22

Fiziksel Belleğin Ötesinde: Politikalar

Bir sanal bellek yöneticisinde, çok fazla boş belleğiniz olduğunda hayat kolaydır. Bir sayfa hatası oluştuğunda, boş sayfa listesinde boş bir sayfa bulur ve hatalı sayfaya atarsınız. Hey, İşletim Sistemi, tebrikler! Yine başardın.

Ne yazık ki, az bellek boş olduğunda işler biraz daha ilginç hale geliyor. Böyle bir durumda, bu **bellek baskısı (memory pressure)** işletim sistemini aktif olarak kullanılan sayfalara yer açmak için sayfaları **sayfalamaya (paging out**) zorlar. Hangi sayfanın (veya sayfaların) **tahliye (evict)** edileceğine karar vermek, işletim sisteminin **değiştirme politikası (replacement policy)** içinde yer alır; Tarihsel olarak, eski sistemlerde çok az fiziksel bellek bulunduğundan,ilk sanal bellek sistemlerinin aldığı en önemli kararlardan biriydi. Minimal olarak, hakkında biraz daha fazla şey bilmeye değer ilginç bir politikalar dizisidir. Ve dolayısıyla bizim sorunumuz:

KRİTİK NOKTA: HANGİ SAYFANIN TAHLİYE EDİLECEĞİNE NASIL KARAR VERİLİR

İşletim sistemi hangi sayfanın (veya sayfaların) bellekten çıkarılacağına nasıl karar verebilir? Bu karar, genellikle bazı genel ilkeleri izleyen (aşağıda tartışılmıştır) ancak köşe durum davranışlarından kaçınmak için bazı ince ayarlar da içeren sistemin değiştirme politikası tarafından verilir.

22.1 Önbellek Yönetimi

Politikalara geçmeden önce, çözmeye çalıştığımız problemi daha ayrıntılı olarak tanımlayacağız. Ana belleğin sistemdeki tüm sayfaların bir alt kümesini tuttuğu düşünüldüğünde, haklı olarak sistemdeki sanal bellek sayfaları için bir **önbellek (cache)** olarak görülebilir. Bu nedenle, bu önbellek için bir değiştirme politikası seçerken amacımız **önbellek kaçırma (cache misses)** sayısını en aza indirmektir, Yani, bir sayfayı diskten getirme sayısını en aza indirmek için. Alternatif olarak, hedefimizi **önbellek isabet (cache hits)** sayısını maksimize etmek olarak görebiliriz, Yani, erişilen bir sayfanın bellekte bulunma sayısı. Önbellek vuruşlarının ve ıskalamalarının sayısını bilmek, bir program için **ortalama bellek erişim süresini (av-erage memory access time (AMAT))** hesaplamamızı sağlar (bilgisayar mimarlarının donanım önbellekleri için hesapladığı bir metrik [HP06]). Spesifik olarak, bunlar göz önüne alındığında değerlerini kullanarak bir programın AMAT değerini aşağıdaki gibi hesaplayabiliriz:

$$AMAT = T_M + (P_{Iska} \cdot T_D) \quad (22.1)$$

Burada TM belleğe erişim maliyetini, TD diske erişim maliyetini temsil eder, ve PIska veriyi önbellekte bulamama olasığı (ıskalama); PIska 0.0 ile 1.0 arasında değişir ve bazen olasılık yerine ıskalama yüzdesine atıfta bulunuruz (örneğin, %10'luk bir ıskalama oranı PIska = 0.10 anlamına gelir). Bellekteki verilere erişmenin maliyetini her zaman ödediğinizi unutmayın; Ancak kaçırdığınızda, ek olarak verileri diskten getirme maliyetini de ödemeniz gerekir. Örneğin, (küçük) bir adres alanına sahip bir makine hayal edelim: 4KB, 256 baytlık sayfalar. Dolayısıyla, bir sanal adresin iki bileşeni vardır: 4 bitlik bir VPN (en önemli bitler) ve 8 bitlik bir ofset (en önemsiz bitler). Dolayısıyla, bu örnekteki bir işlem 24 veya 16 toplam sanal sayfaya erişebilir. Bu örnekte, işlem aşağıdaki bellek referanslarını (yani sanal adresleri) oluşturur: 0x000, 0x100, 0x200, 0x300, 0x400, 0x500, 0x600, 0x700, 0x800, 0x900. Bu sanal adresler, adres alanının ilk on sayfasının her birinin ilk baytını ifade eder (sayfa numarası her sanal adresin ilk onaltılık basamağıdır).

Ayrıca sanal sayfa 3 dışındaki tüm sayfaların zaten bellekte olduğunu varsayalım. Böylece, bellek referansları dizimiz aşağıdaki davranışla karşılaşacaktır: Vur, vur, vur, ıska, vur, vur, vur, vur, vur, vur. **İsabet oranını (hit rate)** (bellekte bulunan referansların yüzdesi) hesaplayabiliriz: 90%, çünkü 10 referanstan 9'u bellektedir Dolayısıyla **ıskalama oranı (miss rate)** %10'dur (P = 0.1). Genel olarak, PVur + PISKA = 1.0; isabet oranı artı ıskalama oranı toplamı %100'dür.

AMAT'ı hesaplamak için belleğe erişim maliyetini ve diske erişim maliyetini bilmemiz gerekir. Belleğe erişim maliyetini varsayarsak (TM) 100 nanosaniye civarındadır, ve diske erişim maliyeti (TD) yaklaşık 10 milisaniyedir, aşağıdaki AMAT'a sahibiz: 100ns + 0.1 - 10ms, Bu da 100ns + 1ms veya 1.0001 ms ya da yaklaşık 1 milisaniyedir. Eğer isabet oranımız %99.9 olsaydı (Pıska = 0.001), sonuç oldukça farklıdır: AMAT 10,1 mikrosaniyedir, ya da kabaca 100 kat daha hızlı. İsabet oranı %100'e yaklaştıkça, AMAT 100 nanosaniyeye yaklaşır.

Ne yazık ki, bu örnekte de görebileceğiniz gibi, Modern sistemlerde disk erişim maliyeti o kadar yüksektir ki, küçük bir ıskalama oranı bile çalışan programların genel AMAT'ını hızla etkileyecektir. Açıkçası, mümkün olduğunca çok ıskalamadan kaçınmamız ya da yavaş çalışmamız gerekiyor, diskin hızında. Bu konuda yardımcı olmanın bir yolu, dikkatli bir şekilde akıllı bir politika geliştirmektir, şimdi yaptığımız gibi.

22.2 Optimal Değiştirme Politikası (OPT)

Belirli bir değiştirme politikasının nasıl işlediğini daha iyi anlamak için, mümkün olan en iyi değiştirme politikasıyla karşılaştırmak güzel olurdu. Görünüşe göre, Böyle bir **uygun (optimal)** politika Belady tarafından yıllar önce geliştirilmiştir [B66] (başlangıçta buna MIN adını vermiştir). Optimum değiştirme politikası, toplamda en az sayıda ıskalamaya yol açar. Belady, basit (ama ne yazık ki uygulaması zor!) bir yöntem olduğunu göstermiştir. *Gelecekte en uzak* erişilecek sayfayı değiştiren yaklaşım en uygun politikadır,Bu da mümkün olan en az önbellek ıskalamasına neden olur

IPUCU: OPTIMAL ILE KARŞILAŞTIRMAK FAYDALIDIR

Optimal olmasına rağmen gerçek bir politika olarak çok pratik değildir, simülasyon veya diğer çalışmalarda bir karşılaştırma noktası olarak son derece faydalıdır. Süslü yeni algoritmanızın %80 isabet oranına sahip olduğunu söylemek tek başına anlamlı değildir; Optimalin %82 isabet oranına ulaştığını (ve dolayısıyla yeni yaklaşımınızın optimale oldukça yakın olduğunu) söylemek sonucu daha anlamlı hale getirir ve bir bağlam kazandırır. Böylece, yaptığınız herhangi bir çalışmada, Optimal olanın ne olduğunu bilmek daha iyi bir karşılaştırma yapmanızı sağlar, hala ne kadar iyileştirmenin mümkün olduğunu ve ayrıca politikanızı daha iyi hale getirmeyi ne zaman bırakabileceğinizi gösterir, çünkü ideale yeterince yakındır [AD03].

Umarım, optimal politikanın arkasındaki sezgi mantıklıdır. Şöyle düşünün: eğer bir sayfayı atmanız gerekiyorsa, neden şu andan itibaren en çok ihtiyaç duyulanı atmayasınız? Bunu yaparak, aslında önbellekteki diğer tüm sayfaların en uzaktakinden daha önemli olduğunu söylemiş olursunuz. Bunun doğru olmasının nedeni basittir: en uzaktakine başvurmadan önce diğer sayfalara başvuracaksınız.

Optimal politikanın aldığı kararları anlamak için basit bir örnek üzerinden ilerleyelim. Bir programın aşağıdaki sanal sayfa akışına eriştiğini varsayın: 0, 1, 2, 0, 1, 3, 0, 3, 1, 2, 1. Şekil 22.1, üç sayfaya sığan bir önbellek varsayıldığında optimal davranışı göstermektedir.

Şekilde, aşağıdaki işlemleri görebilirsiniz. Şaşırtıcı olmayan bir şekilde, önbellek boş bir durumda başladığı için ilk üç erişim ıskalamadır; böyle bir ıskalama bazen **soğuk başlangıç ıskalaması (**veya **zorunlu ıskalama)((cold-start miss (**or **compulsory miss)).** olarak adlandırılır. Ardından, her ikisi de önbellekte isabet eden 0 ve 1 sayfalarına tekrar bakarız. Son olarak, başka bir ıskalamaya ulaşırız (sayfa 3'e), ancak bu sefer önbellek doludur; yeniden yerleştirme yapılmalıdır! Bu da şu soruyu akla getiriyor: Hangi sayfayı değiştirmeliyiz? Optimum politika ile, şu anda önbellekte bulunan her sayfanın (0, 1 ve 2) geleceğini inceleriz ve 0'a neredeyse hemen erişildiğini, 1'e biraz daha geç erişildiğini ve 2'ye gelecekte en uzak erişildiğini görürüz. Bu nedenle, optimum politikanın kolay bir seçimi vardır: sayfa 2'yi çıkarmak, önbellekte sayfa 0, 1 ve 3 ile sonuçlanır.

Erişim (Access)	İsabet/ıska? (Hit/Miss?)	Boşalt (Evict)	Ortaya çıkan önbellek durumu (Resulting Cache State)
0	Iska		0
1	Iska		0, 1
2	Iska		0, 1, 2
0	İsabet		0, 1, 2
1	İsabet		0, 1, 2
3	Iska	2	0, 1, 3
0	İsabet		0, 1, 3
3	İsabet		0, 1, 3
1	İsabet		0, 1, 3
2	Iska	3	0, 1, 2
1	İsabet		0, 1, 2

Şekil 22.1: **Optimal Politikanın İzlenmesi (Tracing The Optimal Policy)**

BİR KENAR:ÖNBELLEK ISKALAMA TÜRLERİ

Bilgisayar mimarisi dünyasında, mimarlar bazen ıskaları türlerine göre üç kategoriden birine ayırmayı faydalı bulurlar: zorunlu, kapasite ve çakışma ıskaları, bazen **Üç C (Three C's)** olarak adlandırılır [H87]. **Zorunlu ıskalama (compulsory miss)** (veya **soğuk başlangıç ıskalaması (cold-start miss)** [EF78]), önbellek başlangıçta boş olduğu ve bu öğeye yapılan ilk referans olduğu için gerçekleşir; Buna karşılık, **kapasite ıskalama (Capacity miss)**, önbellekte yer kalmadığı ve önbelleğe yeni bir öğe getirmek için bir öğeyi çıkarmak zorunda kaldığı için meydana gelir. Üçüncü ıskalama türü **(çakışma ıskalaması (conflict miss)),** bir öğenin donanım önbelleğinde nereye yerleştirilebileceğine ilişkin sınırlamalar nedeniyle donanımda ortaya çıkar, **Küme-birlikteliği (set-associativity)** olarak bilinen bir şey nedeniyle; işletim sistemi sayfa önbelleğinde ortaya çıkmaz çünkü bu tür önbellekler her zaman **tam ilişkilidir (fully-associative),** Yani, bir sayfanın bellekte nereye yerleştirilebileceği konusunda herhangi bir kısıtlama yoktur. Ayrıntılar için H&P'ye bakın [HP06].

Sonraki üç referans isabetlidir, ancak sonra uzun zaman önce tahliye ettiğimiz 2. sayfaya geliriz ve başka bir ıskalama yaşarız. Burada en uygun politika yine önbellekteki her sayfanın (0, 1 ve 3) geleceğini inceler, ve sayfa 1'i (erişilmek üzere olan) tahliye etmediği sürece bunu görür, iyi olacağız. Örnekte 3. sayfanın çıkarıldığı görülüyor, ancak 0 da iyi bir seçim olabilirdi. Son olarak, 1. sayfaya ulaştık ve izleme tamamlandı.

Önbellek için isabet oranını da hesaplayabiliriz: 6 isabet ve 5 ıska ile, isabet oranı $\frac{isabet}{isabet+iska}$ ki bu $\frac{6}{6+5}$ veya 54.5%.

Ayrıca hesaplayabilirsiniz isabet oranı zorunlu ıskalamaları modüle eder (yani, belirli bir sayfaya yapılan ilk ıskalamayı göz ardı eder), Bu da %85,7'lik bir isabet oranıyla sonuçlanıyor.

Ne yazık ki, daha önce programlama politikalarının geliştirilmesinde gördüğümüz gibi, gelecek genel olarak bilinmemektedir; genel amaçlı bir işletim sistemi için en uygun politikayı oluşturamazsınız[1]. Bu nedenle, gerçek ve uygulanabilir bir politika geliştirirken, hangi sayfanın boşaltılacağına karar vermek için başka bir yol bulan yaklaşımlara odaklanacağız. Dolayısıyla optimal politika, "mükemmele" ne kadar yakın olduğumuzu bilmek için yalnızca bir karşılaştırma noktası olarak hizmet edecektir.

22.3 Basit Bir Politika: FIFO

Birçok eski sistem, optimuma yaklaşmaya çalışmanın karmaşıklığından kaçınmış ve çok basit değiştirme politikaları kullanmıştır. Örneğin, bazı sistemler **FIFO** (ilk giren ilk çıkar) değişimini kullanmıştır, Burada sayfalar sisteme girdiklerinde basitçe bir kuyruğa yerleştirilir; yeniden yerleştirme gerçekleştiğinde kuyruğun sonundaki sayfa ("ilk giren" sayfa) çıkarılır. FIFO'nun büyük bir gücü vardır: uygulaması oldukça basittir.

FIFO'nun örnek referans akışımızda nasıl çalıştığını inceleyelim (Şekil 22.2, sayfa 5).

[1] Eğer yapabilirsen, bize haber ver! Birlikte zengin olabiliriz. Ya da soğuk füzyonu "keşfeden" bilim Insanları gibi, geniş çapta küçümsenir ve alay edilir [FP89].

