CSE344 - Homework#1

Muhammet Talha Memişoğlu

210104004009

Introduction

The program handles file and directory operations such as creating, deleting, listing, reading and updating. This program was developed in C using Linux system calls.

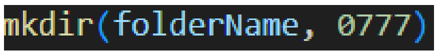
Code Explanation

main()

Main function in the program initialize the loop and take user arguments. Loop isn't terminated until user types "fileManager exit" command. In the beginning, function parses what user types to arguments. For parsing algorithm, I was helped from AI. Then, related function works according to command that user was typed. If command is not appropriate, program prints manual.

createDir()

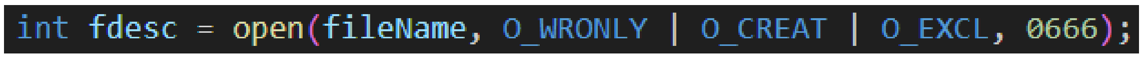
it creates a folder with full read, write and execute permissions to everyone. if directory already exist, function prints error. It uses **mkdir()** system call. In the end, it writes to log.txt for **logging.**



```
mkdir(folderName, 0777)
```

createFile()

If file doesn't already exist, It creates a file with **open()** system call. Open() function creates a file with read & write permissions for everyone. In the end, function writes timestamp to file and **log** the operation to log.txt file.

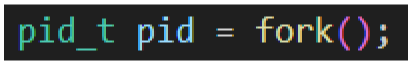


```
int fdesc = open(fileName, O_WRONLY | O_CREAT | O_EXCL, 0666);
```

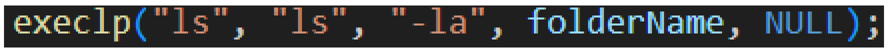
listDir()

It checks whether the directory exist. if it doesn't exist, prints error. In this function, **fork()** system call is used for separating processes. The parent process waits until child process is executed and child process uses **execlp()** system call. Execlp()

system call replaces the current process with a new process and current process is replaced by **ls** process in order to listing.



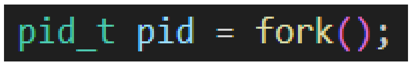
```
pid_t pid = fork();
```



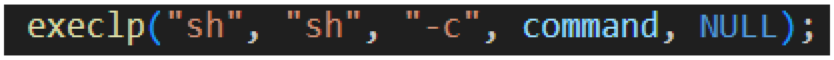
```
execlp("ls", "ls", "-la", folderName, NULL);
```

listFilesByExtension()

If directory that is given for input doesn't exist, it prints error. Additionally, **fork()** system call is used for seperating process. The parent process waits until the child process is executed. snprintf() function builds command that program will execute. It also checks whether files with that extension exist – for this snprintf() function, I benefited from AI -. In the end of the child process, the child process uses **execlp()** in order to execute our command that we built before.



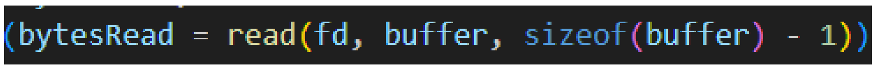
```
pid_t pid = fork();
```



```
execlp("sh", "sh", "-c", command, NULL);
```

readfile()

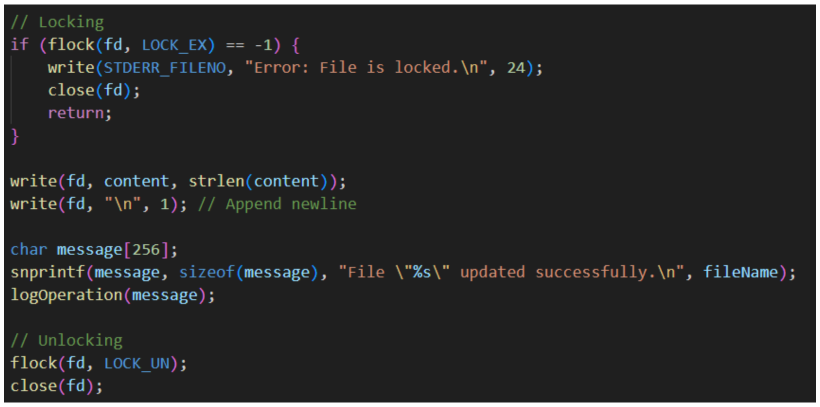
It opens the file in read mode with using **open().** If file not found, it prints error. After that, it reads from file using **read()** system call. If reading is successful, the function closes the file descriptor.



```
(bytesRead = read(fd, buffer, sizeof(buffer) - 1))
```

appendToFile()

