KAMİL DURU 200104004064

CSE 344 SYSTEM PROGRAMMING

HOMEWORK #1

1. Introduction

This program is a file and directory management system that provides various operations for handling files and directories using low-level system calls. It allows users to create, read, append, delete, and list files and directories without relying on the standard I/O library (stdio.h). Instead, it uses system calls like open(), write(), read(), close(), stat(), and unlink() to perform these operations.

Key Features

- **File Management:**
  - Create new files.
  - Read file contents.
  - Append content to files while ensuring file integrity with locks.
  - Delete files securely.
  - Check if a file is locked before modifying it.
- **Directory Management:**
  - Create new directories.
  - List files and subdirectories within a directory.
  - List files by specific extensions.
  - Delete directories (only if they are empty).
  - Check if a directory is empty.
- **Logging System:**
  - All operations performed are recorded in a log file (log.txt).
  - Timestamps are added to logs for tracking purposes.

- Custom error handling is implemented using a replacement for perror().

2. Code Explanation

my_perror (const char *msg)

This function implements perror mechanism that prints an error message along with the system error description using write().

How it works:

- Copies the provided message into errorMsg.
- Appends ": " to separate the custom message from the system error message.
- Retrieves the error message using strerror(errno) and appends it.
- Writes the formatted error message to stderr using file descriptor 2.

isFileLocked (const char *fileName)

Checks if a file is locked for writing.

How it works:

- Opens the file in read-only mode.
- Uses fcntl() with F_GETLK to check if the file is locked.
- If lock.l_type == F_UNLCK, the file is not locked; otherwise, it is locked.
- Closes the file and returns 1 if locked, 0 if not.

getTimestamp()

Generates a formatted timestamp string for logging.

How it works:

- Uses time() to get the current time.
- Uses localtime() to convert it into a human-readable format.
- Uses strftime() to format the time as [YYYY-MM-DD HH:MM:SS].
- Returns the formatted string.

logOperation(char *message)

Writes a log entry to log.txt.

How it works:

- Opens log.txt in append mode, creating it if necessary.
- Writes the message to the file using write().
- Appends a newline character.
- Closes the file.

createDir(char *dirName)

Creates a new directory if it does not already exist.

How it works:

- Uses stat() to check if the directory exists.
- If it does not exist, calls mkdir() to create it.
- Logs the operation with a timestamp.
- If the directory exists, prints an error message.

createFile(char *fileName)

Creates a new file if it does not already exist.

How it works:

- Uses stat() to check if the file exists.
- If it does not exist, opens the file with O_CREAT | O_WRONLY.
- Writes the timestamp to the file.
- Logs the operation.
- If the file exists, prints an error message.

listDir(char *dirName)

Lists all files and subdirectories in a given directory.

How it works:

- Uses opendir() to open the directory.
- Reads directory entries with readdir().
- Prints each entry using write().
- Logs the operation.
- Closes the directory.

listFileByExtension(char *dirName, char *extension)

Lists all files with a specific extension in a directory.

How it works:

- Opens the directory with opendir().
- Iterates through entries with readdir().
- Uses strstr() to check if the file name contains the specified extension.
- Prints matching file names.
- Logs the operation.

readFile(char *fileName)

Reads and prints the content of a file.

How it works:

- Opens the file in read-only mode.
- Uses read() to read up to 100 bytes at a time.
- Prints the content using write().
- Logs the operation.

deleteFile(char *fileName)

Deletes a file from the filesystem.

How it works:

- Uses stat() to check if the file exists.
- Uses unlink() to delete the file.
- Logs the operation.
- If the file does not exist, prints an error message.

isDirEmpty(char *dirName)

Checks if a directory is empty.

How it works:

- Opens the directory with opendir().
- Iterates through entries with readdir(), ignoring "." and "..".
- If it finds any valid entry, returns 0 (not empty).
- If no valid entries are found, returns 1 (empty).

deleteDir(char *dirName)

Deletes an empty directory.

How it works:

- Uses stat() to check if the directory exists.
- Calls isDirEmpty() to ensure the directory is empty.
- Uses rmdir() to delete the directory.
- Logs the operation.

appendToFile(char *fileName, char *content)

Appends content to an existing file.

