



KONYA TEKNİK ÜNİVERSİTESİ

**MÜHENDİSLİK VE DOĞA BİLİMLERİ
FAKÜLTESİ**

YAZILIM MÜHENDİSİLİĞİ BÖLÜMÜ

İŞLETİM SİSTEMLERİ DERSİ


VİZE ÖDEVİ RAPORU

No: 211229050

Ad: MEHMET DOĞAN

Soyad: UYANIK

Github Link: https://github.com/mhmtkgnu/prime_with_thrad_project

| | | |
|---|--|---|
|  | Yazılım Mühendisliği İşletim Sistemleri Dersi Vize Ödevi | Ad-Soyad: Mehmet Doğan UYANIK 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 araştırınız. Her bir algoritmanın avantajlarını ve dezavantajlarını belirleyerek karşılaştırmalı 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 fragmentasyonu ile 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 Roman yazı 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.

- 4 Thread bulduğu çift sayıları ortak bir ArrayList'e ekleyecektir.
- 4 Thread bulduğu tek sayıları ortak bir ArrayList'e ekleyecektir.
- 4 Thread bulduğu asal sayıları ortak bir ArrayList'e ekleyecektir.
- ArrayList'ler ilk oluşturulduklarında boş olacaklardır.

```
1 using System;
2 using System.Collections;
3 using System.Diagnostics;
4 using System.Threading;
5
6 başvuru
7 class Program
8 {
9     public static ArrayList arrayList = new ArrayList();
10    public static ArrayList oddarrayList = new ArrayList();
11    public static ArrayList evenarrayList = new ArrayList();
12    public static ArrayList primearrayList = new ArrayList();
13    public static ArrayList[] dividedarrayLists;
14    static ManualResetEvent even1 = new ManualResetEvent(false);
15    static ManualResetEvent even2 = new ManualResetEvent(false);
16    static ManualResetEvent even3 = new ManualResetEvent(false);
17    static ManualResetEvent even4 = new ManualResetEvent(false);
18 }
```

1. Aşama: Kodun 1. ve 5. Satırları arasında kod için gerekli kütüphaneler projeye dahil edilmiştir. Bizim projemizin ehemmiyeti açısından `using System.Threading;` kütüphanesi bizim projemiz için gerekli thread işlemleri fonksiyonlarını içermektedir.

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

- `ManualResetEvent` .NET'de eş zamanlı (concurrent) programlamada kullanılan bir senkronizasyon primitifidir. `System.Threading` ad alanı altında yer alır ve birden fazla iş parçacığı (thread) arasında sinyal mekanizması olarak kullanılır. `ManualResetEvent` iki ana durumda olabilir: sinyalli (set) ve sinyalsiz (unset). Bu durumlar, iş parçacıklarının beklemesi veya devam etmesi için kullanılır. Bizim kodumuzda parametrelerini `false` değeri almasının sebebi ise ilk olarak sinyalsiz başlatmak içindir.

```

17
18 11 başyuru
19 class ThreadParameters
20 {
21     2 başyuru
22     public int INDEX { get; set; }
23     2 başyuru
24     public ManualResetEvent EventParameter { get; set; }
25
26     4 başyuru
27     public ThreadParameters(int index, ManualResetEvent eventParam)
28     {
29         INDEX = index;
30         EventParameter = eventParam;
31     }
32 }

```

3. Aşama: Kodun 18. ve 28. Satırları arasında ise threadleri çalıştırmak için gerekli parametreleri bulunduran ve parametre geçişini kolaylaştıran classımız vardır.

- Bu classın içinde ise INDEX ve ManualResetEvent EventParameter değişkenlerinin get-set işlemleri bulunmaktadır aynı zamanda ilgili classın içinde kurucu metodumuz olan `public ThreadParameters(int index, ManualResetEvent eventParam)` metodumuz vardır mu metod INDEX özelliğini, verilen index parametresi ile başlatır. Diğer EventParameter özelliği için de aynı işlemleri gerçekleştirir.

```

29 0 başyuru
30 static void Main(string[] args)
31 {
32     // Listeye 1.000.000 öğe ekle
33     int i = 0;
34     while (i < 1000000)
35     {
36         arrayList.Add(i);
37         i++;
38     }
39
40     // ArrayList'teki öğeleri yazdır
41     dividedarrayLists = DivideArrayList(arrayList, 4);

```

4. Aşama: Kodun 29. satırından itibaren main kısmı başlamış bulunmaktadır.

- Kodun 33. satırında başlayıp 37. satırında biten `while` döngüde ise arrayList listesinde 0 dan başlayıp 999.999 a kadar 1.000.000 adet ardışık sayıyı arrayList listesinde eklemektedir.
- arrayList listesine eklenen bu sayılar 40. 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. *Çağırma prensibi: `dividedarrayLists[index][i]` şeklinde bir nevi matris yapısınabüründürüp örnekteki gibi çağrılmaktadır.

```
42 Stopwatch stopwatch = new Stopwatch();
43 stopwatch.Start();
44 Console.WriteLine("Toplam çalışma süresi sayacı başladı.");
45
46 Thread thread1 = new Thread(Function0);
47 Thread thread2 = new Thread(Function0);
48 Thread thread3 = new Thread(Function0);
49 Thread thread4 = new Thread(Function0);
50
51 ThreadParameters parameters1 = new ThreadParameters(0, even1);
52 ThreadParameters parameters2 = new ThreadParameters(1, even2);
53 ThreadParameters parameters3 = new ThreadParameters(2, even3);
54 ThreadParameters parameters4 = new ThreadParameters(3, even4);
55
56 thread1.Priority = ThreadPriority.Highest;
57 thread1.Start(parameters1);
58
59 thread2.Priority = ThreadPriority.AboveNormal;
60 thread2.Start(parameters2);
61
62 thread3.Priority = ThreadPriority.Normal;
63 thread3.Start(parameters3);
64
65 thread4.Priority = ThreadPriority.BelowNormal;
66 thread4.Start(parameters4);
67
68 // Tüm olayların tamamlanmasını bekleyin
69 while(Monitor.WaitAll(new WaitHandle[] { even1, even2, even3, even4 }));
70
71 stopwatch.Stop();
72
73 Console.WriteLine("Toplam thread çalışma süresi: {stopwatch.ElapsedMilliseconds} milisaniye, " + "{stopwatch.ElapsedMilliseconds / 1000.0} saniye.");
74 Console.WriteLine("Eşit sayı adedi: " + parameters1.Count);
75 Console.WriteLine("Tek sayı adedi: " + oddarray1.Count);
76 Console.WriteLine("Çift sayı adedi: " + evenarray1.Count);
77 Console.ReadLine();
```

5. Aşama:

- Kodun 42. ve 43. satırlarında ise süre ölçmek için gerekli tanımlamalar yapıлып süre sayacı başlatılmaktadır.
- 46. ve 50. satırları arasında ise gerekli olan 4 thread `Function0` parametresini alacak şekilde üretmeler tanımlamaları yapılmaktadır.
- Tanımlanan threadlere 51. ve 54. satırlar arasında parametre atama işlemleri gerçekleştirilmektedir.
- Oluşturulan threadler ve atanan parametreler ışığında threadlere öncelik sırası atama işlemleri 57. ve 67. satırları arasında yapılmış ve işlemler şu şekilde gerçekleşmiştir:

➤ `thread1.Priority = ThreadPriority.Highest;`
`thread1.Start(parameters1);`

Highest komutu threadin çalışmasındaki öncelik sırasını belirler

Öncelik sırası ve yapıları şu şekildedir:

- Lowest:** İş parçacığına en düşük öncelik atanır. Diğer iş parçacıklarına göre daha az işlemci zamanı alır.

2. **BelowNormal:** Normalden düşük öncelik. İş parçacığı, normal öncelikli iş parçacıklarına göre daha az işlemci zamanı alır.
3. **Normal:** İş parçacığına varsayılan öncelik atanır. Çoğu iş parçacığı bu öncelik seviyesinde çalışır.
4. **AboveNormal:** Normalden yüksek öncelik. İş parçacığı, normal öncelikli iş parçacıklarına göre daha fazla işlemci zamanı alır.
5. **Highest:** İş parçacığına en yüksek öncelik atanır. Bu, iş parçacığının diğer tüm iş parçacıklarından daha sık çalıştırılmasını sağlar.

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. Bu, düşük öncelikli iş parçacıklarının yeterli işlemci zamanı alamaması anlamına gelir.

