İŞLETİM SİSTEMLERİ ÖDEV

Grup Numarası: 50

Grup Üyeleri

- **Burak Coşkun -B231210351**
- **Furkan Demirelli -B241210904**
- **Eren Ozan Özmen -B221210059**

Github Linki: https://github.com/coskunburak/isletim_sistemleri

GİRİŞ

Projenin temel amacı, **proses yönetimi, I/O (giriş/çıkış) yönlendirmesi, sinyal yönetimi** ve benzeri işletim sistemi kavramlarına **uygulamalı** bir bakış kazandırmaktır.

Kullanıcı, kabuğa (shell) metin şeklinde komutlarını girer. Kabuk bu komutları ayrıştırır (parse eder) ve her bir komut için uygun işletim sistemi çağrılarını kullanarak yeni süreçler oluşturur veya kabuğun kendi yapısında tanımlanmış komutları yürütür.

PROJENİN AMACI VE KAPSAMI

1. **Amaç**:
   - Temel bir kabuk uygulaması geliştirerek, işletim sistemlerinde **process (süreç) yönetimi**, **giriş/çıkış yönlendirmesi** ve **sinyaller** gibi konuları somutlaştırmak.
   - Bir komut satırı yorumlayıcısı oluşturup, kullanıcının girdiği komutları çalıştırmak ve çıktılarını göstermek.
2. **Kapsam**:
   - **Prompt**: Kabuk, bir komut tamamlandığında veya arka plan görevi başlatıldığında kullanıcının yeni komut girmesi için > benzeri bir istem (prompt) gösterecektir.

- **Built-in Komut**: En azından `quit` gibi dahili bir komut kabuğun içinde tanımlı olacaktır.
- **Tekli Komut İcrası**: `ls -l`, `cat file.txt` gibi tekli komutları, alt süreç (child process) oluşturarak çalıştırabilmek.
- **I/O Yönlendirme**: < (giriş yönlendirme) ve > (çıkış yönlendirme) işlemlerini desteklemek.
- **Arka Plan Çalışma**: `&` sembolü ile belirtilen komutları, bekletmeden arka planda (background) çalıştırmak.
- **Boru (Pipe) Desteği**: `|` operatörüyle, bir komutun çıktısını diğer komuta girdi olarak bağlayabilmek.
- **Noktalı Virgül (;)**: Ardışık komutları peş peşe çalıştırabilmek.
- **Sinyal Yönetimi**: Arka plan süreçlerinin tamamlandığını izlemek ve bilgilendirme mesajları yazdırmak için sinyallerden (örneğin SIGCHLD) yararlanmak.

GENEL MİMARİ

1. **Komut İstemi (Prompt) Gösterme**:
   Kullanıcıya > şeklinde bir komut satırı istemi sunulur. Prompt görüntülendikten sonra, standart girdi üzerinden kullanıcının yazdığı komut satırı okunur.
2. **Komutu Ayrıştırma (Parsing)**:
   Kullanıcının girdiği satır, boşluk, `;`, `|`, <, >, `&` gibi özel ayırma (delimit) sembolleri dikkate alınarak **bir veya birden çok** alt-komuta dönüştürülür.
   - **Noktalı Virgül (;)**: Aynı satırdaki farklı komutları ardışık şekilde ayırır.
   - **Boru (|)**: Komutlar arası veri aktarım kanalı oluşturur.
   - **Yönlendirme (<, >)**: Girdi veya çıktı dosyasını ayarlar.
   - **Arka Plan (&)**: Komutun arka planda çalışması gerektiğini belirtir.
3. **Built-in Komutları Kontrol**:
   - Eğer girilen komut `quit` ise, kabuk sonlandırılır.
   - Diğer yerleşik komutlar (varsa) da bu aşamada işlenebilir.
4. **Komut İcra**:
   - **Tekli Komut**: Ayrıştırma adımında elde edilen parametrelerle birlikte, yeni bir alt süreç oluşturulur. Bu alt süreç, belirtilen komutu (örn. `ls`, `cat` vs.) çalıştırır.
   - **I/O Yönlendirme**: < ve > işaretleri tespit edildiğinde alt sürecin standart girdi/çıkışı ilgili dosyalara yönlendirilir.
   - **Arka Plan**: `&` konulduğunda, alt süreç oluşturulduktan sonra kabuk beklemez; yeni prompt hemen görüntülenir. Süreç bittikten sonra sinyal mekanizması üzerinden kullanıcıya `[pid] retval: …` biçiminde sonuç bilgisi verilir.
   - **Boru (Pipe)**: Birden çok komutun birbirine `|` ile bağlı olması halinde, komutlar arasına bir boru (pipe) oluşturulur. Her komutun çıkışı, kendisinden sonra gelen komuta girdi olur. Bütün boru hattındaki komutlar tamamlanana kadar kabuk, son komutun çıkışını ekranda göstermez (ön plan senaryosunda).
5. **Sinyal Yönetimi**:
   - Arka planda çalışan alt süreçler tamamlandığında **SIGCHLD** sinyali yakalanır. Kabuk, biten sürece ait **PID** ve çıkış kodunu (exit code) ekrana basar.
   - Kullanıcı `quit` komutu girdiğinde, varsa çalışan arka plan süreçlerini bekleyerek kapatır.

