Operating Systems Coursework

Zeeshan Kazmi

KAZ22527866

Read Me

This code is split into 3 parts:

Part A

- The program will make a new directory.
- The program will set focus to the new directory.
- The user gets to input the name and type of a file.
- A file will be made with the provided details if it does not already exist.

Part B

- The user is presented with a menu of 4 options.
- Each option will set different permissions for the newly created file.
- “Read Only”, “Read and Write”, “Read, Write, and Execute”, and “Write Only” are the options.
- The user inputs a number between 1 and 4 to select one of the options.

Part C

- The file is then encrypted using a XOR encryption algorithm.
- The encrypted contents are output and can also be seen in the file itself.
- The user can then press enter to decrypt the contents of the file, with the changes being output and visible in the file itself again.

Here are the functions used in the code:

clearBuffer()

This function is used in multiple points throughout the program, right after input is accepted using scanf(). This is to clear the input buffer to prevent excess characters from inputs affecting future inputs later in the program.

```
void clearBuffer()
{
    int i;
    while ((i = getchar()) != '\n' && i != EOF);
}
```

checkExist()

This function is used once after a file is created from user input. It is used to check whether a file of the same name already exists. It does this by attempting to open it in read mode, and then checking whether the returned value from the fopen() function is NULL or not. A bool value of true or false is returned by this function to tell the program whether the file exists or not.

```
bool checkExist(const char *filename)
{
    bool doesExist = false;
    FILE *checker = fopen(filename, "r");
    if (checker != NULL)
    {
        doesExist = true;
        fclose(checker);
    }
    return doesExist;
}
```

encryption()

This function takes a file and an encryption key as a parameter. It opens the file, prints out its contents, encrypts it using XOR and replaces each character of the file with encrypted characters. A pointer is used to make sure the file’s contents are replaced with an encrypted version of the contents. The file is also truncated to clear any excess characters from the encryption process.

```
void encryption(const char *filename, char key)
{
    // open file
    FILE *file = fopen(filename, "r+");

    // print the file's original content
    printf("\n\nOriginal Content:\n");
    int ch;
    while ((ch = fgetc(file)) != EOF)
    {
        putchar(ch);
    }

    // put pointer to start of file
    fseek(file, 0, SEEK_SET);

    // encrypt file
    printf("\n\nEncrypted Content:\n");
    while ((ch = fgetc(file)) != EOF)
    {
        int encryptedChar = ch ^ key;
        putchar(encryptedChar);

        // use pointer to override file's contents with encrypted contents
        fseek(file, -1, SEEK_CUR);
        fputc(encryptedChar, file);
        fseek(file, 0, SEEK_CUR);
    }

    // truncate the file to the current position to remove any extra characters
    int truncateResult = ftruncate(fileno(file), ftell(file));
    if (truncateResult == -1)
    {
        perror("Error truncating file");
    }

    fclose(file);
}
```

Main program

Directory made

The main() function begins with the use of mkdir to make a new directory. The mkdir function returns either 0 or -1 depending on whether it was able to make the directory or not respectively. If it isn’t able to, then it tells the user that it failed to create a directory, this is usually because the directory already exists.

```
char newDirectory[100] = "New Directory";
    int directoryMade = mkdir(newDirectory, 0777);

    // check if directory created
    if (!directoryMade)
    {
        printf("Directory succesfully created\n");
    }
    else
    {
        printf("Failed to create directory\n");
        return 0;
    }
```

Focus set to directory

Next, chdir() is used to set the focus to the new directory that was made.

```
chdir(newDirectory);
```

User inputs

The user is asked to input:

- The name of the file
- The type of the file

scanf() is used to collect these inputs and the input buffer is cleared each time using the clearBuffer() function described earlier.

```
char fileName[100];
    char fileType[10];
    char fileFullName[110];
    fileFullName[0] = '\0';

    printf("Enter file name: ");
    scanf("%s", fileName);
    clearBuffer();

    printf("Enter file type: ");
    scanf("%s", fileType);
    clearBuffer();
```

Concatenate strings

The file name and file type that was input by the user are two separate strings that are now concatenated using strcat. They are concatenated by being added to the variable fileFullName which was created for this purpose.

```
strcat(fileFullName, fileName);
    strcat(fileFullName, fileType);
    printf("\n");
```

Make file

checkExist() is now used to check whether the file exists. The name of the file stored in the variable fileFullName is used to check whether a file of the same name exists. If it doesn't, the file is created, this is done by attempting to open it in write mode (this creates the file).

```
if (!checkExist(fileFullName))
    {
        // store destination in filePath variable
        snprintf(filePath, sizeof(filePath), "%s", fileFullName); //"%s\\%s", newDirectory, fileFullName);

        // create file
        fptr = fopen(filePath, "w");
        fclose(fptr);
    }
    else
    {
        printf("A file by the name of %s already exists.", fileFullName);
        return 0;
    }
```

File stores content

The user inputs text that will be sent to the file. fgets() is used to collect input from the user and place it in a variable called fileContent. The file is then opened in write mode, and the contents of fileContent are written to the file.

```
// user inputs content for file
    char fileContent[500];
    printf("Enter content to send to file: ");
    fgets(fileContent, sizeof(fileContent), stdin);

    // content sent to file
    fptr = fopen(filePath, "w");
    fprintf(fptr, "%s", fileContent);
    fclose(fptr);
```

