İşletim Sistemleri Uygulama 12

Linux'de Pipe Yapısı

Bilgisayar Mühendisliği

İstanbul Teknik Üniversitesi
34469 Maslak, İstanbul

May 11, 2011

Bugün


İSTANBUL TEKNİK ÜNİVERSİTESİ
1773


İTÜ
Bilgisayar Mühendisliği

Pipe Nedir?

- İşletim Sistemi tarafından yönetilen prosesler arası tek yönlü mesajlaşma yöntemi.
- Pipe'lar, sınırlı miktarda veri bulundurabilen FIFO türünde özel bir dosya olarak düşünülebilir.
- Genel olarak bir proses pipe'a veri yazarken diğeri okur.

Pipe ve Eşzamanlılık

İşletim Sistemi, pipe'ı kullanan proseslerin eş zamanlı çalışmasını sağlar.

- Pipe dolu ise, pipe'a yazmaya çalışan proses askıya alınır.
- Pipe boş ise, pipe'dan okuma yapmaya çalışan proses askıya alınır.
- Bir pipe'ın yazma ucu kapanmışsa, okuma ucu EOF görür.
- Bir pipe'ın okuma ucu kapanmışsa, yazma ucu SIGPIPE sinyali verir.

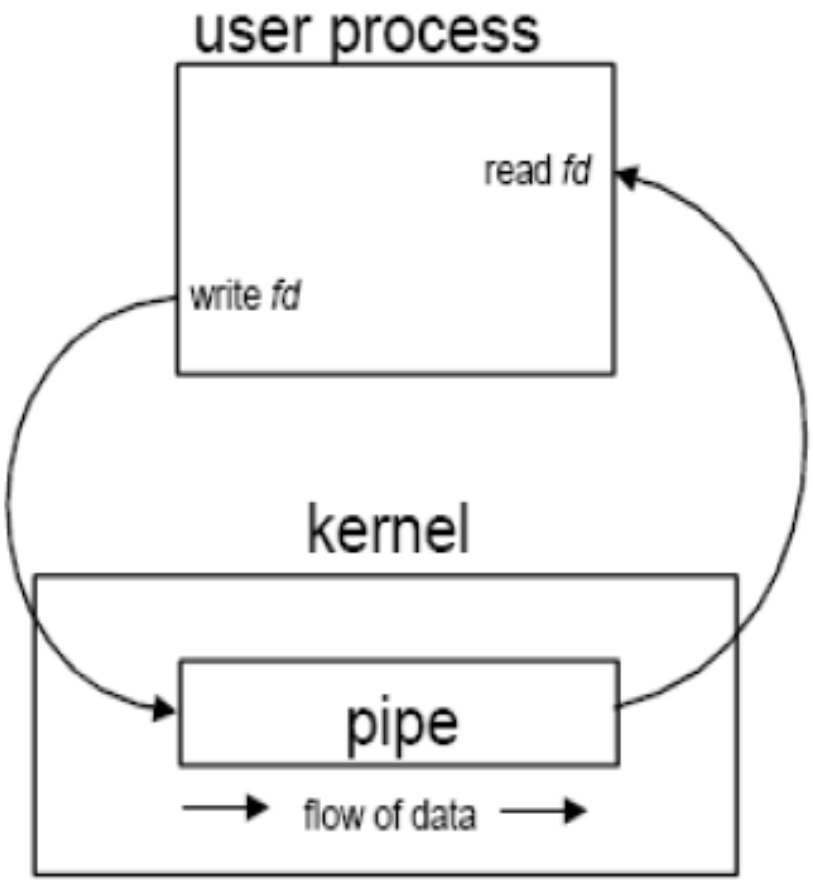
Pipe Türleri

- En önemli kısıtlamaları isimsiz olmalarıdır. Bu durum ancak aynı anneden doğan prosesler tarafından kullanılabilmeleri kısıtını getirir.
- Bu durum Sistem III Unix'lerde FIFO yapısının ortaya çıkmasıyla giderilmeye çalışılmıştır. FIFO'lar isimlendirilmiş pipe'lar olarak da adlandırılırlar. Birbirlerinden haberdar olmayan prosesler tarafından da kullanılabilirler.

