37

Sabit Disk Sürücüleri (Hard Disk Drives)

Geçen bölümde genel G/Ç aygıtı kavramını tanıtmış ve işletim sisteminin böyle bir aygıtla nasıl etkileşime girebileceğini göstermiştik. Bu bölümde, özellikle bir aygıt hakkında daha fazla ayrıntıya giriyoruz: **sabit disk sürücüsü (hard disk drive)**. Bu sürücüler onlarca yıldır bilgisayar sistemlerinde kalıcı veri depolamanın ana biçimi olmuştur ve dosya sistemi teknolojisinin gelişiminin çoğu (yakında) onların davranışlarına dayanmaktadır. Bu nedenle, onu yöneten dosya sistemi yazılımını oluşturmadan önce bir diskin çalışmasının ayrıntılarını anlamaya değer. Bu ayrıntıların çoğu Ruemmler ve Wilkes [RW92] ve Anderson, Dykes ve Riedel'in [ADR03] mükemmel makalelerinde mevcuttur.

CRUX: DISKTEKI VERILER NASIL SAKLANIR VE ERIŞILIR

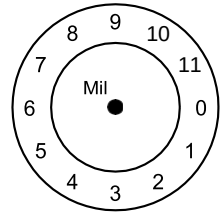
Modern sabit disk sürücüleri verileri nasıl depolar? Arayüz nedir? Veriler gerçekte nasıl düzenlenir ve erişilir? Disk zamanlaması performansı nasıl artırır ?

37.1 Arayüz (Interface)

Modern bir disk sürücünün arayüzünü anlayarak başlayalım. Tüm modern sürücüler için temel arayüz basittir. Sürücü, her biri okunabilen ya da yazılabilen çok sayıda sektörden (512 bytelık bloklar) oluşur. Sektörler n sektörlü bir diskte 0'dan n - 1'e kadar numaralandırılır. Böylece diski bir sektörler dizisi olarak görebiliriz; 0 - n - 1 arası sürücünün **adres alanıdır (address space)**.

Çoklu sektör işlemleri mümkündür; gerçekten de birçok dosya sistemi bir seferde 4KB (veya daha fazla) okuyabilir veya yazabilir. Ancak, diski güncellerken, sürücü üreticilerinin verdiği tek garanti 512 bytelık tek bir yazmanın **atomik (atomic)** olduğudur (yani, ya tamamen tamamlanır ya da hiç tamamlanmaz); bu nedenle, zamansız bir güç kaybımeydana gelirse, daha büyük bir yazmanın yalnızca bir kısmı tamamlanabilir (bazen **yırtık yazma (torn write)** olarak adlandırılır).

Çoğu disk sürücüsü istemcisinin yaptığı, ancak doğrudan arayüzde belirtilmeyen bazı varsayımlar vardır; Schlosser ve Ganger



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

buna disk sürücülerinin "yazılı olmayan sözleşmesi" adını vermiştir [SG04]. Özellikle, genellikle sürücünün adres alanı içinde birbirine yakın iki bloğa[1] erişmenin birbirinden uzak iki bloğa erişmekten daha hızlı olacağı varsayılabilir. Ayrıca genellikle bloklara bitişik bir yığın halinde erişmenin (yani sıralı okuma veya yazma) en hızlı erişim modu olduğu ve genellikle daha rastgele erişim modellerinden çok daha hızlı olduğu varsayılabilir.

37.2 Temel Geometri

Modern bir diskin bazı bileşenlerini anlamaya başlayalım. Bir **plaka (platter)** ile başlıyoruz, verilerin üzerinde manyetik değişiklikler oluşturularak kalıcı olarak saklandığı dairesel sert bir yüzey. Bir disk bir ya da daha fazla plakaya sahip olabilir; her bir plakanın her biri **yüzey (surface)** olarak adlandırılan 2 yüzü vardır. Bu plakalar genellikle sert bir malzemeden (alüminyum gibi) yapılır ve daha sonra sürücünün kapatıldığında bile bitleri kalıcı olarak depolamasını sağlayan ince bir manyetik katmanla kaplanır.

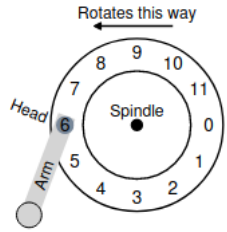
Plakaların tümü, plakaları sabit bir hızda döndüren (sürücü açıkken) bir motora bağlı olan iş **mili (spindle)** etrafında birbirine bağlıdır. Dönme hızı genellikle **dakika başına dönüş (rotations per minute (RPM))** olarak ölçülür vetipik modern değerler 7.200 RPM ila 15.000 RPM aralığındadır. Genellikle 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 her yüzeyde eşmerkezli sektör daireleri halinde kodlanır; bu eşmerkezli dairelerden birine **iz (track)** diyoruz. Tek bir yüzey, sıkıca bir araya getirilmiş binlerce track içerir ve yüzlerce track bir insan saçı genişliğine sığar.

Yüzeyden okumak ve yazmak için, disk üzerindeki manyetik desenleri algılamamızı (yani okumamızı) ya da bunlarda bir değişiklik yapmamızı (yani yazmamızı) sağlayan bir mekanizmaya ihtiyacımız vardır. Bu okuma ve yazma işlemi **disk kafası (disk head)** tarafından gerçekleştirilir; sürücünün her yüzeyinde bu türden bir kafa vardır. Diskkafası, kafayı istenen track üzerinde konumlandırmak için yüzey boyunca hareket eden tek bir **disk koluna (disk arm**) takılıdır.

---

[1] Biz ve diğerleri, okuyucunun bağlama göre tam olarak neyin kastedildiğini bileceğini varsayarak blok ve sektör terimlerini sıklıkla birbirinin yerine kullanıyoruz. Bunun için özürdileriz!



Şekil 37.2: **A Single Track Artı A Head**

37.3 Basit Bir Disk Sürücüsü

Her seferinde bir parça modeli oluşturarak disklerin nasıl çalıştığını anlayalım. Tek bir track'i olan basit bir diskimiz olduğunu varsayalım (Şekil 37.1). Bu izde, her biri 512 byte boyutunda (tipik sektör boyutumuz, hatırlayın) ve bu nedenle 0'dan 11'e kadar sayılarla adreslenen sadece 12 sektör vardır. Burada sahip olduğumuz tek plaka, bir motorun bağlı olduğu mil(spindle) etrafında dönmektedir.

Elbette, track tek başına çok ilginç değildir; 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 takılı disk kafası 6. sektörün üzerine yerleştirilmiştir ve yüzey saat yönünün tersine dönmektedir.

Tek Hat Gecikmesi: Dönme Gecikmesi

(Single-track Latency: The Rotational Delay)

Basit, tek track'li diskimizde bir isteğin nasıl işleneceğini anlamak için, şimdi 0. bloğu okumak için bir istek aldığımızı düşünün. Disk bu isteğe nasıl hizmet vermelidir?

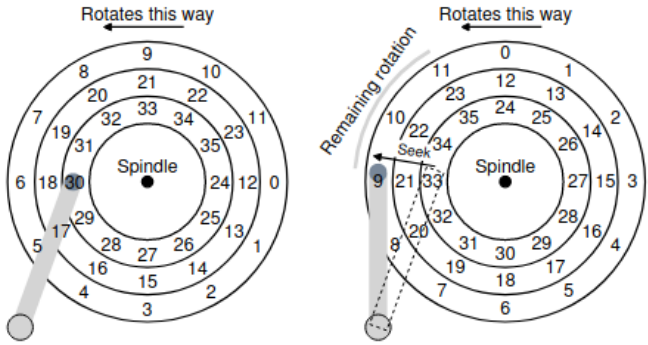
Basit diskimizde, diskin fazla bir şey yapması gerekmez. Özellikle, sadece istenen sektörün disk kafasının altında dönmesini beklemesi gerekir. Bu bekleme modern sürücülerde yeterince sık gerçekleşir ve I/Ohizmet süresinin önemli bir bileşenidir, bu nedenle özel bir adı vardır: **rotasyonel delay(rotational delay)** (bazen **rotasyon gecikmesi (rotation delay),** kulağa garip gelse de). Örnekte, tam dönme gecikmesi R ise, diskin 0'ın okuma/yazma kafasının altına gelmesini beklemek için yaklaşık R/2 kadar bir dönme gecikmesine maruz kalması gerekir (6'dan başlarsak). Bu tek track üzerindeki en kötü durumdaki bir istek 5. sektöre yönelik olacaktır ve böyle bir isteği karşılamak için neredeyse tam bir dönme gecikmesine neden olacaktır.

