

Kilitlenme Tutarlılığı: FSCK ve Günlük Kaydı

Şimdiye kadar gördüğümüz gibi, dosya sistemi beklenen soyutlamaları uygulamak için bir takım veri yapısını yönetir: Dosyalar, dizinler ve bir dosya sisteminden beklediğimiz temel soyutlamayı desteklemek için gereken diğer tüm meta veriler. Çoğu veri yapısının aksine (örneğin, çalışan bir programın belleğinde bulunanlar), dosya sistemi veri yapıları **kalıcı(persist)** olmalıdır, yani, güç kaybına rağmen verileri tutan cihazlarda (sabit diskler veya flash tabanlı SSD'ler gibi) saklanarak uzun mesafe boyunca hayatı kalması gereklidir.

Bir dosya sisteminin karşılaştığı en büyük zorluklardan biri, **güç kaybı (power loss)** veya **sistem kilitlenmesi (system crash)** olmasına rağmen kalıcı veri yapılarının nasıl güncelleneceğidir. Özellikle, disk yapılarını güncellemenin tam ortasında, birisi güç kablosuna takılırsa ve makine güç kaybederse? Veya işletim sistemi bir hataya karşılaşır ve çökerse ne olur? Güç kayıpları ve kitlenmeler nedeniyle, kalıcı bir veri yapısını güncellemek oldukça zor olabilir ve dosya sistemi uygulamasında **kilitlenme tutarlılığı sorunu (crash-consistency problem)** olarak bilinen yeni ve ilginç bir soruna yol açar.

Bu sorunun anlaşılması oldukça basit. Belirli bir işlemi tamamlamak için iki disk üzerinde yapıyı, A ve B'yi güncelleştirmeniz gerektiğini düşünün. Disk aynı anda yalnızca tek bir isteğe hizmet verdiğinde, bu isteklerden biri önce diske ulaşacaktır (A veya B). Bir yazma işlemi tamamlandıktan sonra sistem çökerse veya güç kaybederse, diskteki yapı **tutarsız (inconsistent)** bir durumda bırakılır. Ve böylece, bütün dosya sistemlerinin çözmesi gereken bir sorunuz var:

CRUX: KİLİTLENMELERE RAĞMEN DISK NASIL GÜNCELLENİR

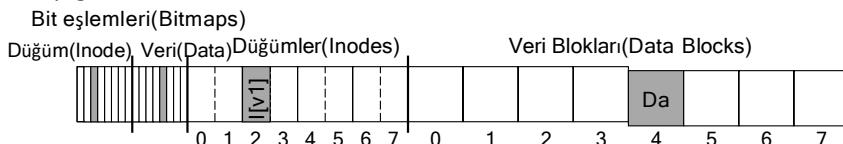
Sistem herhangi iki yazma işlemi arasında çökebilir veya güç kaybedebilir, ve bu nedenle diskteki durum yalnızca kısmen güncelleştirilebilir. Kilitlenmeden sonra, sistem önyüklenir ve dosya sistemi tekrar bağlamak ister (dosyalara ve benzerlerine erişmek için). Kilitlenmelerin zaman içinde rastgele noktalarda meydana gelebileceği göz önüne alındığında, dosya sisteminin diskteki görüntüyü makul bir durumda tutmasını nasıl sağlayabilirim?

Bu bölümde, bu sorunu daha ayrıntılı olarak betimleyeceğiz ve dosya sistemlerinin üstesinden gelmek için kullandığı bazı yöntemlere bakacağız. **fsck** ya da **dosya sistemi kontrolcüsü (file system checker)** olarak bilinen eski dosya sistemleri tarafından benimsenen yaklaşımı inceleyerek başlayacağız. Daha sonra dikkatimizi **günlük kaydı (journaling)** (**önceyen yazma günlüğü (write-ahead logging)**) olarak da bilinir) olarak bilinen, her yazmaya biraz ek yük ekleyen ancak kilitlenmelerden veya güç kayıplarından daha hızlı kurtulan bir teknik olan başka bir yaklaşımına çevireceğiz. Linux ext3'ün [T98,PAA05] (nispeten modern bir günlük kaydı dosya sistemi) uyguladığı birkaç farklı günlük kaydı türü de dahil olmak üzere günlük kaydının temel mekanizmasını tartışacağız.

42.1 Detaylı Bir Örnek

Günlük tutma araştırmamıza başlamak için bir örneğe bakalım. Disk üzeri yapıları bir şekilde güncelleyen bir **iş yükü (workload)** kullanmamız gereklidir. Burada iş yükünün basit olduğunu farz edelim: Tek bir veri bloğunun mevcut bir dosyaye eklenmesi. Ekleme, dosya açarak ve dosya ofsetini dosyanın sonuna taşımak için `lseek()` öğesini çağırarak ve ardından kapatmadan önce dosyaye tek bir 4 KB yazma işlemi göndererek gerçekleştiriliyor.

Ayrıca, daha önce gördüğümüz dosya sistemlerine benzer şekilde, diskte standart basit dosya sistemi yapıları kullandığımızı farz edelim. Bu küçük örnek, bir **düğüm bit eşlemi (inode bitmap)** (her inode için bir tane olmak üzere yalnızca 8 bit), bir **veri bit eşlemi (data bitmap)** (ayrıca 8 bit, veri bloğu başına bir), inode'lar (toplam 8, 0'dan 7'ye kadar numaralandırılmış ve dört bloğa yayılmış), ve veri blokları (toplam 8, 0'dan 7'ye kadar numaralandırılmış). İşte bu dosya sisteminin bir diyagramı:



Eğer resimdeki yapılara bakarsanız, inode bitmap'inde işaretlenen tek bir inode'in (inode numarası 2) ve veri bir eşlemesinde de işaretlenen tek bir tahsis edilmiş veri bloğunu (veri bloğu 4) tahsis edildiğini görebiliriz. Bu inode'in ilk versiyonu olduğu için inode `I[v1]` olarak gösterilir; yakında güncellenecektir (yukarıda açıklanan iş yükü nedeniyle).

Bu basitleştirilmiş inode'un içine de göz atalım. `I[v1]`'in içinde şunları görüyoruz:

```

owner      : remzi
permissions : read-write
size       : 1
pointer    : 4
pointer    : null
pointer    : null
pointer    : null

```

Bu basitleştirilmiş inode'da, dosyanın boyutu 1'dir (bir bloğa tahrif edilmiştir), ve blok 4'e ilk doğrudan işaret işaret eder (dosyanın ilk veri bloğu, Da), ve diğer üç doğrudan işaretinin tümü sıfır (kullanılmadıklarını gösterir) olarak ayarlanır. Elbette, gerçek inode'lerin daha çok alanları vardır; daha fazla bilgi için önceki bölümlere bakın.

Dosyaya eklediğimizde, dosyaya yeni bir veri bloğu ekliyoruz, ve böylece üç disk üstü yapıyı güncellememiz gerekiyor: İnode (yeni bloğa işaret etmeli ve ekleme nedeniyle yeni daha büyük boyutu kaydedilmelidir), yeni veri bloğu Db ve yeni veri bloğunun tahrifini edildiğini belirtmek için veri bitmap'inin (bunu B[v2] olarak çağırın) yeni bir sürümü.

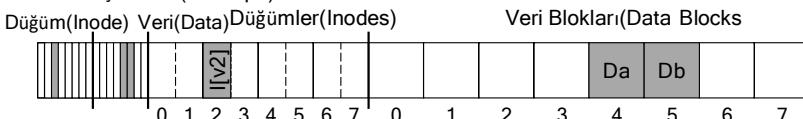
Böylece, sistemin hafızasında, diske yazmamız gereken üç bloğumuz var. Güncellenmiş İnode (inode versiyon 2 veya kısaca I[v2]) şimdi böyle gözüküyor:

```
owner      : remzi
permissions : read-write
size       : 2
pointer    : 4
pointer    : 5
pointer    : null
pointer    : null
```

Güncellenmiş veri bitmap'i (B[v2]) şimdi bunun gibi gözükmür: 00001100. Son olarak, kullanıcıların dosyalara koyduğu herhangi bir şeyle dolu olan bir veri bloğu vardır. Belki de kalıntı müzik?

İstediğimiz dosya sisteminin diskteki son görüntüsünün böyle gözükmesi:

Bit eslemleri(Bitmap)



Bu geçişti başarmak için, dosya sistemi her bir inode (I[v2]), bitmap'i (B[v2]), ve veri bloğu (Db) diske üç parçada yazma işlemi gerçekleştirmelidir. Bu yazmaların genellikle kullanıcı `write()` sistem çağrılarında hemen gerçekleşmediğini unutmayın; bunun yerine, kirli inode, bitmap, ve yeni veriler önce bir süre ana bellekte (**sayfa önbelleğinde (page cache)** ya da **arabellek önbelleğinde (buffer cache)**) durur; Daha sonra, dosya sistemi nihayet bunları diske yazmaya karar verdiğiinde (5 saniye veya 30 saniye sonra), dosya sistemi gerekli yazma isteklerini diske gönderir. Ne yazık ki, kilitlenme meydana gelebilir ve bu nedenle diskteki bu güncelleştirmeleri engelleleyebilir. Özellikle, bu yazmalardan bir veya ikisi gerçekleştikten sonra bir kilitlenme gerçekleşirse, ancak üçü birden gerçekleşmezse, dosya sistemi komik bir durumda bırakılabilir.

Kilitlenme Senaryoları(Crash Scenarios)

Sorunu daha iyi anlamak için, bazı örnek kilitlenme senaryolarına bakalım. Sadece tek bir yazmanın başarılı olduğunu hayal edin; burada listelediğimiz üç olası ihtimal vardır:

- **Sadece veri bloğunun (Db) diske yazılması.** Bu durumda, veri disktedir, fakat bunu ona işaret eden bir inode yoktur ve blok tahsis edilmiş dese bile bitmap'i yoktur. Bu durumda, sanki hiç yazılmamış gibi olur. Bu durum dosya sistemi kilitlenme tutarsızlığı¹ açısından hiç de sorun değildir.
- **Sadece güncellenmiş inode'un (I[v2]) diske yazılması.** Bu durumda, inode, Db'nin yazılmak üzere olduğu disk adresine (5) işaret eder, ancak Db henüz oraya yazılmamıştır. Bu nedenle, bu işaretçiye güvenirsek, **çöp (garbage)** verilerini diskten okuyacağımız (disk adresi 5'in eski içeriği).

Dahası, **dosya sistemi tutarsızlığı(file-system inconsistency)** dediğimiz, yeni bir sorunumuz var. Diskteki bitmap'i bize veri bloğu 5'in henüz tahsis edilmediğini söylüyor, fakat inode edildiğini söyler. Bitmap'i ve inode arasındaki uyuşmazlık bir dosya sisteminin veri yapılarındaki bir tutarsızlıktır; dosya sistemini kullanmak için bu sorunu bir şekilde çözmeliyiz(bununla ilgili daha fazla bilgi aşağıda)

- **Sadece güncellenmiş bitmap'inin (B[v2]) diske yazılması.** Bu durumda, bitmap'i blok 5'in ayrıldığını gösterir, ancak buna işaret eden bir inode yoktur. Böylece dosya sistemi yine tutarsızdır; çözülmenden bırakılırsa, blok 5 dosya sistemi tarafından asla kullanılılmayacağından bu yazma bir **boşluk sızıntısına (space leak)** neden olur.

Diske üç blok yazma girişiminde üç kilitlenme senaryosu daha vardır. Bu gibi durumlarda, iki yazma başarılı olur ve sonucusu başarısız olur:

- **İnode (I[v2]) ve bitmap'i (B[v2]) diske yazılır, ancak veri (Db) yazılmaz.** Bu durumda, dosya sistemi meta verileri tamamen tutarlıdır: İnode 5'i engellemek için bir işaretçiye sahiptir, bitmap'i 5'in kullanımında olduğunu gösterir ve böylece dosya sisteminin meta verilerinin açısından her şey yolunda gözükür. Ama bir sorun var: 5'in içinde yine çöp var.
- **İnode (I[v2]) ve veri bloğu (Db) yazılır, ancak bitmap'i (B[v2]) yazılmaz.** Bu durumda diskteki doğru verileri işaret eden inode'umuz var, ancak yine de inode ile bitmap'inin eski sürümü (B1) arasında bir tutarsızlık var. Bu nedenle, dosya sistemini kullanmadan önce sorunu bir kez daha çözmemiz gerekiyor.
- **Bitmap'i (B[v2]) ve veri bloğu (Db) yazılır, ancak inode (I[V2]) yazılmaz.** Bu durumda, yine inode ve veri bitmap'i arasında bir tutarsızlık var. Ancak, blok yazılmış olmasına rağmen ve bitmap'i kullanımını göstermesine rağmen, hiçbir inode dosyaya işaret etmediğinden, hangi dosyaya ait olduğu hakkında hiçbir fikrimiz yoktur.

¹Ancak, bazı verileri kaybeden kullanıcı için bir sorun olabilir!

Kilitlenme Tutarlılığı Sorunu(The Crash Consistency Problem)

Umarım, bu kilitlenme senaryolarından, çökme nedeniyle disk üzerindeki dosya sistemi görüntümüzde oluşabilecek birçok sorunu görebilirsiniz: Dosya sistemi veri yapılarında tutarsızlık olabilir; Boşluk sizıntılarımız olabilir; çöp verilerini bir kullanıciya döndürebiliriz; ve benzeri. İdeal olarak yapmak istediğimiz şey, dosya sistemini tutarlı bir durumdan(örneğin, dosya eklenmeden önce) diğerine **atomik olarak (atomically)** (Örneğin, inode, bitmap'i ve yeni veri bloğu diske yazıldıktan sonra) olarak taşımaktır. Ne yazık ki, bunu kolayca yapamayız çünkü disk her seferde yalnızca bir yazma işlemi gerçekleştirir ve bu güncellemeler arasında kilitlenmeler ve güç kaybı olabilir. Bu genel soruna **kilitlenme-tutarlılık sorunu (crash-consistency problem)** diyoruz (buna **tutarlı-güncelleme sorunu (consistent-update problem)** da diyebiliriz).

42.2 Çözüm #1: Dosya Sistemi Kontrolcüsü (The File System Checker)

İlk dosya sistemleri, kilitlenme tutarlılığına basit bir yaklaşım benimsedi. Temel olarak, tutarsızlıkların olmasına izin vermeye ve daha sonra bunları düzeltmeye karar verdiler. Bu tembel yaklaşımın klasik bir örneği, bunu yapan bir araçta bulunur: **fsck**². Fsck, bu tür tutarsızlıkları bulmak ve onarmak için kullanılan bir UNIX aracıdır [MK96]; bir disk bölümünü kontrol etmek ve onarmak için benzer araçlar farklı sistemlerde mevcuttur. Böyle bir yaklaşımın tüm sorunları çözümeyeceğini unutmayın; örneğin, dosya sisteminin tutarlı göründüğü ancak inode'un çöp verilerine işaret ettiği yukarıdaki durumu düşünün. Tek gerçek amaç, dosya sistemi meta verilerinin dahili olarak tutarlı olduğundan emin olmaktadır. fsck aracı, McKusick ve Kowalski'nin makalesinde özettendiği gibi birkaç aşamada çalışır [MK96]. Dosya sistemi bağlanmadan ve kullanılabilir hale getirilmeden önce çalıştırılır (fsck, çalışırken başka hiçbir dosya sistemi etkinliğinin devam etmediğini varsayar); tamamlandığında, diskteki dosya sistemi tutarlı olmalı ve böylece kullanıcılar tarafından erişilebilir hale getirilebilir.

İşte Fsck'nin ne yaptığından temel bir özeti:

- **Süper Blok (SuperBlock):** fsck ilk önce süper bloğun makul görünüp görünmediğini kontrol eder, çoğunlukla dosya sistemi boyutunun tahsis edilen blok sayısından daha büyük olduğundan emin olmak gibi mantık kontrolleri yapar. Genellikle bu mantık kontrollerinin amacı, şüpheli (bozuk) bir süper blok bulmaktır; Bu durumda, sistem (veya yönetici) süper bloğun alternatif bir kopyasını kullanmaya karar verebilir.
- **Serbest Bloklar (Free blocks):** Daha sonra fsck, dosya sisteminde hangi blokların şu anda tahsis edildiğini anlamak için inode'ları, dolaylı blokları, çift dolaylı blokları vb. tarar. Bu bilgiyi, tahsis bitmap'lerinin doğru bir versiyonunu üretmek için kullanır; Böylece, bitmap'leri ve inode'lar arasında herhangi bir tutarsızlık varsa, inodelar içindeki bilgilere güvenerek çözülür. Tüm inodelar için aynı tür bir kontrol gerçekleştirilir ve kullanımda

²"eff-ess-see-kay", "eff-ess-check" veya aracı beğenmezseniz "eff-suck" olarak telaffuz edilir. Evet, ciddi profesyonel insanlar bu terimi kullanır.

gibi görünen tüm inodeların inode bitmap'lerinde bu şekilde işaretlendiğinden emin olunur.

- **Düğüm Durumu (Inode state):** Her inode bozulma veya başka sorunlar için kontrol edilir. Örneğin fsck, tahsis edilen her inode'un geçerli bir tür alanına sahip olduğundan emin olur (örneğin normal dosya, dizin, sembolik bağlantı, vb.). inode alanlarıyla ilgili kolayca düzeltilemeyen sorunlar varsa, inode şüpheli olarak kabul edilir ve fsck tarafından temizlenir; inode bitmap'i buna uygun olarak güncellenir.
- **Düğüm Bağlantıları (Inode links):** fsck ayrıca tahsis edilen her inode 'un bağlantı sayısını doğrular. Hatırlayabileceğiniz gibi, bağlantı sayısını belirli dosyaya bir referans (yani bir bağlantı) içeren farklı dizinlerin sayısını gösterir. Bağlantı sayısını doğrulamak için fsck, kök dizinden başlayarak tüm dizin ağacını tarar ve dosya sistemindeki her dosya ve dizin için kendi bağlantı sayılarını oluşturur. Yeni hesaplanan sayımla bir inode içinde bulunan sayı arasında bir uyuşmuzluk varsa, genellikle inode içindeki sayıyı sabitleyerek düzeltici önlem alınmalıdır. Tahsis edilmiş bir inode bulunursa, ancak hiçbir dizin ona atıfta bulunmazsa, *kayıp+bulunan/lost+found* dizinine taşınır.
- **Yenilenenler (Duplicates):** fsck ayrıca yinelenen işaretçileri, yani iki farklı inode'ın aynı bloğa başvurduğu durumları da kontrol eder. Bir inode açıkça kötüye, silinebilir. Alternatif olarak, işaret edilen blok kopyalanabilir, böylece her inode istenildiği gibi kendi kopyasını verir.
- **Kötü Bloklar (Bad Blocks):** Tüm işaretçilerin listesi taranırken hatalı blok işaretçileri için bir kontrol de gerçekleştirilir. Bir işaretçi geçerli aralığının dışındaki bir şeye açıkça işaret ediyorsa, örneğin bölüm boyutundan daha büyük bir bloğa atıfta bulunan bir adrese sahipse "kötü" olarak kabul edilir. Bu durumda, fsck çok zekice bir şey yapamaz; sadece işaretçiyi inode veya dolaylı yoldan kaldırır (temizler).
- **Dizin Kontrolleri (Directory checks):** fsck kullanıcı dosyalarının içeriğini anlamaz; ancak, dizinler dosya sisteminin kendisi tarafından oluşturulan özel olarak biçimlendirilmiş bilgileri tutar. Böylece, fsck her dizinin içeriği üzerinde ek bütünlük kontrolleri yaparak, ".." ve "." girişlerinin ilk girişler olduğundan, bir dizin girişinde başvurulan her inode'un tahsis edildiğinden ve hiçbir dizinin tüm hiyerarşide birden fazla kez bağlanmasıından emin olur.

Gördüğünüz gibi, çalışan bir fsck oluşturmak, dosya sistemi hakkında karmaşık bilgi gerektirir; böyle bir kod parçasının her durumda çalıştığından emin olmak zor olabilir [G+08]. Ancak, fsck'nin (ve benzer yaklaşımların) daha büyük ve belki de daha temel sorunu vardır: onlar çok yavaşlar. Çok büyük bir disk birimiyle, ayrılan tüm blokları bulmak ve tüm dizin ağacını okumak için tüm diski taramak birkaç dakika veya saat sürebilir. Disklerin kapasitesi arttıkça ve RAID'lerin popüleritesi arttıkça fsck'nin performansı engelleyici hale geldi(son gelişmelere rağmen [M+13]). Daha yüksek bir seviyede, fsck'nin temel öncülü biraz mantıksız görünüyor. Diske yalnızca üç bloğun yazıldığını yukarıdaki örneğimizi düşünün; sadece üç blokluk bir güncelleme sırasında ortaya çıkan sorunları çözmek için tüm diski taramak inanılmaz derecede pahalıdır. Bu durum yatak odanızda anahtarlarınızı yere ve ardından bodrumdan başlayarak her odaya doğru ilerleyerek tüm-evi-anahtarlar-icin-ara kurtarma algoritmasını başlatmaya benzer. İşe yarıyor fakat israf. Böylece, diskler (ve

RAID'ler) büyükçe, araştırmacılar ve uygulayıcılar başka çözümler aramaya başladılar.

42.3 Çözüm #2: Günlük Kaydı (veya İleriye Yazma Kaydı) (Journaling (or Write – Ahead Logging))

Muhtemelen tutarlı güncelleme sorunun en popüler çözümü, veritabanı yönetim sistemleri dünyasından bir fikir çalmaktır. **Önceden yazma günlüğü(write-ahead logging)** olarak bilinen bu fikir, tam olarak bu tür bir sorunu çözmek için icat edildi. Dosya sistemlerinde, genellikle tarihsel nedenlerden dolayı önceden yazma **günülgüğü(journaling)** olarak adlandırırız. Bunu yapan ilk dosya sistemi Cedar [H87] idi, ancak Linux ext3 ve ext4, reiserfs, SGI'nin XFS'si ve Windows'un NTFS'si dahil olmak üzere birçok modern dosya sistemi bu fikri kullanıyor.

Temel fikir aşağıdaki gibidir. Diski güncellerken, yapıların üzerine yazmadan önce, ne yapmak üzere olduğunuzu açıklayan küçük bir not yazın (diskte başka bir yerde, iyi bilinen bir yerde). Bu notu yazmak “önceden yaz” kısmıdır ve biz de bunu “günlük (log)” olarak düzenlediğimiz bir yapıya yazarız; bu nedenle, önceden yazma günlüğü.

Notu diske yazarak, güncellediğiniz yapıların güncelleştirilmesi (üzerine yazma) sırasında bir kilitlenme meydada gelirse, geri dönüp yaptığınız nota bakıp tekrar deneyebileceğinizi garanti etmiş oluyorsunuz; böylece, tüm disk taramak yerine, bir kilitlenmeden sonra tam olarak neyi düzeltceğinizi (ve nasıl düzeltceğinizi) bileyeciksiz. Tasarım gereği, günlük kaydı bu nedenle kurtarma sırasında gereken iş miktarını büyük ölçüde azaltmak için güncellemeler sırasında biraz iş ekler.

Şimdi, popüler bir günlük kaydı dosya sistemi olan **Linux ext3**'ün günlük kaydını dosya sisteme nasıl dahil ettiğini açıklayacağız. Disk üstü yapıların çoğu **Linux ext2** ile aynıdır, Örneğin, disk blok gruplarına ayrılmıştır ve her blok grubu bir inode bir eşleniği, veri bir eşleniği, inode'lar ve veri blokları içerir. Yeni anahtar yapısı, bölüm içinde veya başka bir cihazda az miktarda yer kaplayan günlüğün kendisidir. Böylece, bir ext2 dosya sistemi (günlük kaydı olmadan) şöyle görünür:

Süper	Grup 0	Grup 1	...	Grup N	
-------	--------	--------	-----	--------	--

Günlüğün aynı dosya sistemi görüntüsüne yerleştirildiğini varsayırsak (ancak bazen ayrı bir aygıtta veya dosya sistemi içinde bir dosya olarak yerleştirilir), günlük olan bir ext3 dosya sistemi şöyle görünür:

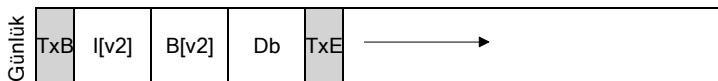
Süper	Günlük	Grup 0	Grup 1	...	Grup N	
-------	--------	--------	--------	-----	--------	--

Gerçek fark, günlüğün varlığı ve elbette nasıl kullanıldığıdır.

