

Sistem Programlama

DR. ÖĞR. ÜYESİ ABDULLAH SEVİN

İçerik

a) Cat and its variants. Buffering

b) Files, Links and Inodes

c) Shell Redirection

- <http://web.eecs.utk.edu/~jplank/plank/classes/cs360/360/notes/Cat/lecture.html>
- <http://web.eecs.utk.edu/~jplank/plank/classes/cs360/360/notes/Links/lecture.html>
- <http://web.eecs.utk.edu/~jplank/plank/classes/cs360/360/notes/Sh/lecture.html>

simcat

❑ İşte standart girdiden okuyan ve standart çıktıya yazan basit bir cat programı yazmanın üç eşdeğer yolu.

[src/simpcat1.c](#)

```
/* Using getchar()/putchar(). */  
  
#include <stdio.h>  
#include <fcntl.h>  
#include <stdio.h>  
  
int main()  
{  
    char c;  
  
    c = getchar();  
    while(c != EOF) {  
        putchar(c);  
        c = getchar();  
    }  
    return 0;  
}
```

[src/simpcat2.c](#)

```
/* Using read()/write(). */  
  
#include <unistd.h>  
  
int main()  
{  
    char c;  
  
    while(read(0, &c, 1) == 1) {  
        write(1, &c, 1);  
    }  
    return 0;  
}
```

[src/simpcat3.c](#)

```
/* Using fread()/fwrite(). */  
  
#include <stdio.h>  
  
int main()  
{  
    char c;  
    int i;  
  
    i = fread(&c, 1, 1, stdin);  
    while(i > 0) {  
        fwrite(&c, 1, 1, stdout);  
        i = fread(&c, 1, 1, stdin);  
    }  
    return 0;  
}
```


simcat

- ❑ sh, bir komut satırı dizesinden, standart girdiden veya belirli bir dosyadan okunan komutları yürüten bir komut dili yorumlayıcısıdır.
- ❑ Hangi makineyi kullandığınıza bağlı olarak, muhtemelen yukarıdakinden farklı zamanlar elde edebilirsiniz. *EOF (End of File)

```
UNIX> sh
sh-3.2$ time bin/simpcat1 < data/large.txt > /dev/null

real    0m1.685s
user    0m1.676s
sys     0m0.007s
sh-3.2$ time bin/simpcat2 < data/large.txt > /dev/null

real    0m23.045s
user    0m9.073s
sys     0m13.798s
sh-3.2$ time bin/simpcat3 < data/large.txt > /dev/null

real    0m2.151s
user    0m1.970s
sys     0m0.006s
sh-3.2$
```

```
abduallah@abduallah-VirtualBox:~/sist_prog/cs360-lecture-notes/Cat$ sh
$ time bin/simpcat1 < data/large.txt > /dev/null
0.65user 0.00system 0:00.66elapsed 99%CPU (0avgtext+0avgdata 1476maxresident)k
0inputs+0outputs (0major+62minor)pagefaults 0swaps
$ time bin/simpcat2 < data/large.txt > /dev/null
12.79user 16.50system 0:30.85elapsed 94%CPU (0avgtext+0avgdata 1320maxresident)k
0inputs+0outputs (0major+55minor)pagefaults 0swaps
$ time bin/simpcat3 < data/large.txt > /dev/null
2.39user 0.24system 0:03.51elapsed 75%CPU (0avgtext+0avgdata 1416maxresident)k
0inputs+0outputs (0major+62minor)pagefaults 0swaps
$
```

Simcat

- ❑ /dev/null, Unix'te yazabileceğiniz özel bir dosyadır, ancak hiçbir zaman diskte hiçbir şey saklamaz.
- ❑ Ana dizininizde 25 milyon dosya oluşturmamanız için bu şekilde işlemler yapılmakta, "data/large.txt" 25.000.000 baytlık bir dosyadır.
- ❑ Bu, simpcat1.c'de getchar() ve putchar()'ın, simpcat2.c'de read() ve write() ve simpcat3.c'de fread() ve fwrite() gibi her birinin 25 milyon kez çağrıldığı anlamına gelir.
- ❑ Açıkçası, simpcat2.c'deki **problem**, programın kütüphane çağrıları yerine sistem çağrıları yapmasıdır. Bir sistem çağrısının işletim sistemine yapılan bir istek olduğunu unutmayın.
- ❑ Bu, her okuma/yazma çağrısında, işletim sisteminin CPU'yu devralması (bu, simpcat2 programının durumunu kaydetmesi anlamına gelir), isteği işlemesi ve geri dönmesi (bu, simpcat2 programının durumunu geri yüklemesi anlamına gelir) gerektiği anlamına gelir.

simcat

*time bin/simpcat4 128 < data/large.txt > /dev/null

[src/simpcat4.c](#)

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main(int argc, char **argv)
{
    int bufsize;
    char *c;
    int i;

    bufsize = atoi(argv[1]);
    c = (char *) malloc(bufsize*sizeof(char));
    i = 1;
    while (i > 0) {
        i = read(0, c, bufsize);
        if (i > 0) write(1, c, i);
    }
    return 0;
}
```

[src/simpcat5.c](#)

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main(int argc, char **argv)
{
    int bufsize;
    char *c;
    int i;

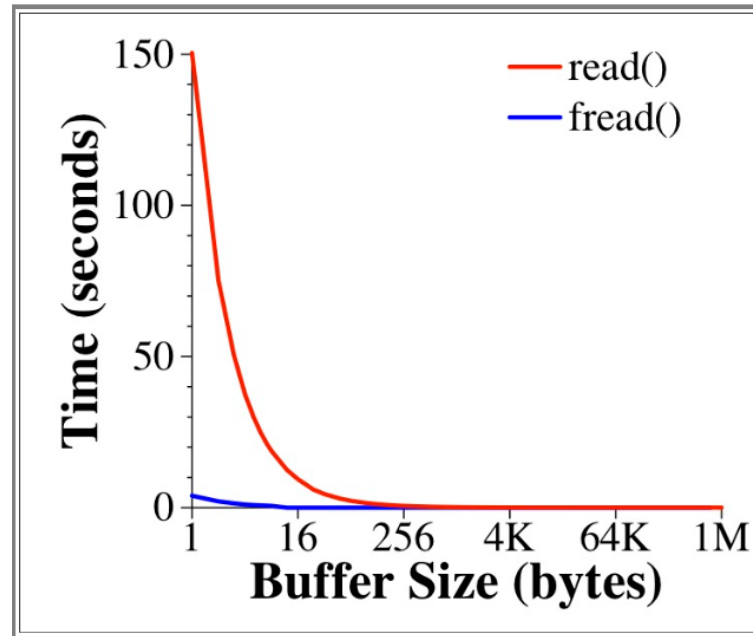
    bufsize = atoi(argv[1]);
    c = (char *) malloc(bufsize*sizeof(char));
    i = 1;
    while (i > 0) {
        i = fread(c, 1, bufsize, stdin);
        if (i > 0) fwrite(c, 1, i, stdout);
    }
    return 0;
}
```