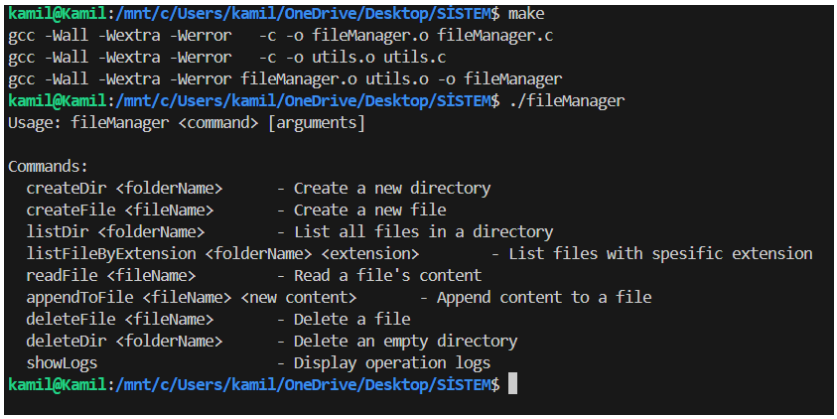
How it works:

- Uses stat() to check if the file exists and is regular.

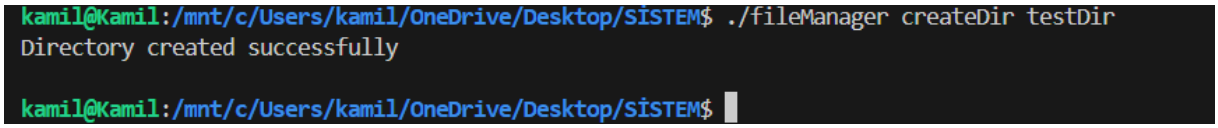
- Opens the file in append mode with O_WRONLY | O_APPEND.
- Uses flock() to lock the file for writing.
- Writes a newline character followed by the content.
- Unlocks the file and logs the operation.

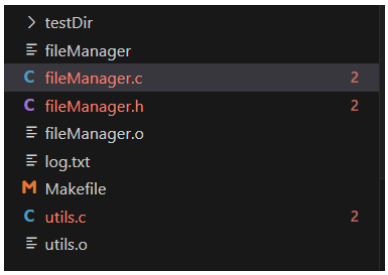
3. Screenshots

a) Usage guide



b) createDir





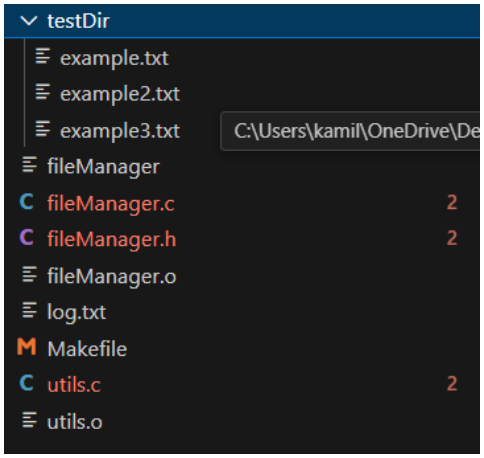
c) createFile

```
kamil@Kamil:/mnt/c/Users/kamil/OneDrive/Desktop/SİSTEM$ ./fileManager createFile ./testDir/example.txt
File created successfully

kamil@Kamil:/mnt/c/Users/kamil/OneDrive/Desktop/SİSTEM$ ./fileManager createFile ./testDir/example2.txt
File created successfully

kamil@Kamil:/mnt/c/Users/kamil/OneDrive/Desktop/SİSTEM$ ./fileManager createFile ./testDir/example3.txt
File created successfully

kamil@Kamil:/mnt/c/Users/kamil/OneDrive/Desktop/SİSTEM$
```



d) listDir

```
kamil@Kamil:/mnt/c/Users/kamil/OneDrive/Desktop/SİSTEM$ ./fileManager listDir testDir
.
..
example.txt
example2.txt
example3.txt
kamil@Kamil:/mnt/c/Users/kamil/OneDrive/Desktop/SİSTEM$
```

e) listFilesByExtension

```
kamil@Kamil:/mnt/c/Users/kamil/OneDrive/Desktop/SİSTEM$ ./fileManager listFilesByExtension testDir .txt
example.txt
example2.txt
example3.txt
kamil@Kamil:/mnt/c/Users/kamil/OneDrive/Desktop/SİSTEM$ ./fileManager listFilesByExtension testDir .bin
No files with extension .bin found in testDir
kamil@Kamil:/mnt/c/Users/kamil/OneDrive/Desktop/SİSTEM$
```

