İŞLETİM SİSTEMLERİ PROJE ÖDEVİ
2019 GÜZ

Veriliş tarihi: 22.10.2018 Salı
Teslim tarihi: 30 Kasım 2019 Cumartesi 23:59
Teslim yeri: SABİS üzerinden
Notlar:
1) Ödevle ilgili sorunlar için Arş.Grv.Dr. Deniz Balta, email: ddural@sakarya.edu.tr ile irtibata geçiniz.
2) Proje C programlama dilinde ve Linux ortamında geliştirilecektir.

TANIM
Bu ödevin amacı, proses (süreç) yönetimi, I/O (Giriş/Çıkış) ve Linux signal kullanımının temellerini öğretmektir. **Kabuk** (shell) derste de anlatıldığı gibi komut satırı yorumlayıcısıdır ve standart girişi okur, komutları alır, çözümler, karşılık gelen programları (varsa) çalıştırır ve kullanıcı tarafından sonlanana kadar çıktılarını görüntüler. Projede göreviniz Linux ortamında çalışacak basit bir **Kabuk** uygulaması geliştirmektir.

Linux Kabuk (Shell)
Genel olarak kabuk için komut satırının amacı, komut istemi (prompt) görüntülemek ve sonra kullanıcı tarafından girilen metnin okunmasıdır. Girilen metin, tam olarak tek bir komut veya bir dizi komut içeriyor olabilir. Komut bazen kabuğun kendi ortamı içinde tanımlanan bir komutun adıdır (builtin komut) veya bir yürütülebilir dosyanın tam yolu (adı ile birlikte) ve ardından isteğe bağlı argümanlarıdır. Yürütülebilir dosya adı ve bağımsız değişkenler karakter adedi/türleri ile sınırlandırılmıştır. Bu nedenle yürütülebilir dosya adları ve bağımsız değişkenler boşluk içermeyen dizelerdir.

Yönlendirme:
Kabuk, komutların standart çıktı ve girdisini yeniden yönlendirebilir. Örneğin **"> echo 12 > test.txt**" girerseniz, echo işleminin çıktısı, yönlendirme nedeniyle test.txt dosyasına yazılır. Ardından, **"> ./p1 < test.txt**" komutunu girdiğinizde ise test.txt dosyasındaki veri girdi olarak okunarak, artımdan sonra ekrana 13 yazdırılır.

```
> echo 12 > test.txt
> ./p1 < test.txt
13
```

(**Not:** Burada standart girişten aldığı tamsayıyı bir arttıran "p1"adlı yürütülebilir dosyanın, test.txt ile aynı dizinde olduğu varsayılmıştır. Test etmek için p1'i elde etmek isterseniz, aşağıda verilen örnek programı kullanabilirsiniz.)

```
// Örnek program
#include<stdio.h>
int main() {
  int i;
  scanf("%d", &i);
  i++;
  printf("%d\n", i);
  return 0;
}
```

Boru (pipe):
Her komutun arasında sınırlayıcılar kullanılarak, standart bir kabuk üzerinde sıralı komutlar yürütmek de mümkündür. Bu projede **boru** (|) ve **noktalı virgül** (;) sınırlayıcılarına odaklanacağız. **Boru** durumunda, her komutun standart çıktısı, ondan sonra gelen komutun standart girişine bağlanır. Aşağıdaki örnekte echo çıktısı p1'e pipe ile bağlandığında, ekranda 13 görülür.

```
> echo 12 | ./p1
13
```

Çok sayıda bağımsız komut:
Kabuktan çok sayıda bağımsız komut çalıştırılmak istendiğinde ";" kullanılır, bu durumda prosesler arasında bir bağlantı yoktur. Komutlar soldan sağa doğru icra edilirler. Aşağıdaki örnekte önce 12 yazılır, 2 sn durulur ve sonra 13 ekrana yazılır.

```
> echo 12; sleep 2; echo 13
12
13
```

Noktalı virgül ile komutlar girildiğinde, her komut için yönlendirme ve pipe mümkündür. Arkaplan çalışma ise (&) sembolü ile gerçekleştirir. Alt-kabuk komutları (sub-shell commands) parantez ile oluşturulur, yani "(", ")". Alt-kabuk komutları için de yönlendirme, pipe veya arkaplan çalışma uygulanabilir. Örnek:

```
> (echo merhaba; echo dünya;) > t2.txt
> cat t2.txt
merhaba
dünya
```

Arkaplan işleme, komutların tamamlanmalarını beklemeden ve hemen her **arkaplan** komutundan sonraki komutun eşzamanlı yürütülmesidir. **Arkaplan** yürütme, komutun sonuna bir ve işareti **(&)** ekleyerek gerçekleştirilir.

İSTENENLER:
Puanlama:

Yapılacak İşler	**Puan**
1) Prompt	**5**
2) Built in komut	**5**
3) Tekli komut icrası	**10**
4) Giriş yönlendirme	**15**
5) Çıkış yönlendirme	**15**
6) Arkaplan çalışma	**10**
7) Boru (pipe)	**20**
8) Çok sayıda bağımsız komut, alt-kabuk kom.	**20**
TOPLAM	**100**

1) **Prompt (5 Puan):**

Başladıktan ve her komutun tamamlanmasından sonra veya her arka plan komutunun hemen ardından, kabuk ">" basmalıdır. Print buffer'ı boşaltmak için aşağıdaki veya eşdeğer bir kod kullanılmalıdır:

```
printf("> ");
fflush(stdout);
```

2) **Quit (Built-in komut) (5 Puan):**

Kabuk, quit komutu ile sonlandırılmalıdır.

```
> quit
```

Tek istisna, önceki komutun arka planda hala çalıştığı zamandır. Bu durumda, programınız yeni komutlara yanıt vermeyi durdurmalı, tüm arka plan işlemlerinin bitmesini beklemeli ve ilişkili bilgileri yazdıktan sonra hemen sonlandırmalıdır.

