

Yaratmak için
pipe() sistem
çağrısı
kullanılır

İki yöntemle yaratılır:

1. **mknod()** sistem çağrısı kullanılarak veya **mkfifo()** kütüphane fonksiyonu kullanılarak
2. **mknod** shell komutu kullanılarak(klavyeden girilir).

Klavyeden girerek isimli bir tünel yaratmaya örnek:

%**mknod benimtünelim p** ← "benimtünelim" ismiyle yaratılır

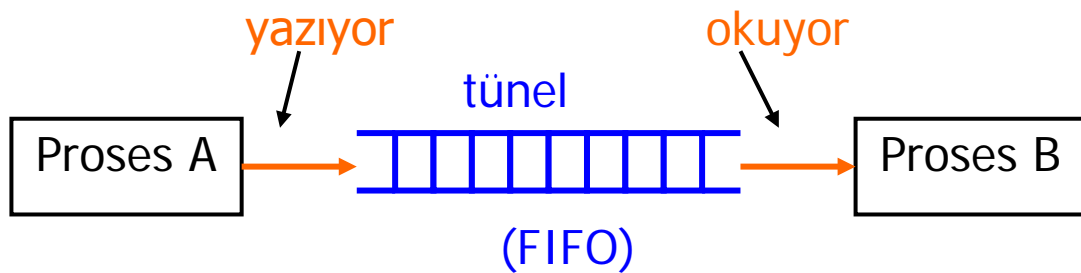
%**ls -l**

...

prw-r--r-- . . . benimtünelim

...

UNIX de isimli tüneller (named pipes)



İsimli tüneller için sistem çağrıları:

mknod() tünel yaratmak için

open()

read()

write()

close()

Örnek:

```
int a ;
```

```
. . .
```

```
a = mknod(tünel_ismi, S_IFIFO | 0666, 0) ;
```

```
if ( a < 0 ) { /* Hata kontrolü */ }
```

Örnekler

```
a = mknod("/tmp/benitünelim", S_IFIFO|0666, 0);
```

```
if (a<0) { perror("mknod"); exit(1); }
```

```
b = mkfifo("/tmp/benimtünelim", 0666);
```

```
if (b<0) { perror("mkfifo"); exit(1); }
```

```
fd = open("/tmp/benimtünelim", O_WRONLY);
```

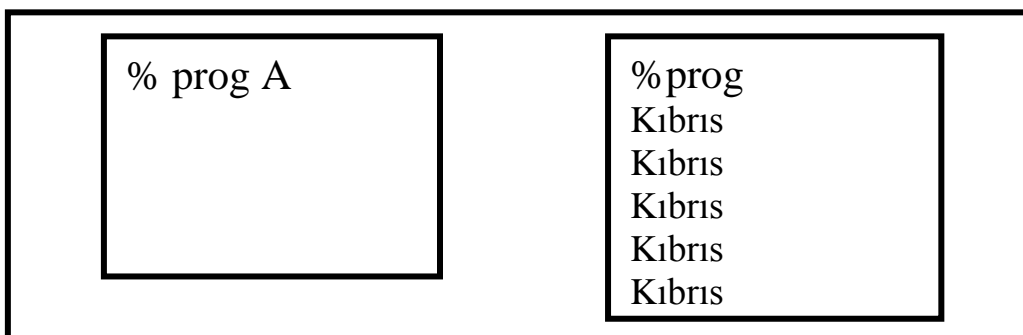
```
if (fd<0) { perror("open"); exit(1); }
```

İki proses için bir program:

- Proses A isimli tünel yaratır ve/veya tünele yazar
- Proses B tünel yaratır ve/veya tünelden okur

```
#include <fcntl.h>
#include <sys/stat.h>
char text[] = "Kıbrıs" ;
main(argc, argv)
int argc ; char *argv[] ;
{ int fd ; char buf[100] ; int i;
/*bir isimli tünel yarat */
    mknod("/tmp/abc", S_IFIFO | 0666, 0) ; /* sadece bir
                                           proses yaratır */
    if (argc == 2 ) fd = open("/tmp/abc", O_WRONLY) ;
    else             fd = open("/tmp/abc", O_RDONLY) ;
    for ( i=1; i<=5; ++i; )
    {
        if ( argc == 2 ) write(fd, text, 7 ) ;
        else
            {read(fd, buf, 8 ) ; write(1, buf, 8);}
    }
}
```

