

```
/*
 * Copyright (c) 2004 Baris Simsek.
 *
 * Permission is granted to copy, distribute and/or modify this document
 * under the terms of the GNU Free Documentation License, Version 1.2
 * or any later version published by the Free Software Foundation;
 * with no Invariant Sections, no Front-Cover Texts, and no Back-Cover
 * Texts. A copy of the license is included in the section entitled "GNU
 * Free Documentation License".
 */
```

## Belge Hakkında

Bu belge 10 Kasım 1998'de KTÜ Bilgisayar Klübü için yazılmıştır.

## Bakış

UNIX benzeri işletim sistemlerinin çok hoş bir özelliği de küçük komut zincirlerini bir araya getirerek büyük programlardan daha kabiliyetli küçük programlar yazmaktır. Bahsettiğim, süreçler arası haberleşme tekniği mekanizması olan pipe'tır. Pipe ilk olarak UNIX'in geliştirildiği Bell laboratuvarlarında kullanılmıştır. Pipe işlemleri "|" simgesi ile gösterilir. Hemen hemen bütün kabuklar bunu destekler. Borulama olarak türkçeleştirebileceğimiz bu işlem, kısaca bir programın standart çıkışını başka bir programa standart giriş olarak verme diye tanımlanabilir.

Örneğin:

```
$ ls -l | grep cpp
```

Yukarıdaki komut dizinde, içerisinde cpp geçen dosyaların ismi ekrana basar. Şöyleki: "ls -l" dizindeki dosyaları çıktı olarak üretir. Bu çıktılar girdi olarak grep komutuna verilir. grep ise girdide arama yapan bir komuttur. Dolayısıyla dizindeki dosyalardan ismi cpp içerenlerin adını yazar. Burada dikkat ederseniz "ls -l" komutunun çıktısı ekrana basılmadı. Aynı şekilde grep'e üçüncü parametre olarak verilmesi gereken dosya isimleri verilmedi.

FIFO, pipe ile transfer edilecek veriler için bir yoldur. İlk giren, ilk çıkar mantığı ile çalışır. Gerçek bir boru düşünün. Boruya kırmızı bir top, sonra mavi bir top, sonra yeşil bir top sokalım. Borunun diğer tarafından sırasıyla kırmızı, mavi ve yeşil top çıkar. Bu bir veri yapısıdır. Bundan başka ters çalışan yığın denilen LIFO vardır. Burada son giren ilk çıkar. Bunu da şu şekilde hayal edin: Bir çuvala üst üste sırasıyla kırmızı, yeşil, mavi paketler koyalım ve kargoya verelim. Kargoyu teslim alan paketleri mavi, yeşil, kırmızı sırasıyla çıkarır.

UNIX sistemler iki türlü pipe açılmasına olanak tanımaktadır.

## popen() -- Biçimli Pipe İşlemi

```
#include <stdio.h>
FILE *popen(const char *command, const char *type);
```

popen() bir pipe oluşturarak yeni bir süreç açar. Bu süreç açılması aşamasında fork() yapılır ve

kabuk (shell) çağrılır. "command" parametresi, komut satırı stringidir ve -c parametresi ile /bin/sh 'a geçirilir. "type" parametresi açılış modunu belirlemek içindir. "r", read olarak; "w", write olarak aç demek. Eğer fork() işlemi veya pipe() işlemi hatalı olursa, veya işletim sisteminden bellek alamazsa NULL değerini döndürür. popen() ile açılan bir pipe, pclose ile kapatılmalıdır.

```
int pclose(FILE *stream);
```

Pipe ile haberleşmek için fprintf() veya fscanf() fonksiyonları kullanılır.

