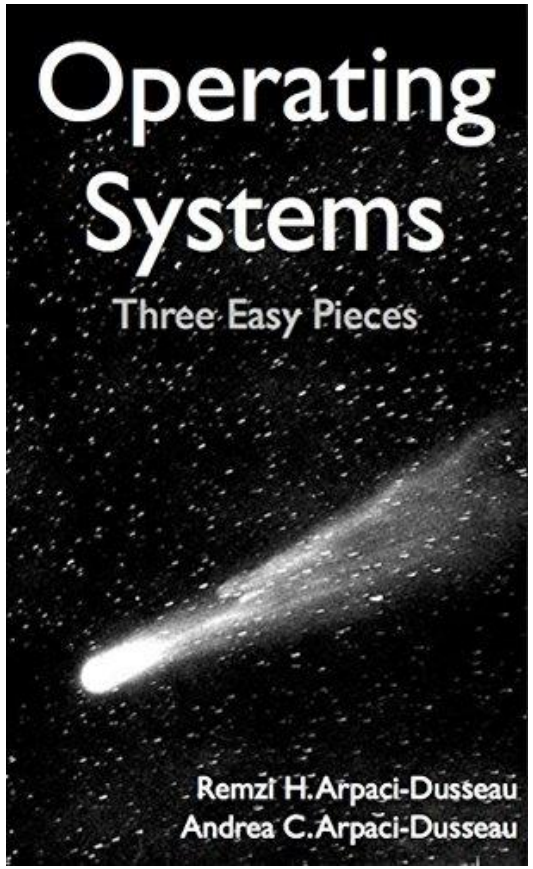


İşletim Sistemleri

11. Ders

Prof. Dr. Kemal Bıçakcı

38. RAID

Operating System: Three Easy Pieces

RAID
(Redundant Array of Inexpensive Disks)
(Ucuz Disklerin Artıklı dizisi)

- Daha hızlı, daha büyük ve daha güvenilir bir disk sistemi oluşturmak için birden fazla diski birlikte kullanmak.
- RAID, ana bilgisayar sistemine büyük bir disk gibi görünür.
- Avantajlar:
  - Performans ve Kapasite: Birden çok diskin paralel olarak kullanılması
  - Güvenilirlik: RAID ile disk kaybı veri kaybı anlamına gelmeyebilir.

RAID, bu avantajları şeffaf bir şekilde sağlar.

RAID Arayüzü

- Bir RAID G/Ç talebi aldığında, RAID hangi diske erişileceğini hesaplar.
- RAID, bunu yapmak için bir veya daha fazla fiziksel G/Ç isteği gönderebilir.
- Örnek olarak İkizlenmiş (Mirrored) bir RAID sistemi düşünelim:
  - Her bloğun iki kopyasını sakla (her biri ayrı bir diskte olacak şekilde)
  - Yazarken, verilen her bir mantıksal G/Ç için iki fiziksel G/Ç gerçekleştir.

RAID İç Yapısı

- Mikrodenetleyici: RAID'in çalışmasını yönlendirmek için «firmware» çalıştırır.
- Geçici bellek (DRAM gibi): Veri bloklarının tamponlanması (buffering)
- Uçucu olmayan bellek (nonvolatile memory): Daha güvenli yazım için.
- Parite hesaplaması için özel bir devre.

Arıza Modeli (Fault Model)

- RAID'ler, belirli türdeki disk hatalarını algılamak ve bunlardan kurtulmak için tasarlanmıştır.
- Fail-stop arıza modeli:
  - Bir disk iki durumdan birinde olabilir: Çalışıyor veya Başarısız.
  - Çalışıyor: tüm bloklar okunabilir veya yazılabilir.
  - Başarısız: disk kalıcı olarak kaybolmuştur ve RAID denetleyicisi, bir disk arızalandığında bu durumu anında gözlemleyebilir.

Bir RAID sistemi nasıl değerlendirilir?

- **Kapasite**
  - Sistem ne kadar kapasite kullanabilir?
- **Güvenilirlik (Reliability)**
  - Verilen tasarım kaç tane disk hatasını tolere edebilir?
- **Performans**

RAID Seviye 0: Şeritleme (Striping)

- RAID Seviye 0, şeritlemeli blok yapısı ile en basit biçimdir.
- Bloklar, sıralı bir şekilde disklere yayılır.
- Artıklık yoktur.
- Maksimum performans ve kapasite sunar.

