Linux’da Pipe Yapısı

PIPE Nedir?

- İşletim Sistemi tarafından yönetilen prosesler arası mesajlaşma yöntemi
- Pipe'lar, sınırlı miktarda veri bulundurabilen FIFO türünde özel bir dosya olarak düşünülebilir.
- Genel olarak bir proses pipe'e 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 proses tarafından yazma için açılan bir pipe, başkası tarafından okuma amaçlı açılmışsa askıya alınır.

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 olarak da adlandırılır. Birbirlerinden haberdar olmayan prosesler tarafından da kullanılabilirler.

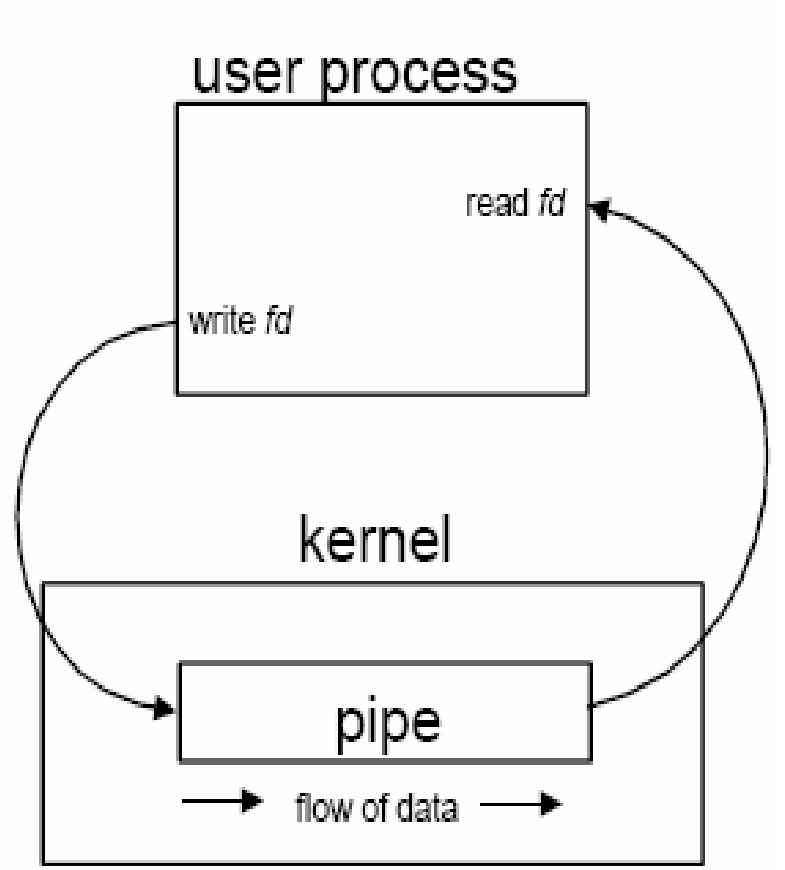
Pipe / FIFO

- Pipe son “close” komutu ile yok olur.
- FIFO’lar ise ancak “unlink” komutu çağırılarak dosya sisteminden silinirler.

Pipe / FIFO (devam)