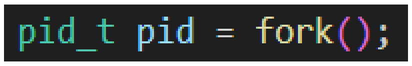
It opens the file in write only and append mode using **open().** Before writing to file, function locks with using **flock()** then appends content to file. After that, the function **logs** the operation to log.txt. In the end, it unlocks.



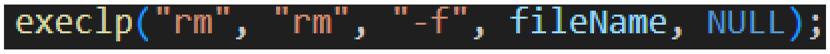
```
// Locking
if (flock(fd, LOCK_EX) == -1) {
    write(STDERR_FILENO, "Error: File is locked.\n", 24);
    close(fd);
    return;
}

write(fd, content, strlen(content));
write(fd, "\n", 1); // Append newline

char message[256];
snprintf(message, sizeof(message), "File \"%s\" updated successfully.\n", fileName);
logOperation(message);

// Unlocking
flock(fd, LOCK_UN);
close(fd);
```

deleteFile()

 It checks whether the file exist using access() function. Then it uses **fork()** system call for separating process. The parent process waits until the child process is executed. Child process uses **execlp()** system call. Execlp() system call replaces the current process with a new process and current process is replaced by **rm** process in order to delete file. After the child process is terminated, parent process **logs** what it has



```
pid_t pid = fork();
```

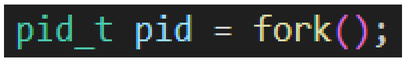


```
execlp("rm", "rm", "-f", fileName, NULL);
```

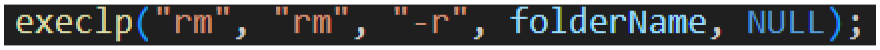
done to log.txt.

deletDir()

In the beginning of function, it checks whether directory is empty or not found. If directory is not empty or cant be found, it prints error. Then it uses **fork()** system call for separating process. The parent process waits until the child process is executed. Child process uses **execlp()** system call. Execlp() system call replaces the current process with a new process and current process is replaced by **rm** process in order to delete directory. After the child process is terminated, parent process **logs** what it has done to log.txt.



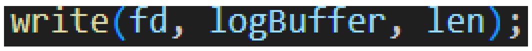
```
pid_t pid = fork();
```



```
execlp("rm", "rm", "-r", folderName, NULL);
```

logOperation()

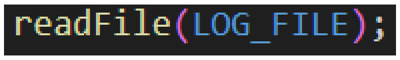
Function opens the file in write only and append mode. If file doesn’t exist, it creates the file. Then it writes the log message in proper format to log file using **write().** In the end, it closes the file.



```
write(fd, logBuffer, len);
```

showLogs()

It execute readFile() function that we have write before.



