KONYA
TEKNİK ÜNİVERSİTESİ

ÜNİVERSİTE	**KONYA TEKNİK ÜNİVERSİTESİ**
FAKÜLTE	**MÜHENDİSLİK VE DOĞA BİLİMLERİ FAKÜLTESİ**
BÖLÜM	**YAZILIM MÜHENDİSLİĞİ**
DERS -KONU	**İŞLETİM SİSTEMLERİ – VİZE ÖDEVİ**
OKUL NO	**211229005**
AD	**ABDULMELİK**
SOYAD	**GÖYMEN**
Github Link	**https://github.com/gymgymm/thread**

	Yazılım Mühendisliği İşletim Sistemleri Dersi Vize Ödevi	Ad-Soyad: Abdulmelik Göymen No: 211229050

Soru 1

1'den 1.000.000 (1 milyon)'a kadar olan sayılardan oluşan bir ArrayList oluşturunuz. Ardından, bu ArrayList'teki 1milyon elemanı 250000 eleman olacak şekilde 4 eşit parçaya ayırınız. Bu 4 ayrı 250000 elemanlık ArrayList'ler içinde asal sayıları bulan 4 ayrı Thread tasarlayınız.

a) 4 Thread bulduğu çift sayıları ortak bir ArrayList'e ekleyecektir.
b) 4 Thread bulduğu tek sayıları ortak bir ArrayList'e ekleyecektir.
c) 4 Thread bulduğu asal sayıları ortak bir ArrayList'e ekleyecektir.
d) ArrayList'ler ilk oluşturulduklarında boş olacaklardır.

Soru 2

Amaç:

Bu ödevde, işletim sistemi üzerindeki sanal bellek yönetimini anlamak, sayfa değiştirme algoritmalarını incelemek, bellek hiyerarşisini kavramak ve bir uygulama üzerinden sanal bellek yönetimi problemlerini analiz etmek amaçlanmaktadır.

Görevler:

1- Sanal Bellek Kavramları:

Sanal bellek nedir, nasıl çalışır, işletim sistemi açısından avantajları ve dezavantajları nelerdir? Bu konuda kapsamlı bir literatür taraması yaparak kavramları açıklayınız.

2- Sayfa Değiştirme Algoritmaları:

FIFO, Optimal, LRU gibi sayfa değiştirme algoritmalarını araştırın. Her bir algoritmanın avantajlarını vedezavantajlarını belirleyerek karşılaştırıcı bir analiz yapınız.

3- Sanal Bellek Hiyerarşisi:

Bellek hiyerarşisinin nasıl işlediğini ve ana bellek (RAM), ikincil bellek (disk), sayfa tablosu gibi kavramların birbirleriyle nasıl etkileşimde bulunduğunu açıklayınız.

4- Sayfa Tablosu Analizi:

Sayfa tablosu nedir, nasıl çalışır? Sayfa tablosu oluşturulurken kullanılan teknikleri inceleyip avantaj ve dezavantajlarını belirtiniz.

5- Bellek Fragmentasyonu:

Bellek fragmentasyonu kavramını açıklayınız. İç ve dış fragmentasyon nedir? Sanal bellek yönetiminin bellek fragmentasyonuyla başa çıkma yöntemlerini araştırınız.

6- Sanal Bellek Yönetimi Uygulama:

İstediğiniz bir işletim sistemi üzerinde belirli bir uygulamanın sanal bellek yönetimini inceleyiniz. Uygulama sırasında ortaya çıkabilecek durumları analiz ediniz.

Raporlama: Yukarıdaki görevleri içeren kapsamlı bir rapor hazırlayınız. Raporunuz, literatür taraması, kavramsal açıklamalar, analiz sonuçları ve öğrenimlerinizi içermelidir. Raporunuzu 12 punto Times New Romanyazı stili ve 1.5 satır aralığıyla hazırlayınız.

2022 – 2023 Akademik Yılı Yazılım Mühendisliği İşletim Sistemleri Vize Ödevi

Soru 1: 1'den 1.000.000 (1 milyon)'a kadar olan sayılardan oluşan bir ArrayList oluşturunuz. Ardından, bu ArrayList'teki 1milyon elemanı 25000 eleman olacak şekilde 4 eşit parçaya ayırınız. Bu 4 ayrı 25000 elemanlık ArrayList'ler içinde asal sayıları bulan 4 ayrı Thread tasarlayınız.

a) 4 Thread bulduğu çift sayıları ortak bir ArrayList'e ekleyecektir.

b) 4 Thread bulduğu tek sayıları ortak bir ArrayList'e ekleyecektir.

c) 4 Thread bulduğu asal sayıları ortak bir ArrayList'e ekleyecektir.

d) ArrayList'ler ilk oluşturulduklarında boş olacaklardır.

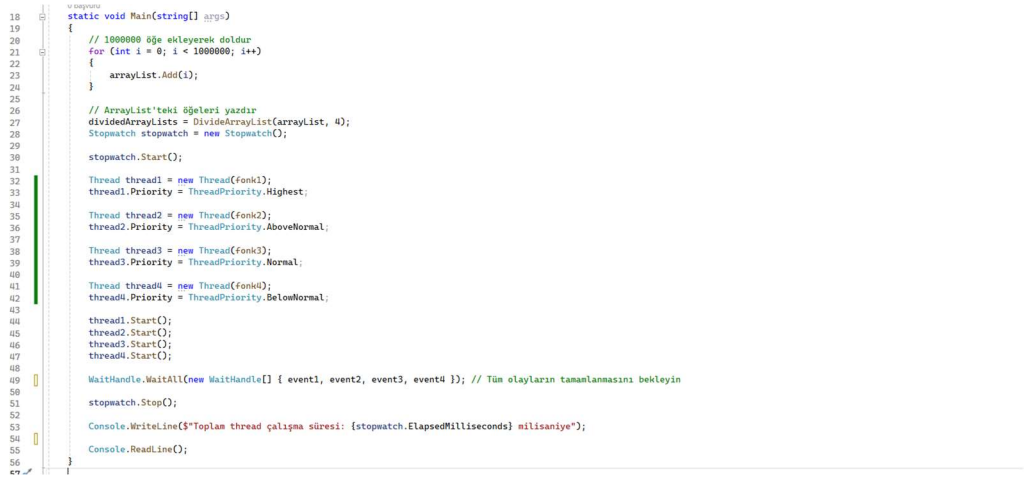
```
1   using System;
2   using System.Collections;
3   using System.Diagnostics;
4   using System.Threading;
5
    0 başvuru
6   class Program
7   {
8       public static ArrayList arrayList = new ArrayList();
9       public static ArrayList tekarray = new ArrayList();
10      public static ArrayList çiftarray = new ArrayList();
11      public static ArrayList asalarray = new ArrayList();
12      public static ArrayList[] dividedArrayLists;
13      static ManualResetEvent event1 = new ManualResetEvent(false);
14      static ManualResetEvent event2 = new ManualResetEvent(false);
15      static ManualResetEvent event3 = new ManualResetEvent(false);
16      static ManualResetEvent event4 = new ManualResetEvent(false);
```

1. Aşama: Kodun 1. ve 5. Satırları arasında sistemi için gerekli kütüphaneler projeye dahil edilmiştir. Bizim projemizin ehemmiyeti açısından System.Threading kütüphanesi dikkat çekmektedir.

2. Aşama: Kodun 6. ve 17. Satırları arasında bulunan **ArrayList** tanımları ve elzem olan static ManualResetEvent even1 = new ManualResetEvent(false); tanımlaması bulunmaktadır. Peki bu tanımlama(lar) nedir:

- ManualResetEvent C# dilinde, System.Threading isim alanı altında yer alan bir senkronizasyon primitifidir. Bu nesne, birden fazla thread arasında sinyalizasyon ve bekleme mekanizmalarını sağlamak için kullanılır. ManualResetEvent özellikle, bir thread'in diğer thread'lerin belirli bir noktaya

ulaşmasını veya bir işlemi tamamlamasını beklemesi gereken durumlarda kullanışlıdır.



```
18  static void Main(string[] args)
19  {
20      // 1000000 öğe ekleyerek doldur
21      for (int i = 0; i < 1000000; i++)
22      {
23          arrayList.Add(i);
24      }
25
26      // ArrayList'teki öğeleri yazdır
27      dividedArrayLists = DivideArrayList(arrayList, 4);
28      Stopwatch stopwatch = new Stopwatch();
29
30      stopwatch.Start();
31
32      Thread thread1 = new Thread(fonk1);
33      thread1.Priority = ThreadPriority.Highest;
34
35      Thread thread2 = new Thread(fonk2);
36      thread2.Priority = ThreadPriority.AboveNormal;
37
38      Thread thread3 = new Thread(fonk3);
39      thread3.Priority = ThreadPriority.Normal;
40
41      Thread thread4 = new Thread(fonk4);
42      thread4.Priority = ThreadPriority.BelowNormal;
43
44      thread1.Start();
45      thread2.Start();
46      thread3.Start();
47      thread4.Start();
48
49      WaitHandle.WaitAll(new WaitHandle[] { event1, event2, event3, event4 }); // Tüm olayların tamamlanmasını bekleyin
50
51      stopwatch.Stop();
52
53      Console.WriteLine($"Toplam thread çalışma süresi: {stopwatch.ElapsedMilliseconds} milisaniye");
54
55      Console.ReadLine();
56  }
```

4. Aşama: Kodun 18. satırından itibaren main kısmı başlamış bulunmaktadır.

- Kodun 20. satırında başlayıp 24. satırında biten `for` döngüdü ise listeye 0 dan başlayıp 999.999 a kadar 1.000.000 adet ardışık sayıyı arrayList listesinde eklemektedir.
- arrayList listesine eklenen bu sayılar 27. satırda bulunan dividedarrayLists = DivideArrayList(arrayList, 4); ifadesi ise bir arrayList' i belli bir sayıda (4) alt gruba bölünmesi işlemini gerçekleştirmektedir. Bölünmesi gerçekleştirilen liste dividedarrayLists değişkenine atılıyor.
- Kodun 28. ve 30. satırlarında ise süre ölçmek için gerekli tanımlamalar yapılıp süre sayacı başlatılmaktadır.
- Oluşturulan threadler ve atanan parametreler ışığında threadlere öncelik sırası atama işlemleri 32. ve 42. satırları arasında yapılmış ve işlemler şu şekilde gerçekleşmiştir:

```
Thread thread4 = new Thread(fonk4);
thread4.Priority = ThreadPriority.BelowNormal;
```

`BelowNormal` komutu threadın çalışmasındaki öncelik sırasını belirler.

