SAYFALAMA (PAGİNG), ÖLÜMCÜL KİLİTLENME(DEADLOCK), SEMAFOR

HAZIRLAYAN

ADI SOYADI: KÜBRA DEMİR

ÖĞRENCİ NUMARASI: 190303070

TESLİM TARİHİ: 20/12/2021

DERS ADI: İŞLETİM SİSTEMLERİ

DERS YÜRÜTÜCÜSÜ: IŞIL KARABEY AKSAKALLI

BÖLÜM I: SAYFALAMA

1. Bir sayfa değiştirme (page replacement) algoritmasından beklenen iş, sayfa hatalarının sayısını en aza indirmektir. Bu indirgeme, yoğun olarak kullanılan sayfaları tüm belleğe eşit olarak dağıtarak yapılabilmektedir. Her sayfa çerçevesiyle o çerçeveyle ilişkili sayfa sayısının bir sayacını ilişkilendirebiliriz. Daha sonra bir sayfayı değiştirmek için en küçük sayaca sahip sayfa çerçevesini ararız.

a) Bu temel fikri kullanarak bir sayfa değiştirme algoritması tanımlayınız. Bu algoritmanın durumlarını tespit etmek için sayaçların başlangıç değerini, ne zaman arttırıldığını, ne zaman azaltıldığını ve değiştirilecek sayfanın nasıl bulunduğunu yorumlayınız.

Page replecement fikrini kullanarak yazılan bu algoritma için ;

1) öncelikle counter için 0 degeri atanır yani sayaç 0 dan başlanmalıdır

2) Algoritmanın getirdiklerine bakarak page replecement de bir counter frame ve page in aktivasyonu ile counter arttırılır

3) Frame ve page in aktivasyonunun bitirilmesi ile counter azaltılı

4) Değiştirilecek sayfa counter ın en küçük olduğu değere göre yapılır

b) Dört adet sayfa çerçevesine sahip bir hafızada aşağıdaki referans dizesine algoritmanız uygulandığında kaç sayfa hatası oluşmaktadır? Hesaplayınız ve çerçeveler üzerinde adım adım gösteriniz

Page replecement e göre FİFO üzerinden işlem yapılmak istediğinde yeni sayfalar getirildikçe kuyruğun sonuna eklenirler ve sıranın başındaki sayfa bir sonraki victim durumuna geçer.Yani first in first out a göre ilk gelen ilk çıkar

1,2,3,4,5,3,4,1,6,7,8,7,8,9,7,8,9,5,4,5,4,2

	Title 2	Title 3	Title 1	Title 2	Title 3	Title 1	Title 2	Title 3	Title 1	Title 2	Title 3	Title 1	Title 2	Title 3
frame1	1	1	1	1	5	5	5	5	5	5	8	8	8	8
frame2		2	2	2	2	1	1	1	1	1	1	5	5	5
frame3			3	3	3	3	6	6	8	9	9	9	9	9
frame4				4	4	4	4	7	7	7	7	7	4	2

- İlk dört page in framelere yerleşmesinden sonra 5 için frameleride ilk giren page yani 1 victim seçilerek yerine yazılır.Daha sonra gelen 3 ve 4 page fault oluşturmadığından bir sonrakine yani 1 page ine geçilir
- 1 page 5 ten sonra ilk giren olan 2 nin yerine yazılır.Bu şekilde 1 den sonra gelen 6 ve 7 2 den sonraki 3 ve 4 framelerine yazılır.

- Bu işlemler kalan sayılar içi de aynı şekilde devam eder ve page fault olduğu durumlarda ilk gelen victim seçilerek gelen sayıyla değiştirilir
- Toplamda 14 page fault vardır.Bunlar 1,2,3,4,5,1,6,7,8,9,8,5,4,2 pageleri olarak belirtilebilir

c) Dört adet sayfa çerçevesi için b bölümünde verilen referans dizesinde en uygun sayfa değiştirme stratejisini kullanarak minimum sayfa hatası sayısını bulunuz ve sayfa hatasının diğer yöntemlere göre neden daha az olduğunu yorumlayınız

Page replecement in bazı page fault çözüm yöntemleri vardır.Bunlar;

FIFO : Basit ve açık bir sayfa değiştirme stratejisi FIFO'dur , yani ilk giren ilk çıkar.