simcat

- ❑ Bunlar, bir seferde **birden fazla bayt** okumamıza izin verir. Buna **tamponlama** denir:
- ❑ Daha az sistem/prosedür çağrısı yapabilmeniz için, bir şeyleri depolamak için bir bellek bölgesi ayırırsınız.
- ❑ fread() ve fwrite()'ın, işletim sistemi yerine standart G/Ç kütüphanesine gitmeleri dışında read() ve write() gibi olduğunu unutmayın.

simcat

- Aşağıdaki grafik göreceli hızlarını gösteriyor (bu, 2016'da MacBook Pro'da, kabaca 8 MB'lık bir girdi dosyasında çalışmıştır. Bunu mevcut large.txt sürümünde çalıştırdığınızda sayılar farklı olacak, ancak grafiğin şekli aynı olmalı)



simcat

- ❑ İlk olarak, standart G/Ç kütüphanesi hakkında şimdi ne çıkarabiliriz? Arabelleğe alma kullanılır!
- ❑ Başka bir deyişle, `getchar()` veya `fread()`'i ilk çağırdığınızda, bir ara belleğe çok sayıda baytlık bir okuma() gerçekleştirir.
- ❑ Böylece sonraki `getchar()` veya `fread()` çağrıları hızlı olacaktır. Geniş bellek segmentlerini `fread()` yapmaya çalıştığınızda, `fread()`'in arabelleğe alınması gerekmediğinden ikisi aynı davranışı sergiler -- sadece `read()` ögesini çağırır.
- ❑ Öyleyse neden `getchar()` `fread(c, 1, 1, stdin)`'den daha hızlı?
- ❑ Çünkü `getchar()` bir karakteri okumak için optimize edilmiştir ve `fread()` değildir.
- ❑ Bir düşünün -- `fread()`'in **dört bağımsız değişkeni okuması gerekir** ve eğer boyutun küçük değerleri için kod yürütüyorsa, kodu çalıştırmadan önce en azından boyutun küçük olduğunu anlaması gerekir. `getchar()`, tek karakterler için gerçekten hızlı olacak şekilde yazılmıştır.

Simcat-Çıkarımlar

1. **Arabelleğe alma-tamponlama**, çok fazla sistem çağrısını azaltmanın iyi bir yoludur.
2. Küçük bayt parçalarını okuyorsanız `getchar()` veya `fread()` kullanın. Sizin için tamponlama yaparlar.
3. Tek karakterli G/Ç yapıyorsanız `getchar()` (veya `fgetc()`) kullanın.
4. Büyük bayt yığınları okuyorsanız, `fread()` ve `read()` aşağı yukarı aynı şekilde çalışır. Ancak, programlamanızı daha tutarlı hale getirdiği ve sizin için biraz daha fazla hata denetimi yaptığı için `fread()`'i kullanmalısınız.

* Aynısı, sınıfta ayrıntılı olarak incelememiş olsak da, **yazmalar (writes)** için de geçerlidir.

Standard I/O vs System calls.

- ❑ Sistem çağrılarını, tamsayı dosya tanımlayıcılarıyla (integer file descriptors) çalışır.
- ❑ Standart G/Ç çağrılarını, DOSYA adı verilen bir yapı tanımlar ve bu yapıya yönelik işaretçilerle çalışır. Kod optimizasyonunda kullanılabilir.

System Call

open
close
read/write

lseek

Standard I/O call

fopen
fclose
getchar/putchar
getc/putc
fgetc/fputc
fread/fwrite
gets/puts
fgets/fputs
scanf/printf
fscanf/fprintf
fseek

simcat

- ❑ Örnek vermek gerekirse, cat programının bağımsız değişkenleri olarak dosya adı ile çağırılması gereken sürümleri aşağıdadır.
- ❑ Cat1.c sistem çağrılarını, cat2.c ise standart G/Ç kütüphanesini kullanır. Her ikisi de read()/fread() ve write()/fwrite() çağrıları için 8K buffer kullanır.

```
abduallah@abduallah-VirtualBox: ~/sist_prog/cs360-lecture-notes/Cat
Dosya Düzenle Görünüm Ara Uçbirim Yardım
abduallah@abduallah-VirtualBox:~/sist_prog/cs360-lecture-notes/Cat$ sh -c "time bin/cat1
data/large.txt > /dev/null"
0.00user 0.01system 0:00.08elapsed 18%CPU (0avgtext+0avgdata 1360maxresident)k
49136inputs+0outputs (0major+60minor)pagefaults 0swaps
abduallah@abduallah-VirtualBox:~/sist_prog/cs360-lecture-notes/Cat$ sh -c "time bin/cat2
data/large.txt > /dev/null"
0.00user 0.00system 0:00.01elapsed 83%CPU (0avgtext+0avgdata 1392maxresident)k
24inputs+0outputs (0major+60minor)pagefaults 0swaps
abduallah@abduallah-VirtualBox:~/sist_prog/cs360-lecture-notes/Cat$
```

simcat

- ❑ Son olarak, src/fullcat.c, cat'ın gerçek sürüme çok benzeyen bir sürümünü içerir -- bir dosya adını atlarsanız, standart girdiyi standart çıktıya yazdırır.
- ❑ Aksi takdirde, komut satırı bağımsız değişkenlerinde belirtilen her dosyayı yazdırır. Hem simpcat1.c hem de cat2.c'ye nasıl benzediğine dikkat edin.