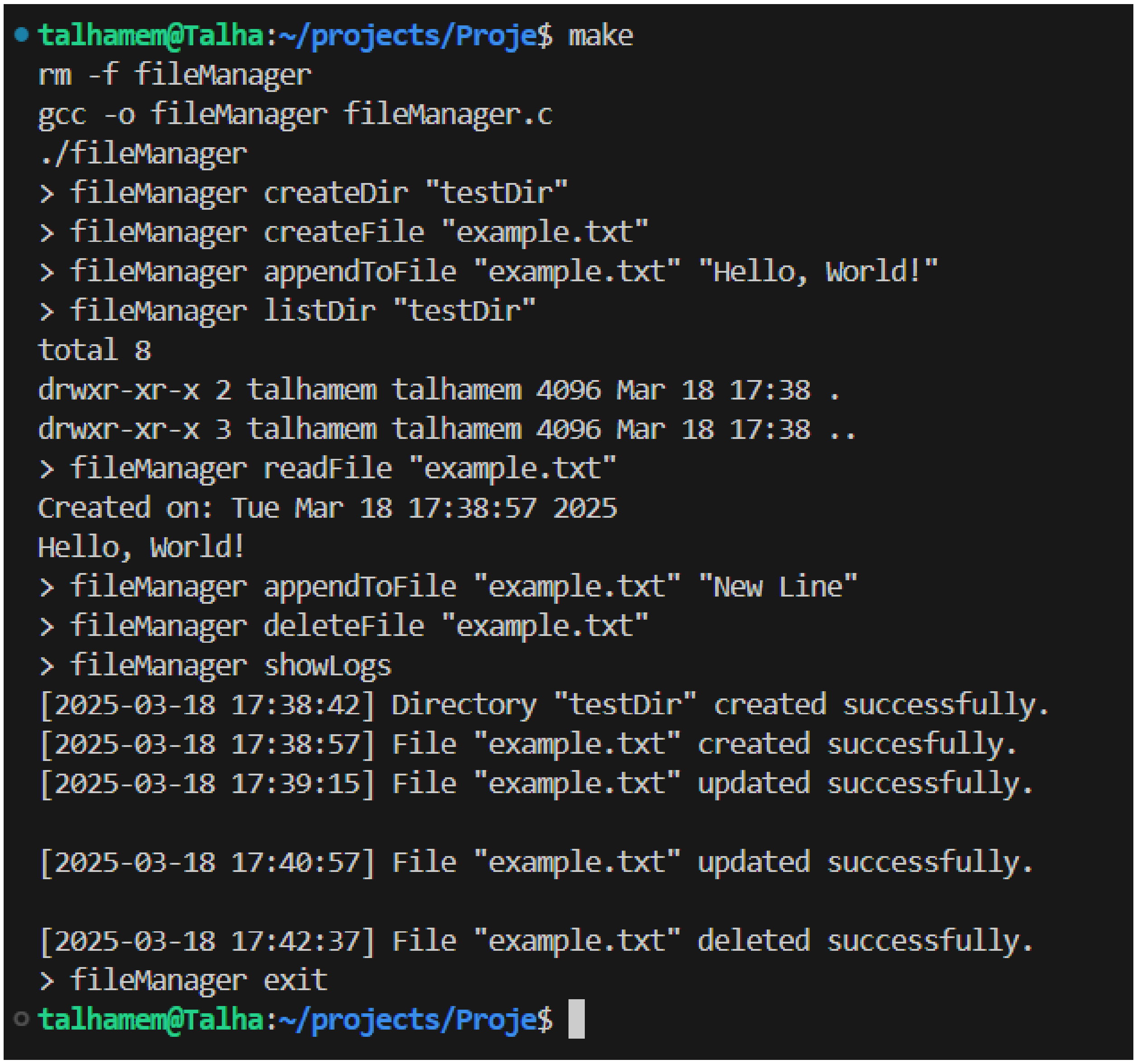
```
readFile(LOG_FILE);
```

showHelp()

It writes text that is given to terminal.