Disk 0	Disk 1	Disk 2	Disk 3
0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

Stripe
(The blocks in the same row)

RAID-0: Simple Striping
(Assume here a 4-disk array)

RAID Seviye 0 (Devam)

Örnek: Daha büyük parça boyutuna (chunk size) sahip RAID-0

- Parça büyüklüğü : 2 blok (8 KB)
- Bir şerit: 4 parça (32 KB)

Disk 0	Disk 1	Disk 2	Disk 3
0	2	4	6
1	3	5	7
8	10	12	14
9	11	13	15

chunk size:
2 blocks

Striping with a Bigger Chunk Size

Parça (Chunk) Büyüklüğü

- Parça büyüklüğü çoğunlukla disk dizisinin performansını etkiler.
  - Küçük ise: Paralellik artar ama bloklara erişmek için konumlandırma süresi de artar.
  - Büyük ise: Paralellik azalır ama konumlandırma süresinin azaltılması da sağlanır.

"En iyi" büyüklüğü belirlemek zordur.

Çoğu kez büyük parçalar kullanılır (ör. 64 KB)

RAID Seviye 0 Analizi

- **Kapasite** → RAID-0 mükemmeldir.
  - Şeritlemede N diskin her biri de kullanılabilir kapasite sağlar.
- **Performans** → RAID-0 mükemmeldir.
  - Tüm disklerden paralel şekilde faydalanılır.

- **Güvenilirlik** → RAID-0 kötüdür.
  - Herhangi bir disk arızası veri kaybına yol açacaktır.

N : disk sayısı

RAID Performansını Değerlendirme

- İki performans ölçütü düşünelim:
  - Tek istek gecikmesi
  - Kararlı durum verimi
- İş yükü
  - Sıralı (Sequential): MB'larca veriye aynı anda erişim (ör. blok B ile blok B + 10MB arası)
  - Rastgele (Random): Rastgele mantıksal bir adresten 4KB'ye erişim.
- Bir diskin veri aktarım hızı:
  - Sıralı bir iş yükü altında S MB/s
  - Rastgele bir iş yükü altında R MB/sn

RAID Performansını Değerlendirme Örneği

- sıralı (S) ve rastgele (R)
  - Sıralı: kesintisiz veri (ortalama 10 MB) aktarma.
  - Rastgele: ortalama 10 KB aktarma.
  - Ortalama arama süresi: 7 ms
  - Ortalama dönme gecikmesi: 3 ms
  - Disk aktarım hızı (transfer rate): 50 MB/s
- Sonuçlar:
  - $\mathrm{S} = \frac{Veri\ miktarı}{Erişim\ süresi} = \frac{10\ MB}{210\ ms}$ = 47.62 MB /s
  - $\mathrm{R} = \frac{Veri\ miktarı}{Erişim\ süresi} = \frac{10\ KB}{10.195\ ms}$ = 0.981 MB /s

RAID-0 Performansını Değerlendirme

- Tek istek gecikmesi:
  - Tek disk ile aynı.
- Kararlı durum verimi:
  - Sıralı iş yükü : $N \cdot S$ MB/s
  - Rastgele iş yükü : $N \cdot R$ MB/s

RAID Seviye 1 : İkizleme

- RAID Seviye 1, disk arızalarını tolere eder.
- Sistemdeki her bloğun birden fazla kopyası vardır.
- Her kopya ayrı bir diske yerleştirilmelidir.

Disk 0	Disk 1	Disk 2	Disk 3
0	0	1	1
2	2	3	3
4	4	5	5
6	6	7	7

Simple RAID-1: Mirroring (Keep two physical copies)

RAID-1 Analizi

- **Capacity**: RAID-1 pahalıdır.
  - Kullanılabilen kapasite: N/2.

- **Güvenilirlik**: RAID-1 iyidir.
  - Herhangi bir diskin arızasını tolere edebilir (hangi diskin arızalandığına bağlı olarak N/2 arızaya kadar).

RAID-1'in Performansı

- Gecikme: Tamamlanacak iki fiziksel yazma (paralel yürütülebilir).
  - İki talepten kötü olan arama ve dönüş gecikmesi etkindir.

- Kararlı durum verimi (Steady-state throughput):
  - **Sıralı Yazma (Sequential Write)**: $\frac{N}{2} \cdot S$ MB/s
    - Her mantıksal yazma, iki fiziksel yazmaya yol açar.
  - **Rastgele Yazma (Random Write)**: $\frac{N}{2} \cdot R$ MB/s
    - Her mantıksal yazma, iki fiziksel yazmaya dönüşür.
  - **Sıralı Okuma (Sequential Read)**: $\frac{N}{2} \cdot S$ MB/s Niye?
    - Her disk, en yüksek bant genişliğinin yalnızca yarısını sağlayabilir.
  - **Random Read** : $N \cdot R$ MB/s Niye?
    - Okumalar tüm disklere dağıtılır.

