RV Educational Institutions ®

RV College of Engineering ®

Autonomous Institution
Affiliated to Visvesvaraya
Technological University,
Belagavi

Approved by AICTE,
New Delhi

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

OPERATING SYSTEMS - CS235AI

Exploring File Operations and Cloud-Integration

Submitted by

RAMCHANDRA M RAYAKAR -1RV22CS157
PRAVEEN PRAKASH HEBBAL-1RV22CS149
NITHIN GOWDA L-1RV22CS132

Computer Science and Engineering
2023-2024

INTRODUCTION:

- The project involves the development of a simple file manager program using the C programming language.
- This file manager allows users to perform various file operations such as creating directories, creating files, deleting files/directories, reading files, listing directory contents, moving files, changing directories, opening files with a text editor, and integrating with cloud services like Dropbox for listing files and uploading files.

SYSTEM ARCHITECTURE:

- The file manager's structure is modular, consisting of distinct components for handling file operations, user interface, text editor integration, and cloud storage integration.
- Components include functionalities for creating directories, creating files, deleting files/directories, reading files, listing directory contents, moving files, and opening files with a text editor.
- Interaction with the operating system is facilitated through system calls such as `chdir`, `open`, `read`, `write`, `mkdir`, `rmdir`, `unlink`, and `rename`, enabling manipulation of files and directories within the file system.
- External APIs, such as the Dropbox API, are utilized for cloud storage integration. The file manager communicates with these APIs over HTTP, sending requests to list files and upload files to cloud storage services.
- The modular structure and interaction with both the operating system and external APIs enable the file manager to provide comprehensive file management capabilities, including local file operations and integration with cloud storage services.

METHODOLOGY:

- The file manager was developed iteratively, breaking down tasks into manageable chunks and prioritizing based on dependencies.
- Modular programming principles were applied to enhance code organization and maintainability, with each functionality encapsulated into separate modules or functions.
- Testing procedures, including unit testing, integration testing, and user acceptance testing, were integrated throughout the development process to ensure reliability and correctness.
- Various debugging techniques, such as print debugging and using debugging tools like gdb, were employed to identify and resolve issues in the codebase.
- Continuous improvement was emphasized, with feedback from testing and debugging used to refine the software and maintain code quality.
- Regular code reviews, refactoring, and documentation updates were conducted to align with project objectives.

SYSTEM CALLS:

- cdir: Changing the current working directory.
- open: Opening files or creating new files.
- write: Writing data to files.
- read: Reading data from files.
- mkdir: Creating directories.
- rmdir: Deleting directories.
- unlink: Deleting files.
- readdir: Reading directory contents.
- rename: Renaming files or directories.

SOURCE CODE

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

#define MAX_PATH 512

const char *dropbox_access_token = "DropBox Console Link";

struct MemoryStruct
{
   char *memory;
   size_t size;
};

char current_directory[MAX_PATH] = ".";

void move_up()
{
```

```
    if (chdir("..") == -1)
    {
        perror("chdir");
        exit(EXIT_FAILURE);
    }
    else
    {
        getcwd(current_directory, sizeof(current_directory));
        printf("Moved up to the parent directory.\n");
    }
}

void change_directory(const char *dirname)
{
    char path[MAX_PATH];
    snprintf(path, sizeof(path), "%s/%s", current_directory, dirname);

    if (chdir(path) == -1)
    {
        perror("chdir");
        exit(EXIT_FAILURE);
    }
    else
    {
        getcwd(current_directory, sizeof(current_directory));
        printf("Changed working directory to '%s'\n", current_directory);
```

```
    }
}

void create_directory(const char *dirname)
{
    char path[MAX_PATH];
    snprintf(path, sizeof(path), "%s/%s", current_directory, dirname);

    if (mkdir(path, 0777) == -1)
    {
        perror("mkdir");
        exit(EXIT_FAILURE);
    }
    else
    {
        printf("Directory '%s' created successfully.\n", path);
    }
}

void create_file(const char *filename)
{
    char path[MAX_PATH];
    snprintf(path, sizeof(path), "%s/%s", current_directory, filename);

    int fd = open(path, O_CREAT | O_RDWR, 0666);
    if (fd == -1)
```

```
    {
        perror("open");
        exit(EXIT_FAILURE);
    }

    const char *content = "Hello, this is a sample file.\n";
    if (write(fd, content, strlen(content)) == -1)
    {
        perror("write");
        exit(EXIT_FAILURE);
    }

    close(fd);
}

void delete_file(const char *filename)
{
    char path[MAX_PATH];
    snprintf(path, sizeof(path), "%s/%s", current_directory, filename);

    if (unlink(path) == -1)
    {
        perror("unlink");
        exit(EXIT_FAILURE);
    }
    else
```