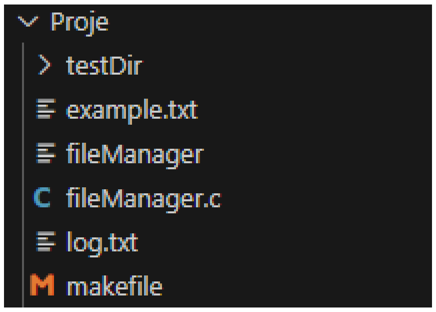
Screenshots

Inputs and Outputs for Test scenarios in the Document

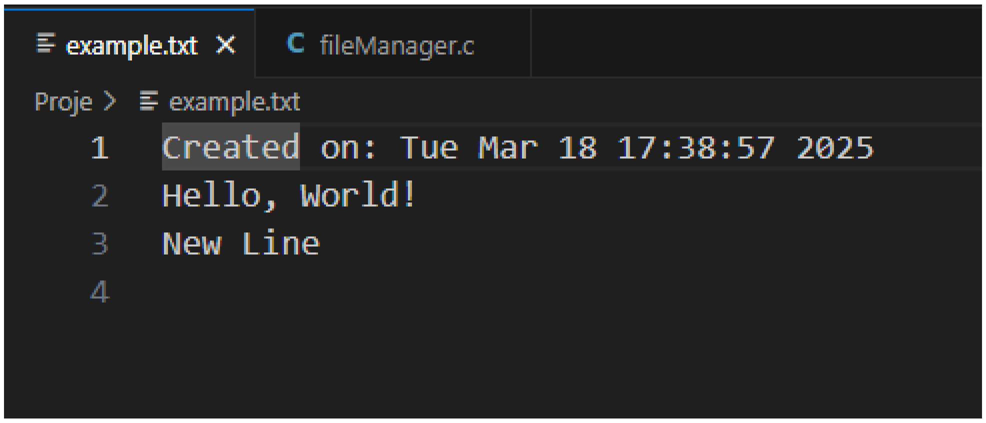


```
talhamem@Talha:~/projects/Proje$ make
rm -f fileManager
gcc -o fileManager fileManager.c
./fileManager
> fileManager createDir "testDir"
> fileManager createFile "example.txt"
> fileManager appendToFile "example.txt" "Hello, World!"
> fileManager listDir "testDir"
total 8
drwxr-xr-x 2 talhamem talhamem 4096 Mar 18 17:38 .
drwxr-xr-x 3 talhamem talhamem 4096 Mar 18 17:38 ..
> fileManager readFile "example.txt"
Created on: Tue Mar 18 17:38:57 2025
Hello, World!
> fileManager appendToFile "example.txt" "New Line"
> fileManager deleteFile "example.txt"
> fileManager showLogs
[2025-03-18 17:38:42] Directory "testDir" created successfully.
[2025-03-18 17:38:57] File "example.txt" created succesfully.
[2025-03-18 17:39:15] File "example.txt" updated successfully.

[2025-03-18 17:40:57] File "example.txt" updated successfully.

[2025-03-18 17:42:37] File "example.txt" deleted successfully.
> fileManager exit
talhamem@Talha:~/projects/Proje$
```

Files in the project and what is inside of example.txt before deleting example.txt.

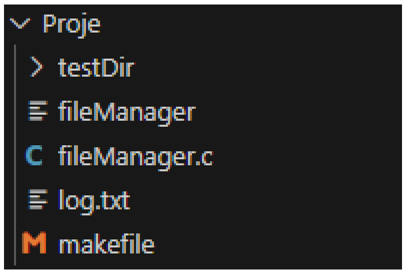


```
Proje
  testDir
  example.txt
  fileManager
  fileManager.c
  log.txt
  makefile
```



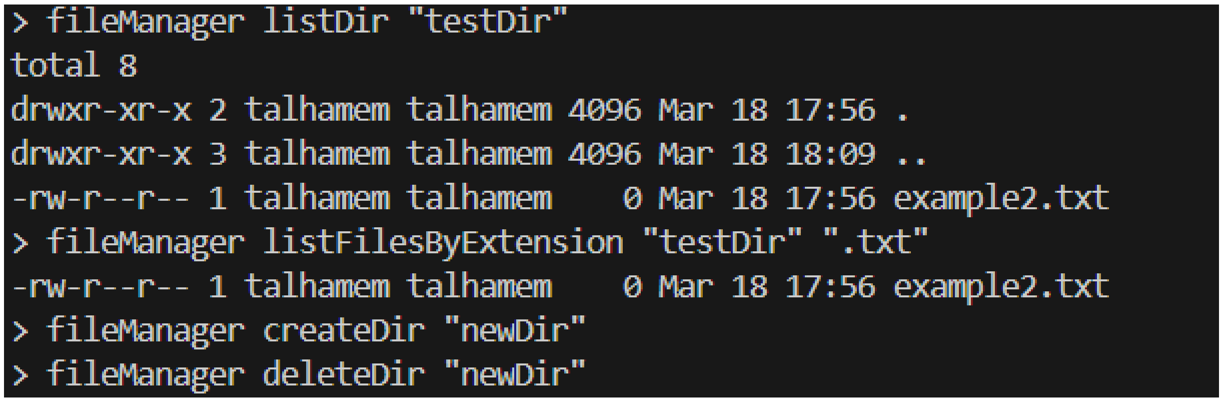
```
Created on: Tue Mar 18 17:38:57 2025
Hello, World!
New Line
```

Files in the project after deleting.

