BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ

İŞLETİM SİSTEMLERİ - KABUK PROJESİ

GÜZ 2020

Veriliş tarihi: 12 Kasım 2020 Perşembe
Teslim tarihi: 20 Aralık 2020 Pazar, Saat 23:59

- **Ödevle ilgili sorunlar için Arş.Grv.Dr. Deniz Balta, email: ddural@sakarya.edu.tr ile irtibata geçiniz.**

Projenin tanımı:

Bu projede, basit bir komut satırı yorumlayıcısı veya kabuk geliştirmeniz isteniyor. Kabuk, bir komut yazdığınızda girdiğiniz komutu yürüten ve tamamlandığında daha fazla kullanıcı girdisi isteyen ve onları da işleyen bir prosestir.

Bu projenin amaçları şöyle listelenebilir:

1. Kendinizi Linux programlama ortamına alıştırmak.
2. C dilinde programlama becerilerinizi geliştirmek.
3. Kabuk (shell) programının işlevlerini anlamak.
4. Proseslerin (süreçlerin) işletim sisteminde nasıl ele alındığını öğrenmek.

Özet:

Geliştireceğiniz kabuk, Linux'te her gün çalıştırdığınız kabuğa benzer olmakla beraber çok daha basit olacaktır. Linux işletim sisteminde çok sayıda kabuk programı vardır; örneğin, "$ echo $SHELL" yazarak sisteminizde hangi kabuğu çalıştırdığınızı öğrenebilirsiniz. Diğer kabuklar hakkında daha fazla bilgi edinmek için (sh, tcsh, zsh veya bash gibi) manuel sayfalarına bakabilirsiniz. Bu projede geliştireceğiniz kabukta çok fazla işlev olmayacak, ancak en azından aynı anda birden fazla komutu çalıştırabilmeniz gerekmektedir.

Geliştireceğiniz kabuk iki şekilde çalışabilmelidir: interaktif (etkileşimli) ve batch (toplu). İnteraktif modda, ekranda bir prompt görüntülenmelidir, prompt seçtiğiniz herhangi bir kelime olabilir. Kabuk kullanıcısı prompt yanına en fazla 512 karakterden oluşan bir komut yazabilmeli ve kabuk bunu okuyabilmelidir. Batch (toplu) modda ise, kabuk komut satırında ismi yazılan dosyanın içindeki komutları okur ve icra eder. Batch modda, dosyadan okunan her satır (komut) çalıştırmadan önce ekrana yazdırmalıdır. Bu özellik kabuk programındaki hataları ayıklamada ve programlarınızı test ederken bize yardımcı olacaktır. İnteraktif veya toplu modda çalışırken kabuğunuz bir satırda "quit" komutunu gördüğünde çıkmalıdır. Bunun dışında batch dosyasının sonuna ulaştığında veya kullanıcı "Ctrl-D" tuşlarına basarsa yeni komut almayı durdurmalıdır. Kabuk, daha önce başlatılan ve çalışan tüm komutlar sona erdikten sonra çıkmalıdır.

İnteraktif modda komut girişinde veya batch dosyasında, her satır ";" ile ayrılmış birden çok komut içerebilir ve ";" ile ayrılmış komutların her biri eşzamanlı (paralel) olarak çalıştırılmalıdır. Kabuk, tüm bu komutların yürütülmesi tamamlanana kadar bir sonraki promptu yazdırmamalı veya daha fazla girdi almamalıdır (yardım: "wait()" ve / veya "waitpid()" sistem çağrıları burada yararlı olabilir). Aşağıdaki örnek ile verilen komutlar için kabuk çalışmalıdır (Kabuk programının adı "shell" olsun):

İnteraktif çalışma örneği:

```
$ shell

prompt>
prompt> ls
prompt> /bin/ls
prompt> ls -l
```

```
prompt> ls -l ; cat file
prompt> ls -l ; cat file ; grep foo file2
```

Batch çalışma örneği:

```
$ shell [batchFile]
```

Komutların bulunduğu dosyanın adı “kmt.txt” olsun ve bu dosyanın içinde aşağıdaki komutlar olsun;

```
ls
/bin/ls
ls –l ; cat file
ls –l ; cat file ; grep foo file2
```

Linux kabuk üzerinde sizin programınızın batch modunda çalıştırılması için:

```
$ shell kmt.txt
```

girilmelidir. Bu durumda “kmt.txt” dosyasının içindeki kabuk komutları çalışmalıdır.