```
    {
        printf("File '%s' deleted successfully.\n", path);
    }
}

void delete_directory(const char *dirname)
{
    char path[MAX_PATH];
    snprintf(path, sizeof(path), "%s/%s", current_directory, dirname);

    if (rmdir(path) == -1)
    {
        perror("rmdir");
        exit(EXIT_FAILURE);
    }
    else
    {
        printf("Directory '%s' deleted successfully.\n", path);
    }
}

void read_file(const char *filename)
{
    char path[MAX_PATH];
    snprintf(path, sizeof(path), "%s/%s", current_directory, filename);
```

```
    int fd = open(path, O_RDONLY);
    if (fd == -1)
    {
        perror("open");
        exit(EXIT_FAILURE);
    }

    char buffer[1024];
    ssize_t bytesRead;

    printf("Content of file '%s':\n", path);

    while ((bytesRead = read(fd, buffer, sizeof(buffer))) > 0)
    {
        write(STDOUT_FILENO, buffer, bytesRead);
    }

    close(fd);
}

void list_directory()
{
    DIR *dir = opendir(current_directory);
    if (dir == NULL)
    {
        perror("opendir");
```

```
        exit(EXIT_FAILURE);
    }

    struct dirent *entry;
    printf("Contents of directory '%s':\n", current_directory);

    while ((entry = readdir(dir)) != NULL)
    {
        printf("%s\n", entry->d_name);
    }

    closedir(dir);
}

void move_file(const char *source, const char *destination)
{
    char source_path[MAX_PATH], dest_path[MAX_PATH];
    snprintf(source_path, sizeof(source_path), "%s/%s",
current_directory, source);
    snprintf(dest_path, sizeof(dest_path), "%s/%s", current_directory,
destination);

    if (rename(source_path, dest_path) == -1)
    {
        perror("rename");
        exit(EXIT_FAILURE);
```

```
    }
    else
    {
        printf("File '%s' moved to '%s' successfully.\n", source_path,
dest_path);
    }
}

void open_file_in_editor(const char *filename)
{
    char path[MAX_PATH];
    snprintf(path, sizeof(path), "%s/%s", current_directory, filename);

    int fd = open(path, O_CREAT | O_RDWR, 0666);
    if (fd == -1)
    {
        perror("open");
        exit(EXIT_FAILURE);
    }

    close(fd);

    char editor_command[MAX_PATH];
    snprintf(editor_command, sizeof(editor_command), "gedit %s",
path);
```

```
    if (system(editor_command) == -1)
    {
        perror("system");
        exit(EXIT_FAILURE);
    }
}
void change_directory_shared(const char *shared_folder_name)
{
    char path[MAX_PATH];
    snprintf(path, sizeof(path), "/media/sf_%s", shared_folder_name);

    if (chdir(path) == -1)
    {
        perror("chdir");
        exit(EXIT_FAILURE);
    }
    else
    {
        getcwd(current_directory, sizeof(current_directory));
        printf("Changed working directory to '%s'\n", current_directory);
    }
}
void rename_file(const char *oldname, const char *newname)
{
    char old_path[MAX_PATH], new_path[MAX_PATH];
    snprintf(old_path,   sizeof(old_path),   "%s/%s",   current_directory,
```

```
oldname);
   snprintf(new_path, sizeof(new_path), "%s/%s", current_directory,
newname);

   if (rename(old_path, new_path) == -1)
   {
      perror("rename");
      exit(EXIT_FAILURE);
   }
   else
   {
      printf("File '%s' renamed to '%s' successfully.\n", oldname,
newname);
   }
}
size_t write_callback(void *data, size_t size, size_t nmemb, struct
MemoryStruct *mem)
{
   size_t realsize = size * nmemb;

   mem->memory = realloc(mem->memory, mem->size + realsize +
1);
   if (mem->memory == NULL)
   {
      fprintf(stderr, "Out of memory\n");
      exit(EXIT_FAILURE);
```