Çoklu Parçalar: Arama Süresi

(Multiple Tracks: Seek Time)

Şimdiye kadar diskimizde sadece tek bir track vardı, ki bu çok gerçekçi değil; modern disklerde elbette milyonlarca track vardır. Şimdi biraz daha gerçekçi bir disk yüzeyine bakalım, bu yüzeyde üç tane track var (Şekil 37.3, solda).

Şekilde, kafa şu anda en içteki track üzerinde konumlandırılmıştır (24'ten 35'e kadar olan sektörleri içerir); bir sonraki track



Şekil 37.3: **3 Track Artı Head (Sağdaki: Seek ile)**

sonraki sektör setini (12 ila 23) ve en dıştaki track ilk sektörleri (0 ila11) içerir.

Sürücünün belirli bir sektöre nasıl erişebileceğini anlamak için, şimdi uzak bir sektöre yapılan bir talepte neler olacağını izleyelim, örneğin 11. Bu okuma işlemini gerçekleştirmek için sürücünün önce disk kolunu **arama (seek)** olarak bilinen bir işlemle doğru yola (bu durumda en dıştaki yola) hareket ettirmesi gerekir. Arama işlemi, döndürme işlemiyle birlikte en maliyetli disk işlemlerinden biridir.

Arama işleminin birçok aşaması olduğu unutulmamalıdır: önce disk kolu hareket ederken bir hızlanma aşaması; sonra kol tam hızda hareket ederken hızlanma, sonra kol yavaşlarken yavaşlama; son olarak da kafa doğru track üzerinde dikkatlice konumlandırılırken yerleşme. **Yerleşme süresi (settling time)** genellikle oldukça önemlidir, örneğin 0,5 ila 2 ms, çünkü sürücünün doğru yolu bulduğundan emin olması gerekir (bunun yerine sadece yaklaştığını hayal edin!).

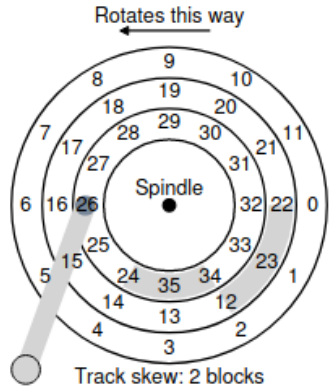
Aramadan sonra, disk kolu kafayı doğru üzerinde konumlandırmıştır. Aramanın bir tasviri Şekil 37.3'te (sağda) bulunmaktadır.

Gördüğümüz gibi, arama sırasında, kol istenen yola hareket ettirildi ve plaka elbette döndü, bu durumda yaklaşık 3 sektör. Böylece, 9. sektör disk kafasının altından geçmek üzeredir ve aktarımı tamamlamak için yalnızca kısa bir dönme gecikmesine katlanmamız gerekir.

Sektör 11 disk kafasının altından geçtiğinde, verilerin yüzeyden okunduğu ya da yüzeye yazıldığı **transfer** olarak bilinen I/O'nun son aşaması gerçekleşecektir. Böylece, I/O süresinin tam bir resmini elde etmiş oluruz: önce arama, sonra dönme gecikmesini bekleme ve son olarak aktarım.

Diğer Bazı Detaylar

Üzerinde çok fazla zaman harcamayacak olsak da, sabit disklerin nasıl çalıştığına dair bazı ilginç ayrıntılar da vardır. Birçok sürücü, track sınırlarını geçerken bile sıralı okumaların düzgün bir şekilde servis edilebildiğinden emin olmak için bir tür track eğriliği **(track skew)** kullanır. Basit örnek diskimizde bu durum Şekil 37.4'te (sayfa 5) görüldüğü gibi olabilir.



Şekil 37.4: **Üç Track: 2'lik Track Eğimi**

Sektörler genellikle bu şekilde çarpıktır çünkü bir track'ten diğerine geçerken diskin kafayı yeniden konumlandırmak için zamana ihtiyacı vardır (komşu track'lere bile). Böyle bir eğrilik olmasaydı, kafa bir sonrakiyola taşınırdı ancak istenen bir sonraki blok zaten kafanın altında dönmüş olurdu ve bu nedenle sürücünün bir sonraki bloğa erişmek için neredeyse tüm dönüş gecikmesini beklemesi gerekirdi.

Bir başka gerçek de dış track'lerin iç track'lerden daha fazla sektöre sahip olma eğiliminde olmasıdır, bu da geometrinin bir sonucudur; orada sadece daha fazla yer vardır. Bu izler genellikle **çok bölgeli (multi-zoned)** disk sürücüleri olarak adlandırılır; burada disk birden fazla bölge halinde düzenlenmiştir ve bir bölge bir yüzey üzerindeki izlerin kon- sekütif kümesidir. Her bölge iz başına aynı sayıda sektöre sahiptir ve dış bölgeler iç bölgelerden daha fazla sektöre sahiptir.

Son olarak, her modern disk sürücüsünün önemli bir parçası, tarihsel nedenlerden dolayı bazen **track tamponu (track buffer)** olarak adlandırılan **önbelleğidir (cache).** Bu cache sadece bazı sürücünün diskten okunan veya diske yazılan verileri tutmak için kullanabileceği küçük miktarda bellek (genellikle yaklaşık 8 veya 16 MB). Örneğin, diskten bir sektör okurken, sürücü o parçadaki tüm sektörleri okumaya ve bunları belleğinde önbelleğe almaya karar verebilir; bunu yapmak sürücünün aynı parçaya yönelik sonraki taleplere hızlı bir şekilde yanıt vermesini sağlar.

Yazma işlemlerinde sürücünün bir seçeneği vardır: Yazma işleminin tamamlandığını veriyi belleğine yerleştirdiğinde mi yoksa yazma işlemi diske gerçekten yazıldıktan sonra mı kabul etmelidir? İlkine **geri yazma (write back)** önbelleği (ya da bazen **anında raporlama(immediate reporting))),** ikincisine ise **yazmaişlemi(write through)** denir. Geri yazma önbelleği bazen sürücünün "daha hızlı"görünmesini sağlar, ancak tehlikeli olabilir; dosya sistemi veya uygulamalar verilerin doğruluğu için diske belirli bir sırada yazılmasını gerektiriyorsa, geri yazma önbelleği sorunlara yol açabilir (ayrıntılar için dosya sistemi günlük kaydı bölümünü okuyun).

ASIDE: BOYUTSAL ANALIZ

Kimya dersinde neredeyse her problemi, birimleri birbirini iptal edecek şekilde ayarlayarak çözdüğünüzü ve sonuç olarak cevapların bir şekilde ortaya çıktığını hatırlıyor musunuz? Bu kimyasal sihir, **boyutsal analiz(dimensional analysis)** olarak bilinir ve bilgisayar sistemleri analizinde de yararlı olduğu ortaya çıktı.

Boyutsal analizin nasıl çalıştığını ve neden yararlı olduğunu görmek için bir örnek yapalım. Bu durumda, bir diskin tek bir dönüşünün milisaniye cinsinden ne kadar sürdüğünü bulmanız gerektiğini varsayalım. Ne yazık ki, size yalnızca diskin **RPM**'si veya **dakika başına dönüş** sayısı veriliyor. Diyelim ki 10K RPM'lik bir diskten bahsediyoruz (yani dakikada 10.000 kez dönüyor). Boyutsal analizi milisaniye cinsinden dönüş başına zaman elde edecek şekilde nasıl ayarlarız?