- Kodumda ise thread1'den thread4'e kadar öncelik sırası yüksekten düşüğe gidecek şekilde ayarlanıp koda dökülmüş ve threadler `thread1.Start(parameters1);` komutuyla başlatılmıştır.
- 70. satırda ise `WaitHandle.WaitAll(new WaitHandle[] { even1, even2, even3, even4 });` komutu ile bütün işlemlerin bitmesi beklenmiştir.
- 72. Satırda da 42. satırda tanımlanıp 43. satırda başlatılan süre sayacı durdurulmuştur.
- 74. ve 78. satırlarda ise console çıktıları verilmiştir.

BURADAN İTİBAREN KOD lock OLMAYAN ve lock OLAN ŞEKLİNDE 2' YE AYRILIYOR

Lock Nedir: Lock ifadesini kullanarak çok iş parçacıklı (multithreaded) bir ortamda thread güvenliği sağlamayı amaçlar. lock ifadesi, belirli bir kod bloğunun sadece bir iş parçacığı tarafından aynı anda erişilebilir olmasını sağlar. Bu sayede, aynı kaynağa birden fazla iş

LOCK OLMAYAN KOD:

```

104 public void run() {
105     try {
106         // Thread parameters
107         ThreadParameters parameter = ThreadParameters(parameters);
108         ManualResetEvent eventParam = parameter.EventParameter;
109         Stopwatch stopwatch = new Stopwatch();
110         stopwatch.Start();
111
112         for (int i = 0; i < dividendarray.Length(parameters.Length); i++)
113         {
114             int count = 0;
115             for (int j = 0; j < ((int)dividendarray.Length(parameters.Length) / 2); j++)
116             {
117                 if (((int)dividendarray.Length(parameters.Length) & j) == 0)
118                 {
119                     count++;
120                     break; // mat vlandığı anı takip etme işi
121                 }
122             }
123
124             if (count == 0)
125             {
126                 if ((int)dividendarray.Length(parameters.Length)[1] < 0 || ((int)dividendarray.Length(parameters.Length)[1] != 1)
127                 {
128                     parameter.List.Add(((int)dividendarray.Length(parameters.Length)[1]));
129                 }
130
131                 if ((int)dividendarray.Length(parameters.Length)[1] >= 0)
132                 {
133                     wwwarray.List.Add(dividendarray.Length(parameters.Length)[1]);
134                 }
135                 else
136                 {
137                     mhrarray.List.Add(dividendarray.Length(parameters.Length)[1]);
138                 }
139             }
140             stopwatch.Stop();
141             Console.WriteLine("C# thread - " + parameter.Length + " - " + stopwatch.ElapsedMilliseconds + " milliseconds - " + stopwatch.ElapsedMilliseconds / 1000.0 + " seconds -");
142             eventParam.Set();
143         }
144     }
145 }

```

- Bu aşamada 104. satır ve 110. satırlar arasında fonksiyona gönderilen parametreler alınıyor ve ilgili thread'in süre ölçüm işlemleri başlatılıyor.
- Devamı ise asal sayı bulma şeklinde devam etmektedir.
- Bulunan asal sayılar `primearrayList` listesine, tek sayılar `oddarrayList` listesinde, çift sayılar ise `evenarrayList` listesine eklenmektedir.
- 140 ile 143. satırlar arasında ise ilgili thread için başlatılan süre sayacı durdurulup ölçüm yapılıyor ve `eventParam.Set()`; komutu ile de thread'in sinyal durumu güncellenmektedir.

The screenshot displays the Visual Studio IDE with a C# project named 'THREAD_LOCK_OFF'. The main code file, 'Program.cs', contains a multi-threaded application. It defines a 'ThreadParameters' struct with 'Index' and 'ManualResetEvent' fields. The 'Main' method creates an array of these parameters and starts four threads. Each thread calls a 'ThreadFunction' that simulates work by sleeping for a duration specified in the 'Index' field. After all threads complete their work, the main thread prints the total execution time and the count of threads.

```
using System;
using System.Threading;

struct ThreadParameters {
    int Index;
    ManualResetEvent EventParam;
}

class Program {
    static void Main() {
        ThreadParameters[] param = new ThreadParameters[4];
        param[0].Index = 10;
        param[1].Index = 20;
        param[2].Index = 30;
        param[3].Index = 40;

        Thread[] threads = new Thread[4];
        for (int i = 0; i < 4; i++) {
            threads[i] = new Thread(() => ThreadFunction(param[i]));
            threads[i].Start();
        }

        Console.WriteLine("Toplam calisma süresi sayaci basladi.");
        Thread.Sleep(10000);
        Console.WriteLine("Thread 01 calisma süresi: 10033 milisaniye, 10,433 saniye.");
        Console.WriteLine("Thread 11 calisma süresi: 20086 milisaniye, 20,086 saniye.");
        Console.WriteLine("Thread 21 calisma süresi: 40010 milisaniye, 40,010 saniye.");
        Console.WriteLine("Thread 31 calisma süresi: 67035 milisaniye, 67,035 saniye.");
        Console.WriteLine("Toplam thread calisma süresi: 67079 milisaniye, 67,079 saniye.");
        Console.WriteLine("Tek sayi addi: 499875");
        Console.WriteLine("Çift sayi addi: 499849");
    }

    static void ThreadFunction(ThreadParameters param) {
        Thread.Sleep(param.Index * 1000);
    }
}
```

The console window shows the output of the program, which matches the code's logic. It prints the start of the timer, the completion time for each thread, and the total execution time for all threads. The final output shows the sum of odd and even numbers, which is 499875 for odd and 499849 for even.

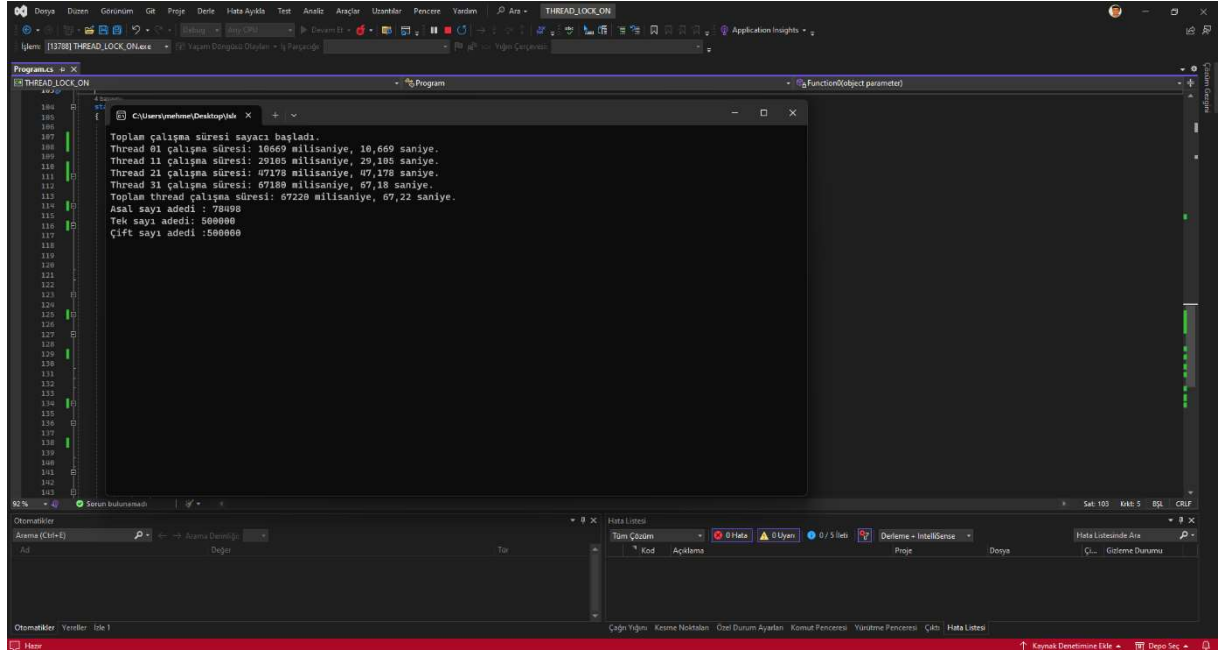
```
Toplam calisma süresi sayaci basladi.
Thread 01 calisma süresi: 10033 milisaniye, 10,433 saniye.
Thread 11 calisma süresi: 20086 milisaniye, 20,086 saniye.
Thread 21 calisma süresi: 40010 milisaniye, 40,010 saniye.
Thread 31 calisma süresi: 67035 milisaniye, 67,035 saniye.
Toplam thread calisma süresi: 67079 milisaniye, 67,079 saniye.
Tek sayi addi: 499875
Çift sayi addi: 499849
```

- LOCK OLAN KOD:**

6.2. Aşama:

- Bu aşamada 104. satır ve 110. satırlar arasında fonksiyona gönderilen parametreler alınıyor ve ilgili threadın süre ölçüm işlemleri başlatılıyor.
- Devamı ise asal sayı bulma şeklinde devam etmektedir.
- Bulunan asal sayılar `primearrayList` listesine, tek sayılar `oddarrayList` listesinde, çift sayılar ise `evenarrayList` listesine eklenmektedir.
- Burada eklenirken `lock` işlemi kullanılmaktadır yani ilgili listeyi sadece ilgili thread kullanabilsin diye.
- 140 ile 143. satırlar arasında ise ilgili thread için başlatılan süre sayacı durdurulup ölçüm yapılıyor ve `eventParam.Set();` komutu ile de threadin sinyal durumu güncellenmektedir.

