SİSTEM PROGRAMLAMA

Deney#5 : UNIX 'te IPC mekanizması olarak Pipe kullanımı

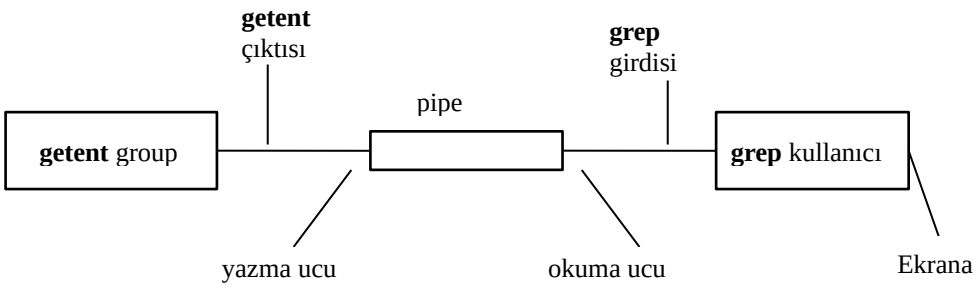
GİRİŞ

Bu deney iki bölümden oluşmaktadır. İlk bölümde adsız pipe, ikinci bölümde adlandırılmış pipe kullanılacaktır. **Aşağıdaki deneyleri gerçekleştiriniz.** Verilen Kodlar da hatalar çıkarsa düzeltiniz(hatalar olabilir). Geliştirdiğiniz kodları, sorulara olan cevapları ve ekran görüntülerini **bir Rapor olarak hazırlayıp, sisteme yükleyiniz**.

Bölüm-1 (Adsız Pipe)

Adsız pipe akraba olan prosesler arasında tek yönlü bir iletişim kanalıdır. Bir pipe' ın iki ucu vardır: **yazma ucu** ve **okuma ucu** . Herbir uç, tıpkı dosyalar gibi integer bir tanımlayıya sahiptir. **ret=pipe(fd)** sistem çağrısı ile okuma ucu **fd[0]** ve yazma ucu **fd[1]** ile tanımlanan bir pipe yaratılır. Dönen **ret** değeri, pipe'ın başarılı yaratılıp, yaratılmasıdığını gösterir. Parent proses isimsiz bir pipe yarattıktan sonra iki child proses yaratır. Child proseslerden birisi kabuk komutu olan "**getent** group" programını çalıştırıp, çıktısını pipe'ın yazma ucuna yazar. Bu program mevcut **grup** adlarını ekranda gösterir. Diğer çocuk proses kabuk komutu olan "**grep** kullanıcı dosya" programını çalıştırıp, girdisini pipe'ın okuma ucundan alır. Grep komutu genel olarak metin arama için kullanılır. Verilen parametrelerle **grep** komutu, verilen dosyada kullanıcı adının geçtiği satırları ekrana yazar. Sonuç olarak, iki çocuk prosesin birlikte çalışması ile verilen kullanıcının hangi gruplara üye olduğu **aşağıdaki** kabuk komutu ile listelenebilir.

getent group | grep kullanıcı,



1. lab5a isminde bir dizin içinde aşağıdaki kodda grep komutu parametresi olan "kullanıcı adınız" kısmını kendi linux kullanıcı adınız ile değiştirdikten sonra derleyip, çalışır halini oluşturun (karakter hataları çıkabilir düzeltiniz):

```
/* A program to experiment with an unnamed pipe */
#include <stdio.h>
#include <unistd.h>

main ()
{
int fd[2];  /*  Array of two descriptors for an unnamed  pipe */
int pid;    /*  Variable for a process identifier */

/*  A pipe should be created before any fork() */
/* Do you understand why? */

if (pipe(fd) < 0)
    {perror ("PIPE CREATION ERROR");
```

```
     exit (1);
    }

pid = fork ();  /*  Parent:  creating the first child process */
if (pid == 0)   /*  The first  child process starts here */
    {
    dup2 (fd[0],0);  /* Standard input will be taken from the  downstream of the pipe  */
    close (fd[1]);     /*  Upstream end of the pipe is closed for this process (not used) */
    execlp ("grep", "grep", "kullanıcı adınız", 0);  /* "grep" command  taking input from the pipe */
    }

else   /* Here the parent process continues */
pid = fork ();  /*  Parent:  creating the second child process */
if (pid == 0)   /*  The second child process starts  here */
    {dup2 (fd[1],1); /* Standard output  will be put  to the upstream end of the pipe */
     close (fd[0]);    /* Downstream end is closed  for this process (not used) */
     execlp ("getent", "getent","group", 0); /*  the command "getent" outputs to the pipe */
    }

else /* Parent process  closes for itself both ends of the pipe and waits for  children to terminate  */
    {
    close (fd[0]);
    close (fd[1]);
    wait (0);
    wait (0);
    }
}
```

Not: kütüphane fonksiyonu **dup2(fd1, fd2) , fd2** tanımlayıcısının gösterdiği dosyayı kapatır ve **fd2'yi**, **fd1'e** yönlendirir.