Veri Günlüğü (Data Journaling)

Veri günlüğü'nün (data journaling) nasıl çalıştığını anlamak için basit bir örneğe bakalım. Veri günlüğü, bu tartışmanın çoğunun dayandığı Linux ext3 dosya sistemi ile bir modifikasyon(mod) olarak kullanılabilir.

Diyelim ki inode (I[v2]), bitmap'i (B[v2]) ve veri bloğunu (Db) tekrar diske yazmak istediğimiz kanonik bir güncellememizi tekrar alındı. Onları son disk konumlarına yazmadan önce, şimdi onları log'a (diğer adıyla günlük) yazacağız. Bu günlüğte şöyle görünecektir:



Burada beş blok yazdığını görebilirsiniz. İşlem başlangıcı (transaction begin (TxB)), dosya sisteminde bekleyen güncelleme (Örneğin, I[v2], B[v2] ve Db bloklarının son adresleri) hakkında bilgiler ve bir tür **İşlem tanımlayıcı (IT) (transaction identifier (TID))** da dahil olmak üzere bize bu güncelleştirme hakkında bilgi verir. Orta üç blok sadece blokların kendilerinin tam içeriğini içerir; bu, güncelleştirmenin tam fizikal içeriğini günlüğe koymuşuz için **fiziksnel günlük kaydı (physical logging)** olarak bilinir (alternatif bir fikir, **mantıksal günlük kaydı(logical logging)**), güncelleştirmenin daha kompakt bir mantıksal temsilini günlüğte gösterir, Örneğin, "bu güncelleme, X dosyasına Db'yi eklemek istiyor", bu biraz daha karmaşıkır, ancak günlüğte yer tasarrufu sağlayabilir ve belki de performansı artırabilir). Son blok (final block (TxE)), bu işlemin sonunun bir göstergesidir ve ayrıca IT'yi de içerecektir.

Bu işlem güvenli bir şekilde diskte olduğunda, dosya sistemindeki eski yapıların üzerine yazmaya hazırız; bu işleme **kontrol noktası (checkpointing)** denir. Bu nedenle, dosya sistemini **kontrol (checkpoint)** etmek için (yani, günlüğte bekleyen güncelleme ile güncel hale getirin), yukarıda görüldüğü gibi I[v2], B[v2], ve Db yazmalarını disk konumuna veriyoruz; bu yazmalar başarıyla tamamlanırsa, dosya sistemini başarıyla kontrol etmiş oluruz ve temel olarak işimiz biter. Böylece, ilk işlem sıramız:

- Günlük Yazma (Journal write):** İşlem başlangıç bloğu, bekleyen tüm veriler ve meta veri güncellemleri ve bir işlem bitiş bloğu dahil olmak üzere işlemi günlüğe yazma; Bu yazmaların tamamlanmasını bekleyin.
- Kontrol Noktası (Checkpoint):** Bekleyen meta verileri ve veri güncellemlerini dosya sistemindeki son konumlarına yazma.

Örneğimizde, günlüğe önce TxB, I[v2], B[v2], Db ve TxE yazacağız. Bu yazma işlemleri tamamlandığında, I[v2], B[v2], ve Db'yi güncelleştirmeyi tamamlarız.

Günlüğe yazma sırasında bir kilitlenme meydana gelirse işler biraz daha zorlaşır. Burada, işlemdeki blok kümesini (örneğin, TxB, I[v2], B[v2], Db, TxE) diske yazmaya çalışıyoruz. Bunu yapmanın basit bir yolu, her birini bir kerede yayılacak, her birinin tamamlanmasını beklemek ve ardından bir sonrakini yayılacak olacaktır. Ancak, bu yavaştır.

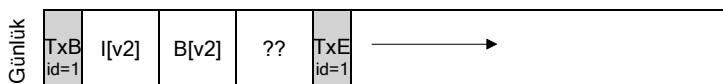
BİR YANA: DISKE YAZMAYI ZORLAMA

İki disk yazma arasında sıralamayı zorlamak için, modern dosya sistemlerinin birkaç ekstra önlem alması gereklidir. Eski zamanlarda, iki yazma arasında sıralamayı zorlamak, A ve B, kolaydı: sadece A'nın diske yazılmasını sağlayın, yazma tamamlandığında diskin işletim sistemini kesmesini bekleyin ve ardından B'nin yazmasını sağlayın.

Disklerdeki yazma önbelleklerinin artan kullanımı nedeniyle işler biraz daha karmaşıklıklaşı. Yazma arabelleğe alma etkinken (bazen **anında raporlama (immediate reporting)** olarak adlandırılır), bir disk, diskin bellek önbelleğine yerleştirildiğinde ve henüz diske ulaşmadığında, işletim sistemine yazma işleminin tamamlandığını bildirir. İşletim sistemi daha sonra bir sonraki yazma işlemini yayınlarsa, önceki yazmalarдан sonra diske ulaşacağı garanti edilmez; böylece yazilar arasındaki sıralama korunmaz. Çözümlerden biri, yazma arabelleğe almayı devre dışı bırakmaktadır. Fakat, daha modern sistemler ekstra önlemler alır ve açık **yazma engelleri (write barriers)** çıkarır; Böyle bir engel, tamamlandığında, engelden önce verilen tüm yazmaların, engelden sonra verilen herhangi bir yazmadan önce diske ulaşmasını garanti eder.

Tüm bu makineler, diskin doğru çalışması için büyük bir güven gerektirir. Ne yazık ki, son araştırmalar bazı disk üreticilerinin, "daha yüksek performanslı" diskler sunmak amacıyla, yazma engeli isteklerini açıkça görmezden geldiğini ve böylece disklerin daha hızlı çalışmasını sağladığını ancak yanlış çalışma riski altında olduğunu gösteriyor [C+13, R+11]. Kahan'ın dediği gibi, hızlı yanlış olsa bile, neredeyse her zaman yavaş olanı yener.

İdeal olarak, beş blok yazmanın tümünü bir kerede paylaşmak isteriz, çünkü bu, beş yazmayı tek bir sıralı yazıya dönüştürür ve böylece daha hızlı olur. Ancak, aşağıdaki nedenlerden dolayı bu güvenli değildir: Bu kadar büyük bir yazma göz önüne alındığında, disk dahili olarak zamanlama yapabilir ve büyük yazmanın küçük parçalarını herhangi bir sırayla tamamlayabilir. Böylece, disk dahili olarak (1) TxB, I[v2], B[v2], ve TxE yazabilir ve ancak daha sonra Db yazabilir. Ne yazık ki, disk (1) ve (2) arasında güç kaybederse, diskte sona eren şey budur:



Bu neden bir problem? Peki, işlem geçerli bir işlem gibi görünüyor (eşleşen sıra numaralarıyla bir başlangıcı ve sonu var). Dahası, dosya sistemi bu dördüncü bloğa bakamaz ve yanlış olduğunu bilemez; sonuçta, keyfi kullanıcı verileridir. Böylece, sistem şimdi yeniden başlatılır ve kurtarma işlemini çalıştırırsa, bu işlemi yeniden yürütücek ve '??' çöp bloğunun içeriğini Db'nin yaşaması gereken konuma cahilce kopyalar. Bu, bir dosyadaki keyfi kullanıcı verileri için kötüdür; dosya sistemini monte edilemez hale getirebilecek süper blok gibi kritik bir dosya sistemi parçasına olursa çok daha kötü olur.

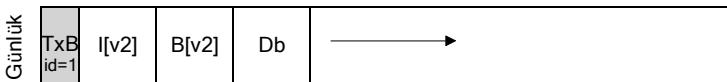
BİR YANA: GÜNLÜK YAZMALARINI OPTİMİZE ETMEK

Günlüğe yazmanın belirli bir verimsizliğini fark etmiş olabilirsiniz. Yani, dosya sistemi önce işlem-başlangıç bloğunu ve işlemin içeriğini yazmalıdır; ancak bu yazma işlemleri tamamlandıktan sonra dosya sistemi işlem-sonu bloğunu diske gönderebilir. Bir diskin nasıl çalıştığını düşünürseniz, performans etkisi açıkta: Genellikle fazladan bir rotasyon meydana gelir (nedenini düşünün).

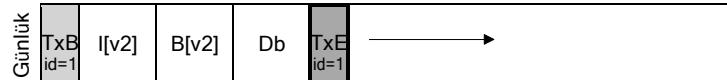
Eski yüksek lisans öğrencilerimizden biri olan Vijayan Prabhakaran'ın, bu sorunu çözmek için basit bir fikri vardı [P+05]. Günlüğe bir işlem yazarken, günlükün içeriğinin bir sağlama toplamını başlangıç ve bitiş bloklarına ekleyin. Bunu yapmak, dosya sisteminin beklemeye gerek kalmadan tüm işlemi bir kerede yazmasını sağlar; kurtarma sırasında, dosya sistemi hesaplanan sağlama toplamında işlemde depolanan sağlama toplamı ile uyumsuzluk görürse, işlemin yazılması sırasında bir kilitlenmeoluştuğu sonucuna varabilir ve böylece dosya-sistemi güncellemesi atabilir. Böylece, yazma protokolünde ve kurtarma sisteminde küçük bir ince ayar ile, dosya sistemi daha hızlı ortak-durum performansı elde edebilir; bunun da ötesinde, günlüğten yapılan tüm okumalar artık bir sağlama toplamı tarafından korunduğundan sistem biraz daha güvenlidir.

Bu basit düzeltme Linux dosya sistemi geliştiricilerinin dikkatini çekecek kadar çekiciydi ve daha sonra onu (tahmin ettiniz!) **Linux ext4** adlı yeni nesil Linux dosya sistemine dahil ettiler. Artık Android el tipi platform da dahil olmak üzere dünya çapında milyonlarca makinede gönderim yapıyor. Böylece, birçok Linux tabanlı sisteme diske her yazdığınızda, Wisconsin'de geliştirilen küçük bir kod, sisteminizi biraz daha hızlı ve güvenilir hale getirir

Bu sorunu önlemek için, dosya sistemi işlemsel yazmayı iki adımda gerçekleştirir. İlk olarak, TxE bloğu dışındaki tüm blokları günlüğe yazar, bu yazıların hepsini bir kerede yayarlar. Bu yazmalar tamamlandığında, günlük şuna benzer (ekleme iş yükümüzü yeniden varsayırsak):



Bu yazmalar tamamlandığında, dosya sistemi TxE bloğunun yazımını yayarlar ve böylece günlüğü bu son, güvenli durumda bırakır:



Bu işlemin önemli bir yönü, disk tarafından sağlanan atomiklik garantisidir. Diskin, herhangi bir 512 baylık yazmanın gerçekleşeceğini

veya gerçekleşmeyeceğini (ve asla yarı yazılmayıacığını) garanti ettiği ortaya çıktı; bu nedenle, TxE'nin yazılmasının atomik olduğundan emin olmak için, onu tek bir 512 baytlık blok haline getirmelisiniz. Böylece, dosya sistemini güncellemek için mevcut protokolümüz, üç aşamasında her biri etiketlenmiş olarak:

1. **Günlük Yazma (Journal write):** İşlemin içeriğini (TxB, meta veriler, ve veriler dahil) günlüğe yazın; bu yazmaların tamamlanmasını bekleyin.
2. **Günlük İşlenme (Journal commit):** İşlem taahhüt bloğunu (Tx E içeren) günlüğe yazın; yazmanın tamamlanmasını bekleyin; işlemin işlendiği (**committed**) söylenir.
3. **Kontrol Noktası (Checkpoint):** Güncellemenin içeriğini (meta veriler ve veriler) diskteki son konumlarına yazın.

Kurtarma (Recovery)

Şimdi bir dosya sisteminin bir kilitlenmeden **kurtulmak (recover)** için günlüğün içeriğini nasıl kullanabileceğini anlayalım. Bu güncelleştirme dizisi sırasında herhangi bir zamanda kilitlenme meydana gelebilir. Kilitlenme, işlem günlüğüne güvenli bir şekilde yazılmadan önce (yani, yukarıdaki Adım 2 tamamlanmadan önce) gerçekleşirse, o zaman işimiz kolaydır: bekleyen güncelleme basitçe atlanır. Kilitlenme, işlem günlüğe işlendikten sonra ancak kontrol noktası tamamlanmadan önce gerçekleşirse, dosya sistemi güncelleştirmeyi aşağıdaki gibi **kurtarabilir (recover)**. Sistem önyüklendiğinde, dosya sistemi kurtarma işlemi günlüğü tarar ve diske kaydedilmiş işlemleri tarar; Böylece bu işlemler **yeniden oynatılır (replayed)** (sırayla), dosya sistemi yeniden işlemdeki blokları diskteki son konumlarına yazmaya çalışır. Bu günlüğe kaydetme biçimini, var olan en basit biçimlerden biridir ve **yineleme günlüğü (redo logging)** olarak adlandırılır. Dosya sistemi, günlükte işlenen işlemleri kurtararak, diskteki yapıların tutarlı olmasını sağlar ve böylece dosya sistemini bağlayarak ve kendisini yeni istekler için hazırlayarak devam edebilir.

Kontrol noktası sırasında herhangi bir noktada, blokların son konumlarına yapılan bazı güncellemeler tamamlandıktan sonra bile bir kilitlenmenin meydana gelmesinin sorun olmadığını unutmayın. En kötü durumda, bu güncellemelerden bazıları kurtarma sırasında basitçe tekrar gerçekleştirilebilir. Kurtarma nadir görülen bir işlem olduğundan (yalnızca beklenmeyen bir sistem kilitlenmesinden sonra gerçekleşir), birkaç gereksiz yazma işlemi endişelenenecek bir şey değildir³.

Toplu Günlük Güncellemeleri (Batching Log Updates)

Temel protokolün çok fazla ekstra disk trafiği ekleyebileceğini fark etmiş olabilirsiniz. Örneğin, aynı dizinde dosya1 ve dosya2 adlı bir satırda iki dosya oluşturduğumuzu düşünün.

³Her şey için endişelendiniz sürece, bu durumda size yardımcı olamayız. Bu kadar endişelenmemeyi bırak, bu sağılsız! Ama şimdilik muhtemelen aşırı endişelenme konusunda endişeleniyorsun.

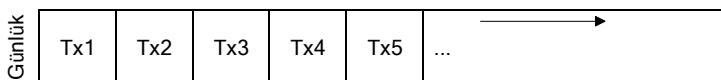
Bir dosya oluşturmak için, en az aşağıdakiler de dahil olmak üzere bir dizi disk yapınızı güncellemek gereklidir: inode bitmap'i (yeni bir inode tahsis etmek için), dosyanın oluşturulan inode'u, yeni dizin girişini içeren üst dizinin veri bloğu ve (artık yeni bir değişiklik zamanına sahip) üst dizin inode'u. Günlük kaydı ile, tüm bu bilgileri mantıksal olarak iki dosya oluşturma işlemimizin her biri için günlüğe işleriz; çünkü dosyalar aynı dizindedir ve hatta aynı inode bloğu içinde inode'ları olduğunu varsayırsak, bu eğer dikkatli olmazsa, aynı blokları tekrar tekrar yazacağımız anlamına gelir.

Bu sorunu çözmek için, bazı dosya sistemleri her güncellemeye diske teker teker işlemmez (örneğin, Linux ext3); bunun yerine, tüm güncellemleri küresel bir işlemde arabelleğe alabilirsiniz. Yukarıdaki örneğimizde, iki dosya oluşturulduğunda, dosya sistemi sadece bellek içi inode bir eşleniğini, dosyaların inode'larını, dizin verilerini ve dizin inode'unu kirli olarak işaretler, ve bunları geçerli işlemi oluşturan bloklar listesine ekler. Sonunda bu blokları diske yazma zamanı geldiğinde (örneğin, 5 saniyelik bir zaman aşımından sonra), yukarıda açıklanan tüm güncellemleri içeren bu tek genel işlem gerçekleştirilir. Böylece, güncelleştirmeleri arabelleğe alarak, bir dosya sistemi çoğu durumda diske aşırı yazma trafigini önleyebilir.

Günlüğü Sonlu Hale Getirme (Making The Log Finite)

Böylece, disk üzerindeki dosya sistemi yapılarını güncellemek için temel bir protokole ulaştık. Dosya sistemi, bellekteki güncellemleri bir süre arabelleğe alır; Nihayet diske yazma zamanı geldiğinde, dosya sistemi önce işlemin ayrıntılarını dikkatlice günlüğe yazar (diğer adıyla ileri yazma günlüğü); işlem tamamlandıktan sonra, dosya sistemi bu blokları diskteki son konumlarına göre kontrol eder.

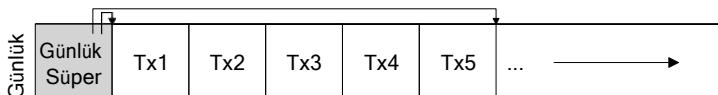
Ancak, günlük sonlu bir boyuttadır. Üzerine işlemleri eklemeye devam ederse (bu şekilde olduğu gibi), yakında dolacaktır. Sizce o zaman ne olur?



Günlük dolduğunda iki sorun ortaya çıkar. İlk sorun basit ama daha az kritik: Günlük ne kadar büyükse, kurtarma işleminin kurtarmak için günlükteki tüm işlemleri (sırayla) yeniden yürütmesi gerektiğinden, kurtarma işlemi o kadar uzun süreler. İkinci sorun: Günlük dolu olduğunda (veya neredeyse doluyken), diske başka işlem yapılamaz, böylece dosya sistemi “kullanışlarından daha az” (yani işe yaramaz) hale gelir.

Bu sorunları gidermek için, günlük kaydı dosya sistemleri günlüğü dairesel bir veri yapısı olarak ele alır ve günlüğü tekrar tekrar kullanır; bu nedenle günlük bazen **dairesel günlük (circular log)** olarak anılır. Bunu yapmak için, dosya sistemi bir kontrol noktasından bir süre sonra harekete geçmesi gereklidir. Özellikle, bir işlem kontrol noktasından geçirildikten sonra, dosya sistemi, günlükte kapladığı alanı boşaltmalı ve günlük alanının

yeniden kullanılmasına izin vermelidir. Bu amaca ulaşmanın birçok yolu var; örneğin, günlüğteki en eski ve en yeni kontrol noktası olmayan işlemleri bir **günlük süper bloğunda (journal superblock)** işaretleyebilirsiniz; diğer tüm alanlar boştur. İşte grafiksel bir tasvir:



Günlük süper bloğunda (ana dosya sistemi süper bloğu ile karıştırılmamalıdır), günlük kaydı sistemi, hangi işlemleri henüz kontrol edilmediğini bilmek için yeterli bilgiyi kaydeder ve böylece kurtarma süresini kısaltır ve günlüğün dairesel bir şekilde yeniden kullanılmasını sağlar. Ve böylece temel protokolümüze bir adım daha ekliyoruz:

- Günlük Yazma (Journal Write):** İşlemin içeriğini (TxB ve güncellemenin içeriğini içeren) günlüğe yazın; bu yazmaların tamamlanmasını bekleyin.
- Günlük İşlenme (Journal commit):** İşlem taahhüt bloğunu (TxE içeren) günlüğe yazın; yazmanın tamamlanmasını bekleyin; işlem artık **işlenmiştir (committed)**.
- Kontrol Noktası (Checkpoint):** Güncellemenin içeriğini dosya sistemindeki son konumlara yazın.
- Bos (Free):** Bir süre sonra, günlük süper bloğunu güncelliyerek işlemi günlüğe boş olarak işaretleyin.

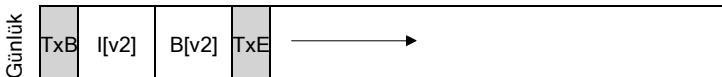
Böylece son veri günlüğü protokolümüze sahibiz. Fakat hala bir sorun var: her veri bloğunu diske *iki kez* yazıyoruz, bu da özellikle sistem kilitlenmesi gibi nadir bir şey için ödemesi gereken ağır bir maliyettir. Verileri iki kez yazmadan tutarlılığı korumanın bir yolunu bulabilir misiniz?

Meta Veri Günlüğe Kaydetme (Metadata Journaling)

Kurtarma artık hızlı olmasına rağmen (tüm disk taramak yerine günlüğü tarama ve birkaç işlemi yeniden yürütme), dosya sisteminin normal çalışması arzu edebileceğimizden daha yavaştır. Özellikle, diske her yazma için, şimdi önce günlüğe yazıyoruz, böylece yazma trafığını ikiye katlıyoruz; Bu ikiye katlama, artık sürücünün en yüksek yazma bant genişliğinin yarısında devam edecek olan sıralı yazma iş yükleri sırasında özellikle acı vericidir. Ayrıca, günlüğe yazmalar ile ana dosya sisteme yazmalar arasında, bazı iş yükleri gözle görülür ek yük oluşturan maliyetli bir arama vardır.

Her veri bloğunu diske iki kez yazmanın maliyeti nedeniyle, insanlar performansı hızlandırmak için birkaç farklı şey denediler. Örneğin, yukarıda tanımladığımız günlüğe kaydetme modu, tüm kullanıcı verilerini günlüğe kaydettiğinden (dosya sisteminin meta verilerine ek olarak) genellikle **veri günlük kaydı (data journaling)** olarak adlandırılır (Linux ext3'te olduğu gibi). Daha basit (ve daha yaygın) bir günlük tutma biçimine bazen **sıralı günlük kaydı(ordered journaling)** (veya sadece **meta veri**

günlük kaydı(metadata journaling) denir ve kullanıcı verilerinin günlüğe *yazılmaması* dışında neredeyse aynıdır. Böylece, yukarıdaki güncellemenin aynısını yaparken, aşağıdaki bilgiler günlüğe yazılacaktır.



Daha önce günlüğe yazılan veri bloğu Db, bunun yerine dosya sistemine uygun şekilde yazılır ve fazladan yazmadan kaçınır; diske gelen G/C trafiginin çoğunuveri olduğu göz önüne alındığında, verileri iki kez yazmamak, günlük kaydının G/C yükünü önemli ölçüde azaltır. Yine de değişiklik ilginç bir soruyu gündeme getiriyor: Veri bloklarını diske ne zaman yazmalıyız?

Sorunu daha iyi anlamak için bir dosya ekleme örneğimizi tekrar ele alalım. Güncellemeye üç bloktan oluşur: I[v2], B[v2], ve Db. İlk ikisi hem meta verilerdir hem de günlüğe kaydedilir ve ardından kontrol noktası işaretlenir; ikincisi dosya sistemine yalnızca bir kez yazılacaktır. Db'yi diske ne zaman yazmalıyız? Önemli mi?

Görünüşe göre, veri yazmanın sırası yalnızca meta veriler günlüğü için önemlidir. Örneğin, işlem tamamlandıktan *sonra* (I[v2] ve B[v2] içeren) diske Db yazarsak nolur? Ne yazık ki, bu yaklaşımın bir sorunu var: Dosya sistemi tutarlıdır, ancak I[v2] çöp verilerine işaret edebilir. Özellikle, I[v2] ve B[v2] yazıldığı ancak Db'nin diske girmediği durumu düşünün. Dosya sistemi daha sonra kurtarmaya çalışacaktır. Db günlüğe *olmadığı* için, dosya sistemi yazmaları I[v2] ve B[v2]'ye yeniden yürütür ve tutarlı bir dosya sistemi üretir (dosya sistemi meta verileri açısından). Ancak, I[v2] çöp verilerine, yani, Db'nin yöneldiği yuvada ne varsa ona işaret ediyor olacak.

