Sistem Programlama

Ders 5

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

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 itibareb 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.

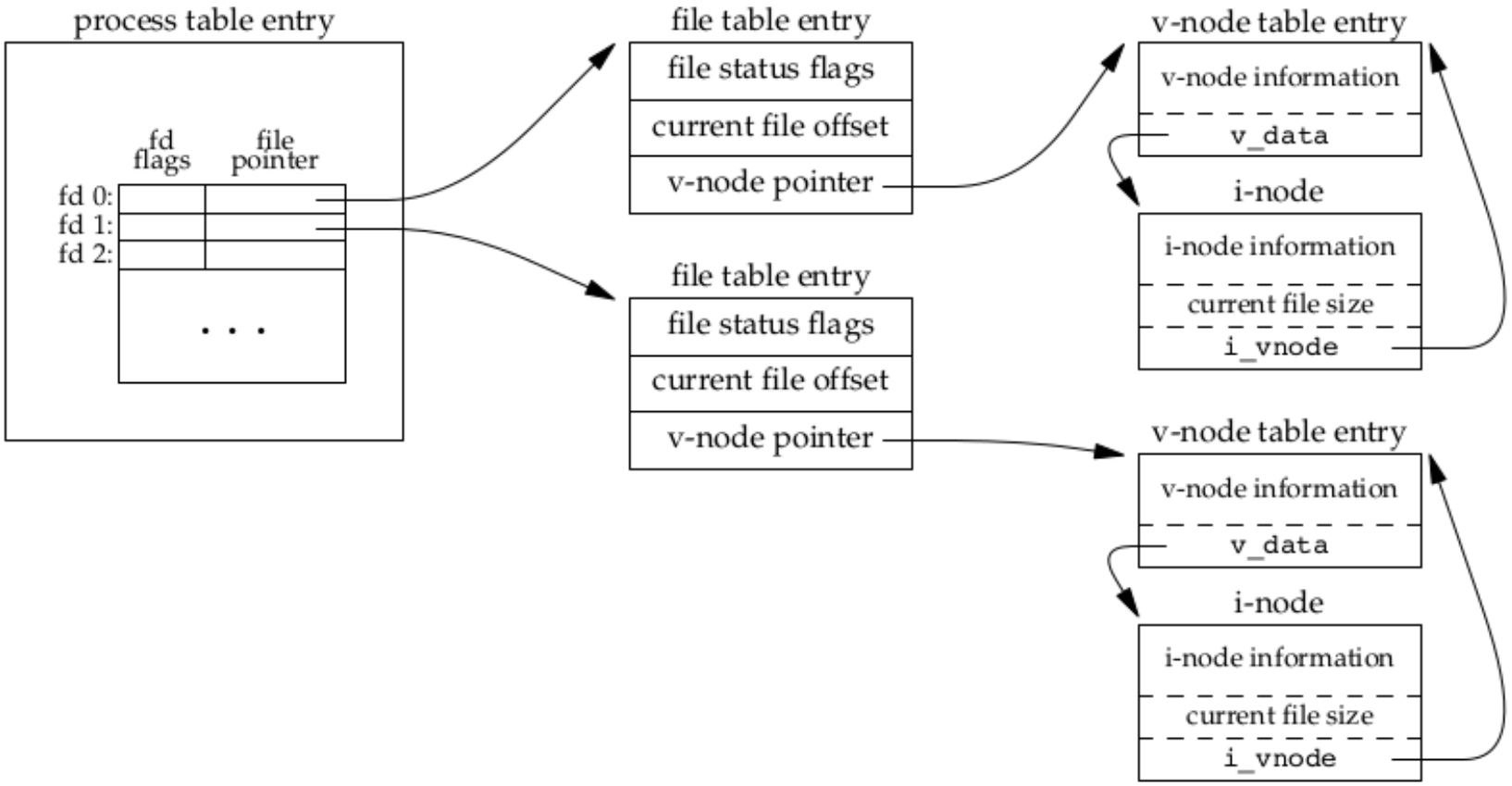
Dosya paylaşma

