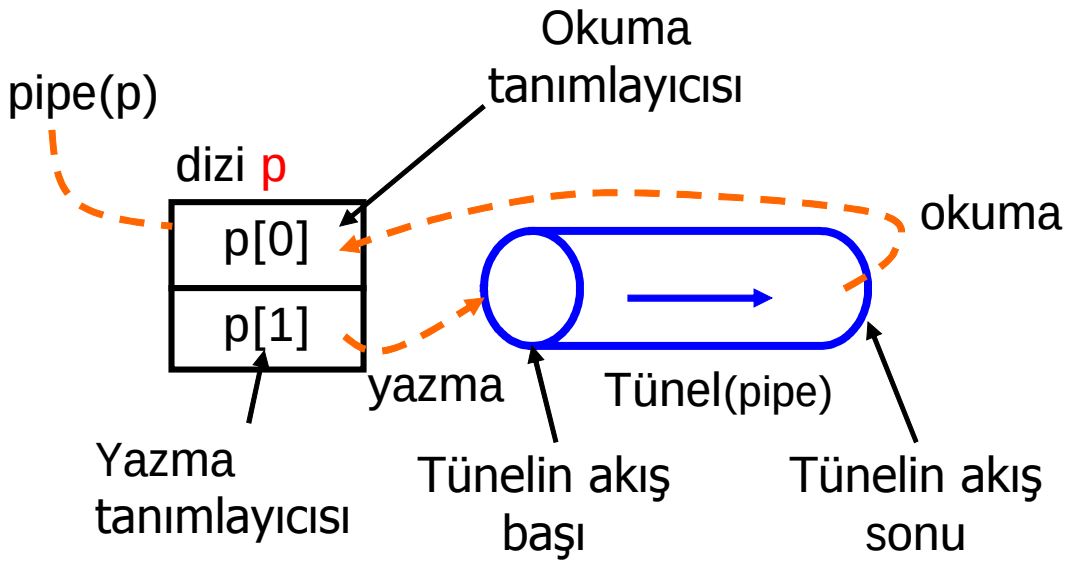
İsimsiz tüneller (Unnamed pipes)

Yaratma: **pipe()** sistem çağrısı kullanılarak

```
int sonuç;
int  p[2] ;                 /* tünel tanımlayıcı dizisi  */
sonuç = pipe(p) ;           /* tünel yaratma */
/* p bir dizi(array) işaretleyicidir(pointer)*/
```

sonuç= { 0, başarılı
-1, başarısız

p[0], p[1] tanımlanan değerler olur



Amaç:

İki ilişkili(related) proses arasında tek yönlü iletişim sağlamaktır.

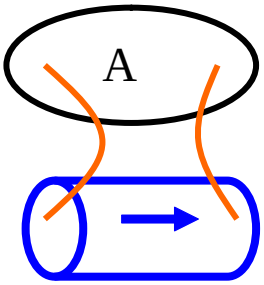
Mevcut sistem çağrıları:

*write	*close
*read	*dup

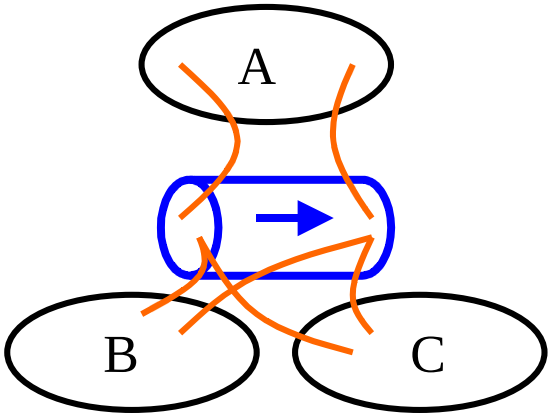
kullanılmayanlar!!!
open()
creat()

İki alt proses B ve C arasında tünel kullanılarak yaratılan iletişimin aşamaları (Tipik olarak)

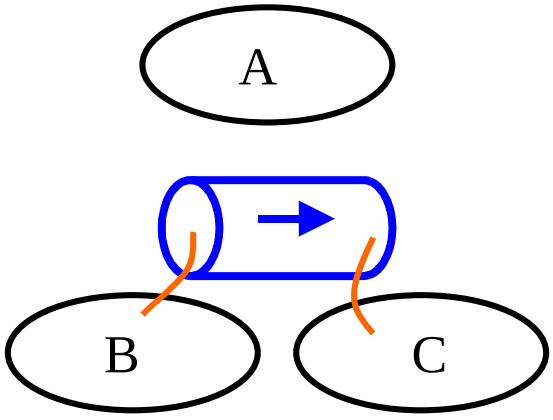
Aşama 1: Proses A bir tünel yaratır (her uç için tanımlayıcılar belirlenir).



