 [Translated from French to English - www.onlinedoctranslator.com](https://www.onlinedoctranslator.com/en/?utm_source=onlinedoctranslator&utm_medium=docx&utm_campaign=attribution)

A black and blue logo with white text

Description automatically generated

PROJECT REPORT

LO14 2023/2024

File Synchronizer

**Version 1.0**

**Members of the team**: Cristina Meda Titu, Julia Stępień

Content

[Introduction 4](#_Toc160295317)

[Architecture 4](#_Toc160295318)

[1. Components and Modules 4](#_Toc160295319)

[2. Communication 4](#_Toc160295320)

[3. Data managment 4](#_Toc160295321)

[4. Concurrency and parallelism 5](#_Toc160295322)

[5. User Interface (UI) 5](#_Toc160295323)

[Application flow 5](#_Toc160295324)

[6. Performance 5](#_Toc160295325)

[File synchronization criteria 6](#_Toc160295326)

[Resolving conflicts for different file types 6](#_Toc160295327)

[Managing non-existent files 6](#_Toc160295328)

[Management of non-compliant files 6](#_Toc160295329)

[Conflict resolution for modified (non-compliant) files with identical content 6](#_Toc160295330)

[Conflict resolution for files with different content, but identical metadata 7](#_Toc160295331)

[Description of the algorithm 7](#_Toc160295332)

[sync <path-to-tree1> <path-to-tree2> 7](#_Toc160295333)

[Step #1: Validation 7](#_Toc160295334)

[Step #2: Preparing the log file 7](#_Toc160295335)

[Step #3: Preparing the output file 7](#_Toc160295336)

[Step #4: Synchronization - simultaneous tree traversal 7](#_Toc160295337)

[Step #5: View logging information 8](#_Toc160295338)

[resolve <option> <path-file1> <path-file2> 8](#_Toc160295339)

[Step #1: Validation 8](#_Toc160295340)

[Step #2: Resolution 9](#_Toc160295341)

[Step #3: Log Management 9](#_Toc160295342)

[Step #4: Feedback 9](#_Toc160295343)

[resolveAll <option> 9](#_Toc160295344)

[Step #1: Validation 9](#_Toc160295345)

[Step #2: Obtain conflict entries 10](#_Toc160295346)

[Step #3: Resolution 10](#_Toc160295347)

[Step #4: Log Management 10](#_Toc160295348)

[Step #5: Feedback 10](#_Toc160295349)

[Conclusion 10](#_Toc160295350)

# Introduction

The main goal of this project is to develop our own file synchronization tool that allows automatic synchronizations of two file systems. The program should be able to handle conflicts, handle symbolic links, and provide a simple, user-friendly interface with clear information about the synchronization process and conflict resolution.

Our project focuses on developing a file synchronizer for two file systems located on the same machine. The synchronizer's task is to parallel compare and synchronize files and directories, taking into account content and metadata while considering potential conflicts. We provide mechanisms to efficiently resolve conflicts by notifying the user of differences. An integral part of the tool is a log file that is updated with each successful synchronization.

# Architecture

## Components and Modules

1. Synchronization script named *sync*
2. Conflict resolution script named *resolve*
3. Script to resolve all conflicts named *resolveAll*
4. The log file that stores the two tree paths to be synchronized and the entries with details of the files that have been successfully updated so far
5. The conflicts file containing all conflicting paths found after running the sync script

## Communication

The sync program manipulates the data contained in the log, adding or modifying the metadata of the files that have been synchronized for both trees. It also generates the content of the conflicts file with the found problematic paths.

The resolve and resolveAll scripts use the data stored in the conflicts file for validation purposes (resolve program) or as an input file (resolveAll program).

## Data managment

Storage mechanisms:

1. The log file which records the details of the modification sources (file path, type, permissions, size, date of last modification).
2. The conflicts file which records all paths having synchronization problems and the reasons for these problems.
3. The auxiliary files paths1, paths2, paths\_tree1, paths\_tree2, only\_tree1, only\_tree2, commonly used in solving the task, are deleted at the end of the program.