```
    }

    memcpy(&(mem->memory[mem->size]), data, realsize);
    mem->size += realsize;
    mem->memory[mem->size] = 0;

    return realsize;
}

void dropbox_api_list_files()
{
    CURL *curl;
    CURLcode res;
    struct MemoryStruct chunk;

    chunk.memory = malloc(1); // Will be grown as needed
    chunk.size = 0;           // No data yet

    // Initialize libcurl
    curl = curl_easy_init();
    if (curl)
    {
        // Set the request URL
        curl_easy_setopt(curl,                                  CURLOPT_URL,
"https://api.dropboxapi.com/2/files/list_folder");
```

```
    // Set HTTP headers
    struct curl_slist *headers = NULL;
    headers = curl_slist_append(headers, "Content-Type: application/json");
    char authorization_header[100];
    sprintf(authorization_header, "Authorization: Bearer %s", dropbox_access_token);
    headers = curl_slist_append(headers, authorization_header);
    curl_easy_setopt(curl, CURLOPT_HTTPHEADER, headers);

    // Set the HTTP POST data
    const char *post_fields = "{\"path\": \"\"}"; // Empty string represents the root directory
    curl_easy_setopt(curl, CURLOPT_POSTFIELDS, post_fields);

    // Set the callback function to handle the response
    curl_easy_setopt(curl, CURLOPT_WRITEFUNCTION, write_callback);
    curl_easy_setopt(curl, CURLOPT_WRITEDATA, &chunk);

    // Perform the request
    res = curl_easy_perform(curl);

    // Check for errors
    if (res != CURLE_OK)
    {
```

```
            fprintf(stderr, "curl_easy_perform() failed: %s\n",
curl_easy_strerror(res));
        }
        else
        {
            printf("Dropbox API response:\n%s\n", chunk.memory);
        }

        // Cleanup
        curl_slist_free_all(headers);
        curl_easy_cleanup(curl);
    }

    // Cleanup memory
    free(chunk.memory);
}

void dropbox_api_upload_file(const char *local_file_path, const char
*remote_file_path)
{
    CURL *curl;
    CURLcode res;

    // Initialize libcurl
    curl = curl_easy_init();
    if (curl)
```

```
    {
        // Set the request URL
        curl_easy_setopt(curl, CURLOPT_URL, "https://content.dropboxapi.com/2/files/upload");

        // Set HTTP headers
        struct curl_slist *headers = NULL;
        headers = curl_slist_append(headers, "Content-Type: application/octet-stream");
        char authorization_header[100];
        sprintf(authorization_header, "Authorization: Bearer %s", dropbox_access_token);
        headers = curl_slist_append(headers, authorization_header);
        char dropbox_api_arg_header[100];
        sprintf(dropbox_api_arg_header, "Dropbox-API-Arg: {\"path\":\"%s\"}", remote_file_path);
        headers = curl_slist_append(headers, dropbox_api_arg_header);
        curl_easy_setopt(curl, CURLOPT_HTTPHEADER, headers);

        // Set the HTTP POST data (file content)
        FILE *file = fopen(local_file_path, "rb");
        if (file == NULL)
        {
            fprintf(stderr, "Failed to open local file for reading\n");
            exit(EXIT_FAILURE);
        }
```

```
        curl_easy_setopt(curl, CURLOPT_READDATA, file);

        // Perform the request
        res = curl_easy_perform(curl);

        // Check for errors
        if (res != CURLE_OK)
        {
            fprintf(stderr, "curl_easy_perform() failed: %s\n",
curl_easy_strerror(res));
        }

        // Cleanup
        curl_slist_free_all(headers);
        curl_easy_cleanup(curl);
        fclose(file);
    }
}

void integrate_with_dropbox()
{
    int option;
    printf("\nDropbox Integration Menu:\n");
    printf("1. List Files\n");
    printf("2. Upload File\n");
    printf("Enter your choice: ");
```