Erişim	İsabet/Iska?	Boşalt	Ortaya çıkan önbellek durumu
0	Iska		İlk giren → 0
1	Iska		İlk giren → 0, 1
2	Iska		İlk giren → 0, 1, 2
0	İsabet		İlk giren → 0, 1, 2
1	İsabet		İlk giren → 0, 1, 2
3	Iska	0	İlk giren → 1, 2, 3
0	Iska	1	İlk giren → 2, 3, 0
3	İsabet		İlk giren → 2, 3, 0
1	Iska	2	İlk giren → 3, 0, 1
2	Iska	3	İlk giren → 0, 1, 2
1	İsabet		İlk giren → 0, 1, 2

Şekil 22.2: **FIFO İlkesinin İzlenmesi(Tracing The FIFO Policy)**

İzlemeye yine üç zorunlu ıskalama ile başlıyoruz 0, 1 ve 2 sayfalarına ve ardından hem 0 hem de 1'e basın. Ardından, sayfa 3'e başvurulur ve bir ıskalamaya neden olur; değiştirme kararı FIFO ile kolaydır: "ilk giren" sayfayı seçin (şekildeki önbellek durumu, ilk giren sayfa solda olacak şekilde FIFO sırasına göre tutulur), bu sayfa 0'dır. Ne yazık ki, bir sonraki erişimimiz sayfa 0'a olduğu için başka bir ıskalama ve değiştirmeye (sayfa 1) neden oluyor. Daha sonra 3. sayfaya ulaştık, ancak 1 ve 2'yi kaçırdık ve sonunda 1'e ulaştık.

FIFO ile optimum karşılaştırıldığında, FIFO oldukça kötü bir performans sergiliyor: %36,4 isabet oranı (veya zorunlu ıskalar hariç %57,1). FIFO basitçe blokların önemini belirleyemez: sayfa 0'a birkaç kez erişilmiş olsa bile, FIFO yine de onu dışarı atar, çünkü belleğe ilk getirilen sayfadır.

BİR KENAR: BELADY ANOMALİSİ

Belady (optimal politikanın) ve meslektaşları, biraz beklenmedik şekilde davranan ilginç bir referans akışı buldular [BNS69]. Bellek referans akışı: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5. İnceledikleri değiştirme politikası FIFO idi. İlginç olan kısım: 3 sayfalık bir önbellek boyutundan 4 sayfaya geçerken önbellek isabet oranının nasıl değiştiği.

Genel olarak, önbellek büyüdükçe önbellek isabet oranının artmasını (daha iyi hale gelmesini) beklersiniz. Ancak bu durumda, FIFO ile durum daha da kötüleşir! İsabet ve kayıpları kendiniz hesaplayın ve görün. Bu garip davranış genellikle **Belady'nin Anomalisi (Belady's Anomaly)** olarak adlandırılır (ortak yazarlarının üzüntüsüne rağmen).

LRU gibi diğer bazı politikalar bu sorundan muzdarip değildir. Nedenini tahmin edebilir misiniz? Anlaşıldığı üzere LRU, **yığın özelliği (stack property)** olarak bilinen bir özelliğe sahiptir [M+70]. Bu özelliğe sahip algoritmalar için, *N + 1* boyutundaki bir önbellek doğal olarak *N* boyutundaki bir önbelleğin içeriğini içerir. Dolayısıyla, önbellek boyutu artırıldığında isabet oranı ya aynı kalacak ya da artacaktır. FIFO ve Random (diğerlerinin yanı sıra) açıkça yığın özelliğine uymaz ve bu nedenle anormal davranışlara açıktır.

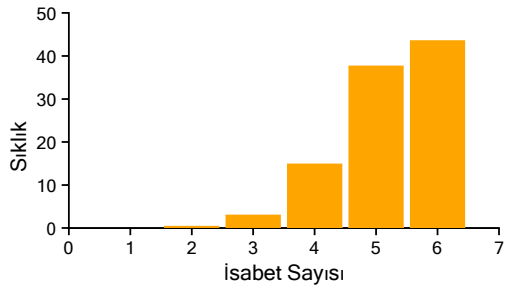
Erişim	İsabet/Iska?	Boşalt	Ortaya çıkan önbellek durumu
0	Iska		0
1	Iska		0, 1
2	Iska		0, 1, 2
0	İsabet		0, 1, 2
1	İsabet		0, 1, 2
3	Iska	0	1, 2, 3
0	Iska	1	2, 3, 0
3	İsabet		2, 3, 0
1	Iska	3	2, 0, 1
2	İsabet		2, 0, 1
1	İsabet		2, 0, 1

Şekil 22.3: **Rastgele Politikanın İzlenmesi(Tracing The Random Policy)**

22.4 Başka Bir Basit Politika: Rastgele

Benzer bir başka değiştirme politikası da, bellek baskısı altında değiştirmek için rastgele bir sayfa seçen Rastgele'dir. Rastgele, FIFO'ya benzer özelliklere sahiptir; uygulaması basittir, ancak hangi blokların tahliye edileceğini seçerken gerçekten çok akıllı olmaya çalışmaz. Random'un ünlü örnek referans akışımızda nasıl çalıştığına bakalım (bkz. Şekil 22.3).

Elbette Random'un nasıl bir performans sergileyeceği tamamen Random'un seçimlerinde ne kadar şanslı (ya da şanssız) olduğuna bağlıdır. Yukarıdaki örnekte Rastgele, FIFO'dan biraz daha iyi ve optimumdan biraz daha kötü performans gösterir. Aslında, Rastgele deneyini binlerce kez çalıştırabilir ve genel olarak nasıl olduğunu belirleyebiliriz. Şekil 22.4, Random'un her biri farklı bir rastgele tohumla yapılan 10.000 denemede kaç isabet elde ettiğini göstermektedir. Gördüğünüz gibi, bazı zamanlarda (zamanın %40'ından biraz fazlası), Rastgele örnek iz üzerinde 6 isabet elde ederek optimal kadar iyidir; bazen çok daha kötüsünü yapar, 2 veya daha az isabet elde eder. Rastgele'nin ne kadar başarılı olacağı çekiliş şansına bağlıdır.



Şekil 22.4: **Rastgele Performans 10.000 Deneme Üzerinde (Random Performance Over 10,000 Trials)**

Erişim	İsabet/Iska?	Boşalt	Ortaya çıkan önbellek durumu
0	Iska		LRU→ 0
1	Iska		LRU→ 0, 1
2	Iska		LRU→ 0, 1, 2
0	İsabet		LRU→ 1, 2, 0
1	İsabet		LRU→ 2, 0, 1
3	Iska	2	LRU→ 0, 1, 3
0	İsabet		LRU→ 1, 3, 0
3	İsabet		LRU→ 1, 0, 3
1	İsabet		LRU→ 0, 3, 1
2	Iska	0	LRU→ 3, 1, 2
1	İsabet		LRU→ 3, 2, 1

Şekil 22.5: **LRU Politikasının İzlenmesi(Tracing The LRU Policy)**

22.5 Geçmişi Kullanma: LRU

Ne yazık ki, FIFO ya da Random kadar basit bir politikanın ortak bir sorunu olması muhtemeldir: tekrar başvurulmak üzere olan önemli bir sayfayı atabilir. FIFO ilk getirilen sayfayı dışarı atar; eğer bu sayfa üzerinde önemli kod veya veri yapıları olan bir sayfaysa, yakında tekrar sayfalanacak olsa bile yine de dışarı atılır. Bu nedenle, FIFO, Rastgele ve benzeri politikaların optimuma yaklaşması muhtemel değildir; daha akıllı bir şeye ihtiyaç vardır.

Programlama politikasında yaptığımız gibi, geleceğe ilişkin tahminlerimizi geliştirmek için bir kez daha geçmişe yaslanıyor ve tarihi rehberimiz olarak kullanıyoruz. Örneğin, bir program yakın geçmişte bir sayfaya erişmişse, yakın gelecekte bu sayfaya tekrar erişmesi muhtemeldir.

Bir sayfa değiştirme politikasının kullanabileceği bir tür tarihsel bilgi **sıklıktır (frequency)**; bir sayfaya birçok kez erişilmişse, belki de bir değeri olduğu için değiştirilmemelidir. Bir sayfanın daha yaygın olarak kullanılan bir özelliği de **erişim (recency)** sıklığıdır; Bir sayfaya ne kadar yakın zamanda erişilmişse, belki de tekrar erişilme olasılığı o kadar yüksektir.

Bu politika ailesi, **insanların yerellik ilkesi (principle of locality)** [D70] olarak adlandırdıkları şeye dayanmaktadır; bu ilke temelde programlar ve davranışları hakkında bir gözlemdir. Bu ilke, oldukça basit bir şekilde, programların belirli kod dizilerine (örneğin, bir döngüde) ve veri yapılarına (örneğin, döngü tarafından erişilen bir dizi) oldukça sık erişme eğiliminde olduğunu söyler; bu nedenle, hangi sayfaların önemli olduğunu anlamak için geçmişi kullanmaya çalışmalı ve tahliye zamanı geldiğinde bu sayfaları bellekte tutmalıyız.

Ve böylece, basit tarihsel tabanlı algoritmalar ailesi doğar. **En Az Sıklıkta Kullanılan (Least-Frequently-Used) (LFU)** politikası, bir tahliye işlemi yapılması gerektiğinde en az sıklıkta kullanılan sayfanın yerini alır. Benzer şekilde, **En Son Kullanılan (Least-Recently- Used) (LRU)** ilkesi en son kullanılan sayfanın yerini alır. Bu algoritmaları hatırlamak kolaydır: bir kez adını öğrendiğinizde tam olarak ne yaptığını da bilirsiniz ki bu da bir isim için mükemmel bir özelliktir.

LRU'yu daha iyi anlamak için, LRU'nun örnek referans akışımızda

BİR KENAR: YERELLIK TÜRLERI

Programların sergileme eğiliminde olduğu iki tür yerellik vardır. Bunlardan ilki **mekansal yerellik (spatial locality)** olarak bilinir ve bir *P* sayfasına erişildiğinde, etrafındaki sayfalara da (örneğin *P - 1* veya *P + 1*) erişilmesinin muhtemel olduğunu belirtir. İkincisi, yakın geçmişte erişilen sayfaların yakın gelecekte tekrar erişilme olasılığının yüksek olduğunu belirten **zamansal yerelliktir (temporal locality)**. Bu tür yerelliklerin varlığı varsayımı, donanım sistemlerinin önbellekleme hiyerarşilerinde büyük bir rol oynamaktadır, Bu tür bir yerellik mevcut olduğunda programların hızlı çalışmasına yardımcı olmak için birçok düzeyde komut, veri ve adres çevirisi önbelleği kullanır.

Elbette **yerellik ilkesi (principle of locality)**, sıklıkla adlandırıldığı gibi, tüm programların uyması gereken katı ve hızlı bir kural değildir. Aslında, bazı programlar belleğe (veya diske) oldukça rastgele bir şekilde erişir ve erişim akışlarında çok fazla veya hiç yerellik göstermez. Dolayısıyla, yerellik herhangi bir türde (donanım veya yazılım) önbellek tasarlarken akılda tutulması gereken iyi bir şey olsa da, başarıyı garanti etmez. Daha doğrusu, bilgisayar sistemlerinin tasarımında genellikle yararlı olduğu kanıtlanan bir sezgisel yöntemdir.

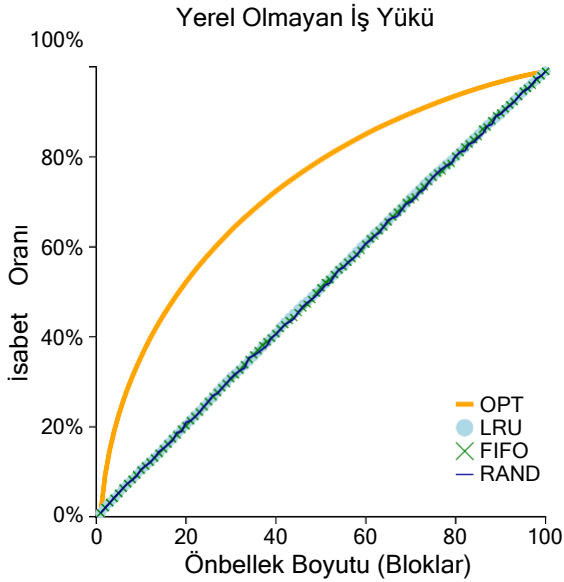
nasıl çalıştığını inceleyelim. Şekil 22.5 (sayfa 7) sonuçları göstermektedir. Şekilden, LRU'nun geçmişi kullanarak Rastgele veya FIFO gibi durum bilgisi olmayan politikalardan nasıl daha iyi sonuç verdiğini görebilirsiniz. Örnekte, LRU bir sayfayı ilk kez değiştirmesi gerektiğinde sayfa 2'yi çıkarır, çünkü 0 ve 1'e daha yakın zamanda erişilmiştir. Daha sonra sayfa 0'ın yerini alır çünkü 1 ve 3'e daha yakın zamanda erişilmiştir. Her iki durumda da LRU'nun geçmişe dayalı kararının doğru olduğu ortaya çıkar ve böylece sonraki referanslar isabet alır. Böylece, örneğimizde LRU mümkün olduğu kadar iyi yapar ve performansında[2] optimuma ulaşır. Bu algoritmaların zıtlarının da mevcut olduğunu belirtmeliyiz: **En Sık Kullanılan (Most- Frequently-Used) (MFU) ve En Yakın Zamanda Kullanılan (Most-Recently-Used) (MRU).** Çoğu durumda (hepsi değil!), bu politikalar çoğu programın sergilediği yerelliği benimsemek yerine görmezden geldiği için iyi çalışmaz.

22.6 İş Yükü Örnekleri

Bu politikalardan bazılarının nasıl davrandığını daha iyi anlamak için birkaç örneğe daha bakalım. Burada, küçük izler yerine daha karmaşık **iş yüklerini (work- loads)** inceleyeceğiz. Ancak, bu iş yükleri bile büyük ölçüde basitleştirilmiştir; daha iyi bir çalışma uygulama izlerini içerecektir.

İlk iş yükümüzde yerellik yoktur, yani her referans erişilen sayfalar kümesi içindeki rastgele bir sayfaya yapılır. Bu basit örnekte, iş yükü zaman içinde 100 benzersiz sayfaya erişir ve başvurulacak bir sonraki sayfayı rastgele seçer; toplamda 10.000 sayfaya erişilir. Deneyde, her bir politikanın önbellek boyutları aralığında nasıl davrandığını görmek için önbellek boyutunu çok küçükten (1 sayfa) tüm benzersiz sayfaları tutmaya yetecek kadar (100 sayfa) değiştiriyoruz.

[2] Tamam, sonuçları pişirdik. Ama bazen bir noktayı kanıtlamak için pişirmek gerekir.