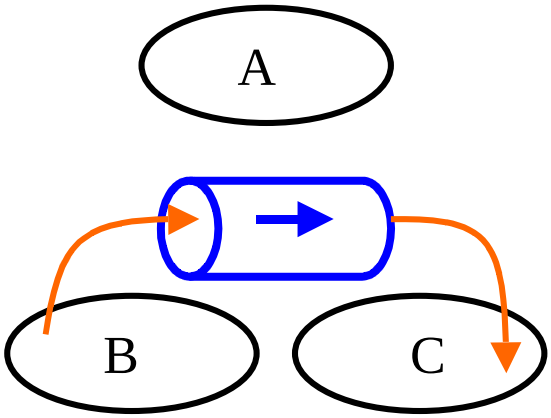
tünel



Aşama 2:
Proses A alt proses B ve C yi yaratmak için iki kez fork yapar. B ve C fork tan sonra tünelin her iki tanımlayıcısını da devralır. İki tanımlayıcı uç da açıktır.



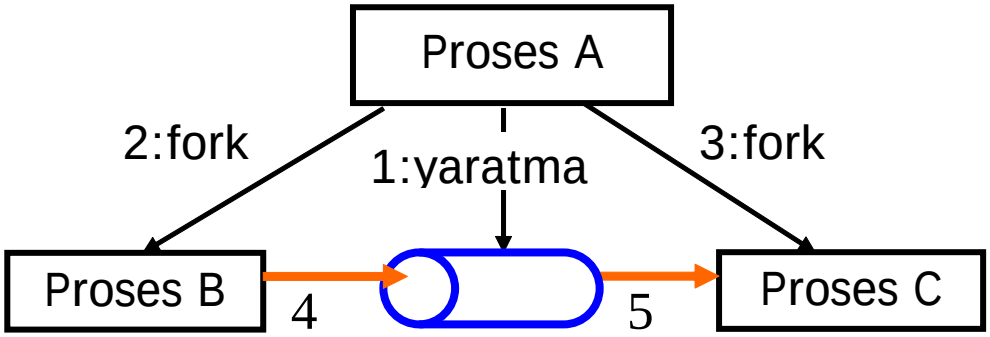
Aşama 3:
Her proses ihtiyacı olmadığı(kullanmayacağı) ucu veya uçları kapatır.



Aşama 4: B ve C prosesleri tünel kullanarak veri değişimi gerçekleştirir.

write(pfd[1], ...) ; read(pfd[0], ...) ;

İlişkili prosesler tarafından isimsiz tünellerin yaratılmasının ve kullanımının genel şeması



Oluş sırası 1, 2, 3, 4, 5 dir.

4: tünele yazma
5: tünelden okuma

herhangi bir sırada olabilir.

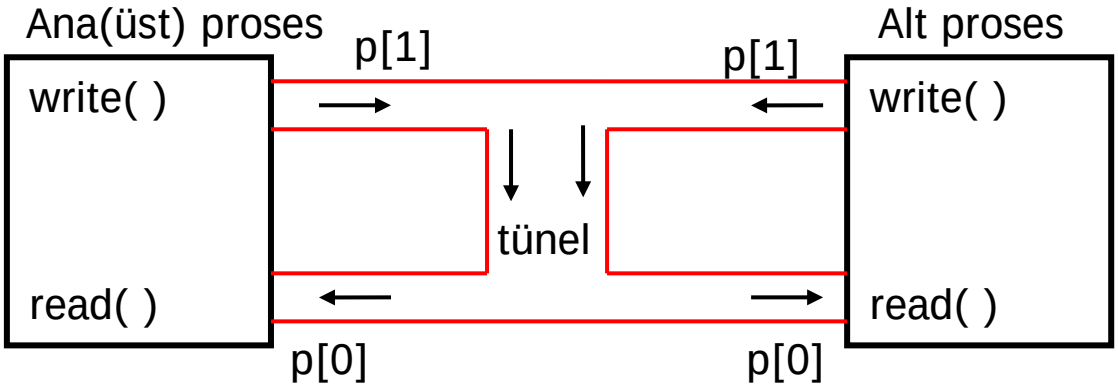
Tünellerin Senkronizasyon özellikleri

Yazarken : Eğer yazan proses **dolu** bir tünele yazmaya çalışıyorsa, tünelin okuma ucu açıksa, yazan proses **bloke** olur (bu durum okuyan proses tünelden veri alana kadar devam eder).

