37

Sabit disk sürücüleri

Son bölüm, bir I/O cihazının genel konseptini tanıttı ve işletim sisteminin böyle bir canavarla nasıl etkileşim kurabileceğini gösterdi. Bu bölümde, özellikle bir cihaz hakkında daha fazla ayrıntıya gireceğiz:**Sabit disk sürücüsü(Hard disk drive)**. Bu sürücüler, onlarca yıldır bilgisayar sistemlerinde kalıcı veri depolamanın ana biçimi olmuştur ve dosya sistemi teknolojisindeki (yakında)gelişimin çoğu, bunların davranışlarına dayanmaktadır. Bu nedenle, onu yöneten dosya sistemi yazılımını oluşturmadan önce bir diskin işleyişinin ayrıntılarını anlamaya değer. Bu ayrıntıların çoğu, Ruemmler ve Wilkes [RW92] ve Anderson, Dykes ve Riedel [ADR03] tarafından hazırlanan mükemmel makalelerde mevcuttur.

CRUX: Diskte veri nasıl saklanır ve bunlara erişilir
Modern sabit disk sürücüleri verileri nasıl depolar? Arayüz nedir? Veriler gerçekte nasıl düzenleniyor? Disk zamanlaması performansı nasıl artırır?

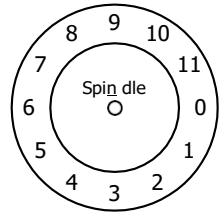
37.1 Arayüz

Modern bir disk sürücüsünün arayüzünü anlayarak başlayalım. Tüm modern sürücüler için temel arabirim basittir. Sürücü, her biri okunabilen veya yazılabilen çok sayıda sektörden (512-byte blocks) oluşur.Sektörler numaralandırılır 0 ile n-1 olan bir diskten sektörler. Böylece diskibir sektör dizisi olarak görebiliriz;0ile n-1 böylece **adres alanı(address space)** sürücünün.

Çok sektörlü işlemler mümkündür; aslında birçok dosya sistemi aynı anda4 KB (veya daha fazla) okur veya yazar. Ancak, diski güncellerken

sürücü üreticilerinin verdiği tek garanti, 512 baytlık tek bir yazmanın **atomik(atomic)** (yani ya tamamen tamamlanacak ya da hiç tamamlanmayacaktır); bu nedenle, zamansız bir güç kaybı meydana gelirse, daha büyük bir yazmanın yalnızca bir kısmı tamamlanabilir (bazen **yırtık yazma(torn write)** denir).

Çoğu disk sürücüsü istemcisinin yaptığı, ancak doğrudan arabirimde belirtilmeyen bazı varsayımlar vardır; Schlosser ve Ganger'in sahip olduğu



Şekil 37.1:**Tek İzli Bir Disk (A Disk With A Single Track)**

buna disk sürücülerinin "yazılı olmayan sözleşmesi" [SG04] adını verdi. özellikle, genellikle iki bloğa erişmenin[1] sürücünün adres alanı içinde birbirine yakın olmak, birbirinden uzak iki bloğa erişmekten daha hızlı olacaktır. Bitişik bir parçadaki (Veriler ve istatistikler (i.e.,a) yani sıralı bir okuma veya yazma) bloklara erişmenin en
hızlı erişim modu olduğu ve genellikle herhangi bir rasgele erişim modelinden çok daha hızlı olduğu da varsayılabilir.

37.2 Temel Geometri

Modern bir diskin bazı bileşenlerini anlamaya başlayalım.

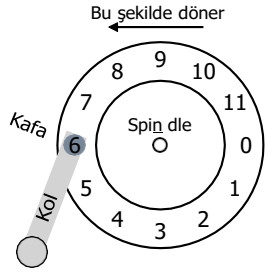
Servis tabağı(platter) ile başlıyoruz, manyetik değişikliklere neden olarak verilerin kalıcı olarak depolandığı dairesel bir sert yüzey. Bir diskin bir veya daha fazla plakası olabilir; her tabağın 2 kenarı vardır ve her birine bir denir **yüzey**. Bu plakalar genellikle bazı sert malzemelerden (alüminyum gibi) yapılır ve daha sonra sürücünün gücü kapalıyken bile bitleri kalıcı olarakdepolamasını sağlayan ince bir manyetik katmanla kaplanır.

Plakaların hepsi etrafında birbirine bağlıdır.**iğ**, plakaları (sürücü açıkken) sabit (sabit) bir hızda döndüren bir motora bağlıdır. Dönme hızı genellikle şu şekilde ölçülür: **dakikadaki dönüş sayısı (RPM)** ve tipik modern değerler 7.200 RPM ila 15.000 RPM aralığındadır. Genellikle tek bir dönüşün süresiyle ilgileneceğimizi unutmayın, örneğin, 10.000 RPM'de dönen bir sürücü, tek bir dönüşün yaklaşık 6 milisaniye (6 ms) sürdüğü anlamına gelir.

Veriler, sektörlerin eşmerkezli dairelerinde her yüzeyde kodlanmıştır; böyle bir eşmerkezli daireye a diyoruz **Izlemek**. Tek bir yüzey, bir insan saçı genişliğine sığan yüzlerce iz ile birlikte sıkıca bir araya getirilmiş binlerce ve binlerce iz içerir.

Yüzeyden okumak ve yazmak için, diskteki manyetik kalıpları algılamamıza (yani okumamıza) veya onlarda bir değişikliğe neden olmamıza (yani yazmamıza) izin veren bir mekanizmaya ihtiyacımız var. Bu okuma ve yazma işlemi,**disk kafası**; sürücünün yüzeyi başına böyle bir kafa vardır. Disk kafası tek bir **disk kolu**, kafayı istenen yolun üzerine konumlandırmak için yüzey boyunca hareket eder.

[1]Biz ve başkaları sıklıkla şu terimleri kullanırız:**engellemek** ve **sektör** okuyucunun bağlam başınatam olarak neyin kastedildiğini bileceğini varsayarsak. Bunun için üzgünüm!



Şekil 37.2:**Tek Parça Artı Bir Kafa**

37.3 Basit Bir Disk Sürücüsü

Her seferinde bir iz olacak bir model oluşturarak disklerin nasıl çalıştığını anlayalım. Tek izli basit bir diskimiz olduğunu varsayalım (Şekil 37.1). Bu iz, her biri 512 bayt boyutunda (bizim tipik sektör boyutumuz, geri çağırma) olan ve bu nedenle 0'dan 11'e kadar olan sayılarla adreslenen yalnızca 12 sektöre sahiptir. Burada sahip olduğumuz tek plaka, bir motorun bağlı olduğu iş mili etrafında döner .

Elbette parkur tek başına çok ilginç değil; bu sektörleri okuyabilmek veya yazabilmek istiyoruz ve bu nedenle şimdi gördüğümüz gibi bir disk koluna bağlı bir disk kafasına ihtiyacımız var (Şekil 37.2). Şekilde, kolun ucuna takılan disk kafası sektör 6'nın üzerinde konumlandırılmış ve yüzey saat yönünün tersine dönmektedir.

Tek İz Gecikmesi: Dönme Gecikmesi

Basit, tek izli diskimizde bir talebin nasıl işleneceğini anlamak için, şimdi blok 0'ı okuma talebi aldığımızı hayal edin. Disk bu talebe nasıl hizmet etmelidir?

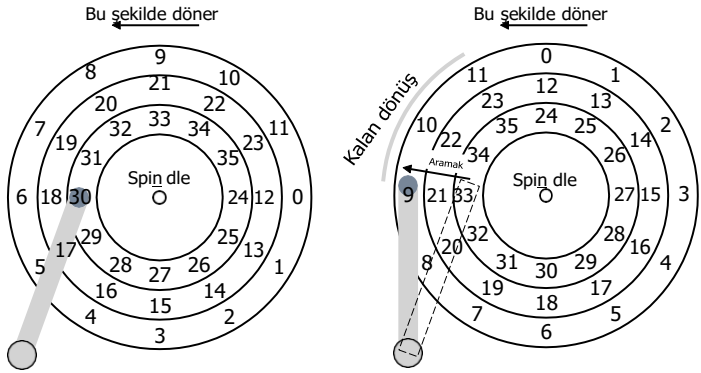
Basit diskimizde, diskin fazla bir şey yapması gerekmiyor. Özellikle, istenen sektörün disk kafasının altında dönmesini beklemesi gerekir. Bu bekleme, modern sürücülerde yeterince sık gerçekleşir ve G/Ç hizmet süresinin yeterince önemli bir bileşenidir ve özel bir adı vardır:**dönme gecikmesi** (Bazen **dönüş gecikmesi** kulağa tuhaf gelse de). Örnekte, eğer tamdönme gecikmesi R, disk bir dönüş gecikmesine maruz kalmalıdır yaklaşıkR0'ın okuma/yazma başlığının altına gelmesini beklemek (eğer 6'dan başlarsak). Bu tek yoldaki en kötü durum talebi, sektör 5'e yapılacak ve böyle bir talebe hizmet vermek için neredeyse tam bir dönüş gecikmesine neden olacaktır.