Öncelik sırası ve yapıları şu şekildedir:

1. **BelowNormal:** Öncelik seviyesi, "Normal"den biraz daha düşüktür. "BelowNormal" öncelikli bir thread, "Normal" öncelikli thread'lere göre daha az işlemci zamanı alır. Bu, daha az kritik işlemleri yürüten thread'ler için uygun olabilir.
2. **Normal:** Varsayılan thread önceliği seviyesidir. Çoğu thread bu seviyede başlatılır. "Normal" öncelik, thread'in işlemci zamanına ortalama bir erişim sağlar. Diğer öncelik seviyeleri, bu "normal" seviyeyle karşılaştırılarak değerlendirilir.
3. **AboveNormal:** Öncelik seviyesi, "Normal"den biraz daha yüksektir. "AboveNormal" öncelikli bir thread, "Normal" öncelikli thread'lere göre daha fazla işlemci zamanı alır. Daha önemli işlemleri yürüten thread'ler için bu seviye tercih edilebilir.
4. **Highest:** En yüksek thread önceliği seviyesidir. "Highest" öncelikli bir thread, diğer tüm öncelik seviyelerindeki thread'lere göre daha fazla işlemci zamanı alır. Çok kritik işlemleri yürüten thread'ler için bu seviye uygun olabilir.

Öncelik ayarlamak, özellikle kaynak yoğun uygulamalarda veya çoklu iş parçacığı kullanılan durumlarda önemlidir. Ancak, önceliklerin dikkatli bir şekilde kullanılması gerekir, çünkü yüksek öncelikli iş parçacıklarının aşırı kullanımı diğer iş parçacıklarının açlığa (starvation) uğramasına neden olabilir. Kodumda ise thread1'den thread4'e kadar öncelik sırası yüksekten düşüğe gidecek şekilde ayarlanıp koda dökülmüş ve threadler `thread1.Start();` komutuyla başlatılmıştır.

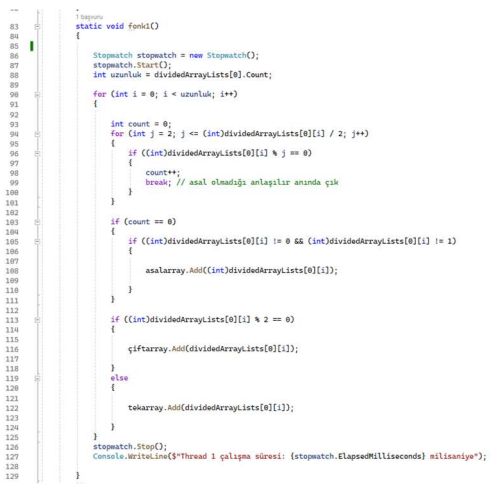
- 49. satırda ise `WaitHandle.WaitAll(new WaitHandle[] { event1, event2, event3, event4 });` komutu ile bütün işlemlerin bitmesi beklenmiştir.
- 51. Satırda da 28. satırda tanımlanıp 30. satırda başlatılan süre sayacı durdurulmuştur.
- 53. satırda console çıktıları verilmiştir.
- 55. satırda ise console uygulamasının bir tuşa basıncaya dek açık kalması için gerekli komut yazılmıştır.

KOD lock OLMAYAN ve lock OLAN ŞEKLİNDE 2' YE AYRILMAKTADIR

Lock Nedir: Çoklu-thread uygulamalarda, birden fazla thread'in aynı kaynağa (örneğin, bir veritabanına, bir dosyaya veya hafızadaki bir değişkene) eş zamanlı olarak erişmeye çalışması durumunda veri tutarsızlığı, yarış koşulları (race conditions) ve diğer senkronizasyon sorunları ortaya çıkabilir. "lock" ifadesi, bu tür problemleri önlemek için kullanılır.

Lock Nasıl Çalışır: C#'ta lock ifadesi, Monitor sınıfının bir kolay kullanım şeklidir. Lock bloğu, belirli bir nesne üzerinde kilit (lock) oluşturur. Bir thread kilitlediği zaman, diğer thread'ler bu nesnenin kilidini alana kadar beklemek zorunda kalır.

LOCK OLMAYAN KOD:

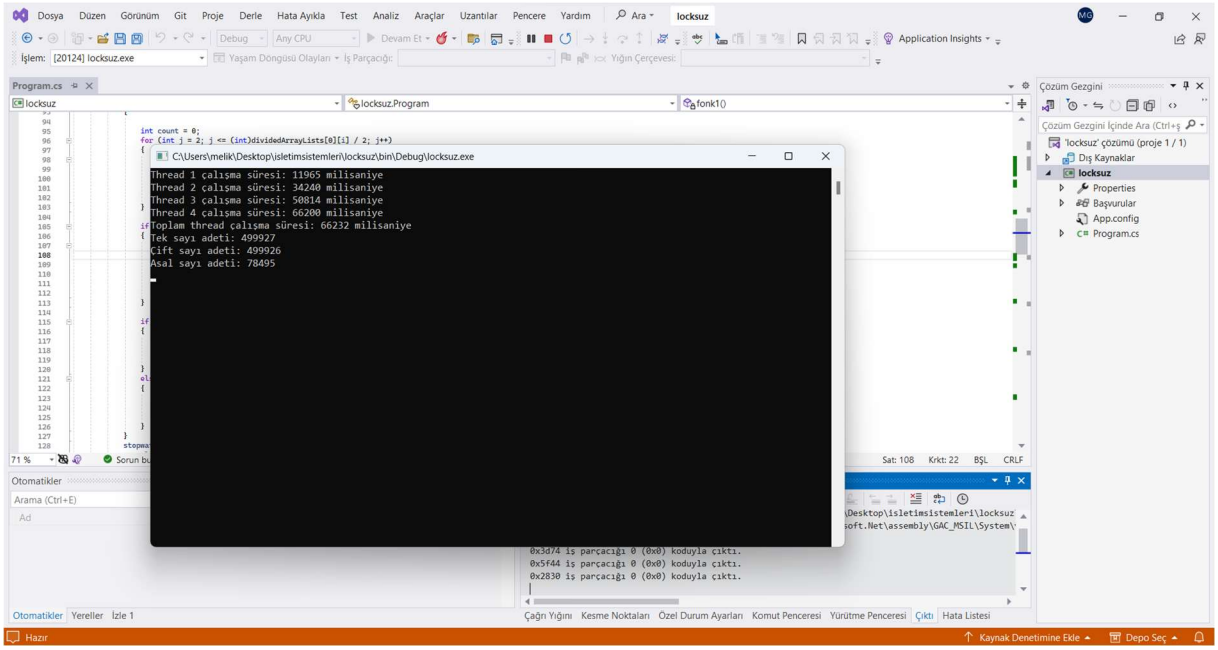


```
1 başvuru
static void fonk1()
{

    Stopwatch stopwatch = new Stopwatch();
    stopwatch.Start();
    int uzunluk = dividedArrayLists[0].Count;

    for (int i = 0; i < uzunluk; i++)
    {

        int count = 0;
        for (int j = 2; j <= (int)dividedArrayLists[0][i] / 2; j++)
        {
            if ((int)dividedArrayLists[0][i] % j == 0)
            {
                count++;
                break; // asal olmadığı anlaşılır anında çık
            }
        }

        if (count == 0)
        {
            if ((int)dividedArrayLists[0][i] != 0 && (int)dividedArrayLists[0][i] != 1)
            {

                asalarray.Add((int)dividedArrayLists[0][i]);

            }
        }

        if ((int)dividedArrayLists[0][i] % 2 == 0)
        {

            çiftarray.Add(dividedArrayLists[0][i]);

        }
        else
        {

            tekarray.Add(dividedArrayLists[0][i]);

        }
    }
    stopwatch.Stop();
    Console.WriteLine($"Thread 1 çalışma süresi: {stopwatch.ElapsedMilliseconds} milisaniye");
}
```

6.1. Aşama:

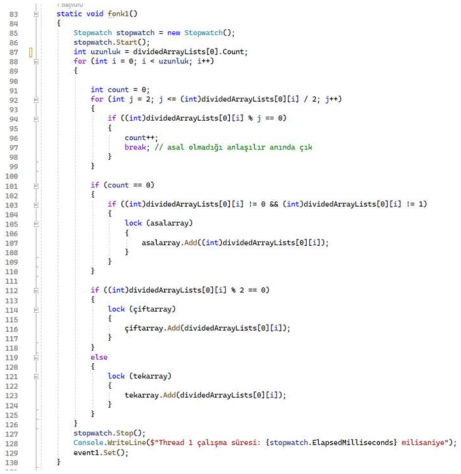
- Bu aşamada her thread için bir fonksiyon oluşturulmuştur. Yani her thread sadece 1 tane fonksiyon ve kendi fonksiyonunu çalıltırmaktadır.
- Devamı ise asal sayı bulma şeklinde devam etmektedir.
- Bulunan asal sayılar `asalarray` listesine, tek sayılar `tekarray` listesinde, çift sayılar ise `çiftarray` listesine eklenmektedir.
- 128. satrda ise ilgili thread için başlatılan süre sayacı durdurulup ölçüm yapılıyor.

İşlemler neticesinde örnek konsol çıktısı şekildeki gibidir:



➢ Geliştirilen thread ile asal sayı bulma kodunun console çıktılar yukarıdaki bulunan fotoğraftaki gibidir. Burada süre çıktıları yanı sıra toplam sayı adeti 1.000.000 ‘dan düşük olduğunu gözlemlemekteyiz. Kod çalışma esnasında bir noktada(larda) listelerde bulunan sayılarıda veri kaybı olmaktadır. Bu durumu bertaraf etmek için C# dilinde bulunan ve yukarıda bahsi geçen lock ifadesini kullancağız.

LOCK OLAN KOD: Burada ise diğer kod yapısı aynı sadece fonksiyonlarda bulunan `asalarray`, `çiftarray`, `tekarray` listelerine ekleme bölgelerine lock ifadesini atacağız ki işlemler yapıldığında veri kaybı olmasın.