```
    scanf("%d", &option);

    switch (option)
    {
    case 1:
        dropbox_api_list_files();
        break;
    case 2:
    {
        char local_file_path[MAX_PATH];
        char remote_file_path[MAX_PATH];
        printf("Enter local file path: ");
        scanf("%s", local_file_path);
        printf("Enter remote file path: ");
        scanf("%s", remote_file_path);
        dropbox_api_upload_file(local_file_path, remote_file_path);
        break;
    }
    default:
        printf("Invalid choice. Please enter 1 or 2.\n");
    }
}
int main()
{
    while (1)
    {
```

```
printf("\nFile Manager Menu:\n");
printf("1. Create Directory\n");
printf("2. Create File\n");
printf("3. Delete File\n");
printf("4. Delete Directory\n");
printf("5. Read File\n");
printf("6. List Directory\n");
printf("7. Move File\n");
printf("8. Change Directory\n");
printf("9. Move Up to Parent Directory\n");
printf("10.write(open with text editor)\n");
printf("11.Enter shared folder\n");
printf("12.rename the file\n");
printf("13.intigrate with cloud(dropbox)");
printf("14. Exit\n");

int choice;
printf("Enter your choice: ");
scanf("%d", &choice);

switch (choice)
{
case 1:
{
   char dirname[MAX_PATH];
   printf("Enter directory name: ");
```

```
        scanf("%s", dirname);
        create_directory(dirname);
        break;
    }
    case 2:
    {
        char filename[MAX_PATH];
        printf("Enter file name: ");
        scanf("%s", filename);
        create_file(filename);
        break;
    }
    case 3:
    {
        char filename[MAX_PATH];
        printf("Enter file name to delete: ");
        scanf("%s", filename);
        delete_file(filename);
        break;
    }
    case 4:
    {
        char dirname[MAX_PATH];
        printf("Enter directory name to delete: ");
        scanf("%s", dirname);
        delete_directory(dirname);
```

```
        break;
    }
    case 5:
    {
        char filename[MAX_PATH];
        printf("Enter file name to read: ");
        scanf("%s", filename);
        read_file(filename);
        break;
    }
    case 6:
    {
        list_directory();
        break;
    }
    case 7:
    {
        char source[MAX_PATH], destination[MAX_PATH];
        printf("Enter source file name: ");
        scanf("%s", source);
        printf("Enter destination directory name: ");
        scanf("%s", destination);
        move_file(source, destination);
        break;
    }
    case 8:
```

```
{
    char dirname[MAX_PATH];
    printf("Enter directory name to change to: ");
    scanf("%s", dirname);
    change_directory(dirname);
    break;
}
case 9:
{
    move_up();
    break;
}
case 10:
{
    char filename[MAX_PATH];
    printf("Enter file name to open or write: ");
    scanf("%s", filename);
    open_file_in_editor(filename);
    break;
}

case 11:
{
    char shared_folder_name[MAX_PATH];
    printf("Enter shared folder name: ");
    scanf("%s", shared_folder_name);
```

```
            change_directory_shared(shared_folder_name);
            break;
        }
        case 12:
        {
            char old_filename[MAX_PATH],
new_filename[MAX_PATH];
            printf("Enter the current file name: ");
            scanf("%s", old_filename);
            printf("Enter the new file name: ");
            scanf("%s", new_filename);
            rename_file(old_filename, new_filename);
            break;
        }
        case 13:
        {
            integrate_with_dropbox();
            break;
        }

        case 14:
        {
            printf("Exiting File Manager. Goodbye!\n");
            exit(EXIT_SUCCESS);
        }
        default:
```

```
            printf("Invalid choice. Please enter a number between 1 and
10.\n");
        }
    }

    return 0;
}
```

Conclusion:

- Key Takeaways:

  Enhanced understanding of file management and system-level programming.

  Proficiency in modular programming for improved code organization.

  Experience in testing and debugging for software reliability.

  Appreciation for iterative development and continuous improvement.

  Acquisition of skills applicable to future software projects.

- Future Enhancements and Applications:

  Additional file operations, advanced features, and cross-platform compatibility.

  Potential applications in education, business, and personal utilities.