Pipe/FIFO

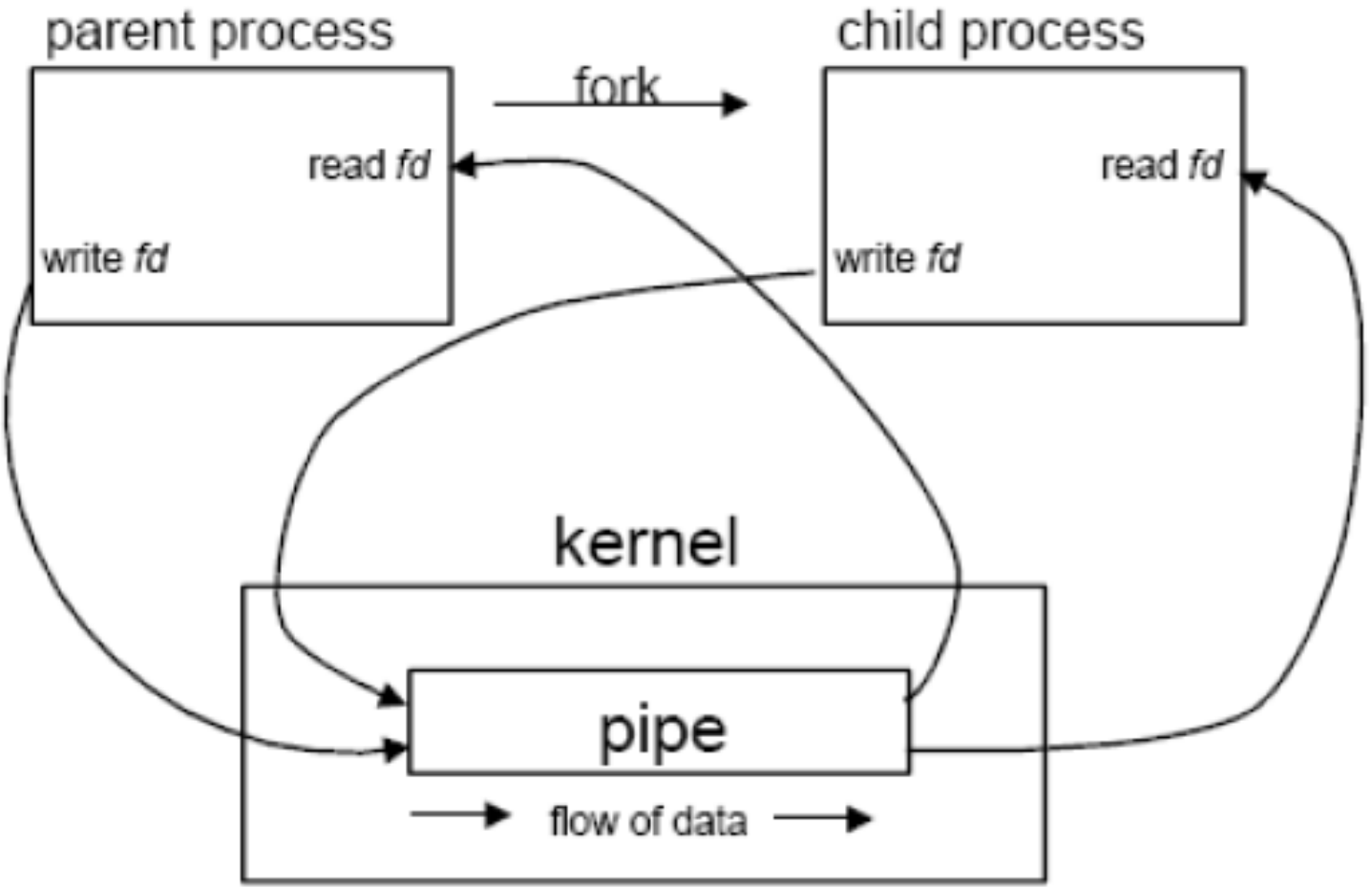
- Pipe son `close` komutu ile yok olur.
- FIFO'lar ise ancak `unlink` komutu çağırılarak dosya sisteminden silinirler.
- Bir pipe yaratmak ve açmak için `pipe()` fonksiyonunu çağırmak yeterlidir.
- FIFO yaratmak ve açmak için sırası ile `mkfifo()` ve `open()` fonksiyonları çağrılmalıdır.