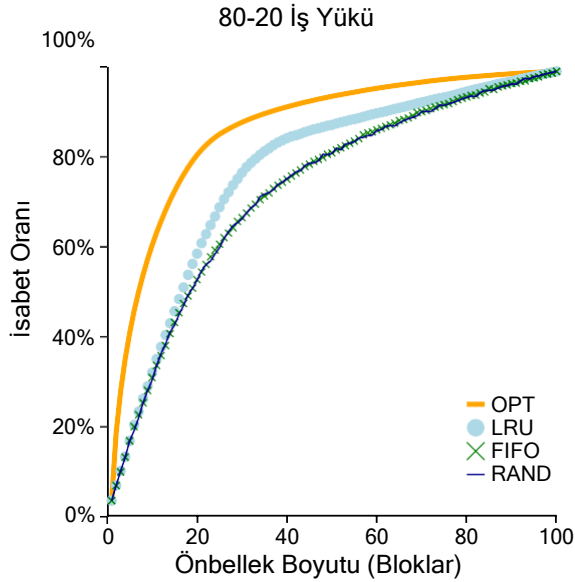
Şekil 22.6: **Yerel Olmayan İş Yükü**

Şekil 22.6 optimum, LRU, Ran- dom ve FIFO için deney sonuçlarını göstermektedir. Şeklin y ekseni her bir politikanın elde ettiği isabet oranını göstermektedir; x ekseni ise yukarıda açıklandığı gibi önbellek boyutunu değiştirmektedir.

Grafikten bir takım sonuçlar çıkarabiliriz. İlk olarak, iş yükünde yerellik olmadığında, hangi gerçekçi politikayı kullandığınız çok önemli değildir; LRU, FIFO ve Rastgele hepsi aynı performansı gösterir ve isabet oranı tam olarak önbelleğin boyutu tarafından belirlenir. İkinci olarak, önbellek tüm iş yükünü sığdıracak kadar büyük olduğunda, hangi politikayı kullandığınız da önemli değildir; tüm politikalar (Rastgele bile olsa), başvurulan tüm bloklar önbelleğe sığdığında %100 isabet oranına yakınsar. Son olarak, optimumun gerçekçi politikalardan belirgin bir şekilde daha iyi performans gösterdiğini görebilirsiniz; geleceğe bakmak, eğer mümkün olsaydı, değiştirme konusunda çok daha iyi bir iş çıkarırdı.

İncelediğimiz bir sonraki iş yükü, yerellik sergileyen "80-20" iş yükü olarak adlandırılmaktadır: Referansların %80'i sayfaların %20'sine ("sıcak" sayfalar) yapılır; referansların kalan %20'si ise sayfaların geri kalan %80'ine ("soğuk" sayfalar) yapılır. İş yükümüzde yine toplam 100 benzersiz sayfa vardır; bu nedenle, çoğu zaman "sıcak" sayfalara, geri kalanında ise "soğuk" sayfalara başvurulur. Şekil 22.7 (sayfa 10) ilkelerin bu iş yükünde nasıl performans gösterdiğini göstermektedir.

Şekilden de görebileceğiniz gibi, hem rastgele hem de FIFO oldukça iyi performans gösterirken, LRU daha iyi performans göstermektedir, çünkü sıcak sayfaları tutma olasılığı daha yüksektir; bu sayfalara geçmişte sıkça başvurulduğu için, yakın gelecekte tekrar başvurulması muhtemeldir. Optimal bir kez daha ,daha iyi performans göstererek LRU'nun geçmiş bilgilerinin mükemmel olmadığını gösteriyor.

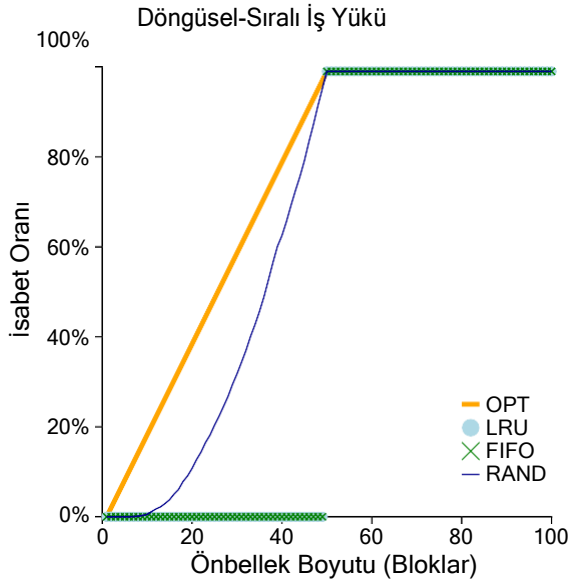


Şekil 22.7: **80-20 İş Yükü**

Şimdi merak ediyor olabilirsiniz: LRU'nun Rastgele ve FIFO'ya göre iyileştirmesi gerçekten o kadar önemli mi? Cevap, her zamanki gibi, "duruma göre değişir". Her ıskalama çok maliyetliyse (nadir değildir), isabet oranındaki küçük bir artış (ıskalama oranındaki azalma) bile performans üzerinde büyük bir fark yaratabilir. Kaçırmalar çok maliyetli değilse, LRU ile mümkün olan faydalar elbette o kadar önemli değildir.

Son bir iş yüküne daha bakalım. Buna "döngüsel sıralı" iş yükü diyoruz, çünkü bu iş yükünde 0'dan başlayarak 1, ..., 49. sayfaya kadar sırayla 50 sayfaya başvuruyoruz ve ardından 50 benzersiz sayfaya toplam 10.000 erişim için bu erişimleri tekrarlayarak döngüye giriyoruz. Şekil 22.8'deki son grafik, bu iş yükü altında politikaların davranışını göstermektedir.

Birçok uygulamada yaygın olan bu iş yükü (veritabanları gibi önemli ticari uygulamalar dahil [CD85]), hem LRU hem de FIFO için en kötü durumu temsil eder. Bu algoritmalar, döngüsel-sıralı bir iş yükü altında, eski sayfaları dışarı atar; ne yazık ki, iş yükünün döngüsel doğası nedeniyle, bu eski sayfalara, politikaların önbellekte tutmayı tercih ettiği sayfalardan daha erken erişilecektir. Gerçekten de, 49 boyutlu bir önbellekle bile, 50 sayfalık bir döngü-sıralı iş yükü %0 isabet oranıyla sonuçlanır. İlginç bir şekilde, Rastgele oldukça iyi bir performans sergiliyor, tam olarak optimuma ulaşmıyor, ancak en azından sıfır olmayan bir isabet oranı elde ediyor. Rastgelenin bazı güzel özellikleri olduğu ortaya çıktı; bu özelliklerden biri de garip köşe durum davranışlarına sahip olmaması.



Şekil 22.8: **Döngüsel İş Yükü**

22.7 Tarihsel Algoritmaların Uygulanması

Gördüğünüz gibi, LRU gibi bir algoritma genellikle önemli sayfaları atabilecek FIFO veya Random gibi daha basit politikalardan daha iyi bir iş çıkarabilir. Ne yazık ki, tarihi politikalar bize yeni bir zorluk sunuyor: bunları nasıl uygulayacağız?

Örneğin LRU'yu ele alalım. Bunu mükemmel bir şekilde uygulamak için çok fazla çalışma yapmamız gerekiyor. Özellikle, her sayfa erişiminde (yani, ister komut getirme ister yükleme ya da depolama olsun, her bellek erişiminde), bu sayfayı listenin önüne (yani MRU tarafına) taşımak için bazı veri yapılarını güncellememiz gerekir. Bunu, FIFO sayfa listesine yalnızca bir sayfa tahliye edildiğinde (ilk giren sayfayı kaldırarak) veya listeye yeni bir sayfa eklendiğinde (son giren tarafa) erişildiği FIFO ile karşılaştırın. Hangi sayfaların en az ve en çok kullanıldığını takip etmek için, sistemin her bellek referansında bazı hesaplama işleri yapması gerekir. Açıkçası, büyük bir özen gösterilmediği takdirde, bu tür bir hesaplama performansı büyük ölçüde düşürebilir.

Bunu hızlandırmaya yardımcı olabilecek bir yöntem, biraz donanım desteği eklemektir. Örneğin, bir makine her sayfa erişiminde bellekteki bir zaman alanını güncelleyebilir (örneğin, bu işlem başına sayfa tablosunda veya sistemin fiziksel sayfası başına bir girişle bellekteki ayrı bir dizide olabilir). Böylece, bir sayfaya erişildiğinde, zaman alanı donanım tarafından güncel zamana ayarlanacaktır. Daha sonra, bir sayfayı değiştirirken, işletim sistemi en son kullanılan sayfayı bulmak için sistemdeki tüm zaman alanlarını tarayabilir.

Ne yazık ki, bir sistemdeki sayfa sayısı arttıkça, sadece en son kullanılan sayfayı bulmak için büyük bir diziyi defalarca taramak çok pahalıya mal olur. 4KB sayfalara bölünmüş 4GB belleğe sahip modern bir makine düşünün. Bu makinede 1 milyon sayfa vardır ve bu nedenle LRU sayfasını bulmak modern CPU hızlarında bile uzun zaman alacaktır. Bu da şu soruyu akla getiriyor: Değiştirmek için gerçekten mutlak en eski sayfayı bulmamız gerekiyor mu? Bunun yerine bir yaklaşımla hayatta kalabilir miyiz?

CRUX: LRU DEĞİŞTİRME POLİTİKASI NASIL UYGULANIR

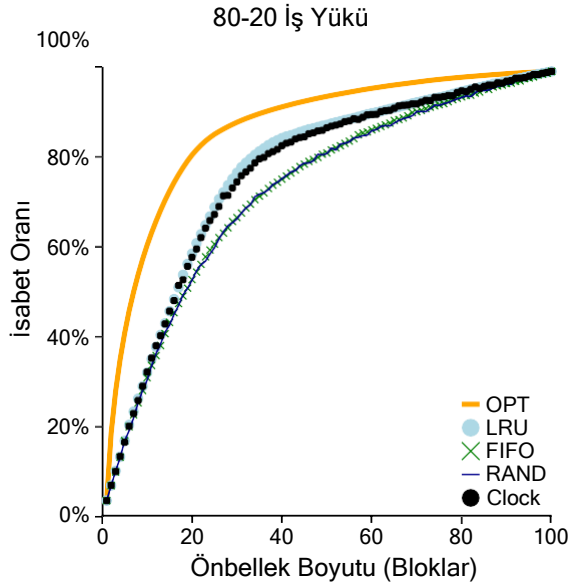
Mükemmel LRU'yu uygulamanın pahalı olacağı düşünüldüğünde, bunu bir şekilde uygulayabilir ve yine de istenen davranışı elde edebilir miyiz?

22.8 LRU'ya yaklaşma

Görünüşe göre cevap evet: LRU'ya yaklaşmak hesaplama yükü açısından daha uygundur ve aslında birçok modern sistemin yaptığı da budur. Bu fikir, **kullanım biti (use bit)** (bazen **referans biti (reference bit)** olarak da adlandırılır) şeklinde bir donanım desteği gerektirir; bu destek ilk olarak Atlas tek seviyeli depolama [KE+62] sisteminde uygulanmıştır. Sistemin her sayfası için bir kullanım biti vardır ve kullanım bitleri bellekte bir yerde bulunur (örneğin işlem başına sayfa tablolarında ya da sadece bir dizide olabilirler). Bir sayfaya her başvurulduğunda (yani okunduğunda veya yazıldığında), use biti donanım tarafından 1'e ayarlanır. Ancak donanım bu biti asla temizlemez (yani 0'a ayarlar); bu işletim sisteminin sorumluluğundadır.

İşletim sistemi LRU'ya yaklaşmak için kullanım bitini nasıl kullanır? Pek çok yol olabilir, ancak **saat algoritması (clock algorithm)** [C69] ile basit bir yaklaşım önerilmiştir. Sistemin tüm sayfalarının dairesel bir liste halinde düzenlendiğini düşünün. Bir **saat ibresi (clock hand)** başlangıçta belirli bir sayfayı işaret eder (hangisi olduğu gerçekten önemli değildir). Bir değiştirme yapılması gerektiğinde, işletim sistemi o anda işaret edilen P sayfasının 1 veya 0 kullanım bitine sahip olup olmadığını kontrol eder. 1 ise, bu P sayfasının yakın zamanda kullanıldığını ve dolayısıyla değiştirilmek için iyi bir aday olmadığını gösterir. Böylece, P için kullanım biti 0'a ayarlanır (temizlenir) ve saat ibresi bir sonraki sayfaya (P + 1) yükseltilir. Algoritma, 0'a ayarlanmış bir kullanım biti bulana kadar devam eder, bu da bu sayfanın yakın zamanda kullanılmadığını gösterir (veya en kötü durumda, tüm sayfaların kullanıldığını ve şimdi tüm sayfa kümesini arayıp tüm bitleri temizlediğimizi gösterir).

Bu yaklaşımın LRU'ya yaklaşmak için bir kullanım biti kullanmanın tek yolu olmadığını unutmayın. Aslında, kullanım bitlerini periyodik olarak temizleyen ve daha sonra hangilerinin değiştirileceğine karar vermek için hangi sayfaların kullanım bitlerinin 1 ve 0 olduğunu ayırt eden herhangi bir yaklaşım iyi olacaktır. Corbato'nun saat algoritması, bazı başarılara ulaşan ilk yaklaşımlardan sadece biriydi ve kullanılmayan bir sayfayı aramak için tüm belleği tekrar tekrar taramama gibi güzel bir özelliğe sahipti.



Şekil 22.9: **Saat ile 80-20 İş Yükü**

Bir saat algoritması varyantının davranışı Şekil 22.9'da gösterilmektedir. Bu varyant değiştirme yaparken sayfaları rastgele tarar; referans biti 1'e ayarlanmış bir sayfaya rastladığında biti temizler (yani 0'a ayarlar); referans biti 0'a ayarlanmış bir sayfa bulduğunda onu kurban olarak seçer. Gördüğünüz gibi, mükemmel LRU kadar iyi olmasa da, geçmişi hiç dikkate almayan yaklaşımlardan daha iyi sonuç veriyor.

22.9 Kirli Sayfaların Değerlendirilimesi

Saat algoritmasında yaygın olarak yapılan küçük bir değişiklik (orijinal olarak Corbato [C69] tarafından da önerilmiştir), bir sayfanın bellekteyken değiştirilip değiştirilmediğinin ek olarak dikkate alınmasıdır. Bunun nedeni: bir sayfa **değiştirilmişse (modified)** ve bu nedenle **kirliyse (dirty)**, onu boşaltmak için diske geri yazılması gerekir, bu da pahalıdır. Değiştirilmemişse (ve bu nedenle **temizse(clean)**), tahliye serbesttir; fiziksel çerçeve ek I/O olmadan başka amaçlar için yeniden kullanılabilir. Bu nedenle, bazı VM sistemleri kirli sayfalar yerine temiz sayfaları tahliye etmeyi tercih eder.

Bu davranışı desteklemek için donanımın **değiştirilmiş bir bit (modified bit)** (diğer adıyla **kirli bit (dirty bit)**) içermesi gerekir. Bu bit her sayfa yazıldığında ayarlanır ve böylece sayfa değiştirme algoritmasına dahil edilebilir. Örneğin saat algoritması, önce tahliye etmek için hem kullanılmayan hem de temiz sayfaları tarayacak şekilde değiştirilebilir; bunları bulamazsa, daha sonra kirli olan kullanılmayan sayfaları tarar ve bu şekilde devam eder.