Bunu yapmak için, istenen birimleri sola koyarak başlıyoruz; bu durumda, dönüş başına zamanı (milisaniye cinsinden) elde etmek istiyoruz, bu yüzden yazdığımız şey tam olarak budur : $\frac{Time(ms)}{1Rotation}$. Sonra bildiğimiz herşeyi yazıyoruz , mümkünse birimleri iptal ettiğinizden emin olun. İlk olarak , $\frac{1\ minute}{10{,}000\ Rotations}$ (solda olduğu gibi altta rotasyonu tutarak), sonra dakikaları sanıyelere çevirip $\frac{1000\ ms}{1\ second}$. Sonucunuz şudur (birimleri güzelce iptal edilmişken) :

$$\frac{Time\ (ms)}{1\ Rot.} = \frac{1\ \cancel{minute}}{10{,}000\ Rot.} \cdot \frac{60\ \cancel{seconds}}{1\ \cancel{minute}} \cdot \frac{1000\ ms}{1\ \cancel{second}} = \frac{60{,}000\ ms}{10{,}000\ Rot.} = \frac{6\ ms}{Rotation}$$

Bu örnekten de görebileceğiniz gibi, boyutsal analiz ne yapar, basit ve tekrarlanabilir bir süreçte sezgisel görünür. Ötesinde yukarıdaki RPM hesaplaması, düzenli olarak G / Ç analizi ile kullanışlı olur. Örneğin, genellikle bir diskin aktarım hızı verilir, örneğin,100 MB/saniye ve ardından şu soruyu sordu: Bir 512 KB blok (milisaniye cinsinden)? Boyutsal analiz ile şunları yapmak kolaydır:

$$\frac{Time\ (ms)}{1\ Request} = \frac{512\ \cancel{KB}}{1\ Request} \cdot \frac{1\ \cancel{MB}}{1024\ \cancel{KB}} \cdot \frac{1\ \cancel{second}}{100\ \cancel{MB}} \cdot \frac{1000\ ms}{1\ \cancel{second}} = \frac{5\ ms}{Request}$$

37.4 I/O Süresi: Matematik Yapıyoruz

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

$$T_{I/O} = T_{seek} + T_{rotation} + T_{transfer} \quad (37.1)$$

I/O oranının ($R_{I/O}$), genellikle aşağıdakiler için daha kolay kullanıldığına dikkat edin

	Cheetah 15K.5	Barracuda
Kapasite	300 GB	1 TB
RPM	15,000	7,200
Ortalama Arama	4 ms	9 ms
Maksimum Transfer	125 MB/s	105 MB/s
Tabaklar	4	4
Önbellek	16 MB	16/32 MB
Üzerinden bağlanır	SCSI	SATA

Şekil 37.5: **Disk Sürücüsünün Özellikleri: SCSI'ye Karşı SATA**

sürücüler arasındaki karşılaştırma (aşağıda yapacağımız gibi), zamandan kolayca hesaplanır. Basitçe aktarımın boyutunu aktarım süresine bölün:

$$R_{I/O} = \frac{Size_{Transfer}}{T_{I/O}} \tag{37.2}$$

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

Rastgele ve sıralı iş yükleri arasındaki performans farkını anlamak için önce disk sürücüsü hakkında birkaç varsayım yapmamız gerekir. Seagate'in birkaç modern diskine bakalım. Birincisi, Cheetah 15K.5 [S09b] olarak bilinen, yüksek performanslı bir SCSI sürücüdür. İkincisi, Barracuda [S09a], kapasite için üretilmiş bir sürücüdür. Her ikisine ilişkin ayrıntılar Şekil 37.5'te yer almaktadır.

Gördüğünüz gibi, sürücüler oldukça farklı özelliklere sahiptir ve birçok yönden disk sürücü pazarının iki önemli bileşenini güzel bir şekilde özetlemektedir. Birincisi, sürücülerin mümkün olduğunca hızlı dönecek, düşük arama süreleri sunacak ve verileri hızlı bir şekilde aktaracak şekilde tasarlandığı "yüksek performanslı" sürücü pazarıdır. İkincisi ise byte başına maliyetin en önemli unsur olduğu "kapasite" pazarıdır; dolayısıyla sürücüler daha yavaştır ancak mevcut alana mümkün olduğunca çok bit sığdırırlar.

Bu sayılardan yola çıkarak, sürücülerin yukarıda özetlenen iki iş yükü altında ne kadar iyi performans göstereceğini hesaplamaya başlayabiliriz. Rastgele iş yüküne bakarak başlayalım. Her 4 KB'lık okumanın disk üzerinde rastgele bir konumda gerçekleştiğini varsayarsak, her bir okumanın ne kadar süreceğini hesaplayabiliriz. Cheetah üzerinde:

$$T_{seek} = 4\ ms,\ T_{rotation} = 2\ ms,\ T_{transfer} = 30\ microsecs \tag{37.3}$$

Ortalama arama süresi (4 milisaniye) üretici tarafından bildiril ortalama süre olarak alınmıştır; tam aramanın (bir uçtan diğer uca

TIP: DISKLERI SIRALI OLARAK KULLANIN

Mümkünse, verileri disklere ve disklerden sıralı bir şekilde aktarın. Sıralı aktarım mümkün değilse, en azından verileri büyük parçalar halinde aktarmayı düşünün: ne kadar büyük olursa o kadar iyi. I/O küçük rastgele parçalar halinde yapılırsa, I/O performansı önemli ölçüde düşecektir. Ayrıca kullanıcılar da zarar görecektir. Ayrıca, dikkatsiz rastgele I/O'lerinizle ne tür acılara yol açtığınızı bildiğiniz için siz de acı çekeceksiniz.

yüzeyden diğerine) muhtemelen iki veya üç kat daha uzun sürecektir. Ortalama dönme gecikmesi doğrudan RPM'den hesaplanır. 15000 RPM 250 RPS'ye (saniye başına dönüş) eşittir; dolayısıyla her dönüş 4 ms sürer. Ortalama olarak, disk yarım dönüşle karşılaşacaktır ve bu nedenle2 ms ortalama süredir. Son olarak, aktarım süresi sadece en yüksek aktarım hızı üzerinden aktarımın boyutudur; burada yok denecek kadar küçüktür (30 *mikrosaniye*; sadece 1 milisaniye elde etmek için 1000 mikrosaniyeye ihtiyacımız olduğunu unutmayın!)

Dolayısıyla, yukarıdaki denklemimizden Cheetah için $T_{I/O}$ kabaca 6 ms'ye eşittir. I/O oranını hesaplamak için transfer boyutunu ortalama süreye böleriz ve böylece yaklaşık 0,66 MB/s'lik rastgele iş yükü altında Cheetah için $R_{I/O}$ değerine ulaşırız. Aynı hesaplama Barracuda için yapıldığında $T_{I/O}$ yaklaşık 13,2 ms, yani iki kattan daha yavaş ve dolayısıyla yaklaşık 0,31 MB/s'lik bir hız elde ediliyor.

Şimdi sıralı iş yüküne bakalım. Burada çok uzun bir aktarımdan önce tek bir arama ve döndürme olduğunu varsayabiliriz. Basit olması için aktarım boyutunun 100 MB olduğunu varsayalım. Böylece, Cheetah ve Barracuda için *TI/O* sırasıyla yaklaşık 800 ms ve 950 ms'dir. Dolayısıyla I/O hızları, sırasıyla 125 MB/s ve 105 MB/s olan en yüksek aktarım hızlarına çok yakındır. Şekil 37.6 bu rakamları özetlemektedir.

	Çita	Barracuda
$R_{I/O}$ Rastgele	0,66 MB/s	0,31 MB/s
$R_{I/O}$ Sıralı	125 MB/s	105 MB/s

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

Şekil bize bir dizi önemli şey gösteriyor. Birincisi ve en önemlisi, rastgele ve sıralı iş yükleri arasında sürücü performansında büyük bir fark vardır; Cheetah için neredeyse 200 kat, Barracuda için ise 300 kattan fazla farkı vardır. Ve böylece bilgisayar tarihindeki en bariz tasarım ipucuna ulaşıyoruz.

İkinci ve daha ince bir nokta: üst seviye "performans" sürücüleri ile düşük seviye "kapasite" sürücüleri arasında büyük bir performans farkı vardır. Bu nedenle (ve diğer nedenlerle), insanlar genellikle birincisi için en yüksek doları ödemeye istekli olurken, ikincisini mümkün olduğunca ucuza almaya çalışırlar.

BIR KENARA: "ORTALAMA" ARAMANIN HESAPLANMASI

Birçok kitap ve makalede, ortalama disk arama süresinin tam arama süresinin yaklaşık üçte biri olarak belirtildiğini göreceksiniz. Bu nereden kaynaklanıyor?