Chars vs ints

- ❑ `getchar()`'ın bir karakter değil, bir int döndürmek üzere tanımlandığını fark edeceksiniz. İlgili olarak, `simpcat1a.c`'ye bakın:
- ❑ `simpcat1a.c` ve `simpcat1.c` arasındaki tek fark, `c`'nin karakter yerine int olmasıdır. Şimdi, bu neden önemli olsun ki? Aşağıdakilere bakın

```
UNIX> ls -l src/simpcat1.c bin/simpcat1
-rwxr-xr-x  1 plank  staff  12604 Feb  5 12:17 bin/simpcat1
-rw-r--r--  1 plank  staff   466 Feb  5 12:15 src/simpcat1.c
UNIX> bin/simpcat1 > tmp1.txt
^C
UNIX> bin/simpcat1 < bin/simpcat1 > tmp1.txt
UNIX> bin/simpcat1 < src/simpcat1.c > tmp2.txt
UNIX> ls -l tmp1.txt tmp2.txt
-rw-r--r--  1 plank  staff  3919 Feb  7 23:37 tmp1.txt
-rw-r--r--  1 plank  staff   466 Feb  7 23:38 tmp2.txt
UNIX>
```

Notice anything wierd? Now:

```
UNIX> bin/simpcat1a < bin/simpcat1 > tmp3.txt
UNIX> ls -l tmp3.txt
-rw-r--r--  1 plank  staff  12604 Feb  7 23:38 tmp3.txt
UNIX>
```

```
#include <stdio.h>
#include <fcntl.h>

int main()
{
    int c;

    c = getchar();
    while(c != EOF) {
        putchar(c);
        c = getchar();
    }
    return 0;
}
```

*Bunun, `getchar()` 255 karakterini okuduğunda olanlarla ilgisi var.

Dosyalar (Files)

- ❑ Dosyalar bilgileri ikincil depolamada depolar.
- ❑ Bu, bilgilerin depolandığı bilgisayar kapalı olsa bile mevcut olması gerektiği anlamına gelir.
- ❑ Bu, yalnızca bilgisayar açıkken çalışan ve bilgisayar kapatıldığında sonsuza kadar kaybolan birincil depolamanın (RAM vb.) tersidir.
- ❑ Unix'te bir dosya oluşturduğunuzda, gerçekleşen pek çok şey gerçekleşir.
- ❑ Bu derste, Unix'te bir dosyanın üç bileşenine odaklanacağız:
 - Dosyanın kendisinin baytları.
 - Dosyanın meta verileri.
 - Bir dizine göre dosyaya olan bağlantılar.

Dosyalar (Files)

❑ `ls -la` gizli dosyalar dahil uzun biçimi listeleyin

❑ `ls -li` liste dosyasının inode dizin numarası

```
abduallah@abduallah-VirtualBox: ~/sist_prog/cs360-lecture-notes/Links
Dosya Düzenle Görünüm Ara Uçbirim Yardım
abduallah@abduallah-VirtualBox:~/sist_prog/cs360-lecture-notes/Links$ echo "This is f1.txt" > f1.txt
abduallah@abduallah-VirtualBox:~/sist_prog/cs360-lecture-notes/Links$ ls -lai
toplam 224
426697 drwxrwxr-x  4 abduallah abduallah  4096 May  7 15:40 .
426589 drwxrwxr-x 42 abduallah abduallah  4096 Mar 12 17:54 ..
407559 -rw-rw-r--  1 abduallah abduallah 137422 Mar 12 17:54 F1-Ex.jpg
397648 -rw-rw-r--  1 abduallah abduallah   15 May  7 15:40 f1.txt
407563 -rwxrwxr-x  1 abduallah abduallah   15 Mar 12 17:54 f2.txt
407565 -rwxrwxr-x  1 abduallah abduallah   15 Mar 12 17:54 f3.txt
407566 -rw-rw-r--  1 abduallah abduallah 11336 Mar 12 17:54 lecture.html
407560 -rw-rw-r--  1 abduallah abduallah 37682 Mar 12 17:54 Link-1.jpg
407561 -rw-rw-r--  1 abduallah abduallah   607 Mar 12 17:54 PN.html
407562 -rw-rw-r--  1 abduallah abduallah   317 Mar 12 17:54 README.md
426698 drwxrwxr-x  2 abduallah abduallah  4096 Mar 12 17:54 src
426699 drwxrwxr-x  2 abduallah abduallah  4096 Mar 12 17:54 txt
```

Dosyalar (Files)

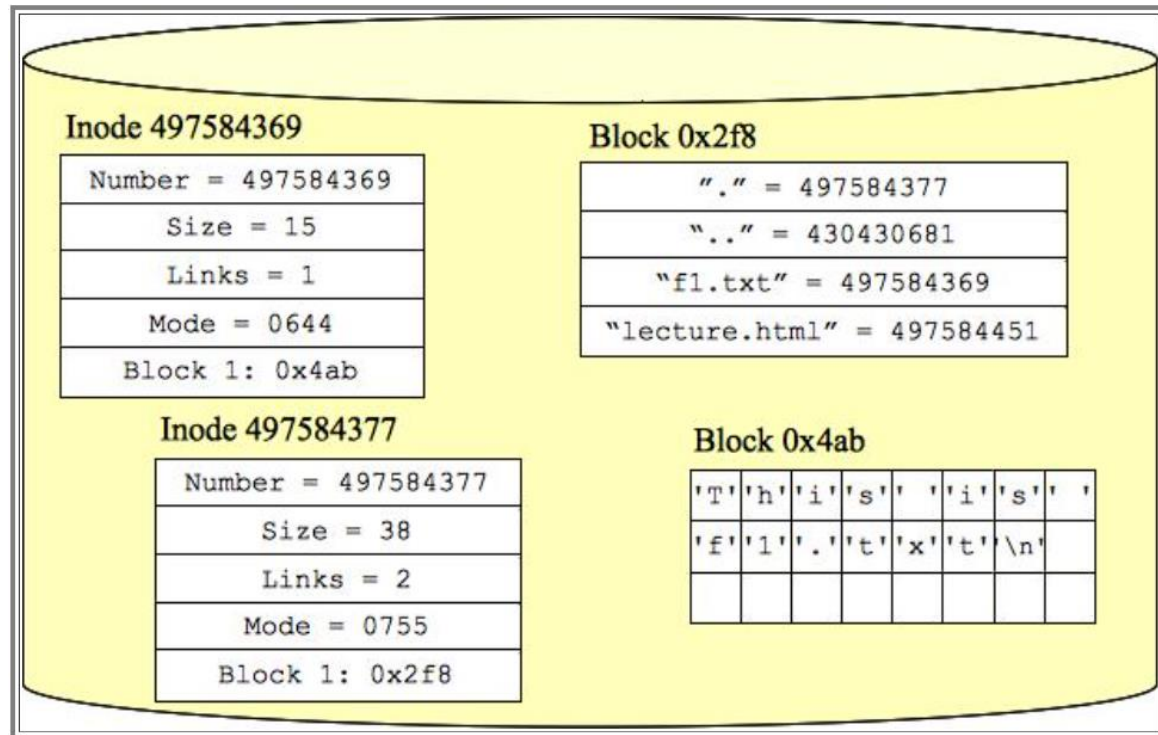
f1.txt adlı bir dosya oluşturduk ve bu, diske üç şey yerleştirir:

1-) Geçerli dizinde "f1.txt" için bir dizin girişi. Bu, "f1.txt" adını numarası 397648 olan bir "inode" ile ilişkilendirir. 397648 numaralı bir "Inode".