22.10 Diğer Sanal Makine(VM) Politikaları

Sayfa değiştirme, VM alt sisteminin kullandığı tek politika değildir (ancak en önemlisi olabilir). Örneğin, işletim sisteminin bir sayfayı belleğe ne zaman getireceğine de karar vermesi gerekir. Bazen **sayfa seçim (page selection)** politikası olarak adlandırılan bu politika (Denning [D70] tarafından adlandırıldığı gibi), işletim sistemine bazı farklı seçenekler sunar.

Çoğu sayfa için işletim sistemi basitçe **talep (demand paging)** üzerine sayfalama kullanır, yani işletim sistemi sayfaya erişildiğinde sayfayı "talep üzerine" belleğe getirir. Elbette, işletim sistemi bir sayfanın kullanılmak üzere olduğunu tahmin edebilir ve böylece onu önceden getirebilir; bu davranış **ön-getirme (prefetching)** olarak bilinir ve yalnızca makul bir başarı şansı olduğunda yapılmalıdır. Örneğin, bazı sistemler bir *P* kod sayfası belleğe getirilirse*, P +1* kod sayfasına muhtemelen yakında erişileceğini ve bu nedenle de belleğe getirilmesi gerektiğini varsayacaktır.

Başka bir politika, işletim sisteminin sayfaları diske nasıl yazacağını belirler. Elbette, bunlar teker teker yazılabilir; ancak, birçok sistem bunun yerine bekleyen bir dizi yazmayı bellekte bir araya getirir ve bunları tek bir (daha verimli) yazma işleminde diske yazar. Bu davranış genellikle **kümeleme (Clustering)** veya basitçe yazma **gruplaması (grouping)** olarak adlandırılır ve tek bir büyük yazmayı birçok küçük yazmadan daha verimli bir şekilde gerçekleştiren disk sürücülerinin doğası nedeniyle etkilidir.

22.11 Hırpalama

Kapanıştan önce, son bir soruyu ele alacağız: bellek basitçe aşırı talep edildiğinde ve çalışan süreçler kümesinin bellek talepleri mevcut fiziksel belleği aştığında işletim sistemi ne yapmalıdır? Bu durumda, sistem sürekli olarak sayfalama yapacaktır,bu durum bazen **hırpalama (thrashing)** olarak adlandırılır [D70].

Daha önceki bazı işletim sistemleri, meydana geldiğinde thrashing'i hem tespit etmek hem de bununla başa çıkmak için oldukça karmaşık bir dizi mekanizmaya sahipti. Örneğin, bir süreçler kümesi verildiğinde, bir sistem, azaltılmış süreçler kümesinin **çalışma kümelerinin (working setse)** (aktif olarak kullandıkları sayfalar) belleğe sığması ve böylece ilerleme kaydedebilmesi umuduyla, bir alt süreçler kümesini çalıştırmamaya karar verebilir. Genel olarak **giriş denetimi (admission control)** olarak bilinen bu yaklaşım, bazen her şeyi bir kerede kötü yapmaya çalışmaktansa daha az işi iyi yapmanın daha iyi olduğunu belirtir ki bu gerçek hayatta olduğu kadar modern bilgisayar sistemlerinde de sıklıkla karşılaştığımız bir durumdur (ne yazık ki).

Bazı mevcut sistemler aşırı bellek yüküne karşı daha acımasız bir yaklaşım benimsemektedir. Örneğin, Linux'un bazı sürümleri bellek aşırı kullanıldığında bir **bellek dışı katili (out-of-memory killer)** çalıştırır;bu arka plan programı, belleğin yoğun olduğu bir süreci seçer ve onu öldürür, böylece belleği çok da ince olmayan bir şekilde azaltır. Bellek baskısını azaltmada başarılı olsa da, bu yaklaşım örneğin X sunucusunu öldürür ve böylece ekrana ihtiyaç duyan tüm uygulamaları kullanılamaz hale getirirse sorunlara yol açabilir.

22.12 Özet

Tüm modern işletim sistemlerinin VM alt sisteminin bir parçası olan bir dizi sayfa değiştirme (ve diğer) politikalarının kullanıldığını gördük. Modern sistemler, saat gibi basit LRU yaklaşımlarına bazı ince ayarlar ekler; örneğin, **tarama direnci (scan resistance)** ARC [MM03] gibi birçok modern algoritmanın önemli bir parçasıdır. Taramaya dayanıklı algoritmalar genellikle LRU benzeri olmakla birlikte, LRU'nun döngü-sıralı iş yükünde gördüğümüz en kötü durum davranışından kaçınmaya çalışırlar. Böylece sayfa değiştirme algoritmalarının evrimi devam etmektedir.

Uzun yıllar boyunca, bellek erişimi ve disk erişimi süreleri arasındaki fark çok büyük olduğu için değiştirme algoritmalarının önemi artmıştı. Özellikle, diske sayfalama çok pahalı olduğu için, sık sık sayfalama yapmanın maliyeti çok yüksekti; basitçe söylemek gerekirse, değiştirme algoritmanız ne kadar iyi olursa olsun, sık sık yeniden yerleştirme yapıyorsanız, sisteminiz dayanılmaz derecede yavaşlıyordu. Dolayısıyla, en iyi çözüm basit (entelektüel açıdan tatmin edici olmasa da) bir çözümdü: daha fazla bellek satın almak.

Ancak, çok daha hızlı depolama cihazlarındaki son yenilikler (örneğin Flash tabanlı SSD'ler ve Intel Optane ürünleri) bu performans oranlarını bir kez daha değiştirerek sayfa değiştirme algoritmalarında bir yenilenmeye yol açmıştır. Bu alandaki son çalışmalar için [SS10,W+21]'e bakınız.

References

[AD03] "Run-Time Adaptation in River" by Remzi H. Arpaci-Dusseau. ACM TOCS, 21:1, February 2003. *A summary of one of the authors' dissertation work on a system named River, where he learned that comparison against the ideal is an important technique for system designers.*

[B66] "A Study of Replacement Algorithms for Virtual-Storage Computer" by Laszlo A. Be- lady. IBM Systems Journal 5(2): 78-101, 1966. *The paper that introduces the simple way to compute the optimal behavior of a policy (the MIN algorithm).*

[BNS69] "An Anomaly in Space-time Characteristics of Certain Programs Running in a Paging Machine" by L. A. Belady, R. A. Nelson, G. S. Shedler. Communications of the ACM, 12:6, June 1969. *Introduction of the little sequence of memory references known as Belady's Anomaly. How do Nelson and Shedler feel about this name, we wonder?*

[CD85] "An Evaluation of Buffer Management Strategies for Relational Database Systems" by Hong-Tai Chou, David J. DeWitt. VLDB '85, Stockholm, Sweden, August 1985. *A famous database paper on the different buffering strategies you should use under a number of common database access patterns. The more general lesson: if you know something about a workload, you can tailor policies to do better than the general-purpose ones usually found in the OS.*

[C69] "A Paging Experiment with the Multics System" by F.J. Corbato. Included in a Festschrift published in honor of Prof. P.M. Morse. MIT Press, Cambridge, MA, 1969. *The original (and hard to find!) reference to the clock algorithm, though not the first usage of a use bit. Thanks to H. Balakrishnan of MIT for digging up this paper for us.*

[D70] "Virtual Memory" by Peter J. Denning. Computing Surveys, Vol. 2, No. 3, September 1970. *Denning's early and famous survey on virtual memory systems.*

[EF78] "Cold-start vs. Warm-start Miss Ratios" by Malcolm C. Easton, Ronald Fagin. Communications of the ACM, 21:10, October 1978. *A good discussion of cold- vs. warm-start misses.*

[FP89] "Electrochemically Induced Nuclear Fusion of Deuterium" by Martin Fleischmann, Stanley Pons. Journal of Electroanalytical Chemistry, Volume 26, Number 2, Part 1, April, 1989. *The famous paper that would have revolutionized the world in providing an easy way to generate nearly-infinite power from jars of water with a little metal in them. Unfortunately, the results pub- lished (and widely publicized) by Pons and Fleischmann were impossible to reproduce, and thus these two well-meaning scientists were discredited (and certainly, mocked). The only guy really happy about this result was Marvin Hawkins, whose name was left off this paper even though he participated in the work, thus avoiding association with one of the biggest scientific goofs of the 20th century.*

[HP06] "Computer Architecture: A Quantitative Approach" by John Hennessy and David Patterson. Morgan-Kaufmann, 2006. *A marvelous book about computer architecture. Read it!*

[H87] "Aspects of Cache Memory and Instruction Buffer Performance" by Mark D. Hill. Ph.D. Dissertation, U.C. Berkeley, 1987. *Mark Hill, in his dissertation work, introduced the Three C's, which later gained wide popularity with its inclusion in H&P [HP06]. The quote from therein: "I have found it useful to partition misses ... into three components intuitively based on the cause of the misses (page 49)."*

[KE+62] "One-level Storage System" by T. Kilburn, D.B.G. Edwards, M.J. Lanigan, F.H. Sumner. IRE Trans. EC-11:2, 1962. *Although Atlas had a use bit, it only had a very small number of pages, and thus the scanning of the use bits in large memories was not a problem the authors solved.*

[M+70] "Evaluation Techniques for Storage Hierarchies" by R. L. Mattson, J. Gecsei, D. R. Slutz, I. L. Traiger. IBM Systems Journal, Volume 9:2, 1970. *A paper that is mostly about how to simulate cache hierarchies efficiently; certainly a classic in that regard, as well for its excellent discussion of some of the properties of various replacement algorithms. Can you figure out why the stack property might be useful for simulating a lot of different-sized caches at once?*

[MM03] "ARC: A Self-Tuning, Low Overhead Replacement Cache" by Nimrod Megiddo and Dharmendra S. Modha. FAST 2003, February 2003, San Jose, California. *An excellent modern paper about replacement algorithms, which includes a new policy, ARC, that is now used in some systems. Recognized in 2014 as a "Test of Time" award winner by the storage systems community at the FAST '14 conference.*

[SS10] "FlashVM: Virtual Memory Management on Flash" by Mohit Saxena, Michael M. Swift. USENIX ATC '10, June, 2010, Boston, MA. *An early, excellent paper by our colleagues at U. Wisconsin about how to use Flash for paging. One interesting twist is how the system has to take* ***wearout****, an intrinsic property of Flash-based devices, into account. Read more about Flash-based SSDs later in this book if you are interested.*

[W+21] "The Storage Hierarchy is Not a Hierarchy: Optimizing Caching on Modern Storage Devices with Orthus" by Kan Wu, Zhihan Guo, Guanzhou Hu, Kaiwei Tu, Ramnatthan Alagappan, Rathijit Sen, Kwanghyun Park, Andrea Arpaci-Dusseau, Remzi Arpaci-Dusseau. FAST '21, held virtually – thanks, COVID-19. *Our own work on a caching approach on modern devices; is it directly related to page replacement? Perhaps not. But it is a fun paper.*

EV ÖDEVLERİ (SİMÜLASYON)

Bu simülatör, paging-policy.py, farklı sayfa değiştirme politikaları ile oynamanıza izin verir. Ayrıntılar için README'ye bakın.

SORULAR

1. Aşağıdaki argümanlarla rastgele adresler oluşturun: `-s 0 -n 10`, `-s 1 -n 10`, ve `-s 2 -n 10`. İlkeyi FIFO'dan LRU'ya ve OPT'ye değiştirin. Söz konusu adres izlerindeki her bir erişimin isabetli mi yoksa hatalı mı olduğunu hesaplayın.

FIFO politikası -s 0 -n 10 için:

```
burak@ubuntu:~/Desktop/ostep-homework-master/vm-beyondphys-policy$ python3 ./paging-policy.py -s 0 -n 10
ARG addresses -1
ARG addressfile
ARG numaddrs 10
ARG policy FIFO
ARG clockbits 2
ARG cachesize 3
ARG maxpage 10
ARG seed 0
ARG notrace False

Assuming a replacement policy of FIFO, and a cache of size 3 pages,
figure out whether each of the following page references hit or miss
in the page cache.

Access: 8  Hit/Miss?  State of Memory?
Access: 7  Hit/Miss?  State of Memory?
Access: 4  Hit/Miss?  State of Memory?
Access: 2  Hit/Miss?  State of Memory?
Access: 5  Hit/Miss?  State of Memory?
Access: 4  Hit/Miss?  State of Memory?
Access: 7  Hit/Miss?  State of Memory?
Access: 3  Hit/Miss?  State of Memory?
Access: 4  Hit/Miss?  State of Memory?
Access: 5  Hit/Miss?  State of Memory?
```

Cevap:

1 isabet 9 ıska isabet oranı ise %10

```
burak@ubuntu:~/Desktop/ostep-homework-master/vm-beyondphys-policy$ python3 ./paging-policy.py -s 0 -n 10 -c
ARG addresses -1
ARG addressfile
ARG numaddrs 10
ARG policy FIFO
ARG clockbits 2
ARG cachesize 3
ARG maxpage 10
ARG seed 0
ARG notrace False

Solving...

Access: 8  MISS FirstIn ->          [8] <- Lastin  Replaced:- [Hits:0 Misses:1]
Access: 7  MISS FirstIn ->       [8, 7] <- Lastin  Replaced:- [Hits:0 Misses:2]
Access: 4  MISS FirstIn ->    [8, 7, 4] <- Lastin  Replaced:- [Hits:0 Misses:3]
Access: 2  MISS FirstIn ->    [7, 4, 2] <- Lastin  Replaced:8 [Hits:0 Misses:4]
Access: 5  MISS FirstIn ->    [4, 2, 5] <- Lastin  Replaced:7 [Hits:0 Misses:5]
Access: 4  HIT  FirstIn ->    [4, 2, 5] <- Lastin  Replaced:- [Hits:1 Misses:5]
Access: 7  MISS FirstIn ->    [2, 5, 7] <- Lastin  Replaced:4 [Hits:1 Misses:6]
Access: 3  MISS FirstIn ->    [5, 7, 3] <- Lastin  Replaced:2 [Hits:1 Misses:7]
Access: 4  MISS FirstIn ->    [7, 3, 4] <- Lastin  Replaced:5 [Hits:1 Misses:8]
Access: 5  MISS FirstIn ->    [3, 4, 5] <- Lastin  Replaced:7 [Hits:1 Misses:9]

FINALSTATS hits 1   misses 9   hitrate 10.00
```

LRU politikası-s 0 -n 10 için:

```
burak@ubuntu:~/Desktop/ostep-homework-master/vm-beyondphys-policy$ python3 ./paging-policy.py -s 0 -n 10 -p LRU
ARG addresses -1
ARG addressfile
ARG numaddrs 10
ARG policy LRU
ARG clockbits 2
ARG cachesize 3
ARG maxpage 10
ARG seed 0
ARG notrace False

Assuming a replacement policy of LRU, and a cache of size 3 pages,
figure out whether each of the following page references hit or miss
in the page cache.

Access: 8  Hit/Miss?  State of Memory?
Access: 7  Hit/Miss?  State of Memory?
Access: 4  Hit/Miss?  State of Memory?
Access: 2  Hit/Miss?  State of Memory?
Access: 5  Hit/Miss?  State of Memory?
Access: 4  Hit/Miss?  State of Memory?
Access: 7  Hit/Miss?  State of Memory?
Access: 3  Hit/Miss?  State of Memory?
Access: 4  Hit/Miss?  State of Memory?
Access: 5  Hit/Miss?  State of Memory?
```