Her iki modda da son satırda görünen, “ls –l”, “cat file” ve “grep foo file2” komutları eşzamanlı (paralel) olarak çalışmalıdır. Eşzamanlı çalışmadan dolayı ekran çıktılarının birbirine karışması normaldir.

Savunmaya dayalı programlama (**defensive programming**) işletim sistemlerinde önemli bir kavramdır. Bir işletim sistemi, hatayla karşılaştığında kolayca başarısız olmamalıdır, bunun sağlanabilmesi için önce girdilerin tüm parametreleri kontrol edilmelidir. Bu projede geliştireceğiniz C programınız asla çökmemeli, süresiz olarak askıda kalmamalı veya zamanından önce sonlanmamalıdır. Programınız tüm girdilere makul bir şekilde yanıt vermelidir. Burada "makul", anlaşılır bir hata mesajı yazdırmak ve duruma bağlı olarak işlemeye devam etmek veya çıkmak anlamına gelmektedir.

Örneğin, aşağıdaki verilen durumlar hata olarak değerlendirilmelidir. Bu durumlarla karşılaşıldığında, kabuğunuz ekrana bir mesaj yazdırmalı (stderr'e) ve nazikçe çıkmalıdır:

- Kabuk programınız için yanlış sayıda komut satırı argümanı.
- Batch dosyası mevcut değil veya açılamıyor.

Aşağıdaki durum için ise kullanıcıya (stderr) bir mesaj yazdırmalı ve işlemeye devam etmelidir:

- Komut yok veya komut yürütülemez.

Kabuk çok uzun komutlar için hata mesajı yazdırmalı ve işlemeye devam etmelidir. Örneğin aşağıdaki durumda verilen mesajı yazıp işlemeye devam etmelidir:

- Çok uzun bir komut satırı (bu proje için '\n' dahil 512 karakterden fazla).

Hata olmayan aşağıdaki senaryoları da kabuk işleyebilmelidir (yani kabuk hata mesajı yazmamalıdır):

- Boş bir komut satırında ENTER’a basılması.
- Komut satırında fazladan karakter boşlukları (white space) olması.
- Batch dosyasında “quit” komutu olmadan sonlanmış olması veya interaktif modda kullanıcı 'Ctrl-D' ye girmesi.

Hiçbir durumda, herhangi bir girdi veya herhangi bir komut satırı biçimi, kabuk programınızın çökmesine veya zamanından önce çıkmasına neden olmamalıdır. Garip biçimde oluşturulmuş komut satırlarını (örneğin, noktalı virgüller arasında hiç komut içermeyen “; ;” satırlar) nasıl yürüteceğinizi dikkatlice düşünmelisiniz. Bu durumlarda, bir uyarı mesajı yazdırmayı ve/veya komutların bazı alt kümelerini çalıştırmayı siz belirlemelisiniz. Ancak her durumda kabuğunuz çalışmaya devam etmelidir.

```
prompt> ; cat file ; grep foo file2
prompt> cat file ; ; grep foo file2
prompt> cat file ; ls -l ;
prompt> cat file ;;;; ls -l
prompt> ;; ls -l
prompt> ;
```

Yol gösterme:

Kabuk programı temelde bir döngüden ibarettir. İnteraktif moddaysa tekrarlı olarak komut promptunu ekrana yazdırır, sonra girişi ayrıştırır, giriş satırında belirtilen komutu/komutları yürütür ve ön plandaysa komutun bitmesini bekler. Bu işlem kullanıcı "quit" yazana veya girişini bitirene (CTRL-D) kadar tekrarlanır.

Kabuk, her yeni komut için yeni bir proses oluşturacak şekilde yapılandırılmalıdır. Komutalr için yeni proses oluşturmanın iki avantajı vardır: İlk olarak, ebeveyn kabuk prosesini yeni komutta meydana gelen hatalardan korur, ikincisi ise; eşzamanlılığın (paralelliğin) kolayca elde edilmesini sağlar. Yani, bu şekilde birden çok komut aynı anda başlatılabilir ve paralel tarzda yürütülmesi sağlanabilir.

