Sistem Programlama

Ders 4

Doç. Dr. Mehmet Dinçer Erbaş
Bolu Abant İzzet Baysal Üniversitesi
Mühendislik Fakültesi
Bilgisayar Mühendisliği Bölümü

Dosya girdi çıktı işlemleri

- Dosya girdi / çıktı
  - Bu işlemler için kullanılan fonksiyonlar genellikler aşağıda belirtilen işlemleri yapar:
    - Dosya aç, oku, yaz vb.
  - **open**, **read**, **write**, **lseek**, **close**
  - Önbellek (ing: buffer) büyüklüğünün **read** ve **write** fonksiyonlarının çalışma hızına etkilerini inceleyeceğiz.
  - Yukarıda belirtilen fonksiyonlar tamponsuz veya önbelleksiz (unbuffered) girdi çıktı fonksiyonları olarak bilinir.
    - Read ve write fonksiyonları kernelde çalışan sistem çağrılarını başlatır.
  - Önbelleksiz I/O fonksiyonları ISO C standartının parçası değildir, ancak POSIX.1 ve Single Unix standartlarında bulunur

Dosya I/O

- Ayrıca birden fazla işlemin dosyaları nasıl paylaştığını ve bu işlemlerin bölünmeden nasıl yapıldığını inceleyeceğiz.
  - dup, fcntl, sync, fsync ve ioctl fonksiyonları

Dosya I/O

- Kernel sistemde kullanılan her dosyaya dosya belirteçleri üzerinden erişir.
  - Dosya belirteçleri negatif olmayan tamsayılardır.
  - Yeni bir dosya oluşturduğumuzda veya bulunan bir dosyayı açtığımızda kernel bu dosyaya bir dosya belirteci döner.
  - Bu dosya belirtecini kullanarak kullanmak istediğimiz dosyaya erişebiliriz.
- Unix kabuğu belli dosya belirteçlerini otomatik olarak oluşturur.
  - Dosya belirteci 0 ==> standart girdi
  - Dosya belirteci 1 ==> standart çıktı
  - Dosya belirteci 2 ==> standart hata
- Direk olarak tamsayı değerlerini kullanmaktansa sembolik sabitleri kullanmak daha güvenlidir: STDIN_FILENO, STDOUT_FILENO, STDERR_FILENO.

Dosya I/O

- Dosya belirteçleri 0 ile OPEN_MAX arasında değer alabilir.
  - En eski Unix versiyonlarında bu değer 19 olarak belirlenmiştir.
  - Sonraki versiyonlarda bu değer 63'e çıkarılmıştır.
  - MAC OS X ve Solaris 9 gibi sistemlerde bu değer limitsizdir ve ancak sistemdeki hafıza miktarı, bir tamsayının alabileceği değer veya sistem yöneticisinin belirlediği bir değer ile sınırlanmıştır.
  - Linux 2.4.22 bir işlemin kullanabileceği dosya belirteci sayısını 1,048,576 olarak belirlemiştir.

I/O fonksiyonları

- Open fonksiyonu: open fonksiyonunu çağıran işlem tarafından bir dosya açılır veya oluşturulur

```
#include <fcntl.h>

int open(const char *pathname, int oflag, .../* mode_t mode */ );
```

Dönüş: OK ise dosya belirteci, hata ise -1.

- Son parametre kısaltılmış şekilde gösterilmiştir. Bunun anlamı geri kalan parametreler farklı sayıda olabilir demektir.
- Bu fonksiyonda son parametre yeni bir dosya oluşturulduğunda kullanılacaktır.
- Pathname değişkeni açılacak veya oluşturulacak dosyanın adıdır.

