TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN

MÔN HỆ ĐIỀU HÀNH

BÁO CÁO THỰC HÀNH LAB 6

Lớp IT007.P11

23520552 – PHAN CẢNH ĐĂNG HUÂN

NỘI DUNG BÁO CÁO

Hãy thiết kế một chương trình C để tạo ra một giao diện shell.
Giao diện này cho phép người dùng nhập các lệnh và sau đó thực thi từng lệnh trong một quy trình riêng biệt. Điểm đặc biệt là chương trình này sẽ hỗ trợ việc chuyển hướng đầu vào và đầu ra, cũng như sử dụng pipes như một cách để truyền thông tin giữa các cặp lệnh.

Source Code:

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <fcntl.h>
#include <signal.h>
#include <termios.h>
#define MAX_LINE 80          // Độ dài tối đa của lệnh
#define HISTORY_SIZE 10      // Lưu 10 lệnh gần nhất
char *history[HISTORY_SIZE]; // Mảng lưu lịch sử lệnh
int history_count = 0;       // Đếm số lệnh trong lịch sử
void add_to_history(const char *command)
{
    if (history_count < HISTORY_SIZE)
    {
        history[history_count++] = strdup(command);
    }
    else
    {
        free(history[0]);
        for (int i = 1; i < HISTORY_SIZE; i++)
        {
            history[i - 1] = history[i];
        }
        history[HISTORY_SIZE - 1] = strdup(command);
    }
}
void print_history()
{
    for (int i = 0; i < history_count; i++)
    {
        printf("%d: %s\n", i + 1, history[i]);
    }
}
```

```c
void execute_command(char *command)
{
    char *args[MAX_LINE / 2 + 1];
    int i = 0;
    // Phân tích lệnh thành các tham số
    args[i] = strtok(command, " ");
    while (args[i] != NULL)
    {
        args[++i] = strtok(NULL, " ");
    }
    // Kiểm tra lệnh rỗng
    if (args[0] == NULL)
        return;
    // Tạo tiến trình con để thực thi lệnh
    pid_t pid = fork();
    if (pid == 0)
    {
        // Tiến trình con
        // Kiểm tra chuyển hướng đầu vào '<'
        for (int j = 0; args[j] != NULL; j++)
        {
            if (strcmp(args[j], "<") == 0)
            {
                int fd = open(args[j + 1], O_RDONLY);
                if (fd == -1)
                {
                    perror("Không mở được file đầu vào");
                    exit(1);
                }
                dup2(fd, STDIN_FILENO);
                close(fd);
                args[j] = NULL; // Xóa toán tử khỏi danh sách tham số
            }
        }
        // Kiểm tra chuyển hướng đầu ra '>'
        for (int j = 0; args[j] != NULL; j++)
        {
            if (strcmp(args[j], ">") == 0)
            {
                int fd = open(args[j + 1], O_WRONLY | O_CREAT | O_TRUNC,
                              0644);
                if (fd == -1)
                {
                    perror("Không mở được file đầu ra");
                    exit(1);
                }
                dup2(fd, STDOUT_FILENO);
                close(fd);
                args[j] = NULL; // Xóa toán tử khỏi danh sách tham số
            }
        }
        // Thực thi lệnh
        if (execvp(args[0], args) == -1)
        {
            perror("Lỗi thực thi lệnh");
        }
        exit(1);
    }
    else if (pid > 0)
    {
        // Tiến trình cha chờ tiến trình con
        wait(NULL);
    }
    else
```

```c
    {
        perror("Lỗi tạo tiến trình");
    }
}
void handle_pipe(char *command)
{
    char *cmd1 = strtok(command, "|");
    char *cmd2 = strtok(NULL, "|");
    if (cmd2 == NULL)
    {
        execute_command(cmd1);
        return;
    }
    int fd[2];
    if (pipe(fd) == -1)
    {
        perror("Lỗi tạo pipe");
        return;
    }
    pid_t pid = fork();
    if (pid == 0)
    {
        // Tiến trình con 1
        dup2(fd[1], STDOUT_FILENO); // Gửi đầu ra tới pipe
        close(fd[0]);
        close(fd[1]);
        execute_command(cmd1);
        exit(0);
    }
    pid = fork();
    if (pid == 0)
    {
        // Tiến trình con 2
        dup2(fd[0], STDIN_FILENO); // Nhận đầu vào từ pipe
        close(fd[0]);
        close(fd[1]);
        execute_command(cmd2);
        exit(0);
    }
    // Tiến trình cha
    close(fd[0]);
    close(fd[1]);
    wait(NULL);
    wait(NULL);
}
void sigint_handler(int sig)
{
    printf("\nĐã nhận tín hiệu Ctrl+C. Nhập lệnh tiếp theo:\n");
    fflush(stdout);
}
int main()
{
    char command[MAX_LINE];
    int should_run = 1;
    // Cài đặt xử lý tín hiệu Ctrl+C
    signal(SIGINT, sigint_handler);
    while (should_run)
    {
        printf("it007sh> ");
        fflush(stdout);
        // Đọc lệnh từ người dùng
        if (!fgets(command, MAX_LINE, stdin))
            break;
        command[strcspn(command, "\n")] = '\0'; // Loại bỏ ký tự newline
        // Lưu vào lịch sử nếu không rỗng
```

