

```
toros@kali: ~/Masaüstü
└─(toros@kali)-[~]
$ cd Masaüstü

└─(toros@kali)-[~/Masaüstü]
$ touch Lab5a.c

└─(toros@kali)-[~/Masaüstü]
$ nano Lab5a.c

└─(toros@kali)-[~/Masaüstü]
$ gcc Lab5a.c -o Lab5a

└─(toros@kali)-[~/Masaüstü]
$ ./Lab5a
adm:x:4:toros
dialout:x:20:toros
cdrom:x:24:toros
floppy:x:25:toros
sudo:x:27:toros
audio:x:29:pulse,toros
dip:x:30:toros
video:x:44:toros
plugdev:x:46:toros
netdev:x:109:toros
wireshark:x:119:toros
bluetooth:x:121:toros
lpadmin:x:126:toros
scanner:x:139:saned,toros
toros:x:1000:
kaboxer:x:144:toros

└─(toros@kali)-[~/Masaüstü]
```

```
(toros@kali)-[~/Masaüstü]  
$ getent group
```

```
root:x:0:  
daemon:x:1:  
bin:x:2:  
sys:x:3:  
adm:x:4:toros  
tty:x:5:  
disk:x:6:  
lp:x:7:  
mail:x:8:  
news:x:9:  
uucp:x:10:  
man:x:12:  
proxy:x:13:  
kmem:x:15:  
dialout:x:20:toros  
fax:x:21:  
voice:x:22:  
cdrom:x:24:toros  
floppy:x:25:toros  
tape:x:26:  
sudo:x:27:toros  
audio:x:29:pulse,toros  
dip:x:30:toros  
www-data:x:33:  
backup:x:34:  
operator:x:37:  
list:x:38:  
irc:x:39:  
src:x:40:  
gnats:x:41:  
shadow:x:42:  
utmp:x:43:  
video:x:44:toros  
sasl:x:45:  
plugdev:x:46:toros  
staff:x:50:  
games:x:60:  
users:x:100:  
nogroup:x:65534:  
systemd-journal:x:101:  
systemd-network:x:102:  
systemd-resolve:x:103:  
crontab:x:104:  
input:x:105:  
sgx:x:106:  
kvm:x:107:  
render:x:108:  
netdev:x:109:toros  
mysql:x:110:  
tss:x:111:  
systemd-timesync:x:112:  
redsocks:x:113:  
messagebus:x:114:  
kismet:x:115:  
_ssh:x:116:  
ssl-cert:x:117:postgres  
plocate:x:118:  
.
```

```
(toros@kali)-[~/Masaüstü]
$ getent group | grep toros
adm:x:4:toros
dialout:x:20:toros
cdrom:x:24:toros
floppy:x:25:toros
sudo:x:27:toros
audio:x:29:pulse,toros
dip:x:30:toros
video:x:44:toros
plugdev:x:46:toros
netdev:x:109:toros
wireshark:x:119:toros
bluetooth:x:121:toros
lpadmin:x:126:toros
scanner:x:139:saned,toros
toros:x:1000:
kaboxer:x:144:toros

(toros@kali)-[~/Masaüstü]
$
```

İki çıktı arasında belirgin bir fark gözlemlenmiyor. Her ikisi de "toros" kelimesini içeren grupları gösteriyor ve benzer sıralama ve grup düzenini sergiliyor. İki çıktı da aynı grupları içeriyor ve görünüşe göre aynı sıralamayı koruyor. "toros" kelimesini içeren gruplar, program çıktısı ile `getent group | grep toros` komutunun çıktısı arasında uyumlu bir şekilde yer alıyor.

```
toros:x:1000:
kaboxer:x:144:toros

La (toros@kali)-[~/Masaüstü]
$ ./Lab5a
adm:x:4:toros
dialout:x:20:toros
cdrom:x:24:toros
floppy:x:25:toros
L sudo:x:27:toros
audio:x:29:pulse,toros
dip:x:30:toros
video:x:44:toros
plugdev:x:46:toros
netdev:x:109:toros
wireshark:x:119:toros
bluetooth:x:121:toros
lpadmin:x:126:toros
scanner:x:139:saned,toros
toros:x:1000:
kaboxer:x:144:toros

(toros@kali)-[~/Masaüstü]
$
```

Sistemdeki bir grup genellikle bir grup adını, grubun numarasını (gid), ve o gruba üye olan kullanıcıları içerir. Örneğin, "root" grup adıyla, "x" grup numarasıyla ve "root" kullanıcısının üye olduğu bir grup bu çıktıda bulunmaktadır. Gruplar, sistemdeki kullanıcıları düzenlemek ve belirli izinlere sahip kullanıcıları gruplandırmak için kullanılır.