f) appendToFile

```
kamil@Kamil:/mnt/c/Users/kamil/OneDrive/Desktop/SİSTEM$ ./fileManager appendToFile ./testDir/example.txt Hello, World!
Content appended successfully
kamil@Kamil:/mnt/c/Users/kamil/OneDrive/Desktop/SİSTEM$
```

testDir > example.txt

```
1  [2025-03-23 19:00:22]
2  Hello, World!
```

g) readFile

```
kamil@Kamil:/mnt/c/Users/kamil/OneDrive/Desktop/SİSTEM$ ./fileManager readFile ./testDir/example.txt
[2025-03-23 19:00:22]
Hello, World!
kamil@Kamil:/mnt/c/Users/kamil/OneDrive/Desktop/SİSTEM$
```

h) deleteFile

```
kamil@Kamil:/mnt/c/Users/kamil/OneDrive/Desktop/SİSTEM$ ./fileManager deleteFile ./testDir/example.txt
File ./testDir/example.txt deleted successfully
kamil@Kamil:/mnt/c/Users/kamil/OneDrive/Desktop/SİSTEM$
```

testDir
- example2.txt
- example3.txt

i) deleteDir

```
kamil@Kamil:/mnt/c/Users/kamil/OneDrive/Desktop/SİSTEM$ ./fileManager deleteDir testDir
Error: Directory testDir is not empty.
kamil@Kamil:/mnt/c/Users/kamil/OneDrive/Desktop/SİSTEM$ ./fileManager deleteDir testDirEmpty
Directory testDirEmpty deleted successfully
kamil@Kamil:/mnt/c/Users/kamil/OneDrive/Desktop/SİSTEM$
```

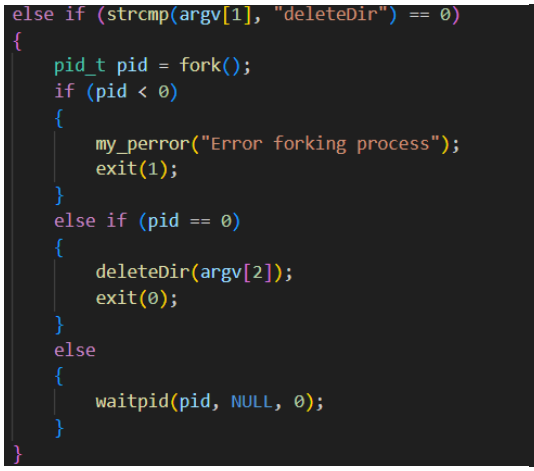
j) showLogs

```
kamil@Kamil:/mnt/c/Users/kamil/OneDrive/Desktop/SİSTEM$ ./fileManager showLogs
[2025-03-23 11:37:47]Content appended to file ./testDir/example1.txt successfully
[2025-03-23 12:48:26]Content appended to file ./testDir/example1.txt successfully
[2025-03-23 12:49:30]Content appended to file ./testDir/example1.txt successfully
[2025-03-23 12:49:54]Content appended to file ./testDir/example1.txt successfully
[2025-03-23 12:50:52]Content appended to file ./testDir/example1.txt successfully
[2025-03-23 12:51:04]Content appended to file ./testDir/example1.txt successfully
[2025-03-23 13:50:22]Directory testDir listed successfully
[2025-03-23 13:50:42]File ./testDir/example1.txt read successfully
[2025-03-23 13:51:06]Content appended to file ./testDir/example1.txt successfully
[2025-03-23 13:52:12]Content appended to file ./testDir/example1.txt successfully
[2025-03-23 13:53:47]Content appended to file ./testDir/example1.txt successfully
[2025-03-23 15:30:06]File ./testDir/example1.txt deleted successfully
[2025-03-23 15:42:45]File ./testDir/example2.txt deleted successfully
[2025-03-23 15:42:56]Directory testDir deleted successfully
[2025-03-23 16:15:47]Directory selam created successfully
[2025-03-23 16:16:00]File ./selam/ex.txt created successfully
[2025-03-23 16:16:15]File ./selam/ex.txt deleted successfully
[2025-03-23 16:16:45]Directory selam deleted successfully
[2025-03-23 18:59:29]Directory testDir created successfully
[2025-03-23 19:00:22]File ./testDir/example.txt created successfully
[2025-03-23 19:00:26]File ./testDir/example2.txt created successfully
[2025-03-23 19:00:35]File ./testDir/example3.txt created successfully
[2025-03-23 19:02:27]Directory testDir listed successfully
[2025-03-23 19:03:19]Files with extension .txt listed successfully
[2025-03-23 19:04:54]Content appended to file ./testDir/example.txt successfully
[2025-03-23 19:06:12]Content appended to file ./testDir/example.txt successfully
[2025-03-23 19:06:46]Content appended to file ./testDir/example.txt successfully
[2025-03-23 19:07:27]Content appended to file ./testDir/example.txt successfully
[2025-03-23 19:07:42]Content appended to file ./testDir/example.txt successfully
[2025-03-23 19:08:29]File ./testDir/example.txt read successfully
[2025-03-23 19:09:30]File ./testDir/example.txt deleted successfully
[2025-03-23 19:10:08]Directory testDirEmpty created successfully
[2025-03-23 19:10:23]Directory testDirEmpty deleted successfully
kamil@Kamil:/mnt/c/Users/kamil/OneDrive/Desktop/SİSTEM$
```