RAID Seviye 4:
Eşlik (Parity) ile Yerden Tasarruf

- Tek bir eşlik bloğu ekle.
- Bir eşlik bloğu, o blok şeridi için fazlalık verisini depolar.

* P: Parity

Disk 0	Disk 1	Disk 2	Disk 3	Disk 4
0	1	2	3	P0
4	5	6	7	P1
8	9	10	11	P2
12	13	14	15	P3

Five-disk RAID-4 system layout

RAID Seviye 4 (Devam)

- **Eşlik (parity) hesaplama**: tüm bitleri XOR'la.

C0	C1	C2	C3	P
0	0	1	1	XOR(0,0,1,1)=0
0	1	0	0	XOR(0,1,0,0)=1

- Eşlik bitini kullanarak veri kurtarma:
- İlk satırdaki C2 bitinin kaybolduğunu düşünelim.
- O satırdaki diğer değerleri okuruz: 0, 0, 1
- Eşlik biti 0 → satırdaki 1'ler çift sayıda olmalıdır. Eksik veri ne olmalı: 1

RAID-4 Analizi

- **Kapasite**
  - Kullanılabilen kapasite: $(N - 1)$.

- **Güvenilirlik**
  - RAID-4, 1 tane disk kaybında veri kaybı yaşatmaz (daha fazla disk kaybında veri kaybı olur).

RAID-4 Analizi (Devam)

- **Performans**
  - Kararlı durum verimi
    - Sıralı okuma (Sequential read): $(N-1)\cdot S$ MB/s
    - Sıralı yazma (Sequential write): $(N-1)\cdot S$ MB/s

Disk 0	Disk 1	Disk 2	Disk 3	Disk 4
0	1	2	3	P0
4	5	6	7	P1
8	9	10	11	P2
12	13	14	15	P3

Full-stripe Writes In RAID-4

    - Rastgele Okuma (Random read): $(N-1)\cdot R$ MB/s

RAID-4'un
Rastgele Yazma Performansı

- Bir bloğun üzerine yaz + eşliği güncelle.

- Yöntem 1: Toplayarak eşlik hesaplama (Additive parity)
  - Şeritteki diğer tüm veri bloklarını oku.
  - Bu blokları yeni yazılan blokla XOR'la.
- Sorun: Performans, disk sayısı ile beraber azalır.

RAID-4'un
Rastgele Yazma Performansı (Devam)

- **Metot 2**: Çıkararak eşlik hesaplama (subtractive parity)

C0	C1	C2	C3	P
0	0	1	1	XOR(0,0,1,1)=0

- C2 değerinin güncellendiğini düşünelim: C2(eski) → C2(yeni)
- C2 (C2(eski)=1) ve Eşlik (P(eski)=0) eski verilerini oku.
- P(yeni) hesapla:
  - C2(yeni)==C2(eski) → P(yeni)==P(eski)
  - C2(yeni)!=C2(eski) ise → Eski eşlik bitini değiştir.

$$P(yeni) = (C2(eski)\ XOR\ C2(yeni)\ XOR\ P(eski))$$

Küçük yazma (Small-write) problemi

- Eşlik diski bir darboğaz olabilir.
  - Örnek: Blok 4 ve 13'ün güncellendiğini varsayalım (* ile işaretli)

Disk 0	Disk 1	Disk 2	Disk 3	Disk 4
0	1	2	3	P0
*4	5	6	7	+P1
8	9	10	11	P2
12	*13	14	15	+P3

Writes To 4, 13 And Respective Parity Blocks.

- Disk 0 ve Disk 1 paralel olarak erişilebilir.
- Öte yandan, Disk 4 bu paralelliği engelleyecektir.

Rastgele küçük yazma durumunda RAID-4 verimi çok kötüdür: $(\frac{R}{2})$ MB/s.

(eşlik diski, mantıksal G/Ç başına iki G/Ç (bir okuma, bir yazma) gerçekleştirmelidir)

RAID-4’de Gecikme

- **Tek bir okuma:**
  - Tek bir disk isteğinin gecikme süresine eşdeğerdir.
- **Tek bir yazma:**
  - İki okuma ve ardından iki yazma: Veri bloğu + Eşlik bloğu
  - Okumalar paralel olarak gerçekleşebilir ve yazmalar paralel olarak gerçekleşebilir.
  - Toplam gecikme, tek bir diskin yaklaşık iki katıdır.