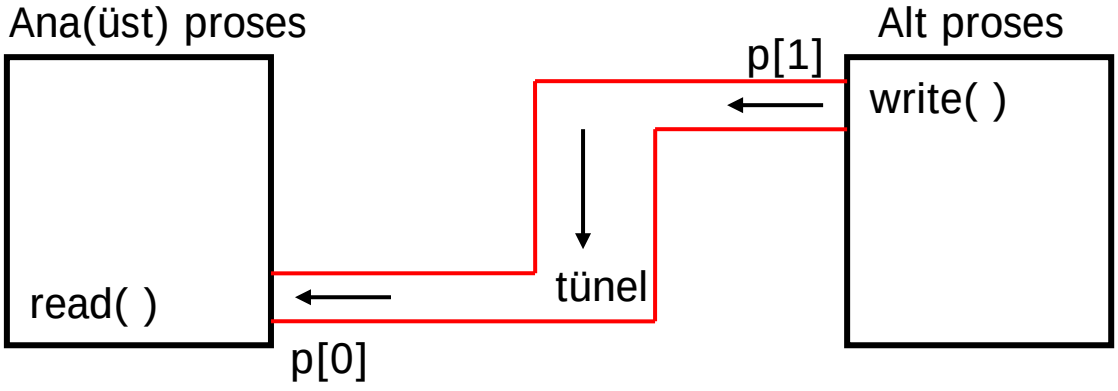
Okurken : Eğer okuyan proses **boş** bir tünelden okumaya çalışıyorsa, okuyan proses **bloke** olur(eğer tünelin yazma ucu açıksa) veya **0** dönderir (eğer tünelin yazma ucu kapalıysa).

Bir üst ve onun alt prosesi arasında isimsiz tüneller kullanarak iletişim sağlama

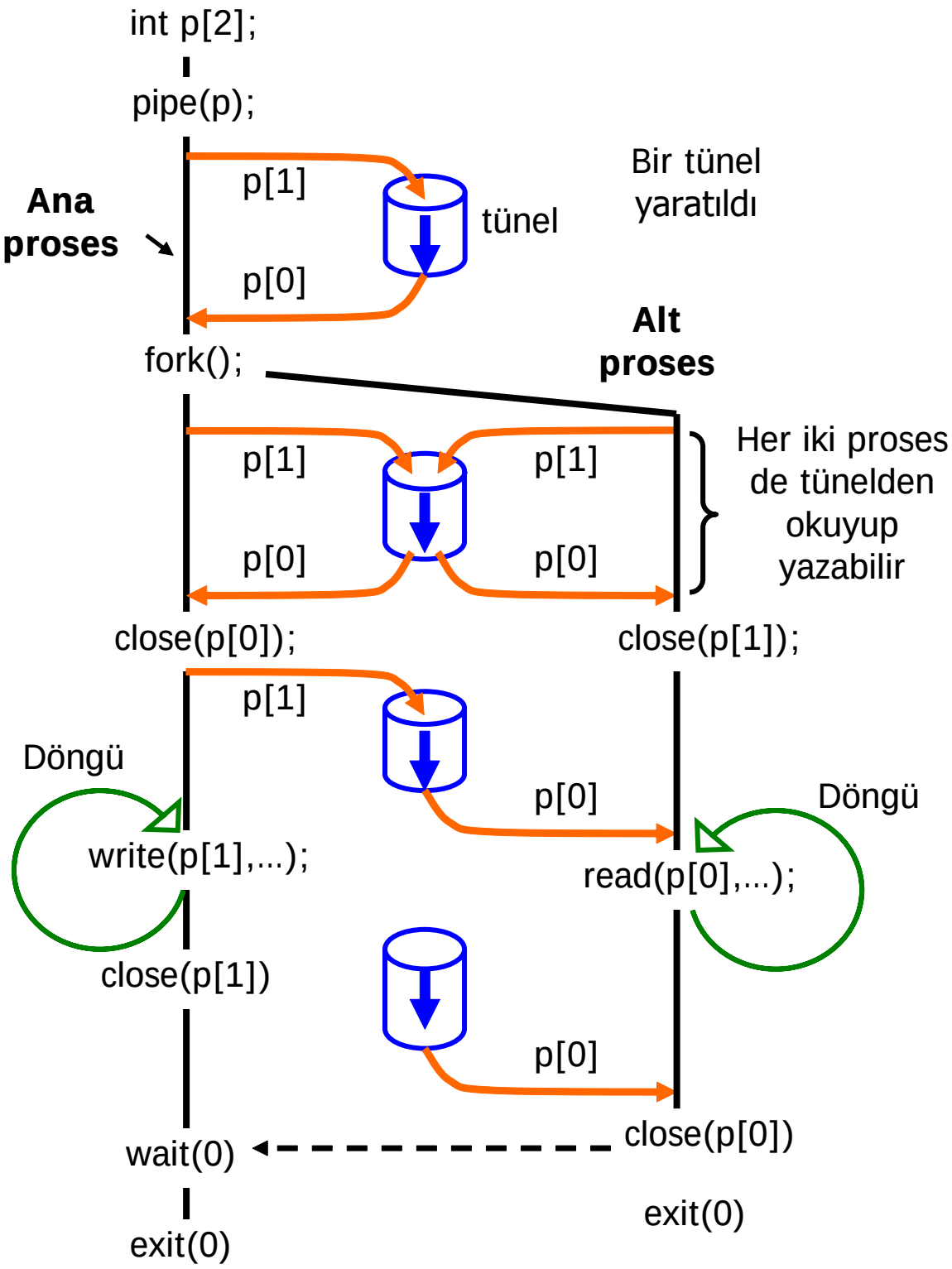
1. Tünelin her iki ucu da iki proses için açılır (pipe(p) sistem çağrısı kullanılır)



