metin, yazı tipi, logo, grafik içeren bir resim

Açıklama otomatik olarak oluşturuldu

|  |  |  |
| --- | --- | --- |
|  | **Student ID** | **Name - Surname** |
| **1** | 150122058 | Mehmet Efe Selamet |
| **2** | 150121016 | Aşkın Yavuz Tuna |
| **3** | 150123825 | Ayşenur Arıcı |

### **Simple Shell Project Report**

This report explains a program created by our project team for homework. We wrote this program in **C language** to act like a simple command-line shell. Below are the main features and how they work.

### **Features and How They Work**

#### **1. Running Commands**

* **Foreground Commands**:

When you type a command (e.g., ls) without adding &, it runs in the foreground. This means the shell will wait for the command to finish before letting you type the next one.

* **Background Commands**:

Adding & at the end of a command (e.g., gedit &) makes it run in the background. The shell doesn’t wait for it to finish and immediately lets you type the next command.

* **How It Works in the Code**:
  + We use fork() to create a new process for the command.
  + execvp() runs the command in the new process.
  + Background tasks are tracked using a list.

#### **2. Command History**

* The shell remembers the last 10 commands you typed.

**You can:**

* + Type history to see these commands.
  + Type history -i <number> to rerun a specific command from the list.
* **How It Works in the Code**:
  + Commands are saved in a list.
  + Memory is managed efficiently to store and update the history.

#### **3. Built-in Commands**

The shell has some special commands built into it:

* **exit**:

This command is used to terminate the shell. If any background tasks are still running, the shell will warn you to stop them first.

* **fg %<number>**:

Moves a background task to the foreground.

* **^Z (Control + Z)**:

Stops the command currently running in the foreground. If there is no foreground processes there will be no effects.

* **How It Works in the Code**:
  + Signals (SIGINT, SIGTSTP) are used to control processes.
  + A table keeps track of background tasks.

#### **4. Redirecting Input and Output**

You can redirect where the input/output of a command goes:

* **Redirect Output to a File**:
  + > saves the command's result to a file (e.g., ls > output.txt).
  + >> adds the result to the end of an existing file.
* **Redirect Input from a File**:
  + < makes the command read input from a file (e.g., cat < file.txt).
* **Redirect Errors to a File**:
  + 2> saves error messages to a file (e.g., gcc program.c 2> errors.txt).
* **How It Works in the Code**:
  + The program uses open() to access files.
  + dup2() redirects the standard input/output streams to/from files.

#### **5. Executions**

metin, ekran görüntüsü, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

metin, ekran görüntüsü, yazı tipi, tasarım içeren bir resim

Açıklama otomatik olarak oluşturuldu



metin, yazı tipi, ekran görüntüsü içeren bir resim

Açıklama otomatik olarak oluşturuldu

metin, yazı tipi, ekran görüntüsü içeren bir resim

Açıklama otomatik olarak oluşturuldu

metin, yazı tipi, ekran görüntüsü içeren bir resim

Açıklama otomatik olarak oluşturuldu

metin, ekran görüntüsü, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

**In Summary**

This program is like a simple version of the Linux shell. You can type commands, run them in the foreground or background, and manage history. It also supports input/output redirection and built-in commands. We used system calls like fork(), execvp(), and open() to make it work, and signals to handle processes efficiently.