Cevap:

2 isabet 8 ıska %20 isabet oranı

```
burak@ubuntu:~/Desktop/ostep-homework-master/vm-beyondphys-policy$ python3 ./paging-policy.py -s 0 -n 10 -c -p LRU
ARG addresses -1
ARG addressfile
ARG numaddrs 10
ARG policy LRU
ARG clockbits 2
ARG cachesize 3
ARG maxpage 10
ARG seed 0
ARG notrace False

Solving...

Access: 8  MISS LRU ->         [8] <- MRU Replaced:- [Hits:0 Misses:1]
Access: 7  MISS LRU ->      [8, 7] <- MRU Replaced:- [Hits:0 Misses:2]
Access: 4  MISS LRU ->   [8, 7, 4] <- MRU Replaced:- [Hits:0 Misses:3]
Access: 2  MISS LRU ->   [7, 4, 2] <- MRU Replaced:8 [Hits:0 Misses:4]
Access: 5  MISS LRU ->   [4, 2, 5] <- MRU Replaced:7 [Hits:0 Misses:5]
Access: 4  HIT  LRU ->   [2, 5, 4] <- MRU Replaced:- [Hits:1 Misses:5]
Access: 7  MISS LRU ->   [5, 4, 7] <- MRU Replaced:2 [Hits:1 Misses:6]
Access: 3  MISS LRU ->   [4, 7, 3] <- MRU Replaced:5 [Hits:1 Misses:7]
Access: 4  HIT  LRU ->   [7, 3, 4] <- MRU Replaced:- [Hits:2 Misses:7]
Access: 5  MISS LRU ->   [3, 4, 5] <- MRU Replaced:7 [Hits:2 Misses:8]

FINALSTATS hits 2   misses 8   hitrate 20.00
```

OPT politikası -s 0 -n 10 için:

```
burak@ubuntu:~/Desktop/ostep-homework-master/vm-beyondphys-policy$ python3 ./paging-policy.py -s 0 -n 10 -p OPT
ARG addresses -1
ARG addressfile
ARG numaddrs 10
ARG policy OPT
ARG clockbits 2
ARG cachesize 3
ARG maxpage 10
ARG seed 0
ARG notrace False

Assuming a replacement policy of OPT, and a cache of size 3 pages,
figure out whether each of the following page references hit or miss
in the page cache.

Access: 8  Hit/Miss?  State of Memory?
Access: 7  Hit/Miss?  State of Memory?
Access: 4  Hit/Miss?  State of Memory?
Access: 2  Hit/Miss?  State of Memory?
Access: 5  Hit/Miss?  State of Memory?
Access: 4  Hit/Miss?  State of Memory?
Access: 7  Hit/Miss?  State of Memory?
Access: 3  Hit/Miss?  State of Memory?
Access: 4  Hit/Miss?  State of Memory?
Access: 5  Hit/Miss?  State of Memory?
```

Cevap:
4 isabet 6 ıska %40 isabet oranı

```
burak@ubuntu:~/Desktop/ostep-homework-master/vm-beyondphys-policy$ python3 ./paging-policy.py -s 0 -n 10 -c -p OPT
ARG addresses -1
ARG addressfile
ARG numaddrs 10
ARG policy OPT
ARG clockbits 2
ARG cachesize 3
ARG maxpage 10
ARG seed 0
ARG notrace False

Solving...

Access: 8  MISS Left  ->         [8] <- Right Replaced:- [Hits:0 Misses:1]
Access: 7  MISS Left  ->      [8, 7] <- Right Replaced:- [Hits:0 Misses:2]
Access: 4  MISS Left  ->   [8, 7, 4] <- Right Replaced:- [Hits:0 Misses:3]
Access: 2  MISS Left  ->   [7, 4, 2] <- Right Replaced:8 [Hits:0 Misses:4]
Access: 5  MISS Left  ->   [7, 4, 5] <- Right Replaced:2 [Hits:0 Misses:5]
Access: 4  HIT  Left  ->   [7, 4, 5] <- Right Replaced:- [Hits:1 Misses:5]
Access: 7  HIT  Left  ->   [7, 4, 5] <- Right Replaced:- [Hits:2 Misses:5]
Access: 3  MISS Left  ->   [4, 5, 3] <- Right Replaced:7 [Hits:2 Misses:6]
Access: 4  HIT  Left  ->   [4, 5, 3] <- Right Replaced:- [Hits:3 Misses:6]
Access: 5  HIT  Left  ->   [4, 5, 3] <- Right Replaced:- [Hits:4 Misses:6]

FINALSTATS hits 4   misses 6   hitrate 40.00
```

* -s 0 -n 10 için isabet oranı sıralaması OPT>LRU> FIFO

FIFO politikası -s 1 -n 10 için:

```
burak@ubuntu:~/Desktop/ostep-homework-master/vm-beyondphys-policy$ python3 ./paging-policy.py -s 1 -n 10
ARG addresses -1
ARG addressfile
ARG numaddrs 10
ARG policy FIFO
ARG clockbits 2
ARG cachesize 3
ARG maxpage 10
ARG seed 1
ARG notrace False

Assuming a replacement policy of FIFO, and a cache of size 3 pages,
figure out whether each of the following page references hit or miss
in the page cache.

Access: 1  Hit/Miss?  State of Memory?
Access: 8  Hit/Miss?  State of Memory?
Access: 7  Hit/Miss?  State of Memory?
Access: 2  Hit/Miss?  State of Memory?
Access: 4  Hit/Miss?  State of Memory?
Access: 4  Hit/Miss?  State of Memory?
Access: 6  Hit/Miss?  State of Memory?
Access: 7  Hit/Miss?  State of Memory?
Access: 0  Hit/Miss?  State of Memory?
Access: 0  Hit/Miss?  State of Memory?
```

Cevap:
2 isabet 8 ıska %20 isabet oranı

```
burak@ubuntu:~/Desktop/ostep-homework-master/vm-beyondphys-policy$ python3 ./paging-policy.py -s 1 -n 10 -c
ARG addresses -1
ARG addressfile
ARG numaddrs 10
ARG policy FIFO
ARG clockbits 2
ARG cachesize 3
ARG maxpage 10
ARG seed 1
ARG notrace False

Solving...

Access: 1  MISS FirstIn ->          [1] <- Lastin  Replaced:- [Hits:0 Misses:1]
Access: 8  MISS FirstIn ->       [1, 8] <- Lastin  Replaced:- [Hits:0 Misses:2]
Access: 7  MISS FirstIn ->    [1, 8, 7] <- Lastin  Replaced:- [Hits:0 Misses:3]
Access: 2  MISS FirstIn ->    [8, 7, 2] <- Lastin  Replaced:1 [Hits:0 Misses:4]
Access: 4  MISS FirstIn ->    [7, 2, 4] <- Lastin  Replaced:8 [Hits:0 Misses:5]
Access: 4  HIT  FirstIn ->    [7, 2, 4] <- Lastin  Replaced:- [Hits:1 Misses:5]
Access: 6  MISS FirstIn ->    [2, 4, 6] <- Lastin  Replaced:7 [Hits:1 Misses:6]
Access: 7  MISS FirstIn ->    [4, 6, 7] <- Lastin  Replaced:2 [Hits:1 Misses:7]
Access: 0  MISS FirstIn ->    [6, 7, 0] <- Lastin  Replaced:4 [Hits:1 Misses:8]
Access: 0  HIT  FirstIn ->    [6, 7, 0] <- Lastin  Replaced:- [Hits:2 Misses:8]

FINALSTATS hits 2   misses 8   hitrate 20.00
```

LRU politikası -s 1 -n 10 için:

```
burak@ubuntu:~/Desktop/ostep-homework-master/vm-beyondphys-policy$ python3 ./paging-policy.py -s 1 -n 10 -p LRU
ARG addresses -1
ARG addressfile
ARG numaddrs 10
ARG policy LRU
ARG clockbits 2
ARG cachesize 3
ARG maxpage 10
ARG seed 1
ARG notrace False

Assuming a replacement policy of LRU, and a cache of size 3 pages,
figure out whether each of the following page references hit or miss
in the page cache.

Access: 1  Hit/Miss?  State of Memory?
Access: 8  Hit/Miss?  State of Memory?
Access: 7  Hit/Miss?  State of Memory?
Access: 2  Hit/Miss?  State of Memory?
Access: 4  Hit/Miss?  State of Memory?
Access: 4  Hit/Miss?  State of Memory?
Access: 6  Hit/Miss?  State of Memory?
Access: 7  Hit/Miss?  State of Memory?
Access: 0  Hit/Miss?  State of Memory?
Access: 0  Hit/Miss?  State of Memory?
```

Cevap:
2 isabet 8 ıska %20 isabet oranı

```
burak@ubuntu:~/Desktop/ostep-homework-master/vm-beyondphys-policy$ python3 ./paging-policy.py -s 1 -n 10 -c -p LRU
ARG addresses -1
ARG addressfile
ARG numaddrs 10
ARG policy LRU
ARG clockbits 2
ARG cachesize 3
ARG maxpage 10
ARG seed 1
ARG notrace False

Solving...

Access: 1  MISS LRU ->         [1] <- MRU Replaced:- [Hits:0 Misses:1]
Access: 8  MISS LRU ->      [1, 8] <- MRU Replaced:- [Hits:0 Misses:2]
Access: 7  MISS LRU ->   [1, 8, 7] <- MRU Replaced:- [Hits:0 Misses:3]
Access: 2  MISS LRU ->   [8, 7, 2] <- MRU Replaced:1 [Hits:0 Misses:4]
Access: 4  MISS LRU ->   [7, 2, 4] <- MRU Replaced:8 [Hits:0 Misses:5]
Access: 4  HIT  LRU ->   [7, 2, 4] <- MRU Replaced:- [Hits:1 Misses:5]
Access: 6  MISS LRU ->   [2, 4, 6] <- MRU Replaced:7 [Hits:1 Misses:6]
Access: 7  MISS LRU ->   [4, 6, 7] <- MRU Replaced:2 [Hits:1 Misses:7]
Access: 0  MISS LRU ->   [6, 7, 0] <- MRU Replaced:4 [Hits:1 Misses:8]
Access: 0  HIT  LRU ->   [6, 7, 0] <- MRU Replaced:- [Hits:2 Misses:8]

FINALSTATS hits 2   misses 8   hitrate 20.00
```

OPT politikası -s 1 -n 10 için:

```
burak@ubuntu:~/Desktop/ostep-homework-master/vm-beyondphys-policy$ python3 ./paging-policy.py -s 1 -n 10 -p OPT
ARG addresses -1
ARG addressfile
ARG numaddrs 10
ARG policy OPT
ARG clockbits 2
ARG cachesize 3
ARG maxpage 10
ARG seed 1
ARG notrace False

Assuming a replacement policy of OPT, and a cache of size 3 pages,
figure out whether each of the following page references hit or miss
in the page cache.

Access: 1  Hit/Miss?  State of Memory?
Access: 8  Hit/Miss?  State of Memory?
Access: 7  Hit/Miss?  State of Memory?
Access: 2  Hit/Miss?  State of Memory?
Access: 4  Hit/Miss?  State of Memory?
Access: 4  Hit/Miss?  State of Memory?
Access: 6  Hit/Miss?  State of Memory?
Access: 7  Hit/Miss?  State of Memory?
Access: 0  Hit/Miss?  State of Memory?
Access: 0  Hit/Miss?  State of Memory?
```

Cevap:
3 isabet 7 ıska %30 isabet oranı

```
burak@ubuntu:~/Desktop/ostep-homework-master/vm-beyondphys-policy$ python3 ./paging-policy.py -s 1 -n 10 -c -p OPT
ARG addresses -1
ARG addressfile
ARG numaddrs 10
ARG policy OPT
ARG clockbits 2
ARG cachesize 3
ARG maxpage 10
ARG seed 1
ARG notrace False

Solving...

Access: 1  MISS Left  ->         [1] <- Right Replaced:- [Hits:0 Misses:1]
Access: 8  MISS Left  ->      [1, 8] <- Right Replaced:- [Hits:0 Misses:2]
Access: 7  MISS Left  ->   [1, 8, 7] <- Right Replaced:- [Hits:0 Misses:3]
Access: 2  MISS Left  ->   [1, 7, 2] <- Right Replaced:8 [Hits:0 Misses:4]
Access: 4  MISS Left  ->   [1, 7, 4] <- Right Replaced:2 [Hits:0 Misses:5]
Access: 4  HIT  Left  ->   [1, 7, 4] <- Right Replaced:- [Hits:1 Misses:5]
Access: 6  MISS Left  ->   [1, 7, 6] <- Right Replaced:4 [Hits:1 Misses:6]
Access: 7  HIT  Left  ->   [1, 7, 6] <- Right Replaced:- [Hits:2 Misses:6]
Access: 0  MISS Left  ->   [1, 7, 0] <- Right Replaced:6 [Hits:2 Misses:7]
Access: 0  HIT  Left  ->   [1, 7, 0] <- Right Replaced:- [Hits:3 Misses:7]

FINALSTATS hits 3   misses 7   hitrate 30.00
```

*-s 1 -n 10 için isabet oranı sıralaması OPT>FIFO=LRU

FIFO politikası -s 2 -n 10 için:

```
burak@ubuntu:~/Desktop/ostep-homework-master/vm-beyondphys-policy$ python3 ./paging-policy.py -s 2 -n 10
ARG addresses -1
ARG addressfile
ARG numaddrs 10
ARG policy FIFO
ARG clockbits 2
ARG cachesize 3
ARG maxpage 10
ARG seed 2
ARG notrace False

Assuming a replacement policy of FIFO, and a cache of size 3 pages,
figure out whether each of the following page references hit or miss
in the page cache.

Access: 9  Hit/Miss?  State of Memory?
Access: 9  Hit/Miss?  State of Memory?
Access: 0  Hit/Miss?  State of Memory?
Access: 0  Hit/Miss?  State of Memory?
Access: 8  Hit/Miss?  State of Memory?
Access: 7  Hit/Miss?  State of Memory?
Access: 6  Hit/Miss?  State of Memory?
Access: 3  Hit/Miss?  State of Memory?
Access: 6  Hit/Miss?  State of Memory?
Access: 6  Hit/Miss?  State of Memory?
```