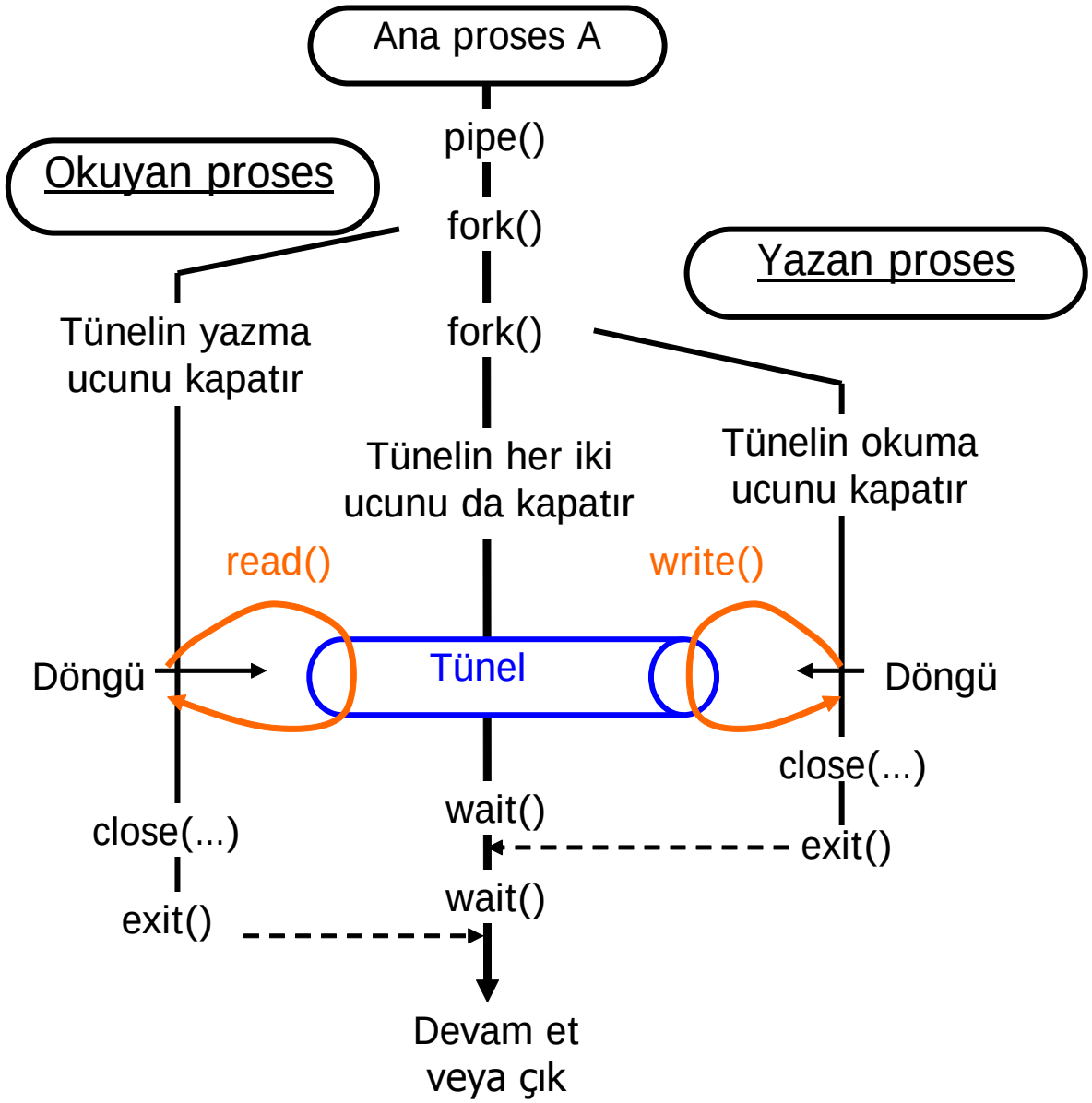
2. Her iki prosesin de sadece bir açık ucu vardır.