2-) Bu, meta verileri veya dosya hakkında bilgileri içerir. Tipik **meta veriler**; boyut, sahip, bağlantılar, koruma modu, en son erişim ve değiştirme zamanı vb.'dir. Bu, diskte kendi konumunda depolanır -- işletim sistemi onu inode numarasına göre nasıl bulacağını bilir.

3-) Gerçek baytlar, "Bu f1.txt". Bunlar bir disk bloğuna yerleştirilir ve inode, disk bloğunun nasıl bulunacağına dair bilgiye sahiptir. Bu örnekte, dosya bir disk bloğu içinde yer almaktadır ve inode onu nasıl bulacağını bilecektir.

Dosyalar



```
UNIX> echo "This is f1.txt" > f1.txt
UNIX> ls -lai
total 20
497584377 drwxr-xr-x  2 plank guest   38 Feb  3 13:54 .
430430681 drwxr-xr-x 51 plank guest 4096 Feb  3 2014 ..
497584369 -rw-r--r--  1 plank loci    15 Feb  3 13:54 f1.txt
497584451 -rw-r--r--  1 plank guest 9896 Feb  3 13:44 lecture.html
UNIX>
```

Dosyalar (Files)

- ❑ f1.txt (397648) için inode'u ve dosyanın baytlarını içeren 0x4ab bloğunu nasıl gösterdiğini görebilirsiniz (işletim sistemi değilseniz bu bilgilere erişiminiz yoktur – rasgele 0x4ab sayısı).
- ❑ Kendisi diskte bir dosya olan dizine ve bu dosyanın inode'una bilgileri ekledik. Her şeyin birbirine nasıl bağlandığını görüyor musunuz?
- ❑ Ayrıca, disk bloğundaki dizgenin sonuna bir boş karakter koymadığımı da fark edeceksiniz. Bunun nedeni boş karakter olmamasıdır -- bu yalnızca bir C programı içinde bir dizge kullandığınızda vardır.
- ❑ Diske yazdığınızda boş karakter yoktur. -i bayrağını ls'ye verdiğinizde, yukarıdaki örnekte olduğu gibi size inode numarasını söyleyecektir.

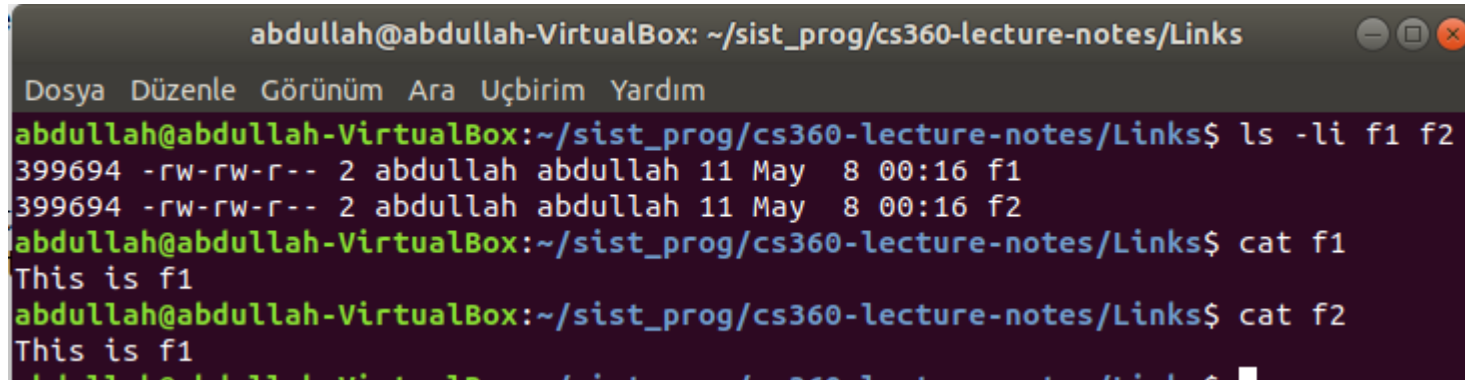
Dosyalar

- ❑ Unix dilinde, bir dosyayı adlandırma şeklimiz, inode'a bir "bağlantı" eklemektir.
- ❑ Bağlantılar "dizinlerde" saklanır - bir dizindeki her giriş, bağlantının adını dosyaya işaret eden inode'un inode numarasına eşler. Bir dosyaya birden fazla bağlantı noktamız olabilir.
- ❑ Yeni bir dizinde olduğumuzu ve f1 dosyasını "Bu f1\n" baytlarını içerecek şekilde oluşturduğumuzu varsayalım.
- ❑ Ayrıca, bu dosyanın inode numarası 34778 olduğunu varsayalım. Şimdi aşağıdakileri yapıyoruz:

UNIX> **ln f1 f2**
- ❑ Bu, f1 dosyasına başka bir bağlantı oluşturmayı ve onu "f2" olarak adlandırmayı söylüyor.
- ❑ Bu bağlantı gerçekten de "f2"yi 34778 inode'una eşleyen dizindeki bir giriştir. Şu anda sahip olduğumuz şey, aynı meta verilere ve diskteki aynı baytlara iki işaretçi.

Dosyalar

- ❑ Listeleme yaptığımızda:



```
abdullah@abdullah-VirtualBox: ~/sist_prog/cs360-lecture-notes/Links
Dosya Düzenle Görünüm Ara Uçbirim Yardım
abdullah@abdullah-VirtualBox:~/sist_prog/cs360-lecture-notes/Links$ ls -li f1 f2
399694 -rw-rw-r-- 2 abdullah abdullah 11 May  8 00:16 f1
399694 -rw-rw-r-- 2 abdullah abdullah 11 May  8 00:16 f2
abdullah@abdullah-VirtualBox:~/sist_prog/cs360-lecture-notes/Links$ cat f1
This is f1
abdullah@abdullah-VirtualBox:~/sist_prog/cs360-lecture-notes/Links$ cat f2
This is f1
```

- ❑ Bağlantıların farklı adlara sahip olması dışında dosyaların tamamen aynı olduğunu görüyoruz.