Zamana değil, ortalama arama *mesafesine* dayanan basit bir hesaplamadan kaynaklandığı ortaya çıktı. Diski 0'dan *N'ye* kadar bir dizi track olarak düşünün. Herhangi iki *x* ve *y tracki arasındaki arama mesafesi,* aralarındaki farkın mutlak değeri olarak hesaplanır: |x - y|.

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

$$\sum_{x=0}^{N}\sum_{y=0}^{N}|x-y|. \tag{37.4}$$

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

$$\int_{x=0}^{N}\int_{y=0}^{N}|x-y|\,\mathrm{d}y\,\mathrm{d}x. \tag{37.5}$$

İç integrali hesaplamak için mutlak değeri ayıralım:

$$\int_{y=0}^{x}(x-y)\,\mathrm{d}y+\int_{y=x}^{N}(y-x)\,\mathrm{d}y. \tag{37.6}$$

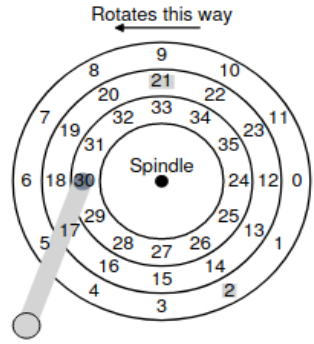
Bu çözmek $(xy-\frac{1}{2}y^2)\big|_0^x+(\frac{1}{2}y^2-xy)\big|_x^N$ 'ye götürür, bu da $(x^2-Nx+\frac{1}{2}N^2)$ 'ye sadeleştirilebilir. Şimdi dış integrali hesaplamalıyız:

$$\int_{x=0}^{N}(x^2-Nx+\frac{1}{2}N^2)\,\mathrm{d}x, \tag{37.7}$$

Sonuç ise:

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

Ortalama arama mesafesini hesaplamak için yine de toplam arama sayısına (N^2) bölmemiz gerektiğini unutmayin : $(\frac{N^3}{3})/(N^2)=\frac{1}{3}N$. Böylece bir diskteki ortalama arama mesafesi, tüm olası aramalar Üzerinden tam mesafe. Ve şimdi ortalama bir aramanın tam aramanın üçte biri olduğunu duyduğunuzda, bunun nereden geldiğini bileceksiniz



Şekil 37.7: **SSTF: Talep 21 ve 2'nin Çizelgelenmesi**

37.5 Disk Zamanlaması

I/O'nun yüksek maliyeti nedeniyle, işletim sistemi geçmişte diske verilen I/O'lerin sırasına karar vermede rol oynamıştır. Daha spesifik olarak, bir dizi I/O isteği verildiğinde, **disk zamanlayıcısı(disk scheduler)** istekleri inceler ve hangisinin daha sonra zamanlanacağına karar verir [SCO90, JW91].

Her bir işin uzunluğunun genellikle bilinmediği iş çizelgelemenin aksine, disk çizelgelemede bir "işin" (yani disk isteğinin) ne kadar süreceği konusunda iyi bir tahminde bulunabiliriz. Disk zamanlayıcı, bir isteğin arama ve olası dönme gecikmesini tahmin ederek, her bir isteğin ne kadar süreceğini bilebilir ve böylece (açgözlülükle) ilk olarak hizmet vermesi en az zaman alacak olanı seçebilir. Böylece, disk zamanlayıcı çalışmasında **SJF (önce en kısa işi) ilkesini** takip etmeye çalışacaktır.

SSTF: Önce En Kısa Arama Süresi

İlk disk zamanlama yaklaşımlarından biri **en kısa-arama-zamanı-ilk (SSTF)** (**en kısa-arama-ilk** veya **SSF** olarak da adlandırılır) olarak bilinir. SSTF, I/O istekleri kuyruğunu izlere göre sıralar ve önce tamamlanması için en yakın izdeki istekleri seçer. Örneğin, kafanın mevcut konumunun iç track üzerinde olduğunu ve 21 (ortadaki track) ve 2 (dıştakı track) sektörleri için isteklerimiz olduğunu varsayarsak, önce 21'e istek gönderir, tamamlanmasını bekler ve ardından 2'ye istek göndeririz (Şekil 37.7).

SSTF bu örnekte iyi çalışmaktadır, önce ortadaki yola sonra da dıştaki yola yönelmektedir. Ancak, SSTF aşağıdaki nedenlerden dolayı her derde deva bir çözüm değildir. İlk olarak, sürücü geometrisi ana işletim sistemi için mevcut değildir; bunun yerine, bir dizi blok görür. Neyse ki bu sorun oldukça kolay bir şekilde çözülebilir. SSTF yerine, bir işletim sistemi basitçe en **yakın önce-yandaki-blok**(**NBF**) uygulayabilir, bu da isteği bir sonraki en yakın blok adresi ile planlar.

İkinci sorun daha temeldir: **açlık(starvatıon)**. Yukarıdaki örneğimizde, başın şu anda konumlandırıldığı inner track sürekli bir istek akışı olduğunu düşünün. Bu durumda diğer yollara yönelik talepler saf bir SSTF yaklaşımı tarafından tamamen göz ardı edilecektir. Ve böylece sorunun crux'ı ortaya çıkar:

CRUX: Disk Açlığı Nasıl İşlenir ?
SSTF benzeri zamanlamayı nasıl uygulanabiliriz ama açlıktan nasıl kaçınabiliriz ?

Asansör (Elevator) (a.k.a. SCAN or C-SCAN)

Bu sorunun cevabı bir süre önce geliştirilmiştir (örneğin [CKR72]' ye bakınız) ve nispeten basittir. Orijinal adı **SCAN olan** algoritma, disk boyunca ileri geri hareket ederek izler arasında sırayla yeniden sorgulama yapar. Disk boyunca tek bir geçişe (dıştan içe ya da içten dışa) *tarama* diyelim. Bu nedenle, diskin bu taramasında zaten hizmet verilmiş olan bir iz üzerindeki bir blok için bir istek gelirse, hemen ele alınmaz, bunun yerine bir sonraki taramaya kadar (diğer yönde) sıraya alınır.

SCAN'ın hepsi yaklaşık olarak aynı şeyi yapan bir dizi çeşidi vardır. Örneğin, Coffman ve arkadaşları, bir tarama yaparken hizmet verilecek kuyruğu donduran **F-SCAN'ı** tanıtmıştır [CKR72]; bu eylem, tarama sırasında gelen istekleri daha sonra hizmet verilmek üzere bir kuyruğayerleştirir. Bunu yapmak, geç gelen (ancak daha yakın olan) isteklerin servisini geciktirerek uzaktaki isteklerin açlıktan ölmesini önler.

C-SCAN, Circular SCAN'ın kısaltması olan bir başka yaygın varyanttır. Disk boyunca her iki yönde de tarama yapmak yerine, algoritma yalnızca dıştan içe doğru tarama yapar ve ardından yeniden başlamak için dış track'inde sıfırlanır. Bunu yapmak iç ve dış trackler için biraz daha adildir, çünkü saf ileri geri SCAN orta trackleri tercih eder, yani dış track'i servis ettikten sonra, SCAN tekrar dış ttrack'e geri dönmeden önce ortadan iki kez geçer.

SCAN algoritması (ve kuzenleri) bazen **asansör(elevator)** algoritması olarak da adlandırılır, çünkü yukarı ya da aşağıya giden bir asansör gibidavranır ve sadece hangi katın daha yakın olduğuna bağlı olarak katlara yönelik taleplere hizmet vermez. Eğer 10. kattan 1. kata iniyor olsaydınız ve birisi 3. kata binip 4'e bassaydı ve asansör 1'den "daha yakın"olduğu için 4'e çıksaydı bunun ne kadar can sıkıcı olacağını bir düşünün! Gördüğünüz gibi, asansör algoritması, gerçek hayatta kullanıldığında, asansörlerde kavga çıkmasını önler. Disklerde ise sadece açlıktan korunur.

Ne yazık ki SCAN ve kuzenleri en iyi çizelgeleme teknolojisini temsil etmemektedir. Özellikle, SCAN (hatta SSTF) aslında SJF prensibine olabildiğince yakın değildir. Özellikle de rotasyonu göz ardı etmektedirler. Ve böylece, başka bir not(crux):

CRUX: DISK ROTASYON MALIYETI NASIL HESAPLANIR?
Hem arama hem de döndürmeyi hesaba katarak SJF'ye daha çok yaklaşan bir algoritmayı nasıl uygulayabiliriz?

