İşletim Sistemleri Ödevi

Berke Pite G221210013

Ahmet Yasin Pekdemir G221210075

Hasan Yasir Arslan B221210096

Üzeyir Emre Türkmen B221210381

Enes Soylu B221210081

Github Linki

Proje Tanımı

Bu proje, bir komut satırı yorumlayıcısı (kabuk) geliştirmeyi amaçlamaktadır. Kabuk, kullanıcıdan gelen komutları yorumlayarak uygun şekilde çalıştırır ve çıktılarını görüntüler. Bu süreçte, proses yönetimi, giriş/çıkış (I/O) yönlendirmesi ve Linux sinyallerinin kullanımı gibi konulara odaklanılmaktadır. Projede başlıca hedefler şunlardır:

1. **Proses Yönetimi:**
    - **Ana kabuk prosesi içerisinde yeni prosesler oluşturabilme.**
    - **İşlem sonlanımlarını ve çıkış kodlarını takip edebilme.**
2. **Komut Yorumlama ve İşletme:**
    - **Tekli ve ardışık komutları çalıştırma.**
    - **Borulama (pipe) ve noktalı virgül (;) sınırlayıcılarının işlevlerini destekleme.**
3. **Giriş/Çıkış Yönlendirmesi:**
    - **Dosyadan girdi okuma ve dosyaya çıktı yazma işlemlerini yönlendirme.**
4. **Arka Plan Prosesleri:**
    - **Komutları arka planda çalıştırma.**

Proje, Linux ortamında geliştirilmiştir ve C programlama dilinde gerçekleştirilmiştir.

---

2. Projenin Temel Bileşenleri

2.1. Prompt

Kabuk başlatıldığında ve her komutun tamamlanmasının ardından “>” komut istemi görüntülenir. Kullanıcı, bu komut isteminden sonra komutları girebilir.

2.2. Yerel Komutlar (Built-in Commands)

- **quit: Kabuğu sonlandırır. Eğer arkaplanda bir işlem devam ediyorsa, kabuk yeni komutlara yanıt vermeyi durdurur, arka plandaki işlemin bitmesini bekler ve sonra kapanır.**

2.3. Tekli Komutlar

Bağımsız değişkenlere sahip veya sahip olmayan tek komutların işlenmesi sağlanır. Kabuk, bir alt proses oluşturarak verilen komutu bu proses üzerinde çalıştırır ve tamamlanmasını bekler.

2.4. Girdi Yönlendirmesi

Bir komutun standart girdisi bir dosyadan alınabilir. Dosya bulunamadığında hata mesajı yazdırılır.

2.5. Çıktı Yönlendirmesi

Bir komutun çıktısı bir dosyaya yazılabilir. Dosya mevcut değilse oluşturulur, mevcutsa üzerine yazılır.

2.6. Arkaplan İşlem

Bir komut “&” ile bitiyorsa, bu komut arkaplanda çalıştırılır. Arkaplan işlem tamamlandığında, proses kimliği (PID) ve çıkış kodu görüntülenir.

2.7. Boru (“|”)

Bir komutun çıktısı bir sonraki komutun girdisine bağlanabilir. Tüm boru zincirinin tamamlanmasının ardından istem yeniden görüntülenir.

2.8. Komut Dizileri

Komutlar “;” ile ayrılarak ardışık olarak çalıştırılabilir. Komutlar birbirinden bağımsızdır.

3. TEKNİK DETAYLAR

3.1. Prompt Gösterimi

Kabuğumuz, her komut çalıştırıldıktan sonra veya arka plan işlemleri tamamlandığında > şeklinde bir komut istemi (prompt) görüntülemektedir. Bu istem, kullanıcının yeni bir komut girmesini beklemek için kullanılır. Komut isteminin hemen yazdırılması ve buffer'ın doğru bir şekilde boşaltılması için fflush(stdout) fonksiyonu kullanılmıştır.

3.2. quit Komutu ile Çıkış

Kabuğun kullanıcı tarafından sonlandırılması için quit adlı bir yerleşik (builtin) komut uygulanmıştır. Bu komutun çalışması şu şekilde tasarlanmıştır:

- **Normal Durum: Eğer herhangi bir arka plan işlemi çalışmıyorsa, quit komutu alındığında kabuk derhal sonlandırılır.**

- **Arka Plan İşlemleri Durumu: Eğer bir veya daha fazla arka plan işlemi çalışıyorsa:**
    - **Kabuk, yeni komut girişlerini kabul etmeyi durdurur.**
    - **Arka plan işlemlerinin tamamlanmasını bekler.**
    - **Her bir arka plan işlemi tamamlandığında, ilgili işlem kimliği (PID) ve çıkış durumu (exit status) kullanıcıya bildirilir.**
    - **Tüm işlemler tamamlandıktan sonra kabuk sonlandırılır.**

Arka plan işlemleri kontrolü için waitpid fonksiyonu ve WNOHANG bayrağı kullanılmıştır. Böylece, kabuk arka plan işlemlerinin durumunu asenkron olarak izleyebilir.