RAID Seviye 5: Dönen Eşlik

- RAID-5, küçük yazma sorununa bir çözümdür.
- Eşlik blokları sürücüler arasında döndürülür.
- RAID-4'deki eşlik diski darboğazı ortadan kaldırılmış olur.

Disk 0	Disk 1	Disk 2	Disk 3	Disk 4
0	1	2	3	P0
5	6	7	P1	4
10	11	P2	8	9
15	P3	12	13	14
P4	16	17	18	19

RAID-5 With Rotated Parity

RAID-5 Analizi

- **Kapasite**
  - Kullanılabilen kapasite: $(N - 1)$.

- **Güvenilirlik**
  - RAID-5, 1 tane disk kaybında veri kaybı yaşatmaz (daha fazla disk kaybında veri kaybı olur).

RAID-5 Analizi (Devam)

- **Performans**
  - Sıralı okuma ve yazma
  - Tek okuma ve Tek yazma gecikmesi

  } RAID-4 ile aynı

  - Rastgele okuma: RAID-4'den biraz daha iyidir.
    - RAID-5 tüm disklerden faydalanır (N-1 → N).
  - Rastgele yazma: $\frac{N}{4} \cdot R$ MB/s
    - Dört değeri, eşlik tabanlı RAID kullanmanın maliyetini yansıtır (her yazmada 4 G/Ç işlemi yapılır).

RAID Karşılaştırma: Özet

N : the number of disks
D : the time that a request to a single disk takes

	RAID-0	RAID-1	RAID-4	RAID-5
Capacity	N.B	N.B/2	(N-1)B	(N-1).B
Reliability	0	1 (for sure) $\frac{N}{2}$ (if lucky)	1	1
Throughput				
Sequential Read	N・S	(N/2) ・S	(N-1) ・S	(N-1) ・S
Sequential Write	N・S	(N/2) ・S	(N-1) ・S	(N-1) ・S
Random Read	N・R	N・R	(N-1) ・R	N・R
Random Write	N・R	(N/2) ・R	$\frac{1}{2}$ R	$\frac{N}{4}$ R
Latency				
Read	D	D	D	D
Write	D	D	2D	2D

RAID Capacity, Reliability, and Performance

RAID Karşılaştırma: Özet

- Performans ve güvenilirlik önemli ise → RAID-0 (Şeritleme)
- Rastgele G/Ç performansı ve Güvenilirlik → RAID-1 (İkizleme)
- Kapasite ve Güvenilirlik → RAID-5
- Sıralı G/Ç ve Kapasiteyi En Üst Düzeye Çıkarmak için → RAID-5

39. Dosyalar ve Dizinler (Files and Directories)

Operating System: Three Easy Pieces

Kalıcı Saklama

- Bir güç kaybı olsa bile bir veriyi kaybetmeme.
  - Sabit disk sürücüsü
  - Katı hal (Solid state) depolama
- Depolamanın sanallaştırılmasında iki temel soyutlama (abstraction):
  - Dosya
  - Dizin

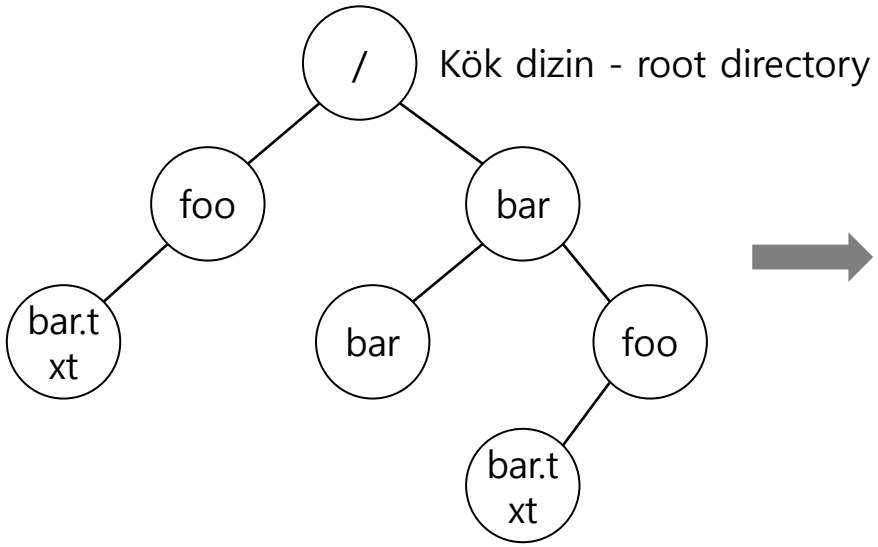
Dosya

- Doğrusal bir bayt dizisi
- Her dosyanın daha düşük seviyede inode numarası (ad - name) vardır. Kullanıcı bu adın farkında değildir.
- Dosya sistemine verileri kalıcı olarak diskte depolama sorumluluğu verilmiştir.