2

Birden Fazla Parça: Zaman Ara

Şimdiye kadar diskimizde çok gerçekçi olmayan tek bir iz var; modern disklerde elbette milyonlarca var. Şimdi her zamankinden biraz daha gerçekçi bir disk yüzeyine bakalım, bu üç izli (Şekil 37.3, sol).

Şekilde, kafa şu anda en içteki yolun (24 ila 35 arasındaki sektörleri içeren) üzerinde konumlanmıştır; sonraki parça şunları içerir:



Şekil 37.3:**Three Tracks Plus A Head (Sağ: Aramalı)**

sonraki sektör grubu (12'den 23'e) ve en dıştaki iz ilk sektörleri (0'dan 11'e) içerir.

Sürücünün belirli bir sektöre nasıl erişebileceğini anlamak için, artık uzak bir sektöre yönelik bir istekte, örneğin sektör 11'e yapılan bir okumada ne olacağını izliyoruz. Bu okumaya hizmet vermek için, sürücünün önce disk kolunu doğru konuma hareket ettirmesi gerekir. iz (bu durumda en dıştaki), olarak bilinen bir süreçte **aramak**. Döndürmelerle birlikte aramalar, en maliyetli disk işlemlerinden biridir.

Aramanın birçok aşaması olduğu belirtilmelidir: önce bir*hızlanma* disk kolu hareket ettikçe aşama; sonra*sahil şeridi*kol tam hızda hareket ederken, ardından *yavaşlama* kol yavaşlarken; en sonunda*yerleşim*kafa dikkatli bir şekilde doğru ray üzerinde konumlandırıldığından. bu **yerleşme zamanı** sürücünün doğru yolu bulacağından emin olması gerektiğinden (bunun yerine yaklaştığını hayal edin!).

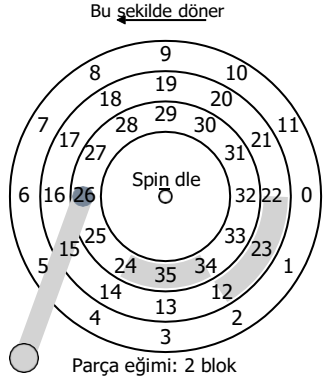
Aramadan sonra, disk kolu kafayı doğru yolun üzerine yerleştirmiştir. Aramanın bir tasviri Şekil 37.3'te (sağda) bulunur.

Gördüğümüz gibi, arama sırasında kol istenen yola hareket ettirildi ve tabla elbette bu durumda yaklaşık 3 sektör döndürüldü. Böylece, sektör 9 disk kafasının altından geçmek üzere ve aktarımı tamamlamak için sadece kısa bir dönme gecikmesine katlanmamız gerekiyor.

Sektör 11 disk başlığının altından geçtiğinde, G/Ç'nin son aşaması gerçekleşecektir.**Aktar**, burada veriler yüzeyden okunur veya yüzeye yazılır. Ve böylece, G/Ç zamanının tam bir resmine sahibiz: önce bir arama, ardından dönüş gecikmesini beklemek ve son olarak aktarım.

Diğer Bazı Detaylar

Üzerinde çok fazla zaman harcamayacak olsak da, sabit disklerin nasıl çalıştığına dair başka ilginç ayrıntılar da var. Birçok sürücü bir tür kullanır **çarpık parça** sıralı okumaların yol sınırlarını geçerken bile düzgün bir şekilde hizmet verilebildiğinden emin olmak için. Basit örnek diskimizde bu, Şekil 37.4'te (sayfa 5) görüldüğü gibi görünebilir.



Şekil 37.4:**Üç İz: 2 Skew İzi**

Sektörler genellikle bu şekilde çarpıktır çünkü bir izden diğerine geçerken diskin kafayı yeniden konumlandırması (hatta komşu izlere) için zamana ihtiyacı vardır. Böyle bir eğrilik olmadan, başlık bir sonraki ize hareket ettirilir, ancak istenen bir sonraki blok zaten başlığın altında dönmüş olur ve bu nedenle sürücünün bir sonraki bloğa erişmek için neredeyse tüm dönme gecikmesini beklemesi gerekir.

Başka bir gerçeklik de, geometrinin bir sonucu olarak, dış izlerin iç izlerden daha fazla sektöre sahip olma eğiliminde olmasıdır; orada daha fazla yer var. Bu izler genellikle şu şekilde adlandırılır:**çok bölgeli** diskin birden çok bölgede düzenlendiği ve bir bölgenin bir yüzey üzerinde birbirini takip eden izler kümesi olduğu disk sürücüleri. Her bölge, iz başına aynı sayıda sektöre sahiptir ve dış bölgeler, iç bölgelere göre daha fazla sektöre sahiptir.

Son olarak, herhangi bir modern disk sürücüsünün önemli bir parçası, **önbellek**, tarihsel nedenlerle bazen denir **arabellek izle**. Bu önbellek, sürücünün diskten okunan veya diske yazılan verileri tutmak için kullanabileceği küçük bir bellek miktarıdır (genellikle yaklaşık 8 veya 16 MB). Örneğin, diskten bir sektör okurken, sürücü o yoldaki tüm sektörleri okumaya karar verebilir ve bunları kendi belleğinde önbelleğe alabilir; bunu yapmak, sürücünün aynı iz üzerindeki sonraki isteklere hızlı bir şekilde yanıt vermesini sağlar.

Yazmalarda sürücünün bir seçeneği vardır: Verileri belleğine koyduğunda yazmanın tamamlandığını mı yoksa yazma gerçekten diske yazıldıktan sonra mı kabul etmelidir? Eski denir **cevap yazmak** önbelleğe alma (veya bazen **acilraporlama**) ve ikincisi **baştan sona yazmak**. Geri yazma önbelleği bazen sürücünün "daha hızlı" görünmesini sağlar, ancak tehlikeli olabilir; dosya sistemi veya uygulamalar, verilerin doğru olması için diske belirli bir sırayla yazılmasını gerektiriyorsa, geri yazma önbelleği sorunlara yol açabilir (ayrıntılar için dosya sistemi günlük kaydı hakkındaki bölümü okuyun).

AYAN: DÖNEMLİ ANALİZ

Kimya dersinde, hemen hemen her sorunu, birimleri birbirlerini yok edecek şekilde ayarlayarak nasıl çözdüğünüzü ve bunun sonucunda bir şekilde yanıtların ortaya çıktığını hatırlıyor musunuz? Bu kimyasal büyü, yüksek falutin adı ile bilinir.**boyutlu analiz** ve bilgisayar sistemleri analizinde de yararlı olduğu ortaya çıktı.

Boyut analizinin nasıl çalıştığını ve neden yararlı olduğunu görmek için bir örnek yapalım. Bu durumda, bir diskin tek bir dönüşünün ne kadar sürdüğünü milisaniye cinsinden bulmanız gerektiğini varsayalım. Ne yazık ki, size yalnızca **RPM** diskin veya **dakika başına dönüş**. Diyelim ki 10K RPM'lik birdiskten bahsediyoruz (yani dakikada 10.000 kez dönüyor). Dönüş başına süreyi milisaniye cinsinden elde edecek şekilde boyutsal analizi nasıl kurarız?