İki proses(ana ve alt) arasında isimsiz tünelin kullanımı

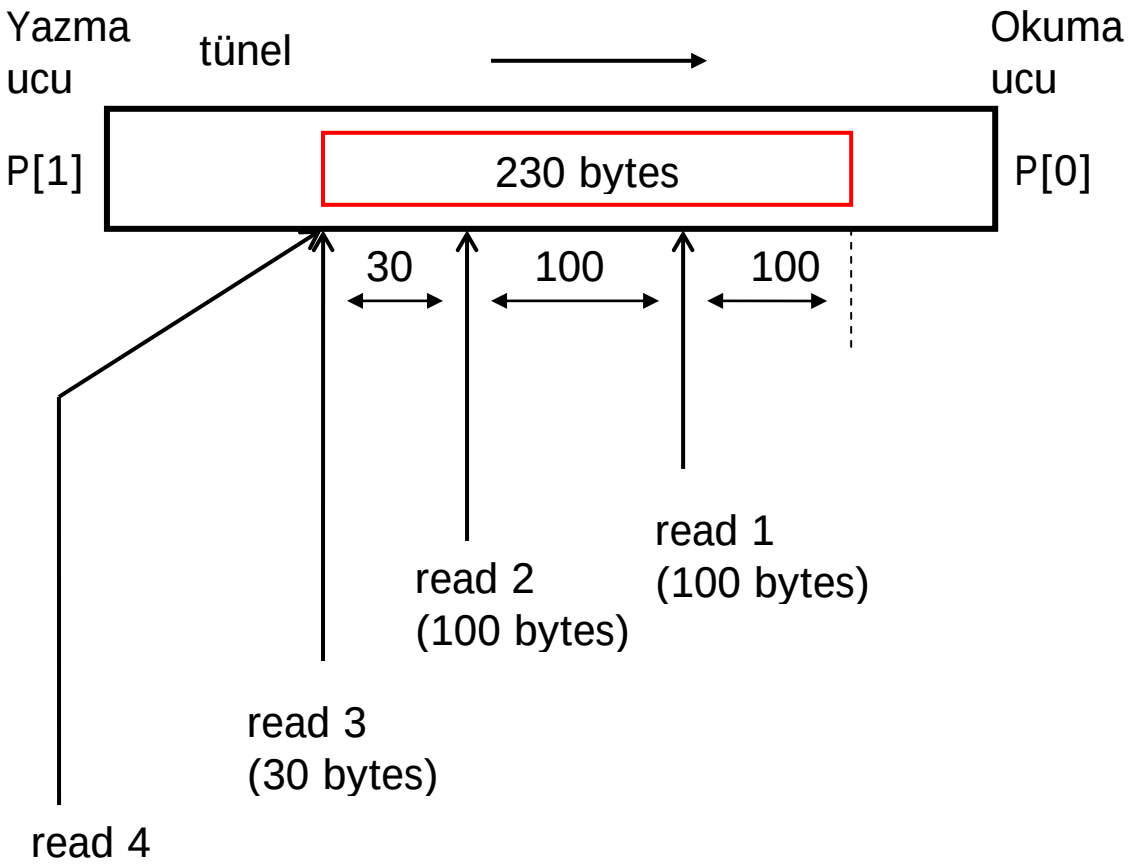


İki alt proses arasında isimsiz bir tünelin genel kullanım şeması



Tünelden okumaya bir örnek

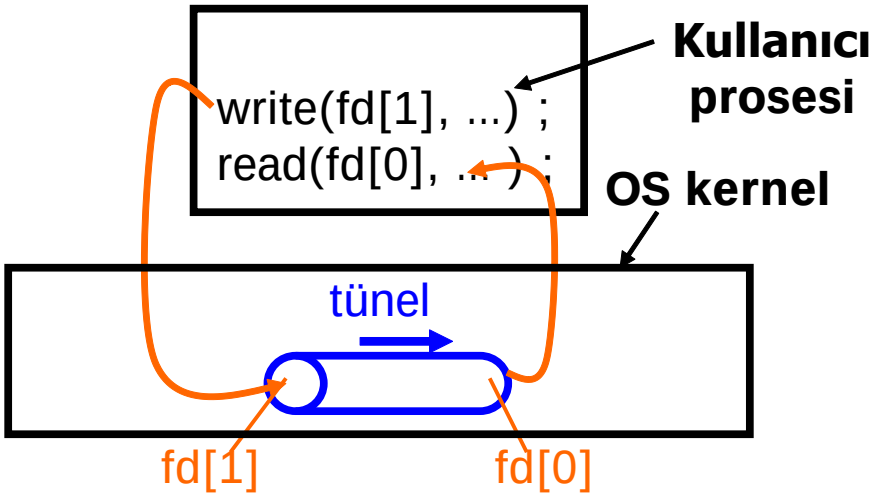
```
char buf[100];
int okunan;
...
while((okunan = read(p[0], buf, 100)>0)

        {/* buf daki veriyi kullan */}
```