Pipe kullanımı



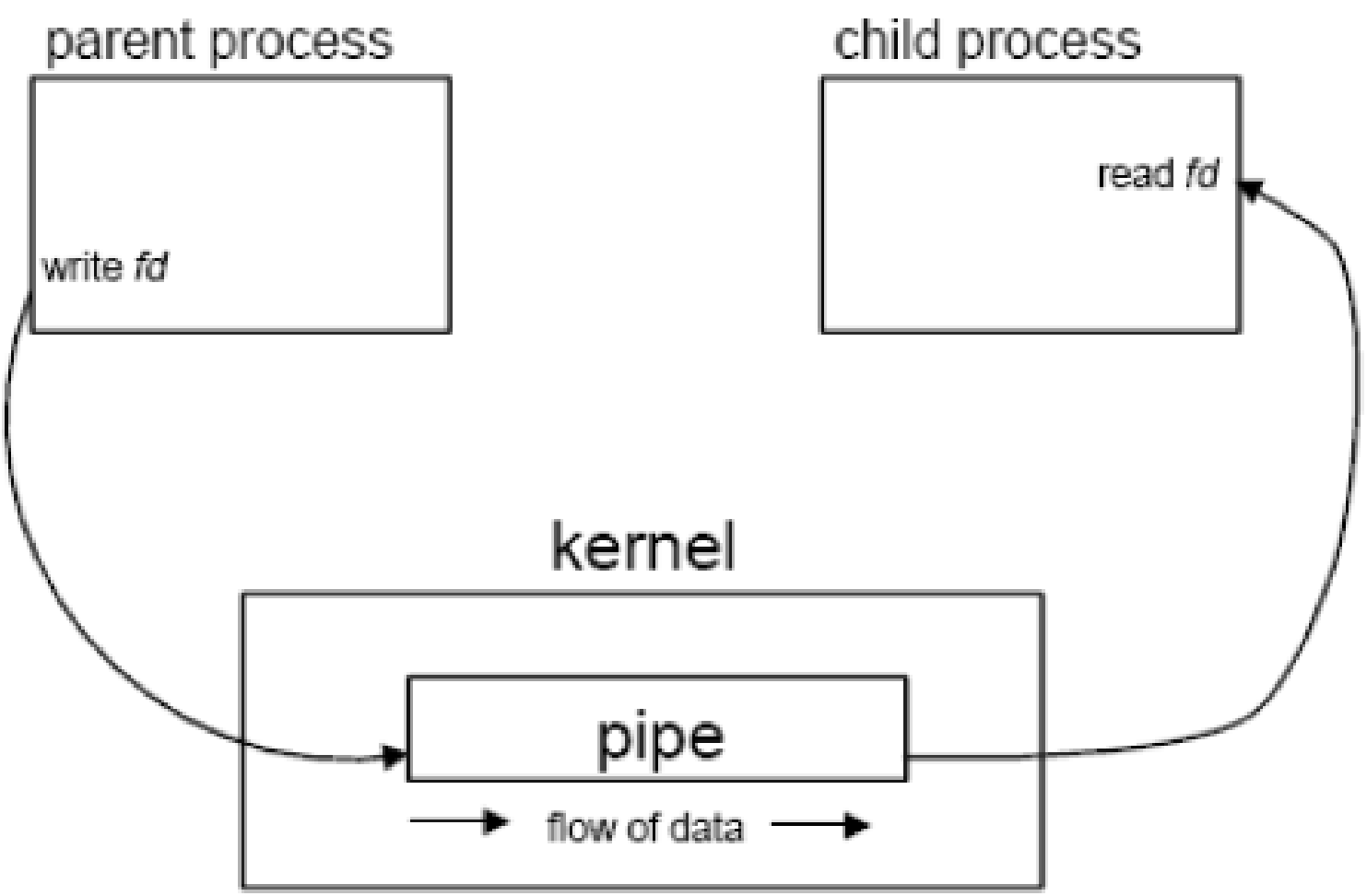
Tek bir proses içerisinde pipe yaratıldığı zaman