Kali kelimesini içeren gruplar genellikle "toros" kullanıcısının bu gruplara üye olduğunu gösterir. Örneğin, "toros" kullanıcısı "audio" grubuna üyedir. Bu, "toros" kullanıcısının sesle ilgili işlemleri gerçekleştirebilmesi için gerekli izinlere sahip olduğunu gösterir.

Bu çıktı, kullanıcıların sistemdeki grup üyeliklerini anlamak için kullanılır. Kullanıcılar, birçok farklı gruba üye olabilir ve bu gruplar, kullanıcının sahip olduğu izinleri ve yetenekleri belirler. Örneğin, belirli bir gruba üye olmak, o gruptaki dosyalara ve kaynaklara erişim yetkisi anlamına gelir.

Özel isimlendirilmiş gruplar, örneğin "_gvm", "_gophish", "mosquitto" gibi, genellikle belirli uygulamalar veya hizmetlerle ilişkilidir. Örneğin, "_gvm" grupları, Güvenlik Açığı Yönetimi (GVM) yazılımıyla ilgili olabilir. Bu tür gruplar, belirli bir hizmet veya uygulamanın kullanıcılarını yönetmek ve bu kullanıcılara özel izinleri sağlamak için oluşturulmuştur.

"getent" komutu, sistemdeki grup bilgilerini getirir. "grep" komutu ise bu grupları filtreler ve sadece "toros" kelimesini içeren grupları gösterir. Bu, sistem yöneticilerinin veya kullanıcıların belirli bir kelimeyi içeren grupları hızlıca bulmalarına yardımcı olur.

Bu bilgi çıktısı, sistemdeki kullanıcıların hangi gruplara üye olduğunu anlamak ve sistemdeki grupları denetlemek amacıyla kullanılabilir. Gruplar, dosya ve dizinlere erişim izinlerini yönetmek, ağ paylaşımlarını düzenlemek ve diğer sistem kaynaklarına erişimi kontrol etmek için önemli bir rol oynar. Bu bilgiler, genel sistem güvenliği ve kullanıcı yönetimi açısından kritik bir öneme sahiptir. Gruplar, belirli yetkilere sahip kullanıcıları bir araya getirerek kaynaklara güvenli ve organize bir şekilde erişmelerini sağlar. Bu nedenle, grup üyelikleri ve grup yapıları, sistem yöneticileri için önemli bir inceleme ve düzenleme alanıdır.

Kod başarıyla çalıştı ve ilk süreç belirtilen kelimeleri bir pipe yazdı. İkinci süreç ise bu kelimeleri okuyarak ekrana yazdırdı. İlk süreç tarafından yazılan kelimeler, ikinci süreç tarafından alınıp ekrana basıldı. Bu pipe iletişimi, süreçler arasında veri iletimini sağlar, paralel veya dağıtık hesaplamalarda kullanılır.