- Yeni sayfalar getirildikçe kuyruğun sonuna eklenirler ve sıranın başındaki sayfa bir sonraki kurban olur.
- FIFO ile ortaya çıkabilecek ilginç bir etki, Belady'nin anormalliğidir ; burada mevcut çerçeve sayısının arttırılması , gerçekte meydana gelen sayfa hatalarının sayısını artırabilir
- fıfo da page fault ve frame ler arasında doğrusal bir durum yoktur

OPTİMAL(OPT) : Gelecekte en uzun süre kullanılmayacak sayfayı değiştirin.

- Referans stringinin bilinmesi gerekir bu yüzden birazcık sezgiseldir ve karşılaştırma amacıyla kullanılır
- Belady'nin anormalliğinin keşfi, olası tüm sayfa hatalarının en düşüğünü veren ve Belady'nin anormalliğinden etkilenmeyen algoritmadır

LRU : Geçmişteki bilgilere göre page, framelere yerleştiriliyor.

- Algoritma en uzun sürede kullanılmamış sayfa yakın gelecekte tekrar kullanılmayacak biri olmasıdır.
- FIFO ve LRU arasındaki farka dikkat edin: Birincisi en eski yükleme süresine, ikincisi en eski kullanım süresine bakar .
- Optimumda ileridekileri yerleştirmede en uzun kullanılana bakılırken burada geçmişe bakılır
- LRU, iyi bir değiştirme politikası olarak kabul edilir ve sıklıkla kullanılır. Sorun, tam olarak nasıl uygulanacağıdır. Yaygın olarak kullanılan iki basit yaklaşım vardır:
    - Counter : Her bellek erişimi bir sayacı artırır ve bu sayacın geçerli değeri o sayfanın sayfa tablosu girişinde saklanır. Ardından, LRU sayfasını bulmak, en küçük sayaç değerine sahip sayfa için basit bir tablo aramasını içerir. Sayacın taşması dikkate alınmalıdır.
    - Stack : Diğer bir yaklaşımda ise yığın kullanmaktır ve bir sayfaya erişildiğinde o sayfayı yığının ortasından çekip en üste yerleştirmektir. LRU sayfası her zaman yığının altında olacaktır. Bu, nesnelerin yığının ortasından kaldırılmasını gerektirdiğinden, önerilen veri yapısı çift bağlantılı bir listedir.

LRU'nun her iki uygulamasının, bu işlemlerin her bellek erişimi için gerçekleştirilmesi gerektiğinden, sayacı artırmak veya yığını yönetmek için donanım desteği gerektirir

Ne LRU ne de OPT, Belady'nin anomalisini göstermez. Her ikisi de , Belady'nin anormalliğini asla gösteremeyen yığın algoritmaları adı verilen bir sayfa değiştirme algoritmaları sınıfına aittir . Yığın algoritması, N boyutundaki bir çerçeve kümesi için bellekte tutulan sayfaların her zaman N+1 çerçeve boyutu için tutulan sayfaların bir alt kümesi olacağı bir algoritmadır. LRU durumunda (ve özellikle bunun yığın uygulaması). yığının en üstteki N sayfası, N veya daha büyük olan tüm çerçeve kümesi boyutları için aynı olacaktır.

Bu bilgiler ışığında 1,2,3,4,5,3,4,1,6,7,8,7,8,9,7,8,9,5,4,5,4,2 page leri en az page fault oluşumu için OPT yöntemini kullanabilir

*Optimalda baştan bakarak uzun süre kullanılmayanın yerine yazılıyor LRU da ise başta sonra ortada farketmeksizin page fault oluşturmayanlar haricinde yerleştirme yapılıyor

*LRU FİFODAN daha iyi ama OPT den daha kötüdür.

*LRU nun OPT den kötü olma durumu LRU zor bir algoritma olmasıdır. LRU için bir bağlantılı liste gerekir bu liste her hafıza erişiminde güncellenmelidir. Ayrıca böyle bir listede sayfaları bulmak çok zaman alır.