İşlemler neticesinde örnek konsol çıktısı şekildeki gibidir:



```
Toplam çalışma süresi sayacı başladı.  
Thread 01 çalışma süresi: 10669 milisaniye, 10,669 saniye.  
Thread 11 çalışma süresi: 29105 milisaniye, 29,105 saniye.  
Thread 21 çalışma süresi: 47178 milisaniye, 47,178 saniye.  
Thread 01 çalışma süresi: 67189 milisaniye, 67,189 saniye.  
Toplam thread çalışma süresi: 67228 milisaniye, 67,22 saniye.  
Asal sayı adedi : 78098  
Tek sayı adedi : 500000  
Çift sayı adedi : 500000
```

- İşlemler neticesinde çıktılar şekildeki gibidir. Peki neyi farketmeliyiz süre çıktıları yanı sıra toplam locksuz koddaki veri kaybı burada yok lock sisteminin faydası.

Soru 2

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

1 - Sanal Bellek Kavramları:

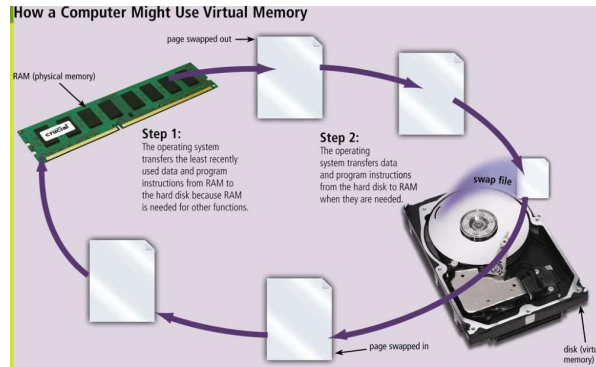
-Sanal bellek nedir, nasıl çalışır, işletim sistemi açısından avantajları ve dezavantajları nelerdir? Bu konuda kapsamlı bir literatür taraması yaparak kavramları açıklayınız.

Springer ile Literatür Taraması:

Sanal Bellek Nedir: Çok iş parçacıklı programlama modeli, büyük programlama kolaylığı sağlar ve donanım paralelliğinden elde edilen performans avantajını artırır [1], [2]. Bu nedenle, büyük veri çerçeveleri [10], web sunucuları [3] ve anahtar-değer depoları [9] gibi gerçekçi sistem ve uygulamalarda yaygın olarak kullanılmaktadır. Bununla birlikte, işletim sistemi çekirdekleri içindeki mevcut sanal bellek sistemleri, donanım sistemi paralelliğinden yeterince yararlanamayan zayıf ölçeklenebilirlik performansı sergilemektedir [1], [2], [4], [5], [6]. Daha da kötüsü, çok çekirdekli sistemlerde çok iş parçacıklı uygulamaların performansının iyileştirilmesini engeller.

Sanal bellek sistemi, fiziksel belleği iki ana sistem kaynağıyla çoğaltır: sanal bellek alanı ve sayfa tablosu hiyerarşisi. Birden çok iş parçacığı için paylaşılan bir adres alanı sağlamak amacıyla işletim sistemi çekirdekleri çeşitli merkezi tasarım yaklaşımlarını benimser. Bu makale, bu sistem tasarımlarının neden olduğu ölçeklenebilirlik sorunlarına odaklanmaktadır. İlk olarak, emtia sanal bellek sistemleri, iş parçacıkları için paylaşılan bir sanal bellek alanı sağlar. Sanal bellek alanı birkaç bellek bölgesine bölünmüştür. Bu bellek bölgeleri belirli bölge meta verileri tarafından yönetilir. VM sistemi bu bölge meta verilerini yönetmek için bir bölge ağacı kullanır. VM sistemiyle ilgili herhangi bir işlemin, karşılık gelen bölge meta verilerini bulmak için bu paylaşılan bölge ağacında arama yapması gerekir. Ne yazık ki, ekleme, kaldırma ve arama gibi eşzamanlı ağaç işlemleri genel bir ağaç kilidi tarafından serileştirilir. İri taneli bir kilidin kullanılması yalnızca çoklu sistem işlemlerini serileştirmekle kalmaz, aynı zamanda ciddi kilit çekişmelerine de neden olur [1], [4], [7], [8].

Ayrıca VM sistemi, sanal-fiziksel adres eşlemelerini depolamak için genel bir sayfa tablosu hiyerarşisini kullanır. Paylaşılan adres alanı üzerinde çok sayıda iş parçacığı çalıştırıldığında, birçok sayfa hatası işleyicisi, merkezi sayfa tablosu ağacına aynı anda erişebilir. VM sistemi, sayfa tablosu ağacını genel bir kilitle (PGDIR kilidi) veya daha ayrıntılı kilitlerle, yani sayfa tablosu sayfası başına ayrı kilitle (PTE kilidi) korur [2]. Bir sayfa tablosu sayfası 512 sayfa tablosu girişinden oluştuğundan, ince taneli bir kilit şeması kullanılsa bile, en kötü senaryoda sayfa tablosu sayfasına erişen yüzlerce paralel yürütme iş parçacığı olacaktır ve bu da ağır kilit çekişmesine neden olacaktır [1], [4].



Şekil 1: Sanal bellek yapısını anlatan bir görsel.

Sanal Bellek Nasıl Çalışır: Bir program çalıştırıldığında, programlar bellek kullanır ve işletim sistemi hangi programın bilgisayarın belleğinin hangi bölümlerini kullandığını takip eder ve her programın ihtiyacı olan bellek miktarını tahsis eder. Eğer gerekli bellek mevcut değilse, işletim sistemi o anda başka bir programın ihtiyaçlarının "daha az önemli" olduğuna karar verebilir ve o program tarafından kullanılan bazı bellekleri serbest bırakır. Bu, önce mevcut içeriği diske yazarak ("swapping out") ve ardından boşalan belleği talepte bulunan programa tahsis ederek yapılır. Daha sonra, belleği diske taşınan programın o belleğe tekrar ihtiyacı olduğunda, bu bellek diske geri okunarak ("swapped in") programın kullanımına sunulur.

Sanal bellek yönetimi için kullanılan üç yaygın algoritma vardır: FIFO (First-in, First-out), LRU (Least Recently Used) ve OPT (Optimal). FIFO, RAM'de en uzun süredir bulunan veriyi sanal belleğe taşıma fikrine dayanır. LRU, RAM'de en uzun süredir kullanılmayan veriyi değiştirir. OPT algoritması, LRU gibi geçmiş veri kullanımını dikkate alırken, hangi verinin yakında ihtiyaç duyulacağını da tahmin eder [15].

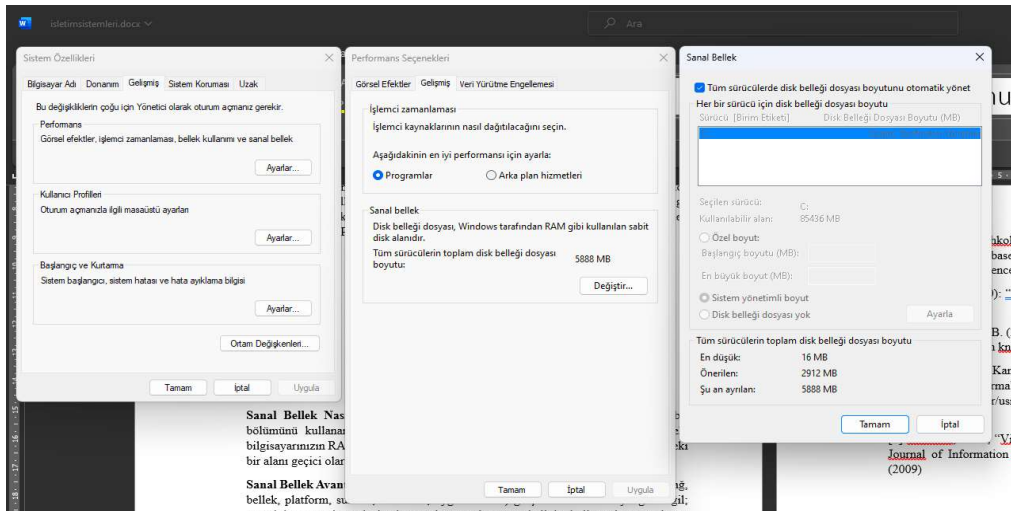
Sanal Bellek Avantajları: Sanallaştırma, bilişim kaynaklarının (işlemci, depolama, ağ, bellek, platform, sunucu, masaüstü, uygulama vb.) gerçekte var olan kaynağın değil; gerçek kaynağa dayandırılarak tanımlanmış olan soyut halinin, kullanıcılara sunulması olarak tanımlanabilir. Böylece gerçek kaynak, göreceli olarak daha az kapasiteli çok sayıda sanal kaynak olarak kullanılabilir [11]. Sunucuyu barındıracak ortamı veya donanımı sanallaştırmak olmak üzere iki tür temel sanallaştırma ortamı bulunmaktadır [12].

