# Code

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <sys/stat.h>

#include <sys/types.h>

#include <signal.h>

#include <wait.h>

#define MAX\_CMD\_LENGTH 1024

#define MAX\_ARGS 100

#define BACKUP\_DIR "./backup/"

#define SOURCE\_DIR "./data/"

// Hàm kiểm tra và tạo thư mục nếu chưa tồn tại

int check\_and\_create\_directory(const char \*path) {

struct stat st;

if (stat(path, &st) == -1) {

printf("Thư mục '%s' không tồn tại. Đang tạo...\n", path);

if (mkdir(path, 0777) == -1) {

perror("Không thể tạo thư mục");

return 0;

}

printf("Thư mục '%s' đã được tạo.\n", path);

}

return 1;

}

// ======== CHỨC NĂNG QUẢN LÝ TỆP ========

FILE\* openFile(const char \*filename, const char \*mode) {

FILE \*file = fopen(filename, mode);

if (!file) {

printf("Không thể mở tệp '%s'.\n", filename);

}

return file;

}

void readFile(const char \*filename) {

FILE \*file = openFile(filename, "r");

if (!file) return;

char line[MAX\_CMD\_LENGTH];

printf("Nội dung tệp '%s':\n", filename);

while (fgets(line, sizeof(line), file)) {

printf("%s", line);

}

fclose(file);

}

void writeFile(const char \*filename) {

FILE \*file = openFile(filename, "w");

if (!file) return;

char content[MAX\_CMD\_LENGTH];

printf("Nhập nội dung muốn ghi vào tệp: ");

getchar();

fgets(content, sizeof(content), stdin);

fprintf(file, "%s", content);

fclose(file);

printf("Đã ghi vào tệp '%s' thành công.\n", filename);

}

void clearFile(const char \*filename) {

FILE \*file = openFile(filename, "w");

if (file) {

fclose(file);

printf("Nội dung tệp '%s' đã được xóa.\n", filename);

}

}

void checkIfFileIsEmpty(const char \*filename) {

FILE \*file = openFile(filename, "r");

if (!file) return;

char ch = fgetc(file);

if (ch == EOF) {

printf("Tệp '%s' hiện tại trống.\n", filename);

} else {

printf("Tệp '%s' có nội dung.\n", filename);

}

fclose(file);

}

// ======== CHỨC NĂNG QUẢN LÝ THƯ MỤC ========

void createFolder(const char \*name) {

if (mkdir(name, 0777) == 0) {

printf("Thư mục '%s' đã được tạo thành công.\n", name);

} else {

perror("Không thể tạo thư mục");

}

}

void deleteFolder(const char \*name) {

if (rmdir(name) == 0) {

printf("Thư mục '%s' đã được xóa.\n", name);

} else {

perror("Không thể xóa thư mục");

}

}

void listDirectory(const char \*path) {

char command[MAX\_CMD\_LENGTH];

snprintf(command, sizeof(command), "ls -l %s", path);

printf("Nội dung thư mục '%s':\n", path);

system(command);

}

// ======== CHỨC NĂNG SAO LƯU DỮ LIỆU ========

void backupData() {

printf("Bắt đầu sao lưu dữ liệu...\n");

if (!check\_and\_create\_directory(BACKUP\_DIR)) return;

if (!check\_and\_create\_directory(SOURCE\_DIR)) return;

char command[MAX\_CMD\_LENGTH];

snprintf(command, sizeof(command), "tar -czf %sbackup\_$(date +%%Y%%m%%d).tar.gz %s", BACKUP\_DIR, SOURCE\_DIR);

system(command);

printf("Sao lưu dữ liệu hoàn tất.\n");

}

// ======== CHỨC NĂNG QUẢN LÝ TIẾN TRÌNH ========

void listProcesses() {

printf("Danh sách tất cả tiến trình đang chạy:\n");

system("ps aux");

}

void suspendProcess() {

pid\_t pid;

printf("Nhập PID của tiến trình bạn muốn tạm dừng: ");

scanf("%d", &pid);

if (kill(pid, SIGSTOP) == 0) {

printf("Tiến trình %d đã bị tạm dừng.\n", pid);

} else {

perror("Không thể tạm dừng tiến trình");

}

}

void resumeProcess() {

pid\_t pid;

printf("Nhập PID của tiến trình bạn muốn tiếp tục: ");

scanf("%d", &pid);

if (kill(pid, SIGCONT) == 0) {

printf("Tiến trình %d đã được tiếp tục.\n", pid);

} else {

perror("Không thể tiếp tục tiến trình");

}

}

void killProcess() {

pid\_t pid;

printf("Nhập PID của tiến trình bạn muốn giết: ");

scanf("%d", &pid);

// Kiểm tra tiến trình có tồn tại hay không trước khi giết

if (kill(pid, 0) == 0) {

// Tiến trình tồn tại, thử giết nó

if (kill(pid, SIGKILL) == 0) {

printf("Tiến trình %d đã bị giết thành công.\n", pid);