Cevap:
4 isabet 6 ıska %40 isabet oranı

```
burak@ubuntu:~/Desktop/ostep-homework-master/vm-beyondphys-policy$ python3 ./paging-policy.py -s 2 -n 10 -c
ARG addresses -1
ARG addressfile
ARG numaddrs 10
ARG policy FIFO
ARG clockbits 2
ARG cachesize 3
ARG maxpage 10
ARG seed 2
ARG notrace False

Solving...

Access: 9  MISS FirstIn ->         [9] <- Lastin  Replaced:- [Hits:0 Misses:1]
Access: 9  HIT  FirstIn ->         [9] <- Lastin  Replaced:- [Hits:1 Misses:1]
Access: 0  MISS FirstIn ->      [9, 0] <- Lastin  Replaced:- [Hits:1 Misses:2]
Access: 0  HIT  FirstIn ->      [9, 0] <- Lastin  Replaced:- [Hits:2 Misses:2]
Access: 8  MISS FirstIn ->   [9, 0, 8] <- Lastin  Replaced:- [Hits:2 Misses:3]
Access: 7  MISS FirstIn ->   [0, 8, 7] <- Lastin  Replaced:9 [Hits:2 Misses:4]
Access: 6  MISS FirstIn ->   [8, 7, 6] <- Lastin  Replaced:0 [Hits:2 Misses:5]
Access: 3  MISS FirstIn ->   [7, 6, 3] <- Lastin  Replaced:8 [Hits:2 Misses:6]
Access: 6  HIT  FirstIn ->   [7, 6, 3] <- Lastin  Replaced:- [Hits:3 Misses:6]
Access: 6  HIT  FirstIn ->   [7, 6, 3] <- Lastin  Replaced:- [Hits:4 Misses:6]

FINALSTATS hits 4   misses 6   hitrate 40.00
```

LRU politikası -s 2 -n 10 için:

```
burak@ubuntu:~/Desktop/ostep-homework-master/vm-beyondphys-policy$ python3 ./paging-policy.py -s 2 -n 10 -p LRU
ARG addresses -1
ARG addressfile
ARG numaddrs 10
ARG policy LRU
ARG clockbits 2
ARG cachesize 3
ARG maxpage 10
ARG seed 2
ARG notrace False

Assuming a replacement policy of LRU, and a cache of size 3 pages,
figure out whether each of the following page references hit or miss
in the page cache.

Access: 9  Hit/Miss?  State of Memory?
Access: 9  Hit/Miss?  State of Memory?
Access: 0  Hit/Miss?  State of Memory?
Access: 0  Hit/Miss?  State of Memory?
Access: 8  Hit/Miss?  State of Memory?
Access: 7  Hit/Miss?  State of Memory?
Access: 6  Hit/Miss?  State of Memory?
Access: 3  Hit/Miss?  State of Memory?
Access: 6  Hit/Miss?  State of Memory?
Access: 6  Hit/Miss?  State of Memory?
```

Cevap:

4 isabet 6 ıska %40 isabet oranı

```
burak@ubuntu:~/Desktop/ostep-homework-master/vm-beyondphys-policy$ python3 ./paging-policy.py -s 2 -n 10 -c -p LRU
ARG addresses -1
ARG addressfile
ARG numaddrs 10
ARG policy LRU
ARG clockbits 2
ARG cachesize 3
ARG maxpage 10
ARG seed 2
ARG notrace False

Solving...

Access: 9  MISS LRU ->          [9] <- MRU Replaced:- [Hits:0 Misses:1]
Access: 9  HIT  LRU ->          [9] <- MRU Replaced:- [Hits:1 Misses:1]
Access: 0  MISS LRU ->       [9, 0] <- MRU Replaced:- [Hits:1 Misses:2]
Access: 0  HIT  LRU ->       [9, 0] <- MRU Replaced:- [Hits:2 Misses:2]
Access: 8  MISS LRU ->    [9, 0, 8] <- MRU Replaced:- [Hits:2 Misses:3]
Access: 7  MISS LRU ->    [0, 8, 7] <- MRU Replaced:9 [Hits:2 Misses:4]
Access: 6  MISS LRU ->    [8, 7, 6] <- MRU Replaced:0 [Hits:2 Misses:5]
Access: 3  MISS LRU ->    [7, 6, 3] <- MRU Replaced:8 [Hits:2 Misses:6]
Access: 6  HIT  LRU ->    [7, 3, 6] <- MRU Replaced:- [Hits:3 Misses:6]
Access: 6  HIT  LRU ->    [7, 3, 6] <- MRU Replaced:- [Hits:4 Misses:6]

FINALSTATS hits 4   misses 6   hitrate 40.00
```

OPT politikası -s 2 -n 10 için:

```
burak@ubuntu:~/Desktop/ostep-homework-master/vm-beyondphys-policy$ python3 ./paging-policy.py -s 2 -n 10 -p OPT
ARG addresses -1
ARG addressfile
ARG numaddrs 10
ARG policy OPT
ARG clockbits 2
ARG cachesize 3
ARG maxpage 10
ARG seed 2
ARG notrace False

Assuming a replacement policy of OPT, and a cache of size 3 pages,
figure out whether each of the following page references hit or miss
in the page cache.

Access: 9  Hit/Miss?  State of Memory?
Access: 9  Hit/Miss?  State of Memory?
Access: 0  Hit/Miss?  State of Memory?
Access: 0  Hit/Miss?  State of Memory?
Access: 8  Hit/Miss?  State of Memory?
Access: 7  Hit/Miss?  State of Memory?
Access: 6  Hit/Miss?  State of Memory?
Access: 3  Hit/Miss?  State of Memory?
Access: 6  Hit/Miss?  State of Memory?
Access: 6  Hit/Miss?  State of Memory?
```

Cevap:
4 isabet 6 ıska %40 isabet oranı

```
burak@ubuntu:~/Desktop/ostep-homework-master/vm-beyondphys-policy$ python3 ./paging-policy.py -s 2 -n 10 -c -p OPT
ARG addresses -1
ARG addressfile
ARG numaddrs 10
ARG policy OPT
ARG clockbits 2
ARG cachesize 3
ARG maxpage 10
ARG seed 2
ARG notrace False

Solving...

Access: 9  MISS Left  ->          [9] <- Right Replaced:- [Hits:0 Misses:1]
Access: 9  HIT  Left  ->          [9] <- Right Replaced:- [Hits:1 Misses:1]
Access: 0  MISS Left  ->       [9, 0] <- Right Replaced:- [Hits:1 Misses:2]
Access: 0  HIT  Left  ->       [9, 0] <- Right Replaced:- [Hits:2 Misses:2]
Access: 8  MISS Left  ->    [9, 0, 8] <- Right Replaced:- [Hits:2 Misses:3]
Access: 7  MISS Left  ->    [9, 0, 7] <- Right Replaced:8 [Hits:2 Misses:4]
Access: 6  MISS Left  ->    [9, 0, 6] <- Right Replaced:7 [Hits:2 Misses:5]
Access: 3  MISS Left  ->    [9, 6, 3] <- Right Replaced:0 [Hits:2 Misses:6]
Access: 6  HIT  Left  ->    [9, 6, 3] <- Right Replaced:- [Hits:3 Misses:6]
Access: 6  HIT  Left  ->    [9, 6, 3] <- Right Replaced:- [Hits:4 Misses:6]

FINALSTATS hits 4   misses 6   hitrate 40.00
```

*-s 2 -n 10 için isabet oranı sıralaması OPT=FIFO=LRU

*Daha önce öğrendiğimiz gibi FIFO ilk getirilen sayfayı dışarı atar; eğer bu sayfa üzerinde önemli kod veya veri yapıları olan bir sayfaysa, yakında tekrar sayfalanacak olsa bile yine de dışarı atılır. Bu nedenle, FIFO, Rastgele ve benzeri politikaların optimuma yaklaşması muhtemel değildir;
*LRU en eski kullanılanı atar.
*Optimal ileriye bakar en son hangisine ulaşılmışssa onu atar.
Yukardaki sonuçları incelersek beklediğimiz gibi OPT nin tüm durumlarda en iyi isabet oranını (hitrate) verdiğini görürüz.

2. Boyutu 5 olan bir önbellek için, aşağıdaki politikaların her biri için en kötü durum adres referans akışları oluşturun: FIFO, LRU ve MRU (en kötü durum referans akışları mümkün olan en fazla ıskalamaya neden olur. En kötü durumdaki referans akışları için, performansı önemli ölçüde artırmak ve OPT'ye yaklaşmak için ne kadar daha büyük bir önbellek gerekir?

Önbellek boyutu (cachesize(C)) =5 En kötü durum referans akışlarını oluşturduk:

FIFO:

```
burak@ubuntu:~/Desktop/ostep-homework-master/vm-beyondphys-policy$ python3 ./paging-policy.py -a 0,1,2,3,4,5,0,1,2,3,4,5 -C 5 -c
ARG addresses 0,1,2,3,4,5,0,1,2,3,4,5
ARG addressfile
ARG numaddrs 10
ARG policy FIFO
ARG clockbits 2
ARG cachesize 5
ARG maxpage 10
ARG seed 0
ARG notrace False

Solving...

Access: 0  MISS FirstIn ->          [0] <- Lastin  Replaced:- [Hits:0 Misses:1]
Access: 1  MISS FirstIn ->       [0, 1] <- Lastin  Replaced:- [Hits:0 Misses:2]
Access: 2  MISS FirstIn ->    [0, 1, 2] <- Lastin  Replaced:- [Hits:0 Misses:3]
Access: 3  MISS FirstIn -> [0, 1, 2, 3] <- Lastin  Replaced:- [Hits:0 Misses:4]
Access: 4  MISS FirstIn -> [0, 1, 2, 3, 4] <- Lastin  Replaced:- [Hits:0 Misses:5]
Access: 5  MISS FirstIn -> [1, 2, 3, 4, 5] <- Lastin  Replaced:0 [Hits:0 Misses:6]
Access: 0  MISS FirstIn -> [2, 3, 4, 5, 0] <- Lastin  Replaced:1 [Hits:0 Misses:7]
Access: 1  MISS FirstIn -> [3, 4, 5, 0, 1] <- Lastin  Replaced:2 [Hits:0 Misses:8]
Access: 2  MISS FirstIn -> [4, 5, 0, 1, 2] <- Lastin  Replaced:3 [Hits:0 Misses:9]
Access: 3  MISS FirstIn -> [5, 0, 1, 2, 3] <- Lastin  Replaced:4 [Hits:0 Misses:10]
Access: 4  MISS FirstIn -> [0, 1, 2, 3, 4] <- Lastin  Replaced:5 [Hits:0 Misses:11]
Access: 5  MISS FirstIn -> [1, 2, 3, 4, 5] <- Lastin  Replaced:0 [Hits:0 Misses:12]

FINALSTATS hits 0   misses 12   hitrate 0.00
```

LRU:

```
burak@ubuntu:~/Desktop/ostep-homework-master/vm-beyondphys-policy$ python3 ./paging-policy.py -a 0,1,2,3,4,5,0,1,2,3,4,5 -C 5 -c -p LRU
ARG addresses 0,1,2,3,4,5,0,1,2,3,4,5
ARG addressfile
ARG numaddrs 10
ARG policy LRU
ARG clockbits 2
ARG cachesize 5
ARG maxpage 10
ARG seed 0
ARG notrace False

Solving...

Access: 0  MISS LRU ->          [0] <- MRU Replaced:- [Hits:0 Misses:1]
Access: 1  MISS LRU ->       [0, 1] <- MRU Replaced:- [Hits:0 Misses:2]
Access: 2  MISS LRU ->    [0, 1, 2] <- MRU Replaced:- [Hits:0 Misses:3]
Access: 3  MISS LRU -> [0, 1, 2, 3] <- MRU Replaced:- [Hits:0 Misses:4]
Access: 4  MISS LRU -> [0, 1, 2, 3, 4] <- MRU Replaced:- [Hits:0 Misses:5]
Access: 5  MISS LRU -> [1, 2, 3, 4, 5] <- MRU Replaced:0 [Hits:0 Misses:6]
Access: 0  MISS LRU -> [2, 3, 4, 5, 0] <- MRU Replaced:1 [Hits:0 Misses:7]
Access: 1  MISS LRU -> [3, 4, 5, 0, 1] <- MRU Replaced:2 [Hits:0 Misses:8]
Access: 2  MISS LRU -> [4, 5, 0, 1, 2] <- MRU Replaced:3 [Hits:0 Misses:9]
Access: 3  MISS LRU -> [5, 0, 1, 2, 3] <- MRU Replaced:4 [Hits:0 Misses:10]
Access: 4  MISS LRU -> [0, 1, 2, 3, 4] <- MRU Replaced:5 [Hits:0 Misses:11]
Access: 5  MISS LRU -> [1, 2, 3, 4, 5] <- MRU Replaced:0 [Hits:0 Misses:12]

FINALSTATS hits 0   misses 12   hitrate 0.00
```

MRU:

```
burak@ubuntu:~/Desktop/ostep-homework-master/vm-beyondphys-policy$ python3 ./paging-policy.py -a 0,1,2,3,4,5,4,5,4,5,4,5 -C 5 -c -p MRU
ARG addresses 0,1,2,3,4,5,4,5,4,5,4,5
ARG addressfile
ARG numaddrs 10
ARG policy MRU
ARG clockbits 2
ARG cachesize 5
ARG maxpage 10
ARG seed 0
ARG notrace False

Solving...

Access: 0  MISS LRU ->          [0] <- MRU Replaced:- [Hits:0 Misses:1]
Access: 1  MISS LRU ->       [0, 1] <- MRU Replaced:- [Hits:0 Misses:2]
Access: 2  MISS LRU ->    [0, 1, 2] <- MRU Replaced:- [Hits:0 Misses:3]
Access: 3  MISS LRU -> [0, 1, 2, 3] <- MRU Replaced:- [Hits:0 Misses:4]
Access: 4  MISS LRU -> [0, 1, 2, 3, 4] <- MRU Replaced:- [Hits:0 Misses:5]
Access: 5  MISS LRU -> [0, 1, 2, 3, 5] <- MRU Replaced:4 [Hits:0 Misses:6]
Access: 4  MISS LRU -> [0, 1, 2, 3, 4] <- MRU Replaced:5 [Hits:0 Misses:7]
Access: 5  MISS LRU -> [0, 1, 2, 3, 5] <- MRU Replaced:4 [Hits:0 Misses:8]
Access: 4  MISS LRU -> [0, 1, 2, 3, 4] <- MRU Replaced:5 [Hits:0 Misses:9]
Access: 5  MISS LRU -> [0, 1, 2, 3, 5] <- MRU Replaced:4 [Hits:0 Misses:10]
Access: 4  MISS LRU -> [0, 1, 2, 3, 4] <- MRU Replaced:5 [Hits:0 Misses:11]
Access: 5  MISS LRU -> [0, 1, 2, 3, 5] <- MRU Replaced:4 [Hits:0 Misses:12]

FINALSTATS hits 0   misses 12   hitrate 0.00
```

Önbellek boyutumuzu 6 ya çıkarıp OPT'ye yaklaştığımızı görebiliriz:

OPT:

```
burak@ubuntu:~/Desktop/ostep-homework-master/vm-beyondphys-policy$ python3 ./paging-policy.py -a 0,1,2,3,4,5,0,1,2,3,4,5 -C 6 -c -p OPT
ARG addresses 0,1,2,3,4,5,0,1,2,3,4,5
ARG addressfile
ARG numaddrs 10
ARG policy OPT
ARG clockbits 2
ARG cachesize 6
ARG maxpage 10
ARG seed 0
ARG notrace False

Solving...

Access: 0  MISS Left  ->          [0] <- Right Replaced:- [Hits:0 Misses:1]
Access: 1  MISS Left  ->       [0, 1] <- Right Replaced:- [Hits:0 Misses:2]
Access: 2  MISS Left  ->    [0, 1, 2] <- Right Replaced:- [Hits:0 Misses:3]
Access: 3  MISS Left  -> [0, 1, 2, 3] <- Right Replaced:- [Hits:0 Misses:4]
Access: 4  MISS Left  -> [0, 1, 2, 3, 4] <- Right Replaced:- [Hits:0 Misses:5]
Access: 5  MISS Left  -> [0, 1, 2, 3, 4, 5] <- Right Replaced:- [Hits:0 Misses:6]
Access: 0  HIT  Left  -> [0, 1, 2, 3, 4, 5] <- Right Replaced:- [Hits:1 Misses:6]
Access: 1  HIT  Left  -> [0, 1, 2, 3, 4, 5] <- Right Replaced:- [Hits:2 Misses:6]
Access: 2  HIT  Left  -> [0, 1, 2, 3, 4, 5] <- Right Replaced:- [Hits:3 Misses:6]
Access: 3  HIT  Left  -> [0, 1, 2, 3, 4, 5] <- Right Replaced:- [Hits:4 Misses:6]
Access: 4  HIT  Left  -> [0, 1, 2, 3, 4, 5] <- Right Replaced:- [Hits:5 Misses:6]
Access: 5  HIT  Left  -> [0, 1, 2, 3, 4, 5] <- Right Replaced:- [Hits:6 Misses:6]

FINALSTATS hits 6   misses 6   hitrate 50.00
```

FIFO:

```
burak@ubuntu:~/Desktop/ostep-homework-master/vm-beyondphys-policy$ python3 ./paging-policy.py -a 0,1,2,3,4,5,0,1,2,3,4,5 -C 6 -c
ARG addresses 0,1,2,3,4,5,0,1,2,3,4,5
ARG addressfile
ARG numaddrs 10
ARG policy FIFO
ARG clockbits 2
ARG cachesize 6
ARG maxpage 10
ARG seed 0
ARG notrace False

Solving...

Access: 0  MISS FirstIn ->          [0] <- Lastin  Replaced:- [Hits:0 Misses:1]
Access: 1  MISS FirstIn ->       [0, 1] <- Lastin  Replaced:- [Hits:0 Misses:2]
Access: 2  MISS FirstIn ->    [0, 1, 2] <- Lastin  Replaced:- [Hits:0 Misses:3]
Access: 3  MISS FirstIn -> [0, 1, 2, 3] <- Lastin  Replaced:- [Hits:0 Misses:4]
Access: 4  MISS FirstIn -> [0, 1, 2, 3, 4] <- Lastin  Replaced:- [Hits:0 Misses:5]
Access: 5  MISS FirstIn -> [0, 1, 2, 3, 4, 5] <- Lastin  Replaced:- [Hits:0 Misses:6]
Access: 0  HIT  FirstIn -> [0, 1, 2, 3, 4, 5] <- Lastin  Replaced:- [Hits:1 Misses:6]
Access: 1  HIT  FirstIn -> [0, 1, 2, 3, 4, 5] <- Lastin  Replaced:- [Hits:2 Misses:6]
Access: 2  HIT  FirstIn -> [0, 1, 2, 3, 4, 5] <- Lastin  Replaced:- [Hits:3 Misses:6]
Access: 3  HIT  FirstIn -> [0, 1, 2, 3, 4, 5] <- Lastin  Replaced:- [Hits:4 Misses:6]
Access: 4  HIT  FirstIn -> [0, 1, 2, 3, 4, 5] <- Lastin  Replaced:- [Hits:5 Misses:6]
Access: 5  HIT  FirstIn -> [0, 1, 2, 3, 4, 5] <- Lastin  Replaced:- [Hits:6 Misses:6]

FINALSTATS hits 6   misses 6   hitrate 50.00
```

LRU:

```
burak@ubuntu:~/Desktop/ostep-homework-master/vm-beyondphys-policy$ python3 ./paging-policy.py -a 0,1,2,3,4,5,0,1,2,3,4,5 -C 6 -c -p LRU
ARG addresses 0,1,2,3,4,5,0,1,2,3,4,5
ARG addressfile
ARG numaddrs 10
ARG policy LRU
ARG clockbits 2
ARG cachesize 6
ARG maxpage 10
ARG seed 0
ARG notrace False

Solving...

Access: 0  MISS LRU ->          [0] <- MRU Replaced:- [Hits:0 Misses:1]
Access: 1  MISS LRU ->       [0, 1] <- MRU Replaced:- [Hits:0 Misses:2]
Access: 2  MISS LRU ->    [0, 1, 2] <- MRU Replaced:- [Hits:0 Misses:3]
Access: 3  MISS LRU -> [0, 1, 2, 3] <- MRU Replaced:- [Hits:0 Misses:4]
Access: 4  MISS LRU -> [0, 1, 2, 3, 4] <- MRU Replaced:- [Hits:0 Misses:5]
Access: 5  MISS LRU -> [0, 1, 2, 3, 4, 5] <- MRU Replaced:- [Hits:0 Misses:6]
Access: 0  HIT  LRU -> [1, 2, 3, 4, 5, 0] <- MRU Replaced:- [Hits:1 Misses:6]
Access: 1  HIT  LRU -> [2, 3, 4, 5, 0, 1] <- MRU Replaced:- [Hits:2 Misses:6]
Access: 2  HIT  LRU -> [3, 4, 5, 0, 1, 2] <- MRU Replaced:- [Hits:3 Misses:6]
Access: 3  HIT  LRU -> [4, 5, 0, 1, 2, 3] <- MRU Replaced:- [Hits:4 Misses:6]
Access: 4  HIT  LRU -> [5, 0, 1, 2, 3, 4] <- MRU Replaced:- [Hits:5 Misses:6]
Access: 5  HIT  LRU -> [0, 1, 2, 3, 4, 5] <- MRU Replaced:- [Hits:6 Misses:6]

FINALSTATS hits 6   misses 6   hitrate 50.00
```

MRU:

```
burak@ubuntu:~/Desktop/ostep-homework-master/vm-beyondphys-policy$ python3 ./paging-policy.py -a 0,1,2,3,4,5,4,5,4,5,4,5 -C 6 -c -p MRU
ARG addresses 0,1,2,3,4,5,4,5,4,5,4,5
ARG addressfile
ARG numaddrs 10
ARG policy MRU
ARG clockbits 2
ARG cachesize 6
ARG maxpage 10
ARG seed 0
ARG notrace False

Solving...

Access: 0  MISS LRU ->          [0] <- MRU Replaced:- [Hits:0 Misses:1]
Access: 1  MISS LRU ->       [0, 1] <- MRU Replaced:- [Hits:0 Misses:2]
Access: 2  MISS LRU ->    [0, 1, 2] <- MRU Replaced:- [Hits:0 Misses:3]
Access: 3  MISS LRU -> [0, 1, 2, 3] <- MRU Replaced:- [Hits:0 Misses:4]
Access: 4  MISS LRU -> [0, 1, 2, 3, 4] <- MRU Replaced:- [Hits:0 Misses:5]
Access: 5  MISS LRU -> [0, 1, 2, 3, 4, 5] <- MRU Replaced:- [Hits:0 Misses:6]
Access: 4  HIT  LRU -> [0, 1, 2, 3, 5, 4] <- MRU Replaced:- [Hits:1 Misses:6]
Access: 5  HIT  LRU -> [0, 1, 2, 3, 4, 5] <- MRU Replaced:- [Hits:2 Misses:6]
Access: 4  HIT  LRU -> [0, 1, 2, 3, 5, 4] <- MRU Replaced:- [Hits:3 Misses:6]
Access: 5  HIT  LRU -> [0, 1, 2, 3, 4, 5] <- MRU Replaced:- [Hits:4 Misses:6]
Access: 4  HIT  LRU -> [0, 1, 2, 3, 5, 4] <- MRU Replaced:- [Hits:5 Misses:6]
Access: 5  HIT  LRU -> [0, 1, 2, 3, 4, 5] <- MRU Replaced:- [Hits:6 Misses:6]

FINALSTATS hits 6   misses 6   hitrate 50.00
```

3. Rastgele bir iz oluşturun (python veya perl kullanın). Farklı politikaların böyle bir iz üzerinde nasıl performans göstermesini beklersiniz?

FIFO

1 isabet 9 ıska isabet oranı %10

```
burak@ubuntu:~/Desktop/ostep-homework-master/vm-beyondphys-policy$ python3 ./paging-policy.py -s 0 -n 10 -c
ARG addresses -1
ARG addressfile
ARG numaddrs 10
ARG policy FIFO
ARG clockbits 2
ARG cachesize 3
ARG maxpage 10
ARG seed 0
ARG notrace False

Solving...

Access: 8  MISS FirstIn ->          [8] <- Lastin  Replaced:- [Hits:0 Misses:1]
Access: 7  MISS FirstIn ->       [8, 7] <- Lastin  Replaced:- [Hits:0 Misses:2]
Access: 4  MISS FirstIn ->    [8, 7, 4] <- Lastin  Replaced:- [Hits:0 Misses:3]
Access: 2  MISS FirstIn ->    [7, 4, 2] <- Lastin  Replaced:8 [Hits:0 Misses:4]
Access: 5  MISS FirstIn ->    [4, 2, 5] <- Lastin  Replaced:7 [Hits:0 Misses:5]
Access: 4  HIT  FirstIn ->    [4, 2, 5] <- Lastin  Replaced:- [Hits:1 Misses:5]
Access: 7  MISS FirstIn ->    [2, 5, 7] <- Lastin  Replaced:4 [Hits:1 Misses:6]
Access: 3  MISS FirstIn ->    [5, 7, 3] <- Lastin  Replaced:2 [Hits:1 Misses:7]
Access: 4  MISS FirstIn ->    [7, 3, 4] <- Lastin  Replaced:5 [Hits:1 Misses:8]
Access: 5  MISS FirstIn ->    [3, 4, 5] <- Lastin  Replaced:7 [Hits:1 Misses:9]

FINALSTATS hits 1   misses 9   hitrate 10.00
```

LRU

2 isabet 8 ıska %20 isabet oranı

```
burak@ubuntu:~/Desktop/ostep-homework-master/vm-beyondphys-policy$ python3 ./paging-policy.py -s 0 -n 10 -c -p LRU
ARG addresses -1
ARG addressfile
ARG numaddrs 10
ARG policy LRU
ARG clockbits 2
ARG cachesize 3
ARG maxpage 10
ARG seed 0
ARG notrace False

Solving...

Access: 8  MISS LRU ->          [8] <- MRU Replaced:- [Hits:0 Misses:1]
Access: 7  MISS LRU ->       [8, 7] <- MRU Replaced:- [Hits:0 Misses:2]
Access: 4  MISS LRU ->    [8, 7, 4] <- MRU Replaced:- [Hits:0 Misses:3]
Access: 2  MISS LRU ->    [7, 4, 2] <- MRU Replaced:8 [Hits:0 Misses:4]
Access: 5  MISS LRU ->    [4, 2, 5] <- MRU Replaced:7 [Hits:0 Misses:5]
Access: 4  HIT  LRU ->    [2, 5, 4] <- MRU Replaced:- [Hits:1 Misses:5]
Access: 7  MISS LRU ->    [5, 4, 7] <- MRU Replaced:2 [Hits:1 Misses:6]
Access: 3  MISS LRU ->    [4, 7, 3] <- MRU Replaced:5 [Hits:1 Misses:7]
Access: 4  HIT  LRU ->    [7, 3, 4] <- MRU Replaced:- [Hits:2 Misses:7]
Access: 5  MISS LRU ->    [3, 4, 5] <- MRU Replaced:7 [Hits:2 Misses:8]

FINALSTATS hits 2   misses 8   hitrate 20.00
```

OPT

4 isabet 6 ıska %40 isabet oranı

```
burak@ubuntu:~/Desktop/ostep-homework-master/vm-beyondphys-policy$ python3 ./paging-policy.py -s 0 -n 10 -c -p OPT
ARG addresses -1
ARG addressfile
ARG numaddrs 10
ARG policy OPT
ARG clockbits 2
ARG cachesize 3
ARG maxpage 10
ARG seed 0
ARG notrace False

Solving...

Access: 8  MISS Left  ->         [8] <- Right Replaced:- [Hits:0 Misses:1]
Access: 7  MISS Left  ->      [8, 7] <- Right Replaced:- [Hits:0 Misses:2]
Access: 4  MISS Left  ->   [8, 7, 4] <- Right Replaced:- [Hits:0 Misses:3]
Access: 2  MISS Left  ->   [7, 4, 2] <- Right Replaced:8 [Hits:0 Misses:4]
Access: 5  MISS Left  ->   [7, 4, 5] <- Right Replaced:2 [Hits:0 Misses:5]
Access: 4  HIT  Left  ->   [7, 4, 5] <- Right Replaced:- [Hits:1 Misses:5]
Access: 7  HIT  Left  ->   [7, 4, 5] <- Right Replaced:- [Hits:2 Misses:5]
Access: 3  MISS Left  ->   [4, 5, 3] <- Right Replaced:7 [Hits:2 Misses:6]
Access: 4  HIT  Left  ->   [4, 5, 3] <- Right Replaced:- [Hits:3 Misses:6]
Access: 5  HIT  Left  ->   [4, 5, 3] <- Right Replaced:- [Hits:4 Misses:6]

FINALSTATS hits 4   misses 6   hitrate 40.00
```

RAND:

0 isabet 10 ıska %0 ıska oranı

```
burak@ubuntu:~/Desktop/ostep-homework-master/vm-beyondphys-policy$ python3 ./paging-policy.py -s 0 -n 10 -c -p RAND
ARG addresses -1
ARG addressfile
ARG numaddrs 10
ARG policy RAND
ARG clockbits 2
ARG cachesize 3
ARG maxpage 10
ARG seed 0
ARG notrace False

Solving...

Access: 8  MISS Left  ->         [8] <- Right Replaced:- [Hits:0 Misses:1]
Access: 7  MISS Left  ->      [8, 7] <- Right Replaced:- [Hits:0 Misses:2]
Access: 4  MISS Left  ->   [8, 7, 4] <- Right Replaced:- [Hits:0 Misses:3]
Access: 2  MISS Left  ->   [8, 7, 2] <- Right Replaced:4 [Hits:0 Misses:4]
Access: 5  MISS Left  ->   [8, 2, 5] <- Right Replaced:7 [Hits:0 Misses:5]
Access: 4  MISS Left  ->   [2, 5, 4] <- Right Replaced:8 [Hits:0 Misses:6]
Access: 7  MISS Left  ->   [2, 5, 7] <- Right Replaced:4 [Hits:0 Misses:7]
Access: 3  MISS Left  ->   [2, 7, 3] <- Right Replaced:5 [Hits:0 Misses:8]
Access: 4  MISS Left  ->   [7, 3, 4] <- Right Replaced:2 [Hits:0 Misses:9]
Access: 5  MISS Left  ->   [7, 3, 5] <- Right Replaced:4 [Hits:0 Misses:10]

FINALSTATS hits 0   misses 10   hitrate 0.00
```