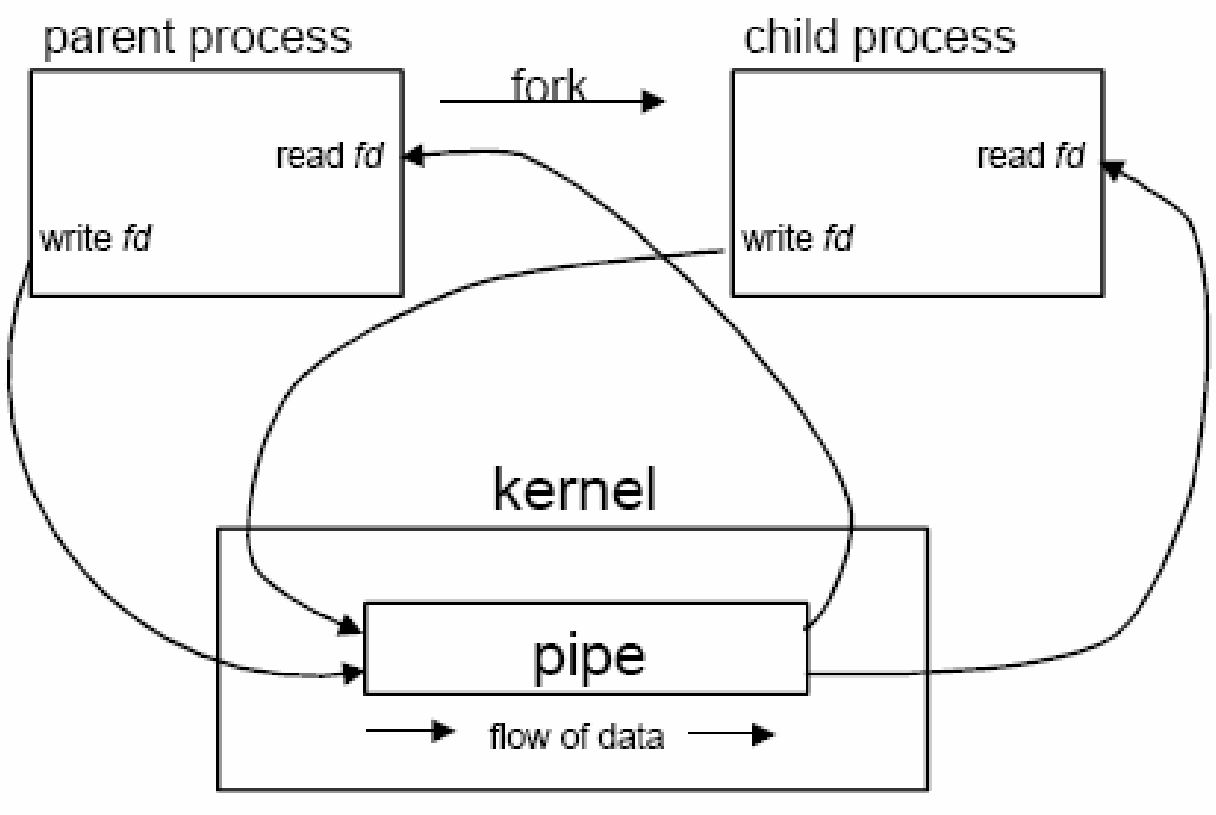
- Bir pipe yaratmak ve açmak için “pipe” fonksiyonunu çağırmak yeterlidir.
- FIFO yaratmak ve açmak için sırası ile “mkfifo” ve “open” fonksiyonları çağrılmalıdır.

Pipe



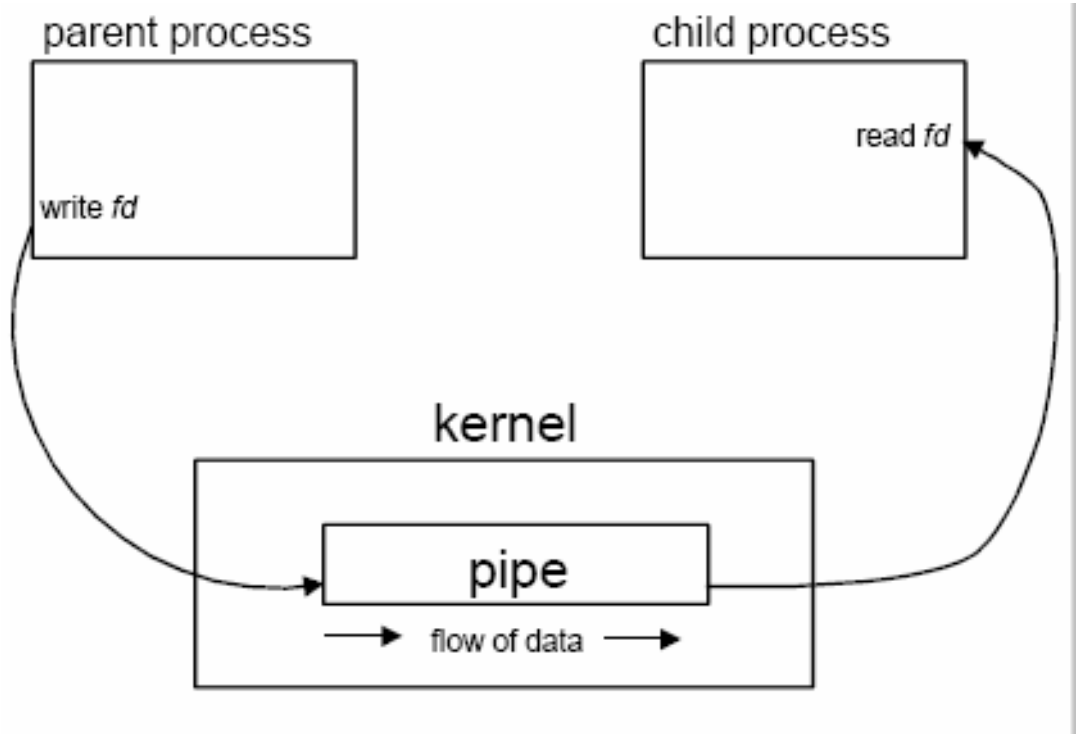
Tek bir proses içerisinde pipe yaratıldığı zaman