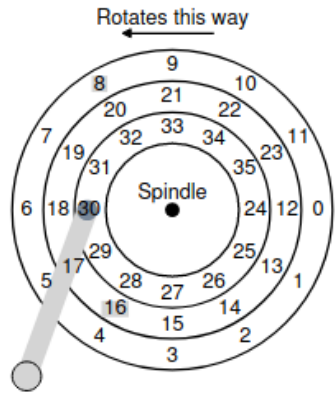
SPTF: Önce En Kısa Konumlandırma Süresi

Problemimizin çözümü olan **önce en kısa konumlandırma zamanı(shortest positioning time first)** veya **SPTF** çizelgelemesini (bazı zamanlar **önce en kısa erişim zamanı** veya **SATF** olarak da adlandırılır) tartışmadan önce, problemi daha iyi anladığımızdan emin olalım. Şekil 37.8 bir örnek sunmaktadır.

Örnekte, kafa şu anda içindeki track'te 30. sektör üzerinde konumlandırılmıştır. Bu nedenle programlayıcı karar vermek zorundadır: bir sonraki talebi için sektör 16'yı mı (ortadak' track üzerinde) yoksa sektör 8'i mi (dıştaki track üzerinde) programlamalıdır. Peki bundan sonra hangisine hizmet vermelidir?

Cevap elbette "**duruma** göre değişir" olacaktır. Mühendislikte, ödünleşimlerin mühendisin hayatının bir parçası olduğunu yansıtan "değişir" neredeyse her zaman cevaptır; **bu** tür özdeyişler, örneğin patronunuzun sorusuna bir cevap bilmediğinizde, bu cevheri denemek isteyebilirsiniz. Bununla birlikte, *neden* değiştiğini bilmek neredeyse her zaman daha iyidir, burada tartıştığımız şey de budur.

Burada önemli olan dönmeye kıyasla arama süresinin göreceli olarak ne kadar olduğudur. Örneğimizde, arama süresi dönme gecikmesinden çok daha yüksekse, SSTF (ve varyantları) gayet iyidir. Ancak, aramanın dönüşten biraz daha hızlı olduğunu düşünün. O zaman, örneğimizde, dış track'teki 8 numaralı servis talebine daha *fazla* arama yapmak, disk kafasının altından geçmeden önce tüm yol boyunca dönmesi gereken 16 numaralı servis için orta track'e daha kısa arama yapmaktan daha mantıklı olacaktır.



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

İPUCU: HER ZAMAN DEĞIŞIR (LIVNY KANUNU)

Meslektaşımız Miron Livny'nin her zaman söylediği gibi, neredeyse her soruya "duruma göre değişir" şeklında cevap verilebilir. Ancak dikkatli olun, çünkü çok fazla soruya bu şekilde cevap verirseniz, insanlar size soru sormayı bırakacaktır. Örneğin, birisi soruyor: "Öğle yemeğine çıkmak ister misin?" Cevap verirsiniz: "Duruma göre değişir, geliyor *musun*? "

Modern sürücülerde, yukarıda gördüğümüz gibi, hem arama hem de döndürme kabaca eşdeğerdir (elbette kesin isteklere bağlı olarak) ve bu nedenle SPTF yararlıdır ve performansı artırır. Bununla birlikte, genellikle track sınırlarının nerede olduğu veya disk kafasının o anda nerede olduğu (rotasyonel anlamda) hakkında iyi bir fikri olmayan bir işletim sisteminde uygulanması daha da zordur. Bu nedenle, SPTF genellikle aşağıda açıklandığı gibi bir sürücünün içinde gerçekleştirilir.

Diğer Programlama Sorunları

Temel disk işlemi, zamanlama ve ilgili konuların bu kısa açıklamasında ele almadığımız birkaç başka konu vardır. Bunlardan biri şudur: modern sistemlerde disk zamanlaması *nerede* yapılır? Eski sistemlerde, tüm zamanlamayı işletim sistemi yapardı; bekleyen istekler kümesine baktıktan sonra, işletim sistemi en iyisini seçer ve diske gönderirdi. Bu istek tamamlandığında, bir sonraki seçilirdi ve bu böyle devam ederdi. O zamanlar diskler daha basitti ve hayat da öyledir.

Modern sistemlerde, diskler birden fazla bekleyen talebi karşılayabilir ve gelişmiş dahili zamanlayıcılara sahiptir (SPTF'yi doğru bir şekilde uygulayabilir; disk denetleyicisinin içinde, tam kafa konumu da dahil olmak üzere ilgili tüm ayrıntılar mevcuttur). Bu nedenle, işletim sistemi zamanlayıcısı genellikle en iyi birkaç isteği (örneğin 16) seçer ve hepsini diske gönderir; disk daha sonra söz konusu istekleri mümkün olan en iyi sırada (SPTF) karşılamak için kafa konumu ve ayrıntılı track düzeni bilgilerine ilişkin dahili bilgisini kullanır.

Disk zamanlayıcıları tarafından gerçekleştirilen bir diğer önemli görev de **G/Ç birleştirme(I/O merging)** işlemidir. Örneğin, Şekil 37.8'deki gibi önce 33, sonra 8, sonra 34 numaralı blokları okumak için bir dizi isteği düşünün. Bu durumda, zamanlayıcı 33 ve 34 numaralı bloklara yönelik talepleri **birleştirerek(merge)** tek bir iki bloklu talep haline getirmelidir; zamanlayıcının yaptığı herhangi bir yeniden sıralama birleştirilmiş talepler üzerinde gerçekleştirilir. Birleştirme özellikle işletim sistemi düzeyinde önemlidir, çünkü diske gönderilen istek sayısını azaltır ve böylece ek yükleri düşürür.

Modern zamanlayıcıların ele aldığı son bir sorun da şudur: sistem diskebir I/O göndermeden önce ne kadar beklemelidir? Safça bir düşünceyle, diskin tek bir I/O bile olsa, talebi derhal sürücüye iletmesi gerektiği düşünülebilir;bu yaklaşıma **iş koruma(work-conserving)** adı verilir, çünkü hizmet edilecek talepler varsa disk asla boşta kalmayacaktır. Ancak, **öngörülü disk çizelgeleme(anticipatory disk scheduling)** üzerine yapılan araştırmalar göstermiştir ki bazen

biraz bekler [ID01], buna **iş korumasız(non-work-conserving)** yaklaşımı denir. Bekleyerek, diske yeni ve "daha iyi" bir istek gelebilir ve böylece genel verimlilik artar. Elbette ne zaman ve ne kadar bekleneceğine karar vermek zor olabilir; ayrıntılar için araştırma makalesine bakın veya bu tür fikirlerin uygulamaya nasıl geçirildiğini görmek için Linux çekirdeği uygulamasına göz atın (eğer hırslı biriyseniz).

37.6 Özet

Disklerin nasıl çalıştığına dair bir özet sunduk. Bu özet aslında ayrıntılı bir işlevsel modeldir; gerçek sürücü tasırımına giren inanılmaz fizik, elektronik ve malzeme bilimini tanımlamaz. Bu türden daha fazla detayla ilgilenenler için farklı bir ana dal (belki yan dal) öneriyoruz; bu modelden memnun olanlar içinse, ne güzel! Artık bu inanılmaz cihazların üzerine daha ilginç sistemler inşa etmek için modeli kullanmaya devam edebiliriz.