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 düşüslere 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 gösterimi.

Çıkarımlar:

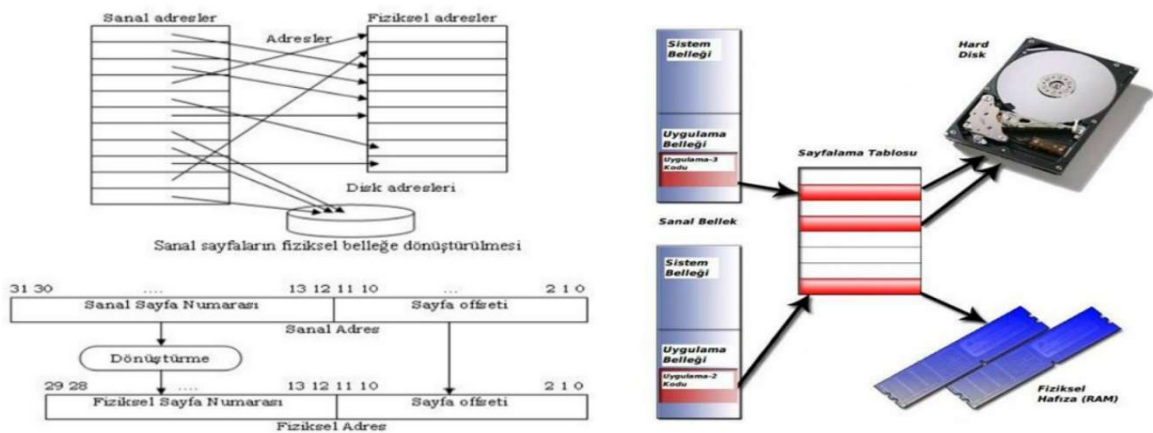
*Kişisel bilgisayarımda sanal bellek yönetimi otomatik olarak gerçekleşmektedir.

*Anlık olarak: en düşük 16MB, önerilen 2912MB, şu an ayrılan 5888MB şeklinde ölçüldüğünü ve saptandığını fark ettim.

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

Sanal Bellek Nedir: Hedef ana makine yeterli belleğe sahip değilse, bellek-sanallaştırılan 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 [13][15].

Sanal bellek, işletim sisteminin sabit disk alanını gerçek bellekmiş gibi kullanmasıdır. Bir bilgisayarın RAM'i yetersiz kaldığında, işletim sistemi RAM'den sabit diske veri aktarır. Bu süreç, işletim sisteminin sabit diske ayrılmış özel bir alanı (paging file) sanal bellek olarak kullanmasıyla gerçekleşir [14][16].

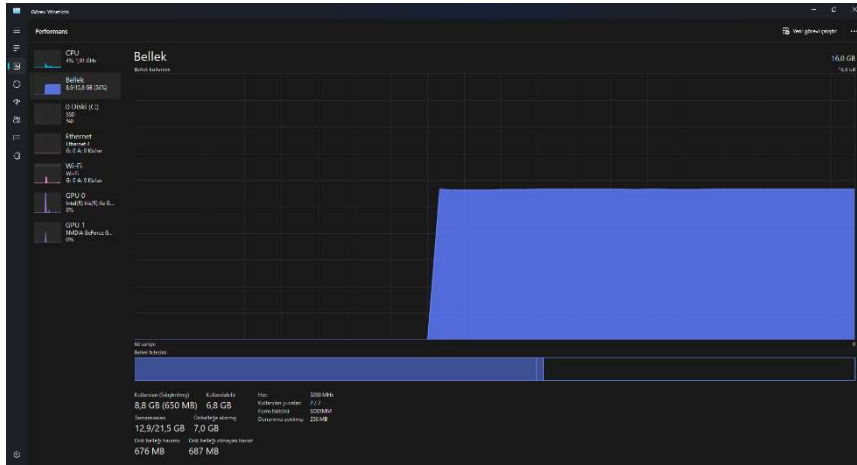


Şekil 3 ve 4: Sanal bellek çalışma prensibi.

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 [16].

Sanal Bellek Dezavantajları: Sanal belleğin başlıca dezavantajları, performans sorunları ve sistem kaynaklarının yönetimindeki zorluklardır. Sabit diskler RAM'e göre daha yavaş olduğundan, sanal belleğin kullanılması genel sistem performansını düşürebilir. Özellikle, işletim sistemi RAM ve sanal bellek arasında sık sık veri alışverişi yapmak zorunda kalırsa, bu durum "thrashing" adı verilen ve bilgisayarın önemli ölçüde yavaşlamasına neden olan bir performans düşüşüne yol açabilir. Thrashing, işletim sisteminin çok fazla kaynağını sanal belleği yönetmeye ve sayfalama dosyasını güncellemeye ayırdığı zaman gerçekleşir. Ayrıca, sanal bellek mekanizması, işletim sistemi ve uygulamaların karmaşıklığını artırır, çünkü bellek yönetimi ve veri erişimini optimize etmek için ek algoritmalar ve yönetim stratejileri gerektirir. Bu durum, özellikle bellek yoğun uygulamalarda ve çoklu görev ortamlarında sistem yöneticileri ve geliştiriciler için ek zorluklar yaratır.

Çıkarımlar:



Şekil 4: Kişisel bilgisayar bellek performansı.

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

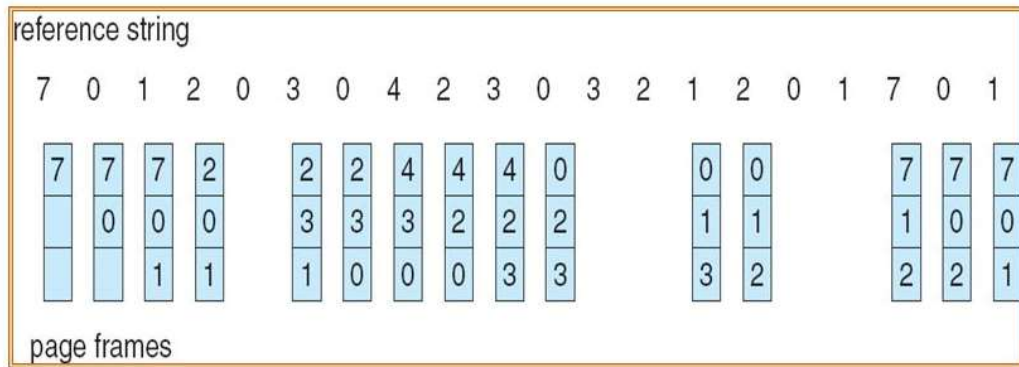
2- Sayfa Değiştirme Algoritmaları:

-FIFO, Optimal, LRU gibi sayfa değiştirme algoritmalarını araştırın. Her bir algoritmanın avantajlarını ve dezavantajlarını belirleyerek karşılaştırmalı 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 sayısı ile 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 5: FIFO Sayfa değiştirme algoritması anlatım şekli.

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

1. Basit Uygulanabilirlik: FIFO algoritması, basit bir veri yapısı olan kuyruk kullanarak sayfaları takip etmektedir. Bu yapı, algoritmanın anlaşılmasını ve uygulanmasını oldukça basit bir hale getirir.

2. Düşük İşlem Maliyeti: FIFO algoritması ek bilgi saklama ihtiyacı olmadan çalıştığı için düşük işlem maliyetine sahiptir. Bu da sistem kaynaklarının verimli kullanımına olanak tanır.

3. Tahmin Edilebilir Davranış: Algoritmanın basit mantığı, hangi sayfanın ne zaman değiştirileceği konusunda tahmin edilebilir bir davranış sergilemesini sağlar, bu da sistemlerde belirli bir düzeyde performans istikrarı sağlar.