2. Programınızı çalıştırıp, sırasıyla aşağıdaki kabuk komutlarin çıktısıyla kıyaslayın. Sonucu yorumlayınız.

% **getent** group
% **getent** group | **grep kullanıcı_adınız**

3. Programınızda **execlp ("getent", . . .)** deyimini**;**
bir kaç **write(fd[1],…)** işlemi yapan bir kod bloğuyla değiştirin: pipe'a şu kelimeleri yazın ("this\n", "is\n", "a\", "message\n", 'from\n", "sending\n", "process\"). Pipe'a yazdıktan sonra **fd[1]** kapatın ve exit yapın. Eklediğiniz kodun, sonucunu analiz edin ve yorumlayın.

Not: bu kod değişikliği için, **dup2(fd[1],1)** yapmanıza gerek YOKTUR.

4. Adım-3 te güncellediğiniz Programınızda **execlp("grep", . . .)** deyimini**;**

bir kaç **read(fd[0],…)** işlemi yapan bir kod bloğuyla değiştirin. Kodunuz pipe'dan tüm kelimeleri (döngü içinde) okusun ve ekrana yazdırsın. Tüm kelimeleri okuduktan ve yazdırtıktan sonra proses **fd[0]' kapatsın ve** exit yapsın. Eklediğiniz kodun, sonucunu analiz edip ve yorumlayınız.

Not: Bu kod değişikliği için **dup2(fd[0], 0)** yapmanıza gerek YOKTUR.

5. Birden fazla proses aynı pipe'a yazabilir yada okuyabilir. Pipe'a yazma ve okuma atomiktir. Bunun etkisini görmek için **Bağımsız bir kod geliştiriniz**. Ana proses pipe yarattıktan sonra üç çocuk proses yaratsın. Çocuklardan ikisi pipe'a bir mesaj yazsın, son çocukta pipe'dan okuyup, ekrana yazsın. Proseslerin mesajlarinda bölünme var mı? Sonucu yorumlayınız.

6. İki proses arasında iki yönlü iletişim sağlamak için, her yön için ayrı bir pipe (toplamda iki pipe) kullanan **bir echo programı** geliştiriniz (ayni mesaj gider ve geri gelir, sonra ekrana yazdırılır) ve test ediniz.

Bölüm-2 (Adlandırılmış Pipe)

Adlandırılmış pipe iki ve daha fazla proces arasında tek yönlü bir iletişim mekanizmasıdır. Gerçekte adlandırılmış pipe proceslerden birisi tarafından yaratılan ve özel muamele gören bir dosyadır ve bu dosya diğer process tarafından iletişim amaçlı kullanılabilir. Adlandırılmış pipe'ı dizini listelerken "p" karakteri ile başladığını görürsünüz ("d" harfi dizin ve "-" karakteri sıradan dosyayı gösterir). Adlandırılmış pipe, **mknod()** sistem çağrısı veya kütüphane çağrısı **mkfifo()** ile yaratabiliriz**.**

Bu kısımda size verilen basit bir istemci/sunucu sisteminin pipe ile gerçekleştirilmiş birkaç versiyonunu analiz edeceksiniz. Sunucu adlandırılmış pipe yaratır ve pipedan okumaya çalışır. İstemci, sunucudan sonra başlar ve pipe'a bazı veri yazar ve sonlanır. Sunucu, istemciden gelen mesajı ekrana yazar ve sonlanır. Dikkat! Hata çıkan yerlerde gerekli düzeltmeleri yapınız.

Aşağıdaki deneyleri gerçekleştiriniz. Verilen Kodlar da hatalar çıkarsa düzeltiniz. Geliştirdiğiniz kodları, sorulara olan cevapları ve ekran görüntülerini **Raporunuza ekleyip, sisteme yükleyiniz**.