I/O fonksiyonları

- oflag değişkeni dosya açma şeklini belirler:
  - O_RDONLY Sadece okumak için aç
  - O_WRONLY Sadece yazmak için aç
  - O_RDWR Okumak ve yazmak için aç
- Bu seçeneklerden tam olarak bir tanesinin belirtilmesi gerekir.
- Aşağıdaki sabitler opsiyonel olarak belirtilebilir
  - O_APPEND Her yazma operasyonunda dosyanın sonuna yazar.
  - O_CREAT Dosya bulunmuyorsa oluşturulur.
  - O_EXCL O_CREAT yapılır ve dosya mevcutsa hata oluşturur.
  - O_TRUNC Eğer dosya mevcutsa ve O_WRONLY veya O_RDWR Ile açılmışsa, dosya uzunluğu 0 olur
  - O_NOCTTY Dosya terminal bir cihaza erişiyorsa, bu cihazı kontrol terminali olarak belirlemez.
  - O_NONBLOCK Dosya FIFO, block özel veya karakter özel bir dosyaya işaret ediyorsa bloklama yapmaz

I/O fonksiyonları

- open fonksiyonu
  - Aşağıda belirtilen bayraklar opsiyonel olarak kullanılır
    - O_DSYNC Her yazma operasyonu fiziksel olarak tamamlanır. Dosya özelliklerinin yazılması beklenmez.
    - O_RSYNC Yazma bekleyen hafıza bloklarında okuma yapılmaz.
    - O_SYNC Her yazma operasyonu dosya özellikleri dahil tamamlanır.
  - open fonksiyonu en düşük değere sahip dosya belirtecini döner.

I/O fonksiyonları

- open fonksiyonu
  - Gereğinden fazla uzun yoladı veya dosya ismi verildiğinde iki farklı durum oluşabilir.
    - Örneğin MAX_NAME 14 ise ve 15 karakterlik bir isme sahip dosya oluşturulmak istendiğinde bir sorun oluşur.
  - System V'nin eski versiyonlarında verilen isim sessizce kısaltılır.
  - BSD-türevi sistemlerde bir hata döner. errno değer ENAMETOOLONG olarak atanır.
  - POSIX.1 standartının bir parçası olan POSIX_NO_TRUNC sabiti uzun dosya isimlerinin kısaltılma veya hataya neden olma durumunu belirtir.

I/O fonksiyonları

- creat fonksiyonu: Yeni bir dosya ayrıca creat fonksiyonu ile yaratılabilir.

```
#include <fcntl.h>

int creat(const char *pathname, mode_t mode);
```

Dönüş: OK ise dosya belirteci, hata ise -1.

- Bu fonksiyon open(2) tarafından kullanılmaz hale getirilmiştir.
  - Bu fonksiyonun kullanımı aşağıdaki ile aynıdır.
    - `open(pathname, O_WRONLY | O_CREAT | O_TRUNC, mode);`
- creat fonksiyonu ile önemli sorunlardan biri dosyanın sadece yazma için açılmasıdır. Kısa süreli kullanacağınız bir dosyaya bir şeyler yazıp okumak istediğinizde creat, close ve sonra open kullanmanız gerekir. Bunun yerine open ile hem yazma hem okuma için açabiliriz.
  - `open(pathname, O_RDWR | O_CREAT | O_TRUNC, mode);`

I/O fonksiyonları

- read fonksiyonu: açık olan dosyadan veri okunması için kullanılır.

```
#include <unistd.h>

sseize_t read(int filedes, void *buff, size t_nbytes);
```

Dönüş: okunan bytes sayısı, dosya sonu 0, hata ise -1.

  - Eğer dosyanın sonunda istenen az byte kalmışsa, okunan kadar byte döner.
  - Read bulunan ofset değerinden başlayarak okur, okuma işlemi tamamladıktan sonra ofseti okunan byte kadar artırır.

I/O fonksiyonları

- write fonksiyonu: Belirtilen açık dosyaya veri yazmak için kullanılır.

```
#include <unistd.h>

ssize_t write(int filedes, void *buff, size_t nbytes);
```

Dönüş: OK ise yazılan byte sayısı, hata ise -1.

  - Write fonksiyonu yazılan byte sayısını veya hata döner.
  - O_APPEND eklenmemişse write dosyanın başlangıcından yazmaya başlar.
  - write tamamladıktan sonra ofset değeri yazılan byte kadar artar.

I/O fonksiyonları

- close fonksiyonu: Açık bir dosya close fonksiyonu ile kapatılır

```
#include <unistd.h>

int close(int fd);
```

Dönüş: OK ise 0, hata ise -1.