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

1. Belady Anomalisi Riski: FIFO algoritması bazı durumlarda, özellikle çerçeve sayısının arttığı durumlarda, sayfa hatalarının beklenmedik şekilde artmasına yol açabilir, bu da performans düşüşlerine neden olabilir.

2. Mekânsal Yerellik İçin Uygunsuzluk: Sayfaların yakın zamanda tekrar kullanılma olasılığının yüksek olduğu mekânsal yerellik durumlarında FIFO algoritması etkili değildir. Bu, sık kullanılan sayfaların gereksiz yere değiştirilmesine yol açabilir.

3. Düzensiz Erişim Desenlerinde Performans Düşüklüğü: Erişim desenlerinin düzensiz olduğu durumlarda, FIFO algoritması sayfa kullanım sıklığını ve zamanlamasını göz önünde bulundurmadığı için yetersiz kalabilir ve bu da sık sayfa hatalarına ve sistem performansının düşmesine neden olabilir.

- Bu bilgiler, FIFO algoritmasının basit yapısının hem avantaj hem de dezavantajlarını gösterir. Algoritma, özellikle basit sistemlerde ve tahmin edilebilir iş yükleri için kullanışlı olabilirken, daha karmaşık ve dinamik iş yükleri için uygun olmayabilir. Kaynak: PrepBytes.

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 başvurulmuş sayfalar arasında ayırma 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 6: LRU sayfa değiştirme algoritması anlatım şekli.

LRU (Least Recently Used) Avantajları:

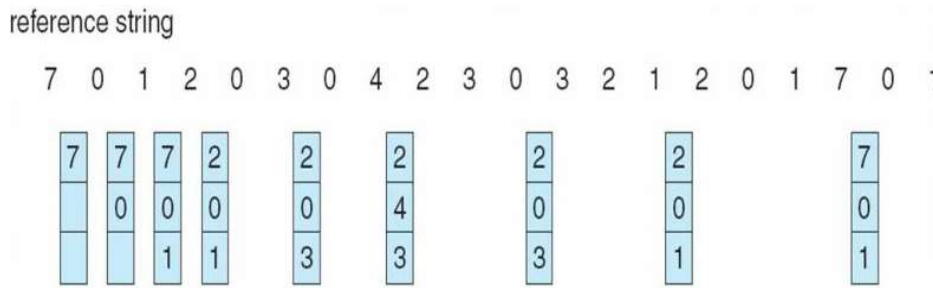
- 1. Yakın Zamanda Kullanılan Sayfaları Korur:** LRU, en son kullanılan sayfaların bellekte kalmasını sağlar, bu da sık kullanılan sayfaların tekrar yüklenme ihtiyacını azaltır ve sistem performansını artırır.
- 2. Kullanım Sıklığına Göre Optimizasyon:** Bu algoritma, sayfaların kullanım sıklıklarını dikkate alarak, nadiren kullanılan sayfaların yerine yeni sayfaların yüklenmesini sağlar, böylece bellek kullanımını optimize eder.
- 3. Dinamik Bellek Yönetimi:** LRU, sürekli değişen çalışma kümelerine iyi uyum sağlar ve belleği dinamik bir şekilde yöneterek farklı uygulama ve iş yükleri için esneklik sunar.

LRU (Least Recently Used) Dezavantajları:

- 1. Yüksek Maliyet ve Karmaşıklık:** LRU algoritmasının uygulanması, sayfa kullanım sırasını takip etmek için ek hesaplamalar ve veri yapıları gerektirir, bu da işlem maliyetini ve karmaşıklığını artırır.
- 2. Performans Düşüklüğü Riski:** Yoğun bellek kullanımı durumlarında, algoritma bellekten hangi sayfaların çıkarılacağını belirlemek için zaman harcar ve bu süreç sistem performansını olumsuz etkileyebilir.
- 3. Düşük Etkinlik Durumları:** LRU, kısa süre önce kullanılmış ancak tekrar kullanılmayacak sayfaların gereksiz yere bellekte tutulmasına neden olabilir, bu da bellek kaynaklarının etkisiz kullanımına yol açar.

- LRU algoritması, genellikle kullanım sıklığına dayalı bellek yönetimi için etkili bir çözüm sunarken, uygulama karmaşıklığı ve bazı durumlarda düşük performans gibi dezavantajlara da sahiptir.

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 7: Optimal sayfa değiştirme algoritması anlatım şekli.

Optimal Avantajları:

1. Minimum Sayfa Hatası: Optimal algoritma, gelecekteki sayfa erişimlerini bilerek en uzun süre kullanılmayacak sayfayı değiştirir, bu da minimum sayfa hatası sağlar ve bellek kullanımını en verimli hale getirir.

2. Teorik Olarak En İyi Performans: Bu algoritma teorik olarak mümkün olan en iyi sayfa değiştirme performansını sunar, çünkü her sayfa hatasında en uzun süre kullanılmayacak sayfayı seçer.

3. Bellek Kullanımının Optimize Edilmesi: Optimal algoritma, bellekteki sayfaların en verimli şekilde kullanılmasını sağlar, bu da özellikle bellek kaynaklarının sınırlı olduğu durumlarda önemlidir.

Optimal Dezavantajları:

1. Pratikte Uygulanabilir Olmaması: Optimal algoritma, gelecekteki sayfa erişimlerini bilmeyi gerektirir, bu da pratik uygulamalarda mümkün olmamaktadır ve bu algoritmayı teorik bir model olarak sınırlar.

2. Yüksek Hesaplama Gereksinimi: Algoritmanın gelecekteki erişimleri tahmin etmesi gerektiğinden, bu süreç karmaşık hesaplamalar gerektirir ve bu da uygulama maliyetini artırır.

3. Belirsiz Çalışma Kümesi Durumlarında Etkisizlik: Eğer gelecekteki sayfa erişimleri belirsiz ise, bu algoritma en etkili şekilde çalışamaz ve bu da bellek yönetiminde etkisizliğe yol açabilir.

- Optimal algoritma, teorik olarak en iyi sayfa değiştirme stratejisini sunmasına rağmen, pratik uygulamalarda kullanılması zordur ve gelecekteki erişimlerin bilinmesini gerektirir. Bu nedenle, genellikle akademik çalışmalarda ve teorik analizlerde kullanılır, ancak gerçek dünya uygulamalarında sınırlıdır.

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 yapmaktadır.

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 olduđu 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: FIFO (First-In, First-Out) algoritması, sıralı işlem gerektiren çeşitli uygulama alanlarında kullanılır. Bu algoritma, verilerin sırasını koruması gereken durumlarda etkilidir.

1. İşletim Sistemleri: Bir işletim sisteminde, FIFO, basit bellek yönetimi ve çoklu işlem yönetimi için kullanılabilir. Özellikle, hafızaya ilk giren işlemlerin ilk çıkarılmasını gerektiren senaryolarda tercih edilir.

2. Veri Akışı Yönetimi: Ağ yönlendiricilerinde ve sunucularda, veri paketlerinin sıralı bir şekilde işlenmesi için FIFO kullanılabilir. Bu, paketlerin alındığı sırayla işlenmesini ve yönlendirilmesini sağlar.

3. Yazılım Uygulamaları: Yazılım geliştirmede, özellikle kuyruk veri yapılarıyla çalışılırken FIFO algoritması tercih edilebilir. Örneğin, bir yazıcı sırası yönetiminde veya müşteri hizmetleri taleplerinin sıralı işlenmesinde FIFO kullanılabilir.

LRU: LRU (Least Recently Used) algoritması, veri erişiminin sıklığına ve zamana dayalı olarak önemli olan durumlarda kullanılır.

1. Web Tarayıcıları: Web tarayıcılarının önbelleğinde, sık ziyaret edilen web sayfalarını hızlı bir şekilde yüklemek için LRU kullanılır. Az kullanılan sayfalar önbellekten silinirken, sık kullanılanlar tutulur.

2. Veri Tabanı Sistemleri: Veri tabanı sorgularında, sık kullanılan sorgu sonuçlarını hızlı erişim için önbellekte tutmak amacıyla LRU algoritması tercih edilebilir.

3. Dosya Sistemleri: Dosya sistemlerinde, sık erişilen dosyaların hızlı bir şekilde erişilebilir durumda tutulması için LRU algoritması kullanılır. Bu, sistem performansını artırır ve dosya erişim sürelerini kısaltır.

Optimal: Optimal sayfa değiştirme algoritması, genellikle teorik veya eğitim amaçları için kullanılır, çünkü gerçek zamanlı uygulamalarda gelecekteki erişimlerin öngörülmesi mümkün değildir. Bu algoritma, sayfa değiştirme stratejilerinin analizinde ve performans ölçümlerinde bir referans noktası olarak hizmet eder.