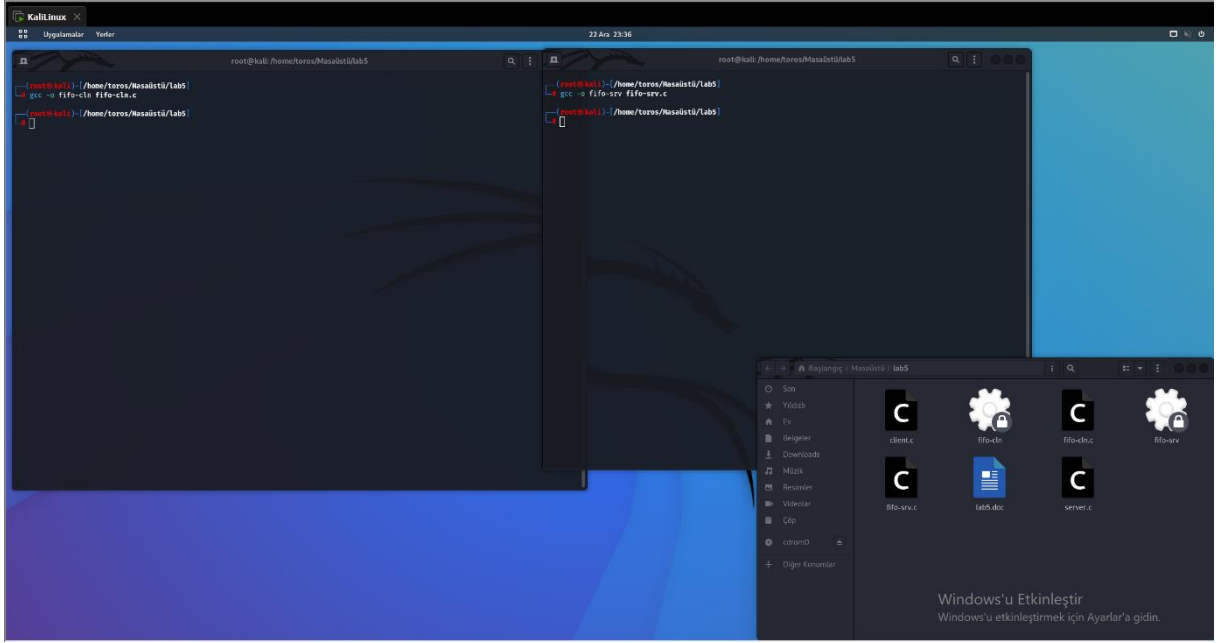
```
(toros@kali)-[~/Masaüstü]
$ nano Lab5n
(toros@kali)-[~/Masaüstü]
$ nano Lab5new.c
(toros@kali)-[~/Masaüstü]
$ gcc Lab5new.c -o Lab5new
(toros@kali)-[~/Masaüstü]
$ ./Lab5new
Message from child 1
Message from child 2
(toros@kali)-[~/Masaüstü]
$
```


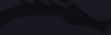
İlk iki çocuk süreci, bir pipe sırasıyla "Mesaj 1 çocuktan." ve "Mesaj 2 çocuktan." mesajlarını yazdı. Üçüncü çocuk süreci ise bu mesajları pipeden okuyarak ekrana yazdırdı. Gönderilen mesajların arasında herhangi bir bölünme olmadığı gözlemlendi, yani mesajlar birbiri ardına ve tamamen aktarıldı.

Bu örnek, pipe kullanarak çocuk süreçler arasında veri paylaşımını gösteriyor. Her çocuk sürecin bağımsız bir şekilde çalıştığı ve pipe'ın verileri doğru bir şekilde aktardığı görülüyor. Bu tür bir iletişim mekanizması, süreçler arasında veri iletimi sağlamak ve koordineli çalışmalarını sağlamak için sıkça kullanılır. Çocuk süreçler arasında böyle bir boru kullanarak iletişim, paralel işlemler ve veri paylaşımı konularında etkili bir yol sunar.

```
(toros@kali)-[~/Masaüstü]
$ nano Lab5echo.c
(toros@kali)-[~/Masaüstü]
$ gcc Lab5echo.c -o Lab5echo
(toros@kali)-[~/Masaüstü]
$ ./Lab5echo
this
is
a
message
from
sending
process
(toros@kali)-[~/Masaüstü]
$
```

