Dow Jones Sorusu

Bu soru birkaç yıl önce sınavlarımdan birinde vardı. Bu derste soruyu ve cevabını ayrıntısıyla ele alacağım. Bu da size **fork(), pipe(), signal()** ve soketler için iyi bir uygulama olacak.

Soru

Farzedin ki Dow Jones Sanayi Şirketi bir borsa serverini desteklemeye karar veriyor. Bu şirkette o andaki borsa fiyatlarını inceleyen bir kişi vardır. Server **stocks.downones.com** da 200 portluk bir soket görevi yapar. Ne zaman bir müşteri bu sokete bağlanırsa o anda kişinin incelediği borsa fiyatları müşteriye gönderilecek. Bu müşteri çıkıncaya kadar devam eder.

Müşteri **telneti** arayarak bu servere açık bir şekilde açık bir şekilde bağlanabilecektir:

UNIX> telnet stocks.dowjones.com 200
...
.
Dow jones sanayi şirketinde yazın çalışan birisiniz ve işiniz de bu serveri yazmak. Neyseki CS360 ı aldığınız için bunu yapmak sadece birkaç saatinizi alacaktır böylece yazın geri kalanını ödeme yapılarak çalışıyormuş gibi geçirebilirsiniz.
Bu serverin merkezinde **broadcast_info()** alt programı vardır.

```
broadcast_info(Dlist clients)
{
char line[100];
/* Read in stock price */

while(fgets(stdin, line, 100) != NULL) { /* from standard input */
  for (tmp = clients->flink; tmp != clients; tmp = tmp->flink) {
   write(tmp->val, line, strlen(line)) { /* Write stock prices */
} /* to every client */
}
}
```

1.Bölüm 7 puan

Eğer müşteri çıkarsa, server müşterinin soketini yazmaya çalıştığında **SIGPIPE** sinyali üretilir. Onu değiştirmek veya onunla doğru şekilde çalışmak için üstteki altprogramı değiştirin. Üstteki prosedürleri nasıl uygun görüyorsanız o şekilde değiştirebilirsiniz. Bu küresel değişkenleri içerir.

Bölüm 2:8 puan

İstemci alımına katılım biraz aldatmacalı bir konudur. Varsayalım ki bizim sunucu işlemi ana işleme p geçidi ile açık ve s soketine servisli olsun.Ana işlem ne zaman olursa olsun istemcinin katılmak istediğini tanır.(Bölüm 3'te göreceksiniz) Ne zaman istemci katılmak istesin,ana işlem sunucuya geçit üzerine yeni hat gönderir.Sunucu daha sonra accept_connection(s) yapar ve accept_connection(s)'u tanır ardından bağlantı ile geri döner.

Yüklemeyi çalıştırmak için broadcast_info()'yu güncelleştirin.Çalıştırmaktan kastetmek istediğim bir

şeyin bir kerede iki kez yapılması— hatların standart girişlere eklenmesini beklemek(bu hatlar istemcilere yayın olarak kullanılır)ve yeni istemcileri beklemek için geçit üzerinden açıklanır.
Şimdi aşağıdaki gibi görülmelidir;

```
broadcast_info(int *p, int s, Dlist clients)
{
...
}
```

Broadcast_info hala SIGPIPE'yi yakalamalı ve doğru şekilde başa çıkmalıdır.

Bölüm 3 :10 puan

Sunucu için kodların geri kalanı buradadır(Ana işlemi de kapsar).Açıklayın,bu kod nasıl çalışıyor? Özellikle bunlar nasıl başlatıldı? İstemciler katılırsa ne olur(telnet ile birlikte? İstemciler çıkış yaparsa ne olur?

```
main()
{
int mainsock, secondsock, i, j;
int p[2];
Dlist clients;
char c;
pipe(p);
if (fork() == 0) {
 secondsock = serve_socket("stocks.dowjones.com", 5000);
 clients = make_dl();
 close(p[1]);
 broadcast_info(p, secondsock, clients);
 exit(0);
} else {
 mainsock = serve_socket("stocks.dowjones.com", 200);
 close(0); close(p[0]); while(1) {
 i = accept_connection(mainsock);
 if (fork() == 0) {
  close(p[1]);
  j = request_connection("stocks.dowjones.com", 5000);
  while(read(j, &c, 1) != 0) write(i, &c, 1);
   exit(0);
 } else {
  close(i);
  write(p[1], "\n", 1);
 } } }  }
```