Dizin

- Dizin bir dosya gibidir, benzer şekilde alt seviye bir adı daha vardır.
- (Kullanıcı tarafından okunan ad, düşük seviye ad) çiftlerinin bir listesini içerir.
- Bir dizindeki her girdi, dosyalara veya diğer dizinlere atıfta bulunur.
- Örnek: Bir dizinde (“foo”, “10”) şeklinde bir girdi var ise:
  - Alt seviye adı "10" olan bir "foo" dosyası

Dizin Ağacı (Dizin Hiyerarşisi)



Örnek Dizin Ağacı

Geçerli dosyalar (Mutlak yol adları - absolute pathname):
/foo/bar.txt
/bar/foo/bar.txt

Geçerli dizinler:
/
/foo
/bar
/bar/bar
/bar/foo/

Alt-dizinler - Sub-directories

Dosya Oluşturma

- `open()` sistem çağrısı ve `O_CREAT` bayrağını kullanarak.

```
int fd = open("foo", O_CREAT | O_WRONLY | O_TRUNC);
```

  - `O_CREAT` : dosya oluşturma.
  - `O_WRONLY` : açıkken dosyaya sadece yaz.
  - `O_TRUNC` : dosya boyutunu sıfır yap (mevcut içeriği kaldır).

- `open()` sistem çağrısı bir **dosya tanımlayıcı (file descriptor)** döndürür.
  - Dosya tanımlayıcı bir tam sayıdır ve dosyalara erişmek için kullanılır.

Soru (Unix’i yazan iki kişiden Ken Thompson’a):
Unix'i yeniden tasarlasaydınız neyi farklı yapardınız?

Cevap:
Creat’i e ile yazardım.

Dosya Okuma ve Yazma

- ‘`foo`’ dosyasına yazma ve okuma

```
prompt> echo hello > foo
prompt> cat foo
hello
prompt>
```

  - `echo`: echo çıktısını foo dosyasına yönlendir
  - `cat`: bir dosyanın içeriğini ekrana dök

cat programı foo dosyasına nasıl erişir?

Bir program tarafından yapılan sistem çağrılarını görmek için strace'i kullanabiliriz.

Dosya Okuma ve Yazma (Devam)

```
prompt> strace cat foo
…
open("foo", O_RDONLY|O_LARGEFILE)  = 3
read(3, "hello\n", 4096)           = 6
write(1, "hello\n", 6)             = 6 // file descriptor 1: standard out
hello
read(3, "", 4096)                  = 0 // 0: no bytes left in the file
close(3)                           = 0
…
prompt>
```

- `open` (dosya tanımlayıcı, bayraklar)
  - Dosya tanımlayıcı döndürür (örnekte 3 değerini)
  - 0, 1, 2 dosya tanımlayıcıları standart girdi/çıktı/hata içindir.
- `read` (dosya tanımlayıcı, önbellek işaretçisi, önbellek büyüklüğü)
  - Kaç bayt okunduğu döndürülür
- `write` (dosya tanımlayıcı, önbellek işaretçisi, önbellek büyüklüğü)
  - Kaç bayt yazıldığı döndürülür

Dosya Okuma ve Yazma (Devam)

- Dosyaya yazma (Okuma'ya benzer adımları içerir)
  - Dosya önce yasmak için açılır (`open()`).
  - `write()` sistem çağrısı yapılır.
    - Büyük dosyalar için birden fazla
  - `close()`

Okuma ve Yazma, Ama Sırayla Değil

- Açık bir dosyanın geçerli bir ofset değeri vardır.
  - Bir sonraki okuma veya yazma işleminin dosya içinde nereden okumaya veya yazmaya başlayacağını belirler.
- Geçerli ofseti güncellemek
  - Örtülü: N baytlık bir okuma veya yazma ile gerçekleşir, geçerli ofsete N eklenir.
  - Açıkça: lseek() sistem çağrısı

Okuma ve Yazma, Ama Sırayla Değil (Devam)

```
off_t lseek(int fildes, off_t offset, int whence);
```