Bölüm 2

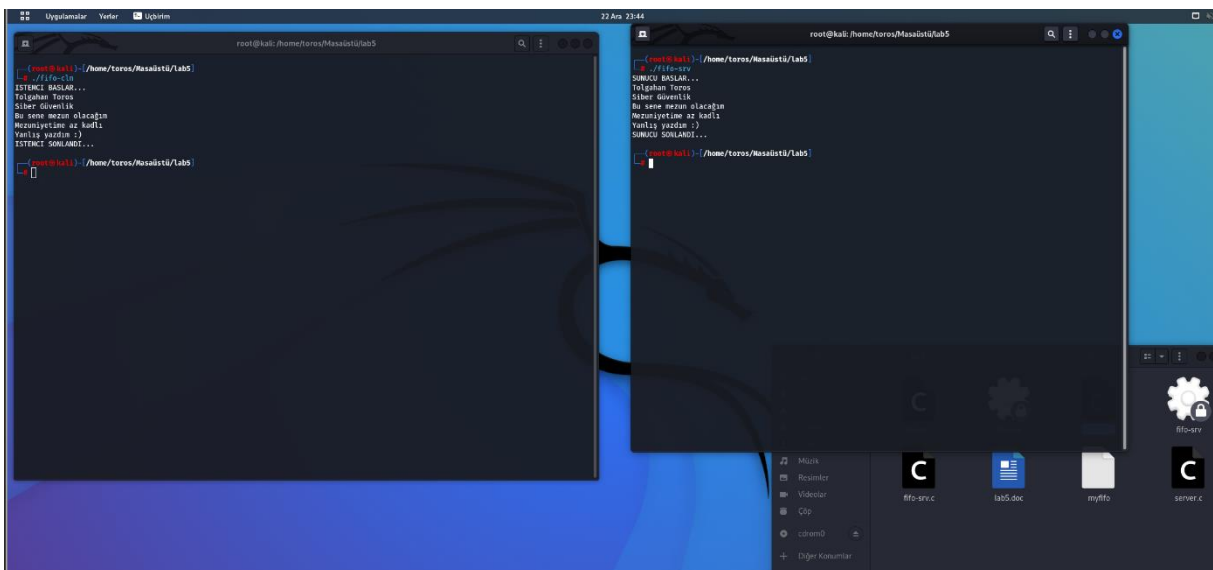
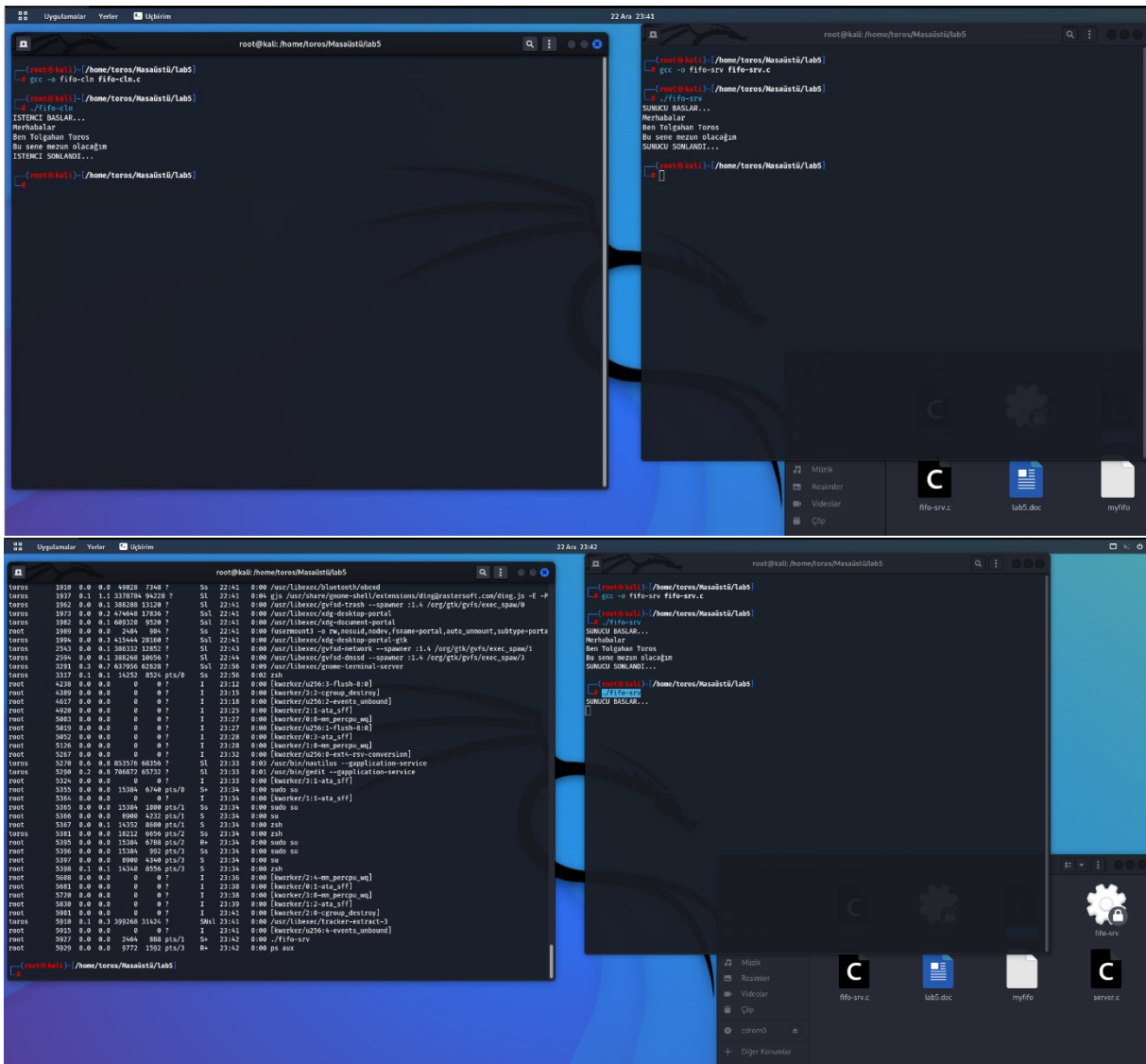