1. Akademik Arařtırmalar: Bilgisayar bilimi arařtırmalarında, yeni geliřtirilen sayfa deęiřtirme algoritmalarının etkinlięini deęerlendirmek için optimal algoritma kullanılır. Bu, teorik en iyi performansla karřılařtırma yapmak için bir temel saęlar.

2. Eęitim ve Ders Materyalleri: İřletim sistemleri veya bilgisayar mimarisi kurslarında, öğrencilere sayfa deęiřtirme konseptlerini anlatmak için optimal algoritma üzerinden gidilir. Öğrencilere, teorik en iyi performansın neye benzedięini göstermek ve dięer algoritmalarla karřılařtırmalar yapmalarını saęlamak için kullanılır.

3. Simölasyon ve Modelleme: Bilgisayar mühendisleri ve arařtırmacıları, bellek yönetim sistemlerinin performansını simöl etmek ve analiz etmek için optimal algoritmayı kullanabilir. Bu, potansiyel geliřtirmeleri deęerlendirmek ve farklı senaryolar altında sistem davranıřını anlamak için önemlidir.

3 – Sanal Bellek Hiyerarřisi:

-Bellek hiyerarřisinin nasıl iřledięini ve ana bellek (RAM), ikincil bellek (disk), sayfa tablosu gibi kavramların birbirleriyle nasıl etkileřimde bulunduęunu açıklayınız.

Öncelikle ilgili soruda geęen kavramlar ve bilgisayar donanımları nedir:

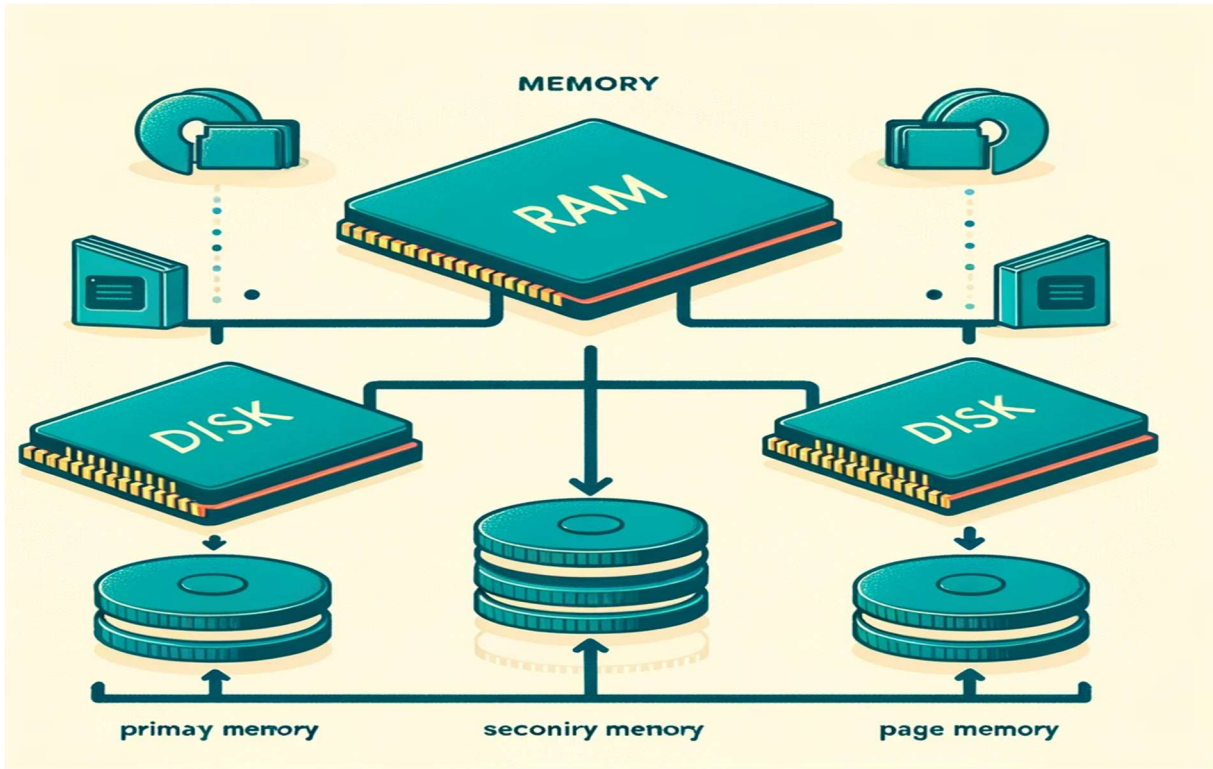
RAM (Random Acces Memory) Rastegele Eriřimli Bellek: RAM bilgisayar biliminde, bilgisayarın hızlı eriřim için geęici verileri ve program kodlarını sakladığı bir hafıza türüdür. Veriler, iřlemci tarafından hızla eriřilebilir ve deęiřtirilebilir, bu da bilgisayarın iřlemleri hızlı ve etkin bir řekilde geręekleřtirmesini saęlar. RAM, bilgisayarın açık olduęu sürece verileri tutar, ancak güç kesildiğinde veriler kaybolur (geęici hafıza). RAM miktarı, bir bilgisayarın aynı anda ne kadar çok iřlem yapabileceęini ve ne kadar hızlı çalışabileceęini doğrudan etkiler. Bilgisayar biliminde RAM, iřletim sistemlerinin ve uygulamaların performansı üzerinde önemli bir rol oynar.

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 [19].

Sayfa Tablosu: Sayfa tablosu, işletim sistemlerinde sanal bellek yönetiminde kullanılan bir veri yapısıdır ve her program için sanal adreslerin fiziksel bellek adreslerine dönüştürülmesini sağlar. En önemli özelliklerinden biri, sanal adres alanını fiziksel bellek alanına eşlemesidir. Ancak, bazı durumlarda sayfa tablosuyla ilgili hatalar ve sorunlar ortaya çıkabilir. Bunlar arasında sayfa hataları, bellek taşması ve performans düşüklüğü yer alır. Sayfa hataları, istenen verinin fiziksel bellekte bulunmaması durumunda meydana gelir ve işletim sistemi bu sayfayı ikincil depolama biriminden (genellikle bir sabit diskten) yüklemek zorunda kalır. Bu işlem, sistem performansını düşürebilir ve bellek yönetimi karmaşıklığını artırabilir. Sayfa tablosunun boyutu ve yönetimi, bellek verimliliği üzerinde önemli bir etkiye sahiptir ve bu, özellikle kısıtlı fiziksel bellek kapasitesine sahip sistemlerde dikkate alınmalıdır.

Donanımlar ve kavramlar arasındaki ilişki ve çalışma prensibi nedir: Ana bellek (RAM), ikincil bellek (disk) ve sayfa tablosu, bilgisayar sistemlerindeki bellek hiyerarşisinin temel bileşenleridir. RAM, işlemcinin hızlı erişim sağladığı, programların ve işletim sisteminin aktif olarak kullandığı geçici depolama alanıdır. İkincil bellek, daha yavaş erişim süresine sahip olmakla birlikte, verileri kalıcı olarak saklar ve genellikle daha büyük kapasiteli bir depolama alanı sunar. Sayfa tablosu, sanal bellek yönetiminin bir parçası olarak çalışır ve sanal adresleri RAM'deki fiziksel adreslere eşler. Bir programın ihtiyaç duyduğu veri RAM'de yoksa, sayfa hatası oluşur ve işletim sistemi bu veriyi ikincil bellekten RAM'e yükler. Bu süreç, bilgisayarın verimli çalışması için kritik öneme sahiptir ve bellek yönetimi stratejilerinin temelini oluşturur.

Bu üç bileşenin etkileşimi, sistem performansı ve veri erişimi açısından son derece önemlidir.



Şekil 8: 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, modern işletim sistemlerinin sanal bellek yönetiminde kritik bir rol oynar. Sanal bellek, fiziksel bellek (RAM) ile ikincil depolama arasında bir soyutlama katmanı oluşturarak programların daha fazla belleğe erişiyormuş gibi çalışmasını sağlar. Sayfa tablosu, bu sanal bellek adreslerinin fiziksel bellek adreslerine nasıl eşlendiğini belirler.

Sayfa Tablosu Nasıl Çalışır: Bir program veya işlem çalıştırıldığında, işlemci sanal adresler kullanır. Bu sanal adresler, sayfa numarası ve offset olmak üzere iki parçadan oluşur. Sayfa tablosu, bu sanal sayfa numarasını fiziksel bellekteki bir sayfa çerçevesine eşler. İşlemci bir veriye erişmek istediğinde, sayfa tablosu sayesinde bu

sanal adres fiziksel adrese dönüştürülür. Eğer sayfa tablosunda eşleşen bir giriş bulunmazsa, bu durum bir sayfa hatası olarak adlandırılır ve işletim sistemi ilgili sayfayı diskten RAM'e yükler.