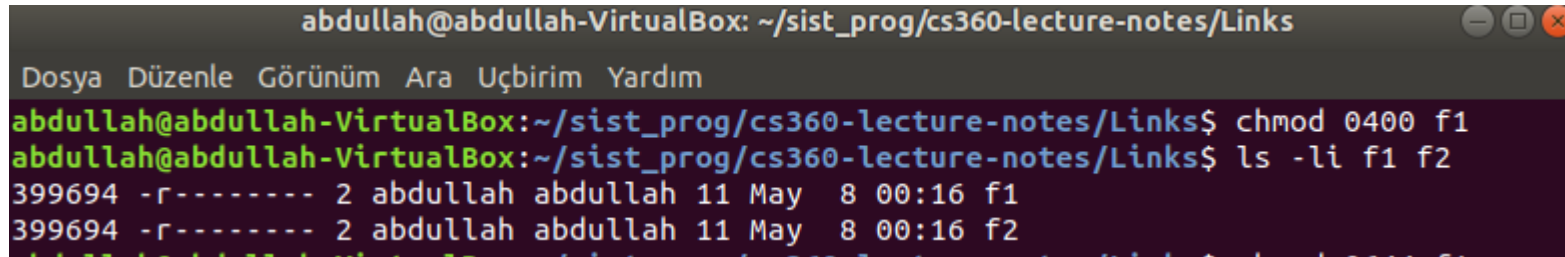
Dosyalar

- ❑ Bu dosyalardan herhangi birini değiştirirsek -- örneğin, f2'yi vi kullanarak düzenleyelim ve "This" kelimesini "That" olarak değiştirelim, o zaman değişiklik hem f1'de hem de f2'de görülür çünkü ikisi de aynı bayta işaret eder.

```
UNIX> vi f2
...
UNIX> cat f2
That is f1
UNIX> cat f1
That is f1
UNIX> ls -li f1 f2
34778 -rw-r--r-- 2 plank 11 Sep 16 10:14 f1
34778 -rw-r--r-- 2 plank 11 Sep 16 10:14 f2
UNIX>
```

Dosyalar

- ❑ Sadece f2'yi değiştirmiş olmamıza rağmen, f1 için dosya değiştirme zamanının da değiştiğini unutmayın.
- ❑ Bunun nedeni, dosya değiştirme süresinin inode'un bir parçası olarak saklanmasıdır - bu nedenle, f2 onu değiştirdiğinde, değişiklik f1'de de görülür.
- ❑ Dosya koruma modlarıyla aynı. f1 için korumayı değiştirirsek, f2'deki değişiklikleri göreceğiz:



```
abduallah@abduallah-VirtualBox: ~/sist_prog/cs360-lecture-notes/Links
Dosya Düzenle Görünüm Ara Uçbirim Yardım
abduallah@abduallah-VirtualBox:~/sist_prog/cs360-lecture-notes/Links$ chmod 0400 f1
abduallah@abduallah-VirtualBox:~/sist_prog/cs360-lecture-notes/Links$ ls -li f1 f2
399694 -r----- 2 abduallah abduallah 11 May  8 00:16 f1
399694 -r----- 2 abduallah abduallah 11 May  8 00:16 f2
```

- ❑ ls komutunun üçüncü sütununa dikkat edin. Dosyaya giden bağlantıların sayısıdır. F1'e başka bir bağlantı yaparsak, bu sütun güncellenecektir:

Dosyalar

- ❑ `ls` komutunun üçüncü sütununa dikkat edin. ***Dosyaya giden bağlantıların sayısıdır.*** F1'e başka bir bağlantı yaparsak, bu sütun güncellenecektir:

```
UNIX> ln f1 f3
UNIX> ls -li f1 f2 f3
34778 -r----- 3 plank          11 Sep 16 10:14 f1
34778 -r----- 3 plank          11 Sep 16 10:14 f2
34778 -r----- 3 plank          11 Sep 16 10:14 f3
```

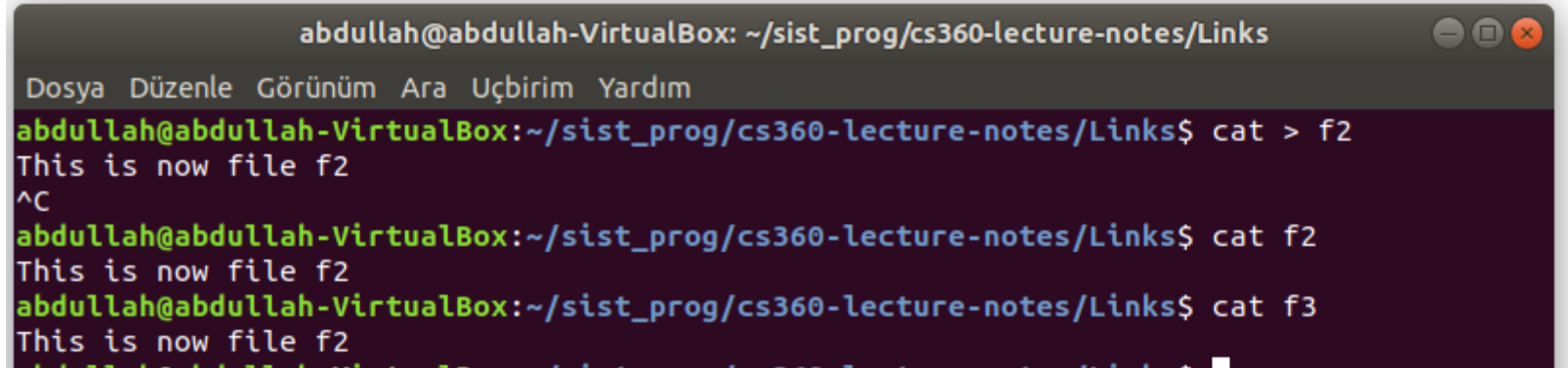
- ❑ "`rm`" komutunu kullandığımızda aslında linkleri kaldırmış oluyoruz. Örneğin.

```
abdullah@abdullah-VirtualBox:~/sist_prog/cs360-lecture-notes/Links$ chmod 0644 f1
abdullah@abdullah-VirtualBox:~/sist_prog/cs360-lecture-notes/Links$ rm f1
abdullah@abdullah-VirtualBox:~/sist_prog/cs360-lecture-notes/Links$ ls -li f*
397648 -rw-rw-r-- 1 abdullah abdullah 15 May  7 15:40 f1.txt
399694 -rw-r--r-- 1 abdullah abdullah 11 May  8 00:16 f2
407563 -rwxrwxr-x 1 abdullah abdullah 15 Mar 12 17:54 f2.txt
407565 -rwxrwxr-x 1 abdullah abdullah 15 Mar 12 17:54 f3.txt
```