Bu durumun ortaya çıkmasını sağlamak için, bazı dosya sistemleri (örneğin, Linux ext3) ilgili meta veriler diske yazılmadan *önce*, önce diske veri bloklarını (normal dosyalardan) yazar. Özellikle, protokol aşağıdaki gibi biridir:

- Veri Yazma (Data write):** Verileri son konuma yazın; tamamlanmasını bekleyin (bekleme isteği bağlıdır; ayrıntılar için aşağıya bakın).
- Günlük Meta Verisi Yazma (Journal metadata write):** Başlangıç bloğunu ve meta verileri günlüğe yazın; yazma işlemlerinin tamamlanmasını bekleyin.
- Günlük İşlenme (Journal commit):** İşlem taahhüt bloğunu (TxE içeren) günlüğe yazın; yazmanın bitmesini bekleyin; işlem (veriler dahil) artık **işlenmiştir (committed)**.
- Kontrol Noktası Meta Verileri (Checkpoint metadata):** Meta veri güncellemesinin içeriğini dosya sistemindeki son konumlara yazın.
- Boş (Free):** Daha sonra, günlük süper bloğunda işlemi boş olarak işaretleyin.

Önce veri yazmayı zorlayarak, bir dosya sistemi bir işaretçinin asla çöpe işaret etmeyeceğini garanti edebilir. Gerçekten de, bu “*işaret edilen nesneyi, ona işaret eden nesneden önce yazın*” kuralı, kilitlenme tutarlılığının merkezinde yer alır ve diğer kilitlenme tutarlılığı şemaları [GP94] tarafından daha da sömürüller (ayıntılar için aşağıya bakın).

Çoğu sistemde, meta veri günlük kaydı (ext3'ün sıralı günlük kaydına benzer) tam veri günlük kaydından daha popülerdir. Örneğin, Windows NTFS ve SGI'ın XFS bir tür meta veri günlük kaydı kullanır. Linux ext3 size veri, sıralı veya sırasız modlardan birini seçme seçeneği sunar (sırasız modda, veriler istediğiniz zaman yazılabilir). Tüm bu modlar, meta verileri tutarlı tutar; veri anımlarında farklılık gösterirler.

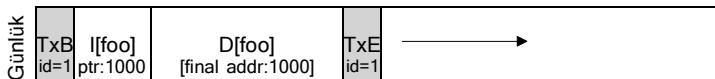
Son olarak, yukarıdaki protokolde belirtildiği gibi, günlüğe yazma işlemi yapmadan (adım 2) önce veri yazmayı tamamlamaya zorlamanın (adım 1) doğruluk için gerekli olmadığını unutmayın. Özellikle, aynı anda verilere, işlem başlangıç bloğuna ve günlüğe kaydedilen meta verilere yazma işlemleri yayımlamak iyi olur; tek gerçek gereklilik, adım 1 ve 2'nin günlük taahhüt bloğunun yayınlanmasından önce tamamlanmasıdır (adım 3).

Zor Durum: Yeniden Kullanımı Engelle (Tricky Case : Block Reuse)

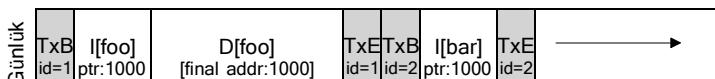
Günlük kaydını daha zor hale getiren ve bu nedenle tartışmaya değer bazı ilginç köşe durumları vardır. Bunların bir kısmı blok yeniden kullanımı etrafında döner; Stephen Tweedie'nin (ext3'ün arkasındaki ana güçlerden biri) dediği gibi:

"Tüm sistemin korkunç kısmı nedir? ... dosyaları siliyor. Silmekle ilgili her şey tehlikeli. Silme ile ilgili her şey... bloklar silinir ve sonra yeniden tahsis edilirse ne olacağına dair kabuslar görürsünüz" [TOO]

Tweedie'nin verdiği özel örnek aşağıdaki gibidir. Bir tür meta veri günlüğü kullandığınızı varsayıyalım (ve bu nedenle dosyalar için veri blokları günlüğe kaydedilmez). Diyelim ki `foo` adlı bir diziniz var. Kullanıcı `foo`'ya bir girdi ekler (diyelim ki bir dosya oluşturarak), ve böylece `foo`'nın içeriği (çünkü dizinler meta veri olarak kabul edilir) günlüğe yazılır; `foo` dizin verilerinin konumunun blok 1000 olduğunu varsayıyalım. Bu nedenle günlük, buna benzer bir şey içerir:



Bu noktada, kullanıcı dizindeki her şeyi ve kendisini siler ve 1000 numaralı bloğu yeniden kullanım için serbest bırakır. Son olarak, kullanıcı yeni bir dosya (`bar` diyelim) oluşturur ve bu da `foo`'ya ait olan aynı bloğu (1000) yeniden kullanılır. `bar` inode'u, verileri gibi diske de bağlıdır; Bununla birlikte, meta veri günlük kaydı kullanımında olduğundan, günlüğe yalnızca `bar`'ın inode'u taahhüt edilir; `bar` dosyasındaki 1000 numaralı blokta yeni yazılan veriler günlüğe kaydedilmez.



TxB	Günlük İçerik (meta veri) (veri)		TxE	Dosya sistemi	
İşlem tamamlandı	İşlem	İşlem		Meta veri	Veri
		tamamlandı			
			tamamlandı		
				İşlem tamamlandı	
					İşlem tamamlandı

Figure 42.1: Veri günlüğü kaydı zaman çizelgesi

Şimdi bir kilitlenme olduğunu ve tüm bilgilerin hala günlüğekte olduğunu varsayıyalım. Yeniden oynarma sırasında, kurtarma işlemi, blok 1000' deki dizin verilerinin yazılması da dahil olmak üzere günlükteki her şeyi yeniden oynatır; böylece yeniden oynatma, mevcut `bar` dosyasının kullanıcı verilerinin üzerine eski dizin içeriklerini yazar! Açıkçası bu doğru bir kurtarma işlemi değildir ve kesinlikle `bar` dosyasını okurken kullanıcı için bir sürpriz olacaktır.

Bu sorunun bir dizi çözümü var. Örneğin, söz konusu blokların silinmesi günlüğten kontrol noktasıyla işaretleneneye kadar bloklar asla yeniden kullanılmaz. Linux ext3'ün bunun yerine yaptığı şey, günlüğe **iptal (revoke)** kaydı olarak bilinen yeni bir kayıt türü eklemektir. Yukarıdaki durumda, dizinin silinmesi günlüğe bir iptal kaydının yazılmasına neden olur. Günlüğü tekrar oynatırken, sistem önce bu tür iptal kayıtlarını tarar; Bu tür iptal edilen veriler asla tekrar oynatılmaz, böylece yukarıda belirtilen sorundan kaçınılabilir.

Günlük Kaydını Tamamlama: Bir Zaman Çizelgesi (Wrapping Up Journaling: A Timeline)

Günlük kaydı tartışmamıza son vermeden önce, tartıştığımız protokollerin her birini gösteren zaman çizelgeleriyle özetliyoruz. Şekil 42.1 verileri ve meta verileri günlüğe kaydeden protokolü gösterirken, Şekil 42.2 yalnızca meta verileri günlüğe kaydeden protokolü gösterir.

Her şekilde, zaman aşağı yönde artar ve şekilde her satır, bir yazma işleminin yapılabileceği veya tamamlanabileceği mantıksal zamanı gösterir. Örneğin, veri günlüğe kaydetme protokolünde (Şekil 42.1), işlemin yazmaları başlangıç bloğu (TxB) ve işlemin içeriği mantıksal olarak aynı anda yayınlanabilir ve böylece herhangi bir sırada tamamlanabilir; ancak, işlem bitiş bloğuna yazma (TxE), söz konusu önceki yazılar tamamlana kadar yayınlanmamalıdır. Benzer şekilde, kontrol noktası verilere yazar ve meta veri blokları, işlem sonu bloğu işlenene kadar başlamaz. Yatay kesikli çizgiler, yazma sırası gereksinimlerine nerede uyulması gerektiğini gösterir.

Meta veri günlüğe kaydetme protokolü için benzer bir zaman çizelgesi gösterilir. Veri yazma işleminin mantıksal olarak işleme yazma

TxB	Günlük İçerik (meta veri)	TxE	Dosya Sistemi
İşlem	İşlem		İşlem
tamamlandı			tamamlandı
	tamamlandı		
		İşlem	
		tamamlandı	
			İşlem
			tamamlandı

Figure 42.2: Meta veri Günlük Kaydı Zaman Çizelgesi

işlemlerinin başlaması ve günlüğün içeriği ile aynı anda verilebileceğini unutmayın; ancak, işlem sona ermeden önce düzenlenmeli ve tamamlanmalıdır.

Son olarak, zaman çizelgelerindeki her yazma için işaretlenen tamamlanma süresinin keyfi olduğunu unutmayın. Gerçek bir sistemde, tamamlanma süresi, performansı artırmak için yazmaları yeniden sıralayabilen G/Ç alt sistemi tarafından belirlenir. Sahip olduğumuz sıralamaya ilgili tek garanti, protokol doğruluğu için uygulanması gerekenlerdir (ve şekillerde yatay kesikli çizgilerle gösterilir).

42.4 Çözüm #3: Diğer Yaklaşımlar (Other Approaches)

Şimdiye kadar dosya sistemi meta verilerini tutarlı tutmak için iki seçenek açıkladık: `fsck`'ye dayanan tembel bir yaklaşım ve günlük kaydı olarak bilinen daha aktif bir yaklaşım. Ancak, bunlar sadece iki yaklaşım değildir. Esnek güncellemeler (Soft Updates) [GP94] olarak bilinen böyle bir yaklaşım, Ganger and Patt tarafından tanıtıldı. Bu yaklaşım, diskteki yapıların hiçbir zaman tutarsız bir durumda bırakılmamasını sağlamak için dosya sistemine tüm yazmaları dikkatli bir şekilde sıralar. Örneğin, ona işaret eden `inode`'dan önce diske işaret eden bir veri bloğunu yazarak, `inode`'un asla çöpe işaret etmemesini sağlarız; dosya sisteminin tüm yapıları için benzer kurallar türetiliblir. Bununla birlikte, esnek güncellemeleri uygulamak zor olabilir; yukarıda açıklanan günlük kaydı katmanı, tam dosya sistemi yapıları hakkında nispeten az bilgi ile uygulanabilirken, esnek güncellemeler, her dosya sistemi veri yapısı hakkında karmaşık bilgi gerektirir ve bu nedenle sisteme makul miktarda karmaşalık ekler.

Diğer bir yaklaşım **yazma üzerine kopyalama** (**copy-on-write**) (evet, COW), olarak bilinir ve Sun'ın ZFS [B07] de dahil olmak üzere bir dizi popüler dosya sisteminde kullanılır. Bu teknik hiçbir zaman dosyaların veya dizilerin üzerine yazmaz; bunun yerine, diskte daha önce kullanılmayan konumlara yeni güncellemeler yerleştirir. Bir dizi güncelleme tamamladıktan sonra, COW dosya sistemleri yeni güncellenen yapılara işaretçiler eklemek için dosya sisteminin kok yapısını çevirir. Bunu yapmak, dosya sisteminin tutarlı kalmasını kolaylaştırır. Gelecekteki bir bölümde günlük yapılandırılmış dosya sistemi

(log-structured file system) (LFS) tartışırken bu teknik hakkında daha fazla bilgi edineceğiz; LFS, bir COW'un erken bir örneğidir.

Başka bir yaklaşım da Wisconsin'de yeni geliştirdiğimiz bir yaklaşımındır. **Geri işaretçi tabanlı tutarlılık (backpointer-based consistency)** (veya **GiTT (BBC)**) başlıklı bu teknikte, yazmalar arasında sıralama yapılmaz. Tutarlılığı sağlamak için, sistemdeki her bloğa bir geri işaretçi (**back Pointer**) eklenir; örneğin, her veri bloğunun ait olduğu inode için bir referans vardır. Bir dosyaya erişirken, dosya sistemi, ileri işaretçinin (örneğin, inode veya doğrudan bloktaki adres) dosyaya geri başvurulan bir bloğa işaret edip etmediğini kontrol ederek dosyanın tutarlı olup olmadığını belirleyebilir. Eğer öyleyse, her şey güvenli bir şekilde diske ulaşmış olmalı ve bu nedenle dosya tutarlıdır; değilse, dosya tutarsızdır ve bir hata döndürür. Dosya sistemine geri işaretçiler ekleyerek, yeni bir tembel kilitlenme tutarlılığı biçimini elde edilebilir [C+12].

Son olarak, bir günlük protokolünün disk yazmalarının tamamlanması için beklemesi gereken süreyi azaltan teknikleri de araştırdık. **İyimser kilitlenme tutarlılığı (optimistic crash consistency)** [C+13] olarak adlandırılan bu yeni yaklaşım, **işlem sağlama toplamının (transaction checksum)** genelleştirilmiş bir biçimini [P+05] kullanarak diske mümkün olduğunda çok yazma işlemi gerçekleştirir ve ortaya çıkışları durumunda tutarsızlıkları tespit etmek için birkaç başka teknik içerir. Bazı iş yükleri için bu iyimser teknikler performansı büyülüksüz sırasına göre artırabilir. Ancak, gerçekten iyi çalışması için, biraz farklı bir disk arayüzü gereklidir [C+13].

42.5 Özet (Summary)

Kilitlenme tutarlılığı sorununu ortaya koyduk ve bu soruna saldırmak için çeşitli yaklaşımları tartıştık. Bir dosya sistemi kontrolcüsü oluşturmanın eski yaklaşımı işe yarar, ancak modern sistemlerde kurtarılmayacak kadar yavaştır. Bu nedenle, birçok dosya sistemi artık günlük kaydı kullanmaktadır. Günlük kaydı, kurtarma süresini O'dan (disk hacminin boyutu) O'ya (günlük boyutu) düşürür, böylece bir kilitlenme ve yeniden başlatmadan sonra kurtarma işlemini önemli ölçüde hızlandırır. Bu nedenle, birçok modern dosya sistemi günlük kaydını kullanır. Günlük kaydının birçok farklı biçimde gelebileceğini de gördük; en sık kullanılan, hem dosya sistemi meta verileri hem de kullanıcı verileri için makul tutarlılık garantilerini korurken, günlüğe giden trafik miktarını azaltan sıralı meta veri günlük kaydırır. Sonunda, kullanıcı verileri üzerinde güclü garantiler muhtemelen sağlanması gereken en önemli şeylerden biridir; garip bir şekilde, son araştırmaların gösterdiği gibi, bu alan devam eden bir çalışma olmaya devam ediyor. [P+14].

References

- [B07] “ZFS: The Last Word in File Systems” by Jeff Bonwick and Bill Moore. Available online: http://www.ostep.org/Citations/zfs_last.pdf. *ZFS uses copy-on-write and journaling, actually, as in some cases, logging writes to disk will perform better.*
- [C+12] “Consistency Without Ordering” by Vijay Chidambaram, Tushar Sharma, Andrea C. Arpacı-Dusseau, Remzi H. Arpacı-Dusseau. FAST ’12, San Jose, California. *A recent paper of ours about a new form of crash consistency based on back pointers. Read it for the exciting details!*
- [C+13] “Optimistic Crash Consistency” by Vijay Chidambaram, Thanu S. Pillai, Andrea C. Arpacı-Dusseau, Remzi H. Arpacı-Dusseau. SOSP ’13, Nemacolin Woodlands Resort, PA, November 2013. *Our work on a more optimistic and higher performance journaling protocol. For workloads that call `fsync()` a lot, performance can be greatly improved.*
- [GP94] “Metadata Update Performance in File Systems” by Gregory R. Ganger and Yale N. Patt. OSDI ’94. *A clever paper about using careful ordering of writes as the main way to achieve consistency. Implemented later in BSD-based systems.*
- [G+08] “SQCK: A Declarative File System Checker” by Haryadi S. Gunawi, Abhishek Rajimwale, Andrea C. Arpacı-Dusseau, Remzi H. Arpacı-Dusseau. OSDI ’08, San Diego, California. *Our own paper on a new and better way to build a file system checker using SQL queries. We also show some problems with the existing checker, finding numerous bugs and odd behaviors, a direct result of the complexity of `fsck`.*
- [H87] “Reimplementing the Cedar File System Using Logging and Group Commit” by Robert Hagmann. SOSP ’87, Austin, Texas, November 1987. *The first work (that we know of) that applied write-ahead logging (a.k.a. journaling) to a file system.*
- [M+13] “ffsck: The Fast File System Checker” by Ao Ma, Chris Dragga, Andrea C. Arpacı-Dusseau, Remzi H. Arpacı-Dusseau. FAST ’13, San Jose, California, February 2013. *A recent paper of ours detailing how to make `fsck` an order of magnitude faster. Some of the ideas have already been incorporated into the BSD file system checker [MK96] and are deployed today.*
- [MK96] “Fsck – The Unix File System Check Program” by Marshall Kirk McKusick and T. J. Kowalski. Revised in 1996. *Describes the first comprehensive file-system checking tool, the eponymous `fsck`. Written by some of the same people who brought you FFS.*
- [MLF84] “A Fast File System for UNIX” by Marshall K. McKusick, William N. Joy, Sam J. Leffler, Robert S. Fabry. ACM Transactions on Computing Systems, Volume 2:3, August 1984. *You already know enough about FFS, right? But come on, it is OK to re-reference important papers.*
- [P+14] “All File Systems Are Not Created Equal: On the Complexity of Crafting Crash-Consistent Applications” by Thanumalayan Sankaranarayana Pillai, Vijay Chidambaram, Ramnathan Alagappan, Samer Al-Kiswany, Andrea C. Arpacı-Dusseau, Remzi H. Arpacı-Dusseau. OSDI ’14, Broomfield, Colorado, October 2014. *A paper in which we study what file systems guarantee after crashes, and show that applications expect something different, leading to all sorts of interesting problems.*
- [P+05] “IRON File Systems” by Vijayan Prabhakaran, Lakshmi N. Bairavasundaram, Nitin Agrawal, Haryadi S. Gunawi, Andrea C. Arpacı-Dusseau, Remzi H. Arpacı-Dusseau. SOSP ’05, Brighton, England, October 2005. *A paper mostly focused on studying how file systems react to disk failures. Towards the end, we introduce a transaction checksum to speed up logging, which was eventually adopted into Linux ext4.*
- [PAA05] “Analysis and Evolution of Journaling File Systems” by Vijayan Prabhakaran, Andrea C. Arpacı-Dusseau, Remzi H. Arpacı-Dusseau. USENIX ’05, Anaheim, California, April 2005. *An early paper we wrote analyzing how journaling file systems work.*
- [R+11] “Coerced Cache Eviction and Discreet-Mode Journaling” by Abhishek Rajimwale, Vijay Chidambaram, Deepak Ramamurthy, Andrea C. Arpacı-Dusseau, Remzi H. Arpacı-Dusseau. DSN ’11, Hong Kong, China, June 2011. *Our own paper on the problem of disks that buffer writes in a memory cache instead of forcing them to disk, even when explicitly told not to do that! Our solution to overcome this problem: if you want A to be written to disk before B, first write A, then send a lot of “dummy” writes to disk, hopefully causing A to be forced to disk to make room for them in the cache. A neat if impractical solution.*

[T98] “Journaling the Linux ext2fs File System” by Stephen C. Tweedie. The Fourth Annual Linux Expo, May 1998. *Tweedie did much of the heavy lifting in adding journaling to the Linux ext2 file system; the result, not surprisingly, is called ext3. Some nice design decisions include the strong focus on backwards compatibility, e.g., you can just add a journaling file to an existing ext2 file system and then mount it as an ext3 file system.*

[TOO] “EXT3, Journaling Filesystem” by Stephen Tweedie. Talk at the Ottawa Linux Symposium, July 2000. olstrans.sourceforge.net/release/OLS2000-ext3/OLS2000-ext3.html *A transcript of a talk given by Tweedie on ext3.*

[T01] “The Linux ext2 File System” by Theodore Ts’o, June, 2001.. Available online here: <http://e2fsprogs.sourceforge.net/ext2.html>. *A simple Linux file system based on the ideas found in FFS. For a while it was quite heavily used; now it is really just in the kernel as an example of a simple file system.*

Ödev (Similasyon)

Bu bölümde, dosya sistemi bozulmalarının nasıl tespit edilebileceğini (ve potansiyel olarak onarılmış) daha iyi anlamak için kullanabileceğiniz basit bir simülör olan `fsck.py` tanıtılmaktadır. Simülörün nasıl çalıştırılacağı hakkında ayrıntılar için lütfen README'ye bakın.