Bölüm 4: 5 puan
CEVAP
Bölüm 1

SIGPIPE'yi yakalamanın ve mevcut işlemci dosya tanımlayıcısını istemci listesinden silmenin püf noktası buradadır.İşte işe yarayacak kodlar:

```
int Sigpipe_caught;
pipe_handler()
{
Sigpipe_caught = 1;
}
broadcast_info(Dlist clients)
{
char line[100];
signal(SIGPIPE, pipe_handler);
while(fgets(stdin, line, 100) != NULL) {
 for (tmp = clients->flink; tmp != clients; tmp = tmp->flink) {
  Sigpipe_caught = 0;
  write(tmp->val, line, strlen(line));
  if (Sigpipe_caught) {
   close(tmp->val); tmp = tmp->blink; dl_delete_node(tmp->flink);
  }
 }
}
}
```

Hem de ***tmp***'yi global yapabilirsiniz ve işleyici içinde silebilirsiniz. Dikkat edin düğümleri silme ve icabına bakma işlemini tmp puanlarını doğru yere silersiniz(düğüm silinmiş olanın gerisinde) öyleki ***for()*** döngüsü doğru yere gidecektir.

BÖLÜM 2

Şimdi , aynı anda iki tane dosya tanımlayıcı okumak için ***select()*** çağrıları gereklidir ; bunlar ***standart input*** ve **pipe** 'dır. Eğer ***select()*** çağrısı kullanılarak ***standart input*** ile okuma yapmak istersek, okunan bu bilgi yayın hattından hazır olarak geri döner.Ancak, ***select()*** çağrısı kullanılarak ***pipe*** ile okuma yapmak istersek iki durum vardır. Birincisi ***pipe*** ile okunması gereken bir bilgi (satır) olduğu ve bağlantı için bekleyen bir kullanıcı olduğu bilinmelidir. Bu durumda ***accept_connection(s)*** çağrısı gerekir ve kullanıcı listesi ile ilgili dosya tanımlayıcısı ortaya çıkar. Kodumuz aşagıda;

```
Broadcast_info(int *p,  int s,  Dlist clients) // broadcast_info() çağrı parametreleri veriliyor.
{
Dlist tmp;
fd_set readset;
int i;
char line[100];

signal (SIGPIPE , pipe_handler);

  while (1)
```

```
  {
    FD ZERO (&readset);
    FD SET(0, &readset);
    FD SET(p[0], &readset);

   Select( p[0]+1 ,&readset , NULL, NULL ,NULL);
   İf (FD ISSET(p[0], &readset))
   {
    if (read(p[0], line, 1) == 0) exit(0);
    i = accept_connection(s);
   dl_insert_b(clients, i);
   }
    else {
     if (fgets(line, 100, stdin) == NULL)
     {
      fprintf(stderr, "Stdin closed.        Bye\n");
       exit(1);
     }
    for (tmp = clients->flink; tmp != clients;
        Sigpipe_caught = 0;
        write(tmp->val, line, strlen(line));
        if (Sigpipe_caught)
    {

       close(tmp->val); tmp = tmp-

     >blink; dl_delete_node(tmp-
    >flink);
       }
     }
  }
}
}
```

BÖLÜM 3

Bu , çok önemli kod parçası bölümüdür. İlk program pipe' a set'lenir (ayarlanır) ve alt programlardan koparılır. Ana program, port 200 soketini ve döngüleri takip eder.Port 200 soketi bağlantı için bekler. Bir bağlantı aldığında, ***pipe***'in alt programına bağlantı boyunca satır (bilgi) gönderir.Diğer alt programların sürecini kapatır, port 5000 üzerinden ***request_connection()*** çağrılarını alır. Alt programdan bağlantıyı aldıktan sonra, telnetten çağırdığı kullanıcı işlemleri ile port 5000 üzerindeki bağlantılarından byte'lar gönderir. İlk alt program port 5000 üzerinde soketi kurar ve ***broadcast_info()*** çalışır.

