4    oncelik: 3  kalan sure: 0 sn)
Saniye  = 32
      proses su an yurutuluyor (id:0    oncelik: 3  kalan sure: 1 sn)
      proses sonlandirildi.(id:0    oncelik: 3  kalan sure: 0 sn)
Saniye  = 33
      proses su an yurutuluyor (id:6    oncelik: 3  kalan sure: 1 sn)
      proses sonlandirildi.(id:6    oncelik: 3  kalan sure: 0 sn)
Saniye  = 34
      proses su an yurutuluyor (id:7    oncelik: 3  kalan sure: 2 sn)
      prosess askiya alindi - (id:7    oncelik: 3  kalan sure: 1 sn)
Saniye  = 35
      proses su an yurutuluyor (id:3    oncelik: 3  kalan sure: 2 sn)
      prosess askiya alindi - (id:3    oncelik: 3  kalan sure: 1 sn)
Saniye  = 36
      proses su an yurutuluyor (id:7    oncelik: 3  kalan sure: 1 sn)
      proses sonlandirildi.(id:7    oncelik: 3  kalan sure: 0 sn)
Saniye  = 37
      proses su an yurutuluyor (id:3    oncelik: 3  kalan sure: 1 sn)
      proses sonlandirildi.(id:3    oncelik: 3  kalan sure: 0 sn)
Görevlendirici Sona Erdi.
```

Projede istenildiği gibi her benzersiz prosesin rastgele oluşturulmuş bir renk şeması kullanarak

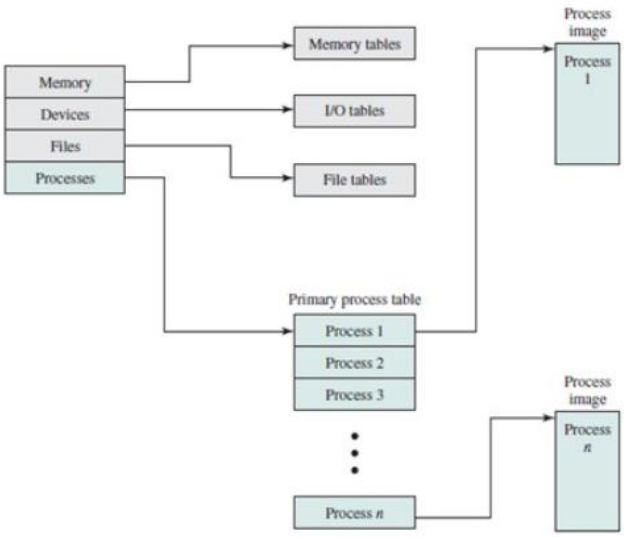
yazdırılması için aşağıda verilen komut cmd üzerinden en az bir kere çalıştırılmalıdır.

```
Microsoft Windows [Version 10.0.19045.2364]
(c) Microsoft Corporation. All rights reserved.

C:\Users\x\Downloads\Proje2023\Proje2023\OS23>reg add HKEY_CURRENT_USER\Console /v VirtualTerminalLevel /t REG_DWORD /d 0x00000001 /f
```

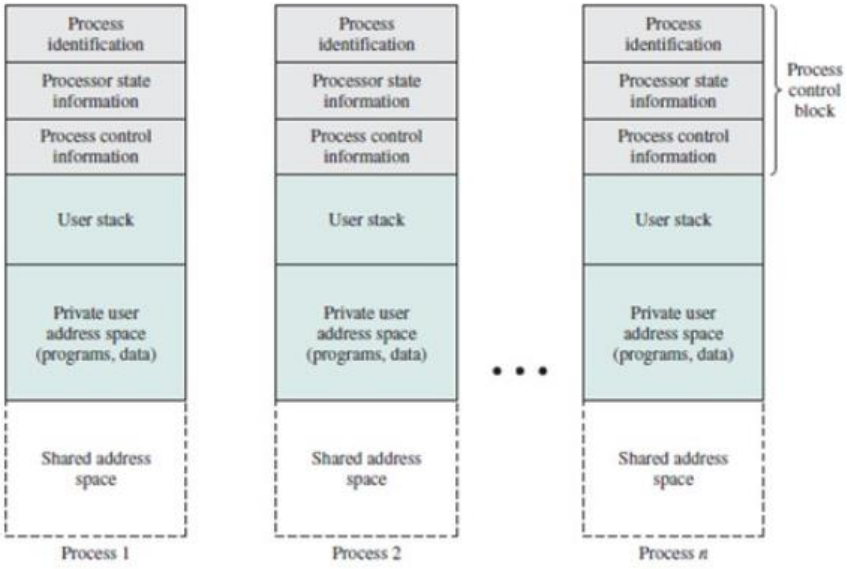
GÖREV YÖNETİMİNİN AMAÇLARI

Makul cevaplama süresi içerisinde, birkaç görevi aynı anda yürüterek işlemciden maksimum faydalanmaı sağlamak, görevleri kaynaklara dağıtmak, görevler arası haberleşmeyi sağlamaktır.