- Bir dosya kapatıldığında o dosya üzerinde kilit bulunduran işlemlerin kilitleri ortadan kalkar.
- Bir işlem sonlandığında o işlemin açtığı bütün dosyalar kernel tarafından kapatılır.
- Örnekler
  - mycat.c
  - ders4.c

I/O fonksiyonları

- lseek fonksiyonu
  - Her açılan dosya ile “okunmakta olan dosya ofseti” değeri saklanır.
    - Bu değer genellikle başlangıçtan itibaren okunan byte sayısıdır.
  - read ve write fonksiyonları okunmakta olan dosya ofsetinden başlar ve fonksiyon sonuçlandığında bu değeri okunan ve yazılan byte kadar ilerletir.
  - Varsayılan olarak bir dosya açıldığında, O_APPEND tanımlanmadığı sürece, ofset 0 olarak atanır.
  - lseek fonksiyonu kullanılarak açılmış dosyanın ofset değerini tanımlayabiliriz.

```
#include <sys/types.h>

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

Dönüş: OK ise yeni dosya ofseti, hata ise -1.

I/O fonksiyonları

- lseek fonksiyonu
  - offset değerinin anlamı whence argümanının değerine göre farklılık gösterir.
    - SEEK_SET => Başlangıçtan itibaren byte sayısı
    - SEEK_CUR => Bulunulan noktadan byte sayısı
    - SEEK_END => Sondan itibaren byte sayısı
  - lseek kullanarak bulunduğumuz ofseti bulabiliriz.