Sorular

- İlk olarak, `fsck.py -D` dosyasını çalıştırın; bu bayrak herhangi bir tutarsızlığı kapatır ve böylece rastgele bir dosya sistemi oluşturmak için kullanabilir ve orada hangi dosya ve dizinlerin olduğunu belirleyip bakabilirsiniz. Öyleyse, devam edin ve bunu yapın! Haklı olup olmadığını görmek için `-p` bayrağını kullanın. Tohumu (-s) 1,2 ve 3 gibi farklı değerlere ayarlayarak bunu rastgele oluşturulmuş birkaç farklı dosya sistemi için deneyin.

```
venes@ubuntu:~/Desktop/ostep/ostep-homework/file-journaling$ python3 fsck.py -D
ARG seed 0
ARG seedCorrupt 0
ARG numInodes 16
ARG numData 16
ARG numRequests 15
ARG printFinal False
ARG whichCorrupt -1
ARG dontCorrupt True

Final state of file system:

inode bitmap 1000100010000101
inodes [d a:0 r:4] [] [] [] [d a:12 r:2] [] [] [d a:0 r:2] [] [] [f a:-1 r:2] [] [f a:-1 r:1]
data bitmap 1000001000001000
data [(..,0) (..,0) (g,8) (w,4) (n,13) (z,13)] [] [] [] [] [(..,8) (..,0) (s,15)] [] [] [] [] [(..,4) (..,0)] [] [] []

Can you figure out which files and directories exist?
```

`-D` bayrağı dosya sisteminin bozulmalarını kapatır ve böylelikle bozuk olan dosya sistemiyle karşılaşlığımızda bozulmayı tanımlayabiliriz. Bozulmanın olmadığını `dontCorrupt True` olduğunda anlıyoruz eğer `False` ise dosya sistemi bozulmaya uğramıştır.

```
venes@ubuntu:~/Desktop/ostep/ostep-homework/file-journaling$ python3 fsck.py -p
ARG seed 0
ARG seedCorrupt 0
ARG numInodes 16
ARG numData 16
ARG numRequests 15
ARG printFinal True
ARG whichCorrupt -1
ARG dontCorrupt False

Final state of file system:

inode bitmap 1000100010000101
inodes [d a:0 r:4] [] [] [] [d a:12 r:2] [] [] [d a:7 r:2] [] [] [f a:-1 r:2] [] [f a:-1 r:1]
data bitmap 1000001000001000
data [(..,0) (..,0) (g,8) (w,4) (n,13) (z,13)] [] [] [] [] [(..,8) (..,0) (s,15)] [] [] [] [] [(..,4) (..,0)] [] [] []

Can you figure out how the file system was corrupted?

Summary of files, directories:
  Files: ['/n', '/z', '/g/s']
  Directories: ['/', '/g', '/w']
```

`-p` bayrağı son dosya veya dizini yazdırır. Burada tohum 0 da bir bozulma vardır ve `-D` bayrağı ile oluşturduğumuz dosya sistemiyle karşılaşlığımızda bozulma inode 8'in ölü blok 7'yi işaret ettiğini görüyoruz.

```
yenes@ubuntu:~/Desktop/ostep/ostep-homework/file-journaling$ python3 fsck.py -s 1
ARG seed 1
ARG seedCorrupt 0
ARG numInodes 16
ARG numData 16
ARG numRequests 15
ARG printFinal False
ARG whichCorrupt -1
ARG dontCorrupt False

Final state of file system:

inode bitmap 100010010010001
inodes [d a:0 r:4] [] [] [] [f a:-1 r:1] [] [] [d a:10 r:2] [] [] [f a:-1 r:3] [] [] [f a:-1 r:1]
data bitmap 100000000010001
data [(.,.0) (...0) (m,7) (a,8) (g,11)] [] [] [] [] [] [] [(.,.7) (...0) (m,15) (e,11)] [] [] [] [] [(.,8) (...0) (r,4) (w,11)]

Can you figure out how the file system was corrupted?
```

-s bayrağı ile rastgele dosya sistemlerinin tohumlarını kullanıyoruz ve böylelikle aynı sorunu farklı dosya sistemlerinde görebiliyoruz. Örneğin, yukarıda seed 0 idi ve inode 8 ölü blok 7'yi işaret ediyordu fakat -s 1 yaptığımızda yani tohumu 1 olarak ayarladığımızda inode 8 ölü blok 6'yi işaret ediyor.

```
yenes@ubuntu:~/Desktop/ostep/ostep-homework/file-journaling$ python3 fsck.py -s2
ARG seed 2
ARG seedCorrupt 0
ARG numInodes 16
ARG numData 16
ARG numRequests 15
ARG printFinal False
ARG whichCorrupt -1
ARG dontCorrupt False

Final state of file system:

inode bitmap 100010010010101
inodes [d a:0 r:3] [] [] [] [] [d a:4 r:3] [] [] [f a:-1 r:1] [f a:-1 r:2] [] [d a:11 r:3] [] [d a:7 r:2]
data bitmap 1000100000010001
data [(.,.0) (...0) (c,13)] [] [] [] [(.,.7) (...13) (u,15) (q,11)] [] [] [] [] [(.,.13) (...0) (o,7)] [] [] [] [(.,15) (...7) (q,11)
) (e,10)]]

Can you figure out how the file system was corrupted?
```

-s 2 ile tohumu 2 olarak ayarladığımızda inode 15 ölü blok 7'yi işaret ediyor.

```
yenes@ubuntu:~/Desktop/ostep/ostep-homework/file-journaling$ python3 fsck.py -s3
ARG seed 3
ARG seedCorrupt 0
ARG numInodes 16
ARG numData 16
ARG numRequests 15
ARG printFinal False
ARG whichCorrupt -1
ARG dontCorrupt False

Final state of file system:

inode bitmap 1010000000000001
inodes [d a:0 r:3] [] [d a:13 r:3] [d a:6 r:2] [] [] [] [] [] [] [] [f a:15 r:2]
data bitmap 10000000001000101
data [(.,.0) (...0) (f,15) (x,15) (r,2)] [] [] [] [] [] [(..,3) (...2)] [] [] [(.,2) (...0) (s,3)] [] [w]

Can you figure out how the file system was corrupted?
```

-s 3 ile tohumu 3 olarak ayarladığımızda ise inode 4 ölü blok 6'yi işaret ediyor. Görüğünüz gibi farklı dosya sistemlerindeki ortak sorun inode'larımızın ölü bloğu işaret etmesiydi bunun sebebi ise seed'in altında yer alan seedCorrupttur. SeedCorrupt Bozulmanın tohumudur sonraki sorularımızda bunu kullanacağız ve bozulmaların nasıl değiştiğini göreceksiniz.

- Şimdi, bir tutarsızlığı tanıtalım. Başlamanın fsck.py -S 1 dosyasını çalıştırın. Hangi tutarsızlığı ortaya çıktığını görebiliyor musunuz? Gerçek bir dosya sistemi onarım aracında bunu nasıl düzeltirdiniz? Haklı olup olmadığını kontrol etmek için -c kullanın.

```
yenes@ubuntu:~/Desktop/0step/0step-homework/file-journaling$ python fsck.py -S1
ARG seed 0
ARG seedCorrupt 1
ARG numInodes 16
ARG numData 16
ARG numRequests 15
ARG printFinal False
ARG whichCorrupt -1
ARG dontCorrupt False

Final state of file system:
inode bitmap 1000100010000001
inodes [d a:0 r:4] [] [] [] [d a:12 r:2] [] [] [d a:6 r:2] [] [] [] [f a:-1 r:2] [] [f a:-1 r:1]
data bitmap 10000010000001000
data [(..,0) (...0) (g,8) (w,4) (m,13) (z,13)] [] [] [] [] [(..,8) (...0) (s,15)] [] [] [] [] [(..,4) (...0)] [] [] []

Can you figure out how the file system was corrupted?
```

-S1 bayrağını çalıştırduğumızda tohumu 0 olan dosya sisteminin bozulma tohumu 1'ini görüyoruz. Öncelikle -D bayrağı ile Bozulmamış haline bakıp karşılaştırıralım.

```
yenes@ubuntu:~/Desktop/0step/0step-homework/file-journaling$ python fsck.py -S1 -D
ARG seed 0
ARG seedCorrupt 1
ARG numInodes 16
ARG numData 16
ARG numRequests 15
ARG printFinal False
ARG whichCorrupt -1
ARG dontCorrupt True

Final state of file system:
inode bitmap 1000100010000001
inodes [d a:0 r:4] [] [] [] [d a:12 r:2] [] [] [d a:6 r:2] [] [] [] [f a:-1 r:2] [] [f a:-1 r:1]
data bitmap 10000010000001000
data [(..,0) (...0) (g,8) (w,4) (m,13) (z,13)] [] [] [] [] [(..,8) (...0) (s,15)] [] [] [] [] [(..,4) (...0)] [] [] []

Can you figure out which files and directories exist?
```

Karşılaştırdığımızda bozulmamızın inode kısmında tahsis edilmiş 13. inode'un, inode bitmapinde işaretlenmemiş olarak görüyoruz. Inode'u Bitmap üzerinde işaretlersek bozulmamız çözülür. -c bayrağı ile bulduğu-muz bozulmanın doğru olup olmadığını kontrol edelim.

```
yenes@ubuntu:~/Desktop/0step/0step-homework/file-journaling$ python fsck.py -S1 -c
ARG seed 0
ARG seedCorrupt 1
ARG numInodes 16
ARG numData 16
ARG numRequests 15
ARG printFinal False
ARG whichCorrupt -1
ARG dontCorrupt False

Initial state of file system:
inode bitmap 1000100010000001
inodes [d a:0 r:4] [] [] [] [d a:12 r:2] [] [] [d a:6 r:2] [] [] [] [f a:-1 r:2] [] [f a:-1 r:1]
data bitmap 10000010000001000
data [(..,0) (...0) (g,8) (w,4) (m,13) (z,13)] [] [] [] [] [(..,8) (...0) (s,15)] [] [] [] [] [(..,4) (...0)] [] [] []

CORRUPTION::INODE BITMAP corrupt bit 13

Final state of file system:
inode bitmap 1000100010000001
inodes [d a:0 r:4] [] [] [] [d a:12 r:2] [] [] [d a:6 r:2] [] [] [] [f a:-1 r:2] [] [f a:-1 r:1]
data bitmap 10000010000001000
data [(..,0) (...0) (g,8) (w,4) (m,13) (z,13)] [] [] [] [] [(..,8) (...0) (s,15)] [] [] [] [] [(..,4) (...0)] [] [] []
```

-c bayrağı ile çalıştığımızda bozulmanın inode bitmap'inde 13. bitte olduğunu söylüyor yani cevabımız doğru. Seed değiştirerek farklı dosya sistemlerinde aynı hatanın nasıl olduğuna bakalım.

```
yenes@ubuntu:~/Desktop/ostep/ostep-homework/file-journaling$ python fsck.py -s1 -s1 -c
ARG seed 1
ARG seedCorrupt 1
ARG numInodes 16
ARG numData 16
ARG numRequests 15
ARG printFinal False
ARG whichCorrupt -1
ARG dontCorrupt False

Initial state of file system:
inode bitmap 10001000100010001
inodes [d a:0 r:4] [] [] [] [f a:-1 r:1] [] [] [d a:10 r:2] [d a:15 r:2] [] [] [f a:-1 r:3] [] [] [f a:-1 r:1]
data bitmap 1000000000100001
data [(.,0) (.,0) (m,7) (a,8) (g,11)] [] [] [] [] [] [] [(.,7) (.,0) (m,15) (e,11)] [] [] [] [(.,8) (.,0) (r,4) (w,11)]

CORRUPTION::INODE BITMAP corrupt bit 13

Final state of file system:
inode bitmap 10001000100010001
inodes [d a:0 r:4] [] [] [] [f a:-1 r:1] [] [] [d a:10 r:2] [d a:15 r:2] [] [] [f a:-1 r:3] [] [] [f a:-1 r:1]
data bitmap 1000000000100001
data [(.,0) (.,0) (m,7) (a,8) (g,11)] [] [] [] [] [] [] [(.,7) (.,0) (m,15) (e,11)] [] [] [] [(.,8) (.,0) (r,4) (w,11)]
```

Tohumu $-s1$ bayrağı ile değiştirdiğimizde tohumu 1 olan dosya sistemimiz.

```
yenes@ubuntu:~/Desktop/ostep/ostep-homework/file-journaling$ python fsck.py -s1 -s2 -c
ARG seed 2
ARG seedCorrupt 1
ARG numInodes 16
ARG numData 16
ARG numRequests 15
ARG printFinal False
ARG whichCorrupt -1
ARG dontCorrupt False

Initial state of file system:
inode bitmap 10001000100010001
inodes [d a:0 r:3] [] [] [] [d a:4 r:3] [] [] [f a:-1 r:1] [f a:-1 r:2] [] [d a:11 r:3] [] [d a:15 r:2]
data bitmap 1000000000100001
data [(.,0) (.,0) (c,13)] [] [] [(.,7) (.,13) (u,15) (q,11)] [] [] [] [(.,13) (.,0) (o,7)] [] [] [(.,15) (.,7) (q,11)
] (e,10)]

CORRUPTION::INODE BITMAP corrupt bit 13

Final state of file system:
inode bitmap 10000000100010001
inodes [d a:0 r:3] [] [] [] [d a:4 r:3] [] [] [f a:-1 r:1] [f a:-1 r:2] [] [d a:11 r:3] [] [d a:15 r:2]
data bitmap 1000000000100001
data [(.,0) (.,0) (c,13)] [] [] [(.,7) (.,13) (u,15) (q,11)] [] [] [] [(.,13) (.,0) (o,7)] [] [] [(.,15) (.,7) (q,11
) (e,10)]
```

Tohumu $-s2$ bayrağı ile değiştirdiğimizde tohumu 2 olan dosya sistemimiz.

```
yenes@ubuntu:~/Desktop/ostep/ostep-homework/file-journaling$ python fsck.py -s1 -s3 -c
ARG seed 3
ARG seedCorrupt 1
ARG numInodes 16
ARG numData 16
ARG numRequests 15
ARG printFinal False
ARG whichCorrupt -1
ARG dontCorrupt False

Initial state of file system:
inode bitmap 1011000000000001
inodes [d a:0 r:3] [] [d a:13 r:3] [d a:9 r:2] [] [] [] [] [] [f a:15 r:2]
data bitmap 10000000001000101
data [(.,0) (.,0) (r,15) (x,15) (r,2)] [] [] [] [] [] [(.,3) (.,2)] [] [] [(.,2) (.,0) (s,3)] [] [w]

CORRUPTION::INODE BITMAP corrupt bit 13

Final state of file system:
inode bitmap 101100000000000101
inodes [d a:0 r:3] [] [d a:13 r:3] [d a:9 r:2] [] [] [] [] [] [f a:15 r:2]
data bitmap 10000000001000101
data [(.,0) (.,0) (r,15) (x,15) (r,2)] [] [] [] [] [] [(.,3) (.,2)] [] [] [(.,2) (.,0) (s,3)] [] [w]
```

Tohumu $-s3$ bayrağı ile değiştirdiğimizde tohumu 3 olan dosya sistemimiz.

Gördüğümüz üzere farklı dosya sistemlerinde aynı bozulmayı yani inode bitmapinde 13. Bitin bozulduğunu görüyoruz.

3. Tohumu `-S 3` veya `-S 19` olarak değiştirin; hangi tutarsızlığı görürünüz? Cevabınızı kontrol etmek için `-c` kullanın. Bu iki durumda farklı olan nedir?

```
yenes@ubuntu:~/Desktop/0step/0step-homework/file-journaling$ python fsck.py -S3
ARG seed 0
ARG seedCorrupt 3
ARG numInodes 16
ARG numData 16
ARG numRequests 15
ARG printFinal False
ARG whichCorrupt :1
ARG dontCorrupt False

Final state of file system:

inode bitmap 1000100010000101
inodes [d a:0 r:4] [] [] [d a:12 r:2] [] [] [d a:6 r:2] [] [] [f a:-1 r:2] [] [f a:-1 r:2]
data bitmap 100001000001000
data [(.,0) (.,0) (g,8) (w,4) (m,13) (z,13)] [] [] [] [] [(.,8) (.,0) (s,15)] [] [] [] [] [(.,4) (.,0)] [] [] []

Can you figure out how the file system was corrupted?
```

`-S3` bayrağını çalıştırduğımızda tohumu 0 olan dosya sisteminin bozulma tohumu 3'ünü görüyoruz. `-D` bayrağı kullanarak bozulmaya uğramamış haliyle karşılaştırma yapıp bozulmayı bulalım.

```
yenes@ubuntu:~/Desktop/0step/0step-homework/file-journaling$ python fsck.py -S3 -D
ARG seed 0
ARG seedCorrupt 3
ARG numInodes 16
ARG numData 16
ARG numRequests 15
ARG printFinal False
ARG whichCorrupt :1
ARG dontCorrupt True

Final state of file system:

inode bitmap 1000100010000101
inodes [d a:0 r:4] [] [] [d a:12 r:2] [] [] [d a:6 r:2] [] [] [f a:-1 r:2] [] [f a:-1 r:1]
data bitmap 100001000001000
data [(.,0) (.,0) (g,8) (w,4) (m,13) (z,13)] [] [] [] [] [(.,8) (.,0) (s,15)] [] [] [] [] [(.,4) (.,0)] [] [] []

Can you figure out which files and directories exist?
```

Gördüğümüz üzere 15. inode'umuzun dosya referans sayısı bir artmış. Bu yüzden bir bozulma meydana gelmiş. Cevabımızı `-c` bayrağı ile doğru olup olmadığını kontrol edelim.

```
yenes@ubuntu:~/Desktop/0step/0step-homework/file-journaling$ python fsck.py -S3 -c
ARG seed 0
ARG seedCorrupt 3
ARG numInodes 16
ARG numData 16
ARG numRequests 15
ARG printFinal False
ARG whichCorrupt :1
ARG dontCorrupt False

Initial state of file system:

inode bitmap 1000100010000101
inodes [d a:0 r:4] [] [] [d a:12 r:2] [] [] [d a:6 r:2] [] [] [f a:-1 r:2] [] [f a:-1 r:1]
data bitmap 100001000001000
data [(.,0) (.,0) (g,8) (w,4) (m,13) (z,13)] [] [] [] [] [(.,8) (.,0) (s,15)] [] [] [] [] [(.,4) (.,0)] [] [] []

CORRUPTION::INODE 15 refcnt increased

Final state of file system:

inode bitmap 1000100010000101
inodes [d a:0 r:4] [] [] [d a:12 r:2] [] [] [d a:6 r:2] [] [] [f a:-1 r:2] [] [f a:-1 r:2]
data bitmap 100001000001000
data [(.,0) (.,0) (g,8) (w,4) (m,13) (z,13)] [] [] [] [] [(.,8) (.,0) (s,15)] [] [] [] [] [(.,4) (.,0)] [] [] []
```

`-c` bayrağı ile çalıştırduğımızda inode 15'in referans sayısının arttığını söylüyor yani cevabımız doğru. Tohumları değiştirerek aynı bozulmayı farklı dosya sistemlerinde görelim.

```
Yenes@ubuntu:~/Desktop/0step/0step-homework/file-journaling$ python3 fsck.py -s3 -s1 -c
ARG seed 1
ARG seedCorrupt 3
ARG numInodes 16
ARG numData 16
ARG numRequests 15
ARG printFinal False
ARG whichCorrupt -1
ARG dontCorrupt False

Initial state of file system:
inode bitmap 100010010010001
inodes [d a:0 r:4] [] [] [] [f a:-1 r:1] [] [] [d a:10 r:2] [d a:15 r:2] [] [] [f a:-1 r:3] [] [] [f a:-1 r:1]
data bitmap 100000000010001
data [(.,0) (...0) (m,7) (a,8) (g,11)] [] [] [] [] [] [] [(.,7) (...0) (m,15) (e,11)] [] [] [] [] [(.,8) (...0) (r,4) (w,11)]

CORRUPTION::INODE 11 refcnt increased

Final state of file system:
inode bitmap 100010010010001
inodes [d a:0 r:4] [] [] [] [f a:-1 r:1] [] [] [d a:10 r:2] [d a:15 r:2] [] [] [f a:-1 r:4] [] [] [f a:-1 r:1]
data bitmap 100000000010001
data [(.,0) (...0) (m,7) (a,8) (g,11)] [] [] [] [] [] [] [(.,7) (...0) (m,15) (e,11)] [] [] [] [] [(.,8) (...0) (r,4) (w,11)]
```

Dosya sistemi tohumunu –s1 bayrağı ile değiştirdiğimizde inode 11'in referansının bir arttığını görüyoruz.

```
Yenes@ubuntu:~/Desktop/0step/0step-homework/file-journaling$ python3 fsck.py -s3 -s2 -c
ARG seed 2
ARG seedCorrupt 3
ARG numInodes 16
ARG numData 16
ARG numRequests 15
ARG printFinal False
ARG whichCorrupt -1
ARG dontCorrupt False

Initial state of file system:
inode bitmap 1000000100100101
inodes [d a:0 r:3] [] [] [] [] [d a:4 r:3] [] [] [f a:-1 r:1] [f a:-1 r:2] [] [d a:11 r:3] [] [d a:15 r:2]
data bitmap 1000100000010001
data [(.,0) (...0) (c,13)] [] [] [(.,7) (...13) (u,15) (q,11)] [] [] [] [] [(.,13) (...0) (o,7)] [] [] [(.,15) (...7) (q,11)
) (e,10)]]

CORRUPTION::INODE 11 refcnt increased

Final state of file system:
inode bitmap 1000000100100101
inodes [d a:0 r:3] [] [] [] [] [d a:4 r:3] [] [] [f a:-1 r:1] [f a:-1 r:3] [] [d a:11 r:3] [] [d a:15 r:2]
data bitmap 1000100000010001
data [(.,0) (...0) (c,13)] [] [] [(.,7) (...13) (u,15) (q,11)] [] [] [] [] [(.,13) (...0) (o,7)] [] [] [(.,15) (...7) (q,11
) (e,10)]]

CORRUPTION::INODE 11 refcnt increased
```

Dosya sistemi tohumunu –s2 bayrağı ile değiştirdiğimizde de inode 11'in referansının bir arttığını fakat dosya sistemlerinin farklı olduğunu açıkça görüyoruz.

```
Yenes@ubuntu:~/Desktop/0step/0step-homework/file-journaling$ python3 fsck.py -s3 -s3 -c
ARG seed 3
ARG seedCorrupt 3
ARG numInodes 16
ARG numData 16
ARG numRequests 15
ARG printFinal False
ARG whichCorrupt -1
ARG dontCorrupt False

Initial state of file system:
inode bitmap 10110000000000001
inodes [d a:0 r:3] [d a:13 r:3] [d a:9 r:2] [] [] [] [] [] [] [f a:15 r:2]
data bitmap 10000000001000101
data [(.,0) (...0) (f,15) (x,15) (r,2)] [] [] [] [] [] [(.,3) (...2)] [] [] [(.,2) (...0) (s,3)] [] [w]

CORRUPTION::INODE 15 refcnt increased

Final state of file system:
inode bitmap 10110000000000001
inodes [d a:0 r:3] [d a:13 r:3] [d a:9 r:2] [] [] [] [] [] [] [f a:15 r:3]
data bitmap 10000000001000101
data [(.,0) (...0) (f,15) (x,15) (r,2)] [] [] [] [] [] [(.,3) (...2)] [] [] [(.,2) (...0) (s,3)] [] [w]
```

Son olarak dosya sistemi tohumunu –s3 bayrağı ile değiştirdiğimizde inode 15'in referansının bir arttığını görüyoruz. –S3 bayrağı ile oluşturulan bozulmalarının ortak özelliği inode'lardan herhangi birinin referans numarasının bir artmasıdır. –s ile –S bayrağının farklı olduğunu unutmamak gereklidir.

```
yenes@ubuntu:~/Desktop/ostep/ostep-homework/file-journaling$ python3 fsck.py -S19
ARG seed 0
ARG seedCorrupt 19
ARG numInodes 16
ARG numData 16
ARG numRequests 15
ARG printFinal False
ARG whichCorrupt -1
ARG dontCorrupt False

Final state of file system:
inode bitmap 10001000100000101
inodes [d a:0 r:4] [] [] [] [d a:12 r:2] [] [] [d a:6 r:1] [] [] [f a:-1 r:2] [] [f a:-1 r:1]
data bitmap 1000001000001000
data [(..,0) (...0) (g,8) (w,4) (m,13) (z,13)] [] [] [] [] [(..,8) (...0) (s,15)] [] [] [] [] [(..,4) (...0)] [] [] []

Can you figure out how the file system was corrupted?
```

-S19 bayrağını çalıştırduğumızda tohumu 0 olan dosya sisteminin bozulma tohumu 19'u görüyoruz. -D bayrağı kullanarak bozulmaya uğramamış haliyle karşılaştırma yapıp bozulmayı bulalım.

```
yenes@ubuntu:~/Desktop/ostep/ostep-homework/file-journaling$ python3 fsck.py -S19 -D
ARG seed 0
ARG seedCorrupt 19
ARG numInodes 16
ARG numData 16
ARG numRequests 15
ARG printFinal False
ARG whichCorrupt -1
ARG dontCorrupt True

Final state of file system:
inode bitmap 10001000100000101
inodes [d a:0 r:4] [] [] [] [d a:12 r:2] [] [] [d a:6 r:2] [] [] [f a:-1 r:2] [] [f a:-1 r:1]
data bitmap 1000001000001000
data [(..,0) (...0) (g,8) (w,4) (m,13) (z,13)] [] [] [] [] [(..,8) (...0) (s,15)] [] [] [] [] [(..,4) (...0)] [] [] []

Can you figure out which files and directories exist?
```

Burada da önceki bozulmamızın tam tersini görüyoruz. Bozulmamızın sebebi inode 8'in referans sayısının bir azalmasıdır. -c bayrağı ile cevabımızın doğruluğunu kontrol edelim.

```
yenes@ubuntu:~/Desktop/ostep/ostep-homework/file-journaling$ python3 fsck.py -S19 -c
ARG seed 0
ARG seedCorrupt 19
ARG numInodes 16
ARG numData 16
ARG numRequests 15
ARG printFinal False
ARG whichCorrupt -1
ARG dontCorrupt False

Initial state of file system:
inode bitmap 10001000100000101
inodes [d a:0 r:4] [] [] [] [d a:12 r:2] [] [] [d a:6 r:2] [] [] [f a:-1 r:2] [] [f a:-1 r:1]
data bitmap 1000001000001000
data [(..,0) (...0) (g,8) (w,4) (m,13) (z,13)] [] [] [] [] [(..,8) (...0) (s,15)] [] [] [] [] [(..,4) (...0)] [] [] []

CORRUPTION::INODE 8 refcnt decreased

Final state of file system:
inode bitmap 10001000100000101
inodes [d a:0 r:4] [] [] [] [d a:12 r:2] [] [] [d a:6 r:1] [] [] [f a:-1 r:2] [] [f a:-1 r:1]
data bitmap 1000001000001000
data [(..,0) (...0) (g,8) (w,4) (m,13) (z,13)] [] [] [] [] [(..,8) (...0) (s,15)] [] [] [] [] [(..,4) (...0)] [] [] []
```

