AOS Lab 1

1 message

Vedica Kandoi <vedica01@gmail.com> Fri, 15 Sept 2023 at 2:26 am
To: Nidhi Vadodariya <nidhivadodariya000@gmail.com>, vk23csm1r23@student.nitw.ac.in

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <dirent.h>
#include <signal.h>
#include <sys/types.h>
#include <sys/stat.h>

// Function to implement the 'ls' command (List Files and Directories)
void list_files(int argc, char *argv[]) {
  int opt;
  int long_format = 0;
  int show_hidden = 0;
  int recursive = 0;
  const char *directory_name = ".";

  // Parse command-line options
  while ((opt = getopt(argc, argv, "lRa")) != -1) {
    switch (opt) {
      case 'l':
        long_format = 1;
        break;
      case 'a':
        show_hidden = 1;
        break;
      case 'R':
        recursive = 1;
        break;
      default:
        fprintf(stderr, "Usage: %s ls [-l] [-a] [-R] [directory]\n", argv[0]);
        exit(1);
    }
  }

  // If there's an extra argument, use it as the directory name
  if (optind < argc) {
    directory_name = argv[optind];
  }

  struct dirent *entry;
  DIR *dp = opendir(directory_name);

  if (dp == NULL) {
    perror("opendir");
    exit(1);
  }

  while ((entry = readdir(dp))) {
    if (!show_hidden && entry->d_name[0] == '.') {
      continue; // Skip hidden files and directories
    }
```

```
        if (long_format) {
            struct stat file_stat;
            char full_path[PATH_MAX];
            snprintf(full_path, PATH_MAX, "%s/%s", directory_name, entry->d_name);
            if (lstat(full_path, &file_stat) < 0) {
                perror("lstat");
                exit(1);
            }
            printf("%s\t%ld\t%s\n", entry->d_name, (long)file_stat.st_size, S_ISDIR(file_stat.st_mode) ? "directory" : "file");
        } else {
            printf("%s\n", entry->d_name);
        }
    }

    closedir(dp);
}

// Function to implement the 'pwd' command (Print Working Directory)
void print_working_directory() {
    char cwd[1024];

    if (getcwd(cwd, sizeof(cwd)) != NULL) {
        printf("%s\n", cwd);
    } else {
        perror("getcwd");
        exit(1);
    }
}

// Function to implement the 'mkdir' command (Create Directory)
void create_directory(const char *directory_name) {
    if (mkdir(directory_name, 0777) == 0) {
        printf("Directory '%s' created successfully.\n", directory_name);
    } else {
        perror("mkdir");
        exit(1);
    }
}

// Function to implement the 'cd' command (Change Directory)
void change_directory(const char *new_directory) {
    if (chdir(new_directory) == 0) {
        printf("Changed directory to '%s'.\n", new_directory);
    } else {
        perror("chdir");
    }
}

// Function to implement the 'touch' command (Create Empty File)
void create_empty_file(const char *file_name) {
    FILE *file = fopen(file_name, "w");

    if (file) {
        fclose(file);
        printf("Empty file '%s' created successfully.\n", file_name);
    } else {
        perror("fopen");
        exit(1);
    }
}


// Function to implement the 'rm' command (Remove File)
void remove_files(int argc, char *argv[]) {
    int interactive = 0;
    int opt;

    // Parse command-line options
    while ((opt = getopt(argc, argv, "i")) != -1) {
        switch (opt) {
            case 'i':
                interactive = 1;
                break;
            default:
                fprintf(stderr, "Usage: %s rm [-i] [file1] [file2] ...\n", argv[0]);
                exit(1);
        }
    }

    // Start processing files after options
    for (int i = optind; i < argc; i++) {
        char *file_name = argv[i];

        if (interactive) {
            // Ask for confirmation before removing each file
            printf("Remove '%s'? (y/n): ", file_name);
            char response;
            scanf(" %c", &response);

            if (response != 'y' && response != 'Y') {
                continue; // Skip this file
            }
        }

        if (remove(file_name) == 0) {
            printf("File '%s' removed successfully.\n", file_name);
        } else {
            perror("remove");
        }
    }
}

// Function to implement the 'cat' command (Concatenate and Display Files)
void cat_files(int argc, char *argv[]) {
    if (argc == 2) {
        // No file specified after 'cat,' print usage and return
        fprintf(stderr, "Usage: %s cat <file> [file2] ...\n", argv[0]);
        exit(1);
    }

    for (int i = 2; i < argc; i++) {
        char *file_name = argv[i];
        FILE *file = fopen(file_name, "r");

        if (file) {
            int c;
            while ((c = fgetc(file)) != EOF) {
                putchar(c);
            }
            fclose(file);
        } else {
            perror("fopen");
            exit(1);
        }
    }
}

// Function to implement the 'kill' command (Terminate a Process)
void kill_process(int argc, char *argv[]) {
    if (argc == 2) {
```