```
off_t currpos;
currpos = lseek(fd, 0, SEEK_CUR);
```

  - Ayrıca fonksiyonu kullarak dosya içerisinde hareket edilip edilecemeyeceğini öğrenebiliriz.
    - Örneğin dosya belirteci bir pipe, FIFO veya sokete işaret ediyorsa, lseek fonksiyonu errno değerini **ESPIPE** yapar ve -1 döner.

I/O fonksiyonları

- lseek fonksiyonu
  - Dosya üzerinde hareket edebilmeyi konrol için seek.c programı
  - Seek.c

    $ ./seek < seek.c

    seek OK

    $ cat seek.c | ./seek

    cannot seek

I/O fonksiyonları

- lseek fonksiyonu
  - Normalde bir dosyanın ofseti negatif olmayan bir sayı olmalıdır. Ancak bazı araçlar negatif ofsetlere izin verir.
  - Normal bir dosya için ofset daima negatif olmayan bir tam sayıdır.
  - Negatif ofset mümkün olduğu için lseek fonksiyonunun dönüş değerinin -1 olduğu duruma dikkat etmeliyiz.
    - -1 değeri ofset olabilir.
  - Dosyanın ofseti dosyanın büyüklüğünde fazla olabilir.
    - Bu işlem yapılırsa dosyada delik oluşturulmuş olur.
    - Henüz dosyaya yazılmayan byte topluluğu 0 değerini alır.
    - Oluşturulan deliğin hafızada karşılığı olmayabilir.

I/O fonksiyonları

- lseek fonksiyonu
  - lseek fonksiyonu ile delik içeren bir program yazalım: hole.c
  - hole.c
  - od -c file.hole

I/O verimliliği

- Ekteki dosyada sonuçları gördüğümüz testte 103,316,352 byte büyüklüğünde bir dosyanın farklı önbellek büyüklüleri ile okunma süresi hesaplanmıştır.
- Bu dosya mycat programı ile okunmuş ve standart çıktı /dev /null dosyasına yönlendirilmiştir.
- Bu testler 4096 byte büyüklüğünde bloklara ayrılmış Linux ext2 dosya sisteminde yapılmıştır.
  - Sonuçlarda görüldüğü üzere en düşük sistem zamanı önbellek büyüklüğü 4096 byte olduğunda elde edilmiştir.
- Çoğu sistem ön-okuma denilen bir yöntem kullanarak performansı artırır.
  - Sonuçlar incelendiğinde önbellek büyüklüğü 128 KB'yi geçtiğinde ön-okumanın etkisi kalmamaktadır.

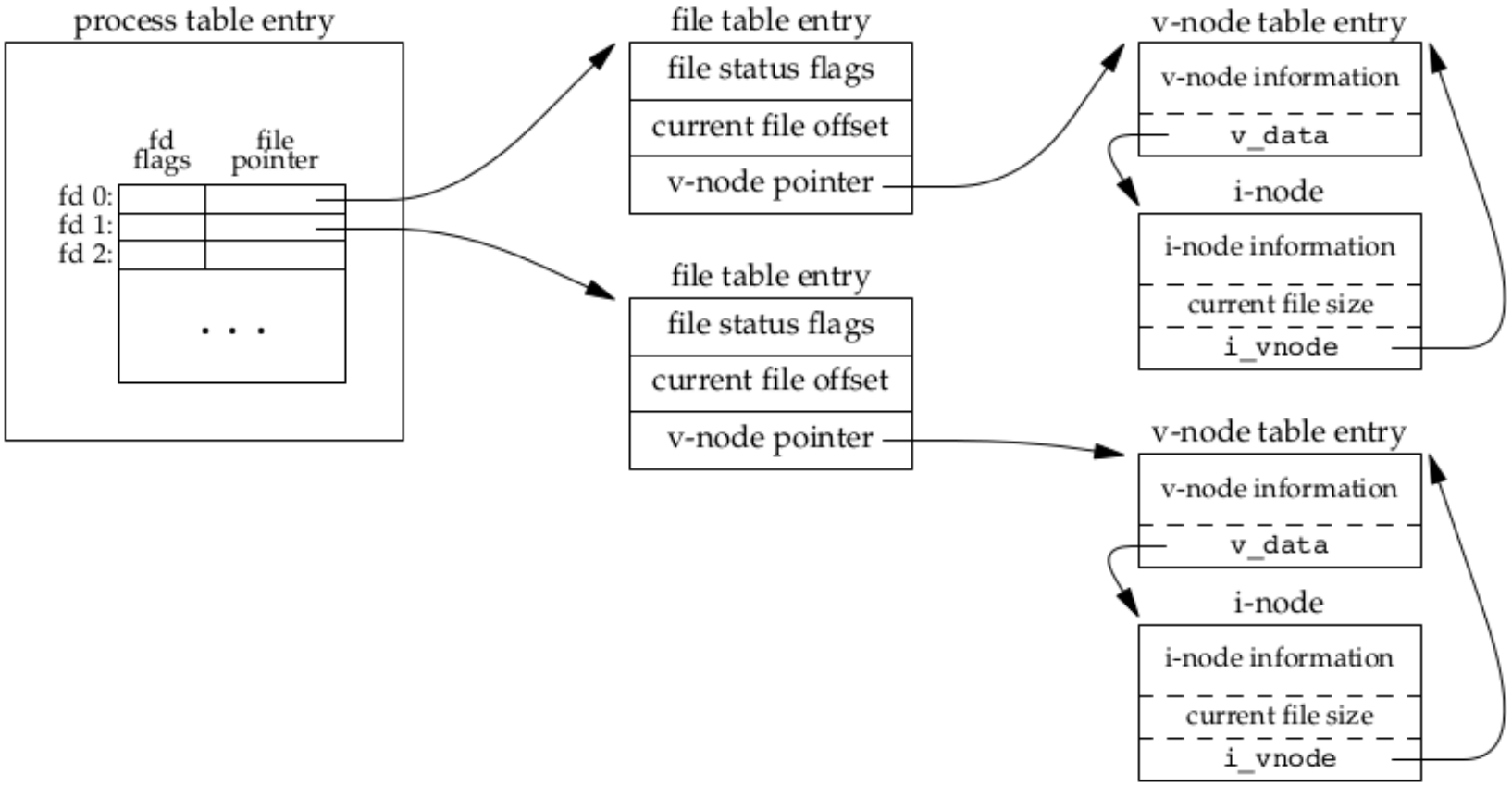
Dosya paylaşma

- Unix çoklu kullanıcı ve çoklu işlem içeren bir sistemdir.
  - Böyle bir sistemde birden fazla işlemin aynı dosya üzerinde çalışması olasıdır.
  - Dosya paylaşımının yapılma şeklini anlamak için girdi çıktı işlemleri için kernel tarafından kullanılan veri yapılarını incelemeliyiz.