Dosyalar

- ❑ Bir dosyanın son bağlantısı kaldırıldığında, dosyanın kendisi, inode ve tümü silinir. Bununla birlikte, bir dosyaya işaret eden bir bağlantı olduğu sürece, dosya kalır.
- ❑ Bağlantı içeren dosyaların üzerine yazıldığında ne olduğunu görmek ilginç.

```
UNIX> cat > f2
This is now file f2
^D
UNIX> cat f2
This is now file f2
UNIX> cat f3
This is now file f2
```



```
abdullah@abdullah-VirtualBox: ~/sist_prog/cs360-lecture-notes/Links
Dosya Düzenle Görünüm Ara Uçbirim Yardım
abdullah@abdullah-VirtualBox:~/sist_prog/cs360-lecture-notes/Links$ cat > f2
This is now file f2
^D
abdullah@abdullah-VirtualBox:~/sist_prog/cs360-lecture-notes/Links$ cat f2
This is now file f2
abdullah@abdullah-VirtualBox:~/sist_prog/cs360-lecture-notes/Links$ cat f3
This is now file f2
```

- ❑ Çıktıyı f2 dosyasına yönlendirmek istediğinizi söyleyerek, sonunda f3'ü değiştirirsiniz. Bu, kabuğun çıktı yeniden yönlendirmesi gerçekleştirdiğinde, dosyayı kaldırıp yeniden oluşturmak yerine dosyayı açıp kestiği anlamına gelir.

Dosyalar

- ❑ c derleyicisi gcc'nin çalıştırılabilir olarak f2'yi oluşturmada önce bir "rm f2" yaptığını fark edeceksiniz.

```
abdullah@abdullah-VirtualBox:~/sist_prog/cs360-lecture-notes/Links$ gcc -o f2 ../Stat/  
src/ls1.c  
../Stat/src/ls1.c: In function 'main':  
../Stat/src/ls1.c:26:20: warning: format '%lld' expects argument of type 'long long in  
t', but argument 2 has type 'off_t {aka long int}' [-Wformat=]  
    printf("%10lld %s\n", size, argv[i]);  
        ~~~~~^  
        %10ld  
abdullah@abdullah-VirtualBox:~/sist_prog/cs360-lecture-notes/Links$ ls -li f*  
397648 -rw-rw-r-- 1 abdullah abdullah  15 May  7 15:40 f1.txt  
399401 -rwxrwxr-x 1 abdullah abdullah 8432 May  8 00:55 f2  
407563 -rwxrwxr-x 1 abdullah abdullah  15 Mar 12 17:54 f2.txt  
399694 -rw-r--r-- 1 abdullah abdullah  20 May  8 00:41 f3  
407565 -rwxrwxr-x 1 abdullah abdullah  15 Mar 12 17:54 f3.txt
```


Dosyalar

- ❑ Tüm dizinlerin en az 2 bağlantısı vardır:

```
abdullah@abdullah-VirtualBox: ~/sist_prog/cs360-lecture-notes/Links
Dosya Düzenle Görünüm Ara Uçbirim Yardım
abdullah@abdullah-VirtualBox:~/sist_prog/cs360-lecture-notes/Links$ mkdir test
abdullah@abdullah-VirtualBox:~/sist_prog/cs360-lecture-notes/Links$ ls -li | grep test
442449 drwxrwxr-x 2 abdullah abdullah 4096 May  8 00:57 test
```

- ❑ Bunun nedeni, her dizinin iki "." alt dizini içermesidir. ve ".." İlki kendisine bir bağlantıdır ve ikincisi üst dizine bir bağlantıdır.
- ❑ Böylece, "test" dizin dosyasına iki bağlantı vardır: "test" ve "test/." Benzer şekilde, bir test alt dizini oluşturduğumuzu varsayalım:

```
UNIX> mkdir test/sub
UNIX> ls -li | grep test
34800 drwxr-xr-x 3 plank 512 Sep 16 10:17 test
UNIX>
```

- ❑ Artık "test" için üç bağlantı vardır: "test", "test/." ve "test/sub/.."

Dosyalar

- ❑ Sizin için otomatik olarak oluşturulan bu bağlantıların yanı sıra, manuel olarak dizinlere bağlantılar oluşturamazsınız.
- ❑ Bunun yerine, "ln -s" komutunu kullanarak oluşturduğunuz "sembolik bağlantı" ("soft link" olarak da adlandırılır) adı verilen özel bir bağlantı türü vardır.
- ❑ Örneğin, test dizinine aşağıdaki gibi hafif bir bağlantı oluşturabiliriz:

```
abdullah@abdullah-VirtualBox:~/sist_prog/cs360-lecture-notes/Links$ ln -s test test-soft
abdullah@abdullah-VirtualBox:~/sist_prog/cs360-lecture-notes/Links$ ls -li | grep test
442449 drwxrwxr-x 2 abdullah abdullah 4096 May  8 00:57 test
403862 lrwxrwxrwx 1 abdullah abdullah  4 May  8 01:02 test-soft -> test
```

- ❑ Hafif bağlantıların farklı türde bir dizin listesine sahip olduğunu unutmayın. Ayrıca, "test" için hafif bir bağlantı oluşturulmasının, test inode'unun bağlantı alanını güncellemediğini unutmayın. Bu, yalnızca normal veya "sabit" bağlantıları kaydeder.