Referanslar

[ADR03] "More Than an Interface: SCSI vs. ATA" by Dave Anderson, Jim Dykes, Erik Riedel. FAST '03, 2003. *One of the best recent-ish references on how modern disk drives really work; a must read for anyone interested in knowing more.*

[CKR72] "Analysis of Scanning Policies for Reducing Disk Seek Times" E.G. Coffman, L.A. Klimko, B. Ryan SIAM Journal of Computing, September 1972, Vol 1. No 3. *Some of the early work in the field of disk scheduling.*

[HK+17] "The Unwritten Contract of Solid State Drives" by Jun He, Sudarsun Kannan, Andrea C. Arpaci-Dusseau, Remzi H. Arpaci-Dusseau. EuroSys '17, Belgrade, Serbia, April 2017. *We take the idea of the unwritten contract, and extend it to SSDs. Using SSDs well seems as complicatedas hard drives, and sometimes more so.*

[ID01] "Anticipatory Scheduling: A Disk-scheduling Framework To Overcome Deceptive Idleness In Synchronous I/O" by Sitaram Iyer, Peter Druschel. SOSP '01, October 2001. *A cool paper showing how waiting can improve disk scheduling: better requests may be on their way!*

[JW91] "Disk Scheduling Algorithms Based On Rotational Position" by D. Jacobson, J. Wilkes. Technical Report HPL-CSP-91-7rev1, Hewlett-Packard, February 1991. *A more modern take on disk scheduling. It remains a technical report (and not a published paper) because the authors were scooped by Seltzer et al. [S90].*

[RW92] "An Introduction to Disk Drive Modeling" by C. Ruemmler, J. Wilkes. IEEE Computer, 27:3, March 1994. *A terrific introduction to the basics of disk operation. Some pieces are out of date, but most of the basics remain*.

[SCO90] "Disk Scheduling Revisited" by Margo Seltzer, Peter Chen, John Ousterhout. USENIX 1990. *A paper that talks about how rotation matters too in the world of disk scheduling.*

[SG04] "MEMS-based storage devices and standard disk interfaces: A square peg in a round hole?" Steven W. Schlosser, Gregory R. Ganger FAST '04, pp. 87-100, 2004 While the MEMS *aspect of this paper hasn't yet made an impact, the discussion of the contract between file systems and disks is wonderful and a lasting contribution. We later build on this work to study the "Unwritten Contract of Solid State Drives" [HK+17]*

[S09a] "Barracuda ES.2 data sheet" by Seagate, Inc.. Available at this website, at least, it was: http://www.seagate.com/docs/pdf/datasheet/disc/ds_barracuda_es.pdf. *A data sheet; read at your own risk. Risk of what? Boredom.*

[S09b] "Cheetah 15K.5" by Seagate, Inc.. Available at this website, we're pretty sure it is: http://www.seagate.com/docs/pdf/datasheet/disc/ds-cheetah-15k-5-us.pdf. *See above commentary on data sheets.*

Ödev (Simülasyon)

Bu ödev sizi modern bir sabit diskin nasıl çalıştığına alıştırmak için disk.py kullanır. Birçok farklı seçeneği vardır ve diğer simülasyonların çoğundan farklı olarak, disk çalışırken tam olarak ne olduğunu göstermek için bir grafik animatörüne sahiptir. Ayrıntılar için README'ye bakın.

1. Aşağıdaki görev kümeleri için arama(seek), döndürme(rotation) ve aktarma sürelerini(transfer times) hesaplayın: -a 0, -a 6, -a 30, -a 7,30,8 ve son olarak -a10,11,12,13.

> ./disk.py -a 0 -c iken :

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

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

> ./disk.py -a 6 -c iken :

```
Block:   6  Seek:  0  Rotate:345  Transfer: 30  Total: 375

TOTALS      Seek:  0  Rotate:345  Transfer: 30  Total: 375
```

> ./disk.py -a 30 -c iken :

```
Block:  30  Seek: 80  Rotate:265  Transfer: 30  Total: 375

TOTALS      Seek: 80  Rotate:265  Transfer: 30  Total: 375
```

> ./disk.py -a 7,30,8 -c iken :
//3 block'leri aynı komutundan çalıştırdığı zaman

```
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
```

> ./disk.py -a 10,11,12,13 -c iken :
//4 block'leri aynı komutundan çalıştırdığı zaman

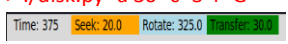
```
Block:  10  Seek:  0  Rotate:105  Transfer: 30  Total: 135
Block:  11  Seek:  0  Rotate:  0  Transfer: 30  Total:  30
Block:  12  Seek: 40  Rotate:320  Transfer: 30  Total: 390
Block:  13  Seek:  0  Rotate:  0  Transfer: 30  Total:  30

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

2. Yukarıdaki isteklerin aynısını yapın, ancak arama hızını(seek) farklı değerlere değiştirin: -S 2, -S 4, -S 8, -S 10, -S 40, -S 0.1. Süreler nasıl değişiyor?

> ./disk.py -a 0 -c -S 2(4,8,10,40,0.1) ve > ./disk.py -a 6 -c -S 2(4,8,10,40,0.1) komutlarında seek (arama hızını) değiştirirken hiç bir değişikliş olmayacaktır , aynı track'inde olduğundan dolayı .

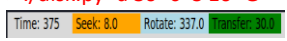
> ./disk.py -a 30 -c -S 2 -G

> ./disk.py -a 30 -c -S 4 -G



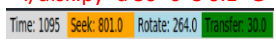
> ./disk.py -a 30 -c -S 8 -G

> ./disk.py -a 30 -c -S 10 -G



> ./disk.py -a 30 -c -S 40 -G

> ./disk.py -a 30 -c -S 0.1 -G



> ./disk.py -a 7,30,8 -c -S 2

```
Block:   7  Seek:  0  Rotate: 15  Transfer: 30  Total:  45
Block:  30  Seek: 40  Rotate:260  Transfer: 30  Total: 330
Block:   8  Seek: 40  Rotate:350  Transfer: 30  Total: 420

TOTALS      Seek: 80  Rotate:625  Transfer: 90  Total: 795
```

> ./disk.py -a 7,30,8 -c -S 4

```
Block:   7  Seek:  0  Rotate: 15  Transfer: 30  Total:  45
Block:  30  Seek: 20  Rotate:280  Transfer: 30  Total: 330
Block:   8  Seek: 20  Rotate: 10  Transfer: 30  Total:  60

TOTALS      Seek: 40  Rotate:305  Transfer: 90  Total: 435
```

> ./disk.py -a 7,30,8 -c -S 8

```
Block:   7  Seek:  0  Rotate: 15  Transfer: 30  Total:  45
Block:  30  Seek: 10  Rotate:290  Transfer: 30  Total: 330
Block:   8  Seek: 10  Rotate: 20  Transfer: 30  Total:  60

TOTALS      Seek: 20  Rotate:325  Transfer: 90  Total: 435
```

> ./disk.py -a 7,30,8 -c -S 10

```
Block:   7  Seek:  0  Rotate: 15  Transfer: 30  Total:  45
Block:  30  Seek:  8  Rotate:292  Transfer: 30  Total: 330
Block:   8  Seek:  8  Rotate: 22  Transfer: 30  Total:  60

TOTALS      Seek: 16  Rotate:329  Transfer: 90  Total: 435
```

> ./disk.py -a 7,30,8 -c -S 40

```
Block:   7  Seek:  0  Rotate: 15  Transfer: 30  Total:  45
Block:  30  Seek:  2  Rotate:298  Transfer: 30  Total: 330
Block:   8  Seek:  2  Rotate: 28  Transfer: 30  Total:  60

TOTALS      Seek:  4  Rotate:341  Transfer: 90  Total: 435
```

> ./disk.py -a 7,30,8 -c -S 0.1

```
Block:   7  Seek:  0  Rotate: 15  Transfer: 30  Total:  45
Block:  30  Seek:801  Rotate:219  Transfer: 30  Total:1050
Block:   8  Seek:801  Rotate:309  Transfer: 30  Total:1140

TOTALS      Seek:1602  Rotate:543  Transfer: 90  Total:2235
```

> ./disk.py -a 10,11,12,13 -c -S 2

```
Block:  10  Seek:  0  Rotate:105  Transfer: 30  Total: 135
Block:  11  Seek:  0  Rotate:  0  Transfer: 30  Total:  30
Block:  12  Seek: 20  Rotate:340  Transfer: 30  Total: 390
Block:  13  Seek:  0  Rotate:  0  Transfer: 30  Total:  30

TOTALS      Seek: 20  Rotate:445  Transfer:120  Total: 585
```

> ./disk.py -a 10,11,12,13 -c -S 4

```
Block:  10  Seek:  0  Rotate:105  Transfer: 30  Total: 135
Block:  11  Seek:  0  Rotate:  0  Transfer: 30  Total:  30
Block:  12  Seek: 10  Rotate:350  Transfer: 30  Total: 390
Block:  13  Seek:  0  Rotate:  0  Transfer: 30  Total:  30

TOTALS      Seek: 10  Rotate:455  Transfer:120  Total: 585
```

> ./disk.py -a 10,11,12,13 -c -S 8