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: Basit ve doğrudan bir eşleme sağlar; küçük ve orta ölçekli bellek adresleme ihtiyaçları için uygundur.

Dezavantaj: Büyük sanal adres uzayları için hafıza tüketimi yüksek olabilir; büyük sistemlerde verimsizdir.

2. Çok Düzeyli Sayfa Tablosu:

Avantaj: Geniş sanal adres uzaylarını destekler; bellek kullanımını optimize eder ve daha esnek bir yapı sunar.

Dezavantaj: Adres çevirme süreci daha karmaşık ve zaman alıcıdır; ek hesaplama maliyeti getirir.

3. Tersine Çevrilmiş Sayfa Tablosu:

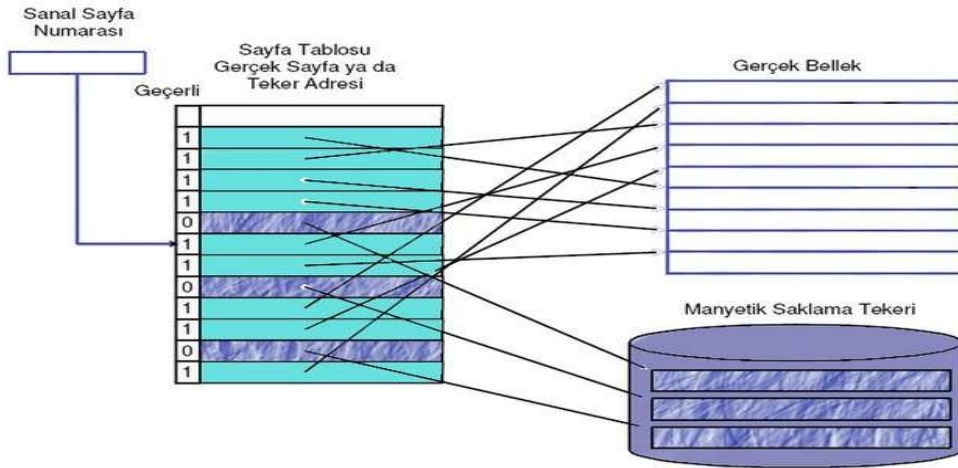
Avantaj: Büyük fiziksel belleklerde hafıza tüketimini azaltır; tek bir global tablo kullanarak yönetimi basitleştirir.

Dezavantaj: Sanal sayfa numarasını bulmak için daha fazla hesaplama gerektirir; büyük sanal adres uzaylarında arama süresi artabilir.

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 9: 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 Fragmentasyonu:

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

Bellek Fragmentasyonu Nedir:

Bellek fragmentasyonu, bilgisayar sistemlerinde kullanılan belleğin zamanla verimsiz bir şekilde kullanılması durumudur. Bu, bellekte boş alanların dağınık bir şekilde oluşmasına ve sürekli olarak kullanılan belleğin verimliliğinin azalmasına neden olur. Bellek fragmentasyonu, genellikle iki türde ele alınır: iç fragmentasyon ve dış fragmentasyon.

İç ve Dış Fragmentasyon:

1. İç Fragmentasyon (Internal Fragmentation): İç fragmentasyon, bir bellek bloğunun tamamının bir süreç veya veri tarafından ayrıldığı, ancak kullanılmayan bir kısmının boşta kaldığı durumdur. Bu, özellikle sabit boyutlu bellek bölümlendirmesi (partitioning) kullanıldığında meydana gelir. Örneğin, bir sürecin 10 KB bellek ihtiyacı varsa ve en yakın bellek bölümü boyutu 12 KB

ise, 2 KB boşta kalır ve başka bir süreç tarafından kullanılamaz. Bu, bellek kullanımının verimsizliğine yol açar.

2. Dış Fragmantasyon (External Fragmentation): Dış fragmantasyon, kullanılmayan bellek alanlarının birçok küçük parçada dağılması durumudur. Bu, dinamik bellek bölümlendirmesi kullanıldığında meydana gelir. Birçok sürecin yaratılıp kaldırılması sonucu bellekte sürekli olarak değişen boyutlarda boş alanlar oluşur. Zamanla, bu boş alanlar yeterli boyutta olmayabilir ve yeni süreçler için yeterli sürekli bellek alanı bulunamayabilir, bu da sistem performansının düşmesine neden olur.

Sanal Bellek Yönetiminin Bellek Fragmantasyonu ile Başa Çıkma Yöntemleri:

1. Sayfalama (Pagination):

Sayfalama, belleği sabit boyutlu birimler olan sayfalara böler ve bu sayfaları sanal bellek alanında depolar. Her sayfa fiziksel bellekte rastgele bir konumda saklanabilir, böylece dış fragmantasyon sorununu büyük ölçüde azaltır. Sayfalama, fiziksel belleğin kesintisiz bir bölümünü gerektirmez ve mevcut boş sayfa çerçevelerine verileri yerleştirebilir. Ancak, sayfalama bazen küçük miktarlarda iç fragmantasyona neden olabilir, çünkü her sayfanın tamamı her zaman kullanılmayabilir.

2. Segmentasyon (Segmentation):

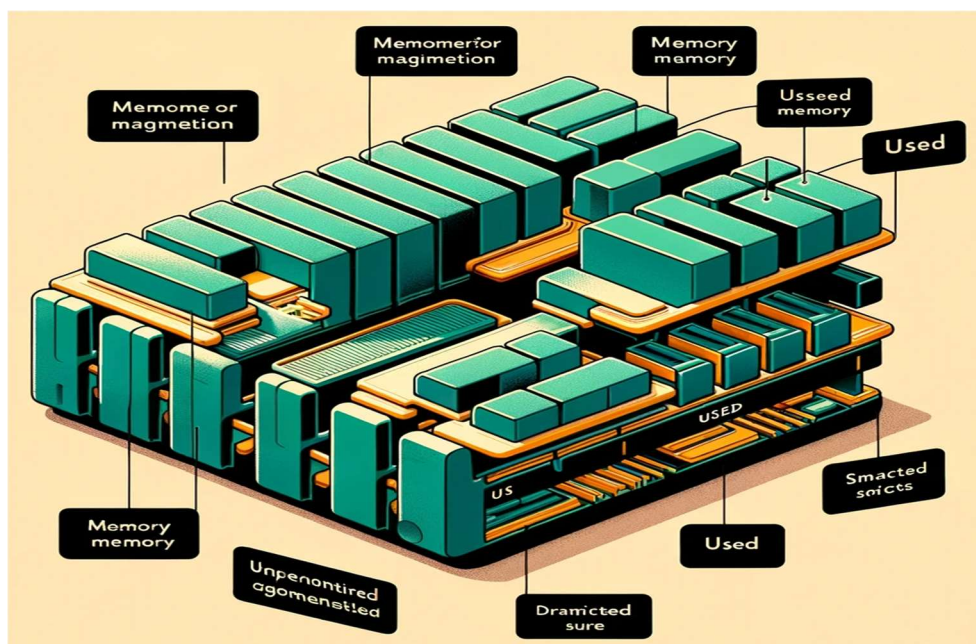
Segmentasyon, belleği anlamsal birimler olan segmentlere böler (örneğin, programın kodu, verileri, yığınlı alanı). Her segment, bellekte farklı boyutlara sahip olabilir. Segmentasyon, programın farklı bölümlerinin ihtiyaç duyduğu bellek miktarına göre dinamik olarak bellek ayırır. Bu yöntem, dış fragmantasyonu azaltır, ancak büyük segmentler için yeterli sürekli alanın bulunamaması durumunda iç fragmantasyon sorunlarına yol açabilir.,

3. Sanal Bellek Kullanımı (Virtual Memory Usage):

Sanal bellek, fiziksel RAM'in yetmediği durumlarda bilgisayarın ikincil depolama cihazlarını (genellikle sabit disk veya SSD) geçici bellek olarak kullanmasını sağlar. Bu, işletim sistemine ve çalışan uygulamalara daha fazla bellekmiş gibi görünmesine olanak tanır. Sanal bellek, bellek ihtiyacını karşılamak için diskte bir alan (sayfa dosyası veya swap dosyası) oluşturur.