Pipe (devam)



Anne proses “fork()” ile bir çocuk proses oluşturduğu zaman, her iki proses de Pipe’ın “oku” ve “yaz” uçlarına sahip olurlar

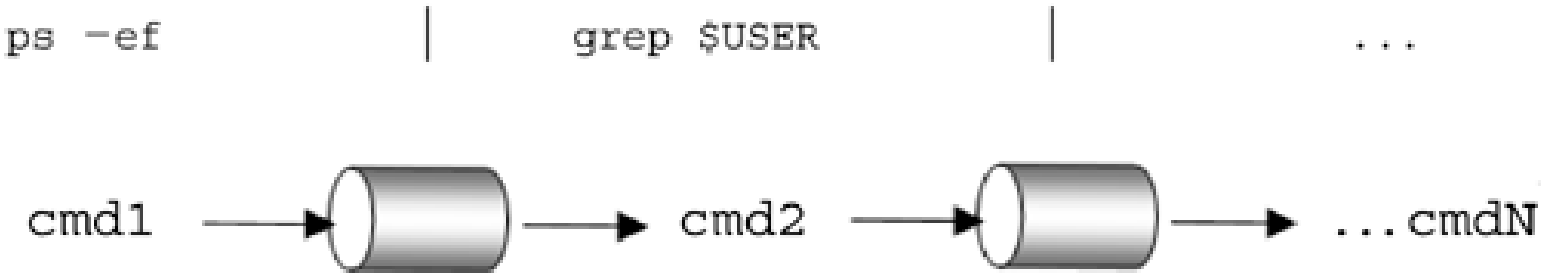
Pipe (devam)



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

Komut Satırında Pipe Kullanımı

- linux$ ps –ef | grep $USER | cat -n



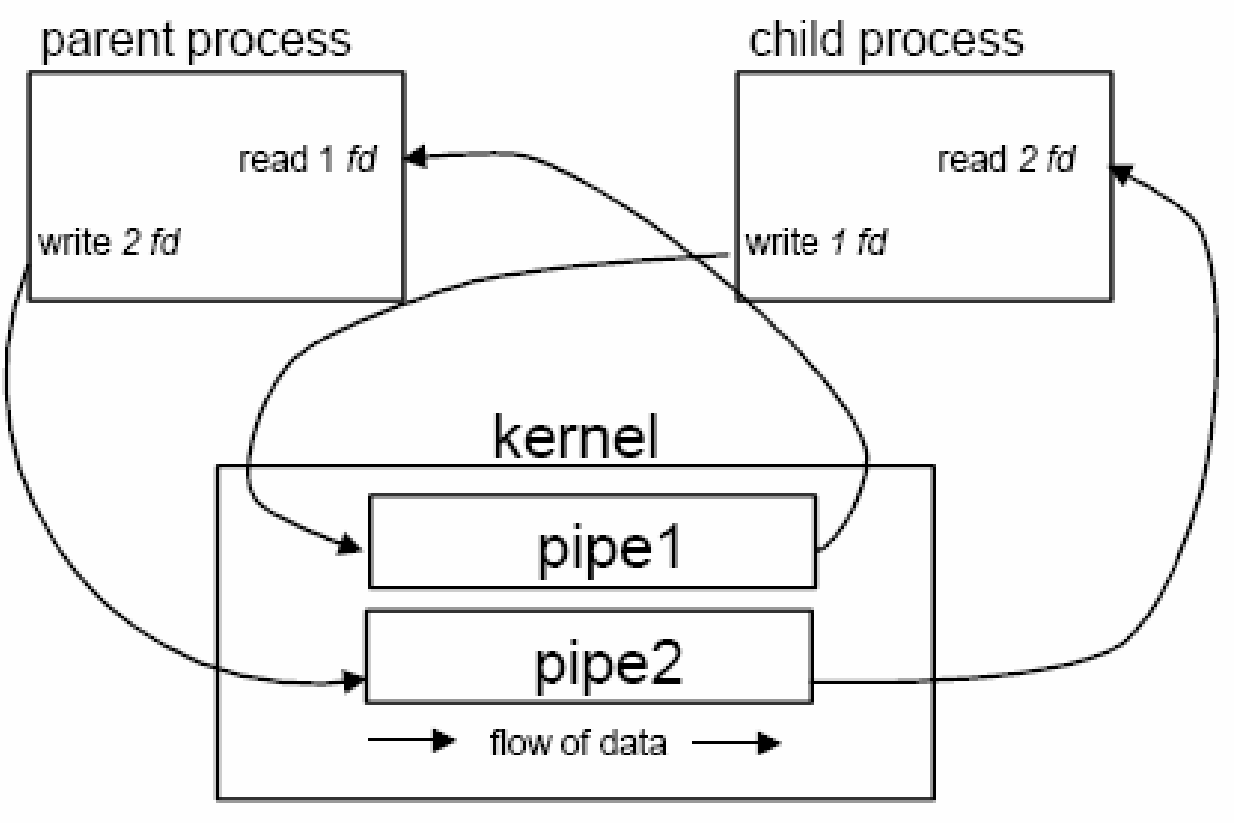
İsimsiz Pipe’lar

<unistd.h>

int pipe(int filedes[2]);