Pipe kullanımı



Anne proses `fork()` ile bir çocuk proses oluşturduğu zaman,
her iki proses de pipe'ın oku(`pipe[0]`) ve yaz(`pipe[1]`) uçlarına sahip olurlar

Pipe kullanımı



Daha sonra Yazıcı okuma ucunu, Okuyucu ise yazma ucunu kapatır.
Tek yönlü iletişim sağlanır.

Pipe kullanımı

```
<unistd.h>
int pipe(int filedes[2]);
int close(int fd);
```

- İki adet akış yoluna sahiptir.
- Normalde biri yazma, diğer okuma amaçlı kullanılır. (LINUX)
- Her ikisi de hem yazma hem de okuma için kullanılırsa : full-duplex (SOLARIS)
- İşlem tamamlanırsa 0, hata durumunda -1 döndürür.
- İki dosya belirteci döndürür.
  - `filedes[0]` okuma için
  - `filedes[1]` yazma için

Pipe kullanımı

```
1   #include <stdio.h>
    #define NOFSEND 3
    #define SOFSEND 4

    void main(){
6           int c, p[2],i;
            char send[NOFSEND][SOFSEND]={"Fee\0","Faa\0","Foo\0"};
            char rec[SOFSEND];

            if (pipe(p) < 0) printf("Can't create a pipe.\n");
11
            if((c=fork()) < 0) printf("Can't fork.\n");
            else if (c > 0){
                    close(p[0]);
                    for(i=0;i<NOFSEND;i++){
16                  if (write(p[1], send[i], SOFSEND) < 0)
                            printf("M: Can't write %d\n",i+1);
                    printf("M: I wrote %d.\n",i+1);
            }
            wait(NULL);
21          }
            else{
                    sleep(1);
                    close(p[1]);
                    for(i=0;i<NOFSEND;i++){
26                          if (read(p[0], &rec, SOFSEND) < 0)
                                    printf("C: Can't read %d\n",i+1);
                            printf("C: I read \"%s\"\n", rec);
                    }
            }
31  }
```

Pipe kullanımı

- Komut satırından kullanım
  `linux$ ps -ef | grep $USER | cat -n`
  `ps -ef` $\Longrightarrow$ `grep $USER` $\Longrightarrow$ `cat -n`

- Program içinden başka prosese çağrı yapma
  `FILE *popen(const char *command, const char *mode);`
  `popen` : süreç içinde bir pipe akışı oluşturur.
  `int pclose(FILE *stream);`
  `pclose` : süreç içindeki pipe akışını kapatır.