```
Block:  10  Seek:  0  Rotate:105  Transfer: 30  Total: 135
Block:  11  Seek:  0  Rotate:  0  Transfer: 30  Total:  30
Block:  12  Seek:  5  Rotate:355  Transfer: 30  Total: 390
Block:  13  Seek:  0  Rotate:  0  Transfer: 30  Total:  30

TOTALS      Seek:  5  Rotate:460  Transfer:120  Total: 585
```

> ./disk.py -a 10,11,12,13 -c -S 10

```
Block:  10  Seek:  0  Rotate:105  Transfer: 30  Total: 135
Block:  11  Seek:  0  Rotate:  0  Transfer: 30  Total:  30
Block:  12  Seek:  4  Rotate:356  Transfer: 30  Total: 390
Block:  13  Seek:  0  Rotate:  0  Transfer: 30  Total:  30

TOTALS      Seek:  4  Rotate:461  Transfer:120  Total: 585
```

> ./disk.py -a 10,11,12,13 -c -S 40

```
Block:  10  Seek:  0  Rotate:105  Transfer: 30  Total: 135
Block:  11  Seek:  0  Rotate:  0  Transfer: 30  Total:  30
Block:  12  Seek:  1  Rotate:359  Transfer: 30  Total: 390
Block:  13  Seek:  0  Rotate:  0  Transfer: 30  Total:  30

TOTALS      Seek:  1  Rotate:464  Transfer:120  Total: 585
```

> ./disk.py -a 10,11,12,13 -c -S 0.1

```
Block:  10  Seek:  0  Rotate:105  Transfer: 30  Total: 135
Block:  11  Seek:  0  Rotate:  0  Transfer: 30  Total:  30
Block:  12  Seek:401  Rotate:319  Transfer: 30  Total: 750
Block:  13  Seek:  0  Rotate:  0  Transfer: 30  Total:  30

TOTALS      Seek:401  Rotate:424  Transfer:120  Total: 945
```

3. Yukarıdaki isteklerin aynısını yapın, ancak dönüş(rotation) oranını değiştirin: -R 0.1, -R 0,5,-R 0.01. Zamanlar nasıl değişiyor?

> ./disk.py -a 0 -c -R 0.1

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

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

> ./disk.py -a 0 -c -R 0.5

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

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

> ./disk.py -a 0 -c -R 0.0,01

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

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

> ./disk.py -a 6 -c -R 0.1

```
Block:   6  Seek:  0  Rotate:3449  Transfer:301  Total:3750
```

> ./disk.py -a 6 -c -R 0.5

```
Block:   6  Seek:  0  Rotate:690  Transfer: 60  Total: 750
```

> ./disk.py -a 6 -c -R 0.01

```
Block:   6  Seek:  0  Rotate:34500  Transfer:3001  Total:37501
```

> ./disk.py -a 7,30,8 -c -R 0.1

```
Block:   7  Seek:  0  Rotate:150  Transfer:299  Total: 449
Block:  30  Seek: 80  Rotate:2920  Transfer:301  Total:3301
Block:   8  Seek: 80  Rotate:219  Transfer:300  Total: 599

TOTALS      Seek:160  Rotate:3289  Transfer:900  Total:4349
```

> ./disk.py -a 7,30,8 -c -R 0.5

```
Block:   7  Seek:  0  Rotate: 30  Transfer: 60  Total:  90
Block:  30  Seek: 80  Rotate:520  Transfer: 60  Total: 660
Block:   8  Seek: 80  Rotate:700  Transfer: 60  Total: 840

TOTALS      Seek:160  Rotate:1250  Transfer:180  Total:1590
```

> ./disk.py -a 7,30,8 -c -R 0.01

```
Block:   7  Seek:  0  Rotate:1500  Transfer:3000  Total:4500
Block:  30  Seek: 80  Rotate:29920  Transfer:3001  Total:33001
Block:   8  Seek: 80  Rotate:2920  Transfer:2999  Total:5999

TOTALS      Seek:160  Rotate:34340  Transfer:9000  Total:43500
```

> ./disk.py -a 10,11,12,13 -c -R 0.1

```
Block:  10  Seek:  0  Rotate:1050  Transfer:300  Total:1350
Block:  11  Seek:  0  Rotate:  0  Transfer:300  Total: 300
Block:  12  Seek: 40  Rotate:3560  Transfer:300  Total:3900
Block:  13  Seek:  0  Rotate:  0  Transfer:300  Total: 300

TOTALS      Seek: 40  Rotate:4610  Transfer:1200  Total:5850
```

> ./disk.py -a 10,11,12,13 -c -R 0.5

```
Block:  10  Seek:  0  Rotate:210  Transfer: 60  Total: 270
Block:  11  Seek:  0  Rotate:  0  Transfer: 60  Total:  60
Block:  12  Seek: 40  Rotate:680  Transfer: 60  Total: 780
Block:  13  Seek:  0  Rotate:  0  Transfer: 60  Total:  60

TOTALS      Seek: 40  Rotate:890  Transfer:240  Total:1170
```

> ./disk.py -a 10,11,12,13 -c -R 0.01

```
Block:  10  Seek:  0  Rotate:10499  Transfer:3000  Total:13499
Block:  11  Seek:  0  Rotate:  0  Transfer:3001  Total:3001
Block:  12  Seek: 40  Rotate:35961  Transfer:3000  Total:39001
Block:  13  Seek:  0  Rotate:  0  Transfer:3000  Total:3000

TOTALS      Seek: 40  Rotate:46460  Transfer:12001  Total:58501
```

4. FIFO her zaman en iyisi değildir, örneğin -a 7,30,8 istek akışı ile, istekler hangi siralamayla işlenmelidir? Bu iş yükü üzerinde en kısa arama süresi(seek) ilk (SSTF) zamanlayıcısını (-p SSTF) çalıştırın; her bir isteğin

sunulması (arama, döndürme, aktarma) ne kadar sürmelidir?
>./disk.py -a 7,30,8 -c -p SSTF
//İlk önce 7. , sonra 8. ve son olarak 30. Blockları çalıştırır.
Kısaca:
FİFO sıralaması :7,30,8 ve SSTF sıralaması :7,8,30

```
Block:   7  Seek:  0  Rotate: 15  Transfer: 30  Total:  45
Block:   8  Seek:  0  Rotate:  0  Transfer: 30  Total:  30
Block:  30  Seek: 80  Rotate:190  Transfer: 30  Total: 300

TOTALS      Seek: 80  Rotate:205  Transfer: 90  Total: 375
```

5. Şimdi en kısa erişim(erişim) zamanlı ilk (SATF) zamanlayıcıyı kullanın (-p SATF). a 7,30,8 iş yükü için herhangi bir fark yaratıyor mu? SATF'nin SSTF'den daha iyi performans gösterdiği bir dizi istek bulun; daha genel olarak, SATF ne zaman SSTF'den daha iyidir?
>./disk.py -a 7,30,8 -c -p SATF
//SSTF ile aynı değerleri göstermektedir ,bır değişiklik olmayacak

```
Block:   7  Seek:  0  Rotate: 15  Transfer: 30  Total:  45
Block:   8  Seek:  0  Rotate:  0  Transfer: 30  Total:  30
Block:  30  Seek: 80  Rotate:190  Transfer: 30  Total: 300

TOTALS      Seek: 80  Rotate:205  Transfer: 90  Total: 375
```

6. İşte denemek için bir istek akışı: -a 10,11,12,13. Çalıştığında ne kötü gidiyor? Bu sorunu gidermek için parça eğriliği(track skew) eklemeyi deneyin (-o skew). Varsayılan arama hızı göz önüne alındığında, performansı en üst düzeye çıkarmak için çarpıklık ne olmalıdır? Peki ya farklı arama hızları için (örneğin, -S 2, -S 4)? Genel olarak, çarpıklığı hesaplamak için bir formül yazabilir misiniz?
> ./disk.py -a 10,11,12,13 -o 2 -c

```
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
```

> ./disk.py -a 10,11,12,13 -o 4 -c

```
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: 80  Transfer: 30  Total: 150
Block:  13  Seek:  0  Rotate:  0  Transfer: 30  Total:  30