Bunun için sol tarafa istenen birimleri koyarak başlıyoruz; bu durumda, dönüş başına süreyi (milisaniye olarak) elde etmek istiyoruz, yani bu ex- aslında ne yazıyoruz: zaman (ms) sonra her şeyi yazarız biliyoruz, mümkünse birimleri iptal ettiğinizden emin olun. İlk önce elde ederiz 1dakika / 10.000Rotasyonlar (dönüş aşağıda tutuluyor, çünkü orası açık sol), ardından ile dakikaları saniyeye dönüştürün 60saniye / 1dakika ve daha sonra nihayet saniyeleri milisaniye cinsinden dönüştürün 1000Hanım / 1ikinci. Nihai sonuç şu şekildedir (birimler güzel bir şekilde iptal edilmiştir):

$$\frac{\text{zaman (ms)}}{\text{1çürük.}} = \frac{1\text{dakika}}{\text{10.000çürük.}} \cdot \frac{60\text{saniye}}{1\text{dakika}} \cdot \frac{1000\text{ms}}{1\text{saniye}} = \frac{60,000\text{Hanım}}{\text{10.000çürük.}} = \frac{6\text{Hanım}}{\text{Döndürme}}$$

Bu örnekte görebileceğiniz gibi, boyutsal analiz, sezgisel görünen şeyi basit ve tekrarlanabilir bir sürece dönüştürür. Yukarıdaki RPM hesaplamasının ötesinde, düzenli olarak G/Ç analizi ile işe yarar. Örneğin, size genellikle bir diskin aktarım hızı verilir, örneğin 100 MB/ saniye ve ardından şu sorulur: 512 KB'lik bir bloğun aktarımı ne kadar sürer (milisaniye cinsinden)? Boyutsal analiz ile kolaydır:

$$\frac{\text{zaman (ms)}}{\text{1Rica etmek}} = 512 \cdot \frac{\text{KB}}{\text{1Rica etmek}} \cdot \frac{1\text{B}}{1024\cdot\text{KB}} \cdot \frac{1\text{Aktarım}}{100\cdot\text{MB}} \cdot \frac{\text{saniyelik}}{1\text{Aktarım}} \cdot \frac{1000\text{Hanım}}{1\text{saniyelik}} = \frac{5\text{Hanım}}{\text{İstek}}$$

37.4 G/Ç Zamanı: Hesabı Yapmak

Artık diskin soyut bir modeline sahip olduğumuza göre, disk performansını daha iyi anlamak için küçük bir analiz kullanabiliriz. Özellikle, artık G/Ç süresini üç ana bileşenin toplamı olarak gösterebiliriz:

$$T_{I/O} = T_{aramak} + T_{rotasyon} + T_{Aktar} \quad (37.1)$$

I/O oranının ($R_{I/O}$), genellikle daha kolay kullanılır

	Çita 15K.5	Barakuda
Kapasite	300 GB	1 TB
RPM	15.000	7.200
Ortalama Arama	4 ms	9 ms
Maksimum Aktarım	125 MB/sn	105 MB/sn
tabaklar	4	4
önbellek	16MB	16/32 MB
aracılığıyla bağlanır	SCSI	SATA

Şekil 37.5:**Disk Sürücüsü Özellikleri: SCSI ve SATA**

sürücüler arasındaki karşılaştırma (aşağıda yapacağımız gibi), zamandan itibaren kolayca hesaplanır. Aktarımın boyutunu geçen süreye bölmeniz yeterlidir:

$$R_{I/O} = \frac{Boyut_{Aktar}}{T_{I/O}} \quad (37.2)$$

I/O süresi için daha iyi bir fikir edinmek için aşağıdaki hesaplamayı yapalım. İlgilendiğimiz iki iş yükü olduğunu varsayalım.**rastgele**iş yükü, diskteki rasgele konumlara küçük (ör. 4 KB) okumalar yapar. Rastgele iş yükleri, veritabanı yönetim sistemleri de dahil olmak üzere birçok önemli uygulamada yaygındır. olarak bilinen ikincisi,**ardışık**iş yükü, çok sayıda sektörü atlamadan diskten art arda okur. Sıralı erişim kalıpları oldukça yaygındır ve bu nedenle de önemlidir.

Rastgele ve sıralı iş yükleri arasındaki performans farkını anlamak için önce disk sürücüsü hakkında birkaç varsayımda bulunmamız gerekir. Seagate'in birkaç modern diskine bakalım. Cheetah 15K.5 [S09b] olarak bilinen ilki, yüksek performanslı bir SCSI sürücüsüdür. İkincisi, Barracuda [S09a], kapasite için üretilmiş bir sürücüdür. Her ikisiyle ilgili ayrıntılar Şekil 37.5'te bulunur.

Gördüğünüz gibi, sürücüler oldukça farklı özelliklere sahiptir ve birçok yönden disk sürücüsü pazarının iki önemli bileşenini güzel bir şekilde özetlemektedir. Birincisi, sürücülerin olabildiğince hızlı dönecek, düşük arama süreleri sağlayacak ve verileri hızlı bir şekilde aktaracak şekilde tasarlandığı "yüksek performanslı" sürücü pazarıdır. İkincisi, bayt başına maliyetin en önemli unsur olduğu "kapasite" pazarıdır; bu nedenle, sürücüler daha yavaştır ancak mevcut alana mümkün olduğu kadar çok bit yerleştirir.

Bu rakamlardan, sürücülerin yukarıda belirtilen iki iş yükü altında ne kadar iyi çalışacağını hesaplamaya başlayabiliriz. Rastgele iş yüküne bakarak başlayalım. Her 4 KB okumanın diskte rastgele bir yerde gerçekleştiğini varsayarsak, bu tür bir okumanın ne kadar süreceğini hesaplayabiliriz. Çita üzerinde:

$$T_{aramak}=4ms,\ T_{rotasyon}=2ms,\ T_{Aktar}=30mikro\ saniye \quad (37.3)$$

Ortalama arama süresi (4 milisaniye), üretici tarafından bildirilen ortalama süre olarak alınır; tam bir aramaya dikkat edin (bir uçtan

TIP: DİSKLERİ SIRALI OLARAK KULLANIN

Mümkün olduğunda, verileri disklere ve disklerden sıralı bir şekilde aktarın. Sıralı mümkün değilse, en azından veri aktarmayı düşünün büyük parçalar halinde: ne kadar büyükse o kadar iyi. I/O küçük rastgele parçalar halinde yapılırsa, I/O performansı önemli ölçüde düşer. Ayrıca, kullanıcılar zarar görecektir. Ayrıca, dikkatsiz rastgele I/O'lerinizle ne kadar acı çektiğinizi bilerek acı çekeceksiniz.

yüzeyden diğerine) muhtemelen iki veya üç kat daha uzun sürecektir. Ortalama dönme gecikmesi doğrudan RPM'den hesaplanır. 15000 RPM, 250 RPS'ye (saniyede dönüş) eşittir; bu nedenle, her dönüş 4 ms sürer. Ortalama olarak, disk yarım dönüşle karşılaşacaktır ve bu nedenle ortalama süre 2 ms'dir. Son olarak, aktarım süresi, en yüksek aktarım hızı üzerinden yalnızca aktarım boyutudur; burada kaybolacak kadar küçük (30*mikrosaniye*; sadece 1 milisaniye elde etmek için 1000 mikrosaniyeye ihtiyacımız olduğunu unutmayın!).

Böylece, yukarıdaki denklemimizden,$T_{I/O}$Çita için kabaca 6 ms'ye eşittir. I/O oranını hesaplamak için transfer boyutunu ortalama süreye böleriz ve böylece şu sonuca varırız:$R_{I/O}$yaklaşık rastgele iş yükü altındaki Çita için0,66MB/sn. Barracuda için aynı hesaplama,$T_{I/O}$yaklaşık 13,2 ms, iki kattan fazla yavaş ve dolayısıyla yaklaşık0.31MB/sn.

Şimdi sıralı iş yüküne bakalım. Burada çok uzun bir transferden önce tek bir arama ve rotasyon olduğunu varsayabiliriz. Basit olması için aktarım boyutunun 100 MB olduğunu varsayalım. Böylece,$T_{I/O}$Çita ve Barracuda için sırasıyla yaklaşık 800 ms ve 950 ms'dir. Dolayısıyla I/O oranları, sırasıyla 125 MB/sn ve 105 MB/sn'lik en yüksek aktarım hızlarına çok yakındır. Şekil 37.6 bu sayıları özetlemektedir.

	çita	Barakuda
$R_{I/O}$Rastgele	0,66 MB/s	0,31 MB/s
$R_{I/O}$Ardışık	125 MB/sn	105 MB/sn

Şekil 37.6:**Disk Sürücüsü Performansı: SCSI'ye Karşı SATA**

Şekil bize bir dizi önemli şeyi gösteriyor. Birincisi ve en önemlisi, rastgele ve sıralı iş yükleri arasında disk performansında büyük bir fark vardır; Cheetah için yaklaşık 200 kat ve Barracuda için 300 kat fazla fark vardır. Ve böylece bilgi işlem tarihindeki en bariz tasarım ipucuna ulaşıyoruz.