Bu projede işleri sizin için basitleştirmek ve kodlamanızı kolaylaştırmak için bazı önerilerimiz olacak. Önerdiğimiz rutinler hakkında daha fazla bilgi bulmak için "man" komutunu kullanarak kılavuz sayfalarına bakmalısınız. Ayrıca, programınıza eklemeniz gereken başlık (header) dosyalarını da "man" komutu ile öğrenebilirsiniz.

Ayrıştırma (parsing):

Öneri olarak, komut girdilerini okumak için "fgets()" kullanabilirsiniz, bir dosyayı açmak için "fopen()" kullanabilirsiniz. Bu çağrıların döndüreceği hatalar için mutlaka geri dönüş kodunu kontrol etmelisiniz. Hatalı dönüşlerde sorunu görüntülemek için "perror()" kullanılabilirsiniz. "strtok()" fonksiyonu komut satırını ayrıştırmak için kullanılabilir. Bu komut boşlukla veya tab ile ayrılmış bir komut kelimelerini veya argümanları ayıklamak için kullanılabilir.

Komutların icrası:

Komutların icrası için "fork()", "execvp()" ve "wait()"/"waitpid()" sistem çağrılarını kullanabilirsiniz.

fork() sistem çağrısı yeni bir proses oluşturur. Fork'tan sonra, kodunuz içinde iki proses çalışacaktır. Fork'un dönüş değerine bakarak yavru prosesi ebeveynden ayırt edebilirsiniz; yavru proses fork geri dönüş değerini 0 olarak görür, ebeveyn ise yavru prosesin pid'sini görür, yani sıfırdan büyük bir değeri görür.

Exec ailesinde çeşitli prototipler veya fonksiyonlar var; bu proje için execvp() kullanmanız gerekir. execvp() başarılı olursa geri dönmez; geri dönerse, bir hata vardır (örneğin, komut yok gibi). En zor kısım ise, argümanları doğru bir şekilde belirlemektir. Exec'in ilk argümanı tam yol ile çalıştırılması gereken programın adıdır. İkinci ve sonraki argümanlar ise programa komut satırından aktarılan parametrelerdir, örneğin:

```
int main(int argc, char *argv[]){ }
```

programını derledikten sonra yürütülecek dosyanın adı "foo" olsun. Eğer program:

```
$ foo 35 45
```

Şeklinde çalıştırılırsa, argv[0] → "foo", argv[1] → "35" ve argv[2] → "45" olacaktır.

Önemli: argüman listesi bir NULL gösterici ile sonlandırılmalıdır; yani argv [3] = NULL. Bu işlemi doğru yapıp, yapmadığınızı dikkatlice kontrol etmeniz şiddetle tavsiye edilir.

"wait()"/"waitpid()" sistem çağrıları, ebeveyn prosesin yavru proseslerin işlemlerini bitirinceye kadar beklemesine izin verir. Daha fazla ayrıntı için "man" sayfalarını okuyun.

Neler geri döndürülecek:

Bu proje için üç farklı dosyayı teslim etmeniz gerekir:

- Kaynak kodunuz (lütfen binary dosyaları (*.o) veya icra edilebilir dosyaları göndermeyin)
- Kaynak kodunuzu derlemek için bir Makefile
- Kodunuz hakkında açıklayıcı bilgiler içeren bir OKUBENİ (README) dosyası.

README dosyası aşağıdaki dört bölümü içermelidir:

1. Proje adı, proje grup elemanlarının adları, soyadları ve öğrenci numaraları
2. **Tasarıma genel bakış:** Kodunuzun genel yapısını ve tüm önemli yapıları açıklayan birkaç paragraf.
3. **Özel durumlar:** İstenenlerdeki herhangi bir belirsizliği nasıl ele aldığınızı açıklayın. Örneğin, bu proje için, kabuğunuzun noktalı virgül arasında komut içermeyen satırları nasıl işleyeceğini açıklayın.
4. **Bilinen hatalar veya sorunlar:** Tamamlayamadığınız kısımlar veya düzgün çalışmadığını bildiğiniz tüm özelliklerin listesi.