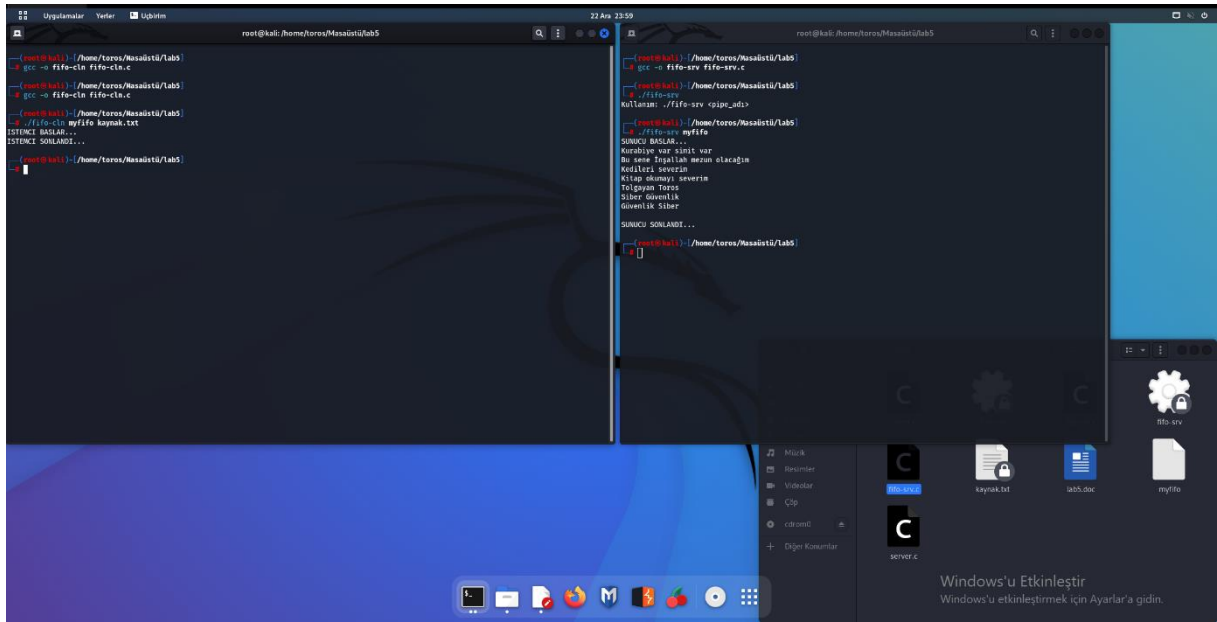
```
Aç  

fifo-srv.c

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <sys/types.h>
4 #include <sys/stat.h>
5 #include <fcntl.h>
6 #include <unistd.h>
7
8 #define FIFONAME "myfifo" /* Sunucudaki pipe ile aynı isim olacak */
9
10 int main(void) {
11     int n, fd;
12     char buf[1024];
13
14     printf("ISTEMCI BASLAR... \n");
15
16     /*
17      * Yazmak için mevcut pipe'ı aç.
18      * Sunucu zaten yaratmıştı.
19      */
20     if ((fd = open(FIFONAME, O_WRONLY)) < 0) {
21         perror("open problem in client");
22         exit(1);
23     }
24
25     /*
26      * Klavyeden metin oku
27      * ve bu metni pipe'a yaz. Metin Ctrl/d ile sonlandırılmalı
28      */
29     while ((n = read(0, buf, sizeof(buf))) > 0)
30         write(fd, buf, n);
31
32     printf("ISTEMCI SONLANDI... \n");
33     close(fd);
34     exit(0);
35 }
```

```
Uygulamalar Yerler gedit 22 Ara 23:38
Aç fifo-srv.c ~/Masaüstü/lab5
fifo-srv.c
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <sys/types.h>
4 #include <sys/stat.h>
5 #include <fcntl.h>
6 #include <unistd.h>
7
8 #define FIFONAME "myfifo" /* Burada pipe için özel bir ad verin */
9
10 int main(void) {
11     int n, fd;
12     char buf[1024]; /* Okuma ve yazma için kullanılacak buffer */
13
14     printf("SUNUCU BASLAR ... \n");
15
16     /* Önceden aynı isimde pipe varsa sil */
17     unlink(FIFONAME);
18
19     /* Okuma ve yazma haklarıyla ilgili pipe oluştur */
20     if (mkfifo(FIFONAME, 0666) < 0) {
21         perror("mkfifo problem in server");
22         exit(1);
23     }
24
25     /* Sadece emin olmak için yetkileri tekrar değiştirelim */
26     if (chmod(FIFONAME, 0666) < 0) {
27         perror("chmod problem in server");
28         exit(1);
29     }
30
31     /* Okuma için pipe aç */
32     if ((fd = open(FIFONAME, O_RDONLY)) < 0) {
33         perror("open problem in server");
34         exit(1);
35     }
36
37     /* Dosya sonuna kadar oku, ve okuduğunu ekrana yaz */
38     while ((n = read(fd, buf, sizeof(buf))) > 0)
39         write(1, buf, n);
40
41     close(fd);
42     printf("SUNUCU SONLANDI ... \n");
43     exit(0);
44 }
```



```
fifo-srv.c
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <sys/types.h>
4 #include <sys/stat.h>
5 #include <fcntl.h>