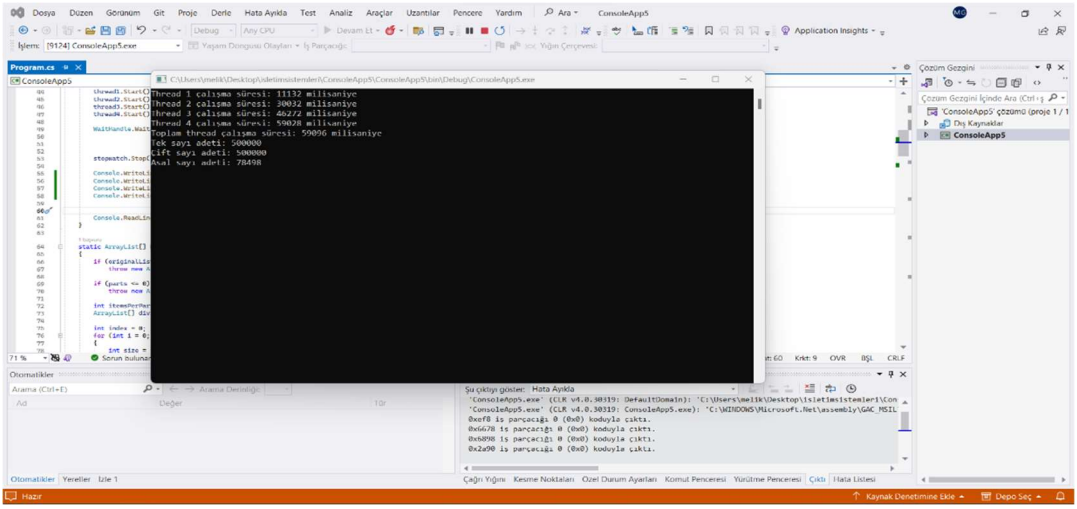
```
83      static void fonk1()
84      {
85          Stopwatch stopwatch = new Stopwatch();
86          stopwatch.Start();
87          int uzunluk = dividedArrayLists[0].Count;
88          for (int i = 0; i < uzunluk; i++)
89          {
90
91              int count = 0;
92              for (int j = 2; j <= (int)dividedArrayLists[0][i] / 2; j++)
93              {
94                  if ((int)dividedArrayLists[0][i] % j == 0)
95                  {
96                      count++;
97                      break; // asal olmadığı anlaşılır anında çık
98                  }
99              }
100
101             if (count == 0)
102             {
103                 if ((int)dividedArrayLists[0][i] != 0 && (int)dividedArrayLists[0][i] != 1)
104                 {
105                     lock (asalarray)
106                     {
107                         asalarray.Add((int)dividedArrayLists[0][i]);
108                     }
109                 }
110             }
111
112             if ((int)dividedArrayLists[0][i] % 2 == 0)
113             {
114                 lock (çiftarray)
115                 {
116                     çiftarray.Add(dividedArrayLists[0][i]);
117                 }
118             }
119             else
120             {
121                 lock (tekarray)
122                 {
123                     tekarray.Add(dividedArrayLists[0][i]);
124                 }
125             }
126         }
127         stopwatch.Stop();
128         Console.WriteLine($"Thread 1 çalışma süresi: {stopwatch.ElapsedMilliseconds} milisaniye");
129         event1.Set();
130     }
```

6.2. Aşama:

- Bu aşamada 85. satırda ilgili threadin süre sayacı başlatılıyor.
- Devamı ise asal sayı bulma şeklinde devam etmektedir.
- Bulunan asal sayılar `asalarray` listesine, tek sayılar `tekarray` listesinde, çift sayılar ise `çiftarray` listesine eklenmektedir.
- Burada eklenirken lock işlemi kullanılmaktadır yani ilgili listeyi sadece ilgili thread kullanabilsin diye.
- 127 ile 129. satırlar arasında ise ilgili thread için başlatılan süre sayacı durdurulup ölçüm yapılıyor ve `event1.Set();` komutu ile de threadin sinyal durumu güncellenmektedir.

İşlemler neticesinde örnek konsol çıktısı şekildeki gibidir:

- Lock ifadesini kullandığımızda çıktılar şekildekki gibidir. Peki neyi farketmeliyiz süre çıktıları yanı sıra locksuz koddaki veri kaybı burada yok lock fonksiyonunun önemi.



2022 – 2023 Akademik Yılı Yazılım Mühendisliği İşletim Sistemleri Vize Ödevi

Soru 2

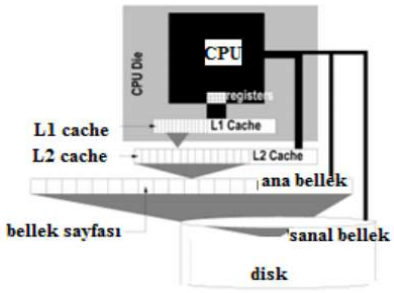
***Sorulan sorular farklı literatürlerde gerekli araştırmalar yapılıp ilgili soru hakkında birden fazla literatürde bulunan cevaplar verilmiştir.**

1 - Sanal Bellek Kavramları:

-Sanal bellek nedir, nasıl çalışır, işletim sistemi açısından avantajları ve dezavantajları nelerdir? Bu konuda kapsamlı bir literatür taraması yaparak kavramları açıklayınız.

Google Akademik ile Literatür Taraması:

Sanal Bellek Nedir: Ana bellek üzerinde sabit sayıda tampondan oluşan bölgeye tampon havuzu denilmektedir (Sacco ve Schkolnik, 1982) [1]. Tampon havuzunda bulunan her tampon üzerinde bir disk bloğu bilgisi tutulmaktadır. Ayrıca kısıtlı olan ana bellek bölgesinin yetersiz kaldığı durumlarda disk üzerinde bir bölge ayrılarak ana bellek gibi kullanılabilmektedir. Ayrılmış olan bu bölgeye de sanal bellek adı verilmektedir (Peter, 1970) [2].



Şekil 1. Bilgisayar üzerindeki donanımsal yapı (Stefan, 2002)

Şekil-1: [3]

Sanal Bellek Nasıl Çalışır: Sanal bellek, bilgisayarınızın sabit disk alanının bir bölümünü kullanarak RAM bellek alanını arttıran bir teknolojidir. Sanal bellek, bilgisayarınızın RAM bellek alanı dolu olduğunda, işletim sisteminin sabit diskinizdeki bir alanı geçici olarak RAM gibi kullanmasına izin verir.

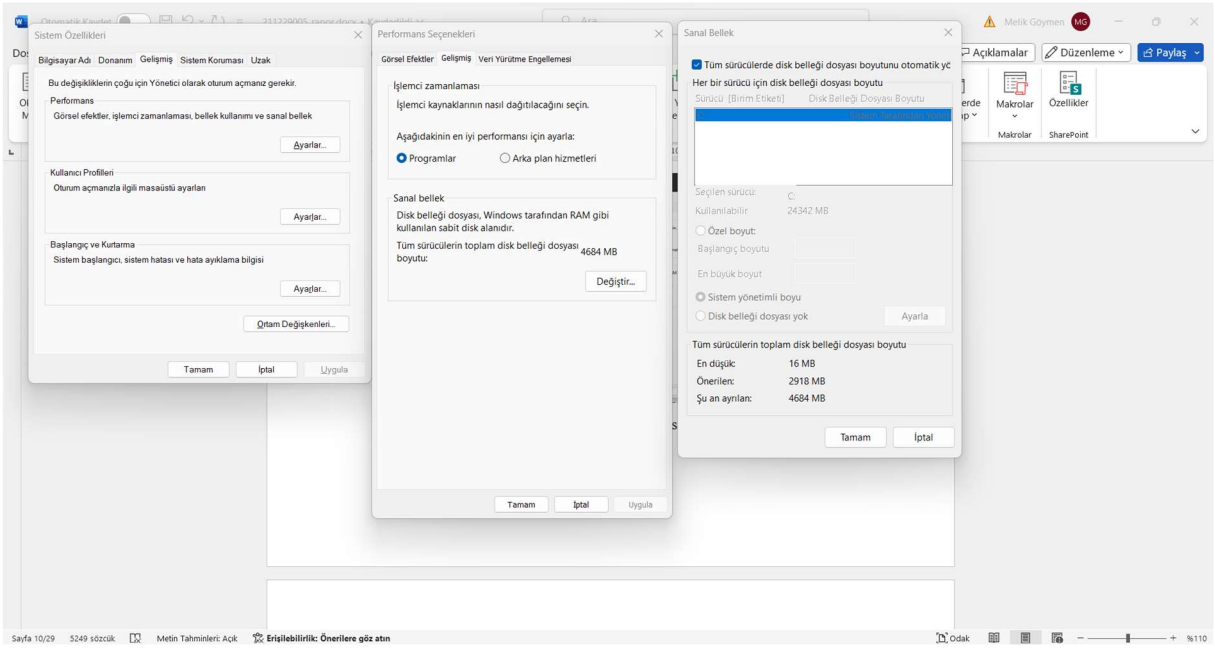
Sanal Bellek Avantajları: Sanallaştırma, bilişim kaynaklarının (işlemci, depolama, ağ, bellek, platform, sunucu, masaüstü, uygulama vb.) gerçekte var olan kaynağın değil; gerçek kaynağa dayandırılarak tanımlanmış olan soyut halinin, kullanıcılara sunulması olarak tanımlanabilir. Böylece gerçek kaynak, göreceli olarak daha az kapasiteli çok sayıda sanal kaynak olarak kullandırılabilmektedir [4]. Sunucuyu barındıracak ortamı veya donanımı sanallaştırmak olmak üzere iki tür temel sanallaştırma ortamı bulunmaktadır [5].

Sanal Bellek Dezavantajları: Sanal belleğin başlıca dezavantajları, performansla ilgilidir. Sanal bellek, fiziksel RAM yetersiz kaldığında devreye girer ve sabit diskin bir kısmını bellek olarak kullanır. Ancak, sabit diskler (özellikle HDD'ler), RAM'e göre çok daha yavaş erişim hızlarına sahiptirler. Bu durum, veri erişiminde önemli gecikmelere ve genel sistem performansında düşüşlere yol açabilir. Özellikle bellek yoğun uygulamalar kullanılırken, sistem hantallaşabilir ve tepki süreleri artabilir. Ayrıca, sabit diskin sürekli olarak sanal bellek olarak kullanılması, diskin ömrünü azaltabilir ve sistem stabilitesini olumsuz etkileyebilir. Kısacası, sanal bellek, acil bir çözüm olarak işlev görse de uzun vadede RAM kapasitesinin artırılmasına kıyasla daha az etkili bir yoldur.