Daha incelikli ikinci bir nokta: Üst düzey "performans" sürücüler ile düşük uç "kapasite" sürücüler arasında büyük bir performans farkı vardır. Bu nedenle (ve diğerleri), insanlar genellikle ikincisini olabildiğince ucuza almaya çalışırken birincisi için en yüksek doları ödemeye isteklidir.

AYAN: "ORTALAMA" ARAYIŞININ HESAPLANMASI

Pek çok kitap ve makalede, ortalama disk arama süresinin kabaca tam arama süresinin üçte biri olduğunu göreceksiniz. Bu nereden geliyor?

Ortalama aramaya dayalı basit bir hesaplamadan ortaya çıktığı ortaya çıktı *mesafe*, zamanı değil. Diski bir dizi iz olarak hayal edin,0 ile N. Herhangi iki iz arasındaki arama mesafesixveyböylece aralarındaki farkın mutlak değeri olarak hesaplanır: |x - y|.

Ortalama arama mesafesini hesaplamak için yapmanız gereken tek şey, önce tüm olası arama mesafelerini toplamaktır:

$$\sum_{x=0}^{N}\sum_{y=0}^{N} |x - y|. \qquad (37.4)$$

Ardından, bunu farklı olası arama sayısına bölün:N^2. Toplamı hesaplamak için sadece integral formunu kullanacağız:

$$\int_{x=0}^{N}\int_{y=0}^{N} |x - y|dydx. \qquad (37,5)$$

İç integrali hesaplamak için mutlak değeri bulalım:

$$\int_{y=0}^{x} (x - y)dsen + \int_{y=x}^{N} (y - x)dy. \qquad (37.6)$$

Bunu çözmek $(xy - \frac{1}{2}y^2) + (\frac{1}{2}y^2 - xy)\Big|_0^N$ basitleştirilebilir $(x^2 - \frac{1}{2}N^2)$.Şimdi dış integrali hesaplamalıyız:

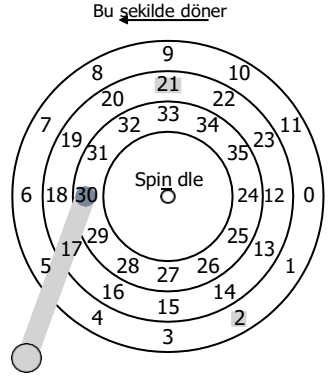
$$\int_{x=0}^{N} (x^2 - \frac{1}{2}N^2)\, dx, \qquad (37.7)$$

hangi sonuçlanır:

$$\left(\frac{1}{3}x^3 - \frac{N}{2}x^2 + \frac{N^2}{2}x\right)\Big|_0^N = \frac{N^3}{3}. \qquad (37.8)$$

Hala toplam arama sayısına bölmemiz gerektiğini unutmayın.

(N^2) ortalama arama mesafesini hesaplamak için: $(N^3)/(N^2) = \frac{1}{3}N$.Böylece bir diskteki ortalama arama mesafesi, tüm olası aramalar üzerinden, tam mesafenin üçte biri kadardır. Ve şimdi ortalama bir aramanın tam aramanın üçte biri olduğunu duyduğunuzda bunun nereden geldiğini anlayacaksınız.



Şekil 37.7:**SSTF: Zamanlama İstekleri 21 ve 2**

37.5 Disk Zamanlama

I/O'nin yüksek maliyeti nedeniyle, işletim sistemi tarihsel olarak diske verilen G/Ç'lerin sırasına karar vermede rol oynamıştır. Daha spesifik olarak, bir dizi I/O talebi verildiğinde,**disk zamanlayıcı**istekleri inceler ve sonra hangisinin planlanacağına karar verir [SCO90, JW91].

Her işin uzunluğunun genellikle bilinmediği iş programlamanın aksine, disk programlama ile bir "iş"in (yani, disk talebinin) ne kadar süreceği konusunda iyi bir tahminde bulunabiliriz. Bir talebin aranmasını ve olası dönme gecikmesini tahmin ederek, disk zamanlayıcı her bir talebin ne kadar süreceğini bilebilir ve böylece (açgözlülükle) önce hizmete girmesi en az zaman alacak olanı seçebilir. Böylece, disk zamanlayıcı aşağıdakileri takip etmeye çalışacaktır: **SJF ilkesi (önce en kısa iş)**operasyonunda.

SSTF: Önce En Kısa Arama Süresi

Bir erken disk zamanlama yaklaşımı şu şekilde bilinir:**en kısa arama süresi ilk** (**SSTF**) (olarak da adlandırılır **en kısa arama ilk** veya **SSF**). SSTF, I/O isteklerinin sırasını yola göre sıralar ve istekleri önce tamamlamak için en yakın yoldaki istekleri seçer. Örneğin, başın mevcut pozisyonunun iç yolun üzerinde olduğunu varsayarsak ve sektör 21 (orta yol) ve 2 (dış yol) için taleplerimiz varsa, talebi önce 21'e gönderir, tamamlanmasını bekleriz, ve ardından talebi 2'ye gönderin (Şekil 37.7).

SSTF bu örnekte iyi çalışıyor, önce orta yolu, sonra dış yolu arıyor. Bununla birlikte, aşağıdaki nedenlerden dolayı SSTF her derde deva değildir. İlk olarak, sürücü geometrisi ana bilgisayar işletim sisteminde mevcut değildir; bunun yerine, bir dizi blok görür. Neyse ki, bu sorun oldukça kolay bir şekilde düzeltildi. SSTF yerine, bir işletim sistemi basitçe uygulayabilir**en yakın blok ilk**(**NBF**), isteği bir sonraki en yakın blok adresiyle planlar.

İkinci sorun daha temel:**açlık**. Yukarıdaki örneğimizde, başın şu anda konumlandığı iç yola sürekli bir istek akışı olup olmadığını hayal edin. Diğer parçalara yapılan istekler, saf bir SSTF yaklaşımı tarafından tamamen göz ardı edilir. Ve böylece sorunun özü:

CRUX: DİSK AÇLIĞI NASIL BAŞA ÇEKİLİR?
SSTF benzeri zamanlamayı nasıl uygulayabilir, ancak açlıktan nasıl kaçınabiliriz?

Asansör (diğer adıyla SCAN veya C-SCAN)

Bu sorunun cevabı bir süre önce geliştirildi (örneğin [CKR72]'ye bakın) ve nispeten basittir. Orijinal olarak adlandırılan algoritma **TARAMA**, izler arasında sırayla disk hizmet istekleri arasında ileri geri hareket eder. Disk boyunca tek bir geçiş (dıştan içe izlere veya içten dışa) diyelim. *süpürmek*. Bu nedenle, diskin bu taramasında zaten hizmet verilmiş bir yolda bir blok için bir istek gelirse, hemen işleme alınmaz, bunun yerine bir sonraki taramaya kadar (diğer yönde) kuyruğa alınır.

SCAN'ın hepsi aynı şeyi yapan bir dizi çeşidi vardır. Örneğin, Coffman ve ark. Tanıttı **F-TARAMA** tarama yaparken hizmet verilecek kuyruğu donduran [CKR72]; bu eylem, tarama sırasında gelen istekleri daha sonra hizmet verilmek üzere bir kuyruğa yerleştirir. Bunu yapmak, geç gelen (ancak daha yakın) isteklerin hizmetini geciktirerek uzaktaki isteklerin aç kalmasını önler.

C-TARAMA başka bir yaygın değişkendir, kısaltması **Dairesel TARAMA**.Algoritma, disk boyunca her iki yönde tarama yapmak yerine, yalnızca dıştan içe doğru tarama yapar ve ardından yeniden başlamak için dış izde sıfırlanır. Saf ileri-geri SCAN orta izleri tercih ettiğinden, bunu yapmak iç ve dış izler için biraz daha adildir, yani dış ize hizmet verdikten sonra, SCAN tekrar dış ize geri dönmeden önce ortadan iki kez geçer.

Şimdi açık olması gereken nedenlerden dolayı, SCAN algoritmasına (ve onun kuzenlerine) bazen **asansör** Algoritma, çünkü yukarı veya aşağıgiden bir asansör gibi davranır ve sadece hangi katın daha yakın olduğuna bağlı olarak katlara hizmet vermez. 10. kattan 1. kata inerken birisi 3'te binip 4'e bassa ve asansör 1'den "yakın" olduğu için 4'e çıksa ne kadar can sıkıcı olurdu bir düşünün! Gördüğünüz gibi asansör algoritması gerçek hayatta kullanıldığında asansörlerde kavga çıkmasını engelliyor. Disklerde, sadece aç kalmayı önler.