```
        if (strlen(command) > 0)
        {
            add_to_history(command);
        }
        // Kiểm tra lệnh "exit"
        if (strcmp(command, "exit") == 0)
        {
            should_run = 0;
            continue;
        }
        // Kiểm tra lệnh "history"
        if (strcmp(command, "history") == 0)
        {
            print_history();
            continue;
        }
        // Kiểm tra có đường ống không
        if (strchr(command, '|') != NULL)
        {
            handle_pipe(command);
        }
        else
        {
            execute_command(command);
        }
    }
    // Giải phóng lịch sử
    for (int i = 0; i < history_count; i++)
    {
        free(history[i]);
    }
    return 0;
```

Giải thích code:

Chương trình shell này mô phỏng một shell đơn giản cho phép người dùng nhập và thực thi các lệnh từ dòng lệnh. Sau đây là giải thích chi tiết các phần của chương trình:

Lịch sử lệnh:

Chương trình hỗ trợ lưu lại các lệnh người dùng đã nhập trong lịch sử. Khi người dùng nhập một lệnh, chương trình sẽ lưu nó vào một mảng. Nếu số lượng lệnh vượt quá giới hạn (10 lệnh), chương trình sẽ loại bỏ lệnh cũ nhất để thêm lệnh mới vào cuối.
Chuyển hướng đầu vào và đầu ra:

Chương trình hỗ trợ chuyển hướng đầu vào và đầu ra thông qua các toán tử "<" và ">". Khi gặp toán tử <, chương trình sẽ chuyển hướng đầu vào từ một tệp, còn khi gặp toán tử >, chương trình sẽ chuyển hướng đầu ra đến một tệp. Điều này cho phép người dùng đọc từ hoặc ghi vào các tệp thay vì chỉ làm việc với đầu vào và đầu ra mặc định.
Sử dụng pipe:

Chương trình hỗ trợ các lệnh được nối với nhau qua pipe (|). Điều này cho phép đầu ra của lệnh trước được truyền làm đầu vào cho lệnh sau. Khi gặp dấu |, chương trình sẽ tạo một pipe và chia lệnh thành hai phần. Sau đó, nó sẽ tạo hai tiến trình con: một tiến trình thực thi lệnh đầu tiên và chuyển đầu ra vào pipe, và một tiến trình khác nhận đầu

vào từ pipe và thực thi lệnh thứ hai.

Tiến trình con và cha:

Khi thực thi một lệnh, chương trình tạo một tiến trình con bằng cách sử dụng fork(). Tiến trình con thực hiện lệnh đã được phân tích và chuyển hướng đầu vào/đầu ra nếu cần. Tiến trình cha sẽ đợi tiến trình con kết thúc trước khi tiếp tục thực hiện các lệnh tiếp theo.

Lệnh đặc biệt:

Chương trình hỗ trợ hai lệnh đặc biệt: exit để thoát khỏi chương trình và history để hiển thị lịch sử các lệnh đã thực thi. Khi người dùng nhập exit, chương trình sẽ kết thúc vòng lặp và thoát. Khi nhập history, chương trình sẽ in ra danh sách các lệnh đã thực thi trước đó.

Xử lý tín hiệu:

Chương trình cài đặt một trình xử lý tín hiệu cho tín hiệu Ctrl+C (SIGINT). Khi người dùng nhấn Ctrl+C, thay vì thoát khỏi chương trình, chương trình sẽ in ra một thông báo và cho phép người dùng nhập lệnh tiếp theo. Điều này ngăn chương trình bị dừng đột ngột.

Kết luận:

Chương trình mô phỏng một shell cơ bản, hỗ trợ các tính năng như chuyển hướng đầu vào/đầu ra, sử dụng pipes, lưu lịch sử lệnh, và xử lý tín hiệu. Các tiến trình con được tạo ra để thực thi các lệnh, trong khi tiến trình cha quản lý các tiến trình con và xử lý tín hiệu từ người dùng.