```c
        fprintf(stderr, "Usage: %s kill pid [pid2] ...\n", argv[0]);
        exit(1);
    }

    int signo = SIGTERM; // Default signal is SIGTERM

    if (argv[2][0] == '-') {
        // Handle the case of 'kill -<signo> <pid> [pid2] ...'
        if (argc < 4) {
            fprintf(stderr, "Usage: %s kill -<signo> <pid> [pid2] ...\n", argv[0]);
            exit(1);
        }

        signo = atoi(&argv[2][1]); // Extract the signal number
        if (signo <= 0) {
            fprintf(stderr, "Invalid signal number: %s\n", &argv[2][1]);
            exit(1);
        }
    }

    for (int i = 2 + (argv[2][0] == '-'); i < argc; i++) {
        int pid = atoi(argv[i]);
        if (pid <= 0) {
            fprintf(stderr, "Invalid process ID: %s\n", argv[i]);
            exit(1);
        }

        if (kill(pid, signo) == 0) {
            printf("Process with PID %d terminated with signal %d.\n", pid, signo);
        } else {
            perror("kill");
            exit(1);
        }
    }
}

// Function to implement the 'ps' command (List Processes)
void list_processes(int argc, char *argv[]) {
    if (argc == 2) {
        // Basic 'ps' command, list all processes
        // You can customize this part based on your requirements
        printf("List of all processes:\n");
        system("ps");
    } else if (strcmp(argv[2], "-a") == 0) {
        // 'ps -a' command, list all processes
        printf("List of all processes:\n");
        system("ps");
    } else if (strcmp(argv[2], "-ae") == 0) {
        // 'ps -ae' command, list all processes (including other users)
        printf("List of all processes (including other users):\n");
        system("ps -e");
    } else if (strcmp(argv[2], "-u") == 0 && argc >= 4) {
        // 'ps -u <username>' command, list processes for a specific user
        printf("List of processes for user '%s':\n", argv[3]);
        char cmd[100];
        snprintf(cmd, sizeof(cmd), "ps -u %s", argv[3]);
        system(cmd);
    } else {
        fprintf(stderr, "Usage: %s ps [-a] [-ae] [-u <username>]\n", argv[0]);
        exit(1);
    }
}
// Function to implement the 'wc' command (Word Count)
void word_count(int argc, char *argv[]) {
    if (argc < 3) {
        fprintf(stderr, "Usage: %s wc [-c] [-l] [-w] <file1> [file2] ...\n", argv[0]);
        exit(1);
    }

    int char_count = 0;
    int line_count = 0;
    int word_count = 0;
    int opt;

    while ((opt = getopt(argc, argv, "clw")) != -1) {
        switch (opt) {
            case 'c':
                char_count = 1;
                break;
            case 'l':
                line_count = 1;
                break;
            case 'w':
                word_count = 1;
                break;
            default:
                fprintf(stderr, "Usage: %s wc [-c] [-l] [-w] <file1> [file2] ...\n", argv[0]);
                exit(1);
        }
    }

    for (int i = optind; i < argc; i++) {
        char *file_name = argv[i];
        FILE *file = fopen(file_name, "r");

        if (file) {
            int c;
            int in_word = 0;
            while ((c = fgetc(file)) != EOF) {
                char_count++;
                if (c == '\n') {
                    line_count++;
                }
                if (c == ' ' || c == '\n' || c == '\t') {
                    in_word = 0;
                } else if (in_word == 0) {
                    in_word = 1;
                    word_count++;
                }
            }
            fclose(file);
        } else {
            perror("fopen");
            exit(1);
        }
    }

    if (char_count) {
        printf("Character count: %d\n", char_count);
    }
    if (line_count) {
        printf("Line count: %d\n", line_count);
    }
    if (word_count) {
        printf("Word count: %d\n", word_count);
    }
}

int main(int argc, char *argv[]) {
    if (argc < 2) {
        printf("Usage: %s <command> [arguments]\n", argv[0]);
        return 1;
    }

    if (strcmp(argv[1], "ls") == 0) {
        list_files(argc, argv);
    } else if (strcmp(argv[1], "pwd") == 0) {
        print_working_directory();
    } else if (strcmp(argv[1], "mkdir") == 0) {
        if (argc < 3) {
            printf("Usage: %s mkdir <directory_name>\n", argv[0]);
            return 1;
        }
        create_directory(argv[2]);
    } else if (strcmp(argv[1], "cd") == 0) {
        if (argc < 3) {
            printf("Usage: %s cd <directory_name>\n", argv[0]);
            return 1;
        }
        change_directory(argv[2]);
    } else if (strcmp(argv[1], "rm") == 0) {
        if (argc < 3) {
            printf("Usage: %s rm [-i] [file1] [file2] ...\n", argv[0]);
            return 1;
        }
        remove_files(argc, argv);
    } else if (strcmp(argv[1], "touch") == 0) {
        if (argc < 3) {
            printf("Usage: %s touch <file_name>\n", argv[0]);
            return 1;
        }
        create_empty_file(argv[2]);
    } else if (strcmp(argv[1], "cat") == 0) {
        cat_files(argc, argv);
    } else if (strcmp(argv[1], "kill") == 0) {
        if (argc < 3) {
            printf("Usage: %s kill pid [pid2] ...\n", argv[0]);
            return 1;
        }
        kill_process(argc, argv);
    } else if (strcmp(argv[1], "ps") == 0) {
        list_processes(argc, argv);
    } else if (strcmp(argv[1], "wc") == 0) {
        word_count(argc, argv);
    } else {
        printf("Unknown command: %s\n", argv[1]);
        return 1;
    }

    return 0;
}
```