	Title 2	Title 3	Title 1	Title 2	Title 3	Title 1	Title 2	Title 3	Title 1	Title 2	Title 3
frame1	1	1	1	1	1	6	6	8	8	8	8
frame2		2	2	2	5	5	5	5	5	5	2
frame3			3	3	3	3	7	7	7	4	4
frame4				4	4	4	4	4	9	9	9

- OPT de frame i 4 olan pageleri yerleştirirken uzu süre kullanılmayacak olan ele alınır. Daha sonra yukarıda da belirtildiği gibi adım adım yerleştirme yapılır
- Verilen pagelerin framelere yerleşmesinden sonra toplamda 11 page fault oluşur. Bunlar 1,2,3,4,5,6,7,8,9,4,2 dir

BÖLÜM II: READERS-WRİTERS PROBLEMİ – DİNİNG PHİLOSOPHERS' PROBLEMİ

1. Reader-writer problemini herhangi bir programlama dilinde kodlayarak kritik kesime aynı anda birçok reader erişebilirken writer proseslerin erişmediği ve tam tersi durumu gösteren çıktıyı belirtiniz.

```
#okuma ve yazma problemıdir.Aynı anda birden fazla kişinin okuması sağlanır
#fakat dosya yazıldığı sırada okuma işlemi olmaz ve bu problem doğar.problem yazma ve okumaya öncelik sırası verilir
#thread kutuphanesi import edilir
import threading as thread
import random
global x   #reader writer islemi sırasında erisilmesi gereken veri burada tutuluyor
x = 0
#reader writer probleminin çozumu icin lock degiskeni tanımlandı
lock = thread.Lock()
#olusturdugumuz global degiskeni senkronize olarak okumak icin reader fonksiyonunu kullanırız
def Reader():
    global x
    print('Reader okuyor')
    #bu global degiskene erismeden önce bir lock aldıgımızdan emin oluruz
    #kilidi aldıktan sonra, global değişkene erişebilir veya okuyabilir ve ne yapmak istiyorsak onu yapabiliriz
    #bunun yapılmasının sebebi global degiskeni calıstırırken kullanılan threadlerin dısında baska bir threadın erismesini
    # onlemek istiyoruz
    lock.acquire()
    print('Paylasilan veri:', x)
    #global degisken alındıktan sonra okuma amaclı edinilmesi gereken kilitte alınır
    lock.release()
    print()
#olusturdugumuz global degiskeni okudugumuz gibi yazmak icinde writer fonksiyonunu kullaniriz
def Writer():
    global x
    print('Writer yaziyor')
    #reader da oldugu gibi burada da aynı sekilde global degiskene erismek icin bir kilit alınır
    lock.acquire()
    #daha sonra degiskene yazılır ve sayac gorevi goren x degiskeni 1 arttırılır
    #global degisken memory e yazılır
    x += 1
    print('Writer kilidi serbest birakir')
```

```
    print()
#olusturdugumuz global degiskeni okudugumuz gibi yazmak icinde writer fonksiyonunu kullaniriz
def Writer():
    global x
    print('Writer yaziyor')
    #reader da oldugu gibi burada da aynı sekilde global degiskene erismek icin bir kilit alınır
    lock.acquire()
    #daha sonra degiskene yazılır ve sayac gorevi goren x degiskeni 1 arttırılır
    #global degisken memory e yazılır
    x += 1
    print('Writer kilidi serbest birakir')
    #yazdiktan sonra diger threadin degiskene erisebilmesi icin kilidi serbest birakiriz
    lock.release()
    print()
#senkronize bir sekilde reader-writer fonksiyonuna ulasabilecek iki thread olusturuldu
if __name__ == '__main__':
    for i in range(0, 10):
        #0 ile 100 arasında rastgele bir sayi uretilir
        randomNumber = random.randint(0, 100)
        #gelen sayinin 50 den buyuk olmasi durumuna gore if else gerekli islem yapilir
        if(randomNumber > 50):
            Thread1 = thread.Thread(target = Reader)
            Thread1.start()
        else:
            Thread2 = thread.Thread(target = Writer)
            Thread2.start()
Thread1.join()
Thread2.join()
# print(x)
```