-c bayrağı ile çalıştırduğumızda inode 8'in referans sayısının bir azaldığını söylüyor ve böyleselikle cevabımızın doğru olduğunu görüyoruz. Dosya sistemi tohumunu değiştirerek diğer dosya sistemlerinde bozulmalarına bakalım.

```
yenes@ubuntu:~/Desktop/ostep/ostep-homework/file-journaling$ python3 fsck.py -S19 -s1 -c
ARG seed 1
ARG seedCorrupt 19
ARG numInodes 16
ARG numData 16
ARG numRequests 15
ARG printFinal False
ARG whichCorrupt -1
ARG dontCorrupt False

Initial state of file system:

inode bitmap 1000100110010001
inodes [d a:0 r:4] [] [] [] [f a:-1 r:1] [] [] [d a:10 r:2] [d a:15 r:2] [] [] [f a:-1 r:3] [] [] [f a:-1 r:1]
data bitmap 1000000000100001
data [(..,0) (...,-8) (m,7) (a,8) (g,11)] [] [] [] [] [] [] [(..,7) (...,-8) (m,15) (e,11)] [] [] [] [] [(..,8) (...,-8) (r,4) (w,11)]

CORRUPTION::INODE 8 refcnt decreased

Final state of file system:

inode bitmap 1000100110010001
inodes [d a:0 r:4] [] [] [] [f a:-1 r:1] [] [] [d a:10 r:2] [d a:15 r:1] [] [] [f a:-1 r:3] [] [] [f a:-1 r:1]
data bitmap 1000000000100001
data [(..,0) (...,-8) (m,7) (a,8) (g,11)] [] [] [] [] [] [] [(..,7) (...,-8) (m,15) (e,11)] [] [] [] [] [(..,8) (...,-8) (r,4) (w,11)]
```

Dosya sistemi tohumunu –s1 bayrağı ile değiştirdiğimizde inode 8'in referansının bir azalduğunu görüyoruz.

```
yenes@ubuntu:~/Desktop/ostep/ostep-homework/file-journaling$ python3 fsck.py -S19 -s2 -c
ARG seed 2
ARG seedCorrupt 19
ARG numInodes 16
ARG numData 16
ARG numRequests 15
ARG printFinal False
ARG whichCorrupt -1
ARG dontCorrupt False

Initial state of file system:

inode bitmap 1000000100110101
inodes [d a:0 r:3] [] [] [] [] [d a:4 r:3] [] [] [f a:-1 r:1] [f a:-1 r:2] [] [d a:11 r:3] [] [d a:15 r:2]
data bitmap 1000100000010001
data [(..,0) (...,-8) (c,13)] [] [] [] [(..,-7) (...,-13) (u,15) (q,11)] [] [] [] [(..,13) (...,-8) (o,7)] [] [] [] [(..,15) (...,-7) (q,11)
) (e,10)]]

CORRUPTION::INODE 15 refcnt decreased

Final state of file system:

inode bitmap 1000000100110101
inodes [d a:0 r:3] [] [] [] [] [d a:4 r:3] [] [] [f a:-1 r:1] [f a:-1 r:2] [] [d a:11 r:3] [] [d a:15 r:1]
data bitmap 1000100000010001
data [(..,0) (...,-8) (c,13)] [] [] [] [(..,-7) (...,-13) (u,15) (q,11)] [] [] [] [(..,13) (...,-8) (o,7)] [] [] [] [(..,15) (...,-7) (q,11)
) (e,10)]]
```

Dosya sistemi tohumunu –s2 bayrağı ile değiştirdiğimizde inode 15'in referansının bir azalliğini görüyoruz.

```
yenes@ubuntu:~/Desktop/ostep/ostep-homework/file-journaling$ python3 fsck.py -S19 -s3 -c
ARG seed 3
ARG seedCorrupt 19
ARG numInodes 16
ARG numData 16
ARG numRequests 15
ARG printFinal False
ARG whichCorrupt -1
ARG dontCorrupt False

Initial state of file system:

inode bitmap 101100000000000001
inodes [d a:0 r:3] [] [d a:13 r:3] [d a:9 r:2] [] [] [] [] [] [] [f a:15 r:2]
data bitmap 10000000001000101
data [(..,0) (...,-8) (f,15) (x,15) (r,2)] [] [] [] [] [] [(..,3) (...,-2)] [] [] [(..,-2) (...,-8) (s,3)] [] [w]

CORRUPTION::INODE 3 refcnt decreased

Final state of file system:

inode bitmap 101100000000000001
inodes [d a:0 r:3] [] [d a:13 r:3] [d a:9 r:1] [] [] [] [] [] [f a:15 r:2]
data bitmap 10000000001000101
data [(..,0) (...,-8) (f,15) (x,15) (r,2)] [] [] [] [] [] [(..,3) (...,-2)] [] [] [(..,-2) (...,-8) (s,3)] [] [w]
```

Son olarak dosya sistemi tohumunu –s3 bayrağı ile değiştirdiğimizde inode 3'ün referans sayısının bir azalliğini görüyoruz. –S19 bayrağı ile oluşturulan bozulmalarının ortak özelliği inode'lardan herhangi birinin referans numarasının bir azalmasıdır.

Sonuç olarak **-S3** bayrağı referans numarasının bir artmasını **-S19** bayrağı ise referans numarasının bir azaldığı bozulmaları gösterir.

4. Tohumu **-S 5** olarak değiştirin; hangi tutarsızlığı görüyorsunuz? Bu sorunu otomatik bir şekilde düzeltmek ne kadar zor olurdu? Cevabınızı kontrol etmek için **-c** kullanın. Ardından, **-S 38** ile benzer bir bozulma ekleinyin; bunu tespit etmek daha mı zor/mümkün? Son olarak, **-S 642** kullanın; bu bozulma tespit edilebilir mi? Eğer öyleyse, dosya sisteminin nasıl düzeltirsiniz?

```
yenes@ubuntu:~/Desktop/0step/0step-homework/file-journaling$ python3 fsck.py -S5
ARG seed 0
ARG seedCorrupt 5
ARG numInodes 16
ARG numData 16
ARG numRequests 15
ARG printFinal False
ARG whichCorrupt -1
ARG dontCorrupt False

Final state of file system:

inode bitmap 1000100010000101
inodes [d a:0 r:4] [] [] [d a:12 r:2] [] [] [d a:6 r:2] [] [] [f a:-1 r:2] [] [f a:-1 r:1]
data bitmap 10000010000001000
data [(..,0) (...0) (g,8) (w,4) (m,13) (z,13)] [] [] [] [] [(..,8) (...0) (y,15)] [] [] [] [] [(.,4) (...0)] [] [] []

Can you figure out how the file system was corrupted?
```

Tohumu **-S5** bayrağı olarak değiştirdiğimizde tohumu 0 olan dosya sisteminin bozulma tohumu 5'ini görmekteyiz. **-D** bayrağını kullanarak bozulmaya uğramayan haliyle karşıştıralım.

```
yenes@ubuntu:~/Desktop/0step/0step-homework/file-journaling$ python3 fsck.py -S5 -D
ARG seed 0
ARG seedCorrupt 5
ARG numInodes 16
ARG numData 16
ARG numRequests 15
ARG printFinal False
ARG whichCorrupt -1
ARG dontCorrupt True

Final state of file system:

inode bitmap 1000100010000101
inodes [d a:0 r:4] [] [] [d a:12 r:2] [] [] [d a:6 r:2] [] [] [f a:-1 r:2] [] [f a:-1 r:1]
data bitmap 10000010000001000
data [(..,0) (...0) (g,8) (w,4) (m,13) (z,13)] [] [] [] [] [(..,8) (...0) (s,15)] [] [] [] [] [(.,4) (...0)] [] [] []

Can you figure out which files and directories exist?
```

Karşılaştırdığımızda 8. inode'un işaret ettiği data bölgesindeki 6. Adres bloğundaki **(s,15)** verisi **(y,15)** olarak değiştiği için bozulma meydana gelmektedir. Bozulmalar data bölgesinde olduğu için düzeltmek zor olmaz. Cevabımızı **-c** bayrağını çalıştırarak doğru olup olmadığını kontrol edelim.

```
yenes@ubuntu:~/Desktop/0step/0step-homework/file-journaling$ python3 fsck.py -S5 -c
ARG seed 0
ARG seedCorrupt 5
ARG numInodes 16
ARG numData 16
ARG numRequests 15
ARG printFinal False
ARG whichCorrupt -1
ARG dontCorrupt False

Initial state of file system:

inode bitmap 1000100010000101
inodes [d a:0 r:4] [] [] [d a:12 r:2] [] [] [d a:6 r:2] [] [] [f a:-1 r:2] [] [f a:-1 r:1]
data bitmap 10000010000001000
data [(..,0) (...0) (g,8) (w,4) (m,13) (z,13)] [] [] [] [] [(..,8) (...0) (s,15)] [] [] [] [] [(.,4) (...0)] [] [] []

CORRUPTION::INODE 8 with directory [('..', 8), ('..', 0), ('s', 15)]:
  entry ('s', 15) altered to refer to different name (y)

Final state of file system:

inode bitmap 1000100010000101
inodes [d a:0 r:4] [] [] [d a:12 r:2] [] [] [d a:6 r:2] [] [] [f a:-1 r:2] [] [f a:-1 r:1]
data bitmap 10000010000001000
data [(..,0) (...0) (g,8) (w,4) (m,13) (z,13)] [] [] [] [] [(..,8) (...0) (y,15)] [] [] [] [] [(.,4) (...0)] [] [] []
```

-c bayrağı ile çalıştığımızda bize bozulmanın sebebinin inode 8'in dizininde giriş ('s', 15) farklı bir isme (y) referans için değiştirildiğini söylüyor yani cevabımız doğrudur.

-s ile dosya sistemi tohumlarını değiştirdip farklı dosya sistemlerinde bozulmamıza tekrar bakalım.

```
venes@ubuntu:~/Desktop/ostep/ostep-homework/file-journaling$ python3 fsck.py -s5 -s1 -c
ARG seed 1
ARG seedCorrupt 5
ARG numInodes 16
ARG numData 16
ARG numRequests 15
ARG printFinal False
ARG whichCorrupt -1
ARG dontCorrupt False

Initial state of file system:

inode bitmap 10001000110010001
inodes [d a:0 r:4] [] [] [] [f a:-1 r:1] [] [] [d a:10 r:2] [d a:15 r:2] [] [] [f a:-1 r:3] [] [] [f a:-1 r:1]
data bitmap 10000000000100001
data [(..,0) (.,0) (.,8) (.,7) (.,8) (g,11)] [] [] [] [] [] [] [(.,7) (.,0) (.,15) (e,11)] [] [] [] [(.,8) (.,8) (.,8) (r,4) (w,11)]

CORRUPTION::INODE 8 with directory [('.', 8), ('.', 8), ('r', 4), ('w', 11)]:
  entry ('w', 11) altered to refer to different name (y)

Final state of file system:

inode bitmap 10001000110010001
inodes [d a:0 r:4] [] [] [f a:-1 r:1] [] [] [d a:10 r:2] [d a:15 r:2] [] [] [f a:-1 r:3] [] [] [f a:-1 r:1]
data bitmap 10000000000100001
data [(..,0) (.,0) (.,8) (g,11)] [] [] [] [] [] [] [(.,7) (.,0) (.,15) (e,11)] [] [] [] [(.,8) (.,8) (.,8) (r,4) (y,11)]
```

Dosya sistemi tohumunu -s1 bayrağı ile değiştirdiğimizde 8. İnode'un işaret ettiği data bölgesindeki 15. adres bloğundaki (w,11) verisi (y,11) olarak değiştiğini görüyoruz.

```
venes@ubuntu:~/Desktop/ostep/ostep-homework/file-journaling$ python3 fsck.py -s5 -s2 -c
ARG seed 2
ARG seedCorrupt 5
ARG numInodes 16
ARG numData 16
ARG numRequests 15
ARG printFinal False
ARG whichCorrupt -1
ARG dontCorrupt False

Initial state of file system:

inode bitmap 10000000100110101
inodes [d a:0 r:3] [] [] [] [d a:4 r:3] [] [] [f a:-1 r:1] [f a:-1 r:2] [] [d a:11 r:3] [] [d a:15 r:2]
data bitmap 10001000000100001
data [(..,0) (.,0) (c,13)] [] [] [(.,7) (.,13) (u,15) (q,11)] [] [] [] [(.,13) (.,0) (o,7)] [] [] [(.,15) (.,7) (q,11)
) (e,10)]

CORRUPTION::INODE 13 with directory [('.', 13), ('.', 8), ('o', 7)]:
  entry ('o', 7) altered to refer to different name (y)

Final state of file system:

inode bitmap 10000000100110101
inodes [d a:0 r:3] [] [] [] [d a:4 r:3] [] [] [f a:-1 r:1] [f a:-1 r:2] [] [d a:11 r:3] [] [d a:15 r:2]
data bitmap 10001000000100001
data [(..,0) (.,0) (c,13)] [] [] [(.,7) (.,13) (u,15) (q,11)] [] [] [] [(.,13) (.,0) (y,7)] [] [] [(.,15) (.,7) (q,11
) (e,10)]
```

Dosya sistemi tohumunu -s2 bayrağı ile değiştirdiğimizde 13. İnode'un işaret ettiği data bölgesindeki (o,7) verisi (y,7) olarak değiştiğini görüyoruz.

```
yenes@ubuntu:~/Desktop/0step/0step-homework/file-journaling$ python3 fsck.py -S3 -c
ARG seed 0
ARG seedCorrupt 5
ARG numInodes 16
ARG numData 16
ARG numRequests 15
ARG printFinal False
ARG whichCorrupt -1
ARG dontCorrupt False

Initial state of file system:

inode bitmap 1011000000000001
inodes [d a:0 r:3] [] [d a:13 r:3] [d a:9 r:2] [] [] [] [] [] [] [] [f a:15 r:2]
data bitmap 1000000000100010
data [(..,0) (..,0) (f,15) (x,15) (r,2)] [] [] [] [] [] [] [(..,3) (..,2)] [] [] [] [(..,2) (..,0) (s,3)] [] [w]

CORRUPTION: INODE 3 with directory ('..', 3), ('..', 2):
entry ('..', 2) altered to refer to different name (y)

Final state of file system:

inode bitmap 1011000000000001
inodes [d a:0 r:3] [] [d a:13 r:3] [d a:9 r:2] [] [] [] [] [] [] [f a:15 r:2]
data bitmap 1000000000100010
data [(..,0) (..,0) (f,15) (x,15) (r,2)] [] [] [] [] [] [(..,3) (y,2)] [] [] [] [(..,2) (..,0) (s,3)] [] [w]
```

Son olarak Dosya sistemi tohumunu –s3 bayrağı ile değiştirdiğimizde 3. İnode'un işaret ettiği data bölgesindeki 13. adres bloğundaki (..,2) verisi (y,2) olarak değiştiğini görüyoruz.

```
yenes@ubuntu:~/Desktop/0step/0step-homework/file-journaling$ python3 fsck.py -S38
ARG seed 0
ARG seedCorrupt 38
ARG numInodes 16
ARG numData 16
ARG numRequests 15
ARG printFinal False
ARG whichCorrupt -1
ARG dontCorrupt False

Final state of file system:

inode bitmap 1000100010000101
inodes [d a:0 r:4] [] [] [d a:12 r:2] [] [] [d a:6 r:2] [] [] [f a:-1 r:2] [] [f a:-1 r:1]
data bitmap 1000001000001000
data [(..,0) (..,0) (g,8) (w,4) (m,13) (z,13)] [] [] [] [] [(..,8) (..,0) (s,15)] [] [] [] [] [(..,4) (b,0)] [] [] []

Can you figure out how the file system was corrupted?
```

-S38 bayrağı ile farklı bir bozulmayı görüyoruz. Bozulmayı tespit etmek için –D bayrağı ile bozulmamış halini çalışıralım ve karşılaştıralım.

```
yenes@ubuntu:~/Desktop/0step/0step-homework/file-journaling$ python3 fsck.py -S38 -D
ARG seed 0
ARG seedCorrupt 38
ARG numInodes 16
ARG numData 16
ARG numRequests 15
ARG printFinal False
ARG whichCorrupt -1
ARG dontCorrupt True

Final state of file system:

inode bitmap 1000100010000101
inodes [d a:0 r:4] [] [] [d a:12 r:2] [] [] [d a:6 r:2] [] [] [f a:-1 r:2] [] [f a:-1 r:1]
data bitmap 1000001000001000
data [(..,0) (..,0) (g,8) (w,4) (m,13) (z,13)] [] [] [] [] [(..,8) (..,0) (s,15)] [] [] [] [] [(..,4) (..,0)] [] [] []

Can you figure out which files and directories exist?
```

Karşılaştırdığımızda 4. İnode'un işaret ettiği data bölgesindeki 12. Adres bloğundaki (..,0) verisi (b,0) olarak değiştiği için bozulma meydana gelmektedir. Bu bozulmamız da data bölgesinde gerçekleşmiştir ve bozulmamızı tespit etmek kolaydır.
Cevabımızı –c bayrağını çalıştırarak doğru olup olmadığını kontrol edelim.

```
venes@ubuntu:~/Desktop/ostep/ostep-homework/file-journaling$ python3 fsck.py -S38 -c
ARG seed 0
ARG seedCorrupt 38
ARG numInodes 16
ARG numData 16
ARG numRequests 15
ARG printFinal False
ARG whichCorrupt -1
ARG dontCorrupt False

Initial state of file system:

inode bitmap 1000100010000101
inodes [d a:0 r:4] [] [] [d a:12 r:2] [] [] [d a:6 r:2] [] [] [f a:-1 r:2] [] [f a:-1 r:1]
data bitmap 1000001000001000
data [(..,0) (...,(g,8) (w,4) (m,13) (z,13)) [] [] [] [] [(..,8) (...,(s,15)) [] [] [] [] [(..,4) (...,(b,0)) [] [] []]

CORRUPTION: INODE 4 with directory [('..', 4), ('..', 0)]:
entry ('..', 0) altered to refer to different name (b)

Final state of file system:

inode bitmap 1000100010000101
inodes [d a:0 r:4] [] [] [d a:12 r:2] [] [] [d a:6 r:2] [] [] [f a:-1 r:2] [] [f a:-1 r:1]
data bitmap 1000001000001000
data [(..,0) (...,(g,8) (w,4) (m,13) (z,13)) [] [] [] [] [(..,8) (...,(s,15)) [] [] [] [] [(..,4) (...,(b,0)) [] [] []
```

-c bayrağı ile çalıştığımızda bize bozulmanın sebebinin inode 4'ün dizininde giriş ('..', 0) farklı bir isme (b) referans için değiştirildiğini söylüyor yani cevabımız doğrudur. -s ile dosya sistemi tohumlarını değiştirdip farklı sistemlerinde bozulmamıza tekrar bakalım.

```
venes@ubuntu:~/Desktop/ostep/ostep-homework/file-journaling$ python3 fsck.py -S38 -s1 -c
ARG seed 1
ARG seedCorrupt 38
ARG numInodes 16
ARG numData 16
ARG numRequests 15
ARG printFinal False
ARG whichCorrupt -1
ARG dontCorrupt False

Initial state of file system:

inode bitmap 100010001100010001
inodes [d a:0 r:4] [] [] [f a:-1 r:1] [] [] [d a:10 r:2] [d a:15 r:2] [] [] [f a:-1 r:3] [] [] [f a:-1 r:1]
data bitmap 100000000000100001
data [(..,0) (...,(m,7) (a,8) (g,11)) [] [] [] [] [] [] [(..,7) (...,(s,15) (e,11)) [] [] [] [] [(..,8) (...,(r,4) (w,11))]

CORRUPTION: INODE 7 with directory [('..', 7), ('..', 0), ('m', 15), ('e', 11)]:
entry ('m', 15) altered to refer to different name (b)

Final state of file system:

inode bitmap 100010001100010001
inodes [d a:0 r:4] [] [] [f a:-1 r:1] [] [] [d a:10 r:2] [d a:15 r:2] [] [] [f a:-1 r:3] [] [] [f a:-1 r:1]
data bitmap 100000000000100001
data [(..,0) (...,(m,7) (a,8) (g,11)) [] [] [] [] [] [] [(..,7) (...,(b,15) (e,11)) [] [] [] [] [(..,8) (...,(r,4) (w,11))]
```

Dosya sistemi tohumunu -s1 bayrağı ile değiştirdiğimizde 7. inode'un işaret ettiği data bölgesindeki 10. adres bloğundaki (m,15) verisi (b,15) olarak değiştigini görüyoruz.

```
venes@ubuntu:~/Desktop/ostep/ostep-homework/file-journaling$ python3 fsck.py -S38 -s2 -c
ARG seed 2
ARG seedCorrupt 38
ARG numInodes 16
ARG numData 16
ARG numRequests 15
ARG printFinal False
ARG whichCorrupt -1
ARG dontCorrupt False

Initial state of file system:

inode bitmap 100000010010101
inodes [d a:0 r:3] [] [] [] [] [d a:4 r:3] [] [] [f a:-1 r:1] [f a:-1 r:2] [] [d a:11 r:3] [] [d a:15 r:2]
data bitmap 10001000000100001
data [(..,0) (...,(c,13)) [] [] [] [(..,7) (...,(u,15) (q,11)) [] [] [] [] [(..,13) (...,(o,7) (q,11)) [] [] [] [(..,15) (...,(q,11) (e,10))
```

```
CORRUPTION: INODE 7 with directory [('..', 7), ('..', 13), ('u', 15), ('q', 11)]:
entry ('u', 15) altered to refer to different name (b)
```

```
Final state of file system:
```

```
inode bitmap 100000010010101
inodes [d a:0 r:3] [] [] [] [] [d a:4 r:3] [] [] [f a:-1 r:1] [f a:-1 r:2] [] [d a:11 r:3] [] [d a:15 r:2]
data bitmap 10001000000100001
data [(..,0) (...,(c,13)) [] [] [] [(..,7) (...,(b,15) (q,11)) [] [] [] [] [(..,13) (...,(o,7) (q,11)) [] [] [] [(..,15) (...,(q,11) (e,10))]
```

OPERATING

SYSTEMS

[VERSION 1.01]

WWW.OSTEP.ORG

Dosya sistemi tohumunu –s2 bayrağı ile değiştirdiğimizde 7. İnode'un işaret ettiği data bölgesindeki 4. adres bloğundaki (u,15) verisi (b,15) olarak değiştigini görüyoruz.

```
yenes@ubuntu:~/Desktop/0step/0step-homework/file-journaling$ python3 fsck.py -s3 -s3 -c
ARG seed 3
ARG seedCorrupt 38
ARG numInodes 16
ARG numData 16
ARG numRequests 15
ARG printFinal False
ARG whichCorrupt -1
ARG dontCorrupt False

Initial state of file system:

inode bitmap 1011000000000000
inodes [d a:0 r:3] [] [d a:13 r:3] [d a:9 r:2] [] [] [] [] [] [] [] [f a:15 r:2]
data bitmap 10000000001000101
data [(.,0) (.,0) (f,15) (x,15) (r,2)] [] [] [] [] [] [(.,3) (.,2)] [] [] [(.,2) (.,0) (s,3)] [] [w]

CORRUPTION::INODE 2 with directory [(.,', 2), (.,', 0), ('s', 3)]:
  entry ('s', 3) altered to refer to different name (b)

Final state of file system:

inode bitmap 1011000000000001
inodes [d a:0 r:3] [] [d a:13 r:3] [d a:9 r:2] [] [] [] [] [] [] [f a:15 r:2]
data bitmap 10000000001000101
data [(.,0) (.,0) (f,15) (x,15) (r,2)] [] [] [] [] [] [(.,3) (.,2)] [] [] [(.,2) (.,0) (b,3)] [] [w]
```