- 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)

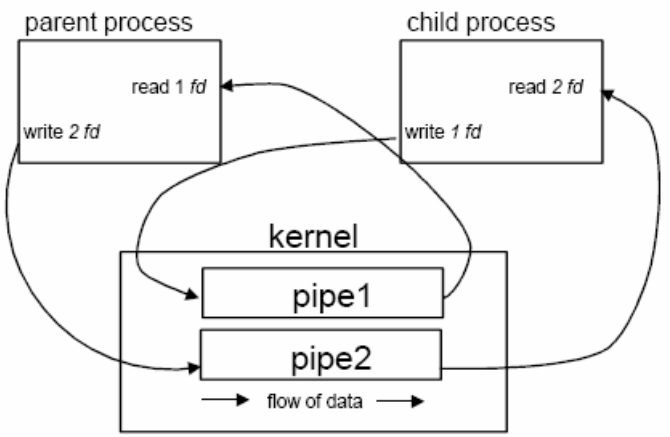
Tek yönlü Pipe'larla Çift Yönlü İletişim



int pipe(int filedes[2]);

- İşlem tamamlanırsa 0, hata durumunda -1 döndürür.
- İki dosya belirteci döndürür.
  - filedes[0] okuma için
  - filedes[1] yazma için

int close(int fd);




```
main()
{       int child, pipe1[2], pipe2[2];
        if (pipe(pipe1) < 0 || pipe(pipe2) < 0)   err_sys("can't create pipes");
        if (( child = fork()) < 0)      err_sys("can't fork");
        else if (child > 0) {  /* PARENT CODE */
                close(pipe1[1]);      close(pipe2[0]);
                client(pipe1[0], pipe2[1]);
                while (wait((int *) 0) != child) ;    /* WAIT FOR THIS CHILD */
                close(pipe1[0]);      close(pipe2[1]);
                exit(0);
                }
        else {                  /* CHILD CODE */
                close(pipe1[0]);      close(pipe2[1]);
                server(pipe2[0], pipe1[1]);
                close(pipe2[0]);      close(pipe1[1]);
                exit(0);
                }
}
```

FULL DUPLEX PIPE

```
#include   "unpipc.h"
int main(int argc, char **argv)
{     int                    fd[2], n;
      char   c;
      pid_t  childpid;
      Pipe(fd);                        /* assumes a full-duplex pipe (e.g., SVR4) */
      if ( (childpid = Fork()) == 0) {                        /* child */
            sleep(3);
            if ( (n = Read(fd[0], &c, 1)) != 1)
                  err_quit("child: read returned %d", n);
            printf("child read %c\n", c);
            Write(fd[0], "c", 1);
            exit(0);
      }
            /* 4parent */
      Write(fd[1], "p", 1);
      if ( (n = Read(fd[1], &c, 1)) != 1)
            err_quit("parent: read returned %d", n);
      printf("parent read %c\n", c);
      exit(0);
}
```

Fduplex
Child read p
Parent read c

Popen & Pclose

- #include <stdio.h>

FILE *popen(const char **command*, const char **mode*);

popen - initiate pipe streams to or from a process

int pclose(FILE **stream*);

pclose - close a pipe stream to or from a process

```
#include   <stdio.h>
#define    MAXLINE              1024
main()
{    int      n;
     char   line[MAXLINE], command[MAXLINE + 10];
     FILE   *fp;
     /* Read the filename from standard input. */
     if (fgets(line, MAXLINE, stdin) == NULL)
               err_sys("filename read error");

     /* Use popen to create a pipe and execute the command. */
     sprintf(command, "cat %s", line);
     if ( (fp = popen(command, "r")) == NULL)
               err_sys("popen error");

     /* Read the data from the FILE pointer and write  to standard output.          */
     while ((fgets(line, MAXLINE, fp)) != NULL) {
               n = strlen(line);
               if (write(1, line, n) != n)
                         err_sys("data write error");
     }
     if (ferror(fp))
               err_sys("fgets error");
     pclose(fp);
     exit(0);
}
```

References

Stevens, Richard. Unix network programming, Chapter 4