## Concurrency and parallelism

Both tree paths are traversed in parallel during verification and synchronization.

## User Interface (UI)

The user interface is configured as a command line interface, providing the following functionality:

Synchronize two tree paths, display and store conflicts (conflicts file) and suggest resolution methods: sync <path-to-tree1> <path-to-tree2>

Resolve a conflict between two paths from the file generated by the sync command (conflicts) with the selected method (option): resolve <option> <path-file1> <path-file2>

Resolve all conflicts from the file generated by the sync command (conflicts) with the selected method (option): resolveAll <option>

### Application flow

The user enters the associated command, sync, providing the two tree paths to be synchronized as arguments. After validation, if the directories are found, the application launches the synchronization process. If the paths do not match any existing directories, the first step is repeated until the user enters two valid paths. After finalization, the application displays the list of conflicts and resolution options for the user to choose from. The program also saves the conflict paths and their reason in a new file. To resolve a conflict, the user enters the resolve command, mentioning the resolution option and the two conflicting paths. A message is displayed at the end. Another possibility is to resolve all conflicts at once, using the resolveAll command which takes all conflicting file paths from the output file generated by the sync command and resolves them using the chosen method (option).

## Performance

Use of efficient algorithms for comparison and synchronization.

Cases covered

In the context of file synchronization between two file systems A and B, the following rules are applied:

## File synchronization criteria

If the file is ordinary with identical data and metadata in A and B, consider it synchronized with no further action needed.

Recursive checking of directories

If the file is present in both A and B and of type directory, initiate a recursive scan of the files contained in the respective directories.

### Resolving conflicts for different file types

If the file is present in both file systems but one occurrence is a regular file while the other is a directory, categorize it as a conflict.

### Managing non-existent files

If the file exists in A but cannot be found in B, perform the operation to copy the file from A to B. If the file exists in B but is missing in A, act accordingly.

### Management of non-compliant files

If the file in A conforms to the log file, but the file in B does not, the file in B has been updated. Therefore, the contents and metadata of the file in B must be copied to the file in A, and vice versa.

If the two files in A and B do not match the log file (have been modified) and have different contents, mark them as conflict. In case of identical content, the following section describes the appropriate solution.

### Conflict resolution for modified (non-compliant) files with identical content

If both files have been modified but their contents are identical:

has. if the metadata is the same, no further action is necessary.

b. If the first file's metadata matches the metadata stored in the log, which means the second file has been modified, update its metadata to match the modified file.

vs. If the metadata of the two files differs from that of the log or the entry is not found, categorize it as a conflict and warn the user accordingly.

### Conflict resolution for files with different content, but identical metadata

If both files have the same metadata, whether logged or not (size, date, and time coincidence), but the content is different, mark it as a conflict.

These rules establish an efficient and systematic approach to file synchronization in order to properly work through the synchronization process and manage conflicts between file systems A and B.

With each successful sync, the log is updated for the associated entry.

# Description of the algorithm

## sync <path-to-tree1> <path-to-tree2>

The synchronization script takes two absolute file paths representing two file trees as arguments and synchronizes them while marking conflicts between them.

### Step #1: Validation

Paths are checked to ensure they are existing directories. In case of invalid arguments, an error message will be displayed.

### Step #2: Preparing the log file

The file tracking the synchronization of the two trees, called "log", is being prepared. If it doesn't exist, create a new empty one. The tree paths are written on the first line, separated by a space, to indicate the synchronization sources. If it already exists, and the paths do not match those marked in the log, delete all content and create a new log file for the new tree paths.

### Step #3: Preparing the output file

The output file, named "conflicts", is being prepared for further use. If it doesn't exist, create a new empty one. Otherwise, erase its contents.

### Step #4: Synchronization - simultaneous tree traversal

The algorithm starts by searching for files exclusively present in the first tree and copies them to the other. Then the same process is repeated for the second tree. This is done by sorting the two files containing all the paths of the two trees and comparing them with the comm command. The "-23" option means to remove lines unique to both files and print only lines unique to the first file. The "-13" option will only take unique lines from the second file. Finally, we call the same command but with the "-12" option to retrieve only the common paths of the two files.