6. **Yeni Komut Bekleme**:
    - Her işlem tamamlandığında veya arka plan komutunun başlatılmasının hemen ardından yeniden > prompt görüntülenir ve kullanıcıdan gelecek yeni komutlar beklenir.

PROSES VE I/O YÖNETİMİ

Süreç Oluşturma

- Her harici komut icrasında, kabuk yeni bir alt süreç (`fork`) oluşturur. Ana süreç (kabuk) alt sürecin bitmesini ön planda çalıştırma durumunda bekler (`wait` veya `waitpid`).
- Arka planda çalıştırma durumunda (`&`), kabuk beklemez; alt sürecin bitişini sinyal üzerinden takip eder.

Giriş ve Çıkış Yönlendirme

- < sembolü tespit edilirse, alt sürecin standart girişi ilgili dosyaya yönlendirilir. Dosya bulunamazsa kullanıcıya "Giriş dosyası bulunamadı" şeklinde bilgi verilir.
- > sembolü tespit edilirse, alt sürecin standart çıkışı ilgili dosyaya aktarılır. Dosya açılma hatalarında kullanıcıya uyarı gönderilir.

Boru (Pipe)

- Eğer komutlar `|` sembolüyle ayrılmışsa, birden fazla alt süreç, **pipe** mekanizması ile birbirine bağlanır.
- İlk komutun çıktısı, pipe'ın yazma ucuna, sonraki komutun girdisi ise pipe'ın okuma ucuna bağlanır. Böylece veriler, komutlar arasında otomatik olarak aktarılır.

SİNYAL YÖNETİMİ

- Projede, özellikle **SIGCHLD** sinyaliyle arka plan süreçlerin bitişi yakalanarak, `[pid] retval: <exitcode>` formatında kullanıcıya geri bildirim yapılır.
- Kabuk, bu sayede arka plan süreçlerin bitmesini anlık takip edebilir.

UYGULAMA SENARYOLARI

Tekli Komut

1. Kullanıcı `> ls -l` komutu girer.
2. Kabuk komutu ayrıştırır ve bir alt süreç oluşturur.
3. Alt süreç `ls -l` programını çalıştırır, tamamlanınca kabuğa döner.
4. Kabuk prompt'u tekrar gösterir.

Giriş / Çıkış Yönlendirme

1. Kullanıcı `> echo 12 > test.txt` girer.
2. `echo 12` komutunun standart çıktısı `test.txt` dosyasına yönlendirilir.
3. Devamında `> cat < test.txt` komutu girdi dosyası olarak `test.txt`'yi kullanır ve ekrana `12` çıktısını yansıtır.

Boru Kullanımı

1. Kullanıcı `> echo 12 | increment` komutu girer.
2. Birinci komut (`echo 12`) çıkışını pipe'a yazar, ikinci komut (`increment`) bu değeri okur ve 13 şeklinde çıktıyı ekrana basar.

Arka Plan Çalıştırma

1. Kullanıcı `> sleep 5 &` komutunu girer.
2. `sleep 5` arka planda çalışır. Kabuk hemen yeni prompt gösterir.
3. `sleep 5` tamamlandığında kabuk `[24617] retval: 0` gibi mesaj yazar.

Noktalı Virgül (;) Kullanımı

1. Kullanıcı `> echo 12; sleep 2; echo 13` komutunu girer.
2. Komutlar ardışık olarak işlenir. Önce `echo 12` ekrana 12 basar, sonra 2 saniye bekler, ardından `echo 13` yazar.

PROJENİN SONLANDIRILMASI (QUIT KOMUTU)

- Kullanıcı `quit` komutunu girdiğinde, varsa çalışan arka plan görevleri beklenir ve kabuk düzenli biçimde kapanır.
- Bu sayede veriler ya da süreçler yarım kalmamış olur.

DEĞERLENDİRME VE SONUÇ

1. **Proses Yönetimi**: `fork`, `exec`, `wait` gibi fonksiyonların kullanımı ve süreçlerin yönetimi.

2. **Giriş/Çıkış Yönlendirmesi**: <, > sembolleriyle standart giriş ve çıkışın dosyalarla ilişkilendirilmesi.
3. **Sinyal İşleme**: Özellikle `SIGCHLD` yardımıyla arka plan süreçlerin takibi ve sonuç bildirimleri.
4. **Boru (pipe) Mekanizması**: Komutlar arası veri iletimini sağlama.
5. **Arka Plan İşleme**: Kullanıcıya bloklama yapmadan komutları çalıştırma becerisi.