

# 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 genellikle 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
- Direkt 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 değeri 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>
```

```
ssize_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 offset değerinden başlayarak okur, okuma işlemi tamamladıktan sonra offseti 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 kullanarak dosya içerisinde hareket edilip edilececeğini 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 kontrol için seek.c programı
  - Seek.c

```
$ ./seek < seek.c
seek OK
$ cat seek.c | ./seek
cannot seek
```