Permission menu

The user is presented with a choice of permissions they can apply to the file. They can input a number between 1 and 4 to decide which permissions will be applied. A switch case statement is used here to provide different outcomes based on which number is inputted. A bool variable encryptPerm is created to determine whether the permissions of the file will allow its contents to be encrypted.

```
// show user choices of perms and allow input
    int userChoice;
    bool encryptPerm = false;
    printf("\n1 - Read Only\n\n2 - Read and Write\n\n3 - Read, Write, and Execute\n\n4 - Write Only\n\nEnter number to set permission of fi
    scanf("%d", &userChoice);
    clearBuffer();

    // different inputs to set different perms
    switch(userChoice)
    {
        case 1:
            // read only
            chmod(filePath, 0444);
            encryptPerm = false;
            break;
        case 2:
            // read and write
            chmod(filePath, 0666);
            encryptPerm = true;
            break;
        case 3:
            // read write execute
            chmod(filePath, 0777);
            encryptPerm = true;
            break;
        case 4:
            // write only
            chmod(filePath, 0222);
            encryptPerm = false;
            break;
        default:
            printf("Invalid Input.");
    }
```

Encrypt the file

If encryptPerm is true, the program will begin the encryption and decryption process. The key is predefined by me. The encryption function is called here to both encrypt and decrypt the file. This is because it uses the XOR encryption algorithm, which has the exact same process for encrypting and decrypting. After encryption, the user is asked to press enter to continue the program and decrypt the file’s contents, this is so that the user has time to review the encrypted content that’s output into the terminal as well as the encrypted content in the file itself.

```
if (encryptPerm == true)
    {
        // encrypt and decrypt file
        char key = "OPyear2Pelumi";
        encryption(filePath, key);
        printf("\n\nPress enter to decrpypt file\n");
        while (getchar() != '\n');
        encryption(filePath, key);
    }
```

Code

```
// Zeeshan Kazmi
// KAZ22527866

#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <sys/stat.h>
#include <unistd.h>
#include <string.h>


// function to clear input buffer
void clearBuffer()
{
    int i;
    while ((i = getchar()) != '\n' && i != EOF);
}


// function to check if a file exists
bool checkExist(const char *filename)
{
    bool doesExist = false;
    FILE *checker = fopen(filename, "r");
    if (checker != NULL)
    {
        doesExist = true;
        fclose(checker);
    }
    return doesExist;
}


// function to encode and decode a file
void encryption(const char *filename, char key)
{
    // open file
    FILE *file = fopen(filename, "r+");

    // print the file's original content
    printf("\n\nOriginal Content:\n");
    int ch;
    while ((ch = fgetc(file)) != EOF)
    {
        putchar(ch);
    }

    // put pointer to start of file
    fseek(file, 0, SEEK_SET);

    // encrypt file
    printf("\n\nEncrypted Content:\n");
    while ((ch = fgetc(file)) != EOF)
    {
        int encryptedChar = ch ^ key;
        putchar(encryptedChar);

        // use pointer to override file's contents with encrypted contents
        fseek(file, -1, SEEK_CUR);
        fputc(encryptedChar, file);
        fseek(file, 0, SEEK_CUR);
    }

    // truncate the file to the current position to remove any extra characters
```

```c
    int truncateResult = ftruncate(fileno(file), ftell(file));
    if (truncateResult == -1)
    {
        perror("Error truncating file");
    }

    fclose(file);
}


int main()
{
    // create new directory
    char newDirectory[100] = "New Directory";
    int directoryMade = mkdir(newDirectory, 0777);

    // check if directory created
    if (!directoryMade)
    {
        printf("Directory succesfully created\n");
    }
    else
    {
        printf("Failed to create directory\n");
        return 0;
    }

    // change focus to new directory
    chdir(newDirectory);

    // user input for name and type
    char fileName[100];
    char fileType[10];
    char fileFullName[110];
    fileFullName[0] = '\0';

    printf("Enter file name: ");
    scanf("%s", fileName);
    clearBuffer();

    printf("Enter file type: ");
    scanf("%s", fileType);
    clearBuffer();

    // concatenate strings
    strcat(fileFullName, fileName);
    strcat(fileFullName, fileType);
    printf("\n");

    // file
    FILE *fptr;
    char filePath[500];

    // check if file already exists using function
    if (!checkExist(fileFullName))
    {
        // store destination in filePath variable
        snprintf(filePath, sizeof(filePath), "%s", fileFullName); //"%s\\%s", newDirectory, fileFullName);

        // create file
        fptr = fopen(filePath, "w");
        fclose(fptr);
    }
    else
    {
        printf("A file by the name of %s already exists.", fileFullName);
        return 0;
    }

    // user inputs content for file
    char fileContent[500];
    printf("Enter content to send to file: ");
    fgets(fileContent, sizeof(fileContent), stdin);

    // content sent to file
    fptr = fopen(filePath, "w");
    fprintf(fptr, "%s", fileContent);
    fclose(fptr);
    // show user choices of perms and allow input
    int userChoice;
    bool encryptPerm = false;
    printf("\n1 - Read Only\n\n2 - Read and Write\n\n3 - Read, Write, and Execute\n\n4 - Write Only\n\nEnter number to set permission of fi
    scanf("%d", &userChoice);
    clearBuffer();

    // different inputs to set different perms
    switch(userChoice)
    {
        case 1:
            // read only
            chmod(filePath, 0444);
            encryptPerm = false;
            break;
        case 2:
            // read and write
            chmod(filePath, 0666);
            encryptPerm = true;
            break;
        case 3:
            // read write execute
            chmod(filePath, 0777);
            encryptPerm = true;
            break;
        case 4:
            // write only
            chmod(filePath, 0222);
            encryptPerm = false;
            break;
        default:
            printf("Invalid Input.");
    }

    if (encryptPerm == true)
    {
        // encrypt and decrypt file
        char key = "OPyear2Pelumi";
        encryption(filePath, key);
        printf("\n\nPress enter to decrpypt file\n");
        while (getchar() != '\n');
        encryption(filePath, key);
    }

    return 0;
}
```