Son olarak Dosya sistemi tohumunu –s3 bayrağı ile değiştirdiğimizde 2. İnode'un işaret ettiği data bölgesindeki 13. adres bloğundaki (s,3) verisi (b,3) olarak değiştigini görüyoruz.

```
yenes@ubuntu:~/Desktop/0step/0step-homework/file-journaling$ python3 fsck.py -s642
ARG seed 0
ARG seedCorrupt 642
ARG numInodes 16
ARG numData 16
ARG numRequests 15
ARG printFinal False
ARG whichCorrupt -1
ARG dontCorrupt False

Final state of file system:

inode bitmap 1000100010000101
inodes [d a:0 r:4] [] [] [d a:12 r:2] [] [] [d a:6 r:2] [] [] [f a:-1 r:2] [] [f a:-1 r:1]
data bitmap 100001000001000
data [(.,0) (.,0) (w,8) (w,4) (n,13) (z,13)] [] [] [] [] [(.,8) (.,0) (s,15)] [] [] [] [] [(.,4) (.,0)] [] [] []

Can you figure out how the file system was corrupted?
```

-S642 bayrağını çalıştırduğumda yeni bir bozulma görüyoruz. Bozulmayı tespit etmek için yeniden –D bayrağı ile dosya sisteminin bozulmaya uğramamış halini çalıştırımız gerekiyor.

```
yenes@ubuntu:~/Desktop/0step/0step-homework/file-journaling$ python3 fsck.py -s642 -D
ARG seed 0
ARG seedCorrupt 642
ARG numInodes 16
ARG numData 16
ARG numRequests 15
ARG printFinal False
ARG whichCorrupt -1
ARG dontCorrupt True

Final state of file system:

inode bitmap 1000100010000101
inodes [d a:0 r:4] [] [] [d a:12 r:2] [] [] [d a:6 r:2] [] [] [f a:-1 r:2] [] [f a:-1 r:1]
data bitmap 100001000001000
data [(.,0) (.,0) (g,8) (w,4) (n,13) (z,13)] [] [] [] [] [(.,8) (.,0) (s,15)] [] [] [] [] [(.,4) (.,0)] [] [] []

Can you figure out which files and directories exist?
```

Çalıştırıp karşılaştırdığımızda 0. inode'un işaret ettiği data bölgesindeki 0. Adres bloğundaki (g,8) verisi (w,8) olarak değiştiği için bozulma meydana gelmektedir. Bozulmamız data bölgesinde ve böylelikle kullanıcı tarafından değiştirilebilir, düzeltilebilir. Cevabımızı –c bayrağını çalıştırıp kontrol edebiliriz.

```
yenesh@ubuntu:~/Desktop/ostep/ostep-homework/file-journaling$ python3 fsck.py -S642 -c
ARG seed 0
ARG seedCorrupt 642
ARG numInodes 16
ARG numData 16
ARG numRequests 15
ARG printFinal False
ARG whichCorrupt -1
ARG dontCorrupt False

Initial state of file system:
inode bitmap 1000100010000101
inodes [d a:0 r:4] [] [] [d a:12 r:2] [] [] [d a:6 r:2] [] [] [f a:-1 r:2] [] [f a:-1 r:1]
data bitmap 1000010000010000
data [(.,0) (...,(g,8) (w,4) (m,13) (z,13)) [] [] [] [] [(.,8) (...,(s,15)) [] [] [] [(.,4) (...,(6)) [] [] []]

CORRUPTION::INODE 0 with directory [('..', 0), ('..', 0), ('g', 8), ('w', 4), ('m', 13), ('z', 13)]:
entry ('g', 8) altered to refer to different name (w)

Final state of file system:
inode bitmap 1000100010000101
inodes [d a:0 r:4] [] [] [d a:12 r:2] [] [] [d a:6 r:2] [] [] [f a:-1 r:2] [] [f a:-1 r:1]
data bitmap 1000010000010000
data [(.,0) (...,(w,8) (w,4) (m,13) (z,13)) [] [] [] [] [(.,8) (...,(s,15)) [] [] [] [(.,4) (...,(6)) [] [] []
```

-c bayrağı ile çalıştığımızda bize bozulmanın sebebinin inode 0'ın dizininde giriş ('g', 8) farklı bir isme (w) referans için değiştirildiğini söylüyor yani cevabımız doğrudur. -s ile dosya sistemi tohumlarını değiştirdip farklı dosya sistemlerinde bozulmamıza tekrar bakalım.

```
yenesh@ubuntu:~/Desktop/ostep/ostep-homework/file-journaling$ python3 fsck.py -S642 -s1 -c
ARG seed 1
ARG seedCorrupt 642
ARG numInodes 16
ARG numData 16
ARG numRequests 15
ARG printFinal False
ARG whichCorrupt -1
ARG dontCorrupt False

Initial state of file system:
inode bitmap 1000100110010001
inodes [d a:0 r:4] [] [] [f a:-1 r:1] [] [] [d a:10 r:2] [d a:15 r:2] [] [] [f a:-1 r:3] [] [] [f a:-1 r:1]
data bitmap 1000000000010001
data [(.,0) (...,(m,7) (a,8) (g,11)) [] [] [] [] [] [(.,7) (...,(s,15) (e,11)) [] [] [] [(.,8) (...,(r,4) (w,11)) []

CORRUPTION::INODE 0 with directory [('..', 0), ('..', 0), ('m', 7), ('a', 8), ('g', 11)]:
entry ('m', 7) altered to refer to different name (w)

Final state of file system:
inode bitmap 1000100110010001
inodes [d a:0 r:4] [] [] [f a:-1 r:1] [] [] [d a:10 r:2] [d a:15 r:2] [] [] [f a:-1 r:3] [] [] [f a:-1 r:1]
data bitmap 1000000000010001
data [(.,0) (...,(w,7) (a,8) (g,11)) [] [] [] [] [] [(.,7) (...,(m,15) (e,11)) [] [] [] [(.,8) (...,(r,4) (w,11)) []
```

Dosya sistemi tohumunu -s1 bayrağı ile değiştirdiğimizde 0. inode'un işaret ettiği data bölgesindeki 0. adres bloğundaki (m,7) verisi (w,7) olarak değiştigini görüyoruz.

```
yenesh@ubuntu:~/Desktop/ostep/ostep-homework/file-journaling$ python3 fsck.py -S642 -s2 -c
ARG seed 2
ARG seedCorrupt 642
ARG numInodes 16
ARG numData 16
ARG numRequests 15
ARG printFinal False
ARG whichCorrupt -1
ARG dontCorrupt False

Initial state of file system:
inode bitmap 10000000100110101
inodes [d a:0 r:3] [] [] [] [d a:4 r:3] [] [] [f a:-1 r:1] [f a:-1 r:2] [] [d a:11 r:3] [] [d a:15 r:2]
data bitmap 1000100000010001
data [(.,0) (...,(c,13)) [] [] [] [(.,7) (...,(u,15) (q,11)) [] [] [] [(.,13) (...,(o,7) (q,11)) [] [] [] [(.,15) (...,(7) (q,11)) (e,10)]]

CORRUPTION::INODE 7 with directory [('..', 7), ('..', 13), ('u', 15), ('q', 11)]:
entry ('..', 13) altered to refer to different name (w)

Final state of file system:
inode bitmap 10000000100110101
inodes [d a:0 r:3] [] [] [] [d a:4 r:3] [] [] [f a:-1 r:1] [f a:-1 r:2] [] [d a:11 r:3] [] [d a:15 r:2]
data bitmap 1000100000010001
data [(.,0) (...,(c,13)) [] [] [] [(.,7) (...,(w,13) (u,15) (q,11)) [] [] [] [(.,13) (...,(o,7) (q,11)) [] [] [] [(.,15) (...,(7) (q,11)) (e,10)]
```

Dosya sistemi tohumunu -s2 bayrağı ile değiştirdiğimizde 7. inode'un işaret ettiği data bölgesindeki 4. adres bloğundaki (..,13) verisi (w,13) olarak değiştğini görüyoruz.

```
venes@ubuntu:~/Desktop/0step/ostep-homework/file-journaling$ python3 fsck.py -S642 -s3 -c
ARG seed 3
ARG seedCorrupt 642
ARG numInodes 16
ARG numData 16
ARG numRequests 15
ARG printFinal False
ARG whichCorrupt -1
ARG dontCorrupt False

Initial state of file system:

inode bitmap 1011000000000000
inodes [d a:0 r:3] [] [d a:13 r:3] [d a:9 r:2] [] [] [] [] [] [] [] [f a:15 r:2]
data bitmap 10000000001000101
data [(.,0) (.,0) (f,15) (x,15) (r,2)] [] [] [] [] [] [] [(.,3) (.,2)] [] [] [(.,2) (.,0) (s,3)] [] [w]

CORRUPTION: INODE 0 with directory [(' ', 0), (' ', 0), ('f', 15), ('x', 15), ('r', 2)];
entry ('f', 15) altered to refer to different name ('w')

Final state of file system:

inode bitmap 1011000000000001
inodes [d a:0 r:3] [] [d a:13 r:3] [d a:9 r:2] [] [] [] [] [] [] [f a:15 r:2]
data bitmap 10000000001000101
data [(.,0) (.,0) (w,15) (x,15) (r,2)] [] [] [] [] [] [] [(.,3) (.,2)] [] [] [(.,2) (.,0) (s,3)] [] [w]
```

Son olarak Dosya sistemi tohumunu -s3 bayrağı ile değiştirdiğimizde 0. inode'un işaret ettiği data bölgesindeki 0. adres bloğundaki (f,15) verisi (w,15) olarak değiştğini görüyoruz.

4. Sorudaki sorulan tüm bozulmalar adres bloğundaki verinin değişmesiyle oluşmuştur.

5. Tohumu -S 6 veya -S 13 olarak değiştirin; hangi tutarsızlığı görürorsunuz? Cevabinizi kontrol etmek için -c kullanın. Bu iki durum arasındaki fark nedir? Onarım aracı böyle bir durumla karşılaşlığında ne yapmalıdır?

```
venes@ubuntu:~/Desktop/0step/ostep-homework/file-journaling$ python3 fsck.py -S6
ARG seed 0
ARG seedCorrupt 6
ARG numInodes 16
ARG numData 16
ARG numRequests 15
ARG printFinal False
ARG whichCorrupt -1
ARG dontCorrupt False

Final state of file system:

inode bitmap 1000100010000101
inodes [d a:0 r:4] [] [] [d a:12 r:2] [] [] [d a:6 r:2] [] [] [d a:-1 r:1] [f a:-1 r:2] [] [f a:-1 r:1]
data bitmap 1000001000001000
data [(.,0) (.,0) (g,8) (w,4) (m,13) (z,13)] [] [] [] [] [(.,8) (.,0) (s,15)] [] [] [] [] [(.,4) (.,0)] [] [] []

Can you figure out how the file system was corrupted?
```

Tohumu -S6 bayrağı ile değiştirdiğimizde tohumu 0 olan dosya sisteminde bir bozulma meydana gelmektedir. Bozulmayı daha iyi görmek için -D bayrağını çalıştırıp bozulmaya uğramamış haliyle karşılaşalım.

```
venes@ubuntu:~/Desktop/0step/ostep-homework/file-journaling$ python3 fsck.py -S6 -D
ARG seed 0
ARG seedCorrupt 6
ARG numInodes 16
ARG numData 16
ARG numRequests 15
ARG printFinal False
ARG whichCorrupt -1
ARG dontCorrupt True

Final state of file system:

inode bitmap 1000100010000101
inodes [d a:0 r:4] [] [] [d a:12 r:2] [] [] [d a:6 r:2] [] [] [f a:-1 r:2] [] [f a:-1 r:1]
data bitmap 1000001000001000
data [(.,0) (.,0) (g,8) (w,4) (m,13) (z,13)] [] [] [] [] [(.,8) (.,0) (s,15)] [] [] [] [] [(.,4) (.,0)] [] [] []

Can you figure out which files and directories exist?
```

Karşılaştığımızda inode bitmap'inde tahsis edilmemesine rağmen 12. inode doludur ve bozulma bu yüzündür. Cevabımızı –c bayrağı kullanarak doğrulunu kontrol edelim.

```
yenesi@ubuntu:~/Desktop/ostep/ostep-homework/file-journaling$ python3 fsck.py -s6 -c
ARG seed 0
ARG seedCorrupt 6
ARG numInodes 16
ARG numData 16
ARG numRequests 15
ARG printFinal False
ARG whichCorrupt -1
ARG dontCorrupt False

Initial state of file system:

inode bitmap 1000100010000101
inodes [d a:0 r:4] [] [] [] [d a:12 r:2] [] [] [d a:6 r:2] [] [] [f a:-1 r:2] [] [f a:-1 r:1]
data bitmap 10000010000001000
data [(..,0) (...0) (g,8) (w,4) (n,13) (z,13)] [] [] [] [] [(..,8) (...0) (s,15)] [] [] [] [] [(.,4) (...0)] [] [] []

CORRUPTION::INODE 12 orphan

Final state of file system:

inode bitmap 1000100010000101
inodes [d a:0 r:4] [] [] [] [d a:12 r:2] [] [] [d a:6 r:2] [] [] [d a:-1 r:1] [f a:-1 r:2] [] [f a:-1 r:1]
data bitmap 10000010000001000
data [(..,0) (...0) (g,8) (w,4) (n,13) (z,13)] [] [] [] [] [(..,8) (...0) (s,15)] [] [] [] [] [(.,4) (...0)] [] [] []
```

-c bayrağını kullandığımızda inode 12'nin yetim olduğunu yani tahsis edilmediğini söylüyor. Burdan yola çıktığımızda cevabımız doğrudur. -s bayrakları ile farklı dosya sistemlerinde aynı bozulmayı görelim.

```
yenesi@ubuntu:~/Desktop/ostep/ostep-homework/file-journaling$ python3 fsck.py -s6 -s1 -c
ARG seed 1
ARG seedCorrupt 6
ARG numInodes 16
ARG numData 16
ARG numRequests 15
ARG printFinal False
ARG whichCorrupt -1
ARG dontCorrupt False

Initial state of file system:

inode bitmap 1000100010000001
inodes [d a:0 r:4] [] [] [f a:-1 r:1] [] [] [d a:10 r:2] [d a:15 r:2] [] [] [f a:-1 r:3] [] [] [f a:-1 r:1]
data bitmap 1000000000000001
data [(..,0) (...0) (m,7) (a,8) (g,11)] [] [] [] [] [] [] [(.,7) (...0) (m,15) (e,11)] [] [] [] [] [(.,8) (...0) (r,4) (w,11)]]

CORRUPTION::INODE 13 orphan

Final state of file system:

inode bitmap 1000100010000001
inodes [d a:0 r:4] [] [] [f a:-1 r:1] [] [] [d a:10 r:2] [d a:15 r:2] [] [] [f a:-1 r:3] [] [d a:-1 r:1] [] [f a:-1 r:1]
data bitmap 1000000000000001
data [(..,0) (...0) (m,7) (a,8) (g,11)] [] [] [] [] [] [] [(.,7) (...0) (m,15) (e,11)] [] [] [] [] [(.,8) (...0) (r,4) (w,11)]
```

Dosya sistemi tohumunu -s1 bayrağı ile değiştirdiğimizde inode bitmap'inde tahsis edilmemesine rağmen 13. inode doludur.

```
yenesi@ubuntu:~/Desktop/ostep/ostep-homework/file-journaling$ python3 fsck.py -s6 -s2 -c
ARG seed 1
ARG seedCorrupt 6
ARG numInodes 16
ARG numData 16
ARG numRequests 15
ARG printFinal False
ARG whichCorrupt -1
ARG dontCorrupt False

Initial state of file system:

inode bitmap 10000000100010101
inodes [d a:0 r:3] [] [] [] [] [d a:4 r:3] [] [] [f a:-1 r:1] [f a:-1 r:2] [] [d a:11 r:3] [] [d a:15 r:2]
data bitmap 1000000000000001
data [(..,0) (...0) (c,13)] [] [] [(.,7) (...13) (u,15) (q,11)] [] [] [] [] [(.,13) (...0) (o,7)] [] [] [(.,15) (...7) (q,11) (e,10)]]

CORRUPTION::INODE 12 orphan

Final state of file system:

inode bitmap 10000000100010101
inodes [d a:0 r:3] [] [] [] [] [d a:4 r:3] [] [] [f a:-1 r:1] [f a:-1 r:2] [d a:-1 r:1] [d a:11 r:3] [] [d a:15 r:2]
data bitmap 1000000000000001
data [(..,0) (...0) (c,13)] [] [] [(.,7) (...13) (u,15) (q,11)] [] [] [] [] [(.,13) (...0) (o,7)] [] [] [(.,15) (...7) (q,11) (e,10)]
```

Dosya sistemi tohumunu –s2 bayrağı ile değiştirdiğimizde inode bitmap'inde tahsis edilmemesine rağmen 12. İnode doludur.

```
yenex@ubuntu:~/Desktop/ostep/ostep-homework/file-journaling$ python3 fsck.py -S6 -s3 -c
ARG seed 3
ARG seedCorrupt 6
ARG numInodes 16
ARG numData 16
ARG numRequests 15
ARG printFinal False
ARG whichCorrupt :1
ARG dontCorrupt False

Initial state of file system:

inode bitmap 10110000000000001
inodes [d a:0 r:3] [] [d a:13 r:3] [d a:9 r:2] [] [] [] [] [] [] [] [f a:15 r:2]
data bitmap 10000000001000101
data [(..,0) (..,0) (f,15) (x,15) (r,2)] [] [] [] [] [] [] [(..,3) (..,2)] [] [] [(..,2) (..,0) (s,3)] [] [w]

CORRUPTION::INODE 12 orphan

Final state of file system:

inode bitmap 10110000000000001
inodes [d a:0 r:3] [] [d a:13 r:3] [d a:9 r:2] [] [] [] [] [] [d a:-1 r:1] [] [] [f a:15 r:2]
data bitmap 10000000001000101
data [(..,0) (..,0) (f,15) (x,15) (r,2)] [] [] [] [] [] [] [(..,3) (..,2)] [] [] [(..,2) (..,0) (s,3)] [] [w]
```

Son olarak Dosya sistemi tohumunu –s3 bayrağı ile değiştirdiğimizde de inode bitmap'inde tahsis edilmemesine rağmen 12. İnode doludur.

Bir de tohumu –S13 olarak değiştirelim ve bozulmaya bakalım.

```
yenex@ubuntu:~/Desktop/ostep/ostep-homework/file-journaling$ python3 fsck.py -S13
ARG seed 0
ARG seedCorrupt 13
ARG numInodes 16
ARG numData 16
ARG numRequests 15
ARG printFinal False
ARG whichCorrupt :1
ARG dontCorrupt False

Final state of file system:

inode bitmap 10001000010000101
inodes [d a:0 r:4] [] [] [d a:12 r:2] [] [] [d a:6 r:2] [] [f a:-1 r:1] [] [f a:-1 r:2] [] [f a:-1 r:1]
data bitmap 10000000000000000
data [(..,0) (..,0) (g,8) (w,4) (m,13) (z,13)] [] [] [] [] [(..,8) (..,0) (s,15)] [] [] [] [] [(..,4) (..,0)] [] [] []

Can you figure out how the file system was corrupted?
```

Tohumu 0 olan dosya sistemindeki bozulmayı saptamak için –D bloğu ile dosya sisteminin bozulmamış halini karşılaştırıralım.

```
yenex@ubuntu:~/Desktop/ostep/ostep-homework/file-journaling$ python3 fsck.py -S13 -D
ARG seed 0
ARG seedCorrupt 13
ARG numInodes 6
ARG numData 16
ARG numRequests 15
ARG printFinal False
ARG whichCorrupt :1
ARG dontCorrupt True

Final state of file system:

inode bitmap 10001000010000101
inodes [d a:0 r:4] [] [] [d a:12 r:2] [] [] [d a:6 r:2] [] [] [f a:-1 r:2] [] [f a:-1 r:1]
data bitmap 10000000000000000
data [(..,0) (..,0) (g,8) (w,4) (m,13) (z,13)] [] [] [] [] [(..,8) (..,0) (s,15)] [] [] [] [] [(..,4) (..,0)] [] [] []

Can you figure out which files and directories exist?
```

Karşılaştırdığımızda inode bitmap'inde tahsis edilmemesine rağmen 10. İnode doludur ve bozulma bu yüzündedir. Önceki bozulmamızın aynısı diyebilirsiniz fakat aralarında bir fark var. Dikkatli bakarsak önceki yanı –S6 bayrağı ile çalıştırıldığımızda inode bitmap'de gözükmeyen ama içi dolu olan inode [d a:-1 r:1] şeklindeydi. Burada baştaki d dosyanın türünün dizin olduğunu belirtir. Şuanki yanı –S13 bayrağı ile çalıştırıldığımızda inode bitmap'de gözükmeyen ama içi dolu olan inode [f a:-1 r:1] şeklindeydi. Burada ise baştaki f dosyanın türünün normal dosya olduğunu gösterir. Yani iki bozulma arasındaki fark dosya türlerinin farklı olmasıdır. Cevabımızın doğru olduğunu kontrol etmemiz için –c bayrağı ile çalışıralım.

```
venes@ubuntu:~/Desktop/0step/ostep-homework/file-journaling$ python3 fsck.py -S13 -c
ARG seed 0
ARG seedCorrupt 13
ARG numInodes 16
ARG numData 16
ARG numRequests 15
ARG printFinal False
ARG whichCorrupt -1
ARG dontCorrupt False

Initial state of file system:

inode bitmap 10001000100000101
inodes [d a:0 r:4] [] [] [] [d a:12 r:2] [] [] [] [d a:6 r:2] [] [] [] [f a:-1 r:2] [] [f a:-1 r:1]
data bitmap 10000010000001000
data [(..,0) (..,0) (g,8) (w,4) (m,13) (z,13)] [] [] [] [] [(..,8) (..,0) (s,15)] [] [] [] [] [(..,4) (..,0)] [] [] []

CORRUPTION::INODE 10 orphan

Final state of file system:

inode bitmap 10001000100000101
inodes [d a:0 r:4] [] [] [] [d a:12 r:2] [] [] [] [d a:6 r:2] [] [f a:-1 r:1] [] [f a:-1 r:2] [] [f a:-1 r:1]
data bitmap 10000010000001000
data [(..,0) (..,0) (g,8) (w,4) (n,13) (z,13)] [] [] [] [] [(..,8) (..,0) (s,15)] [] [] [] [] [(..,4) (..,0)] [] [] []


```

-c bayrağını kullandığımızda inode 10'nın yetim olduğunu yani tahsis edilmediğini söylüyor. Burdan yola çıktığımızda cevabımız doğrudur. →s bayrakları ile farklı dosya sistemlerinde aynı bozulmayı görelim.

```
yenes@ubuntu:~/Desktop/0step/ostep-homework/file-journaling$ python3 fsck.py -S13 -s1 -c
ARG seed 1
ARG seedCorrupt 13
ARG numInodes 16
ARG numData 16
ARG numRequests 15
ARG printFinal False
ARG whichCorrupt -1
ARG dontCorrupt False

Initial state of file system:

inode bitmap 10001000110010001
inodes [d a:0 r:4] [] [] [] [f a:-1 r:1] [] [] [d a:10 r:2] [d a:15 r:2] [] [] [f a:-1 r:3] [] [] [f a:-1 r:1]
data bitmap 100000000000010001
data [(..,0) (..,7) (a,8) (g,11)] [] [] [] [] [] [(..,7) (..,0) (m,15) (e,11)] [] [] [] [] [(..,8) (..,0) (r,4) (w,11)]

CORRUPTION::INODE 10 orphan

Final state of file system:

inode bitmap 10001000110010001
inodes [d a:0 r:4] [] [] [] [f a:-1 r:1] [] [] [d a:10 r:2] [d a:15 r:2] [] [f a:-1 r:1] [f a:-1 r:3] [] [] [f a:-1 r:1]
data bitmap 100000000000010001
data [(..,0) (..,0) (m,7) (a,8) (g,11)] [] [] [] [] [] [(..,7) (..,0) (m,15) (e,11)] [] [] [] [] [(..,8) (..,0) (r,4) (w,11)]
```

Dosya sistemi tohumunu →s1 bayrağı ile değiştirdiğimizde inode bitmap'inde tahsis edilmemesine rağmen 10. İnode doludur.

```
yenes@ubuntu:~/Desktop/0step/ostep-homework/file-journaling$ python3 fsck.py -S13 -s2 -c
ARG seed 0
ARG seedCorrupt 13
ARG numInodes 16
ARG numData 16
ARG numRequests 15
ARG printFinal False
ARG whichCorrupt -1
ARG dontCorrupt False

Initial state of file system:

inode bitmap 1000000100110101
inodes [d a:0 r:3] [] [] [] [] [d a:4 r:3] [] [] [f a:-1 r:1] [f a:-1 r:2] [] [d a:11 r:3] [] [d a:15 r:2]
data bitmap 100000000000010001
data [(..,0) (..,0) (c,13)] [] [] [(..,7) (..,13) (u,15) (q,11)] [] [] [] [] [(..,13) (..,0) (o,7)] [] [] [(..,15) (..,7) (q,11) (e,10)]
```

CORRUPTION::INODE 8 orphan

Final state of file system:

```
inode bitmap 1000000100110101
inodes [d a:0 r:3] [] [] [] [] [d a:4 r:3] [f a:-1 r:1] [f a:-1 r:2] [] [d a:11 r:3] [] [d a:15 r:2]
data bitmap 100000000000010001
data [(..,0) (..,0) (c,13)] [] [] [(..,7) (..,13) (u,15) (q,11)] [] [] [] [] [(..,13) (..,0) (o,7)] [] [] [(..,15) (..,7) (q,11) (e,10)]
```

Dosya sistemi tohumunu →s2 bayrağı ile değiştirdiğimizde inode bitmap'inde tahsis edilmemesine rağmen 8. İnode doludur.

```
yenes@ubuntu:~/Desktop/ostep/ostep-homework/file-journaling$ python3 fsck.py -S13 -s3 -c
ARG seed 3
ARG seedCorrupt 13
ARG numInodes 16
ARG numData 16
ARG numRequests 15
ARG printFinal False
ARG whichCorrupt -1
ARG dontCorrupt False

Initial state of file system:
inode bitmap 1011000000000000
inodes [d a:0 r:3] [] [d a:13 r:2] [d a:9 r:2] [] [] [] [] [] [] [f a:15 r:2]
data bitmap 1000000001000101
data [(.,0) (.,0) (f,15) (x,15) (r,2)] [] [] [] [] [] [(.,3) (.,2)] [] [] [(.,2) (.,0) (s,3)] [] [w]

CORRUPTION: INODE 11 orphan

Final state of file system:
inode bitmap 1011000000000001
inodes [d a:0 r:3] [] [d a:13 r:2] [d a:9 r:2] [] [] [] [] [f a:-1 r:1] [] [] [f a:15 r:2]
data bitmap 1000000001000101
data [(.,0) (.,0) (f,15) (x,15) (r,2)] [] [] [] [] [] [(.,3) (.,2)] [] [] [(.,2) (.,0) (s,3)] [] [w]
```

Son olarak Dosya sistemi tohumunu –s3 bayrağı ile değiştirdiğimizde de inode bitmap'inde tahsis edilmemesine rağmen 11. İnode doludur. Onarım aracı böyle bir durumla karşılaşlığında tahsis edilmeyen bölgeyi siler ve böylece bozulma ortadan kalkar.