- `fildes`: Dosya tanımlayıcı
- `offset`: Dosya ofsetini dosya içinde belirli bir konuma konumlandır
- `whence`: Ofset konumlandırmanın nasıl yapıldığını belirler

man sayfasından:

```
If whence is SEEK_SET, the offset is set to offset bytes.
If whence is SEEK_CUR, the offset is set to its current
location plus offset bytes.
If whence is SEEK_END, the offset is set to the size of the
file plus offset bytes.
```

`fsync()` ile Hemen Yazmak

- Dosya sistemi, yazma işlemlerini bir süre arabelleğe alır.
  - Örnek: 5 veya 30 saniye
- Performans nedenlerinden dolayı
- Yazma(lar) fiilen daha sonraki bir zamanda yapılacaktır.
- Yazma hızlı bir şekilde tamamlanmış gibi görünür.
- Öte yandan, veriler kaybolabilir (örneğin, makine çöktüğünde).

`fsync()` ile Hemen Yazmak (Devam)

- Ancak, bazı uygulamalar nihai garantiden daha fazlasını gerektirir.
  - Örnek: Veritabanı sistemleri, diske zorunlu yazma işlemleri gerektirebilir.

- `off_t fsync(int fd)`
  - Dosya sistemi, dosya tanımlayıcı tarafından atıfta bulunulan dosya için tüm kirli (yani henüz yazılmamış) verilerin diske yazılmasını zorlar.
  - `fsync()` ancak tüm bu yazma işlemleri tamamlandıktan sonra geri döner.

`fsync()` ile Hemen Yazmak (Devam)

- `fsync()` örneği:

```
int fd = open("foo", O_CREAT | O_WRONLY | O_TRUNC);
assert (fd > -1)
int rc = write(fd, buffer, size);
assert (rc == size);
rc = fsync(fd);
assert (rc == 0);
```

- Bazı durumlarda, foo dosyasını içeren dizin için de fsync() gerekir.

Dosyaları Yeniden Adlandırmak

- `rename(char* old, char *new)`
  - Bir dosyayı farklı bir adla yeniden adlandırır.
  - Atomik bir çağrı olarak uygulanır.
  - Örnek: `foo` adını `bar` ile değiştirelim :

```
prompt> mv foo bar        // mv uses the system call rename()
```

  - Örnek: `emacs` gibi bir dosya düzenleyici kullanarak bir dosyayı atomik olarak nasıl güncelleyebilirim?

```
int fint fd = open("foo.txt.tmp", O_WRONLY|O_CREAT|O_TRUNC);
write(fd, buffer, size); // write out new version of file
fsync(fd);
close(fd);
rename("foo.txt.tmp", "foo.txt");
```

Dosyalar Hakkında Bilgi Edinmek

- `stat()`,`fstat()`: Dosya meta verilerini gösteren sistem çağrıları
  - **Meta veriler (Metadata):** dosya hakkındaki veriler.
  - Örnek: Boyut, düşük-seviye ad, izinler,…
- `stat` yapısı:

```
struct stat {
    dev_t st_dev;         /* ID of device containing file */
    ino_t st_ino;         /* inode number */
    mode_t st_mode;       /* protection */
    nlink_t st_nlink;     /* number of hard links */
    uid_t st_uid;         /* user ID of owner */
    gid_t st_gid;         /* group ID of owner */
    dev_t st_rdev;        /* device ID (if special file) */
    off_t st_size;        /* total size, in bytes */
    blksize_t st_blksize; /* blocksize for filesystem I/O */
    blkcnt_t st_blocks;   /* number of blocks allocated */
    time_t st_atime;      /* time of last access */
    time_t st_mtime;      /* time of last modification */
    time_t st_ctime;      /* time of last status change */
};
```

Dosyalar Hakkında Bilgi Edinmek (Devam)

- Stat bilgilerini görmek için `stat` komut satırı aracını kullanabiliriz:

```
prompt> echo hello > file
prompt> stat file

File: ‘file’
Size: 6 Blocks: 8 IO Block: 4096 regular file
Device: 811h/2065d Inode: 67158084 Links: 1
Access: (0640/-rw-r-----) Uid: (30686/ root) Gid: (30686/ remzi)
Access: 2011-05-03 15:50:20.157594748 -0500
Modify: 2011-05-03 15:50:20.157594748 -0500
Change: 2011-05-03 15:50:20.157594748 -0500
```

- Dosya sistemi bu tür bilgileri bir `inode` yapısında tutar..