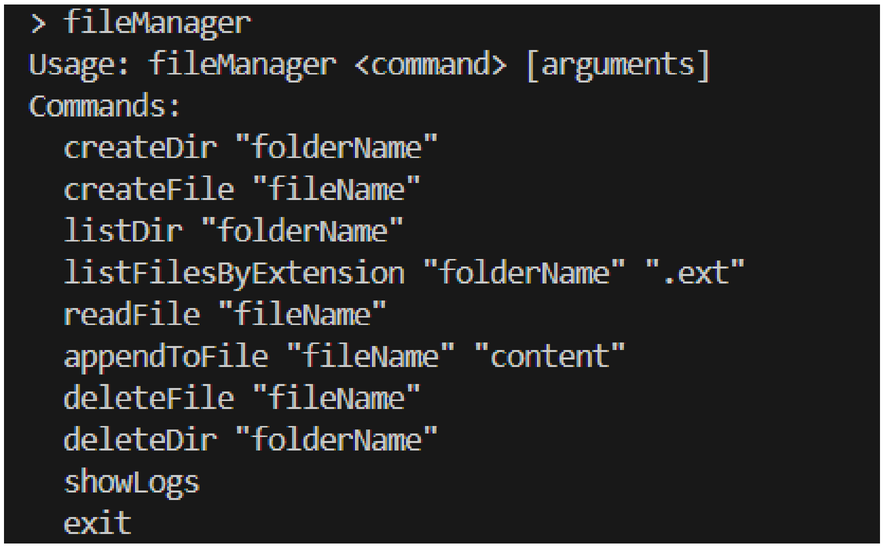


```
Proje
  testDir
  fileManager
  fileManager.c
  log.txt
  makefile
```

Input outputs for other functions

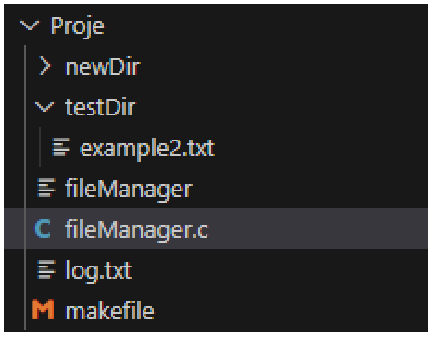


```
> fileManager listDir "testDir"
total 8
drwxr-xr-x 2 talhamem talhamem 4096 Mar 18 17:56 .
drwxr-xr-x 3 talhamem talhamem 4096 Mar 18 18:09 ..
-rw-r--r-- 1 talhamem talhamem    0 Mar 18 17:56 example2.txt
> fileManager listFilesByExtension "testDir" ".txt"
-rw-r--r-- 1 talhamem talhamem    0 Mar 18 17:56 example2.txt
> fileManager createDir "newDir"
> fileManager deleteDir "newDir"
```



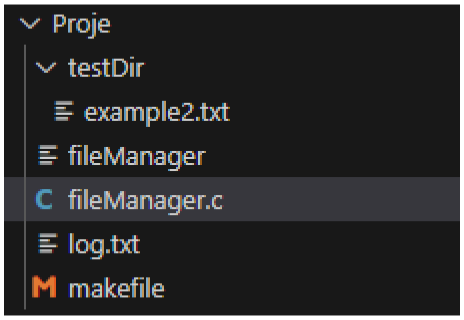
```
> fileManager
Usage: fileManager <command> [arguments]
Commands:
  createDir "folderName"
  createFile "fileName"
  listDir "folderName"
  listFilesByExtension "folderName" ".ext"
  readFile "fileName"
  appendToFile "fileName" "content"
  deleteFile "fileName"
  deleteDir "folderName"
  showLogs
  exit
```

Files in the project before deleting “newDir”.



```
Proje
  newDir
  testDir
    example2.txt
  fileManager
  fileManager.c
  log.txt
  makefile
```

Files in the project after deleting “newDir”.



```
Proje
  testDir
    example2.txt
  fileManager
  fileManager.c
  log.txt
  makefile
```

Conclusion

In this project, I had a hard time finding some functions and algorithms especially for the places I mentioned in this document. But I was helped by AI and some websites in that point. Additionally, debugging was hard so I put some function to handle these bugs. Finally, I have managed to complete this homework, and all requirements are fulfilled properly.