Dosyalar

- ❑ Hafif bağlantı, dosyanın inode'unu değiştirmeden bir dosyaya işaret etmenin bir yoludur.
- ❑ Bununla birlikte, hafif bağlantılar, sabit bağlantıların yapabildiği hemen hemen her şeyi yapabilir:

```
UNIX> cat > f1
This is f1
UNIX> ln -s f1 f2
UNIX> cat f2
This is f1
UNIX> cat > f2
This is f2
UNIX> cat f1
This is f2
UNIX> ls -l f*
-rw-r--r-- 1 plank 11 Sep 16 10:19 f1
lrwxrwxrwx 1 plank 2 Sep 16 10:18 f2 -> f1
UNIX> chmod 0600 f2
UNIX> ls -l f*
-rw----- 1 plank 11 Sep 16 10:19 f1
lrwxrwxrwx 1 plank 2 Sep 16 10:18 f2 -> f1
UNIX>
```

Dosyalar

- ❑ O halde sert ve hafif bağlantılar arasındaki temel fark nedir? Peki, hafif bağlantının işaret ettiği dosya silinir veya taşınırsa, bağlantı kullanılamaz hale gelir:

```
UNIX> rm f1
UNIX> ls -l f*
lrwxrwxrwx  1 plank          2 Sep 16 10:18 f2 -> f1
UNIX> cat f2
cat: f2: No such file or directory
UNIX>
```

The link is called "unresolved."

Soru-1

1-) konsoldan bir tane klasör altına dosya oluşturup içerisine isminizi yazın. Buna bağlı olan 3 tane daha dosya oluşturun-konsoldan(link).
Sonra bu üç dosyayı cat ile konsola yazdırın.

Yönlendirme (Shell Redirection)

- ❑ Unix altında insanların kullandığı birçok farklı kabuk vardır.
- ❑ Ders, C kabuğuna bazı referanslarla birlikte Bourne Kabuğuna odaklanır. "Bash" kabuğu, Bourne Kabuğundan türetilmiştir, dolayısıyla Bourne Kabuğu betikleriniz bash üzerinde çalışacaktır.
- ❑ Burada ilgilendiğimiz şey yönlendirme ilkelleridir.
- ❑ Bunların çoğu basittir ve hemen hemen tüm kabuklarda aynıdır.
- >f: Standart çıktıyı f dosyasına yazar.
- >> f: Standart çıktıyı f dosyasına ekler.
- < f: f dosyasından standart girdi alır.

Yönlendirme

- ❑ Örneğin, f1 dosyasının ``This is f1" baytlarını içerdiğini varsayalım. Aşağıdaki yönlendirmeler kafanızı karıştırmamalı. Her durumda, kendinize komutun çıktısının ne olması gerektiğini sorun:

```
UNIX> cat f1
This is f1
UNIX> cat < f1
This is f1
UNIX> < f1 cat      You can put the redirection anywhere in the command line.
This is f1
UNIX> < cat f1      This is the same as f1 < cat - it can't find the file "cat".
cat: No such file or directory.
UNIX> cat f1 > f2
UNIX> cat f2
This is f1
UNIX> cat f1 >> f2
UNIX> cat f2
This is f1
This is f1
UNIX> > f2 < f1 cat  This is the same as cat < f1 > f2
UNIX> cat f2
This is f1
UNIX>
```

Yönlendirme

- ❑ Şimdi, f3 dosyası olmadığını varsayalım. `cat f1 f3` dediğimizde f1'in içeriğini standart çıktıya ve hata mesajını standart hataya yazdıracaktır. Tipik olarak, bunların ikisi de ekrana gider:

```
UNIX> cat f1 f3
This is f1
cat: f3: No such file or directory
UNIX>
```

- ❑ Ancak, standart çıktıyı bir dosyaya yönlendirirseniz, f1 dosyaya gidecek ve hata mesajı ekrana gidecektir. Neden? Kabuk, çıktı dosyasına `dup2(fd, 1)` çağırdığı, ancak dosya tanımlayıcı 2 için hiçbir şey çağırmadığı için:

```
UNIX> cat f1 f3 > f2
cat: f3: No such file or directory
UNIX> cat f2
This is f1|
UNIX>
```

Yönlendirme

- ❑ C kabuğuyla, `>&` kullanarak **hem standart çıktıyı hem de standart hatayı** aynı dosyaya yönlendirebilirsiniz:

```
UNIX> csh -c 'cat f1 f3 >& f2'
UNIX> cat f2
This is f1
cat: f3: No such file or directory
UNIX>
```

- `csh` komutu C kabuğunu çağırır.

- ❑ Bourne kabuğu, standart çıktı ve standart hatayla başa çıkmak için farklı ilkelere sahiptir. Ne zaman `x>` dersanız, `x` dosya tanıtıcısını yeniden yönlendirir. Örneğin, standart çıktıyı Bourne kabuğunun içerdiği bir dosyaya yönlendirmenin başka bir yolu vardır:

```
UNIX> cat f1 f3 1>f2
cat: cannot open f3
UNIX> cat f2
This is f1
UNIX>
```

Yönlendirme

- Ve standart çıktıyı ve standart hatayı farklı dosyalara çok kolay bir şekilde yönlendirebiliriz:

```
UNIX> cat f1 f3 1>f2 2>f5
UNIX> cat f2
This is f1
UNIX> cat f5
cat: cannot open f3
UNIX>
```

- Kabuk bu ifadeleri soldan sağa işler, böylece standart çıktının birden çok yeniden yönlendirmesini yapabilirim ve kabuk belirttiğiniz tüm dosyaları oluşturur:

```
UNIX> rm f2
UNIX> cat f1 f3 1>f2 1>f5
cat: cannot open f3
UNIX> cat f2
UNIX> cat f5
This is f1
UNIX>
```

Yönlendirme

- ❑ Gördüğünüz gibi f2 oluşturuldu ve boş. Bunun nedeni, ilk yönlendirmede yazmak için açılmış, ardından ikinci yönlendirme deyiminde kapatılmış olmasıdır. Standart girişi birden çok kez yönlendirebilir miyiz? Kabuğunuza bağlıdır:

```
UNIX> echo "This is f1" > f1
UNIX> echo "This is f2" > f2
UNIX> sh -c "cat < f1 < f2"
This is f2
UNIX> sh -c "cat < f2 < f1"
This is f1
UNIX> csh -c "cat < f1 < f2"
Ambiguous input redirect.
UNIX>
```

- ❑ Girdi ve çıktıyı aynı dosyadan almaya çalıştığınızda ne olduğuna bakın:

```
UNIX> cat f2
This is f2
UNIX> head f2 > f2
UNIX> cat f2
```