Ne yazık ki, SCAN ve kuzenleri en iyi zamanlama teknolojisini temsil etmiyor. Özellikle, SCAN (veya hatta SSTF), SJF ilkesine olabildiğince sıkı sıkıya bağlı değildir. Özellikle, dönüşü göz ardı ederler. Ve böylece, başka bir dönüm noktası:

CRUX: DİSK DÖNDÜRME MALİYETLERİ NASIL HESAPLANIR
Alarak SJF'ye daha yakından yaklaşan bir algoritmayı nasıluygulayabiliriz? *ikisi birden* arama ve döndürme hesabı?

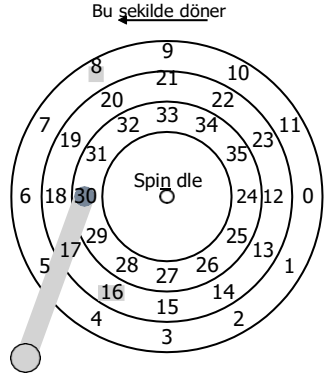
SPTF: Önce En Kısa Konumlandırma Süresi

tartışmadan **önce en kısa konumlandırma süresi** veya **SPTF** zamanlama (bazen **önce en kısa erişim süresi** veya **SATF**) problemimizinçözümü olan sorunu daha detaylı anladığımızdan emin olalım. Şekil 37.8bir örnek sunar.

Örnekte, kafa şu anda iç rayda sektör 30'un üzerinde konumlanmıştır. Bu nedenle programlayıcının bir sonraki talebi için sektör 16'yı (orta yol üzerinde) mi yoksa sektör 8'i (dış yol üzerinde) mi programlaması gerektiğine karar vermesi gerekir. Peki bundan sonra hangisine hizmet etmeli?

Cevap, elbette, "duruma göre değişir". Mühendislikte, değiş tokuşların mühendisin yaşamının bir parçası olduğunu yansıtan "duruma bağlıdır" ifadesinin neredeyse her zaman yanıt olduğu ortaya çıktı; bu tür özdeyişler de bir tutamda iyidir, örneğin, patronunuzun sorusuna bir yanıt bilmiyorsanız, bu taşı denemek isteyebilirsiniz. Ancak, bilmek neredeyse her zaman daha iyidir*Niye*Burada tartıştığımız şeyin hangisi olduğuna bağlı.

Burada bağlı olduğu şey, dönmeye kıyasla aramanın göreli süresidir. Örneğimizde, arama süresi dönme gecikmesinden çok daha yüksekse, SSTF (ve varyantları) gayet iyi. Ancak, aramanın döndürmeden biraz daha hızlı olduğunu hayal edin. O zaman, örneğimizde, aramak daha mantıklı olacaktır. *daha öte*disk kafasının altından geçmeden önce tüm yol boyunca dönmesi gereken hizmet 16'ya giden orta hatta daha kısa aramayı gerçekleştirmektense, dış hat üzerinde hizmet talebi 8'i kullanmak.



Şekil 37.8:**SSTF: Bazen Yeterince İyi Değil**

TIP: HER ZAMAN (LIVNY YASASINA) BAĞLIDIR

Meslektaşımız Miron Livny'nin her zaman söylediği gibi, hemen hemen her soru "duruma göre değişir" şeklinde yanıtlanabilir. Ancak, cevap veriyormuş gibi dikkatli kullanın bu şekilde çok fazla soru sorulursa, insanlar size soru sormayı tamamen bırakacaktır. Örneğin, biri "öğle yemeğine gitmek ister misin?" diye sorar. Cevap veriyorsunuz: "duruma göre değişir*sen*birlikte geliyor?"

Modern sürücülerde, yukarıda gördüğümüz gibi, hem arama hem de döndürme kabaca eşdeğerdir (tabii ki tam isteklere bağlı olarak) ve bu nedenle SPTF kullanışlıdır ve performansı artırır. Bununla birlikte, genellikle iz sınırlarının nerede olduğu veya disk kafasının şu anda nerede olduğu (dönme anlamında) hakkında iyi bir fikri olmayan bir işletim sisteminde uygulanması daha da zordur. Bu nedenle, SPTF genellikle aşağıda açıklanan bir sürücü içinde gerçekleştirilir.

Diğer Planlama Sorunları

Temel disk işlemleri, zamanlama ve ilgili konuların bu kısa açıklamasında ele almadığımız birçok başka konu var. Böyle bir sorun şudur:*nerede*disk zamanlaması modern sistemlerde yapılıyor mu? Daha eski sistemlerde, tüm zamanlamayı işletim sistemi yapıyordu; Bekleyen istekleri inceledikten sonra, işletim sistemi en iyisini seçer ve diske verir. Bu istek tamamlandığında, bir sonraki seçilir ve bu böyle devam ederdi. O zamanlar diskler daha basitti, hayat da öyle.

Modern sistemlerde, diskler çok sayıda bekleyen isteği karşılayabilir ve karmaşık dahili programlayıcılara sahiptir (bunlar SPTF'yi doğru bir şekilde uygulayabilir; disk denetleyicinin içinde, tam kafa konumu dahil ilgili tüm ayrıntılar mevcuttur). Bu nedenle, işletim sistemi programlayıcısı genellikle en iyi birkaç istek olduğunu düşündüğü şeyi seçer (diyelim ki 16) ve hepsini diske gönderir; disk daha sonra söz konusu isteklere mümkün olan en iyi (SPTF) sırada hizmet vermek için baş pozisyonuna ilişkin dahili bilgisini ve ayrıntılı yol düzeni bilgisini kullanır.

Disk zamanlayıcılar tarafından gerçekleştirilen bir diğer önemli ilgili görev,**I/O birleştirme**. Örneğin, Şekil 37.8'deki gibi blok 33'ü, ardından 8'i ve ardından 34'ü okumak için bir dizi istek hayal edin. Bu durumda zamanlayıcı,**birleştirmek** 33 ve 34 bloklarına yönelik talepler tek bir iki bloklu talepte; zamanlayıcının yaptığı herhangi bir yeniden sıralama, birleştirilmiş istekler üzerine gerçekleştirilir. Birleştirme, diske gönderilen isteklerin sayısını azalttığı ve böylece genel giderleri azalttığı için işletim sistemi düzeyinde özellikle önemlidir.

Modern programlayıcıların ele aldığı son bir sorun şudur: Sistem diske bir I/O vermeden önce ne kadar beklemeli? Biri safça, diskin tek bir I/O'ye sahip olduğunda bile hemen sürücüye istek göndermesi gerektiğini düşünebilir; bu yaklaşım denir **iş tasarrufu**, hizmet verme istekleri varsa disk asla boşta kalmayacağından. Bununla birlikte, araştırma **öngörülü disk zamanlaması** bazen daha iyi olduğunu göstermiştir.

biraz [ID01] bekleyin, buna a denir **iş tasarrufu olmayan** yaklaşmak. Bekleyerek, diske yeni ve "daha iyi" bir istek gelebilir ve böylece genel verimlilik artar. Elbette, ne zaman ve ne kadar süre bekleyeceğinize karar vermek zor olabilir; ayrıntılar için araştırma makalesine bakın veya bu tür fikirlerin uygulamaya nasıl geçtiğini görmek için Linux çekirdeği uygulamasına bakın (eğer hırslı biriyseniz).

37.6 Özet

Disklerin nasıl çalıştığının bir özetini sunduk. Özet, aslında ayrıntılı bir işlevsel modeldir; gerçek sürücü tasarımına giren inanılmaz fizik, elektronik ve malzeme bilimini açıklamaz. Bu türden daha fazla ayrıntıyla ilgilenenler için farklı bir ana dal (veya belki de yan dal) öneriyoruz; Bu modelden memnun olanlar için, aferin! Artık bu inanılmaz cihazların üzerine daha ilginç sistemler inşa etmek için modeli kullanmaya devam edebiliriz.