(yeni veri yazılana kadar bloke olur) ← tünelin yazma ucu açıksa

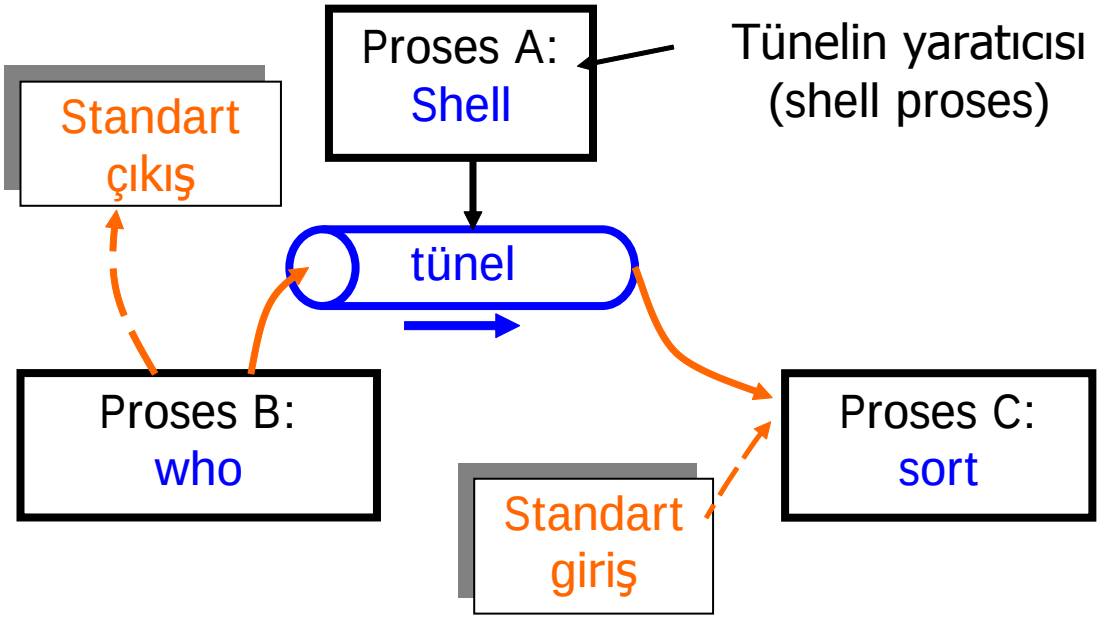
yada sıfır döndürür) ← tünelin yazma ucu kapalıysa

Tek proseste tünelin kullanıma örnek




```
#include <stdio.h>

main()
{  int  n, fd[2]  ;

   char  buf[128] ;

   if ( pipe(fd) < 0 )  {  /*  hata mesajı yazdır ve çık  */
}

   if ( write(fd[1], “Kıbrıs\n”, 7)  != 7 )
         {  /* hata mesajı yazdır ve çık */   }

   if ( (n = read(fd[0], buf, sizeof(buf))) <= 0  )
         {  /* hata mesajı yazdır ve çık */   }

   write(1, buf, n) ;/* fd = 1 standart  çıktı (stdout) için  */

   exit(0)  ;                    /*  normal çıkış  */
}
```

Detaylı bir örnek: program şeması



Amaç: **who | sort**

Proseslerin yürütümü:

who ve ***sort*** ,

who nun çıktısı ***sort*** a girdi olur.

"***who***" nun standard çıktısı ile ***tünelin yazma*** ucunu, "***sort***" un standard girdisi ile ***tünelin okuma*** ucunu bağdaştırmamız(birleştirmemiz) gerekiyor.