Dosyaları Kaldırma

- `rm`, bir dosyayı kaldırmak için kullanılan Linux komutudur.
  - `rm, unlink()` (sistem çağrısı) çağrısını gerçekleştirir.

```
prompt> strace rm foo
…
unlink("foo")                = 0      // return 0 upon success
…
prompt>
```

Niye `unlink()`?
Niye "`remove` veya `delete`" değil?

Dizin Oluşturmak

- `mkdir()` : Dizin oluşturur

```
prompt> strace mkdir foo
…
mkdir("foo", 0777)                      = 0
prompt>
```

  - Dizin ilk başta boştur
  - Fakat boş dizinde iki girdi vardır: . (kendisi), .. (ata-parent)

```
prompt> ls -a
./      ../
prompt> ls -al
total 8
drwxr-x---  2 remzi remzi    6 Apr 30 16:17 ./
drwxr-x--- 26 remzi remzi 4096 Apr 30 16:17 ../
```

Dizinleri Okumak

- Dizin girdilerini okumak için örnek bir kod (`ls` komutuna benzer şekilde):

```
int main(int argc, char *argv[]) {
    DIR *dp = opendir(".");              // open current directory
    assert(dp != NULL);
    struct dirent *d;
    while ((d = readdir(dp)) != NULL)    // read one directory entry
    {
        // print out the name and inode number of each file
        printf("%d %s\n", (int) d->d_ino, d->d_name);
    }
    closedir(dp);                        // close current directory
    return 0;
}
```

- `struct dirent` içindeki bilgiler:

```
struct dirent {
    char            d_name[256];        /* filename */
    ino_t           d_ino;              /* inode number */
    off_t           d_off;              /* offset to the next dirent */
    unsigned short  d_reclen;           /* length of this record */
    unsigned char   d_type;             /* type of file */
}
```

Dizin Silme

- `rmdir()` : Dizini siler.
  - Dizinin boş olması gerekir.
    - sadece “.” and “..” girdilerinin olması.
  - Eğer `rmdir()` boş olmayan bir dizin için çağrılırsa hata verir.

Sabit Bağlantılar (Hard Links)

- `link(old pathname, new one)`
  - Yeni bir dosya adını eskisine bağlama
  - Aynı dosyaya erişmek için başka bir yol oluşturma
  - Komut satırı bağlantı programı: `ln`

```
prompt> echo hello > file
prompt> cat file
hello
prompt> ln file file2  // create a hard link, link file to file2
prompt> cat file2
hello
```

Sabit Bağlantılar (Devam)

- `link` nasıl çalışır:
  - Dizinde başka bir ad oluşturur.
  - Orijinal dosyanın aynı inode numarasına atfedilir.
  - Dosya hiçbir şekilde kopyalanmaz.
  - Bu durumda, artık her iki ad (ör. dosya ve dosya2) da aynı dosyaya atıfta bulunmaktadır.

Sabit Bağlantılar (Devam)

- `link()`:

```
prompt> ls -i file file2
67158084 file  /* inode value is 67158084 */
67158084 file2 /* inode value is 67158084 */
prompt>
```

- İki dosya aynı inode numarasına, ancak iki farklı ada (file, file2) sahiptir.
- file ve file2 arasında fark yoktur.
- Her ikisi de dosyayla ilgili aynı temel meta verilere bağlanır.

Sabit Bağlantılar (Devam)

- Dosyayı kaldırmak için, hangi çağrı yapılır: `unlink()`

```
prompt> rm file
removed ‘file’
prompt> cat file2          // Still access the file
hello
```

- ***referans sayısı:***
  - Bir inode'a kaç farklı dosya adının bağlandığını gösterir.
  - unlink() çağrıldığında referans sayısı bir azalır.
  - Referans sayısı sıfıra ulaşırsa, dosya sistemi inode'u ve ilgili veri bloklarını serbest bırakır → dosyayı gerçekten "siler "
  - Gerçekten silme ne demek???