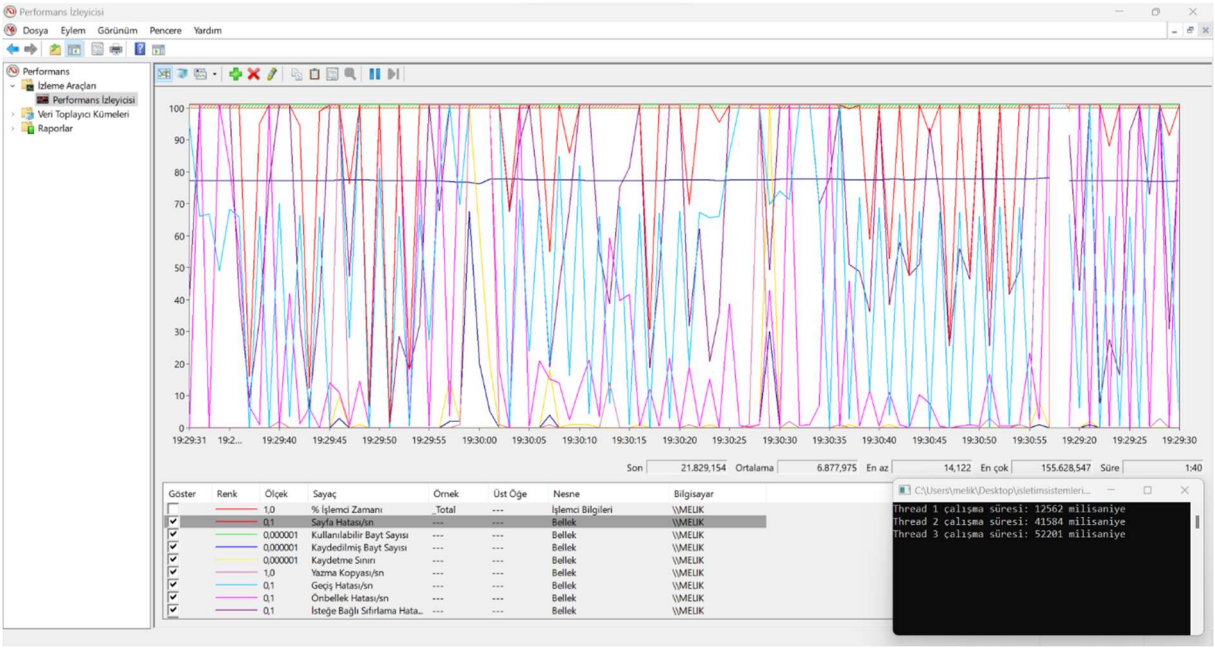
Kendi Edindiğim Öğrenimler: Windows işletim sistemi ile çalışan kişisel bilgisayarımda sanal bellek ayarları ile bazı değişiklikler yapmış bulunmaktayım örn:

Sanal Belleğin Ayarlarını Kontrol Etme:

Windows: Kontrol Paneli > Sistem > Gelişmiş sistem ayarları > Performans Ayarları > Gelişmiş > Sanal Bellek şeklinde ilgili pencereye yol aldığımda açılan sayfa:



Şekil 2: Kişisel bilgisayarda sanal bellek ayarlarının gösterimi.



Şekil 3: Thread kodu çalıştığında bellek ve diğer değişkenlerin değer grafiği.

Çıkarımlar:

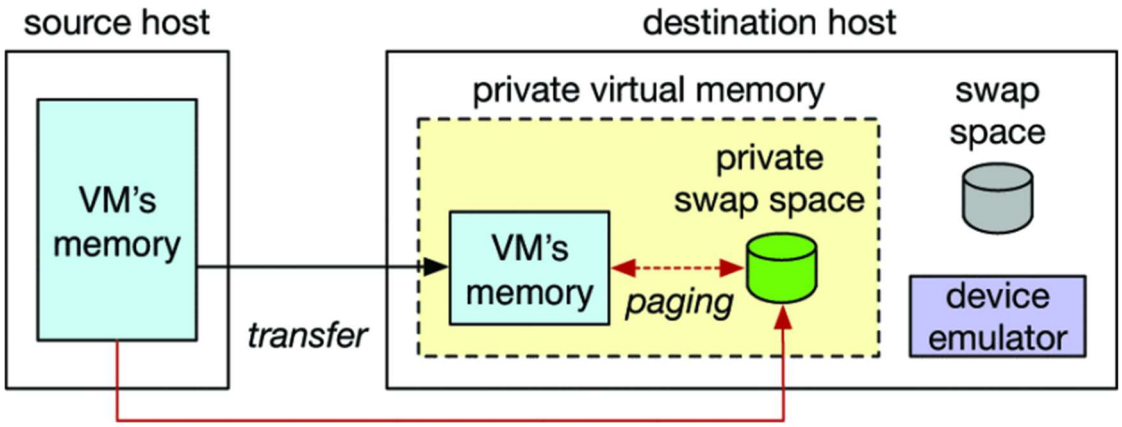
*Kişisel bilgisayarımda sanal bellek yönetimi otomatik olarak gerçekleşmektedir.

*Anlık olarak: en düşük 16MB, önerilen 2918MB, şu an ayrılan 4684MB şeklinde ölçüldüğünü ve saptandığını fark ettim.

*Bilgisayarımın koşturduğum sisteme göre bellek değerlerinş ve diğer bir çok performans değerlerini Performans İzleyicisi uygulamasından izleyebilirim.

IEEE (Institute of Electrical and Electronics Engineer) Xplore ile Literatür Taraması:

Sanal Bellek Nedir: Hedef ana makine yeterli belleğe sahip değilse, bellek-sanallaştıran VM göçü mümkündür. Bu, hedef ana makinede sanal belleği kullanır. VM'nin belleğinin bir kısmını ikincil depolamadaki takas alanında saklayabilir ve gerekli miktarda belleği kullanabilir. VM, takas alanındaki bellek verilerini gerektiğinde, bir sayfa içine alma işlemi gerçekleştirilir ve veriler fiziksel belleğe taşınır. Bunun karşılığında, fiziksel bellekteki muhtemelen erişilmeyecek veriler takas alanına taşınır. Ancak, geleneksel sanal belleğin VM göçü için uygun olmadığı belirtilmiştir [6].



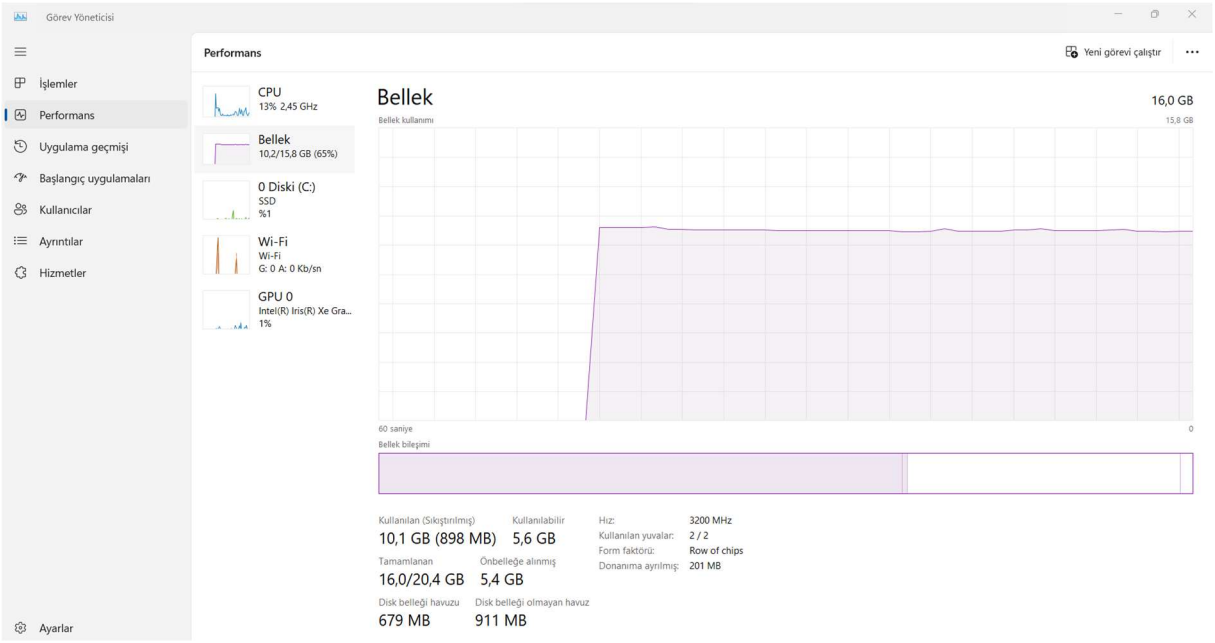
Şekil 4: Özel sanal bellek kullanarak belleği sanallaştıran VM geçişi.

Sanal Bellek Dezavantajları: Son zamanlarda, Altyapı olarak Hizmet (IaaS) bulutları, büyük miktarda belleğe sahip sanal makineler (VM'ler) sağlamaktadır. Örneğin, Amazon EC2 tarafından sağlanan VM'lerin belleği 24 TB'a kadar çıkabilmektedir. Bir VM çalıştıran bir ana makine bakım altına alındığında, VM'in başka bir ana makineye önceden taşınarak çalışmasının devam ettirilmesi mümkündür. VM göçü, bir VM'in durumunu, örneğin sanal CPU'ları, belleği ve sanal cihazları transfer eder ve VM'i hedef ana makinede yeniden başlatır. Bu nedenle, VM göçü, VM'in tüm belleğini barındırabilecek yeterli belleğe sahip büyük bir ana makine gerektirir. Ancak, ara sıra olan VM göçlerinin hedefi olarak bu tür büyük ana makineleri her zaman korumak

maliyet açısından verimli değildir. Özellikle, özel bulutlar, yeterli sayıda büyük ana makine hazırlayamayabilir [7].

Sanal Bellek Avantajları: Bu sorunları gidermek için bölünmüş göç [1] önerilmiştir. Bu yöntem, bir VM'nin belleğini böler ve bellek parçalarını daha küçük ana makineler arasında transfer eder. Muhtemelen erişilecek bellek verilerini ana ana makineye ve geri kalan bellek verilerini yardımcı ana makineler arasında aktarır. Eğer VM, göç sonrasında yardımcı ana makinelerdeki bellek verilerine ihtiyaç duyarsa, bu veriler uzaktan sayfalama yoluyla ana ana makineye aktarılır. VM göçü sırasında herhangi bir sayfalama olmadığı için, bölünmüş göç göç performansını artırabilir. Ayrıca, gerekli bellek verileri önceden ana ana makineye aktarıldığı için göç eden VM'nin çalışma performansını artırabilir [7].