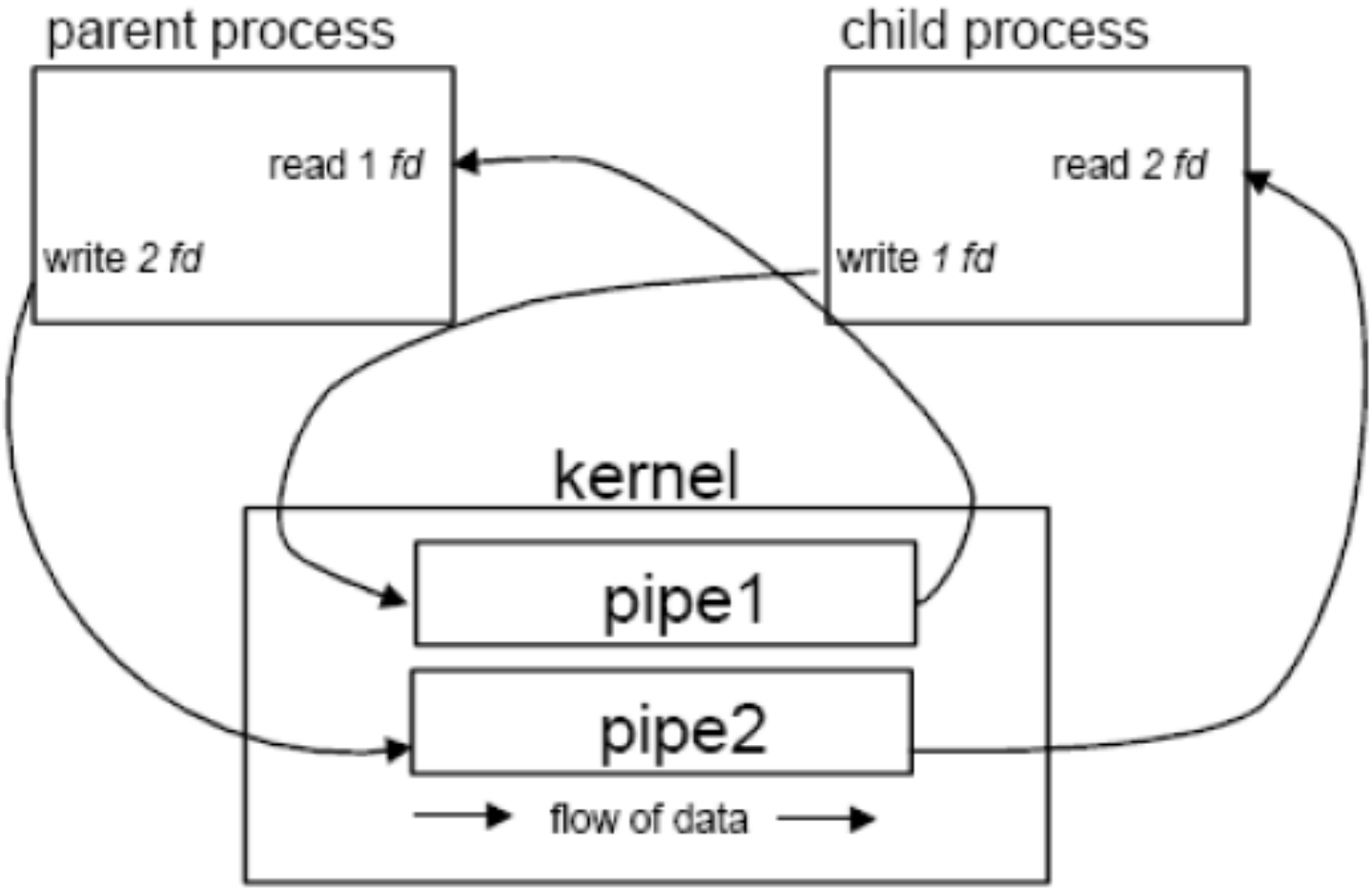
Pipe kullanımı

```
#include <stdio.h>

void main(){
        FILE *f;
        char line[80];

        if( (f=popen("ls -la", "r")) == NULL)
                printf("Can't open pipe.\n");

        while(fgets(line, 80, f) != NULL)
                printf("%s", line);

        pclose(f);
}
```

Pipe kullanımı

```
1  #include <stdio.h>

   void main(){
           FILE *f, *g;
           char line[80];
6
           if( (f=popen("ls -la", "r")) == NULL)
                   printf("Can't open pipe.\n");

           if( (g=popen("grep -i *.out", "w")) == NULL)
11                 printf("Can't open pipe.\n");

           while(fgets(line, 80, f) != NULL)
                   fputs(line, g);

16         pclose(f);
           pclose(g);
   }
```

Çift yönlü Pipe kullanımı



Çift yönlü Pipe kullanımı

```
#include <stdio.h>

void main(){
        int c, p[2], q[2];
        if (pipe(p) < 0 || pipe(q) < 0) printf("Can't create pipes.\n");
        if((c=fork()) < 0) printf("Can't fork.\n");
        else if (c > 0){
                close(p[0]);
                close(q[1]);
                char r[4];
                if (write(p[1], "Foo\0", 4) < 0) printf("M: Can't write\n");
                printf("M: I wrote.\n");
                if (read(q[0], &r, 4) < 0) printf("M: Can't read\n");
                printf("M: I read \"%s\"\n", r);
        }else{
                close(p[1]);
                close(q[0]);
                char r[4];
                if (write(q[1], "Bar\0", 4) < 0) printf("C: Can't write\n");
                printf("C: I wrote.\n");
                if (read(p[0], &r, 4) < 0) printf("C: Can't read\n");
                printf("C: I read \"%s\"\n", r);
        }
}
```