## pipe() -- Alt Seviye Pipe İşlemi

```
#include <unistd.h>
int pipe(int filedes[2]);
```

Bu fonksiyon iki adet dosya tanımlayıcısı oluşturur ve filedes[] ile gösterilen diziye yerleştirir: okumak için filedes[0] ve yazmak için filedes[1]. Fonksiyon sorunsuz çalıştığında 0 değerini geri döndürür.

```
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>

int main()
{
    pid_t pid;
    int mypipe[2];
    char buffer[100];

    /* Pipe oluştur. */
    if (pipe(mypipe) == -1) {
        perror("pipe");
        return -1;
    }

    /* Çocuk süreç oluştur. */
    pid = fork();
    if(pid == (pid_t) 0) { /* Çocuk süreç. */
        read(mypipe[0], buffer, 100);
        printf("%s ", buffer);
        return -1;
    }
    else if(pid < (pid_t) 0) {
        perror("fork");
        return -1;
    }
    else { /* Ana süreç. */
        write(mypipe[1], "hello world!", 13);
        return 1;
    }

    return 0;
}
```

Yukarıdaki örnekte fork() ile ikinci bir süreç oluşturduk. Ve sonra bu süreç ile pipe'in okunan kısmından (mypipe[0]) 100 byte'lık buffer okuduk. Ana süreç içerisinde de pipe'in yazılacak kısmına (mypipe[1]), "hello world!" mesajını yazdık.

Fork yaptığınız zaman çocuk süreç ana sürecin bütün durumuna aynen sahip olur. Açık dosya tanımlayıcılarını da kopyalar. Bu nedenle pipe oluşturduktan sonra fork ederseniz, oluşturduğunuz pipe'ı iki process'in haberleşmesi için kullanabilirsiniz. Yukarıdaki kod buna bir örnektir. Ekranaya yazılan "hello world!" mesajı, ana sürecin çıktısı olsa da ekrana basan çocuk süreçtir. Ana süreç bu mesajı pipe'a yazmış, çocuk süreç te pipe'dan okumuştur.

\* Dikkat: pipe çağrısını fork() 'tan önce yaptığıma dikkat edin.

Eğer pipe'a yazılardan çok veri okumaya çalışırsanız, ancak yazılan kadar veri alabilirsiniz. Eğer pipe'dan son okumanızdan sonra pipe'a hiçbir şey yazılmamış ise ve siz pipe'dan okumaya çalışıyorsanız programınız bloke edilecektir. Ta ki yeni veri yazılana kadar veya pipe'ın yazılan ucu kapatılana kadar. Bunu göz önünde tutarak programlarınızı istenmeyen sonuçlardan koruyunuz. UNIX işletim sistemlerinde dışardan veri bekleyen süreçlerin "zamanlayıcı" (scheduler) tarafından bloke edildiğini hatırlayın.

Linux pipe'ları yazılırken 4096 byte sınırına sahiptir. 4k'dan daha büyük veri pipe'a yazacağı zaman süreç bloke olur ve pipe'dan okuma yapılmasını bekler.

## **FIFO Dosyalar (Named PIPE)**

Eğer iki süreç bir pipe üzerinden haberleşmek istiyor ve aynı soydan gelmiyorsa (fork ile) pipe() fonksiyonu işimizi görmeyecektir. Ancak üzülmeyin, bu durumda FIFO dosyaları (named pipe) kullanabilirsiniz.

FIFO dosyası, dosya sisteminizdeki sıradan bir dosya gibidir, ancak tıpkı pipe gibi özellikleri vardır. İki süreçten birisi FIFO dosyasını okumak, diğeri yazmak için açabilir. Bu iki süreç FIFO aracılığı ile haberleşebilir. Birinin yazdığını diğeri okur. Yukarıdaki gibi bloke olma durumu burada da söz konusu. Sözlere bir yana bırakıp gösteri yapmaya ne dersiniz?

```
cclub:~> mkfifo myfifo
cclub:~> ls -l myfifo
prw-r--r-- 1 root root 0 Dec 23 20:04 myfifo|
```

Yukarda yaptıklarına bir göz atalım: "mkfifo" komutu ile myfifo adında özel dosyamızı oluşturdu. "ls -l" komutu ile dosyamıza baktığımızda, dosya isminin sonuna "|" karakterinin eklendiğini göreceksiniz. Bu karakter ile dosya sistemimiz onun bir FIFO dosyası olduğunu işaretliyor. Biraz man sayfası okuyalım:

### **DESCRIPTION**

mkfifo creates FIFOs (also called "named pipes") with the specified filenames.