CLOCK:

1 isabet 9 ıska %10 isabet oranı

```
burak@ubuntu:~/Desktop/ostep-homework-master/vm-beyondphys-policy$ python3 ./paging-policy.py -s 0 -n 10 -c -p CLOCK
ARG addresses -1
ARG addressfile
ARG numaddrs 10
ARG policy CLOCK
ARG clockbits 2
ARG cachesize 3
ARG maxpage 10
ARG seed 0
ARG notrace False

Solving...

Access: 8  MISS Left  ->         [8] <- Right Replaced:- [Hits:0 Misses:1]
Access: 7  MISS Left  ->      [8, 7] <- Right Replaced:- [Hits:0 Misses:2]
Access: 4  MISS Left  ->   [8, 7, 4] <- Right Replaced:- [Hits:0 Misses:3]
Access: 2  MISS Left  ->   [8, 7, 2] <- Right Replaced:4 [Hits:0 Misses:4]
Access: 5  MISS Left  ->   [8, 2, 5] <- Right Replaced:7 [Hits:0 Misses:5]
Access: 4  MISS Left  ->   [2, 5, 4] <- Right Replaced:8 [Hits:0 Misses:6]
Access: 7  MISS Left  ->   [2, 5, 7] <- Right Replaced:4 [Hits:0 Misses:7]
Access: 3  MISS Left  ->   [2, 5, 3] <- Right Replaced:7 [Hits:0 Misses:8]
Access: 4  MISS Left  ->   [2, 5, 4] <- Right Replaced:3 [Hits:0 Misses:9]
Access: 5  HIT  Left  ->   [2, 5, 4] <- Right Replaced:- [Hits:1 Misses:9]

FINALSTATS hits 1   misses 9   hitrate 10.00
```

FIFO = %10

LRU = %20

OPT = %40

RAND = %0

CLOCK = %10

*OPT>LRU>FIFO=CLOCK>RAND

*FIFO ilk getirilen sayfayı dışarı atar; eğer bu sayfa üzerinde önemli kod veya veri yapıları olan bir sayfaysa, yakında tekrar sayfalanacak olsa bile yine de dışarı atılır

*LRU ise daha akıllı bir şekilde çalışır bir sayfayı ilk kez değiştirmesi gerektiğinde yakın zamanda erişilen sayfaları değil diğer sayfayı çıkarır.LRU geçmişe dayalı kararlarının doğruluğunu kanıtlamıştır.

*RAND verilerin kullanım veya erişim modellerini dikkate almaz. Bunun yerine, verileri rastgele depolar ve çıkarır.

*OPT ileriye bakar en son hangisine ulaşılmışssa onu atar.

*CLOCK, sistemin performansını artırmak için kullanılan bir bilgisayarın belleğindeki verileri düzenleme yöntemidir. LRU ilkesinin bir çeşididir ve verileri bellekte depolamak ve yönetmek için başvuru bitleri ve dairesel bir kuyruk kullanır.

4. Şimdi biraz yerelliğe sahip bir iz oluşturun. Böyle bir izi nasıl oluşturabilirsiniz? LRU üzerinde nasıl bir performans sergiliyor? LRU, RAND'dan ne kadar daha iyi? SAAT(CLOOK) nasıl çalışıyor? Farklı saat bit sayılarına sahip SAAT nasıl olur?

```
burak@ubuntu:~/Desktop/ostep-homework-master/vm-beyondphys-policy$ python3 ./paging-policy.py -a 3,0,6,6,6,6,7,0,6,6 -p LRU -c
ARG addresses 3,0,6,6,6,6,7,0,6,6
ARG addressfile
ARG numaddrs 10
ARG policy LRU
ARG clockbits 2
ARG cachesize 3
ARG maxpage 10
ARG seed 0
ARG notrace False

Solving...

Access: 3  MISS LRU ->         [3] <- MRU Replaced:- [Hits:0 Misses:1]
Access: 0  MISS LRU ->      [3, 0] <- MRU Replaced:- [Hits:0 Misses:2]
Access: 6  MISS LRU ->   [3, 0, 6] <- MRU Replaced:- [Hits:0 Misses:3]
Access: 6  HIT  LRU ->   [3, 0, 6] <- MRU Replaced:- [Hits:1 Misses:3]
Access: 6  HIT  LRU ->   [3, 0, 6] <- MRU Replaced:- [Hits:2 Misses:3]
Access: 6  HIT  LRU ->   [3, 0, 6] <- MRU Replaced:- [Hits:3 Misses:3]
Access: 7  MISS LRU ->   [0, 6, 7] <- MRU Replaced:3 [Hits:3 Misses:4]
Access: 0  HIT  LRU ->   [6, 7, 0] <- MRU Replaced:- [Hits:4 Misses:4]
Access: 6  HIT  LRU ->   [7, 0, 6] <- MRU Replaced:- [Hits:5 Misses:4]
Access: 6  HIT  LRU ->   [7, 0, 6] <- MRU Replaced:- [Hits:6 Misses:4]

FINALSTATS hits 6   misses 4   hitrate 60.00
```

```
burak@ubuntu:~/Desktop/ostep-homework-master/vm-beyondphys-policy$ python3 ./paging-policy.py -a 3,0,6,6,6,6,7,0,6,6 -p RAND -c
ARG addresses 3,0,6,6,6,6,7,0,6,6
ARG addressfile
ARG numaddrs 10
ARG policy RAND
ARG clockbits 2
ARG cachesize 3
ARG maxpage 10
ARG seed 0
ARG notrace False

Solving...

Access: 3  MISS Left  ->         [3] <- Right Replaced:- [Hits:0 Misses:1]
Access: 0  MISS Left  ->      [3, 0] <- Right Replaced:- [Hits:0 Misses:2]
Access: 6  MISS Left  ->   [3, 0, 6] <- Right Replaced:- [Hits:0 Misses:3]
Access: 6  HIT  Left  ->   [3, 0, 6] <- Right Replaced:- [Hits:1 Misses:3]
Access: 6  HIT  Left  ->   [3, 0, 6] <- Right Replaced:- [Hits:2 Misses:3]
Access: 6  HIT  Left  ->   [3, 0, 6] <- Right Replaced:- [Hits:3 Misses:3]
Access: 7  MISS Left  ->   [3, 0, 7] <- Right Replaced:6 [Hits:3 Misses:4]
Access: 0  HIT  Left  ->   [3, 0, 7] <- Right Replaced:- [Hits:4 Misses:4]
Access: 6  MISS Left  ->   [3, 0, 6] <- Right Replaced:7 [Hits:4 Misses:5]
Access: 6  HIT  Left  ->   [3, 0, 6] <- Right Replaced:- [Hits:5 Misses:5]

FINALSTATS hits 5   misses 5   hitrate 50.00
```

```
burak@ubuntu:~/Desktop/ostep-homework-master/vm-beyondphys-policy$ python3 ./paging-policy.py -a 3,0,6,6,6,6,7,0,6,6 -p CLOCK -c -b 2
ARG addresses 3,0,6,6,6,6,7,0,6,6
ARG addressfile
ARG numaddrs 10
ARG policy CLOCK
ARG clockbits 2
ARG cachesize 3
ARG maxpage 10
ARG seed 0
ARG notrace False

Solving...

Access: 3  MISS Left  ->         [3] <- Right Replaced:- [Hits:0 Misses:1]
Access: 0  MISS Left  ->      [3, 0] <- Right Replaced:- [Hits:0 Misses:2]
Access: 6  MISS Left  ->   [3, 0, 6] <- Right Replaced:- [Hits:0 Misses:3]
Access: 6  HIT  Left  ->   [3, 0, 6] <- Right Replaced:- [Hits:1 Misses:3]
Access: 6  HIT  Left  ->   [3, 0, 6] <- Right Replaced:- [Hits:2 Misses:3]
Access: 6  HIT  Left  ->   [3, 0, 6] <- Right Replaced:- [Hits:3 Misses:3]
Access: 7  MISS Left  ->   [3, 6, 7] <- Right Replaced:0 [Hits:3 Misses:4]
Access: 0  MISS Left  ->   [3, 7, 0] <- Right Replaced:6 [Hits:3 Misses:5]
Access: 6  MISS Left  ->   [7, 0, 6] <- Right Replaced:3 [Hits:3 Misses:6]
Access: 6  HIT  Left  ->   [7, 0, 6] <- Right Replaced:- [Hits:4 Misses:6]

FINALSTATS hits 4   misses 6   hitrate 40.00
```

```
burak@ubuntu:~/Desktop/ostep-homework-master/vm-beyondphys-policy$ python3 ./paging-policy.py -a 3,0,6,6,6,6,7,0,6,6 -p CLOCK -c -b 0
ARG addresses 3,0,6,6,6,6,7,0,6,6
ARG addressfile
ARG numaddrs 10
ARG policy CLOCK
ARG clockbits 0
ARG cachesize 3
ARG maxpage 10
ARG seed 0
ARG notrace False

Solving...

Access: 3  MISS Left  ->         [3] <- Right Replaced:- [Hits:0 Misses:1]
Access: 0  MISS Left  ->      [3, 0] <- Right Replaced:- [Hits:0 Misses:2]
Access: 6  MISS Left  ->   [3, 0, 6] <- Right Replaced:- [Hits:0 Misses:3]
Access: 6  HIT  Left  ->   [3, 0, 6] <- Right Replaced:- [Hits:1 Misses:3]
Access: 6  HIT  Left  ->   [3, 0, 6] <- Right Replaced:- [Hits:2 Misses:3]
Access: 6  HIT  Left  ->   [3, 0, 6] <- Right Replaced:- [Hits:3 Misses:3]
Access: 7  MISS Left  ->   [3, 0, 7] <- Right Replaced:6 [Hits:3 Misses:4]
Access: 0  HIT  Left  ->   [3, 0, 7] <- Right Replaced:- [Hits:4 Misses:4]
Access: 6  MISS Left  ->   [3, 7, 6] <- Right Replaced:0 [Hits:4 Misses:5]
Access: 6  HIT  Left  ->   [3, 7, 6] <- Right Replaced:- [Hits:5 Misses:5]

FINALSTATS hits 5   misses 5   hitrate 50.00
```

```
burak@ubuntu:~/Desktop/ostep-homework-master/vm-beyondphys-policy$ python3 ./paging-policy.py -a 3,0,6,6,6,6,7,0,6,6 -p CLOCK -c -b 1
ARG addresses 3,0,6,6,6,6,7,0,6,6
ARG addressfile
ARG numaddrs 10
ARG policy CLOCK
ARG clockbits 1
ARG cachesize 3
ARG maxpage 10
ARG seed 0
ARG notrace False

Solving...

Access: 3  MISS Left  ->         [3] <- Right Replaced:- [Hits:0 Misses:1]
Access: 0  MISS Left  ->      [3, 0] <- Right Replaced:- [Hits:0 Misses:2]
Access: 6  MISS Left  ->   [3, 0, 6] <- Right Replaced:- [Hits:0 Misses:3]
Access: 6  HIT  Left  ->   [3, 0, 6] <- Right Replaced:- [Hits:1 Misses:3]
Access: 6  HIT  Left  ->   [3, 0, 6] <- Right Replaced:- [Hits:2 Misses:3]
Access: 6  HIT  Left  ->   [3, 0, 6] <- Right Replaced:- [Hits:3 Misses:3]
Access: 7  MISS Left  ->   [3, 0, 7] <- Right Replaced:6 [Hits:3 Misses:4]
Access: 0  HIT  Left  ->   [3, 0, 7] <- Right Replaced:- [Hits:4 Misses:4]
Access: 6  MISS Left  ->   [3, 7, 6] <- Right Replaced:0 [Hits:4 Misses:5]
Access: 6  HIT  Left  ->   [3, 7, 6] <- Right Replaced:- [Hits:5 Misses:5]

FINALSTATS hits 5   misses 5   hitrate 50.00
```

```
burak@ubuntu:~/Desktop/ostep-homework-master/vm-beyondphys-policy$ python3 ./paging-policy.py -a 3,0,6,6,6,6,7,0,6,6 -p CLOCK -c -b 3
ARG addresses 3,0,6,6,6,6,7,0,6,6
ARG addressfile
ARG numaddrs 10
ARG policy CLOCK
ARG clockbits 3
ARG cachesize 3
ARG maxpage 10
ARG seed 0
ARG notrace False

Solving...

Access: 3  MISS Left  ->         [3] <- Right Replaced:- [Hits:0 Misses:1]
Access: 0  MISS Left  ->      [3, 0] <- Right Replaced:- [Hits:0 Misses:2]
Access: 6  MISS Left  ->   [3, 0, 6] <- Right Replaced:- [Hits:0 Misses:3]
Access: 6  HIT  Left  ->   [3, 0, 6] <- Right Replaced:- [Hits:1 Misses:3]
Access: 6  HIT  Left  ->   [3, 0, 6] <- Right Replaced:- [Hits:2 Misses:3]
Access: 6  HIT  Left  ->   [3, 0, 6] <- Right Replaced:- [Hits:3 Misses:3]
Access: 7  MISS Left  ->   [3, 6, 7] <- Right Replaced:0 [Hits:3 Misses:4]
Access: 0  MISS Left  ->   [6, 7, 0] <- Right Replaced:3 [Hits:3 Misses:5]
Access: 6  HIT  Left  ->   [6, 7, 0] <- Right Replaced:- [Hits:4 Misses:5]
Access: 6  HIT  Left  ->   [6, 7, 0] <- Right Replaced:- [Hits:5 Misses:5]

FINALSTATS hits 5   misses 5   hitrate 50.00
```

5. Gerçek bir uygulama oluşturmak ve sanal bir sayfa referans akışı oluşturmak için valgrind gibi bir program kullanın. Örneğin, valgrind --tool=lackey --trace-mem=yes ls çalıştırıldığında, ls programı tarafından yapılan her komut ve veri referansının neredeyse eksiksiz bir referans izi çıkarılacaktır. Bunu yukarıdaki simülatör için kullanışlı hale getirmek için, önce her sanal bellek referansını sanal bir sayfa numarası referansına dönüştürmeniz gerekir (ofseti maskeleyerek ve ortaya çıkan bitleri aşağı kaydırarak yapılır). Taleplerin büyük bir kısmını karşılamak için uygulama iziniz için ne kadar büyüklükte bir önbellek gerekir? Önbelleğin boyutu arttıkça çalışma kümesinin bir grafiğini çizin.

valgrind --tool=lackey --trace-mem=yes ls

komutunu çalıştırdık:

```
==2837==
==2837== Counted 0 calls to main()
==2837==
==2837== Jccs:
==2837==   total:         100,026
==2837==   taken:         38,982 (39%)
==2837==
==2837== Executed:
==2837==   SBs entered:   105,082
==2837==   SBs completed: 71,310
==2837==   guest instrs:  532,650
==2837==   IRStmts:       3,200,469
==2837==
==2837== Ratios:
==2837==   guest instrs : SB entered  = 50 : 10
==2837==        IRStmts : SB entered  = 304 : 10
==2837==        IRStmts : guest instr = 60 : 10
==2837==
==2837== Exit code:       0
```