Referanslar

[ADR03] "Bir Arayüzden Daha Fazlası: SCSI'ye Karşı ATA", Dave Anderson, Jim Dykes, Erik Riedel. HIZLI '03, 2003.*Modern disk sürücülerin gerçekte nasıl çalıştığına dair en yeni yakın tarihli referanslardan biri; daha fazlasını öğrenmek isteyen herkesin okuması gereken bir kitap.*

[CKR72] "Disk Arama Sürelerini Azaltmak İçin Tarama İlkelerinin Analizi" EG Coffman, LA Klimko, B. Ryan SIAM Journal of Computing, Eylül 1972, Cilt 1. Sayı 3.*Disk zamanlama alanındaki ilk çalışmalardan bazıları.*

[HK+17] "Katı Hal Sürücülerinin Yazılı Olmayan Sözleşmesi", yazan Jun He, Sudarsun Kannan, Andrea C. Arpaci-Dusseau, Remzi H. Arpaci-Dusseau. EuroSys '17, Belgrad, Sırbistan, Nisan 2017.*Yazılı olmayan sözleşme fikrini alıp SSD'lere kadar genişletiyoruz. SSD'leri iyi kullanmak, sabit sürücüler kadar karmaşık ve bazen daha da karmaşık görünüyor.*

[ID01] "Beklenti Çizelgeleme: Eşzamanlı G/Ç'de Aldatıcı Boştalığın Üstesinden Gelmek İçin Bir Disk Zamanlama Çerçevesi", Sitaram Iyer, Peter Druschel. SOSP '01, Ekim 2001.*Beklemenin disk zamanlamasını nasıl iyileştirebileceğini gösteren harika bir makale: daha iyi istekler yolda olabilir!*

[JW91] "Dönme Konumuna Dayalı Disk Zamanlama Algoritmaları", D. Jacobson, J. Wilkes. Teknik Rapor HPL-CSP-91-7rev1, Hewlett-Packard, Şubat 1991.*Disk zamanlamasına daha modern bir yaklaşım. Yazarlar Seltzer ve ark. [S90].*

[RW92] "Disk Sürücüsü Modellemeye Giriş", C. Ruemmler, J. Wilkes. IEEE Bilgisayar, 27:3, Mart 1994. *Disk kullanımının temellerine müthiş bir giriş. Bazı parçalar güncelliğini yitirdi, ancak temel bilgilerin çoğu duruyor.*

[SCO90] Margo Seltzer, Peter Chen, John Ousterhout tarafından yazılan "Disk Zamanlaması Yeniden Ziyaret Edildi". USENIX 1990.*Disk programlama dünyasında dönmenin de ne kadar önemli olduğundan bahseden bir makale.*

[SG04] "MEMS tabanlı depolama aygıtları ve standart disk arayüzleri: Yuvarlak bir delikte kare bir dübel mi?" Steven W. Schlosser, Gregory R. Ganger FAST '04, s. 87-100, 2004*Bu makalenin MEMS yönü henüz bir etki yaratmamış olsa da, dosya sistemleri ve diskler arasındaki sözleşmenin tartışılması harika ve kalıcı bir katkıdır. Daha sonra "Katı Hal Sürücülerinin Yazılı Olmayan Sözleşmesini" [HK+17] incelemek için bu çalışmanın üzerine inşa edeceğiz.*

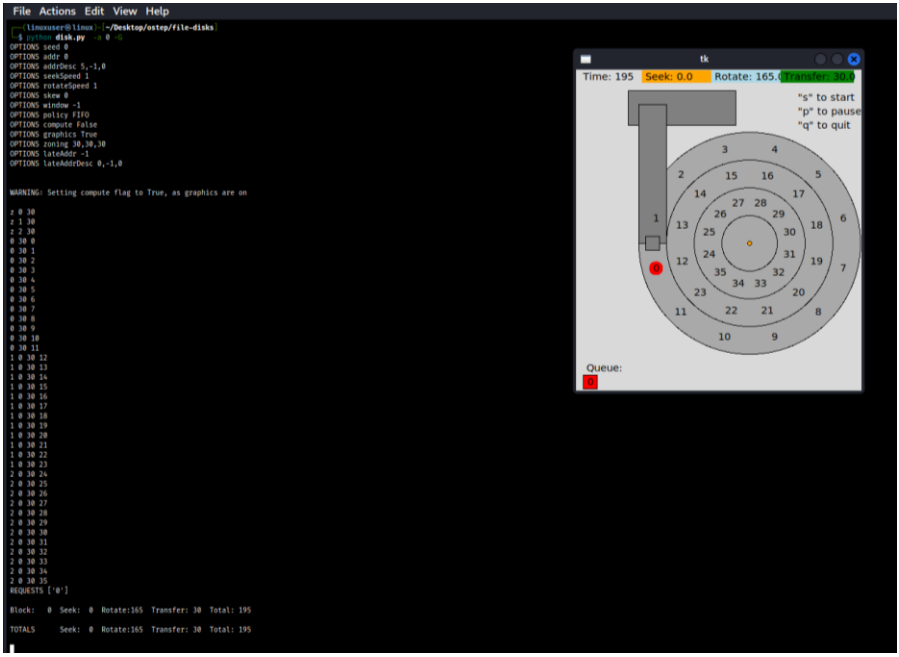
[S09a] Seagate, Inc.'den "Barracuda ES.2 veri sayfası". En azından bu web sitesinde mevcuttu: http://www.seagate.com/docs/pdf/datasheet/disc/ds_barracuda_es.pdf.*Bir veri sayfası; kendi sorumluluğunuzdadır okuyun. Neyin riski? Can sıkıntısı.*

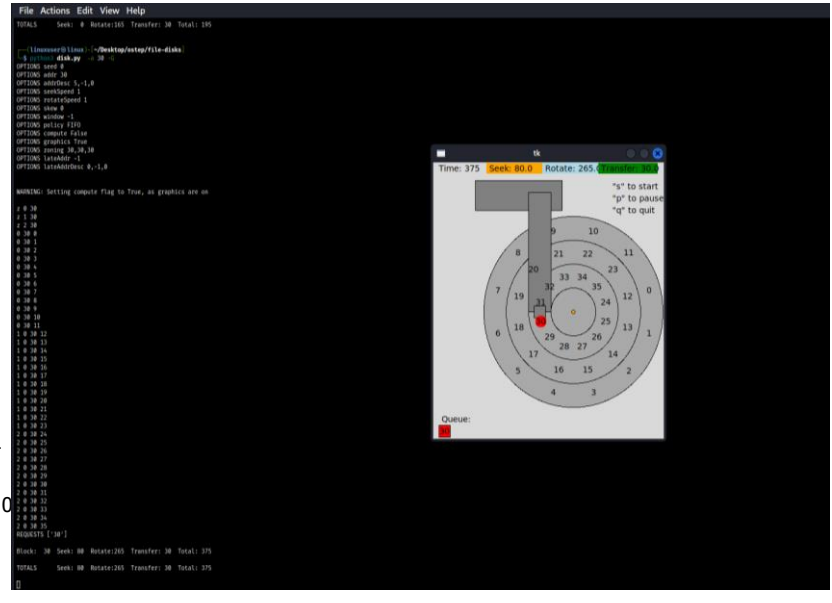
[S09b] Seagate, Inc. imzalı "Cheetah 15K.5". Bu web sitesinde mevcut olduğundan eminiz: http://www.seagate.com/docs/pdf/datasheet/disc/ds-cheetah-15k-5-us.pdf. *Veri sayfalarındaki yukarıdaki yorumlara bakın.*

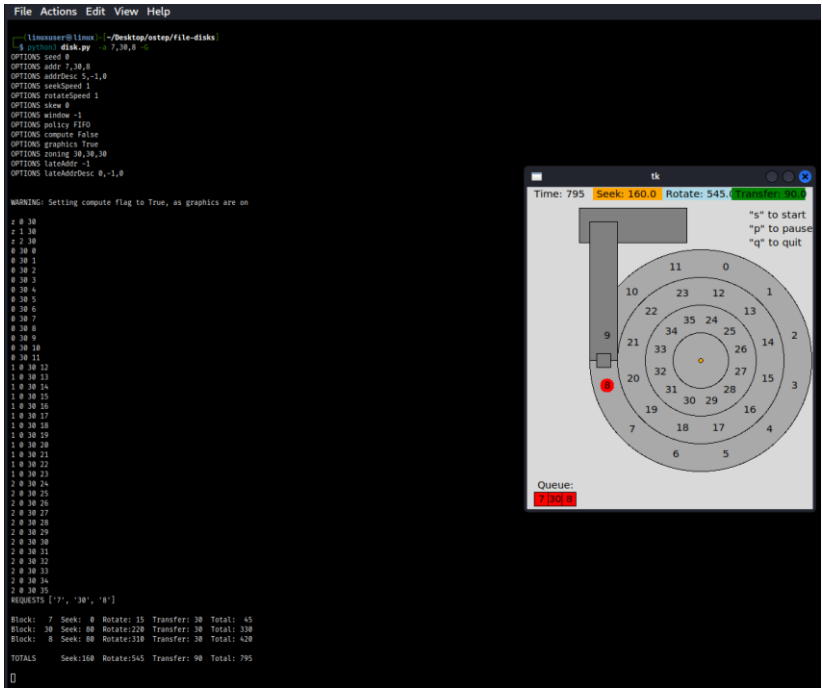
Ödev (Simülasyon)