İşletim sistemi kontrol şemasına göre;

Memory Tables kısmında bellek tablolarımızı tahsisini yapıldı(RAM kullanıldı), oluşturulan proses kuyrukları bu bölgede saklandı, I/O tables kapsamında .txt uzantılı dosyada giriş kısmı ve cmd terminal ekranında da çıkış değerleri gösterildi. File Tables kısmında çoklu dosya hiyerarşisi olmadığı için projede kullanmayı tercih etmedik, Proces Tables; burada kuyruktaki işlemler için porses işlemleri gerçekleştirildi.



EKSİKLİKLERİ

CLI yerine GUI kullanılması kullanıcı açısından daha Pratik ve görsel açıdan kolaylık sağlayacaktır.

Txt yerine excel kullanılması USERBASED öncelik sıralamasındaki datalarımızın parçalanıp işlenmesi açısından daha optimum bir çözüm olabileceği görüşündeyiz.

SONUÇ

Projemiz kapsamında dört seviyeli, öncelikli, proses görevlendiricisine sahip bir çoklu programlama sistemi tasarlanmıştır.

Proje içerisinde diğer proseslerden daha yüksek önceliğe sahip olan gerçek zamanlı prosesler, İlk Gelen İlk Çalışır (FCFS) algoritmasını baz alarak çalıştırılmaktadır. Bu prosesler tamamlanıncaya kadar çalışmaya devam ederler.

Normal kullanıcı prosesleri, üç seviyeli bir geri beslemeli görevlendiricide çalıştırılır.

Bu anlatılanlar neticesinde gerçekleştirilmesi istenilen proseslerin öncelik tiplerinin tutulduğu ProcessType sınıfını baz alarak REALTIME, USERBASED1, USERBASED2, USERBASED3 (Buradaki rakamlar önceliği belli etmektedir.) proses tiplerimizi kuyruğumuza atarken, aynı zamanda ProcessSituation sınıfı ile proseslerimizin durumunu belirtmiş olduk.

Gerçek zamanlı kuyruktaki prosesler tamamlanmadan kullanıcı tanımlı proseslerin çalıştırıldığı görevlendirici çalışmasına devam edemez. Gerçek zamanlı prosesler tamamlandığında normal kullanıcı prosesleri öncelik değerlerine göre geri beslemeli kuyruk tarafından işlenir ve önceliği düşürülerek bir alt kuyruğa yerleştirilir.

Eğer ki tüm prosesler işlenip hepsi en alt seviyeli kuyruğa indirilmişse o zaman kuyruk üzerinde Round Robin algoritmasını baz alan bir işlem gerçekleştirilir.

Özetlemek gerekirse; Proses listesi, her zaman adımında sürekli işlenir ve gelen prosesler uygun kuyruğa aktarılır. Kuyruklar işleme alınır; tüm gerçek zamanlı prosesler tamamlanmak üzere çalıştırılır ve o anda çalışmakta olan diğer düşük öncelikli prosesler kesilir.

Sonuç olarak program istenilen özellikleri eksiksiz bir şekilde sağlamaktadır.

KAYNAKÇA

1.https://bilgisayarkavramlari.com/2008/11/19/round-robin/#:~:text=Bir%20zamanlama%20(scheduling)%20algoritmas%C4%B1d%C4%B1r.,quadrant)%20i%C5%9Flemciyi%20terk%20etmek%20zorundad%C4%B1r

2. https://slideplayer.biz.tr/slide/8500145/

3. https://yazilimdnyasi.wordpress.com/2020/01/26/fcfs-algoritmasi/

4. https://www.koseburak.net/blog/first-come-first-serve/

5. https://yazilimdnyasi.wordpress.com/2020/02/01/round-robin-algoritmasi/