2. Yemek yiyen filozoflar problemini herhangi bir programlama dili ile kodlayınız. Problemde ölümcül kilitlenmeyi önlemek için semaforları kullanınız

```python
#Bir yuvarlak masa etrafında bes tane filozof oturmuş masadada pirinc var ve o pirinci yiyecekler hopsticklerden aynı anda iki tanesi elimizde
# olmalı ki o pirinci yiyebilelim.Her filozofun sağında ve solunda bir tane cubuk var yani bes filozofa beş çubuk var
#sağ ve soldaki chopstickleri alıp pirinci yedikten sonra bırakmayı bes filozof için yazdığımız zaman herkes aynı anda aldıgında sağ taraftan
# başladıysa soldaki diğer filozofa gidecek yani aslında hiçbiri sağ ve sol chopstigi almış olmayacak,kimsenin iki chopstigi olmamıs olacak.
# Olmadığı içinde pirinc yenmeyecek.Sonuc olarak kilitlenecek yani deadlock durumuna gececek bunun çözümü çalışabilecek process sayısını
# sınırlamaktır yani beş filozof için sınır dörttür
#thread kutuphanesi import edildi
import threading
import random
import time
#zaman alindi ve random olarak filozoflar atanir
class Philosopher(threading.Thread):
    #filozof adinda bir class tanimlanir, thread e entegre edilir ve true oldugu surece gerekli islemler yapilir
    running = True
    #subclasslar constructoru gecersiz kildiği icin threadlerde islem ypilmadan önce base class constructoru cagrilir
    def __init__(self, index, forkOnLeft, forkOnRight):
        threading.Thread.__init__(self)
        self.index = index
        self.forkOnLeft = forkOnLeft
        self.forkOnRight = forkOnRight
    def run(self):
        while(self.running):
        #filozoflarin dusunme kismidir temelde halen uyuyor olurlar
            time.sleep(30)
            #filozofuna gore random olarak getirilenler ac olarak belirtilir ve doymasi saglanir
            print ('Filozof %s ac' % self.index)
            self.dine()
    def dine(self):
        #masadaki her iki chopstickte bossa o zaman ac olan filozof yemegini yer
        fork1, fork2 = self.forkOnLeft, self.forkOnRight
        while self.running:
            #fork1 yani sol taraftaki filozofta bekleme islemi
```

```python
            #fork1 yani sol taraftaki filozofta bekleme islemi
            fork1.acquire()
            locked = fork2.acquire(False)
            #eger sag chopstick yoksa sola bakilir
            if locked: break
            fork1.release()
            print ('Filozof %s degisimi yapilir ' % self.index)
            fork1, fork2 = fork2, fork1
        else:
            return
        self.dining()
        #yemekten sonra her iki chopstickte birakilir
        fork2.release()
        fork1.release()
    def dining(self):
        print ('Filozof %s yemeye baslar'% self.index)
        time.sleep(30)
        print ('Filozof %s yemegi bitirir ve birakir' % self.index)
def main():
    #baslangıc semafor dizisi forklar yani chopstickler
    forks = [threading.Semaphore() for n in range(5)]
    #burada kullanilan (i+1)%5 problemde de belirtilen dairesel masa formunu elde etmektir
    philosophers= [Philosopher(i, forks[i%5], forks[(i+1)%5])
            for i in range(5)]
    Philosopher.running = True
    for p in philosophers: p.start()
    time.sleep(100)
    Philosopher.running = False
    print ("Tamamlandi")
if __name__ == "__main__":
    main()
```

KAYNAKÇA

https://en.wikipedia.org/wiki/Readers%E2%80%93writers_problem

https://www.geeksforgeeks.org/readers-writers-problem-writers-preference-solution/

https://www.geeksforgeeks.org/dining-philosophers-problem/

https://www.javatpoint.com/os-dining-philosophers-problem

https://code.activestate.com/recipes/577803-reader-writer-lock-with-priority-for-writers/

https://www.geeksforgeeks.org/program-page-replacement-algorithms-set-2-fifo/

https://tutorialspoint.dev/data-structure/queue-data-structure/program-page-replacement-algorithms-set-2-fifo

https://er.yuvayana.org/lru-page-replacement-algorithm-in-operating-system/

https://www.geeksforgeeks.org/program-for-least-recently-used-lru-page-replacement-algorithm/

https://www.geeksforgeeks.org/optimal-page-replacement-algorithm/

https://afteracademy.com/blog/what-are-the-page-replacement-algorithms