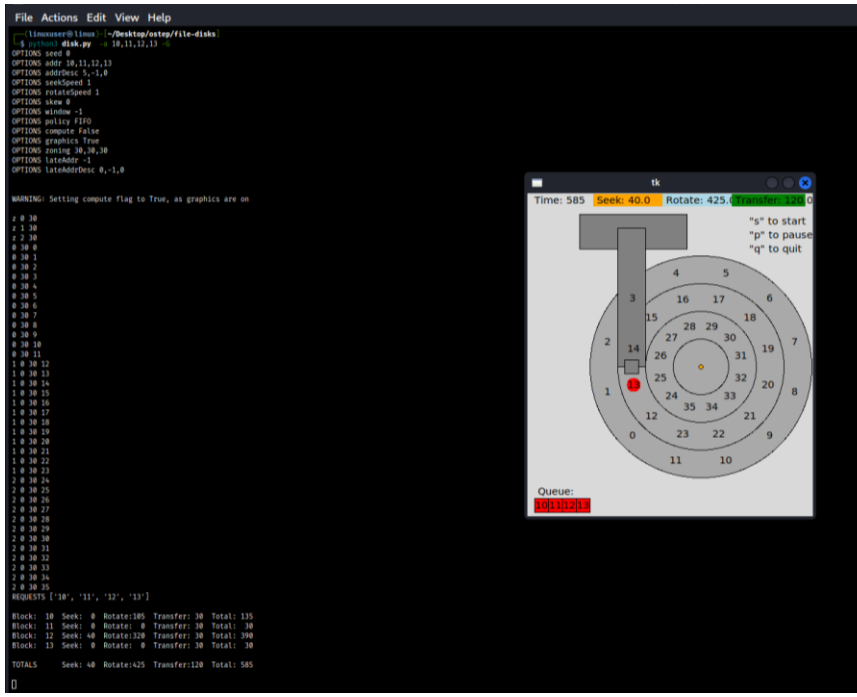
Bu ödev kullanırdisk.pymodern bir sabit diskin nasıl çalıştığını size tanıtmak için. Pek çok farklı seçeneğe sahiptir ve diğer simülasyonların çoğundan farklı olarak, disk hareket halindeyken tam olarak ne olduğunu size gösteren bir grafik animatöre sahiptir. Ayrıntılar için BENİOKU'ya bakın.

1. Aşağıdaki talep grupları için arama, döndürme ve aktarma sürelerini hesaplayın: -0, -a 6, -a 30, -a 7,30,8, ve sonunda -10,11,12,13.






File Actions Edit View Help
linuxuser@linux:~/Desktop/ostep/file-disks
$ python3 disk.py -a 7,30,8 -G
OPTIONS seed 0
OPTIONS addr 7,30,8
OPTIONS addrDesc 5,-1,0
OPTIONS seekSpeed 1
OPTIONS rotateSpeed 1
OPTIONS skew 0
OPTIONS window -1
OPTIONS policy FIFO
OPTIONS compute False
OPTIONS graphics True
OPTIONS zoning 30,30,30
OPTIONS lateAddr -1
OPTIONS lateAddrDesc 0,-1,0
WARNING: Setting compute flag to True, as graphics are on
z 0 30
z 1 30
z 2 30
0 30 0
0 30 1
0 30 2
0 30 3
0 30 4
0 30 5
0 30 6
0 30 7
0 30 8
0 30 9
0 30 10
0 30 11
1 0 30 12
1 0 30 13
1 0 30 14
1 0 30 15
1 0 30 16
1 0 30 17
1 0 30 18
1 0 30 19
1 0 30 20
1 0 30 21
1 0 30 22
1 0 30 23
2 0 30 24
2 0 30 25
2 0 30 26
2 0 30 27
2 0 30 28
2 0 30 29
2 0 30 30
2 0 30 31
2 0 30 32
2 0 30 33
2 0 30 34
2 0 30 35
REQUESTS ['7', '30', '8']
Block: 7 Seek: 0 Rotate: 15 Transfer: 30 Total: 45
Block: 30 Seek: 80 Rotate:220 Transfer: 30 Total: 330
Block: 8 Seek: 80 Rotate:310 Transfer: 30 Total: 420
TOTALS Seek:160 Rotate:545 Transfer: 90 Total: 795
tk
Time: 795 Seek: 160.0 Rotate: 545.0 Transfer: 90.0
"s" to start
"p" to pause
"q" to quit
Queue:
7 30 8


File Actions Edit View Help
linuxuser@linux:~/Desktop/ostep/file-disks
$ python3 disk.py -a 10,11,12,13 -G
OPTIONS seed 0
OPTIONS addr 10,11,12,13
OPTIONS addrDesc 5,-1,0
OPTIONS seekSpeed 1
OPTIONS rotateSpeed 1
OPTIONS skew 0
OPTIONS window -1
OPTIONS policy FIFO
OPTIONS compute False
OPTIONS graphics True
OPTIONS zoning 30,30,30
OPTIONS lateAddr -1
OPTIONS lateAddrDesc 0,-1,0
WARNING: Setting compute flag to True, as graphics are on
z 0 30
z 1 30
z 2 30
0 30 0
0 30 1
0 30 2
0 30 3
0 30 4
0 30 5
0 30 6
0 30 7
0 30 8
0 30 9
0 30 10
0 30 11
1 0 30 12
1 0 30 13
1 0 30 14
1 0 30 15
1 0 30 16
1 0 30 17
1 0 30 18
1 0 30 19
1 0 30 20
1 0 30 21
1 0 30 22
1 0 30 23
2 0 30 24
2 0 30 25
2 0 30 26
2 0 30 27
2 0 30 28
2 0 30 29
2 0 30 30
2 0 30 31
2 0 30 32
2 0 30 33
2 0 30 34
2 0 30 35
REQUESTS ['10', '11', '12', '13']
Block: 10 Seek: 0 Rotate:105 Transfer: 30 Total: 135
Block: 11 Seek: 0 Rotate: 0 Transfer: 30 Total: 30
Block: 12 Seek: 40 Rotate:320 Transfer: 30 Total: 390
Block: 13 Seek: 0 Rotate: 0 Transfer: 30 Total: 30
TOTALS Seek: 40 Rotate:425 Transfer:120 Total: 585
tk
Time: 585 Seek: 40.0 Rotate: 425.0 Transfer: 120.0
"s" to start
"p" to pause
"q" to quit
Queue:
10 11 12 13

Toplam süre:
5,5 * 30 + 30 = 195,0

11,5 * 30 + 30 = 375,0

0,5 * 30 + 30 + 10 * 30 + 30 + 40 * 2 + 310 + 30 = 795.0 30 sektörden 8 sektöre dönerken (yol değiştir), 8 sektörün başlangıç noktası aşıldı, yani daha fazlası 310 zaman birimi alıyor 8 sektörün başına döndürmek için

3,5 * 30 + 2 * 30 + 40 + 320 + 30 * 2 = 585,0

2. Yukarıdaki isteklerin aynısını yapın, ancak arama oranını farklı değerlerle değiştirin: -S 2, -S 4, -S 8, -S 10, -S 40, -S 0.1.Zaman nasıl değişir?

s = 2
Block: 0 Seek: 0 Rotate:165 Transfer: 30 Total: 195

TOTALS Seek: 0 Rotate:165 Transfer: 30 Total: 195

s = 4
Block: 0 Seek: 0 Rotate:165 Transfer: 30 Total: 195

TOTALS Seek: 0 Rotate:165 Transfer: 30 Total: 195

s = 8
Block: 0 Seek: 0 Rotate:165 Transfer: 30 Total: 195

TOTALS Seek: 0 Rotate:165 Transfer: 30 Total: 195

s = 10
Block: 0 Seek: 0 Rotate:165 Transfer: 30 Total: 195

TOTALS Seek: 0 Rotate:165 Transfer: 30 Total: 195

s = 40
Block: 0 Seek: 0 Rotate:165 Transfer: 30 Total: 195

TOTALS Seek: 0 Rotate:165 Transfer: 30 Total: 195

s = 0.1

disk.py: error: option -s: invalid integer value: '0.1'

