İşletim Sistemleri Komut Satırı Kabuğu (Shell) Ödevi

Hazırlayanlar

Yavuz Selim Say / B221210086

Mehmet Çapar / B201210102

Muhammed Yusuf Öngel / B221210100

Yiğit Özpolat / B221210004

Ahmet Yunus Çavuş / B221210082

Programın Amacı

Bu program bir **komut satırı kabuğu** (shell) işlevi görür. Kullanıcıdan komut alır ve bu komutları işletim sistemi seviyesinde çalıştırır. Aşağıdaki özellikleri içerir:

- **Yeniden yönlendirme** (input/output redirection): <,>,>>
- **Pipe desteği**: Komutları | kullanarak birbirine bağlama.
- **Arka plan işlemleri**: & işaretiyle belirtilen komutları arka planda çalıştırma.
- **Basit hata yönetimi**: Hatalı komutlarda kullanıcıya bilgi verme.

Kodda Kullanılan Önemli Fonksiyonlar ve İşlevleri

1. **detectCommand()**
    - Kullanıcının girdiği komut listesinin hangi işlemi yapacağını belirtir ve ona göre işlemler yaptırır.
2. **executeSingleCommand()**
    - Tek bir komutu çalıştırır. Örneğin: ls -l
    - /bin/ dizinindeki bir binary dosyayı çalıştırır (/bin/ls gibi).
    - Eğer komut başarısız olursa bir hata mesajı yazdırır.
3. **pipeFonk()**
    - Pipe (|) kullanan komutları işler. Örneğin: ls | grep txt
    - İki komut arasında bir pipe kurar. Ancak, iki veya daha fazla pipe kullanımı için sınırlama belirtilmiş.
4. **fileOutput()**
    - Çıkış yönlendirme işlemlerini gerçekleştirir (>, >>).
    - Yeni bir dosya oluşturur ya da mevcut dosyaya ekleme yapar.
5. **fileOutput()**
    - Giriş yönlendirme işlemlerini gerçekleştirir (<).
    - Var olan bir dosya üzerindeki sayıyı okur ve onu bir arttırıp ekrana yazdırır (increment).
6. execution
    - Arka planda bir programda çalışıyorsa farklı çalışmıyorsa farklı bir çalıştırma işlemi yapar.
7. pipeFonkOrder

- ; olan komutları çalıştırır.
- Her komut birbirinden farklıdır. En fazla 3 komut girilebilir. Komutlar içinde yönlendirme işlemleri yapılabilir.

,Programın Çalışma Prensibi

1. Program Başlangıcı

Shell uygulaması çalıştırıldığında:

- Program bir döngü içine girer ve sürekli kullanıcıdan komut bekler.
- Kullanıcıdan girdi almak için genellikle getline() veya fgets() gibi bir fonksiyon kullanılır.
- Kullanıcı girdisi alındığında, bu komut bir işlem olarak değerlendirilir ve işlenmek üzere parçalanır.

2. Komut Satırının Parçalanması

- Kullanıcının girdiği komut, kelimelere (token) ayrılır. Bu işlem genellikle boşluk veya özel karakterler (|, <, >) ile yapılır.
- strtok() gibi fonksiyonlar kullanılarak, komut ve argümanlar bir diziye bölünür:
  - Örneğin: "ls -l | grep .txt > output.txt" şu şekilde ayrılır:
    - Komut: ls
    - Argümanlar: -l
    - Pipe: |
    - Yönlendirme: > output.txt

3. Çocuk Süreç (Child Process) Oluşturma

- Her bir komut bir sistem çağrısı (fork()) kullanılarak yeni bir süreçte (child process) çalıştırılır.
- Ana süreç (parent process) ise child sürecin tamamlanmasını bekler (waitpid() ile).
- Yeni süreçte komut çalıştırılırken execvp() gibi bir fonksiyon kullanılır:
  - Bu fonksiyon belirtilen komutu ve argümanları çalıştırır.
  - Eğer komut geçersizse, bir hata mesajı döner ve shell yeni bir komut bekler.

4. Pipe (Boru) İşlemleri

- Pipe, iki sürecin birbirine veri aktarabilmesini sağlar:
  - Örneğin: ls | grep .txt komutu, ls çıktısını grep girişine yönlendirir.

- Pipe işlemleri için pipe() sistemi kullanılır:
    - İki süreç arasında bir boru hattı oluşturulur.
    - stdout (çıkış) diğer sürecin stdinine (giriş) bağlanır.
    - Her süreç kendi görevini tamamlar ve çıktılar boru hattı boyunca aktarılır.

5. Dosya Yönlendirme

- Kullanıcı, çıktı veya girdiyi bir dosyaya yönlendirdiğinde (> veya <), standart giriş/çıkış yeniden yapılandırılır:
    - dup2() fonksiyonu kullanılarak, stdin veya stdout belirtilen dosyaya bağlanır.
    - Örneğin:
        - ls > output.txt: Çıktıyı output.txt dosyasına yazdırır.
        - sort < input.txt: input.txt dosyasını giriş olarak kullanır.

6. Dahili Komutlar (Built-in Commands)

- Shell uygulaması bazı komutları doğrudan kendi içinde işler:
    - cd: Kullanıcıyı başka bir dizine yönlendirir. Bu işlem chdir() ile yapılır.
    - exit: Shell'den çıkış yapar.
    - pwd: Bulunduğunuz dizini ekrana yazdırır.
- Bu komutlar için yeni bir süreç oluşturulmaz; doğrudan ana süreçte çalıştırılır.

7. Hata Kontrolü

- Programın kararlılığını artırmak için:
    - Geçersiz komutları algılar ve kullanıcıya hata mesajı döner.
    - Dosya veya boru yönlendirmelerinde hatalı giriş/çıkış olursa kullanıcı uyarılır.

8. Döngüsel Çalışma

- Her işlem tamamlandıktan sonra shell, tekrar kullanıcı girdisi bekler.
- Kullanıcı shell'den çıkana kadar (exit komutu girilene kadar) bu döngü devam eder.

Mini Shell Çalışma Akışı (Şema)

1. **Kullanıcı Girdisi Alınır**
    - Örneğin: ls -l | grep .txt > output.txt

2. **Girdi Parçalanır**
    - Komut: ls
    - Argümanlar: -l
    - Pipe: |
    - Yönlendirme: >
3. **Çocuk Süreçler (fork) Çalıştırılır**
    - Her bir komut yeni bir süreçte çalışır.
4. **Pipe ve Yönlendirme İşlenir**
    - Çıktılar ve girdiler borular veya dosyalar arasında yönlendirilir.
5. **Sonuç Döndürülür**
    - output.txt dosyasına yazdırılır.
6. **Shell Döngüsü Devam Eder**
    - Yeni bir komut beklenir.

SONUÇ

Bu ödevde geliştirdiğimiz basit komut satırı kabuğu (shell) programı, işletim sistemlerinin temel işlevlerini kullanıcıya sunarak, komut satırında çalıştırılabilecek çeşitli komutları ve işlemleri destekler. Kullanıcı, bu shell aracılığıyla dosya yönlendirme, pipe (boru) işlemleri, arka plan komutları gibi işlemleri rahatlıkla gerçekleştirebilir. Programın temel işlevleri, komutları parçalayarak işlemlerini doğru bir şekilde yürütmek, pipe ve yönlendirme işlemlerini yönetmek, arka plan işlemleri ile daha verimli bir kullanım sağlamak ve kullanıcı hatalarına karşı bilgilendirme yapmaktır.