Yönlendirme

- ❑ Kabuk, yönlendirmeyi **head** komutunu çalıştırmadan önce yapar. Bu, f2'nin **head** çağrılmadan önce kesildiği ve **head** çağrıldığında f2'nin boş olduğu anlamına gelir.
- ❑ Bu nedenle, **head** oluşur ve f2 boş kalır. **Head'in** standart girdisini standart çıktıya yönlendirirseniz aynı şey olur:

```
UNIX> echo "This is f2" > f2  
UNIX> head < f2 > f2  
UNIX> cat f2  
UNIX>
```

 - head komutu, belirtilen dosyaların ilk 10 satırını yazdıran bir komut satırı yardımcı programıdır.
- ❑ Şimdi x>y'yi yeniden ele alalım. y'yi &y olarak belirtirseniz, **programdaki x dosya tanımlayıcısının y dosya tanımlayıcısıyla aynı olmasını sağlar.**
- ❑ Bunu söylemenin başka bir yolu da şudur: "Y dosya tanıtıcısı şu anda nereye gidiyorsa, şimdi x dosya tanımlayıcısı da oraya gidiyor ve ikisi aynı."

Yönlendirme

- (Bunun dup2() sistem çağrısıyla nasıl çalıştığını daha sonra öğreneceksiniz). Bu nedenle, aşağıdakilere bakın:

```
UNIX> cat f1 f3 > f2 2>&1
UNIX> cat f2
This is f1
cat: cannot open f3
```

- Ne oluyor? İlk olarak, standart çıktıyı f2'ye yönlendirirsiniz. Bu, dosya tanımlayıcı 1'in f2'ye gideceği anlamına gelir.
- Sonra 2>&1 kısmı, dosya tanımcı 2'yi dosya tanımcı 1 ile aynı hale getirmenizi söylüyor. Bu, standart hatanın f2'ye de gideceği anlamına geliyor. **Yani çıktı ve hata beraber olacak**

Yönlendirme

- ❑ Yine, bunlar kabuk tarafından **soldan sağa doğru** işlenir. İfadelerin sırasını *tersine çevirdiğinizi* varsayalım:

```
UNIX> cat f1 f3 2>&1 > f2
cat: cannot open f3
UNIX> cat f2
This is f1
```

- ❑ Şimdi, 2>&1 kısmı, dosya tanıtıcı 2'yi, kabuğun bu komutu gördüğü anda ekrana giden dosya tanımlayıcı 1 ile aynı yapmasını söylüyor.
- ❑ Ardından, dosya tanımlayıcı 1'i f2'ye yönlendirir. Böylece ekrana standart hata gider ve standart çıktı f2'ye gider.

Yönlendirme

```
UNIX> cat f1 f3 >f2 2>&1 1>f5
UNIX> cat f2
cat: cannot open f3
UNIX> cat f5
This is f1
```

- ❑ Şimdi, standart çıkış önce f2'ye gider, ardından 2>&1 kısmı standart hatayı standart çıkışla aynı yapar.
- ❑ Başka bir deyişle, her ikisi de f2'ye gidiyor. Daha sonra 1>f5 kısmı f5'e standart çıktı verir.

Yönlendirme

□ Bu nedenle, bu satır şuna eşdeğerdır: ``cat f1 f3 >f5 2>f2."İsterseniz diğér dosya tanımlayıcılardan yararlanabilirsiniz:

```
UNIX> cat f1 f3 3>f2 1>f5 2>&1 1>&3
UNIX> cat f2
This is f1
UNIX> cat f5
cat: cannot open f3
```

Yönlendirme

```
/* This is a program that assumes file descriptor 3 is open, and writes to it. */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>

int main()
{
    char *s = "Hi!\n";
    int i;
    int fd;

    i = write(3, s, strlen(s));
    printf("%d\n", i);
    if (i < 0) perror("write");
    return 0;
}
```

Yönlendirme

- ❑ Şimdi aşağıdakileri kontrol edelim. Bu iyi bir şey ya da kötü bir şey mi?
- ❑ Aslında, hata ayıklamaya yardımcı olması için aşağıdakileri yaptım:
- ❑ Diyelim ki oldukça büyük bir kod parçasında ufak bir hatanız var.
- ❑ Ve size yardımcı olacak bazı çıktıları oluşturmak istiyorsunuz, ancak standart çıktıyı kullanamayacak kadar çok çöpe attınız.
- ❑ Daha da kötüsü, hatanın çok fazla prosedür çağrısında yuvalandığını biliyorsunuz ve kontrol akışının oraya gitmesi veya FILE '*'ları tüm bu prosedür çağrılarına iletmesi konusunda endişelenmek istemiyorsunuz.

```
UNIX> bin/badbadcode
-1
write: Bad file number
UNIX> bin/badbadcode 3>f5
4
UNIX> cat f5
Hi!
UNIX> bin/badbadcode 3>&1
Hi!
4
```

Yönlendirme

[src/dont_admit_i_taught_you_this.c](#):

....

- ❑ Gördüğünüz gibi, `v()` çok çağrılıyor. Her çağrıldığında, dosya tanımlayıcı 9'a bir dize yazıyor. Dosya tanımlayıcı 9 nedir? Peki, bunu Shell ile belirleyebilirsiniz:

```
UNIX> bin/dont_admit_i_taught_you_this > /dev/null 9>txt/elog.txt
UNIX> cat txt/elog.txt
Here's my error message.  V was called with 4
Here's my error message.  V was called with 4
Here's my error message.  V was called with 4
Here's my error message.  V was called with 4
Here's my error message.  V was called with 4
Here's my error message.  V was called with 4
Here's my error message.  V was called with 4
Here's my error message.  V was called with 4
Here's my error message.  V was called with 4
Here's my error message.  V was called with 5
Here's my error message.  V was called with 4
Here's my error message.  V was called with 4
Here's my error message.  V was called with 4
Here's my error message.  V was called with 4
Here's my error message.  V was called with 4
Here's my error message.  V was called with 4
Here's my error message.  V was called with 4
Here's my error message.  V was called with 4
Here's my error message.  V was called with 5
Here's my error message.  V was called with 6
Here's my error message.  V was called with 7
Here's my error message.  V was called with 8
UNIX> bin/dont_admit_i_taught_you_this > /dev/null
.....
```

- ❑ O son çağrıda, dosya tanımlayıcı 9'u yeniden yönlendirmedik.
- ❑ Bu nedenle, `write()` ifadesi başarısız oldu ve -1 döndürdü. Hiçbir şey olmamış gibi.