Çift yönlü Pipe kullanımı

```
1   #include <stdio.h>
    #include <pthread.h>

    #define NOFSEND 3
    #define SOFSEND 4
6   #define NOFITER 10
    int p[2], q[2];

    void* sender(void *arg){
            char* me=(char*)arg;
11          int i;
            char send[NOFSEND][SOFSEND]={"Fee\0","Faa\0","Foo\0"};

            if((*me)=='M'){
                    for(i=0;i<NOFITER;i++){
16                          if (write(p[1], send[i%NOFSEND], SOFSEND) < 0)
                                    printf("M: Can't write\n");
                            printf("M: I wrote.\n");
                    }
            }
21          else{
                    for(i=2;i<NOFITER+2;i++){
                            if (write(q[1], send[i%NOFSEND], SOFSEND) < 0)
                                    printf("C: Can't write\n");
                            printf("C: I wrote.\n");
26                  }
            }
    }
```

Çift yönlü Pipe kullanımı

```
void* reciever(void *arg){
        char* me=(char*)arg;
        int i; char rec[SOFSEND];

        if((*me)=='M'){
                for(i=0;i<NOFITER;i++){
                        if (read(q[0], &rec, SOFSEND) < 0)
                                printf("M: Can't read\n");
                        printf("M: I read %s.\n",rec);
                }
        }
        else{
                for(i=0;i<NOFITER;i++){
                        if (read(p[0], &rec, SOFSEND) < 0)
                                printf("C: Can't read\n");
                        printf("C: I read %s.\n",rec);
                }
        }
}
```

Çift yönlü Pipe kullanımı

```
1   int main(){
        int c;
        pthread_t mSend,mRecv,cSend,cRecv;
        char mother='M',child='C';

6       if (pipe(p) < 0 || pipe(q) < 0)
            printf("Can't create pipes.\n");

        if((c=fork()) < 0) printf("Can't fork.\n");
        else if (c > 0){
11          close(p[0]);
            close(q[1]);
            if( pthread_create(&mSend,NULL,sender,&mother) ||
            pthread_create(&mRecv,NULL,reciever,&mother)){
                printf("error creating thread");
16              return 1;
            }
            if( pthread_join(mSend,NULL) || pthread_join(mRecv,NULL) ){
                printf("error joining thread");
                return 1;
21          }
            wait(NULL);
```

Çift yönlü Pipe kullanımı

```
        }else{
                close(p[1]);
3               close(q[0]);
                if( pthread_create(&cSend,NULL,sender,&child) ||
                pthread_create(&cRecv,NULL,reciever,&child)){
                        printf("error creating thread");
                        return 1;
8               }
                if( pthread_join(cSend,NULL) || pthread_join(cRecv,NULL) ){
                        printf("error joining thread");
                        return 1;
                }
13      }
    }
```

FIFO kullanımı

```
1  #include <stdio.h>
   #include <unistd.h>

   void main(){
           int f;
6          FILE *a, *b;
           char r[7];
           mkfifo("myFifo", 0777);
           if( (f=fork()) < 0) printf("Can't fork.\n");
           else if(f > 0){
11                 a = fopen("myFifo", "w"); //Write
                   fputs("FooBar\0", a);
                   fclose(a);
           }else{
                   b = fopen("myFifo", "r"); //Read
16                 fgets(r, 7, b);
                   fclose(b);
                   printf("Read: %s\n",r);
           }
           unlink("myFifo");
21 }
```

Komut satırından FIFO kullanımı

```
1  #!/bin/bash
2  mkfifo MYFIFO
3  grep -i *.out < MYFIFO &
4  echo Do Something...
5  ls -la > MYFIFO
6  rm MYFIFO
```