The copy\_content function takes as arguments the file containing the paths to copy, the absolute path of the source tree and the absolute path of the destination tree. Its role is to copy the files and directories mentioned in the arguments file from the source tree to the destination tree.

Therefore, copy\_content is applied to both files generated with the comm command, ensuring that content found only in one of the trees is copied to the other.

The next step is to read each path from the common paths file, construct the absolute paths in both trees, then check each case mentioned above, in the Covered Cases section, and handle them efficiently as described.

If there is a conflict, the problematic paths and the reason for the conflict are added as a single line to the output file (conflicts).

When synchronized, the file path, type, size, permissions, and last modification date are added to the log file, or the corresponding entry, if existing, is updated.

### Step #5: View logging information

Synchronization information is displayed during the process: the current file being checked, the metadata in both trees, the possible entry if it exists, and the sync/conflict reasons.

Step #6: Show suggestions for resolving conflicts

An informational message is displayed at the end, containing the commands to use to resolve the conflicts (resolve and resolveAll) with the available options (keep-first, keep-second, keep-initial, keep-latest).

## resolve <option> <path-file1> <path-file2>

The resolution script takes as arguments a variable specifying the way to manage the conflict and the twoabsolute pathsfiles requiring this resolution. The script resolves a specific conflict between two given files using the option selected by the user and the metadata resolution history is updated in the log file.

### Step #1: Validation

The option handling algorithm checks if the option specified by the user is one of those supported by the script. If the option is invalid, then the program returns an error message and indicates that the selected option is invalid.

The conflicts file checks if the conflict actually comes from the files provided. If no conflicts are found within the provided files, then an error message is displayed stating that no conflicts were found between the provided files.

### Step #2: Resolution

The script reads each line from a file named "conflicts", where each line contains two conflicting paths.

For each conflict entry: it determines the types of conflicting paths (directories or regular files), depending on the resolution option, performs actions such as copying, deleting or updating files.

### Step #3: Log Management

The log file is updated with the resolved metadata.

### Step #4: Feedback

During resolution the script displays messages indicating the resolution status and synchronization of each file or directory. If all steps of the script were executed successfully and the log was successfully updated, a success message is displayed. At the end, checks if there are any conflicts remaining in the "conflicts" file and displays a summary message.

#### Options available:

***keep-first*** *-*selects and keeps the conflicting part of the first file, the other is updated with the selected data as well as its metadata

***keep-second*** *-*selects and keeps the conflicting part of the second file, the other is updated with the selected data as well as its metadata

***keep-latest***- selects and keeps the conflicting part of the most recently modified file using the timestamp, the other is updated with the selected data as well as its metadata

***keep-initial*** *-*selects and keeps the conflicting part of the file modified the longest ago using the timestamp, the other is updated with the selected data as well as its metadata

## resolveAll <option>

The resolution script takes a single argument as input, a potion specifying how to resolve the conflict. The script resolves any issues coming from the conflicts file using the option selected by the user and the resolved metadata is updated in the log file.

### Step #1: Validation

The script checks whether the option parameter is supported by the program. If the option is invalid, then an error message is displayed indicating an invalid option.

### Step #2: Obtain conflict entries

The lines in the conflicts file are read in order to obtain the file paths for each conflict.

### Step #3: Resolution

While reading the file, the resolution method used is the one specified by the user in options on all conflict lines.

### Step #4: Log Management

For each resolved conflict, the log file is updated with the resolution metadata.

### Step #5: Feedback

If all steps went well and the log was successfully updated, a success message is displayed.

# Conclusion

The File Synchronizer project presents a simple solution for synchronizing two file systems, handling conflicts with a wide possibility of resolution options available. The command line interface, combined with the data manager and conflict resolution mechanism, allows for a user-friendly and efficient management of the file synchronization experience. The modular design of the project and the methods to comply with what the user requested allow versatility and use of the project in various synchronization scenarios.