Sabit Bağlantılar (Devam)

- `unlink()` sonucu:
  - `stat()` bir dosyanın referans sayısını gösterir.

```
prompt> echo hello > file            /* create file*/
prompt> stat file
... Inode: 67158084 Links: 1 ...     /* Link count is 1 */
prompt> ln file file2                /* hard link file2 */
prompt> stat file
... Inode: 67158084 Links: 2 ...     /* Link count is 2 */
prompt> stat file2
... Inode: 67158084 Links: 2 ...     /* Link count is 2 */
prompt> ln file2 file3               /* hard link file3 */
prompt> stat file
... Inode: 67158084 Links: 3 ...     /* Link count is 3 */
prompt> rm file                      /* remove file */
prompt> stat file2
... Inode: 67158084 Links: 2 ...     /* Link count is 2 */
prompt> rm file2                     /* remove file2 */
prompt> stat file3
... Inode: 67158084 Links: 1 ...     /* Link count is 1 */
prompt> rm file3
```

Sembolik Bağlantı (Soft Link)

- Sembolik bağlantı bazen Sabit bağlantıdan daha kullanışlı olabilir.
- Sabit Bağlantı bir dizine oluşturulamaz.
- Başka bir disk bölümündeki bir dosyaya Sabit Bağlantı oluşturulamaz.
  - Çünkü inode numaraları yalnızca bir dosya sistemi içinde benzersizdir.
- Sembolik bir bağlantı oluşturmak için: `ln -s`

```
prompt> echo hello > file
prompt> ln -s file file2  /* option -s: create a symbolic link, */
prompt> cat file2
hello
```

Sembolik Bağlantı (Devam)

- Sembolik bağlantı ile Sabit Bağlantı arasındaki fark nedir?
  - Sembolik bağlantılar, dosya sisteminin bildiği üçüncü bir türdür.

```
prompt> stat file
  ... regular file ...
prompt> stat file2
  ... symbolic link ...        // Actually a file it self of a different type
```

  - Sembolik bağlantı boyutu (`file2`): **4 bayt**.

```
prompt> ls -al
drwxr-x---  2 remzi remzi   29 May 3 19:10 ./
drwxr-x--- 27 remzi remzi 4096 May 3 15:14 ../              // directory
-rw-r-----  1 remzi remzi    6 May 3 19:10 file             // regular file
lrwxrwxrwx  1 remzi remzi    4 May 3 19:10 file2 -> file    // symbolic link
```

  - Sembolik bağlantı, bağlantılan dosyanın yol adını bağlantı dosyasının verisi olarak tutar.

Sembolik Bağlantı (Devam)

- Daha uzun bir yol adına bağlanırsak, bağlantı dosyamız da daha büyük olur.

```
prompt> echo hello > alongerfilename
prompt> ln -s alongerfilename file3
prompt> ls -al alongerfilename file3
-rw-r----- 1 remzi remzi  6 May 3 19:17 alongerfilename
lrwxrwxrwx 1 remzi remzi 15 May 3 19:17 file3 -> alongerfilename
```

Sembolik Bağlantı (Devam)

- Asılı kalan referans (Dangling reference):
  - Orijinal dosyayı kaldırdığımızda, sembolik bağlantı hiçbir şeye işaret etmemektedir.

```
prompt> echo hello > file
prompt> ln -s file file2
prompt> cat file2
hello
prompt> rm file            // remove the original file
prompt> cat file2
cat: file2: No such file or directory
```

Dosya Sistemi Oluşturma ve Kurma («Mount» işlemi)

- `mkfs` aracı: Dosya sistemi oluşturur
  - Bir disk bölümüne kök dizinden başlayarak boş bir dosya sistemi yazar.
  - Girdiler:
    - Bir cihaz (disk bölümü gibi, ör. /dev/sda1) ve bir dosya sistemi türü (ör. ext3)

Dosya Sistemi Oluşturma ve Kurma (Devam)

- `mount()`
  - Mevcut bir dizini hedef bağlama noktası olarak alır.
  - Esasen bu noktada dizin ağacına yeni bir dosya sistemi yapıştırır.
  - **Örnek:**

```
prompt> mount -t ext3 /dev/sda1 /home/users
prompt> ls /home/users
a b
```

  - `/home/users/` yol adı artık yeni eklenen dizinin kök dizinidir.

Dosya Sistemi Oluşturma ve Kurma (Devam)

- `mount` programı: bir sisteme neyin monte (mount) edildiğini gösterir.

```
/dev/sda1 on / type ext3 (rw)
proc on /proc type proc (rw)
sysfs on /sys type sysfs (rw)
/dev/sda5 on /tmp type ext3 (rw)
/dev/sda7 on /var/vice/cache type ext3 (rw)
tmpfs on /dev/shm type tmpfs (rw)
AFS on /afs type afs (rw)
```

  - `ext3:` Standart bir disk tabanlı dosya sistemi
  - `proc:` Geçerli işlemler hakkındaki bilgilere erişmek için bir dosya sistemi
  - `tmpfs:` Yalnızca geçici dosyalar için bir dosya sistemi
  - `AFS:` Dağıtık bir dosya sistemi