6. Tohumu –S 9 olarak değiştirin; hangi tutarsızlığı görünsünüz?
 Cevabınızı kontrol etmek için –c kullanın. Bu durumda bir kontrol ve onarım aracı hangi bilgi parçasına güvenmelidir?

```
yenes@ubuntu:~/Desktop/ostep/ostep-homework/file-journaling$ python3 fsck.py -S9
ARG seed 0
ARG seedCorrupt 9
ARG numInodes 16
ARG numData 16
ARG numRequests 15
ARG printFinal False
ARG whichCorrupt -1
ARG dontCorrupt False

Final state of file system:
inode bitmap 1000100010000101
inodes [d a:0 r:4] [] [] [d a:12 r:2] [] [] [d a:6 r:2] [] [] [d a:-1 r:2] [] [f a:-1 r:1]
data bitmap 1000000001000100
data [(.,0) (.,0) (g,8) (w,4) (n,13) (z,13)] [] [] [] [] [(.,8) (.,0) (s,15)] [] [] [] [] [(.,4) (.,0)] [] [] []

Can you figure out how the file system was corrupted?
```

Tohumu –S9 bayrağı ile değiştirdiğimizde tohumu 0 olan dosya sistemindeki bozulmayı tespit etmek için –D bayrağını kullanarak bozulmayı üagramamış dosya sistemiyle karşılaştırıralım.

```
yenes@ubuntu:~/Desktop/ostep/ostep-homework/file-journaling$ python3 fsck.py -S9 -D
ARG seed 0
ARG seedCorrupt 9
ARG numInodes 16
ARG numData 16
ARG numRequests 15
ARG printFinal False
ARG whichCorrupt -1
ARG dontCorrupt True

Final state of file system:
inode bitmap 1000100010000101
inodes [d a:0 r:4] [] [] [d a:12 r:2] [] [] [d a:6 r:2] [] [] [f a:-1 r:2] [] [f a:-1 r:1]
data bitmap 1000000001000100
data [(.,0) (.,0) (g,8) (w,4) (n,13) (z,13)] [] [] [] [] [(.,8) (.,0) (s,15)] [] [] [] [] [(.,4) (.,0)] [] [] []

Can you figure out which files and directories exist?
```

Karşılaştırdığımızda 13. İnode dosya türü normal dosya iken bozulmaya uğradığında tür dizin olarak değişmiştir. Burada dikkatli olursak a:-1 yani dosyamız boştur.
 Cevabın doğruluunu kontrol etmek için –c bayrağını çalışıralım.

```
yenes@ubuntu:~/Desktop/ostep/ostep-homework/file-journaling$ python3 fsck.py -s9 -c
ARG seed 0
ARG seedCorrupt 9
ARG numInodes 16
ARG numData 10
ARG numRequests 15
ARG printFinal False
ARG whichCorrupt -1
ARG dontCorrupt False

Initial state of file system:
inode bitmap 1000100010000101
inodes [d a:0 r:4] [] [] [] [d a:12 r:2] [] [] [d a:6 r:2] [] [] [f a:-1 r:2] [] [f a:-1 r:1]
data bitmap 1000001000001000
data [(..,0) (...0) (g,8) (w,4) (n,13) (z,13)] [] [] [] [] [(.,8) (...0) (s,15)] [] [] [] [] [(.,4) (...0)] [] [] []

CORRUPTION::INODE 13 was type file, now dir

Final state of file system:
inode bitmap 1000100010000101
inodes [d a:0 r:4] [] [] [d a:12 r:2] [] [] [d a:6 r:2] [] [] [d a:-1 r:2] [] [f a:-1 r:1]
data bitmap 1000001000001000
data [(..,0) (...0) (g,8) (w,4) (n,13) (z,13)] [] [] [] [] [(.,8) (...0) (s,15)] [] [] [] [] [(.,4) (...0)] [] [] []
```

-c bayrağını çalıştırduğumızda inode 13'ün tipinin normal dosya türü olduğunu fakat şuan dizin türünde olduğunu söylüyor ve böylelikle cevabımızın doğru olduğunu görüyoruz.

-s bayrağı ile farklı dosya sistemlerinde bozulmaya bakalım.

```
yenes@ubuntu:~/Desktop/ostep/ostep-homework/file-journaling$ python3 fsck.py -s9 -s1 -c
ARG seed 1
ARG seedCorrupt 9
ARG numInodes 16
ARG numData 10
ARG numRequests 15
ARG printFinal False
ARG whichCorrupt -1
ARG dontCorrupt False

Initial state of file system:
inode bitmap 100010001100010001
inodes [d a:0 r:4] [] [] [f a:-1 r:1] [] [] [d a:10 r:2] [d a:15 r:2] [] [] [f a:-1 r:3] [] [] [f a:-1 r:1]
data bitmap 100000000000100001
data [(..,0) (...0) (m,7) (s,8) (g,11)] [] [] [] [] [] [(.,7) (...0) (m,15) (e,11)] [] [] [] [] [(.,8) (...0) (r,4) (w,11)] []

CORRUPTION::INODE 11 was type file, now dir

Final state of file system:
inode bitmap 100010001100010001
inodes [d a:0 r:4] [] [] [f a:-1 r:1] [] [] [d a:10 r:2] [d a:15 r:2] [] [] [d a:-1 r:3] [] [] [f a:-1 r:1]
data bitmap 100000000000100001
data [(..,0) (...0) (m,7) (s,8) (g,11)] [] [] [] [] [] [(.,7) (...0) (m,15) (e,11)] [] [] [] [] [(.,8) (...0) (r,4) (w,11)]
```

Dosya sistemi tohumunu -s1 bayrağı ile çalıştırduğumızda 11. İnode dosya türü normal dosya iken bozulmaya uğradığında tür dizin olarak değişmiştir.

```
yenes@ubuntu:~/Desktop/ostep/ostep-homework/file-journaling$ python3 fsck.py -s9 -s2 -c
ARG seed 2
ARG seedCorrupt 9
ARG numInodes 16
ARG numData 16
ARG numRequests 15
ARG printFinal False
ARG whichCorrupt -1
ARG dontCorrupt False

Initial state of file system:
inode bitmap 10000000100110101
inodes [d a:0 r:3] [] [] [] [] [d a:4 r:3] [] [] [f a:-1 r:1] [f a:-1 r:2] [] [d a:11 r:3] [] [d a:15 r:2]
data bitmap 100010000000100001
data [(..,0) (...0) (c,13)] [] [] [(.,7) (...13) (u,15) (q,11)] [] [] [] [(.,13) (...0) (o,7)] [] [] [(.,15) (...7) (q,11) (e,10)]
```

CORRUPTION::INODE 10 was type file, now dir

Final state of file system:

```
(inode bitmap 10000000100110101
inodes [d a:0 r:3] [] [] [] [] [d a:4 r:3] [] [] [d a:-1 r:1] [f a:-1 r:2] [] [d a:11 r:3] [] [d a:15 r:2]
data bitmap 100010000000100001
data [(..,0) (...0) (c,13)] [] [] [(.,7) (...13) (u,15) (q,11)] [] [] [] [(.,13) (...0) (o,7)] [] [] [(.,15) (...7) (q,11) (e,10)])
```

Dosya sistemi tohumunu -s1 bayrağı ile çalıştırduğumızda 10. İnode dosya türü normal dosya iken bozulmaya uğradığında tür dizin olarak değişmiştir.

```
yenes@ubuntu:~/Desktop/ostep/ostep-homework/file-journaling$ python3 fsck.py -S3 -c
ARG seed 3
ARG seedCorrupt 9
ARG numInodes 16
ARG numData 16
ARG numRequests 15
ARG printFinal False
ARG whichCorrupt -1
ARG dontCorrupt False

Initial state of file system:

inode bitmap 10110000000000001
inodes [d a:0 r:3] [] [d a:13 r:3] [d a:9 r:2] [] [] [] [] [] [] [] [f a:15 r:2]
data bitmap 10000000001000101
data [(..,0) (...0) (f,15) (x,15) (r,2)] [] [] [] [] [] [] [(..,3) (...2)] [] [] [(..,2) (...0) (s,3)] [] [w]

CORRUPTION::INODE 15 was type file, now dir

Final state of file system:

inode bitmap 10110000000000001
inodes [d a:0 r:3] [] [d a:13 r:3] [d a:9 r:2] [] [] [] [] [] [] [d a:15 r:2]
data bitmap 10000000001000101
data [(..,0) (...0) (f,15) (x,15) (r,2)] [] [] [] [] [] [(..,3) (...2)] [] [] [(..,2) (...0) (s,3)] [] [w]
```

Son olarak Dosya sistemi tohumunu -s3 bayrağı ile çalıştırduğumızda 15. İnode dosya türü normal dosya iken bozulmaya uğradığında tür dizin olarak değişmiştir. Onarım aracı tahsis edilen her inode'un geçerli bir tür alanına sahip olduğundan emin olur eğer inode alanlarıyla ilgili kolayca düzeltilemeyecek sorunlar varsa, inode şüpheli olarak kabul edilir ve onarım aracı tarafından temizlenir ve inode bitmap uygun olarak güncellenir

7. Tohumu -S 15 olarak değiştirin; hangi tutarsızlığı görüyorsunuz? Cevabınızı kontrol etmek için -c kullanın. Bu durumda bir onarım aracı ne yapabilir? Onarım mümkün değilse, ne kadar veri kaybedilir?

```
yenes@ubuntu:~/Desktop/ostep/ostep-homework/file-journaling$ python3 fsck.py -S15
ARG seed 0
ARG seedCorrupt 15
ARG numInodes 16
ARG numData 16
ARG numRequests 15
ARG printFinal False
ARG whichCorrupt -1
ARG dontCorrupt False

Final state of file system:

inode bitmap 1000100010000101
inodes [f a:0 r:4] [] [] [] [d a:12 r:2] [] [] [d a:6 r:2] [] [] [f a:-1 r:2] [] [f a:-1 r:1]
data bitmap 1000001000001000
data [(..,0) (...0) (g,8) (w,4) (m,13) (z,13)] [] [] [] [] [(..,8) (...0) (s,15)] [] [] [] [] [(..,4) (...0)] [] [] []

Can you figure out how the file system was corrupted?
```

Tohumu -S15 bayrağı ile değiştirdiğimizde tohumu 0 olan dosya sisteminin bozulmasını görüyoruz. Bozulmanın nedenini -D bayrağı ile bozulma olmayan dosya sistemini çalıştırıp karşılaştırıldığımızda bulacağız.

```
yenes@ubuntu:~/Desktop/ostep/ostep-homework/file-journaling$ python3 fsck.py -S15 -D
ARG seed 0
ARG seedCorrupt 15
ARG numInodes 16
ARG numData 16
ARG numRequests 15
ARG printFinal False
ARG whichCorrupt -1
ARG dontCorrupt True

Final state of file system:

inode bitmap 1000100010000101
inodes [d a:0 r:4] [] [] [d a:12 r:2] [] [] [d a:6 r:2] [] [] [f a:-1 r:2] [] [f a:-1 r:1]
data bitmap 1000001000001000
data [(..,0) (...0) (g,8) (w,4) (m,13) (z,13)] [] [] [] [] [(..,8) (...0) (s,15)] [] [] [] [] [(..,4) (...0)] [] [] []

Can you figure out which files and directories exist?
```

Karşılaştırıldığımızda 0. İnode dosya türü normal dosya iken bozulmaya uğradığında tür dizin olarak değişmiştir fakat önceki sorudan farkı önceki dosya boş iken bu dosyamız doludur ve işaret eder. Cevabın doğrulunu kontrol etmek için -c bayrağını çalıştırıralım.

```
yenes@ubuntu:~/Desktop/ostep/ostep-homework/file-journaling$ python3 fsck.py -S15 -c
ARG seed 0
ARG seedCorrupt 15
ARG numInodes 16
ARG numData 16
ARG numRequests 15
ARG printFinal False
ARG whichCorrupt -1
ARG dontCorrupt False

Initial state of file system:

inode bitmap 1000100010000101
inodes [d:a:0 r:4] [] [] [d:a:12 r:2] [] [] [d:a:6 r:2] [] [] [f:a:-1 r:2] [] [f:a:-1 r:1]
data bitmap 1000001000001000
data [(..,0) (...0) (w,4) (n,13) (z,13)] [] [] [] [] [(..,8) (...0) (s,15)] [] [] [] [] [(..,4) (...0)] [] [] []

CORRUPTION::INODE 0 was type file, now dir

Final state of file system:

inode bitmap 1000100010000101
inodes [f:a:0 r:4] [] [] [d:a:12 r:2] [] [] [d:a:6 r:2] [] [] [f:a:-1 r:2] [] [f:a:-1 r:1]
data bitmap 1000001000001000
data [(..,0) (...0) (g,8) (w,4) (n,13) (z,13)] [] [] [] [] [(..,8) (...0) (s,15)] [] [] [] [] [(..,4) (...0)] [] [] []
```

-c bayrağını çalıştırduğumızda inode 0'ın tipinin normal dosya türü olduğunu fakat şuan dizin türünde olduğunu söylüyor ve böylelikle cevabımızın doğru olduğunu görüyoruz.

-s bayrağı ile farklı dosya sistemlerinde bozulmaya bakalım.

```
yenes@ubuntu:~/Desktop/ostep/ostep-homework/file-journaling$ python3 fsck.py -S15 -s1 -c
ARG seed 1
ARG seedCorrupt 15
ARG numInodes 16
ARG numData 16
ARG numRequests 15
ARG printFinal False
ARG whichCorrupt -1
ARG dontCorrupt False

Initial state of file system:

inode bitmap 10001000100001001
inodes [d:a:0 r:4] [] [] [f:a:-1 r:1] [] [] [d:a:10 r:2] [d:a:15 r:2] [] [] [f:a:-1 r:3] [] [] [f:a:-1 r:1]
data bitmap 10000000000100001
data [(..,0) (...0) (n,7) (a,8) (g,11)] [] [] [] [] [] [(..,7) (...0) (n,15) (e,11)] [] [] [] [] [(..,8) (...0) (r,4) (w,11)] []

CORRUPTION::INODE 0 was type file, now dir

Final state of file system:

inode bitmap 10001000100001001
inodes [f:a:0 r:4] [] [] [f:a:-1 r:1] [] [] [d:a:10 r:2] [d:a:15 r:2] [] [] [f:a:-1 r:3] [] [] [f:a:-1 r:1]
data bitmap 10000000000100001
data [(..,0) (...0) (m,7) (a,8) (g,11)] [] [] [] [] [] [(..,7) (...0) (m,15) (e,11)] [] [] [] [] [(..,8) (...0) (r,4) (w,11)]
```

Dosya sistemi tohumunu -s1 bayrağı ile çalıştırduğumızda 0. inode dosya türü normal dosya iken bozulmaya uğradığında tür dizin olarak değişmiştir.

```
yenes@ubuntu:~/Desktop/ostep/ostep-homework/file-journaling$ python3 fsck.py -S15 -s2 -c
ARG seed 2
ARG seedCorrupt 15
ARG numInodes 16
ARG numData 16
ARG numRequests 15
ARG printFinal False
ARG whichCorrupt -1
ARG dontCorrupt False

Initial state of file system:

inode bitmap 100000010010101
inodes [d:a:0 r:3] [] [] [] [] [d:a:4 r:3] [] [] [f:a:-1 r:1] [f:a:-1 r:2] [] [d:a:11 r:3] [] [d:a:15 r:2]
data bitmap 1000100000010001
data [(..,0) (...0) (c,13)] [] [] [(..,7) (...13) (u,15) (q,11)] [] [] [] [] [(..,13) (...0) (o,7)] [] [] [(..,15) (...7) (q,11) (e,10)]]

CORRUPTION::INODE 0 was type file, now dir

Final state of file system:

inode bitmap 100000010010101
inodes [f:a:0 r:3] [] [] [] [] [d:a:4 r:3] [] [] [f:a:-1 r:1] [f:a:-1 r:2] [] [d:a:11 r:3] [] [d:a:15 r:2]
data bitmap 1000100000010001
data [(..,0) (...0) (c,13)] [] [] [(..,7) (...13) (u,15) (q,11)] [] [] [] [] [(..,13) (...0) (o,7)] [] [] [(..,15) (...7) (q,11) (e,10)]
```

Dosya sistemi tohumunu -s2 bayrağı ile çalıştırduğumızda 0. inode dosya türü normal dosya iken bozulmaya uğradığında tür dizin olarak değişmiştir.

```
yenes@ubuntu:~/Desktop/ostep/ostep-homework/file-journaling$ python3 fsck.py -S15 -s3 -c
ARG seed
ARG seedCorrupt 15
ARG numInodes 16
ARG numData 16
ARG numRequests 15
ARG printFinal False
ARG whichCorrupt -1
ARG dontCorrupt False

Initial state of file system:

inode bitmap 10110000000000001
inodes [d a:8 r:3] [] [d a:13 r:3] [d a:9 r:2] [] [] [] [] [] [] [f a:15 r:2]
data bitmap 10000000001000101
data [(..,0) (...,(f,15) (x,15) (r,2)) [] [] [] [] [] [(..,3) (...2)] [] [] [(..,2) (...0) (s,3)] [] [w]

CORRUPTION::INODE 0 was type file, now dir

Final state of file system:

inode bitmap 10110000000000001
inodes [f a:0 r:3] [] [d a:13 r:3] [d a:9 r:2] [] [] [] [] [] [f a:15 r:2]
data bitmap 10000000001000101
data [(..,0) (...,(f,15) (x,15) (r,2)) [] [] [] [] [] [(..,3) (...2)] [] [] [(..,2) (...0) (s,3)] [] [w]
```

Son olarak Dosya sistemi tohumunu –s3 bayrağı ile çalıştırduğumızda 0. İnode dosya türü normal dosya iken bozulmaya uğradığında tür dizin olarak değişmiştir.

Cevapların aynıdır fakat dosya sistemleri gördüğümüz üzere farklıdır. Onarım aracı İnode Stati kontrol ederek bozulmayı onarabilir fakat dizin içeriği normal dosya sistemine dönüştüğü kadar veri kaybı yaşar.