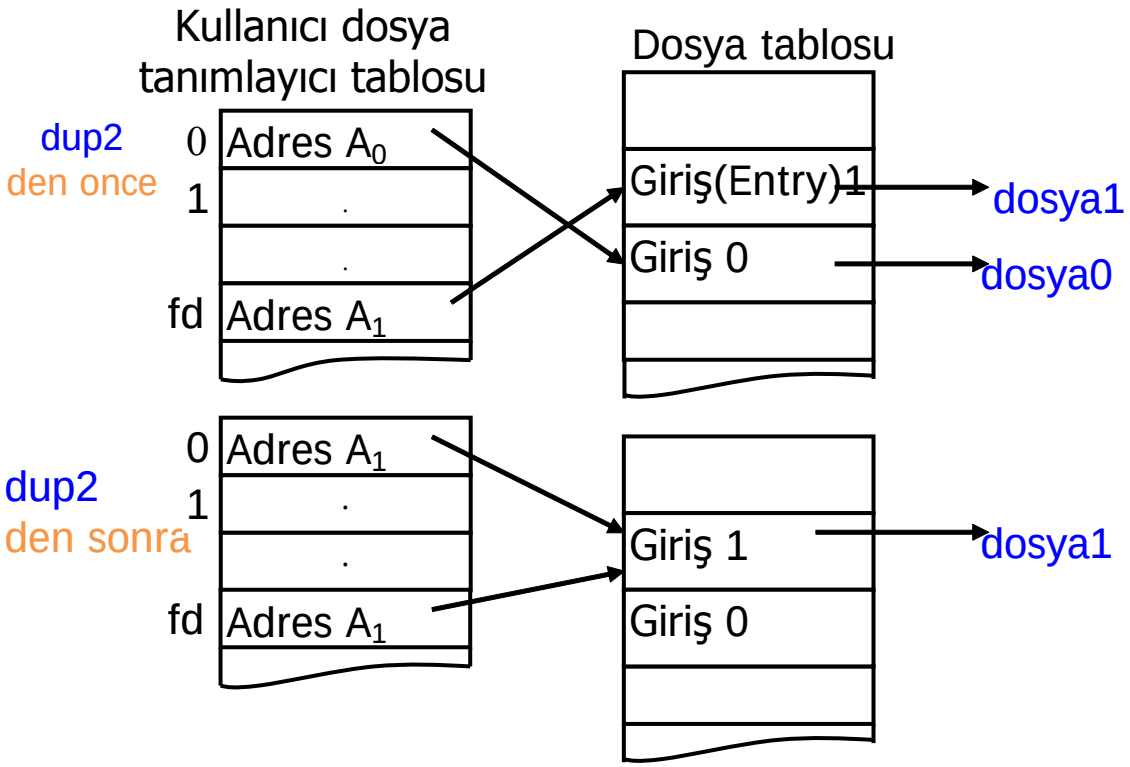
Bu durum **girdi/çıktıları** yönlendirmedir.

Dosya tanımlayıcılarını kapatma ve çoğaltma

dup2(fd, 0) ; /* **0** tanımlayıcısını kapatarak **fd** dosya tanımlayıcısını **0** dosya tanımlayıcısı olarak çoğaltır*/

Sonuç: **fd, 0** olarak çoğaltılmıştır.
Standart giriş fd den okunur.

Kütüphane fonksiyonu



0 dosya tanımlayıcı kullanıldığı zaman, **dosya1** kastedilmektedir.

Bir dosya tanımlayıcısını **dup2()** ile çoğaltma

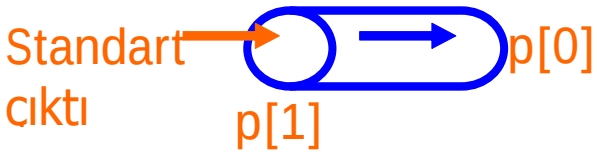
```
dup2(fd1, fd2) ;    /* fd2 yi kapatır. Şimdi
                       fd2 de fd1 in gösterdiği
                       dosyayı gösterir */
```

Örnekler:

a)

```
int p[2] ;
pipe(p) ;
```


```
dup2(p[1], 1) ;
/* standart çıkının ekrana gitmesi yerine tünele
gitmesi sağlanır */
```