6 #include <unistd.h>
7
8 int main(int argc, char *argv[]) {
9     if (argc != 2) {
10         fprintf(stderr, "Kullanım: %s <pipe_adi>\n", argv[0]);
11         exit(1);
12     }
13
14     char *pipeName = argv[1];
15
16     int n, fd;
17     char buf[1024]; /* Okuma ve yazma için kullanılacak buffer */
18
19     printf("SUNUCU BASLAR...\n");
20
21     /* Önceden aynı isimde pipe varsa sil */
22     unlink(pipeName);
23
24     /* Okuma ve yazma haklarıyla ilgili pipe oluştur */
25     if (mkfifo(pipeName, 0666) < 0) {
26         perror("mkfifo problem in server");
27         exit(1);
28     }
29
30     /* Sadece emin olmak için yetkileri tekrar değiştirelim */
31     if (chmod(pipeName, 0666) < 0) {
32         perror("chmod problem in server");
33         exit(1);
34     }
35
36     /* Okuma için pipe aç */
37     if ((fd = open(pipeName, O_RDONLY)) < 0) {
38         perror("open problem in server");
39         exit(1);
40     }
41
42     /* Dosya sonuna kadar oku, ve okuduğunu ekrana yaz */
43     while ((n = read(fd, buf, sizeof(buf))) > 0)
44         write(1, buf, n);
45
46     close(fd);
47     printf("SUNUCU SONLANDI...\n");
48     exit(0);
49 }
```

```
fifo-srv.c
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <sys/types.h>
4 #include <sys/stat.h>
5 #include <fcntl.h>
6 #include <unistd.h>
7
8 #define FIFONAME "myfifo" /* Sunucudaki pipe ile aynı isim olacak */
9
10 int main(void) {
11     int n, fd;
12     char buf[1024];
13
14     printf("ISTEMCI BASLAR ... \n");
15
16     /*
17      * Yazmak için mevcut pipe'ı aç.
18      * Sunucu zaten yaratmıştı.
19      */
20     if ((fd = open(FIFONAME, O_WRONLY)) < 0) {
21         perror("open problem in client");
22         exit(1);
23     }
24
25     /* Kaynak dosyayı aç */
26     int sourceFile = open("kaynak.txt", O_RDONLY);
27     if (sourceFile < 0) {
28         perror("open problem in source file");
29         exit(1);
30     }
31
32     /*
33      * Dosyadan oku
34      * ve bu veriyi pipe'a yaz. Dosya sonuna gelinceye kadar devam et.
35      */
36     while ((n = read(sourceFile, buf, sizeof(buf))) > 0)
37         write(fd, buf, n);
38
39     printf("ISTEMCI SONLANDI ... \n");
40
41     /* Dosyaları kapat */
42     close(fd);
43     close(sourceFile);
44
45     exit(0);
46 }
```