3. Yukarıdaki isteklerin aynısını yapın, ancak dönüş hızını değiştirin: -R 0.1, -R 0.5, - R 0.01.Zaman nasıl değişir?

R = 0.1

Block: 0 Seek: 0 Rotate:1650 Transfer:300 Total:1950

TOTALS Seek: 0 Rotate:1650 Transfer:300 Total:1950

R = 0.5

Block: 0 Seek: 0 Rotate:330 Transfer: 60 Total: 390

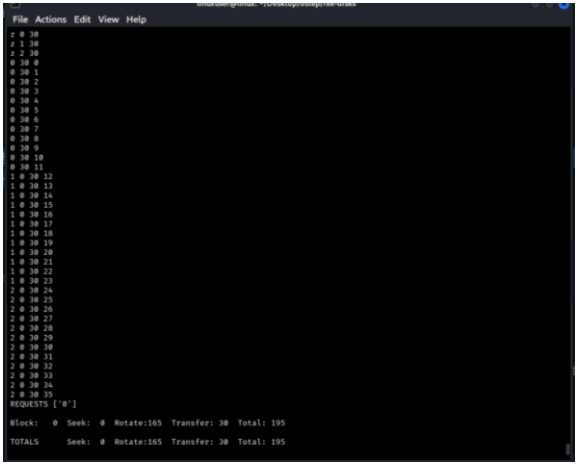
TOTALS Seek: 0 Rotate:330 Transfer: 60 Total: 390

R = 0.01

Block: 0 Seek: 0 Rotate:16500 Transfer:3000 Total:19500

TOTALS Seek: 0 Rotate:16500 Transfer:3000 Total:19500

4. FIFO, örneğin istek akışı ile her zaman en iyisi değildir -bir 7,30,8,talepler hangi sırayla işlenmelidir? Önce en kısa arama süresi (SSTF) planlayıcısını çalıştırın (-p SSTF)bu iş yükü üzerinde; Her talebin yerine getirilmesi ne kadar sürmelidir (arama, döndürme, aktarma)?



-p SSTF ile

Block: 0 Seek: 0 Rotate:165 Transfer: 30 Total: 195

TOTALS Seek: 0 Rotate:165 Transfer: 30 Total: 195

5. Şimdi en kısa erişim zamanı öncelikli (SATF) planlayıcıyı kullanın (-SATF).için bir fark yaratır mı?7,30,8iş yoğunluğu? SATF'nin SSTF'den daha iyi performans gösterdiği bir dizi istek bulun; daha genel olarak, SATF ne zaman SSTF'den daha iyidir?

-p SATF ile

Block: 0 Seek: 0 Rotate:165 Transfer: 30 Total: 195

TOTALS Seek: 0 Rotate:165 Transfer: 30 Total: 195

fark etmez

6. İşte denenecek bir istek akışı: -10,11,12,13.Çalışırken kötü giden nedir? Bu sorunu çözmek için iz eğimi eklemeyi deneyin (-o çarpık).Varsayılan arama oranı göz önüne alındığında, performansı en üst düzeye çıkarmak için sapma ne olmalıdır? Peki ya farklı arama oranları (örneğin, -S 2, -S 4)?Genel olarak, çarpıklığı bulmak için bir formül yazabilir misiniz?

Block: 10 Seek: 0 Rotate:105 Transfer: 30 Total: 135
Block: 11 Seek: 0 Rotate: 0 Transfer: 30 Total: 30
Block: 12 Seek: 40 Rotate:320 Transfer: 30 Total: 390
Block: 13 Seek: 0 Rotate: 0 Transfer: 30 Total: 30

TOTALS Seek: 40 Rotate:425 Transfer:120 Total: 585

Arama süresi çok uzun, bu da parkuru değiştirirken 12'nin biraz üzerinde dönüşe neden oluyor ve bu da bir döngü için yeniden döndürme ihtiyacına neden oluyor

-o skew ile :

Block: 10 Seek: 0 Rotate:105 Transfer: 30 Total: 135
Block: 11 Seek: 0 Rotate: 0 Transfer: 30 Total: 30
Block: 12 Seek: 40 Rotate: 20 Transfer: 30 Total: 90
Block: 13 Seek: 0 Rotate: 0 Transfer: 30 Total: 30

TOTALS Seek: 40 Rotate:125 Transfer:120 Total: 285

buna gore alinmis zaman daha az oldu

7. Bölge başına farklı yoğunluğa sahip bir disk belirleyin, örn., -z 10,20,30,dış, orta ve iç raylardaki bloklar arasındaki açısal farkı belirtir. Bazı rastgele istekleri çalıştırın (örneğin, -bir -1 -A 5,-1,0,rasgele isteklerin - aracılığıyla kullanılması gerektiğini belirtir.bir -1 flag ve 0 ile maksimum arasında değişen beş istek üretilir) ve arama, döndürme ve aktarma sürelerini hesaplar. Farklı rastgele tohumlar kullanın. Dış, orta ve iç hatlardaki bant genişliği (birim zaman başına sektörler olarak) nedir?

REQUESTS [7, 45, 41, 13, 26]

Block: 7 Seek: 0 Rotate:245 Transfer: 10 Total: 255
Block: 45 Seek: 40 Rotate: 55 Transfer: 20 Total: 115
Block: 41 Seek: 0 Rotate:260 Transfer: 20 Total: 280
Block: 13 Seek: 40 Rotate:335 Transfer: 10 Total: 385
Block: 26 Seek: 0 Rotate:120 Transfer: 10 Total: 130

TOTALS Seek: 80 Rotate:1015 Transfer: 70 Total:1165

7, 13, 26 dış çemberdir ve harcanan zaman 255, 385, 120'dir. Dış çemberin bant genişliği: 3/(255+385+120) = 0.0039
45, 41 orta yol ve harcanan süre 115, 280. Orta bant genişliği: 2/(115+280) = 0.00506

8. Bir zamanlama penceresi, diskin aynı anda kaç isteği inceleyebileceğini belirler. Rastgele iş yükleri oluşturun (örn. -1000,-1,0,Farklı tohumlarla) ve planlama penceresi 1'den istek sayısına değiştirildiğinde SATF zamanlayıcısının ne kadar sürdüğünü görün. Performansı en üst düzeye çıkarmak için ne kadar büyük bir

pencere gereklidir? İpucu: -'yi kullanınc flag ve grafikleri açmayın (-G)Bunları hızlı bir şekilde çalıştırmak için. Zamanlama penceresi 1 olarak ayarlandığında, hangi politikayı kullandığınız fark eder mi?

```
-A 1000,-1,0 -w 1 -p SATF -c
```

TOTALS Seek:20960 Rotate:169165 Transfer:30000 Total:220125

```
-A 1000,-1,0 -w 10 -p SATF -c
```

TOTALS Seek:8080 Rotate:26555 Transfer:30000 Total:64635

```
-A 1000,-1,0 -w 1000 -p SATF -c
```

TOTALS Seek:1520 Rotate:3955 Transfer:30000 Total:35475

9. Bir SATF ilkesi varsayarak, belirli bir isteği aç bırakmak için bir dizi istek oluşturun. Bu sıra göz önüne alındığında, bir kullanırsanız nasıl performans gösterir? **sınırlı SATF**(**BSATF**) planlama yaklaşımı? Bu yaklaşımda, zamanlama penceresini belirtirsiniz (örneğin, -w 4);zamanlayıcı yalnızca şu durumlarda bir sonraki istek penceresine geçer:*tüm*geçerli penceredeki isteklere hizmet verildi. Bu açlığı çözüyor mu? SATF ile karşılaştırıldığında performansı nasıl? Genel olarak, bir disk performans ve açlıktan kaçınma arasındaki bu değiş tokuşu nasıl yapmalıdır?

10. Şimdiye kadar incelediğimiz tüm zamanlama politikaları **aç gözlü**; optimalbir program aramak yerine bir sonraki en iyi seçeneği seçerler. Açgözlülüğün optimal olmadığı bir dizi istek bulabilir misiniz?

```
-a 9,20 -c
```

Block: 9 Seek: 0 Rotate: 75 Transfer: 30 Total: 105
Block: 20 Seek: 40 Rotate:260 Transfer: 30 Total: 330

TOTALS Seek: 40 Rotate:335 Transfer: 60 Total: 435

```
-a 9,20 -c -p SATF
```

Block: 20 Seek: 40 Rotate: 5 Transfer: 30 Total: 75
Block: 9 Seek: 40 Rotate:320 Transfer: 30 Total: 390

TOTALS Seek: 80 Rotate:325 Transfer: 60 Total: 465

Açgözlü algoritma global optimumu garanti edemez, sadece yerel optimumu garanti eder.