8. Tohumu –S 10 olarak değiştirin; hangi tutarsızlığı görüyorsunuz? Cevabınızı kontrol etmek için –c kullanın. Burada dosya sistemi yapısında onarıma yardımcı olabilecek bir fazlalık var mı?

```
yenes@ubuntu:~/Desktop/ostep/ostep-homework/file-journaling$ python3 fsck.py -S10
ARG seed 0
ARG seedCorrupt 10
ARG numInodes 16
ARG numData 16
ARG numRequests 15
ARG printFinal False
ARG whichCorrupt -1
ARG dontCorrupt False

Final state of file system:

inode bitmap 1000100010000101
inodes [d a:8 r:4] [] [] [d a:12 r:2] [] [] [d a:6 r:2] [] [] [f a:-1 r:2] [] [f a:-1 r:1]
data bitmap 10000000000100000
data [(..,0) (...,(g,8) (w,4) (m,13) (z,13)) [] [] [] [] [(..,8) (...0) (s,15)] [] [] [] [] [(..,4) (...3)] [] [] []

Can you figure out how the file system was corrupted?
```

Tohumu –S15 bayrağı ile değiştirdiğimizde tohumu 0 olan dosya sisteminin bozulmasını görüyoruz. Bozulmayı anlamamız için –D bayrağı ile bozulma meydana gelmemiş haliyle karşılaştırılalım.

```
yenes@ubuntu:~/Desktop/ostep/ostep-homework/file-journaling$ python3 fsck.py -S10 -D
ARG seed 0
ARG seedCorrupt 10
ARG numInodes 16
ARG numData 16
ARG numRequests 15
ARG printFinal False
ARG whichCorrupt -1
ARG dontCorrupt True

Final state of file system:

inode bitmap 1000100010000101
inodes [d a:0 r:4] [] [] [d a:12 r:2] [] [] [d a:6 r:2] [] [] [f a:-1 r:2] [] [f a:-1 r:1]
data bitmap 10000000000100000
data [(..,0) (...,(g,8) (w,4) (m,13) (z,13)) [] [] [] [] [(..,8) (...0) (s,15)] [] [] [] [] [(..,4) (...3)] [] [] []

Can you figure out which files and directories exist?
```

Karşılaştırdığımızda 4. İnode'in işaret ettiği data bölgesindeki 12. adres bloğundaki (...) verisi (...3) olarak değiştğini ve bozulma meydana geldiğini görüyoruz. Cevabımızı –c bayrağını çalıştırarak kontrol edelim.

```
yenes@ubuntu:~/Desktop/ostep/ostep-homework/file-journaling$ python3 fsck.py -S10 -c
ARG seed 0
ARG seedCorrupt 10
ARG numInodes 16
ARG numData 16
ARG numRequests 15
ARG printFinal False
ARG whichCorrupt -1
ARG dontCorrupt False

Initial state of file system:

inode bitmap 10000100010000101
inodes [d a:0 r:4] [] [] [] [d a:12 r:2] [] [] [d a:6 r:2] [] [] [f a:-1 r:2] [] [f a:-1 r:1]
data bitmap 10000000000000000
data [(..,0) (...,(g,8) (w,4) (m,13) (z,13)) [] [] [] [] [(..,8) (...,(s,15))] [] [] [] [] [(..,4) (...,(s,15))] [] [] []]

CORRUPTION::INODE 4 with directory [(..,4), ('..', 0)]:
entry ('..', 0) altered to refer to unallocated inode (3)

Final state of file system:

inode bitmap 10000100010000101
inodes [d a:0 r:4] [] [] [] [d a:12 r:2] [] [] [d a:6 r:2] [] [] [f a:-1 r:2] [] [f a:-1 r:1]
data bitmap 10000000000000000
data [(..,0) (...,(g,8) (w,4) (m,13) (z,13)) [] [] [] [] [(..,8) (...,(s,15))] [] [] [] [] [(..,4) (...,(s,15))] [] [] []]


```

-c bayrağını çalıştırıldığımızda giriş ('..', 0) ayrılmamış inode'a (3) referans için değiştirildiğini söylüyor yani cevabımız doğrudur. -s ile dosya sistemi tohumlarını değiştirdip farklı dosya sistemlerinde bozulmamıza tekrar bakalım.

```
yenes@ubuntu:~/Desktop/ostep/ostep-homework/file-journaling$ python3 fsck.py -S10 -s1 -c
ARG seed 1
ARG seedCorrupt 10
ARG numInodes 16
ARG numData 16
ARG numRequests 15
ARG printFinal False
ARG whichCorrupt -1
ARG dontCorrupt False

Initial state of file system:

inode bitmap 100001000100001001
inodes [d a:0 r:4] [] [] [] [f a:-1 r:1] [] [] [d a:10 r:2] [d a:15 r:2] [] [] [f a:-1 r:3] [] [] [f a:-1 r:1]
data bitmap 10000000000000001
data [(..,0) (...,(m,7) (a,8) (g,11)) [] [] [] [] [] [(..,7) (...,(s,15) (e,11))] [] [] [] [] [(..,8) (...,(r,4) (w,11))]

CORRUPTION::INODE 7 with directory [(..,7), ('..', 0), ('m', 15), ('e', 11)]:
entry ('m', 15) altered to refer to unallocated inode (3)

Final state of file system:

inode bitmap 100001000100001001
inodes [d a:0 r:4] [] [] [] [f a:-1 r:1] [] [] [d a:10 r:2] [d a:15 r:2] [] [] [f a:-1 r:3] [] [] [f a:-1 r:1]
data bitmap 10000000000000001
data [(..,0) (...,(m,7) (a,8) (g,11)) [] [] [] [] [] [(..,7) (...,(s,15) (e,11))] [] [] [] [] [(..,8) (...,(r,4) (w,11))]


```

Dosya sistemi tohumunu -s1 bayrağı ile değiştirdiğimizde 7. inode'un işaret ettiği data bölgesindeki 10. adres bloğundaki (m,15) verisi (m,3) olarak değiştigini görüyoruz.

```
yenes@ubuntu:~/Desktop/ostep/ostep-homework/file-journaling$ python3 fsck.py -S10 -s2 -c
ARG seed 0
ARG seedCorrupt 10
ARG numInodes 16
ARG numData 16
ARG numRequests 15
ARG printFinal False
ARG whichCorrupt -1
ARG dontCorrupt False

Initial state of file system:

inode bitmap 10000000100100101
inodes [d a:0 r:3] [] [] [] [] [d a:4 r:3] [] [] [f a:-1 r:1] [f a:-1 r:2] [] [d a:11 r:3] [] [d a:15 r:2]
data bitmap 10000000000000001
data [(..,0) (...,(c,13)) [] [] [(..,7) (...,(u,15) (q,11)) [] [] [] [] [(..,13) (...,(o,7) (q,11))] [] [] [(..,15) (...,(q,11) (c,10))]

CORRUPTION::INODE 7 with directory [(..,7), ('..', 13), ('u', 15), ('q', 11)]:
entry ('u', 15) altered to refer to unallocated inode (3)

Final state of file system:

inode bitmap 10000000100100101
inodes [d a:0 r:3] [] [] [] [] [d a:4 r:3] [] [] [f a:-1 r:1] [f a:-1 r:2] [] [d a:11 r:3] [] [d a:15 r:2]
data bitmap 10000000000000001
data [(..,0) (...,(c,13)) [] [] [(..,7) (...,(u,3) (q,11)) [] [] [] [] [(..,13) (...,(o,7) (q,11))] [] [] [(..,15) (...,(q,11) (e,10))]


```

Dosya sistemi tohumunu -s2 bayrağı ile değiştirdiğimizde 7. İnode'un işaret ettiği data bölgesindeki 4. adres bloğundaki (u,15) verisi (u,3) olarak değiştiğini görüyoruz.

```
venes@ubuntu:~/Desktop/0step/0step-homework/file-journaling$ python3 fsck.py -S10 -s3 -c
ARG seed 3
ARG seedCorrupt 10
ARG numInodes 16
ARG numData 16
ARG numRequests 15
ARG printFinal False
ARG whichCorrupt -1
ARG dontCorrupt False

Initial state of file system:

inode bitmap 1011000000000001
inodes [d a:0 r:3] [] [d a:13 r:3] [d a:9 r:2] [] [] [] [] [] [] [] [f a:15 r:2]
data bitmap 10000000001000101
data [(.,0) (.,0) (f,15) (x,15) (r,2)] [] [] [] [] [] [] [(.,3) (.,2)] [] [] [] [(.,2) (.,0) (s,3)] [] [w]

CORRUPTION::INODE 2 WITH DIRECTORY [(.,1), 2], (.,0), ('s', 3):
  entry (.,0) altered to refer to unallocated inode (13)

Final state of file system:

inode bitmap 1011000000000001
inodes [d a:0 r:3] [] [d a:13 r:3] [d a:9 r:2] [] [] [] [] [] [] [f a:15 r:2]
data bitmap 10000000001000101
data [(.,0) (.,0) (f,15) (x,15) (r,2)] [] [] [] [] [] [] [(.,3) (.,2)] [] [] [] [(.,2) (.,0) (s,3)] [] [w]
```

Son olarak Dosya sistemi tohumunu -s3 bayrağı ile değiştirdiğimizde 2. İnode'un işaret ettiği data bölgesindeki 13. adres bloğundaki (..,0) verisi (..,13) olarak değiştiğini görüyoruz.

9. Tohumu -S 16 ve -S 20 olarak değiştirin; hangi tutarsızlığı görürsünüz? Cevabınızı kontrol etmek için -c kullanın. Onarım aracı sorunu nasıl çözmelidir?

```
venes@ubuntu:~/Desktop/0step/0step-homework/file-journaling$ python3 fsck.py -S16
ARG seed 0
ARG seedCorrupt 16
ARG numInodes 16
ARG numData 16
ARG numRequests 15
ARG printFinal False
ARG whichCorrupt -1
ARG dontCorrupt False

Final state of file system:

inode bitmap 10000100010000101
inodes [d a:0 r:4] [] [] [d a:12 r:2] [] [] [d a:6 r:2] [] [] [f a:7 r:2] [] [f a:-1 r:1]
data bitmap 1000001000001000
data [(.,0) (.,0) (g,8) (w,4) (n,13) (z,13)] [] [] [] [] [(.,8) (.,0) (s,15)] [] [] [] [(.,4) (.,0)] [] [] []

Can you figure out how the file system was corrupted?
```

-S16 bayrağı ile tohumu 0 olan dosya sistemimizin yeni bir bozulma durumunu görüyoruz. Bozulmayı daha iyi anlamamız için -D bayrağı ile bozulmaya uğramamış halini karşılaştırmamız gerekiyor.

```
venes@ubuntu:~/Desktop/0step/0step-homework/file-journaling$ python3 fsck.py -S16 -D
ARG seed 0
ARG seedCorrupt 16
ARG numInodes 16
ARG numData 16
ARG numRequests 15
ARG printFinal False
ARG whichCorrupt -1
ARG dontCorrupt True

Final state of file system:

inode bitmap 1000100010000101
inodes [d a:0 r:4] [] [] [d a:12 r:2] [] [] [d a:6 r:2] [] [] [f a:-1 r:2] [] [f a:-1 r:1]
data bitmap 1000001000001000
data [(.,0) (.,0) (g,8) (w,4) (n,13) (z,13)] [] [] [] [] [(.,8) (.,0) (s,15)] [] [] [] [(.,4) (.,0)] [] [] []

Can you figure out which files and directories exist?
```

Karşılaştırdığımızda 13. İnode'daki normal dosya türümüz normalde -1 ile boşken bozulmaya uğradığında 7. Bölgeyi işaret etmiş gibi gösteriyor yani ölü bir bloğa işaret etmiş oluyor. -c bayrağını çalıştırarak cevabımızı kontrol edelim.

```
venes@ubuntu:~/Desktop/ostep/ostep-homework/file-journaling$ python3 fsck.py -S16 -c
ARG seed 0
ARG seedCorrupt 16
ARG numInodes 16
ARG numData 16
ARG numRequests 15
ARG printFinal False
ARG whichCorrupt -1
ARG dontCorrupt False

Initial state of file system:

inode bitmap 10000100000000101
inodes [d a:0 r:4] [] [] [d a:12 r:2] [] [] [d a:6 r:2] [] [] [f a:-1 r:2] [] [f a:-1 r:1]
data bitmap 10000010000001000
data [(..,0) (..,0) (g,8) (w,4) (n,13) (z,13)] [] [] [] [] [(..,8) (..,0) (s,15)] [] [] [] [] [(..,4) (..,0)] [] [] []

CORRUPTION::INODE 13 points to dead block 7

Final state of file system:

inode bitmap 10001000000000101
inodes [d a:0 r:4] [] [] [d a:12 r:2] [] [] [d a:6 r:2] [] [] [f a:7 r:2] [] [f a:-1 r:1]
data bitmap 10000010000001000
data [(..,0) (..,0) (g,8) (w,4) (n,13) (z,13)] [] [] [] [] [(..,8) (..,0) (s,15)] [] [] [] [] [(..,4) (..,0)] [] [] []
```

-c bloğunu çalıştırduğımızda inode 13'ün ölü blok 7yi işaret ettiğini söylüyor böylelikle cevabımızın doğru olduğunu görüyoruz. Diğer dosya sistemlerinde hatamızı bilmek için -s bayraklarını kullanalım.

```
venes@ubuntu:~/Desktop/ostep/ostep-homework/file-journaling$ python3 fsck.py -S1 -s1 -c
ARG seed 1
ARG seedCorrupt 16
ARG numInodes 16
ARG numData 16
ARG numRequests 15
ARG printFinal False
ARG whichCorrupt -1
ARG dontCorrupt False

Initial state of file system:

inode bitmap 1000100010000001
inodes [d a:0 r:4] [] [] [f a:-1 r:1] [] [] [d a:10 r:2] [d a:15 r:2] [] [] [f a:-1 r:3] [] [] [f a:-1 r:1]
data bitmap 10000000000000000001
data [(..,0) (..,0) (m,7) (a,8) (g,11)] [] [] [] [] [] [(..,7) (..,0) (m,15) (e,11)] [] [] [] [] [(..,8) (..,0) (r,4) (w,11)] []

CORRUPTION::INODE 11 points to dead block 6

Final state of file system:

inode bitmap 1000100000000001
inodes [d a:0 r:4] [] [] [f a:-1 r:1] [] [] [d a:10 r:2] [d a:15 r:2] [] [] [f a:6 r:3] [] [] [f a:-1 r:1]
data bitmap 10000000000000000001
data [(..,0) (..,0) (m,7) (a,8) (g,11)] [] [] [] [] [] [(..,7) (..,0) (m,15) (e,11)] [] [] [] [] [(..,8) (..,0) (r,4) (w,11)]
```

Dosya sistemi tohumunu -s1 bayrağı ile değiştirdiğimizde inode 11'in ölü blok 6'yi işaret ettiğini görüyoruz.

```
venes@ubuntu:~/Desktop/ostep/ostep-homework/file-journaling$ python3 fsck.py -S16 -s2 -c
ARG seed 2
ARG seedCorrupt 16
ARG numInodes 16
ARG numData 16
ARG numRequests 15
ARG printFinal False
ARG whichCorrupt -1
ARG dontCorrupt False

Initial state of file system:

inode bitmap 1000000010001001
inodes [d a:0 r:4] [] [] [] [] [d a:4 r:3] [] [] [f a:-1 r:1] [f a:-1 r:2] [] [d a:11 r:3] [] [d a:15 r:2]
data bitmap 10000000000000000001
data [(..,0) (..,0) (c,13)] [] [] [] [(..,7) (..,13) (u,15) (q,11)] [] [] [] [] [(..,13) (..,0) (o,7)] [] [] [(..,15) (..,7) (q,11) (e,10)]]

CORRUPTION::INODE 10 points to dead block 7

Final state of file system:

inode bitmap 1000000010001001
inodes [d a:0 r:3] [] [] [] [] [d a:4 r:3] [] [] [f a:7 r:1] [f a:-1 r:2] [] [d a:11 r:3] [] [d a:15 r:2]
data bitmap 10000000000000000001
data [(..,0) (..,0) (c,13)] [] [] [] [(..,7) (..,13) (u,15) (q,11)] [] [] [] [] [(..,13) (..,0) (o,7)] [] [] [(..,15) (..,7) (q,11) (e,10)]
```

Dosya sistemi tohumunu –s2 bayrağı ile değiştirdiğimizde inode 10'un ölü blok 7'yi işaret ettiğini görüyoruz.

```
yenes@ubuntu:~/Desktop/0step/ostep-homework/file-journaling$ python3 fsck.py -S16 -s3 -c
ARG seed 3
ARG seedCorrupt 16
ARG numInodes 16
ARG numData 16
ARG numRequests 15
ARG printFinal False
ARG whichCorrupt -1
ARG dontCorrupt False

Initial state of file system:

inode bitmap 10110000000000001
inodes [d a:0 r:3] [] [d a:13 r:3] [d a:9 r:2] [] [] [] [] [] [] [f a:15 r:2]
data bitmap 10000000001000101
data [(.,0) (...0) (f,15) (x,15) (r,2)] [] [] [] [] [] [(.,3) (...2)] [] [] [(.,2) (...0) (s,3)] [] [w]

CORRUPTION::INODE 15 points to dead block 6

Final state of file system:

inode bitmap 10110000000000001
inodes [d a:0 r:3] [] [d a:13 r:3] [d a:9 r:2] [] [] [] [] [] [f a:6 r:2]
data bitmap 10000000001000101
data [(.,0) (...0) (f,15) (x,15) (r,2)] [] [] [] [] [] [(.,3) (...2)] [] [] [(.,2) (...0) (s,3)] [] [w]
```

Son olarak dosya sistemi tohumunu –s3 bayrağı ile değiştirdiğimizde inode 15'in ölü blok 6'yi işaret ettiğini görüyoruz. Bir de –S20 bayrağı ile çalışıtmayı deneyelim.

```
yenes@ubuntu:~/Desktop/0step/ostep-homework/file-journaling$ python3 fsck.py -S20
ARG seed 0
ARG seedCorrupt 20
ARG numInodes 16
ARG numData 16
ARG numRequests 15
ARG printFinal False
ARG whichCorrupt -1
ARG dontCorrupt False

Final state of file system:

inode bitmap 1000100010000101
inodes [d a:0 r:4] [] [] [d a:12 r:2] [] [] [d a:11 r:2] [] [] [f a:-1 r:2] [] [f a:-1 r:1]
data bitmap 1000001000001000
data [(.,0) (...0) (g,8) (w,4) (m,13) (z,13)] [] [] [] [] [(.,8) (...0) (s,15)] [] [] [] [(.,4) (...0)] [] [] []

Can you figure out how the file system was corrupted?
```

-S20 bayrağı ile tohumu 0 olan dosya sisteminde bozulmayı çalıştırıyoruz. Bozulmayı tespit etmek için –D bayrağını çalıştırarak bozulmaya uğramamış haliyle karşılaşmamız gerekiyor.

```
yenes@ubuntu:~/Desktop/0step/ostep-homework/file-journaling$ python3 fsck.py -S20 -D
ARG seed 0
ARG seedCorrupt 20
ARG numInodes 16
ARG numData 16
ARG numRequests 15
ARG printFinal False
ARG whichCorrupt -1
ARG dontCorrupt True

Final state of file system:

inode bitmap 1000100010000101
inodes [d a:0 r:4] [] [] [d a:12 r:2] [] [] [d a:6 r:2] [] [] [f a:-1 r:2] [] [f a:-1 r:1]
data bitmap 1000001000001000
data [(.,0) (...0) (g,8) (w,4) (m,13) (z,13)] [] [] [] [] [(.,8) (...0) (s,15)] [] [] [] [(.,4) (...0)] [] [] []

Can you figure out which files and directories exist?
```

Karşılaştırdığımızda türü dizin olan 8. inode'umuz normalde 6. Adresi işaret ederken bozulmaya uğradığında 11. Adresi işaret ediyor yani ölü bir bloğa işaret ediyor. Cevabımızı kontrol etmek için –c bayrağını kullanalım.

```
venes@ubuntu:~/Desktop/0step/ostep-homework/file-journaling$ python3 fsck.py -s20 -c
ARG seed 0
ARG seedCorrupt 20
ARG numInodes 16
ARG numData 16
ARG numRequests 15
ARG printFinal False
ARG whichCorrupt -1
ARG dontCorrupt False

Initial state of file system:

inode bitmap 10001000100000101
inodes [d a:0 r:4] [] [] [] [d a:12 r:2] [] [] [d a:6 r:2] [] [] [f a:-1 r:2] [] [f a:-1 r:1]
data bitmap 10000010000001000
data [(.,0) (.,0) (g,8) (w,4) (m,13) (z,13)] [] [] [] [] [(.,8) (.,0) (s,15)] [] [] [] [] [(.,4) (.,0)] [] [] []

CORRUPTION::INODE 8 points to dead block 11

Final state of file system:

inode bitmap 10001000100000101
inodes [d a:0 r:4] [] [] [] [d a:12 r:2] [] [] [d a:11 r:2] [] [] [f a:-1 r:2] [] [f a:-1 r:1]
data bitmap 10000010000001000
data [(.,0) (.,0) (g,8) (w,4) (m,13) (z,13)] [] [] [] [] [(.,8) (.,0) (s,15)] [] [] [] [] [(.,4) (.,0)] [] [] []


```

-c bloğunu çalıştırduğumızda inode 8'ün ölü blok 11'i işaret ettiğini söylüyor
böylelikle cevabımızın doğru olduğunu görüyoruz. Diğer dosya sistemlerinde
hatamiza bilmek için → bayraklarını kullanalım.

```
venes@ubuntu:~/Desktop/0step/ostep-homework/file-journaling$ python3 fsck.py -s20 -s1 -c
ARG seed 1
ARG seedCorrupt 20
ARG numInodes 16
ARG numData 16
ARG numRequests 15
ARG printFinal False
ARG whichCorrupt -1
ARG dontCorrupt False

Initial state of file system:

inode bitmap 10001000100000101
inodes [d a:0 r:4] [] [] [] [f a:-1 r:1] [] [] [d a:10 r:2] [d a:15 r:2] [] [] [f a:-1 r:3] [] [] [f a:-1 r:1]
data bitmap 1000000000010001
data [(.,0) (.,0) (m,7) (a,8) (g,11)] [] [] [] [] [] [(.,7) (.,0) (m,15) (e,11)] [] [] [] [] [(.,8) (.,0) (r,4) (w,11)]

CORRUPTION::INODE 8 points to dead block 11

Final state of file system:

inode bitmap 10001000100000101
inodes [d a:0 r:4] [] [] [] [f a:-1 r:1] [] [] [d a:10 r:2] [d a:11 r:2] [] [] [f a:-1 r:3] [] [] [f a:-1 r:1]
data bitmap 1000000000010001
data [(.,0) (.,0) (m,7) (a,8) (g,11)] [] [] [] [] [] [(.,7) (.,0) (m,15) (e,11)] [] [] [] [] [(.,8) (.,0) (r,4) (w,11)]
```

Dosya sistemi tohumunu → s1 bayrağı ile değiştirdiğimizde inode 8'in ölü blok 11'yi
şİaret ettiğini görüyoruz.

```
venes@ubuntu:~/Desktop/0step/ostep-homework/file-journaling$ python3 fsck.py -s20 -s2 -c
ARG seed 2
ARG seedCorrupt 20
ARG numInodes 16
ARG numData 16
ARG numRequests 15
ARG printFinal False
ARG whichCorrupt -1
ARG dontCorrupt False

Initial state of file system:

inode bitmap 100000010001001
inodes [d a:0 r:3] [] [] [] [] [d a:4 r:3] [] [] [f a:-1 r:1] [f a:-1 r:2] [] [d a:11 r:3] [] [d a:15 r:2]
data bitmap 1000000000010001
data [(.,0) (.,0) (c,13)] [] [] [(.,7) (.,13) (u,15) (q,11)] [] [] [] [] [(.,13) (.,0) (o,7)] [] [] [(.,15) (.,7) (q,11)
] (e,10)]
```

CORRUPTION::INODE 13 points to dead block 12

Final state of file system:

```
inode bitmap 100000010001001
inodes [d a:0 r:3] [] [] [] [] [d a:4 r:3] [] [] [f a:-1 r:1] [f a:-1 r:2] [] [d a:12 r:3] [] [d a:15 r:2]
data bitmap 1000000000010001
data [(.,0) (.,0) (c,13)] [] [] [(.,7) (.,13) (u,15) (q,11)] [] [] [] [] [(.,13) (.,0) (o,7)] [] [] [(.,15) (.,7) (q,11)
] (e,10)]
```

Dosya sistemi tohumunu → s2 bayrağı ile değiştirdiğimizde inode 13'un ölü blok 12'yi
şİaret ettiğini görüyoruz.

```
venes@ubuntu:~/Desktop/ostep/ostep-homework/file-journaling$ python3 fsck.py -S20 -s3 -c
ARG seed 3
ARG seedCorrupt 20
ARG numInodes 16
ARG numData 16
ARG numRequests 15
ARG printFinal False
ARG whichCorrupt -1
ARG dontCorrupt False

Initial state of file system:
inode bitmap 10110000000000000000
inodes [d a:0 r:3] [] [d a:13 r:3] [d a:9 r:2] [] [] [] [] [] [] [] [f a:15 r:2]
data bitmap 1000000001000101
data [(.,0) (.,0) (f,15) (x,15) (r,2)] [] [] [] [] [] [(.,3) (.,2)] [] [] [(.,2) (.,0) (s,3)] [] [w]

CORRUPTION::INODE 3 points to dead block 11

Final state of file system:
inode bitmap 10110000000000000000
inodes [d a:0 r:3] [] [d a:13 r:3] [d a:11 r:2] [] [] [] [] [] [f a:15 r:2]
data bitmap 1000000001000101
data [(.,0) (.,0) (f,15) (x,15) (r,2)] [] [] [] [] [] [(.,3) (.,2)] [] [] [(.,2) (.,0) (s,3)] [] [w]
```

Son olarak dosya sistemi tohumunu `-s3` bayrağı ile değiştirdiğimizde inode 3'ün ölü blok 11'i işaret ettiğini görüyoruz. Onarım aracı inode'u kontrol edip bozulmayı bulmalı ve düzeltmelidir.