- Kernel her açık dosya için üç farklı veri yapısı saklar.
  - Her işlem tablosu kaydında bir dosya belirteçleri tablosu bulunur. Bu tabloda aşağıdaki bilgiler bulunur:
    - Dosya belirteci bayrakları (örneğin FD_CLOEXEC)
    - Dosya tablosu kayıtlarına bir işaretçi
  - Kernel bir dosya tablosu saklar. Bu tabloda her kayıt için aşağıdaki bilgiler saklanır
    - Dosya durum bayrakları (O_APPEND, O_SYNC, O_RDONLY gibi).
    - Şuanki ofset değeri
    - Dosyanın v-node bilgisine bir işaretçi

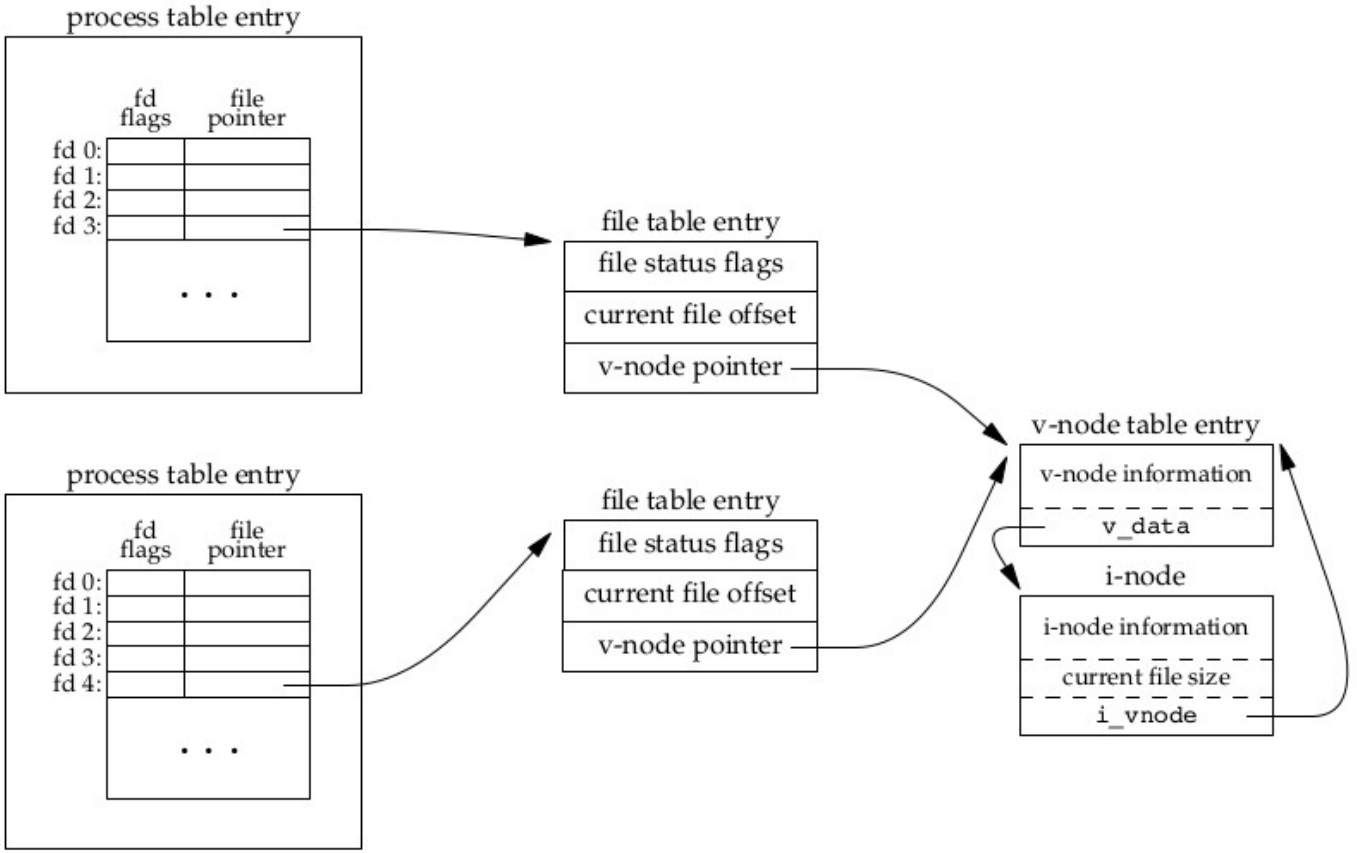
Dosya paylaşma

- Kernel her açık dosya için üç farklı veri yapısı saklar.
  - Her dosyanın bir v-node bilgisi bulunur. v-node içerisinde
    - v-node bilgisi (dosya tipi, dosya üzerinde çalışan fonksiyonlara işaretçiler gibi)
    - i-node bilgisi (dosyanın sahibi, dosya büyüklüğü, dosyanın diskte bulunduğu yeri gösteren işaretçiler)

Dosya paylaşımı



Dosya paylaşımı



Dosya paylaşımı

- Iki farklı işlem aynı dosyayı açtığında
  - Her write işleminden sonra, dosya tablosu kaydındaki dosya ofseti artırılır. Eğer şu anki ofset dosyanın boyutunu geçerse, i-node tablo kaydındaki dosya büyüklüğü değiştirilir.
  - Eğer dosya O_APPEND modunda açıldıysa, ilgili bayrak ayarlanır. Bu sayede her write öncesi i-node kaydından o anki büyüklük alınır ve dosyanın bittiği yere yazma işlemi yapılır.
  - lseek fonksiyonu sadece ilgili dosya tablosu kaydındaki ofset değerini değiştirir.
  - Dosyanın sonuna gidilmek istenirse yapılması gereken i-node kaydından dosyanın büyüklüğünün alınması ve bu değerin ofset olarak belirlenmesidir.

Atomik operasyonlar

- Önceki slaytlarda gördüğümüz üzere aynı anda birden fazla işlemin aynı dosyada değişiklik yapması mümkündür.
  - Bu durumda dosyanın içeriğinin tutarlı olması gerekir.
  - Bunun için belirtilen işlemlerin atomik olması gerekir.
    - Bir operasyon ya bütün adımlarıyla bölünmeden yapılıyor yada hiçbir adımı yapılmıyor ise bu operasyona atomik operasyon denir.
  - Bir işlemin bir dosyanın sonuna bir ekleme yaptığını düşünelim. Eski Unix versiyonlarında O_APPEND opsiyonu bulunmamaktaydı. Bu sebeple ilgili operasyon şu şekilde yapılmaktaydı.