A "FIFO" is a special file type that permits independent processes to communicate. One process opens the FIFO file for writing, and another for reading, after which data can flow as with the usual anonymous pipe in shells or elsewhere.

By default, the mode of created FIFOs is 0666 (^a+rw') minus the bits set in the umask.

Şimdi gösteriye devam edelim.

```
cclub:~> cat myfifo
```

Bu komut sonucunda kabuk program (shell) bekleyecektir. Çünkü sürecimiz bloklanmıştır. myfifo'ya henüz bir şey yazılmadığı halde biz ondan okumak istiyoruz. Şimdi başka bir terminal açın.

```
cclub:~> cat > myfifo
```

Bu komutun anlamı klavyeden alınacak girdileri "myfifo" dosyasına yaz. ">" sembolü standart çıktıyı yönlendirmek için kullanılır. "cat" fonksiyonu normalde parametre olarak verilen dosyayı görüntüler. Eğer parametre vermezseniz girdi olarak standart girişi (burada klavye) kullanır. Şimdi en son açtığımız terminalde bir şeyler yazalım. Örneğin "hello world!" yazın. Diğer terminale geçtiğinizde ne göreceksiniz, izleyelim:

## 2. terminal

```
cclub:~> cat > myfifo  
hello world!
```

## 1. terminal

```
cclub:~> cat myfifo  
hello world!
```

Harika değil mi! İkinci açtığımız terminalde klavyeden girdiklerimizi FIFO dosyamıza yazdık. İlk açtığımız terminalde bloke olmuş sürecimiz pipe üzerine veri yazıldığını gördü ve okuyup "cat" komutu ile ekrana bastı. Gördüğümüz gibi iki süreç arasında haberleşmeyi sağladık.

Bu gösteriyi kabuk programı (shell) kullanarak yaptık. "cat" komutu sayesinde dosyadan okumayı ve yine "cat" komutu ile kabuk programın bir özelliği olan yönlendirmeyi (>) kullanarak ta yazmayı sağladık. FIFO dosyalarını kendi yazacağınız programlarda da kullanabilirsiniz. C içinde FIFO dosyası oluşturmak için `mkfifo()` fonksiyonunu kullanabilirsiniz. Yine man sayfalarımıza göz atalım:

NAME

`mkfifo` - make a FIFO special file (a named pipe)

SYNOPSIS

```
#include <sys/types.h>  
#include <sys/stat.h>
```

```
int mkfifo ( const char *pathname, mode_t mode );
```

Prototipi yukarda gördüğümüz gibi. Kullanabilmek için `sys/types.h` ve `sys/stat.h` başlık dosyalarını yüklemeniz gerekir. Yazacağınız iki süreçten birisi FIFO dosyasından okumaya, diğeri de FIFO dosyasına yazmaya çalışsın. Yukarıdaki iki örneği bu şekilde simule edebilirsiniz.

### *fifo-reader.c*

```
#include <sys/types.h>
#include <sys/stat.h>

int main()
{
    int fd;
    char buffer[100];

    fd=fopen("myfifo","r");

    fgets(buffer,100,fd);
    printf("Buffer: %s ",buffer);

    return 0;
}
```

### *fifo-writer.c*

```
#include <sys/types.h>
#include <sys/stat.h>

int main()
{
    int fd;
    char buffer[13];

    strcpy(buffer,"hello world!");
    fd=fopen("myfifo","w");

    fputs(buffer,fd);

    return 0;
}
```

- Not: Eğer bir süreç okumak için açılmamış FIFO'ya yazmaya çalışıyorsa SIGPIPE sinyali üretir.  
Pipe ve FIFO, temelde birbiri ile haberleşmek için dizayn edilmemiş iki süreci haberleştirebilir.

Barış Şimşek  
EnderUNIX Yazılım Geliştirme Takımı  
<http://www.enderunix.org/simsek/>