TOTALS      Seek: 40  Rotate:185  Transfer:120  Total: 345
```

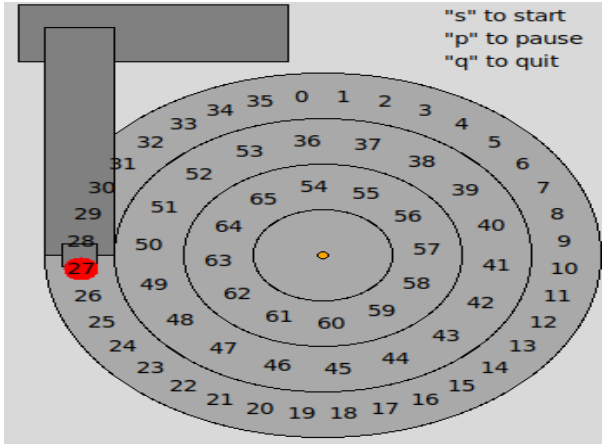
Formül:
skew(eğirlik)=track distance(mesafe=40)/seek speed(arama hızı)*rotation – speed(dönme hızı)/rotational-space-degrees (dönme aramanın dereceleri=360/12) = math.ceil((40 / 1) * 1 / 30) = 2

-S 2: math.ceil((40 / 2) * 1 / 30) = 1

-S 4: math.ceil((40 / 4) * 1 / 30) = 1

-R 2: math.ceil((40 / 1) * 2 / 30) = 3

7. Bölge başına farklı yoğunluğa sahip bir disk belirtin, örneğin -z 10,20,30, dış, orta ve iç tracklerdeki bloklar arasındaki açısal farkı belirtir. Bazı rastgele istekler çalıştırın (örneğin, -a -1 -A 5,-1,0, rastgele isteklerin -a -1 bayrağı aracılığıyla kullanılması gerektiğini ve 0 ile maksimum arasında değişen beş istek oluşturulacağını belirtir) ve arama, döndürme ve aktarım sürelerini hesaplayın. Farklı rastgele tohumlar kullanın. Dış, orta ve iç tracklerdeki bant genişliği (birim zaman başına sektör olarak)



>./disk.py -z 10,20,30 -a -1 -A 5,-1,0 -c
Dıştaki: 3/(135+270+140)=0.0055
Ortadaki: 2/(370+260)=0.0032
İç track'i dokunmadık.

```
Block:  45  Seek: 40  Rotate:310  Transfer: 20  Total: 370
Block:  40  Seek:  0  Rotate:240  Transfer: 20  Total: 260
Block:  22  Seek: 40  Rotate: 85  Transfer: 10  Total: 135
Block:  13  Seek:  0  Rotate:260  Transfer: 10  Total: 270
Block:  27  Seek:  0  Rotate:130  Transfer: 10  Total: 140

TOTALS      Seek: 80  Rotate:1025  Transfer: 70  Total:1175
```

>./disk.py -z 10,20,30 -a -1 -A 5,-1,0 -s 1 -c
Dıştaki: 3/(255+385+130)=0.0039
Ortadaki: 2/(115+280)=0.0051
İç track'i dokunmadık.

```
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
```

>./disk.py -z 10,20,30 -a -1 -A 5,-1,0 -s 2 -c
Dıştaki: 2/(85+10)=0.0211
Ortadaki: 3/(130+360+145)=0.0047

```
Block:  51  Seek: 40  Rotate: 70  Transfer: 20  Total: 130
Block:  51  Seek:  0  Rotate:340  Transfer: 20  Total: 360
Block:   3  Seek: 40  Rotate: 35  Transfer: 10  Total:  85
Block:   4  Seek:  0  Rotate:  0  Transfer: 10  Total:  10
Block:  45  Seek: 40  Rotate: 85  Transfer: 20  Total: 145

TOTALS      Seek:120  Rotate:530  Transfer: 80  Total: 730
```

>./disk.py -z 10,20,30 -a -1 -A 5,-1,0 -s 3 -c
Dıştaki: 5/875=0.0057

```
Block:  12  Seek:  0  Rotate:295  Transfer: 10  Total: 305
Block:  29  Seek:  0  Rotate:160  Transfer: 10  Total: 170
Block:  19  Seek:  0  Rotate:250  Transfer: 10  Total: 260
Block:  32  Seek:  0  Rotate:120  Transfer: 10  Total: 130
Block:  33  Seek:  0  Rotate:  0  Transfer: 10  Total:  10

TOTALS      Seek:  0  Rotate:825  Transfer: 50  Total: 875
```

8. Zamanlama penceresi diskin aynı anda kaç isteği inceleyebileceğini belirler. Rastgele iş yükleri oluşturun (örneğin, -A 1000,-1,0, farklı tohumlarla) ve zamanlama penceresi 1'den istek sayısına kadar değiştirildiğinde SATF zamanlayıcısının ne kadar zaman aldığını görün. Performansı en üst düzeye çıkarmak için ne kadar büyük bir pencere gereklidir? İpucu: -c bayrağını kullanın ve bunları hızlı bir şekilde çalıştırmak için grafikleri (-G) açmayın. Zamanlama penceresi 1 olarak ayarlandığında, hangi politikayı kullandığınız önemli midir?
   Performansı en üst düzeye çıkarmak için diskin boyutu gerekir (-w -1).
   Kullandığımız politika önemli değil :

   >./disk.py -A 1000,-1,0 -p SATF -w 1 -c
   //TOTALS Seek:20960 Rotate:169165 Transfer:30000 Total:220125

   >./disk.py -A 1000,-1,0 -p FIFO -w 1 -c
   //TOTALS Seek:20960 Rotate:169165 Transfer:30000 Total:220125

9. Bir SATF politikası varsayarak belirli bir isteği aç bırakmak için bir dizi istek oluşturun. Bu dizi göz önüne alındığında, **sınırlı SATF (BSATF)** zamanlama yaklaşımını kullanırsanız nasıl bir performans gösterir? Bu yaklaşımda, zamanlama penceresini belirlersiniz (örneğin, -w 4); zamanlayıcı yalnızca mevcut penceredeki *tüm* talepler karşılandığında bir sonraki talep penceresine geçer. Bu açlık sorununu çözüyor mu? SATF ile karşılaştırıldığında nasıl bir performans sergiliyor? Genel olarak, bir disk performans ve açlıktan kaçınma arasındaki bu ödünleşimi nasıl yapmalıdır?
   >./disk.py -a 12,7,8,9,10,11 -p SATF -c
   //7,8,9,10,11,12 sırasıyla Total 555'dır

```
Block:   7  Seek:  0  Rotate: 15  Transfer: 30  Total:  45
Block:   8  Seek:  0  Rotate:  0  Transfer: 30  Total:  30
Block:   9  Seek:  0  Rotate:  0  Transfer: 30  Total:  30
Block:  10  Seek:  0  Rotate:  0  Transfer: 30  Total:  30
Block:  11  Seek:  0  Rotate:  0  Transfer: 30  Total:  30
Block:  12  Seek: 40  Rotate:320  Transfer: 30  Total: 390

TOTALS      Seek: 40  Rotate:335  Transfer:180  Total: 555
```

   >./disk.py -a 12,7,8,9,10,11 -p BSATF -w 4 -c
   //7,8,9,12,10,11 sırasıyla Total 525'dır

```
Block:   7  Seek:  0  Rotate: 15  Transfer: 30  Total:  45
Block:   8  Seek:  0  Rotate:  0  Transfer: 30  Total:  30
Block:   9  Seek:  0  Rotate:  0  Transfer: 30  Total:  30
Block:  12  Seek: 40  Rotate: 20  Transfer: 30  Total:  90
Block:  10  Seek: 40  Rotate:230  Transfer: 30  Total: 300
Block:  11  Seek:  0  Rotate:  0  Transfer: 30  Total:  30

TOTALS      Seek: 80  Rotate:265  Transfer:180  Total: 525
```

10. Şimdiye kadar incelediğimiz tüm çizelgeleme politikaları **açgözlüdür (greedy)**; en uygun çizelgeyi aramak yerine bir sonraki en iyi seçeneği seçerler.Açgözlülüğün optimal olmadığı bir istek kümesi bulabilir misiniz?
    Optimal olmayan durumlar:
    >./disk.py -a 9,20 -c
    //Total = 435

    >/disk.py -a 9,20 -c -p SATF
    //Total= 46