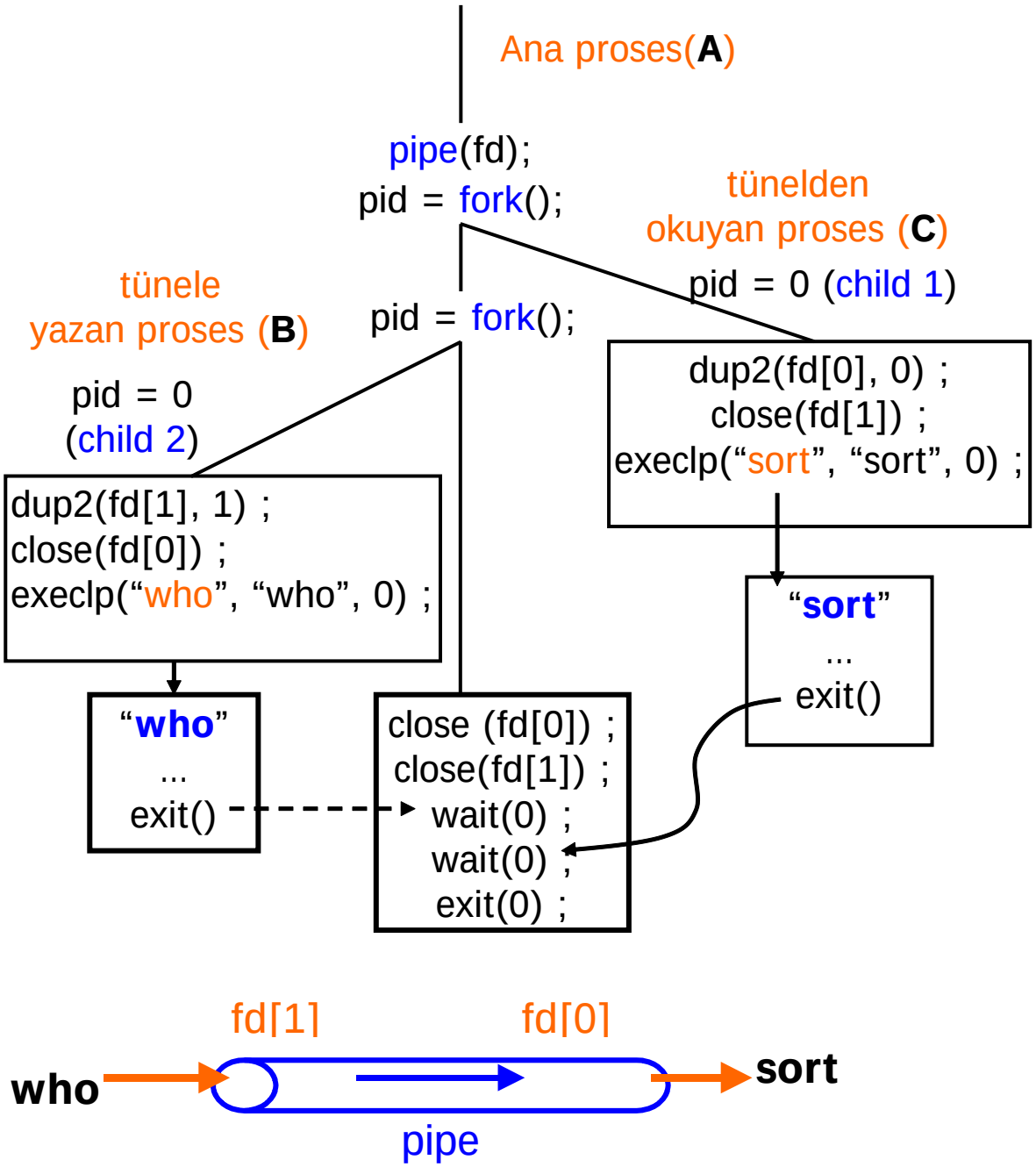


b)

```
int p[2] ;
pipe(p) ;
```


```
dup2(p[0], 0) ;
/*standart girdinin klavyeden gelmesi yerine
tünelden gelmesi sağlanır */
```

"who" ve "sort" u tünel aracılığıyla birbirine bağlayan programın yapısı



```
#include <stdio.h>
#include <unistd.h>
main()
{  int  fd[2] ;  int  pid  ;
   pipe(fd) ;                    /* Proses A bir pipe yaratır */
   pid = fork() ;          /* Proses A proses C yi yaratır */
   if ( pid == 0)                  /* Proses C burda başlar  */
    { dup2( fd[0], 0) ;          /* C de standart girdiyi
                                    pipe ın alt ucuna bağlar */
      close(fd[1]) ; /* Üst (yazma) uç C  için kapatılır */
       execlp("sort", "sort", 0 ) ; /*  "sort"u çalıştır  */
    }
                        /* proses A devam eder  */
   pid = fork() ;           /* Proses A proses B  yi yaratır*/
   if  (pid == 0 )                 /* Proses B burda başlar  */
     {  dup2(fd[1], 1) ;        /* B de standart çıktıyı
                                   pipe ın üst ucuna bağlar  */
        close ( fd[0]) ; /* Alt (okuma) uç B  için kapatılır*/
        execlp("who", "who", 0) ;  /*  "who" yu çalıştır ,
                                   0 – argümanların sonu */
     }
                      /* Proses A devam eder  */
  close ( fd[0]) ;       /* pipe A için kapatılır */
  close(fd[1]) ;
  wait(0) ;            /* B ve C nin bitmesi için bekler  */
  wait(0) ;
}
```