Çıkarımlar:



Şekil 5: Kişisel bilgisayar bellek performansı.

*Kişisel bilgisayarımın bellek yönetimini Görev Yöneticisi sekmesinden izleyip çıkarımlarımı yaptım.

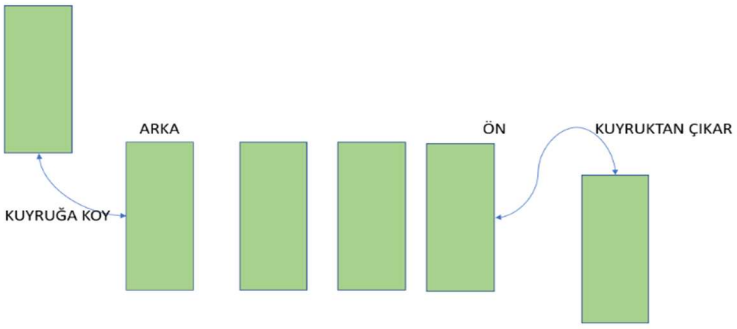
2- Sayfa Değiştirme Algoritmaları:

-FIFO, Optimal, LRU gibi sayfa değiştirme algoritmalarını araştırın. Her bir algoritmanın avantajlarını ve dezavantajlarını belirleyerek karşılaştırıcı bir analiz yapınız.

ArXiv, Google Akademi ve Academia.edu ile Literatür Taraması:

FIFO Sayfa Değiştirme Algoritması (FIFO Page Replacement Algorithm (İlk giren ilk çıkar sayfa değiştirme algoritması)): Sayfalama algoritmalarının düzgünlüğünü inceliyoruz. İstek sırasının bozulması nedeniyle sayfa hatalarının sayısı ne kadar artabilir? Sayfa hatalarındaki maksimum artış, istek dizisindeki değişikliklerin sayısıyla orantılıysa, sayfalama algoritmasına düzgün adını veriyoruz. Ayrıca bir algoritmanın düzgünlüğünü ölçen niceliksel düzgünlük kavramlarını da tanıtıyoruz. Deterministik ve rastgele talep çağrıları ve rekabetçi algoritmaların düzgünlüğüne ilişkin alt ve üst sınırları türetiyoruz. Son derece rekabetçi deterministik algoritmalar arasında LRU alt sınırla eşleşirken FIFO üst sınırla eşleşir. Partition, Equitable veya Mark gibi iyi bilinen rastgeleleştirilmiş algoritmaların düzgün olmadığı gösterilmiştir. Smoothed-LRU ve LRU-Random adı verilen iki yeni rastgeleleştirilmiş algoritma sunuyoruz. Düzleştirilmiş-LRU, ödünleşimin bir parametre tarafından kontrol edildiği akıcılık için rekabet gücünden fedakârlık edilmesine olanak sağlar. LRU-Random, daha sorunsuz olmasına rağmen en az herhangi bir deterministik algoritma kadar rekabetçidir [8]. Sayfalama, küçük ve hızlı bir bellek olan önbellek (cache) ve büyük ancak yavaş bir bellek olan ana bellekten oluşan iki seviyeli bir bellek sistemini modellemektedir. Bir programın çalışması sırasında, veriler sayfalar olarak bilinen veri birimleri halinde önbellek ile bellek arasında transfer edilir. Önbelleğin boyutu, sayfa olarak k ifade edilir ve belleğin boyutu sonsuz kabul edilebilir. Sayfalama problemine giriş, önbellekte bulunması gereken bir dizi sayfa talebidir. Bir sayfa talebi geldiğinde ve bu sayfa zaten önbellekteyse, herhangi bir işlem gerekmez. Bu durum "isabet" olarak adlandırılır. Aksi takdirde, sayfa bellekten önbelleğe getirilmelidir, bu da önbellekten başka bir sayfanın çıkarılmasını gerektirebilir. Bu, "sayfa hatası" veya "kaçırma" olarak bilinir. Sayfalama algoritması, hata sayısını en aza indirmek için önbellekte hangi sayfaların tutulacağına karar vermelidir. Bir sayfalama algoritması,

yalnızca tam dolu bir önbellekte hata olduğunda bir sayfayı önbellekten çıkaran bir sayfa hatası durumunda talep sayfalama olarak adlandırılır. Talep dışı herhangi bir sayfalama algoritması, performansı feda etmeden talep sayfalama yapılabilir duruma getirilebilir [9]. Bu algoritmaya göre bir sayfa ihlali (page fault) olduğunda, yani hafızada (RAM) bulunmayan bir sayfaya erişilmek istendiğinde, yani diskteki bir sayfaya erişilmek istendiğinde, Diskten ilgili sayfa hafızaya (RAM) yüklenirken, hafızadaki en eski sayfa yerine yüklenir ve bu en eski sayfa da diske geri yazılır [11].



Şekil 6: FIFO Sayfa değiştirme algoritması anlatım şekli.

FIFO (First-In, First-Out) Avantajları:

1. Basitlik: FIFO, uygulanması ve anlaşılması kolay bir algoritmadır.

2. Adil Kullanım: Tüm sayfalar eşit süre boyunca bellekte tutulur, bu da adil bir kullanım sağlar.

3. Öngörülebilir Davranış: Sayfaların bellekten ne zaman çıkarılacağı öngörülebilir.

FIFO (First-In, First-Out) Dezavantajları:

1. Sık Kullanılan Sayfaların Atılması: Sık kullanılan veya önemli sayfalar, bellekte yeterince uzun süre kalmadan atılabilir.

2. Performans Sorunları: FIFO, sayfa hata oranını azaltmada etkili değildir, bu da performans sorunlarına yol açabilir.

3. **Kötü Hafıza Kullanımı:** Bellekteki verimli kullanımı sağlamada zayıf olabilir, özellikle değişen veya yoğun erişim desenlerinde.

LRU Sayfa Değiştirme Algoritması (Least Recently Used (LRU) Page replacement (En nadir kullanılan sayfa değiştirme algoritması)): Bu makale, LRU-K yöntemi olarak adlandırılan, veritabanı disk arabelleğe alma işlemine yeni bir yaklaşım sunmaktadır. *LRU-K'nin temel fikri, popüler veritabanı sayfalarına yapılan son K* referanslarının zamanlarını takip etmek ve bu bilgiyi kullanarak referansların varışlar arası zamanlarını sayfa sayfa istatistiksel olarak tahmin etmektir. LRU-K yaklaşımı, nispeten standart varsayımlar altında optimal istatistiksel çıkarımı gerçekleştirmesine rağmen oldukça basittir ve çok az muhasebe yükü gerektirir. Simülasyon deneyleriyle gösterdiğimiz gibi, LRU-K algoritması, sık ve seyrek başvurulan sayfalar arasında ayrım yapma konusunda geleneksel ara belleğe alma algoritmalarını geride bırakıyor. Aslında LRU-K, bilinen erişim frekanslarına sahip sayfa kümelerinin özel olarak ayarlanmış boyutlardaki farklı arabellek havuzlarına manuel olarak atandığı ara belleğe alma algoritmalarının davranışına yaklaşabilir. Bununla birlikte, bu tür özelleştirilmiş tamponlama algoritmalarının aksine, LRU-K yöntemi kendi kendini ayarlar ve iş yükü özelliklerine ilişkin harici ipuçlarına dayanmaz. Ayrıca LRU-K algoritması, değişen erişim modellerine gerçek zamanlı olarak uyum sağlar [10].

Veritabanı sistemleri, diskten okunduktan ve belirli bir uygulama tarafından erişildikten sonra, disk sayfalarını bir süre bellek tamponlarında tutar. Bunun amacı, popüler sayfaları bellekte tutarak disk giriş/çıkışını azaltmaktır. Gray ve Putzolu'nun "Beş Dakika Kuralı"nda şu ticaret-off dile getirilmiştir: Bellek tamponları için daha fazla ödemeye, sistem için disk kollarının maliyetini azaltmak adına belirli bir noktaya kadar razıyızdır. Kritik tamponlama kararı, diskten okunmak üzere yeni bir tampon yuvasına ihtiyaç duyulduğunda ve tüm mevcut tamponlar kullanımda olduğunda ortaya çıkar: Hangi mevcut sayfa tampondan çıkarılmalıdır? Bu, sayfa değiştirme politikası olarak bilinir ve farklı tamponlama algoritmaları, uyguladıkları değiştirme politikasının türüne göre adlandırılır. Çoğu ticari sistem tarafından kullanılan algoritma, En Az Son Kullanılan (LRU) olarak bilinir. Yeni bir tampona ihtiyaç duyulduğunda, LRU politikası en uzun süredir erişilmemiş sayfayı tampondan çıkarır. LRU tamponlaması, başlangıçta talimat mantığındaki kullanım kalıpları için geliştirilmiş olup, veritabanı ortamına her zaman iyi uyum sağlamaz. Gerçekten de LRU tamponlama algoritmasının, son referans

zamanına dayanarak hangi sayfanın tampondan çıkarılacağına karar verme konusunda yetersiz bilgiye sahip olması gibi bir problemi vardır. Özellikle, LRU, nispeten sık referanslara sahip sayfalar ile çok nadir referanslara sahip sayfalar arasındaki farkı, sistem nadiren referans verilen sayfaları uzun bir süre için tamponda tuttuğu noktaya kadar ayırt edemez [10]. Bu algoritmaya göre bir sayfa ihlali (page fault) olduğunda, yani hafızada (RAM) bulunmayan bir sayfaya erişilmek istendiğinde, yani diskteki bir sayfaya erişilmek istendiğinde, Diskten ilgili sayfa hafızaya (RAM) yüklenirken, hafızadaki en az erişilen sayfa yerine yüklenir ve bu en az kullanılan sayfa da diske geri yazılır [11].

LRU için

String	3	4	5	1	2	3	4	5	3	4	5	5
F1	3	3	3	1	1	1	4	4	4	4	4	4
F2		4	4	4	2	2	2	5	5	5	5	5
F3			5	5	5	3	3	3	3	3	3	3
Hata	VAR	VAR	VAR	VAR	VAR	VAR	VAR	VAR	YOK	YOK	YOK	YOK

* Sayfa hatası değeri: **8** dir. Sayfa hatası oranı ise; 8 / 12 = 0,66 dır.