1. Bu kısım için bir dizin yaratıp, içinde aşağıdaki istemci ve sunucu programlarını derleyip, çalıştırınız.

```
/*  SUNUCU PROGRAM */
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

#define FIFONAME    "myfifo" /* burada pipe ozel bir ad verin  */


int
main(void)

{
   int n, fd;
   char buf[1024];  /* okuma ve yazma icin kullanilacak buffer */

   printf ("SUNUCU BASLAR…\n");

   /*
    * oncesinden ayni isimde pipe varsa sil
    */
   unlink(FIFONAME);

   /*
    * okuma ve yazma hakkiyla ilgili pipe yarat.
    */
   if (mkfifo(FIFONAME, 0666) < 0) {
      perror("mkfifo problem in server");
      exit(1);
   }

   /* sadece emin olmak icin yetkileri tekrar degistirelim */
   if (chmod(FIFONAME, 0666) < 0)
   {perror("chmod problem in server"); exit(1);}

   /*
    * okuma icin pipe ac.
    */
```

```
   if ((fd = open(FIFONAME, O_RDONLY)) < 0) {
      perror("open problem in server");
      exit(1);
   }

   /*
    * dosya sonuna kadar oku ,
    * ve okudugunu ekrana yaz.
    */
   while ((n = read(fd, buf, sizeof(buf))) > 0)
      write(1, buf, n);

   close(fd);
   printf ("SUNUCU SONLANDI…\n");
   exit(0);
}


/*  ISTEMCI  PROGRAM*/
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

#define FIFONAME    "myfifo"  /* Sunucudaki pipe ile ayni isim olacak*/

int
main(void)

{
   int n, fd;
   char buf[1024];

   printf ("ISTEMCI BASLAR…\n");

   /*
    * yazmak icin mevcut pipe ac.
    * sunucu zaten yaratmisti.
    */
   if ((fd = open(FIFONAME, O_WRONLY)) < 0) {
      perror("open problem in client");
      exit(1);
   }

   /*
    * Klavyeden metin oku
    * ve bu metni pipe a yaz. Metin Ctrl/d ile sonlandirilmali
    */
   while ((n = read(0, buf, sizeof(buf))) > 0)
      write(fd, buf, n);

   printf ("ISTEMCI SONLANDI…\n");
   close(fd);
   exit(0);
}
```

2. Sunucuyu arkaalan prosesi olarak başlatın (herhangi bir parametre olmadan). Arkalan prosesin ID 'sini göreceksiniz. Dizini listeleyin ve verdiğiniz isimde pipe'ın yaratıldığını görün.

3. İstemci prosesi herhangi bir parametre vermeden başlatın. Klavyeden bir mesaj girin ve Ctrl/d ile sonlandırın. Mesajınızın bir kopyasını ekranda göreceksiniz. Bu mesajlar arkaalan da pipe aracılığıyla sunucuya gönderilecektir.

4. Sunucuyu önalan prosesi olarak başlatın. Sonra uzaktan telnet/ssh yada ssh problemi yaşıyorsanız yerelden ikinci bir terminal açınız. **Şu an biri sunucu diğer istemci için iki tane sanal terminaliniz oldu.** İkinci terminalden, istemci prosesi başlatın ve mesajınızı girin ve Ctrl/d ile girişi sonlandırın. Sunucu terminalinizde aynı mesajın geldiğini görün.

5. Adım 2-3' ü tekrar edin (daha önce çalıştırılan sunucunun sonlandığından emin olunuz. Bunun is **ps** komutunu kullanabilirsiniz. Proses yaşıyorsa **kill 9 pid** komutu ile öldürünüz). Fakat, bu kez istemci proses klavye yerine, kaynak bir dosyadan okuyacak şekilde komut satırından yönlendirme yapınız (C kaynak kodunuzu verebilirsiniz). Sunucu terminalinde ne görüyorsunuz?

6. Her iki programı, pipe isimlerini program parametresi alacak şekilde güncelleyin ve adım 2- 3 'ü tekrar edin. Yada alternatif olarak ekteki programları (**fifo-cln.c** ve **fifo-srv.c)** kullanınız

7. Sunucu ve istemci arasında iki pipe kullanarak iki yönlü bir iletişim sağlayan iki programı(size verilen **server.c ve client.c)** programlarını analiz ediniz. Pipe isimleri her iki programın parametreleri olarak verilmiştir. Pipeları yaratma ve silme işini sunucu üstlenmiştir. Akış şu şekildedir: sunucu önce başlar(parametrelerle), istemciden istek mesajı için bekler, istek mesajı gelince ona karşılık bir yanıt mesajı gönderir. Son olarak, istemcinin bye mesajı için bekler ki pipe ları silebilsin. İstemci de sunucuya istek gönderir ve cevabını alınca, işinin bittiğini bildirmek için bye mesajını sunucuya gönderir.

8. Dizini kontrol edin, eger pipe halen varsa silin.