Bu özellik, kabuğun güvenli bir şekilde kapanmasını ve işlem kaynaklarının doğru bir şekilde serbest bırakılmasını sağlar. Ayrıca, arka plan işlemlerinin tamamlanma durumu kullanıcıya açık bir şekilde bildirildiğinden, kullanıcı deneyimi geliştirilmiştir.

3.3. Tekli Komutların Çalıştırılması

Kabuk, kullanıcıdan gelen bağımsız bir komut veya bir dizi bağımsız değişkeni yorumlayarak çalıştırır. Her komut, aşağıdaki işlemleri içerir:

- **Yeni bir alt süreç oluşturulur (fork).**
- **Alt süreçte, belirtilen komut execvp ile çalıştırılır.**
- **Ebeveyn süreç (kabuk), alt sürecin tamamlanmasını waitpid ile bekler.**

Bu tasarım, kullanıcıya tek bir komutun doğru bir şekilde çalıştırılmasını ve çıktıların standart çıktıya yazdırılmasını sağlar.

3.4. Giriş Yönlendirme

Giriş yönlendirme özelliği, komutun standart girdisinin belirtilen bir dosyadan alınmasını sağlar. Örneğin:

> cat < input.txt

Bu komut, input.txt dosyasının içeriğini okuyarak standart çıktıya yazdırır. Giriş yönlendirme işlemleri şu şekilde gerçekleştirilmiştir:

- **Komut satırı, < sembolü ve ardından gelen dosya adı için taranır.**
- **Dosya adı belirlenir ve open sistemi çağrısı ile açılır.**
- **Standart giriş (STDIN_FILENO), dup2 ile açılan dosyaya yönlendirilir.**
- **Eğer dosya açılamazsa, kullanıcıya bir hata mesajı gösterilir ve işlem sonlandırılır.**

3.5. Çıkış Yönlendirme

Çıkış yönlendirme özelliği, komutun standart çıktısının belirtilen bir dosyaya yazdırılmasını sağlar. Örneğin:

> ls > output.txt

Bu komut, ls komutunun çıktısını output.txt dosyasına yazar. Çıkış yönlendirme işlemleri şu şekilde gerçekleştirilmiştir:

- **Komut satırı, > sembolü ve ardından gelen dosya adı için taranır.**
- **Dosya adı belirlenir ve open sistemi çağrısı ile uygun modlarda (O_WRONLY | O_CREAT | O_TRUNC) açılır.**
- **Standart çıkış (STDOUT_FILENO), dup2 ile açılan dosyaya yönlendirilir.**
- **Eğer dosya açılamazsa, kullanıcıya uygun bir hata mesajı gösterilir ve işlem sonlandırılır.**

3.6. Arka Plan Çalıştırma

Komutun sonunda & sembolü yer alıyorsa, kabuk bu komutu arka planda çalıştırır. Örneğin:

> sleep 5 &

> echo "Arka planda çalışma"

Bu örnekte, sleep 5 komutu arka planda çalışırken, kullanıcı hemen başka bir komut girebilir. Arka plan çalıştırma özelliği şu şekilde gerçekleştirilmiştir:

- **Komutun sonunda & sembolü kontrol edilir ve kaldırılır.**
- **Yeni bir alt süreç oluşturulur (fork).**
- **Ebeveyn süreç, alt sürecin durumunu izlemek için süreç bilgilerini bir yapı içinde saklar.**
- **Alt süreç tamamlandığında, sinyal işleyici (SIGCHLD) ile süreç durumu kullanıcıya bildirilir.**

3.7. Borulama

Boru hattı (pipeline) özelliği, birden fazla komutun birbirine bağlanmasını sağlar. Örneğin:

> ls | grep "txt" | sort

Bu örnekte, ls komutunun çıktısı grep komutuna aktarılır ve ardından sort komutuna yönlendirilir. Borulama işlemleri şu şekilde gerçekleştirilmiştir:

- **Komut satırı | sembolüne göre bölünür.**
- **Her bir komut için ayrı bir süreç oluşturulur.**
- **Süreçler arasında veri aktarımı için pipe sistemi çağrısı kullanılır.**
- **Standart giriş ve çıkış, uygun süreçler için dup2 ile ayarlanır.**

- **Boru hattındaki tüm süreçler tamamlandıktan sonra kabuk, yeni komut girişi için hazır hale gelir.**