The screenshot displays a Windows 10 desktop environment. Two terminal windows are open, both showing the same directory: `root@kali: /home/torun/Masaüstü/lab5`.

The left terminal window shows the execution of the client program:

```
root@kali: /home/torun/Masaüstü/lab5
# gcc -o client client.c
root@kali: /home/torun/Masaüstü/lab5
# ./client request_fifo response_fifo
İSTENİMLİ BAŞLADI...
SUNUCU: İstek alındı! Cevap: Merhaba Sunucu!
İSTENİM SONLANDI...
```

The right terminal window shows the execution of the server program:

```
root@kali: /home/torun/Masaüstü/lab5
# gcc -o server server.c
root@kali: /home/torun/Masaüstü/lab5
# ./server request_fifo response_fifo
SUNUCU BAŞLADI...
Sunucu: İstek alındı! Merhaba Sunucu!
Sunucu: İstek alındı! bye
```

The desktop background is a blue and purple abstract image. The taskbar at the bottom contains icons for the Start menu, File Explorer, Edge browser, and several application shortcuts. The system tray in the bottom right corner shows the date and time as 23 Arv 00:04.

1. **FIFO İsimleri:** **REQUEST_FIFO** ve **RESPONSE_FIFO** isimlerinde iki adet FIFO tanımlanmıştır. Sunucu, istemciden gelen istekleri **REQUEST_FIFO** üzerinden alır ve cevapları da **RESPONSE_FIFO** üzerinden gönderir.
2. **FIFO Oluşturma:** **mkfifo** fonksiyonu kullanılarak **REQUEST_FIFO** ve **RESPONSE_FIFO** boruları oluşturulur. Bu borular, iletişim için kullanılır.
3. **Dosya İşlemleri:** Sunucu, **REQUEST_FIFO**'yu okuma (**O_RDONLY**) ve **RESPONSE_FIFO**'yu yazma (**O_WRONLY**) modunda açar.
4. **İstekleri ve Cevapları İşleme:** Sonsuz bir döngü içinde sunucu, **REQUEST_FIFO**'dan gelen istekleri okur, bu isteklere cevap verir ve cevapları da **RESPONSE_FIFO**'ya yazar.
5. **FIFO Silme:** Sunucu, işini tamamladıktan sonra **REQUEST_FIFO** ve **RESPONSE_FIFO**'yu siler ve programı sonlandırır.

1. **FIFO İsimleri:** **REQUEST_FIFO** ve **RESPONSE_FIFO** adlarında iki FIFO tanımlanır. İstemci, sunucuya istekleri **REQUEST_FIFO** üzerinden gönderir ve sunucudan gelen cevapları **RESPONSE_FIFO** üzerinden okur.
2. **FIFO Oluşturma:** **mkfifo** fonksiyonu kullanılarak **REQUEST_FIFO** ve **RESPONSE_FIFO** boruları oluşturulur.

3. **Dosya İşlemleri:** İstemci, **REQUEST_FIFO**'yu yazma (**O_WRONLY**) ve **RESPONSE_FIFO**'yu okuma (**O_RDONLY**) modunda açar.
4. **İstek Gönderme ve Cevap Alma:** İstemci, sunucuya bir istek gönderir (**REQUEST_FIFO**'ya yazar) ve ardından sunucudan gelen cevabı alır (**RESPONSE_FIFO**'dan okur).
5. **FIFO Silme:** İstemci, işini tamamladıktan sonra **REQUEST_FIFO** ve **RESPONSE_FIFO**'yu siler ve programı sonlandırır.
6. **Uyarı:** İstemci, sunucuya **bye** mesajını gönderir ancak sunucu programında bu mesaj işlenmemiştir. Daha gelişmiş bir uygulama için bu tür durumların ele alınması önemlidir.

Her iki program da, gelen isteği basit bir cevapla yanıtlayan temel bir iletişim mekanizması sunmaktadır. Gerçek uygulamalarda, bu temel mekanizmalar genellikle daha karmaşık işlemler ve güvenlik kontrolleri ile birleştirilir.