4. Sayfalama ve Segmentasyonun Kombinasyonu (Combination of Paging and Segmentation):

Bazı işletim sistemleri, sayfalama ve segmentasyon tekniklerini birleştirerek her ikisinin avantajlarından faydalanır. Bu yaklaşımda, her segment kendi içinde sayfalara bölünür. Bu, segmentasyonun sağladığı anlamsal yönetim avantajları ile sayfalamadan gelen esneklik ve verimlilik avantajlarını bir araya getirir. Bu teknik, segmentlerin anlamsal gruplamasını korurken, sayfalamanın getirdiği rastgele erişim ve dış fragmentasyonla başa çıkma yeteneğini de içerir. Bu yaklaşımın temel avantajı, her segmentin ayrı sayfa tablosu kullanarak fiziksel bellekte istenilen yerlere serpiştirilebilmesidir. Bu, dış fragmentasyon sorununu önemli ölçüde azaltır. Ancak, bu yöntem yönetim karmaşıklığını ve bellek kullanımını artırabilir, çünkü her segment için ayrı bir sayfa tablosu tutulması gerekmektedir. Ayrıca, sayfa ve segment tablolarının yönetimi ek hesaplama kaynakları gerektirebilir.



Sekil 10: Bellek fragmantasyonunu anltan bir görsel.

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

6 – Sanal Bellek Yönetimi Uygulama:

- İstedığınız 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 işletim sistemi, sayfalama mekanizması kullanarak her uygulamanın kendi sanal adres alanını yaratır ve bu alanı fiziksel belleğe eşler. Bu yöntem, sistem kaynaklarının etkili kullanımını sağlar ve uygulamalar arasında bellek izolasyonu oluşturur.

1. Sanal Bellek Yönetimi Süreci:

1. Sayfa Dosyası Oluşturma: Windows, fiziksel belleğin yetmediği durumlarda kullanmak üzere sabit disk üzerinde bir sayfa dosyası (pagefile.sys) oluşturur. Bu dosya, genellikle sistem sürücüsünde yer alır ve kullanıcı tarafından boyutlandırılabilir.

2. Sayfa Hataları ve Taşıma: Bir uygulama, hafızada bulunmayan bir veriye erişmeye çalıştığında, sistem bir sayfa hatası oluşturur ve gerekli veriyi sayfa dosyasından veya uygulamanın kendisinden RAM'e taşır.

3. Bellek Yönetimi ve Sayfa Değiştirme: Windows, kullanılmayan sayfaları sayfa dosyasına aktararak RAM'de alan açar. Bu işlem, mevcut belleğin daha verimli kullanılmasını sağlar.

2. Uygulama Sırasında Ortaya Çıkabilecek Durumlar:

1. Yetersiz Sanal Bellek (Insufficient Virtual Memory): Sayfa dosyasının ve RAM'in yeterli olmaması durumunda, sistem performansı düşebilir ve sistem, sayfa dosyasının genişletilmesi için uyarı verebilir.

2. Aşırı Sayfa Değiştirme (Thrashing): Eğer sistem sürekli olarak bellek ve disk arasında veri taşıyorsa, bu thrashing olarak adlandırılır ve sistem performansını olumsuz etkiler.

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.

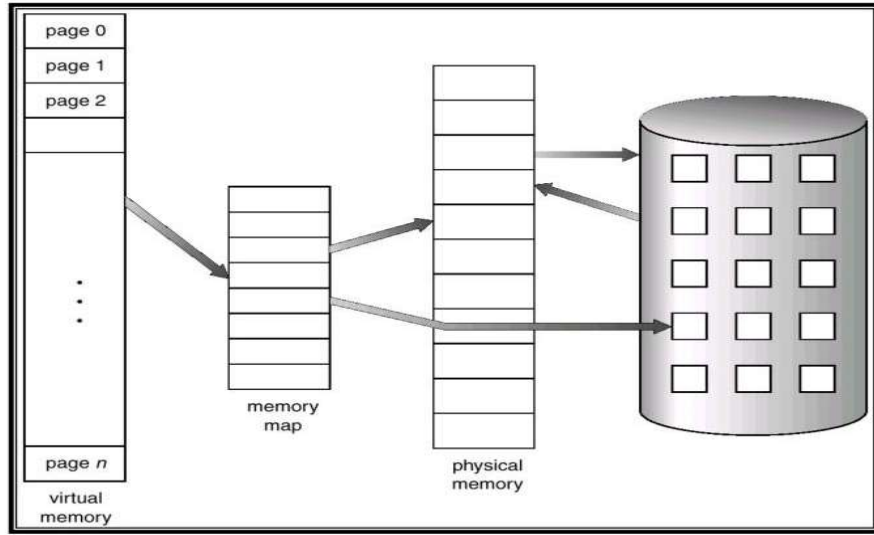
3.Sistem İzleme Araçları: Windows Görev Yöneticisi ve Performans İzleyicisi gibi araçlar, bellek kullanımını izlemek ve potansiyel sorunları tespit etmek için kullanılabilir.

4. Ek Bilgiler ve Yöntemler:

1. Sanal Bellek Özelleştirmesi: Gelişmiş kullanıcılar, performansı optimize etmek için sanal bellek ayarlarını özelleştirebilirler.

2. Disk Türünün Önemi: SSD'ler HDD'lere göre daha hızlı erişim süreleri sunduğundan, sayfa dosyasının bir SSD üzerinde bulunması performansı iyileştirebilir.

3. Bellek Yönetimi Algoritmaları: Windows, farklı bellek yönetimi algoritmaları kullanarak, uygulamaların bellek ihtiyaçlarını dinamik olarak karşılar ve sistem kaynaklarını verimli bir şekilde yönetir.



Şekil 11: Sanal bellek yönetimi örnek şeması.

KAYNAKÇA:

- [1] M. Cai, S. Liu, H. Huang, tscale: A contention-aware multithreaded framework for multicore multiprocessor systems, in: Proc. International Conference on Parallel and Distributed Systems, 2017.
- [2] A.T. Clements, M.F. Kaashoek, N. Zeldovich, Scalable address spaces using RCU balanced trees, in: Proc. International Conference on Architectural Support for Programming Languages and Operating Systems, 2012.
- [3] Y. Cui, Y. Wang, Y. Chen, Y. Shi, Requester-based spin lock: a scalable and energy efficient locking scheme on multicore systems, IEEE Transactions on Computers 64 (1) (2015) 166–179.
- [4] A.T. Clements, M.F. Kaashoek, N. Zeldovich, Radixvm: scalable address spaces for multithreaded applications, in: Proc. Eurosys Conference on Computer Systems, 2013.
- [5] S. Boyd-Wickizer, A.T. Clements, Y. Mao, A. Pesterev, M.F. Kaashoek, R.T. Morris, N. Zeldovich, An analysis of linux scalability to many cores, in: Proc. USENIX Symposium on Operating Systems Design and Implementation, 2010.
- [6] X. Song, H. Chen, R. Chen, Y. Wang, B. Zang, A case for scaling applications to many core with OS clustering, in: Proc. European conference on Computer systems, 2011.
- [7] R. Liu, H. Zhang, H. Chen, Scalable read-mostly synchronization using passive reader-writer locks, in: Proc. USENIX Annual Technical Conference, 2014.
- [8] S. Boyd-Wickizer, M.F. Kaashoek, R. Morris, N. Zeldovich, Non-scalable locks are dangerous, in: Proceedings of the Linux Symposium, 2012, pp. 119–130.
- [9] F.X. Lin, X. Liu, memif: Towards programming heterogeneous memory asynchronously, in: Proc. International Conference on Architectural Support for Programming Languages and Operating Systems, 2016.
- [10] E. Jeong, S. Woo, M. Jamshed, H. Jeong, S. Ihm, D. Han, K. Park, mTCP: a highly scalable user-level TCP stack for multicore systems, Proc. USENIX Symposium on Networked Systems Design and Implementation, 2014.

- [11] Đnternet: TBD Kamu Bilgi Đşlem Merkezleri Yöneticileri Birliđi (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).
- [12] Lunsford, D. L., “Virtualization Technologies in Information Systems Education”, Journal of Information Systems Education (JISE), Volume 20, Article 3, 339-348, (2009)
- [13] 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.
- [14] <https://www.codecademy.com/resources/blog/virtual-memory/>
- [15] https://askleo.com/what_is_virtual_memory/
- [16] <https://ieeexplore.ieee.org/document/10197098>
- [17] Edward G. Coffman, Jr. and Peter J. Denning Operating Systems Theory. Prentice-Hall, 1973.
- [18] <https://bilgisayarkavramlari.com/2009/05/29/sayfa-degistirme-algoritmasi-page-replacement/#4>
- [19] <https://web.archive.org/web/20201025165312/https://www.hurriyet.com.tr/teknoloji/ssd-nedir-nasil-takilir-ve-calisir-41408909>