Böylece, bir kullanıcı katıldığında, ana program **accept_connection()** 'dan geri döner, ***broadcast_info()*** işlemlerine bilgi(satır) gönderir ve port 5000 üzerinden ***broadcast_info()*** işlemine bağlanan alt program kapatılır ve kullanıcı işlemlerinden byte'lar alınıp verilir.Kullanıcı bağlantıdan ayrıldığı zaman, alt program byte alıp vermek suretiyle tekil bir ***SIGPIPE*** sinyali oluşturur ve bağlantı sona erer. Sonra ***broadcast_info()*** işlemleri ***SIGPIPE*** sinyali üretir ve bu şekilde anlaşırlar.

Neden bu karmaşık süreç yapısına ihtiyaç duyarız? Çünkü ***broadcast_info()*** kendi işlemi için ***select()*** çağrısı yapamaz, bunun için isteklerin ya ***standart input*** üzerine gelmesini bekler yada port 200 üzerinde bir bağlantı talebinde bulunması gerekir. Bu nedenle port 200 üzerinde beklemek için

extra olarak ana proses eklenir.. Bu ana proses ile yapılan bağlantıdan ve ***broadcast_info()*** servis bitleri ile bağlanmak istememizden dolayı extra proses çatallanmasını kapatır
Bu nedenle biz port 200 üzerinde beklemek için extra olarak ana proses ekleriz. Bu ana proses ile yapılan bağlantından ve broadcast_info() servis bitleri ile bağlanmak istememizden dolayı extra proses çatallanmasını kapatır.

Bunu kullanmaktan daha iyi bir olasılık var, listen() kullanmak ya da belki de select(), ama ben ,Bu çözümün bize Unixte sıklıkla yaptığımız son şeylerin çeşidini, nisbeten daha basit bir şekilde, vereceğini bilmiyorum.Verimsiz mi?Evet. Bu proses bize neler yapmamamızı gösteren şeylerin servis baytlarını içerir fakat biz bunu yapıyoruz çünkü hayatı kolaylaştırıyor. Eğer biz işletim sistemi tarafından destekenen iyi bir thread tabanlı progama modelini yapabilirsek, biz bunu daha kolay bir şekilde kullanabiliriz.

BÖLUM 4

Burada her bir istemci başına byte alış verişi yapmak için çatallama kapatan bir proses var.Bu proses bittiği zaman (via SIGPIPE) ,bu bir zombi haline dönüşür çünkü bunun ana programı hala hayatta ve hiçbir zaman wait() i çağırmaz. Her halükarda bu accept_connection() in geri dönmesini beklediğinden bu zamana kadar , bu ebeveyn çocukları öldüğünde ve gelişigüzel bir şekilde wait() çağrısı yapamadığını gerçekten bilemez. Bu nedenle ,bu sorunu gidermek için, bir class içerisinde tanımlanan bir hileyi kullanabiliriz.--(byte-shuttling yapan) torunu çatalamaya kapatılan çocukta çatallamaya kapatılır ve sonra anında geri döner. Çocuk aniden geri dönüş yaptığından dolayı ana proses bunu bekleyebilir ve torun **init** tarafından kalıtım aldırılmış duruma gelecektir ve bu öldüğünde zombi olmayacaktır.

(while(1) durumunda başlayan) yeni kod içeriği aşağıdadır:

```
while(1) {
i = accept_connection(mainsock);
if (fork() == 0) {
if (fork() == 0) {
close(p[1]);
j = request_connection("stocks.dowjones.com", 5000);
while(read(j, &c, 1) != 0) write(i, &c, 1);
exit(0);
} else {
exit(0);
}
} else { wait(&statusp); close(i);
write(p[1], "\n", 1);
```