Şekil 7: LRU sayfa değiştirme algoritması anlatım şekli.

LRU (Least Recently Used) Avantajları:

1. Verimlilik: LRU, sık kullanılan sayfaları bellekte tutarak verimliliği artırır.

2. Azaltılmış Sayfa Hataları: Daha az sayfa hatası oluşur, çünkü sık erişilen sayfalar genellikle bellekte kalır.

3. Adaptif Davranış: Çalışma zamanında erişim desenlerine adapte olabilme yeteneği sağlar.

LRU (Least Recently Used) Dezavantajları:

1. Yönetim Maliyeti: LRU'nun takip etmesi gereken ek bellek ve işlem maliyetleri vardır.

2. Uygulama Karmaşıklığı: Algoritmanın uygulanması, basit algoritmalara göre daha karmaşıktır.

3. Zayıf Yaşlanma Politikası: Uzun süre bellekte kalan, ancak artık sık kullanılmayan sayfaların gereksiz yere tutulması problemi olabilir.

Optimal Sayfa Değiştirme Algoritması (Optimal Replacement (Mükemmel Sayfa Değiştirme Algoritması)): Bu algoritma, hiçbir zaman gerçekleştirilemeyecek hayali bir algoritmadır. Akademik olarak ortaya atılmıştır ve algoritmanın çalışması için daha sonra gelecek olan sayfa ihtiyaçlarının önceden bilinmesi gerekir. Bu sayfa değiştirme algoritmasına göre bir sayfa ihlali (page fault) olduğunda, yani hafızada (RAM) bulunmayan bir sayfaya erişilmek istendiğinde, yani diskteki bir sayfaya erişilmek istendiğinde, Diskten ilgili sayfa hafızaya (RAM) yüklenirken, hafızada bundan sonra

en uzun süre erişilmeyecek olan yerine yüklenir ve bu en az kullanılan sayfa da diske geri yazılır [11].

Şekil 8: Optimal sayfa değiştirme algoritması anlatım şekli.

Optimal Avantajları:

1. En Az Sayfa Hatası: Optimal algoritma, teorik olarak mümkün olan en az sayfa hatasını sağlar.

2. İdeal Performans Referansı: Diğer algoritmaların performansını değerlendirmek için bir referans noktası sağlar.

3. Etkin Bellek Kullanımı: Bellek kaynaklarını en verimli şekilde kullanır.

Optimal Dezavantajları:

1. Gerçek Zamanlı Uygulanabilirlik: Gelecekteki sayfa isteklerini önceden bilmek gerektiği için pratikte uygulanamaz.

2. Tahmin Edilemezlik: Gelecekteki erişimleri tahmin etmek zor olduğundan, gerçek dünya senaryolarında kullanılamaz.

3. Uygulama Karmaşıklığı: Teorik olarak ideal olsa da gerçek uygulamalar için aşırı karmaşıktır.

FIFO (First-In, First-Out), LRU (Least Recently Used) ve Optimal sayfa değiştirme algoritmalarını kendi aralarında Performans ve Verimlilik, Uygulanabilir ve Karmaşıklık, Uygunluk perspektiflerinden incelediğimde elde ettiğim sonuçlar ve kazanımlar şunlardır:

1. Performans ve Verimlilik: Optimal algoritma teorik olarak en iyi performansı sunmaktadır, çünkü gelecekteki sayfa isteklerini biliyor ve en az sayfa hatası üreterek bu işlemi gerçekleştiriyor. LRU, gerçek zamanlı senaryolarda daha iyi performans göstermektedir çünkü en son kullanılan sayfaları bellekte tutarak bu işlemleri gerçekleştirir. FIFO ise daha düşük performans göstermektedir, çünkü sadece giriş sırasına göre işlem yapmkatadır.

2. Uygulanabilirlik ve Karmaşıklık: FIFO en basit uygulamaya yapısına sahiptir, ancak bu basitlik, etkin bellek kullanımı ve düşük sayfa hata oranı açısından dezavantajları ortaya çıkartmaktadır. LRU, FIFO'ya göre daha karmaşık bir uygulamaya sahip olduğu halde, daha iyi bir performans sunar. Bunun nedeni ise; LRU'nun sayfaların kullanım sıklığını ve zamanını dikkate almasıdır. FIFO yalnızca sayfaların belleğe giriş sırasına göre işlem yaparken, LRU en son ne zaman kullanıldıklarına bakarak sayfaları yönetmektedir. Bu yaklaşım, sık kullanılan ve önemli sayfaların bellekte kalmasını sağlayarak sayfa hatalarını azaltır ve bellek erişimini optimize etmesini sağlamaktadır. Optimal algoritma ise pratikte uygulanabilir değildir, çünkü gelecekteki sayfa isteklerini bilmek gereklidir.

3. Uygunluk: FIFO, öngörülebilir ve adil bir bellek yönetimi sunar, ancak sık kullanılan sayfaları atabilir. FIFO algoritması, sayfaları belleğe giriş sırasına göre yönetmektedir. Bu, ilk giren sayfanın ilk çıkarılacağı anlamına gelir. Bu yöntem öngörülebilir ve adildir çünkü her sayfa aynı süre boyunca bellekte tutulmaktadır. Ancak, bu yaklaşım, bir sayfanın ne kadar sık kullanıldığını veya ne kadar önemli olduğunu dikkate almaz. Bu nedenle, FIFO, sık kullanılan veya kritik öneme sahip sayfaları, bellekte daha az süre

kalmış olsalar bile atabilir. Bu, özellikle yoğun bellek erişim desenlerinde verimsizliklere yol açabilir. LRU, bellek yönetimini kullanım sıklığına göre optimize eder yani sayfaların bellekte ne kadar süre tutulacağını belirlemek için son kullanım zamanlarını kullanarak optimize işlemlerini gerçekleştirir. Optimal ise teorik olarak en etkin bellek yönetimini sunar, ancak bu, gerçek dünyada uygulanabilir değildir. Bunun sebebi ise, Optimal sayfa değiştirme algoritması, gelecekteki sayfa isteklerini önceden bilebilmeyi gerektirir. Gerçek dünyada bu, mümkün değildir çünkü gelecekte hangi sayfaların isteneceğini önceden tahmin etmek pratikte uygulanabilir bir yöntem değildir. Bu nedenle, Optimal algoritma daha çok teorik bir model olarak kullanılır ve gerçek zamanlı sistemlerde uygulanabilir bir yöntem değildir. Bu algoritma, ideal koşullarda mümkün olan en düşük sayfa hata oranını sağlamak için tasarlanmış bir kavramsal modeldir.

Kullanım Alanları Örnekleri:

FIFO: Az değişkenlik gösteren veri erişim desenleri olan uygulamalarda kullanışlıdır.

1. Basit Web Siteleri: Statik içerikli web siteleri, ziyaretçi trafiği az ve sabit olduğunda FIFO kullanımı için uygun olabilir.

2. Arşivleme Sistemleri: Düzenli aralıklarla erişilen, ancak sıklıkla güncellenmeyen arşiv dosyalarını yöneten sistemler.

3. Belirli Endüstriyel Kontrol Sistemleri: Sabit ve tahmin edilebilir veri toplama ve işleme desenlerine sahip sistemler.

LRU: Veri erişim desenlerinin sık değiştiği, örneğin web sunucuları veya çok kullanıcılı veri tabanı sistemleri gibi ortamlarda tercih edilir.

1. Çok Kullanıcılı Veri Tabanı Sistemleri: Sürekli olarak farklı veri tabanı sorguları alan ve çeşitli kullanıcı taleplerine cevap veren sistemler.

2. Web Sunucuları: Farklı içeriklere sürekli erişim sağlanan ve sıkça güncellenen web sitelerini barındıran sunucular.

3. Uygulama Sunucuları: Çok sayıda kullanıcı tarafından sürekli erişilen ve dinamik içerik sunan uygulama sunucuları.

Optimal: Teorik olarak en iyi performansı sunar, ancak gelecekteki erişimleri önceden bilmek gerektiğinden gerçek uygulamalarda kullanılamaz. Araştırma ve teorik analizlerde, diğer algoritmaların performansını değerlendirmek için ideal bir referanstır.

1. Akademik ve Araştırma Çalışmaları: Bilgisayar bilimi ve veri tabanı sistemleri üzerine araştırma yapılırken, diğer algoritmaların performansını karşılaştırmak için bir referans noktası olarak kullanılır.

2. Algoritma Simülasyonları: Bilgisayar mimarisi veya işletim sistemleri derslerinde, öğrencilere sayfa değiştirme stratejilerinin etkinliğini öğretmek için simülasyonlar kullanılır.

3. Performans Testleri: Yeni geliştirilen sayfa değiştirme algoritmalarının teorik performans sınırlarını belirlemek için kullanılır.

3 – Sanal Bellek Hiyerarşisi:

-Bellek hiyerarşisinin nasıl işlediğini ve ana bellek (RAM), ikincil bellek (disk), sayfa tablosu gibi kavramların birbirleriyle nasıl etkileşimde bulunduğunu açıklayınız.

Öncelikle ilgili soruda geçen kavramlar ve bilgisayar donanımları nedir:

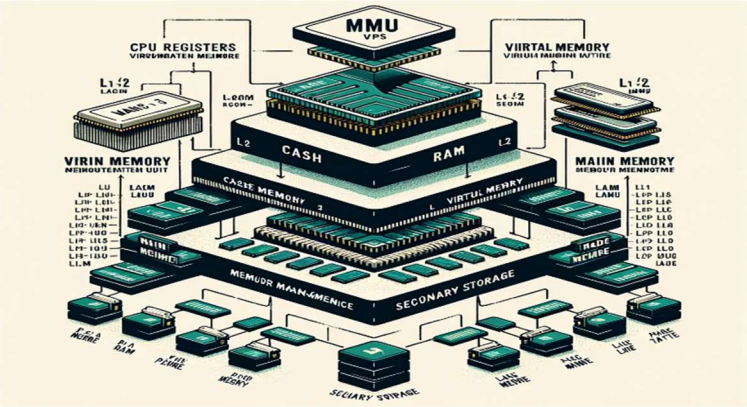
RAM (Random Acces Memory) Rastegele Erişimli Bellek: RAM bilgilerin geçici olarak depolandığı bir hafıza türüdür. Bilgisayarlar genellikle o an üzerinde çalıştıkları programlar ve işlemlerle ilgili bilgileri RAM denen bu hafıza parçasında tutarlar. RAM ve sabit sürücü temel olarak aynı bilgileri saklarlar, ancak işlemcinin RAM'deki bilgilere erişme ve onları işleme hızı, sabit sürücüdeki bilgilere erişme ve onları işleme hızından çok daha büyüktür.