3) Tekli komutlar (10 Puan):

Kabuk bağımsız değişkenlerle veya bağımsız değişkenlerle tek bir komutu okuduğunda komutu çalıştırmalı, komut tamamlanıncaya kadar engellemeli ve ardından istemine geri dönmelidir. Kabuk bir alt proses oluşturmalı ve onu komutla belirtilen programa dönüştürmelidir. Örnek:

```
> ls -l
```

Yukarıdaki örnekte, programınız kendisini bir ls prosesi haline getiren bir alt süreç oluşturmalıdır. Programınız daha sonra ls işleminin tamamlanmasını beklemeli ve daha sonra komut prompt'unu görüntüleyerek yeni bir komut beklemelidir.
İlgili LINUX system çağrıları: fork, exec, wait ve/veya waitpid

4) Giriş yönlendirme (15 Puan):

Girişe "komut < GirişDosyası" şeklinde yönlendirme (redirection) yapan bir komut girildiğinde, program verilen komutu çalıştırmadan önce alt prosesin standart girdisini verilen giriş dosyasına yönlendirmelidir. Hiçbir girdi dosyası yoksa, "**Giriş dosyası bulunamadı"** yazdırılmalıdır.
Örnek:

```
> cat < file.txt
file.txt'nin içinde ne varsa ekrana yazdırılır.
> cat < nofile.txt
nofile.txt giriş dosyası bulunamadı.
```

İlgili LINUX system çağrıları: dup2

5) Çıkış yönlendirme (15 Puan):

"komut > çıkışDosyası" şeklinde yönlendirme (redirection) yapan bir komut girildiğinde, program alt prosesin standart çıktısını çıkışDosyası'na yönlendirmelidir.
Örnek:

```
> cat file1 > file2
```

İlgili LINUX system çağrıları: dup2

6) Arkaplan çalışma (10 Puan):

Bir komut '&' ile bittiğinde arka planda çalıştırılmalıdır. Komutun tamamlanması için kabuk engellenmemelidir ve hemen komut prompt'u görüntülenmelidir. Daha sonra kullanıcı yeni komutlar girilebilmelidir. Bununla birlikte, arka plan işlemi sona erdiğinde, kabuk çıkış durumunu proses kimliği ile birlikte, aşağıdaki bilgileri kullanıcıya bildirmelidir.

```
[pid] retval: <exitcode>
```

Örnek:

```
> sleep 5 &
> cat file.txt
file.txt'nin içindekiler ekrana yazdırılır.
> [24617] retval: 0
```

Burada, ikinci prompt kabuk tarafından derhal yazdırılır, oysa pid ve dönüş değeri arka plan işleminin sonlandırılmasından sonra yazdırılır.

Arka planda çalışan bir proses varken, kabuğa bir **quit** komutu girilirse; prompt görüntülenir ve yeni komut girişi durdurulur. Sonra tüm arka plan işlemleri sona erene kadar beklenir. Daha sonra ilgili bilgiler bastırılır ve derhal çıkılır.

İlgili LINUX system çağrıları: sigaction, WEXITSTATUS

7) Boru (pipe) (20 Puan):

Birden fazla komut birbiriyle borular aracılığıyla bağlandığında, i. komutunun çıkışı (i + 1) inci komutunun girişine beslenir. Son komutun çıkışı, dosyaya yönlendirilmediği sürece standart çıktıya bağlanır. İlk komutun girişi, işlemin giriş gerektirmesi durumunda standart girdiden başka bir dosyadan da yönlendirilebilir. Örnek:

```
> find /etc | grep ssh | grep conf
```

Bu görevde, komut promptu görüntülenmeden önce, pipe içindeki tüm komutların sonlanması gerektir.
İlgili LINUX system çağrıları: pipe, dup2. Ayrıca proseslerin stdin ve stdout dosya descriptörlerine de bir göz atın.

8) Çok sayıda bağımsız komut ve alt-kabuk komutları (20 Puan):

Komutlar noktalı virgül ile ayrılmışsa, i. komutun tamamlanmasından sonra (i+1). komut yürütülmelidir. Örneğin:

```
> echo başladı ; sleep 1 ; echo bitti
başladı
(1 saniye sleep)
bitti
```

Daha önce belirtildiği gibi I/O yönlendirmesi yapılabilmeli:

```
> echo message > message.txt ; sleep 1 ; cat < message.txt
message
```

Kolaylık için: Ardışıl komutların ve arkaplan yürütmelerin ardışıl olarak (noktalı virgül ile) olmayacağını varsayınız. Örnekler:

```
> ( echo MERHABA ; sleep 2 ; echo DÜNYA) > out.txt
> cat out.txt
MERHABA
DÜNYA
> ( cat | tr /A-Z/ /a-z/ ) < out.txt
merhaba
dünya
> ( echo MERHABA ; sleep 2 ; echo DÜNYA ) | ( cat | tr /A-Z/ /a-z/ )
merhaba
dünya
> ( echo "TIR" > test.txt ; cat < test.txt ; sleep 5 ; echo ali ) &
> TIR
(5 saniye sleep)
ali
[28201] retval:0
```

İlgili LINUX sistem çağrıları: pipe, dup2, fork, exec family, wait and/or waitpid

---

Geri döndürülecekler:

1) Kaynak dosyalar: *.c ve *.h dosyaları
2) Makefile
3) Readme dosyası

Tekrar hatırlatalım, proje C programlama dilinde ve Linux ortamında geliştirilecektir.

Kolay gelsin…