Ekrandaki görüntü

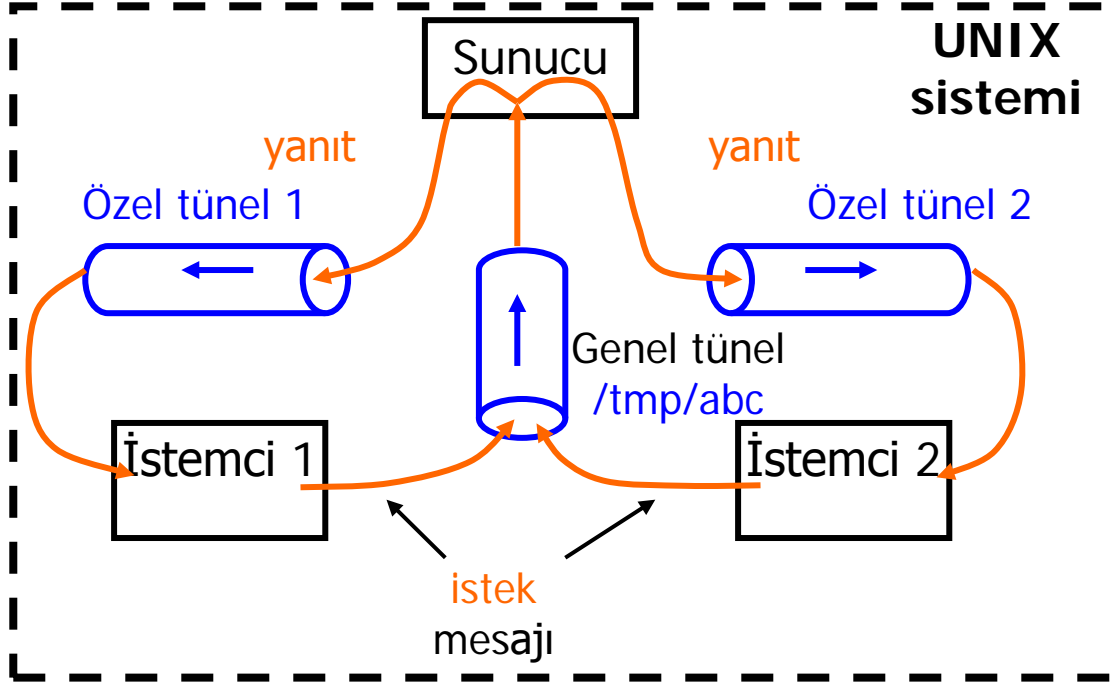


Programın çalıştırılması:

%<prog_ismi> <yapay_parametre> &
%<prog_ismi>

←proses A için
←proses B için

İsimli tüneller kullanarak sunucu/istemci
(client/server) sistemi oluşturulması



İstemci program parçası

```
#include ...
```

```
struct mesaj {
```

```
    char tünel_ismi[128] ;
```

```
    char istek[512] ;
```

```
} ;
```

/* sunuya mesaj */

Kullanımı:

%program_ismi özel_tünel

```
main ( argc, argv )
```

```
    int argc ; char *argv[] ;
```

```
{ int n, genel, ozel ;
```

```
    struct mesaj msj ;
```

```
    char buf[PIPE_BUF] ;
```

```
    msj.tünel_ismi = argv[1] ;
```

mknod(msj.tünel_ismi, ...); /*özel tünel yaratılıyor*/
genel = open("/tmp/abc", O_WRONLY) ;

```
msj.istek = ... ;
```

```
write( genel, &msj, sizeof(msj) ) ;
```

```
ozel = open(argv[1], O_RDONLY) ;
```

```
while ( (n = read(ozel, buf, PIPE_BUF) ) > 0 )
```

```
{ sunucudan gelen cevabı kullan }
```

```
close( ozel ) ; close( genel ) ; unlink( argv[1] ) ;
```

```
}
```

Gray, sayfa. 137-138

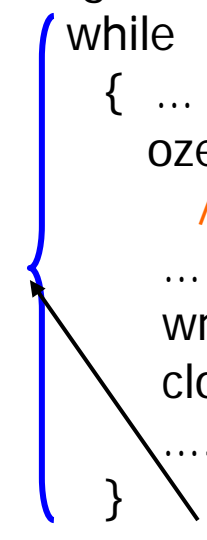
Sunucu program parçası

```
#include ...
struct mesaj { ... } ; /* istemci ninkile aynı */

main() /* komut satırı parametresi yok */
{ int n, genel, özel ;
  char buf[PIPE_BUF] ;
  struct mesaj msj ;

  mknod("tmp/abc", ... ) ; /* genel tüneli yarat */

  genel = open("tmp/abc", O_RDONLY ) ;
  while ( read(genel, &msj, sizeof(msj)) > 0 )
  { ...
    özel = open( msj.tüneli_ismi, O_WRONLY ) ;
    /* cevabı buf a koy ve gönder */
    ...
    write( özel, buf, n ) ;
    close( özel ) ;
    ....
  }
  ... /* isteği oku ve cevap gönder */
```



Tünellerin kullanımındaki sınırlamalar

1. Tünel tekyönlüdür.

2. Prosesler ilişkili olmalıdır (alt prosesler arasında veya ana ile alt proses arasında): Ana proses tüneli yaratır ve tanımlayıcılarını alt procese aktarır.

3. Tüneller geçicidir (Yani tüneli kullanan son proses olduğu sürece vardır).

2. ve 3. sınırlama isimli tünellerde yoktur.

İsimsiz tünellerle ilgili ayrıntılı örnek için bakınız

Curry,D., Unix Systems, sayfa. 358-363

İsimli tünellerle ilgili: sayfa. 364 - 366