İkincil Bellek (Disk): Bilgisayar depolama hiyerarşisinde ikincil depolama sistemi olarak işlev gören, verileri kalıcı olarak depolamak için tümleşik devre tertibatlarını kullanan bir veri depolama aygıtıdır [12].

Sayfa Tablosu: Sayfa tablosu, sanal bellek yönetiminde kritik bir rol oynayan bir veri yapısıdır. İşletim sistemi tarafından kullanılır ve sanal adresleri fiziksel adreslere çevirmek için kullanılır. Bir program çalıştığında, kod ve veriler sanal adreslerle ifade edilir. Sayfa tablosu, bu sanal adreslerin hangi fiziksel RAM adreslerine karşılık geldiğini belirler. Böylece, programların daha büyük ve sürekli bir bellek alanında çalışıyormuş gibi hissetmelerini sağlar. Sayfa tablosu, RAM'de olmayan verilerin diskten yüklenmesi gerektiğinde (sayfa hatası) de devreye girer.

Donanımlar ve kavramlar arasındaki ilişki ve çalışma prensibi nedir: Sanal bellek hiyerarşisi, farklı bellek türlerinin ve hızlarının bir araya gelmesiyle oluşan bir sistemdir. Bu hiyerarşi, işlemcilerin hızı ve işlem gücü arttıkça daha da önem kazanır. Bellek sınırlı bir kaynak olduğu için, daha fazla uygulamanın kullanılmasıyla performans düşebilir veya durma noktasına gelebilir. İşte burada sanal bellek devreye girer ve sistemdeki RAM'i taklit ederek sabit sürücünün bir bölümünü kullanır. Sanal bellek, sabit disk sürücüsünü geçici depolama alanı gibi kullanır ve böylece yazılımın ek bellek olarak kullanılmasına olanak tanır. Sanal bellek hiyerarşisinde farklı bellek türleri vardır. Bu bellek türleri, işlemciye yakınlık ve hız, düşük gecikme ve maliyet bakımından yukarıdan aşağıya doğru sıralanabilir. Örneğin, CPU içindeki yazmaçlar, L1 ve L2 önbellekler, ana bellek (DRAM), yerel ve uzaktan erişimli ikincil depolama birimleri hiyerarşinin farklı seviyelerini oluşturur. Her bir bellek türü, bir üst seviyedeki bellek türünü önbellek olarak kullanır. Örneğin, yerel diskte kayıtlı bir dosya işlenirken ana bellek, işletim sistemi tarafından geçici depolama birimi olarak kullanılabilir. Bu hiyerarşinin en üst kısmında, hız açısından işlemciye en yakın olan bellek birimleri yer alır. L1 önbellek doğrudan işlemcinin içinde bulunurken, L2 önbellek genellikle işlemci kartına yerleştirilir ve bu nedenle erişim hızı L1'e göre biraz daha düşük, kapasitesi ise daha yüksektir. Hiyerarşinin üst kısımlarındaki bellek birimleri genellikle daha hızlı fakat küçük kapasitelidir, alt kısımlarındakiler ise daha büyük kapasiteli fakat hızları düşüktür. İşlemci bir veriyi ana bellekten okumak veya yazmak istediğinde, önce ön belleğe bakar. Eğer istenen veri varsa ve içeriği değiştirilmemişse bu veriyi okur. Yazma

sürecinde ise, önce ön bellekteki, sonra ise uygun bir zaman bulunduğunda ana bellekteki bir adresin içerikleri değiştirilir. Bu şekilde, bellek erişim hızı ve kapasitelerine göre farklılık gösteren bir hiyerarşi oluşturulmuş olur.



Şekil 9: Sanal bellek hiyerarşisini anlatan şekil.

4 – Sayfa Tablosu Analizi:

-Sayfa tablosu nedir, nasıl çalışır? Sayfa tablosu oluşturulurken kullanılan teknikleri inceleyip avantaj ve dezavantajlarını belirtiniz.

Sayfa Tablosu Nedir:

Sayfa tablosu, sanal bellek kullanan bilgisayar sistemlerindeki bir veri yapısıdır. İşletim sistemi, sanal adresleri fiziksel adreslere dönüştürmek için sayfa tablosunu kullanır. Bu, özellikle sanal adres alanının fiziksel bellekten daha büyük olduğu durumlarda önemlidir.

Sayfa Tablosu Nasıl Çalışır:

1. Adres Dönüşümü: Sayfa tablosu, sanal bellek adreslerini (bir program tarafından kullanılan adresler) fiziksel bellek adreslerine (RAM'daki gerçek adresler) dönüştürür.

2. Sayfa Tablosu Girdileri: Her tablo girdisi, sanal sayfa numarasını (VPN) ve eşleşen fiziksel sayfa numarasını (PPN) içerir. Ayrıca, erişim hakları, değiştirilme durumu ve sayfanın bellekte olup olmadığı gibi bilgileri de içerebilir.

3. Arama İşlemi: Bir program belirli bir sanal adresi talep ettiğinde, işletim sistemi bu adresi sayfa tablosunda arar ve eşleşen fiziksel adresi bulur. Eğer

sayfa bellekte yoksa, sayfa hatası (page fault) oluşur ve ilgili sayfa diske kaydedilir veya diskten yüklenir.

Sayfa Tablosu Oluşturma Teknikleri:

1. Basit Sayfa Tablosu: Tek bir global tablo tüm sanal sayfaları fiziksel sayfalara eşler. Bu yöntem basit ve anlaşılırdır, ancak büyük bellek alanları için verimsiz olabilir.

2. Çok Düzeyli Sayfa Tablosu: Büyük adres uzaylarını yönetmek için birden fazla sayfa tablosu kullanılır. Bu, sayfa tablolarının boyutunu azaltır ve bellek kullanımını optimize eder.

3. Tersine Çevrilmiş Sayfa Tablosu: Her fiziksel sayfa için yalnızca bir girdi içerir. Bu, sayfa tablosunun boyutunu fiziksel bellek boyutuna orantılar, büyük sanal adres uzayları için daha verimlidir.

4. Hashed Sayfa Tablosu: Büyük sanal adres uzaylarını daha verimli bir şekilde yönetmek için hash tablolarını kullanır.

Avantajlar ve Dezavantajlar

1. Basit Sayfa Tablosu:

Avantaj: Yapılandırması ve anlaşılması kolay.

Dezavantaj: Büyük sanal adres uzayları için verimsiz ve bellek tüketimi yüksek.

2. Çok Düzeyli Sayfa Tablosu:

Avantaj: Büyük adres uzaylarını daha etkili yönetir, bellek kullanımını azaltır.

Dezavantaj: Adres dönüşümü daha karmaşık hale gelir.

3. Tersine Çevrilmiş Sayfa Tablosu:

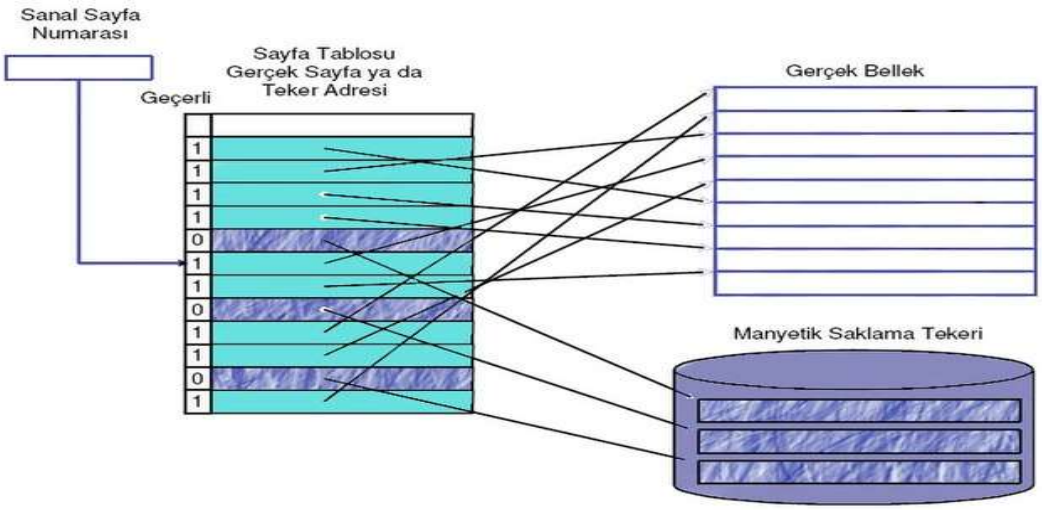
Avantaj: Bellek tüketimi düşük, büyük bellek alanlarını etkili bir şekilde yönetir.

Dezavantaj: Adres dönüşüm süreci daha karmaşık ve zaman alıcı olabilir.

4. Hashed Sayfa Tablosu:

Avantaj: Büyük sanal adres uzaylarını etkin şekilde yönetir, hızlı arama yapısı sağlar.

Dezavantaj: Hash çakışmaları yönetimi gerektirir ve karmaşık olabilir.



Şekil 10: Sayfa tablosu analizi örnek gösterim şekli.

Sayfa tablolarının seçimi, bellek büyüklüğü, adres uzayının boyutu ve sistem performans gereksinimlerine bağlı olarak değişir. Her teknik, belirli senaryolara ve sistem mimarilerine göre avantajlar ve dezavantajlar sunar.

5 – Bellek Fragmantasyonu:

-Bellek fragmantasyonu kavramını açıklayınız. İç ve dış fragmantasyon nedir? Sanal bellek yönetiminin bellek fragmantasyonuyla başa çıkma yöntemlerini araştırınız.

Bellek Fragmantasyonu Nedir:

Bellek fragmantasyonu, bir bilgisayar sisteminin belleğinin kullanımı sırasında meydana gelen ve belleğin etkin kullanımını engelleyen bir sorundur. Bu, belleğin parçalı ve kullanılamaz hale gelmesine yol açar.

İç ve Dış Fragmantasyon:

1. İç Fragmantasyon (Internal Fragmentation):

1. Bellek, bir sürecin ihtiyacından daha büyük bloklara bölündüğünde meydana gelir.

2. Her blokta, atanan belleğin kullanılmayan bir kısmı vardır.

3. Örneğin, bir süreç 5KB'lık bir bellek talep eder ancak en küçük kullanılabilir blok 10KB'dır. Geriye kalan 5KB kullanılmaz ve iç fragmantasyona yol açar.