```
if (lseek(fd, 0L, 2) < 0)
   err_sys("lseek error");
if (write(fd, buff, 100) != 100)
   err_sys("write error");
```

Atomik operasyonlar

- Bir işlemin bir dosyanın sonuna bir ekleme yaptığını düşünelim.
  - Tam bu işlem esnasında başka bir işlem aynı şekilde yazmak isterse.
    - Önemli bir sorun oluşur
  - Bu sorunun nedeni dosyanın sonuna gitme ve yazma işlemi iki ayrı operasyon olarak yapılmasıdır.
  - İki farklı fonksiyon ile yapılan işlemler atomik sayılmaz. Çünkü bu iki işlem arasında kernel çalışmakta olan işlemi durdurup başka bir işlemi çalıştırabilir.
- Unix sistemi bu sorunu çözmek için atomik olarak bu işlemlerin yapılmasına olanak verir.
  - Bunun için O_APPEND bayrağı ayarlanmalıdır.
    - Bu yöntem ile önce lseek yapmamız gerekmez.

Atomik operasyonlar

#include <unistd.h>

```
ssize_t pread(int filedes, void *buf, size_t nbytes, off_t offset);
```

Dönüş: okunan byte sayısı, dosya sonuysa 0, hata ise -1

```
ssize_t pwrite(int filedes, const void *buf, size_t nbytes, off_t offset);
```

Dönüş: yazılan byte sayısı, hata ise -1.

- pread veya pwrite kullanılırsa bu operasyonları bölmek mümkün değildir.
- İşlem bitmeden dosya ofset değeri değişmez.

Atomik operasyonlar

- Dosya oluşturma

```
if ((fd = open(pathname, O_WRONLY)) < 0) {
    if (errno == ENOENT) {
        if ((fd = creat(pathname, mode)) < 0)
            err_sys("creat error");
        } else {
            err_sys("open error");
        }
    }
```

- Bu yöntem yerine atomik operasyon kullanılmalıdır.
- `open(pathname, O_WRONLY | O_CREAT | O_TRUNC, mode);`