- 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

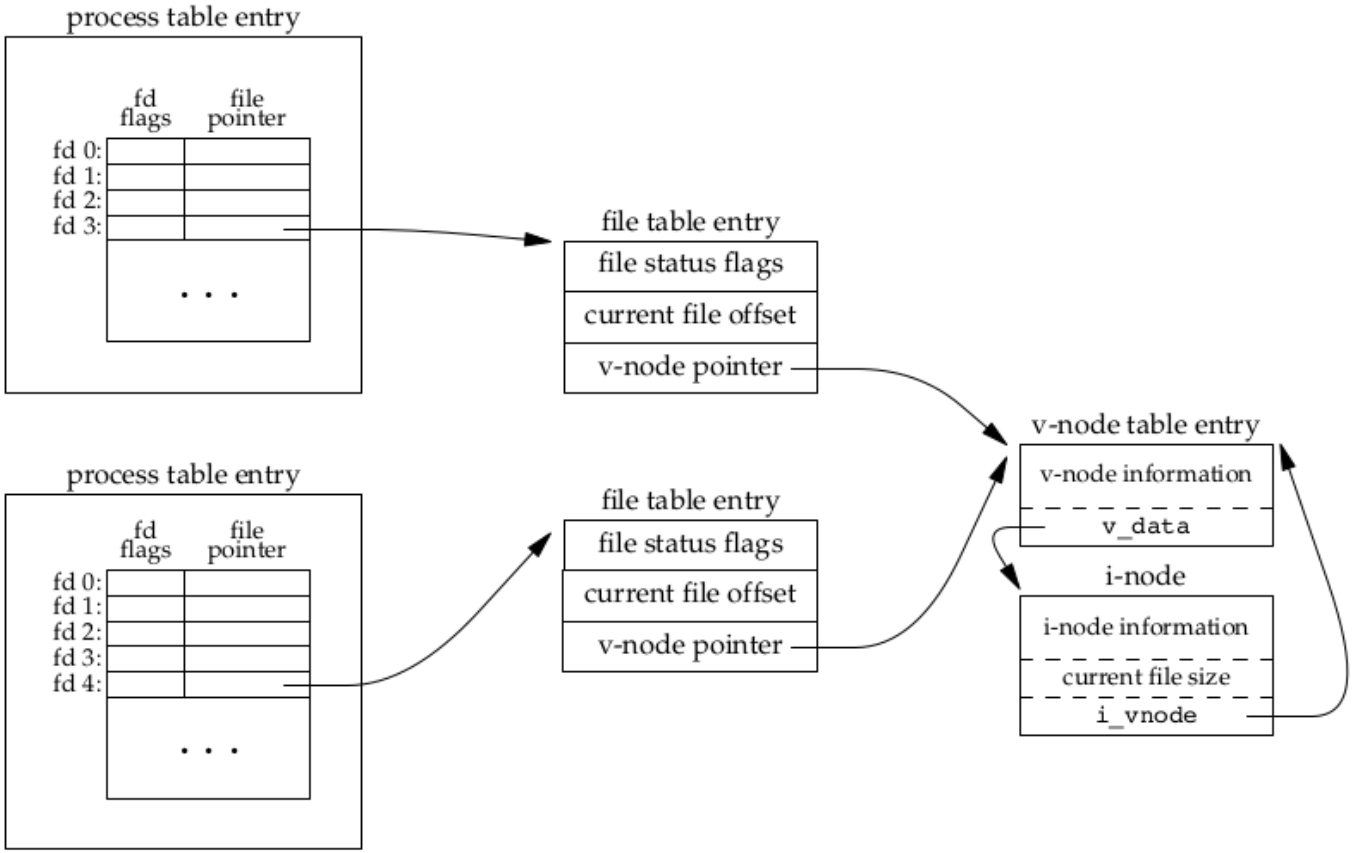
Dosya paylaşma

- 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 dosya 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 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);`