2. Dış Fragmantasyon (External Fragmentation):

1. Toplamda yeterli boş bellek olsa bile, süreçlerin ihtiyaçları için yeterli ardışık boş alanın olmaması durumudur.

2. Zamanla, sürekli olarak belleğin ayrılması ve serbest bırakılması, küçük, kullanılamaz boşlukların oluşmasına neden olur.

3. Örnek: Bellekte 10KB boş yer var, ancak bu, 2KB'lık beş ayrı parçada mevcuttur. 6KB'lık bir sürecin talebi karşılanamaz.

Sanal Bellek Yönetiminin Bellek Fragmantasyonuyla Başa Çıkma Yöntemleri:

1. Sayfalama (Pagination):

1. Bellek, sabit boyutlu bloklar (sayfalar) halinde bölünür.

2. Her süreç kendi sanal adres alanına sahiptir ve bu alan fiziksel belleğe sayfa sayfa eşlenir.

3. Sayfalama, dış fragmantasyonu ortadan kaldırır çünkü her sayfa, belleğin herhangi bir yerinde yer alabilir.

2. Segmentasyon (Segmentation):

1. Bellek, değişken boyutlu segmentlere bölünür.

2. Her segment, bir sürecin mantıksal birimini (örneğin kod, veri, yığın) temsil eder.

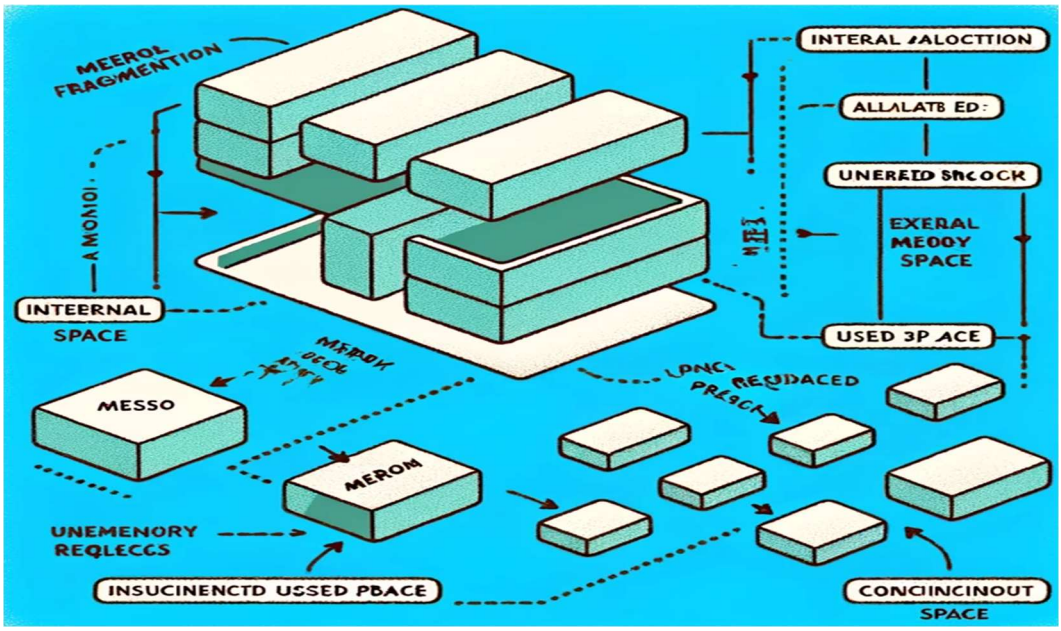
3. Segmentasyon, iç fragmantasyonu azaltmaya yardımcı olur, çünkü her segment sadece gereken kadar bellek alır.

3. Sanal Bellek Kullanımı (Virtual Memory Usage):

1. Fiziksel bellekten daha büyük bir adres alanı sağlar.

2. Belleğin daha verimli kullanılmasına izin verir ve süreçlerin bellek ihtiyaçlarını daha iyi karşılar.

3. Aktif olmayan sayfalar disk gibi ikincil depolama alanına taşınarak, bellekte yer açılır ve fragmantasyon azaltılır**.**



Şekil 11: Bellek fragmantasyonunu anltan bir görsel.

Bellek fragmantasyonu, belleğin etkin kullanımını engelleyen bir sorun olsa da modern işletim sistemlerinin bellek yönetimi teknikleri bu sorunu büyük ölçüde azaltmaktadır.

6 – Sanal Bellek Yönetimi Uygulama:

- İstediğiniz bir işletim sistemi üzerinde belirli bir uygulamanın sanal bellek yönetimini inceleyiniz. Uygulama sırasında ortaya çıkabilecek durumları analiz ediniz.

Windows İşletim Sistemi Üzerinde Sanal Bellek Yönetimi:

Windows, sanal bellek yönetimini, her uygulamanın kendi adres alanına sahip olduğu ve bu alanın fiziksel belleğe sayfa sayfa eşlendiği bir sayfalama mekanizması üzerinden gerçekleştirir.

1. Sanal Bellek Yönetimi Süreci:

1. Sayfa Dosyası Oluşturma: Windows, sabit disk üzerinde bir sayfa dosyası (pagefile.sys) oluşturur. Bu dosya, fiziksel bellekten (RAM) taşan veriler için kullanılır.

2. Sayfa Hataları ve Taşıma: Eğer bir uygulama, bellekte olmayan bir sayfaya erişmeye çalışırsa, bir sayfa hatası meydana gelir. Bu durumda, işletim sistemi gerekli sayfayı sayfa dosyasından (veya uygulamanın executable dosyasından) belleğe taşır.

3. Bellek Yönetimi ve Sayfa Değiştirme: Windows, aktif olarak kullanılmayan sayfaları sayfa dosyasına taşıyarak, RAM'de yer açar. Bu, belleği daha etkin kullanmaya olanak tanır.

2. Uygulama Sırasında Ortaya Çıkabilecek Durumlar:

1. Yetersiz Sanal Bellek (Insufficient Virtual Memory): Eğer sayfa dosyası ve fiziksel bellek yetersizse, sistem performansı düşebilir. Bu durumda, kullanıcıdan sayfa dosyasının boyutunu artırması istenebilir.

2. Aşırı Sayfa Değiştirme (Thrashing): Eğer sistem sürekli olarak sayfaları bellekten diske ve diskten belleğe taşıyorsa, bu aşırı sayfa değiştirme (thrashing) olarak adlandırılır ve sistem performansını ciddi şekilde etkileyebilir.

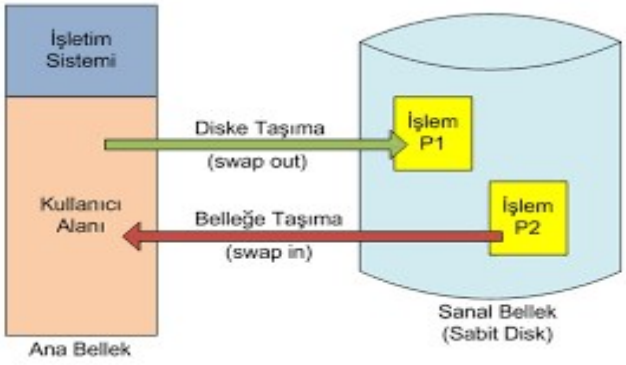
3. Bellek Sızıntıları (Memory Leaks): Uygulamalar hatalı bellek yönetimi nedeniyle gereksiz bellek alanı kaplayabilir. Bu, zamanla belleğin dolmasına ve sistem performansının düşmesine neden olabilir.

3. Önlemler ve Yönetim:

1. Sayfa Dosyası Boyutunu Yönetme: Kullanıcılar, sayfa dosyasının boyutunu manuel olarak ayarlayabilir veya sistemin otomatik yönetimine bırakabilir.

2. Uygulama Güncellemeleri: Bellek sızıntıları genellikle uygulama güncellemeleriyle düzeltilir. Kullanıcıların uygulamalarını düzenli olarak güncellemeleri önerilir.

3.Sistem İzleme Araçları: Windows Görev Yöneticisi ve Performans İzleyicisi gibi araçlar, bellek kullanımını izlemek ve potansiyel sorunları tespit etmek için kullanılabilir.



Şekil 12: Sanal bellek yönetimi örnek şeması.

KAYNAKÇA:

[1] Sacco G. M., Schkolnick M. (1982): "A Mechanism for Managing the Buffer Pool in a Relational Database System Using the Hot Set Model", In Proceedings of the 8th International Conference on Very Large DataBases, sf. 257–262, Mexico City, Meksika

[2] Peter J. D. (1970): "Virtual Memory", Computing Surveys, Cilt 2, No. 3, sf. 120-135.

[3] Stefan M., Peter B. (2002):"Optimizing Main-Memory Join on Modern Hardware", IEEE transactions on knowledge and data engineering, Cilt 14, No. 4, sf. 210-220.

[4] Đnternet: TBD Kamu Bilgi Đşlem Merkezleri Yöneticileri Birliği (TBD Kamu BĐB), "Sanallaştırma", TBD Kamu-BĐB Kamu Bilişim Platformu XII, http://www.tbd.org.tr/usr_img/cd/kamubib12/rapor larPDF/RP1-2010.pdf, 7,13-16, (2010).

[5] Lunsford, D. L., "Virtualization Technologies in Information Systems Education", Journal of Information Systems Education (JISE), Volume 20, Article 3, 339-348, (2009)

[6] M. Suetake, T. Kashiwagi, H. Kizu, and K. Kourai, "S-memV: Split Migration of Large-memory Virtual Machines in IaaS Clouds," in Proc. Int. Conf. Cloud Computing, 2018, pp. 285–293.

[7] https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=10197098

[8] https://arxiv.org/abs/1510.03362

[9] . Borodin, A., El-Yaniv, R.: Online computation and competitive analysis. Cambridge University Press, New York, NY, USA (1998)

[10] Edward G. Coffman, Jr. and Peter J. Denning Operating Systems Theory. Prentice-Hall, 1973

[11] https://bilgisayarkavramlari.com/2009/05/29/sayfa-degistirme-algoritmasi-page-replacement/#4

[12] https://web.archive.org/web/20201025165312/https://www.hurriyet.com.tr/teknoloji/ssd-nedir-nasil-takilir-ve-calisir-41408909