} else {

perror("Không thể giết tiến trình");

}

} else {

perror("PID không hợp lệ hoặc tiến trình không tồn tại");

}

}

void createProcess() {

pid\_t pid = fork(); // Tạo tiến trình mới

if (pid < 0) {

perror("Không thể tạo tiến trình mới");

return;

}

if (pid == 0) {

// Đây là tiến trình con

printf("Tiến trình con (PID: %d) đang chạy...\n", getpid());

// Tiến trình con thực hiện tác vụ ví dụ

for (int i = 0; i < 5; i++) {

printf("Tiến trình con (PID: %d): Hoạt động lần %d\n", getpid(), i + 1);

sleep(2); // Chờ 2 giây giữa mỗi lần in

}

printf("Tiến trình con (PID: %d) kết thúc.\n", getpid());

exit(0); // Thoát khỏi tiến trình con

} else {

// Đây là tiến trình cha

printf("Tiến trình con đã được tạo với PID: %d\n", pid);

wait(NULL);

}

}

// ======== CHỨC NĂNG THỰC THI LỆNH ========

void parseCommand(char \*cmd, char \*\*args) {

char \*token = strtok(cmd, " \t\r\n");

int i = 0;

while (token) {

args[i++] = token;

token = strtok(NULL, " \t\r\n");

}

args[i] = NULL;

}

void executeCommand(char \*\*args) {

pid\_t pid = fork();

if (pid == 0) {

if (execvp(args[0], args) == -1) {

perror("Không thể thực thi lệnh");

}

exit(0);

} else {

wait(NULL);

}

}

void executeFromFile(const char \*filename) {

FILE \*file = openFile(filename, "r");

if (!file) return;

char cmd[MAX\_CMD\_LENGTH];

char \*args[MAX\_ARGS];

while (fgets(cmd, sizeof(cmd), file)) {

parseCommand(cmd, args);

if (args[0] && strcmp(args[0], "exit") == 0) break;

executeCommand(args);

}

fclose(file);

}

// ======== MENU CHÍNH ========

void showMainMenu() {

printf("\nCHỌN MỘT TÙY CHỌN:\n");

printf("1. Quản lý tệp\n");

printf("2. Quản lý thư mục\n");

printf("3. Sao lưu dữ liệu\n");

printf("4. Quản lý tiến trình\n");

printf("5. Thực thi lệnh\n");

printf("6. Thoát\n");

printf("Lựa chọn: ");

}

int main() {

int choice;

while (1) {

showMainMenu();

scanf("%d", &choice);

switch (choice) {

case 1: {

int fileChoice;

printf("\n1. Đọc tệp\n2. Ghi tệp\n3. Xóa nội dung tệp\n4. Kiểm tra tệp trống\n");

printf("Lựa chọn: ");

scanf("%d", &fileChoice);

char filename[100] = "input.txt";

switch (fileChoice) {

case 1: readFile(filename); break;

case 2: writeFile(filename); break;

case 3: clearFile(filename); break;

case 4: checkIfFileIsEmpty(filename); break;

default: printf("Lựa chọn không hợp lệ.\n");

}

break;

}

case 2: {

int folderChoice;

printf("\n1. Tạo thư mục\n2. Xóa thư mục\n3. Liệt kê thư mục\n");

printf("Lựa chọn: ");

scanf("%d", &folderChoice);

char path[100];

switch (folderChoice) {

case 1: printf("Nhập tên thư mục: "); scanf("%s", path); createFolder(path); break;

case 2: printf("Nhập tên thư mục: "); scanf("%s", path); deleteFolder(path); break;

case 3: printf("Nhập đường dẫn: "); scanf("%s", path); listDirectory(path); break;

default: printf("Lựa chọn không hợp lệ.\n");

}

break;

}

case 3: backupData(); break;

case 4: {

int processChoice;

printf("\n1. Liệt kê tiến trình\n2. Tạm dừng tiến trình\n3. Tiếp tục tiến trình\n4. Giết tiến trình\n5. Tạo tiến trình\n");

printf("Lựa chọn: ");

scanf("%d", &processChoice);

switch (processChoice) {

case 1: listProcesses(); break;

case 2: suspendProcess(); break;

case 3: resumeProcess(); break;

case 4: killProcess(); break;

case 5: createProcess(); break;

default: printf("Lựa chọn không hợp lệ.\n");

}

break;

}

case 5: {

char cmd[MAX\_CMD\_LENGTH];

char \*args[MAX\_ARGS];

printf("Nhập lệnh: ");

getchar(); // Loại bỏ newline còn lại trong buffer

fgets(cmd, sizeof(cmd), stdin);

parseCommand(cmd, args);

executeCommand(args);

break;

}

case 6: printf("Thoát chương trình.\n"); return 0;

default: printf("Lựa chọn không hợp lệ.\n");

}