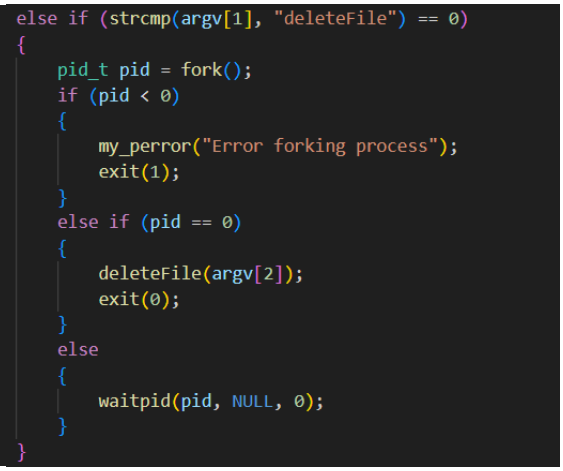
Note: Listing and deleting operations are handled with child prosesses

```c
else if (strcmp(argv[1], "listDir") == 0)
{
    pid_t pid = fork();
    if (pid < 0)
    {
        my_perror("Error forking process");
        exit(1);
    }
    else if (pid == 0)
    {
        listDir(argv[2]);
        exit(0);
    }
    else
    {
        waitpid(pid, NULL, 0);
    }
}
```

```c
else if (strcmp(argv[1], "listFilesByExtension") == 0)
{
    pid_t pid = fork();
    if (pid < 0)
    {
        my_perror("Error forking process");
        exit(1);
    }
    else if (pid == 0)
    {
        listFileByExtension(argv[2], argv[3]);
        exit(0);
    }
    else
    {
        waitpid(pid, NULL, 0);
    }
}
```



```c
else if (strcmp(argv[1], "deleteDir") == 0)
{
    pid_t pid = fork();
    if (pid < 0)
    {
        my_perror("Error forking process");
        exit(1);
    }
    else if (pid == 0)
    {
        deleteDir(argv[2]);
        exit(0);
    }
    else
    {
        waitpid(pid, NULL, 0);
    }
}
```



```c
else if (strcmp(argv[1], "deleteFile") == 0)
{
    pid_t pid = fork();
    if (pid < 0)
    {
        my_perror("Error forking process");
        exit(1);
    }
    else if (pid == 0)
    {
        deleteFile(argv[2]);
        exit(0);
    }
    else
    {
        waitpid(pid, NULL, 0);
    }
}
```

4) Conclusion

Challenges Faced

During the development of this file and directory management program, several challenges were encountered:

1. Handling File Locks:
    - Ensuring that files were properly locked during writing to prevent data corruption was a challenge.
    - Solution: Used flock() to lock files before writing and unlocking them after the operation.
2. Checking If a Directory is Empty Before Deletion:
    - The program needed to check whether a directory was empty before deleting it to prevent accidental removal of files.
    - Solution: Implemented isDirEmpty() to iterate through directory entries and verify emptiness.
3. Efficient Error Handling:
    - Properly handling system errors while ensuring meaningful error messages was crucial.
    - Solution: Implemented a custom my_perror() function to standardize error messages and improve debugging.

4. Managing File Permissions and Existence Checks:
    - Ensuring files and directories were not created or deleted unintentionally.
    - Solution: Used stat() to check file/directory existence before performing operations.

Final Thoughts

This project provided valuable experience in working with low-level file system operations using system calls. It reinforced the importance of error handling, logging, and resource management. The implementation ensures safe file and directory operations while maintaining system integrity.